MET

Model Evaluation Tools

MET User's Guide

version 12.2.2

Aug 28, 2026

Contents

1 Overview of MET **3**
- 1.1 Purpose and Organization of the User's Guide . . . 3
- 1.2 The Developmental Testbed Center (DTC) . . . 4
- 1.3 MET Goals and Design Philosophy . . . 4
- 1.4 MET Components . . . 5
- 1.5 Future Development Plans . . . 9
- 1.6 User Support . . . 9
- 1.7 Fortify and SonarQube . . . 9

2 MET Release Information **11**
- 2.1 MET Release Notes . . . 11
  - 2.1.1 MET Version 12.2.2 Release Notes (20260828) . . . 11
  - 2.1.2 MET Version 12.2.1 Release Notes (20260227) . . . 12
  - 2.1.3 MET Version 12.2.0 Release Notes (20251113) . . . 12
- 2.2 MET Upgrade Instructions . . . 14
  - 2.2.1 MET Version 12.2.0 Upgrade Instructions . . . 14

3 Installation **17**
- 3.1 Introduction . . . 17
- 3.2 Required External Libraries . . . 17
- 3.3 Suggested External Utilities . . . 18
- 3.4 Using the compile_MET_all.sh script . . . 18
  - 3.4.1 Environment Variables to Run Script . . . 19
  - 3.4.2 Environment Variable Descriptions . . . 20
  - 3.4.3 External Library Handling in compile_MET_all.sh . . . 22
  - 3.4.4 Executing the compile_MET_all.sh script . . . 24
- 3.5 Using Docker for Running MET . . . 25
  - 3.5.1 Installing Docker . . . 25
  - 3.5.2 Loading the Latest Docker Image of MET . . . 25
  - 3.5.3 Running the Docker version of MET . . . 25
- 3.6 Using Apptainer for Running MET . . . 26
  - 3.6.1 Installing Apptainer . . . 26
  - 3.6.2 Loading the Latest MET Image . . . 26
  - 3.6.3 Running the MET Container . . . 26

3.6.4 Stopping the Apptainer Instance . . . 27

4 MET Data I/O 29
4.1 Input Data Formats . . . 29
4.1.1 ASCII File Lists . . . 30
4.1.2 Requirements for CF Compliant NetCDF . . . 31
4.1.3 Performance with NetCDF Input Data . . . 33
4.2 Intermediate Data Formats . . . 33
4.3 Output Data Formats . . . 34
4.4 Data Format Summary . . . 35
4.5 Configuration File Details . . . 39

5 Configuration File Overview 41
5.1 Runtime Environment Variables . . . 45
5.1.1 User-Specified Environment Variables . . . 45
5.1.2 MET_AIRNOW_STATIONS . . . 45
5.1.3 MET_NDBC_STATIONS . . . 45
5.1.4 MET_BASE . . . 46
5.1.5 MET_OBS_ERROR_TABLE . . . 46
5.1.6 MET_GRIB_TABLES . . . 47
5.1.7 OMP_NUM_THREADS . . . 48
5.1.7.1 Introduction . . . 48
5.1.7.2 Setting the number of threads . . . 48
5.1.7.3 Thread Binding . . . 49
5.1.8 MET_KEEP_TEMP_FILE . . . 49
5.1.9 MET_PYTHON_DEBUG . . . 50
5.1.10 MET_PYTHON_TMP_FORMAT . . . 50
5.1.11 MET_USE_WRF_SUBGRID . . . 50
5.2 Settings Common to Multiple Tools . . . 50
5.2.1 exit_on_warning . . . 50
5.2.2 time_offset_warning . . . 50
5.2.3 nc_compression . . . 51
5.2.4 output_precision . . . 51
5.2.5 tmp_dir . . . 51
5.2.6 message_type_group_map . . . 51
5.2.7 obtype_as_group_val_flag . . . 52
5.2.8 message_type_map . . . 52
5.2.9 model . . . 53
5.2.10 desc . . . 53
5.2.11 obtype . . . 53
5.2.12 regrid . . . 53
5.2.12.1 to_grid . . . 54
5.2.12.2 vld_thresh . . . 54
5.2.12.3 method . . . 54
5.2.12.4 width . . . 55
5.2.12.5 shape . . . 55
5.2.12.6 gaussian_dx . . . 55
5.2.12.7 gaussian_radius . . . 55

5.2.12.8 convert, censor_thresh, and censor_val . . . 56
5.2.13 fcst . . . 56
5.2.13.1 field . . . 56
5.2.13.1.1 field.name . . . 56
5.2.13.1.2 field.level . . . 56
5.2.13.1.3 field.prob . . . 57
5.2.13.1.4 field.prob_as_scalar . . . 58
5.2.13.1.5 field.convert . . . 58
5.2.13.1.6 field.censor_thresh and field.censor_val . . . 58
5.2.13.1.7 field.mpr_column and field.mpr_thresh . . . 59
5.2.13.1.8 field.cat_thresh . . . 59
5.2.13.1.9 field.cnt_thresh . . . 59
5.2.13.1.10field.cnt_logic . . . 59
5.2.13.1.11Override and Correct Metadata . . . 60
5.2.13.2 file_type . . . 60
5.2.13.3 wind_thresh . . . 61
5.2.13.4 wind_logic . . . 61
5.2.13.5 eclv_points . . . 61
5.2.13.6 init_time . . . 61
5.2.13.7 valid_time . . . 62
5.2.13.8 lead_time . . . 62
5.2.13.9 File-format Specific Settings for the “field” Entry . . . 62
5.2.13.9.1 GRIB1 and GRIB2 . . . 62
5.2.13.9.2 NetCDF . . . 64
5.2.13.9.3 Python . . . 64
5.2.14 obs . . . 66
5.2.14.1 message_type . . . 67
5.2.14.2 sid_inc and sid_exc . . . 68
5.2.15 climo_mean . . . 68
5.2.15.1 file_names . . . 68
5.2.15.2 field . . . 68
5.2.15.3 regrid . . . 69
5.2.15.4 time_interp_method . . . 69
5.2.15.5 day_interval . . . 69
5.2.15.6 hour_interval . . . 69
5.2.16 climo_stdev . . . 70
5.2.17 climo_cdf . . . 71
5.2.18 climate_data . . . 72
5.2.19 seeps_p1_thresh . . . 72
5.2.20 seeps_grid_climo_name . . . 72
5.2.21 seeps_point_climo_name . . . 72
5.2.22 mask_missing_flag . . . 73
5.2.23 obs_window . . . 73
5.2.24 mask . . . 73
5.2.24.1 grid . . . 73
5.2.24.2 poly . . . 74
5.2.24.3 sid and llpnt . . . 75
5.2.25 ci_alpha . . . 76

5.2.26 boot . . . 76
5.2.26.1 interval . . . 76
5.2.26.2 rep_prop . . . 76
5.2.26.3 n_rep . . . 77
5.2.26.4 rng . . . 77
5.2.26.5 seed . . . 77
5.2.27 interp . . . 77
5.2.27.1 field . . . 78
5.2.27.2 vld_thresh . . . 78
5.2.27.3 shape . . . 78
5.2.27.4 type . . . 78
5.2.27.4.1 type.width . . . 78
5.2.27.4.2 type.method . . . 79
5.2.28 land_mask . . . 80
5.2.29 topo_mask . . . 80
5.2.30 lapse_rate_correction . . . 81
5.2.31 msl_agl_conversion . . . 82
5.2.32 hira . . . 83
5.2.32.1 flag . . . 83
5.2.32.2 width . . . 83
5.2.32.3 vld_thresh . . . 84
5.2.32.4 cov_thresh . . . 84
5.2.32.5 shape . . . 84
5.2.32.6 prob_cat_thresh . . . 84
5.2.33 gradient . . . 84
5.2.34 output_flag . . . 85
5.2.35 nc_pairs_flag . . . 86
5.2.36 nc_pairs_var_name . . . 86
5.2.37 nc_pairs_var_suffix . . . 87
5.2.38 ps_plot_flag . . . 87
5.2.39 grid_weight_flag . . . 87
5.2.40 point_weight_flag . . . 88
5.2.41 hss_ec_value . . . 88
5.2.42 rank_corr_flag . . . 89
5.2.43 duplicate_flag . . . 89
5.2.44 obs_summary . . . 89
5.2.45 obs_perc_value . . . 90
5.2.46 obs_quality_inc . . . 90
5.2.47 obs_quality_exc . . . 90
5.2.48 met_data_dir . . . 90
5.2.49 many_plots . . . 90
5.2.49.1 color_table . . . 91
5.2.49.2 plot_min and plot_max . . . 91
5.2.49.3 colorbar_flag . . . 91
5.2.50 output_prefix . . . 91
5.2.51 version . . . 91
5.2.52 time_summary . . . 92
5.3 Settings Specific to Individual Tools . . . 93

5.3.1 GenEnsProdConfig_default . . . 93
5.3.1.1 ens . . . 93
5.3.1.1.1 ens.censor_thresh and ens.censor_val . . . 93
5.3.1.1.2 ens.ens_thresh . . . 93
5.3.1.1.3 ens.vld_thresh . . . 93
5.3.1.1.4 ens.field . . . 94
5.3.1.2 nbrhd_prob . . . 94
5.3.1.3 nmep_smooth . . . 95
5.3.1.4 ensemble_flag . . . 95
5.3.2 EnsembleStatConfig_default . . . 96
5.3.2.1 fcst, obs . . . 96
5.3.2.1.1 obs.ens_ssvar_bin_size . . . 96
5.3.2.1.2 obs.ens_phist_bin_size . . . 96
5.3.2.1.3 fcst.prob_cat_thresh and obs.prob_cat_thresh . . . 97
5.3.2.2 nc_var_str . . . 97
5.3.2.3 obs_thresh . . . 97
5.3.2.4 skip_const . . . 98
5.3.2.5 obs_error . . . 98
5.3.2.5.1 obs_error.flag . . . 98
5.3.2.5.2 obs_error.dist_type . . . 98
5.3.2.5.3 obs_error.dist_parm . . . 98
5.3.2.5.4 obs_error.inst_bias_scale and obs_error.inst_bias_offset . . . 99
5.3.2.6 rng . . . 99
5.3.3 MODEAnalysisConfig_default . . . 99
5.3.3.1 Toggles . . . 100
5.3.3.2 Multiple-set String Options . . . 100
5.3.3.3 Multiple-set Integer Options . . . 101
5.3.3.4 Integer max/min Options . . . 101
5.3.3.5 Date/time max/min Options . . . 102
5.3.3.6 Floating-point max/min Options . . . 103
5.3.4 MODEConfig_default . . . 105
5.3.4.1 quilt . . . 105
5.3.4.2 fcst, obs . . . 105
5.3.4.2.1 fcst/obs.censor_thresh and fcst/obs.censor_val . . . 105
5.3.4.2.2 fcst/obs.conv_radius . . . 105
5.3.4.2.3 fcst/obs.conv_thresh . . . 105
5.3.4.2.4 fcst/obs.vld_thresh . . . 106
5.3.4.2.5 fcst/obs.filter_attr_name and fcst/obs.filter_attr_thresh . . . 106
5.3.4.2.6 fcst/obs.merge_thresh . . . 106
5.3.4.2.7 fcst/obs.merge_flag . . . 106
5.3.4.3 grid_res . . . 107
5.3.4.4 match_flag . . . 107
5.3.4.5 max_centroid_dist . . . 107
5.3.4.6 weight . . . 108
5.3.4.7 interest_function . . . 108
5.3.4.8 total_interest_thresh . . . 109
5.3.4.9 print_interest_thresh . . . 109
5.3.4.10 plot_valid_flag . . . 110

5.3.4.11 plot_gcarc_flag . . . 110
5.3.4.12 ct_stats_flag . . . 110
5.3.4.13 shift_right . . . 110
5.3.5 PB2NCConfig_default . . . 110
5.3.5.1 message_type . . . 111
5.3.5.2 station_id . . . 112
5.3.5.3 elevation_range . . . 112
5.3.5.4 pb_report_type . . . 112
5.3.5.5 in_report_type . . . 113
5.3.5.6 instrument_type . . . 113
5.3.5.7 level_range . . . 113
5.3.5.8 level_category . . . 113
5.3.5.9 obs_bufr_var . . . 114
5.3.5.10 obs_bufr_map . . . 115
5.3.5.11 obs_prepbufr_map . . . 115
5.3.5.12 quality_mark_thresh . . . 115
5.3.5.13 event_stack_flag . . . 116
5.3.6 SeriesAnalysisConfig_default . . . 116
5.3.6.1 block_size . . . 116
5.3.6.2 vld_thresh . . . 116
5.3.6.3 output_stats . . . 116
5.3.7 STATAnalysisConfig_default . . . 117
5.3.7.1 jobs . . . 117
5.3.7.1.1 -job filter . . . 117
5.3.7.1.2 -job summary . . . 117
5.3.7.1.3 -job aggregate . . . 118
5.3.7.1.4 -job aggregate_stat . . . 119
5.3.7.1.5 -job ss_index . . . 120
5.3.7.1.6 -job go_index . . . 120
5.3.7.1.7 -job ramp . . . 121
5.3.7.2 wmo_sqrt_stats and wmo_fisher_stats . . . 125
5.3.7.3 vif_flag . . . 125
5.3.8 WaveletStatConfig_default . . . 125
5.3.8.1 grid_decomp_flag . . . 125
5.3.8.2 tile . . . 126
5.3.8.2.1 tile.width . . . 126
5.3.8.2.2 tile.location . . . 126
5.3.8.3 wavelet . . . 126
5.3.8.3.1 wavelet.type . . . 126
5.3.8.3.2 wavelet.member . . . 127
5.3.8.3.3 Valid Combinations of type and member . . . 127
5.3.8.4 obs_raw_wvlt_object_plots . . . 127
5.3.9 WWMCARegridConfig_default . . . 127
5.3.9.1 to_grid . . . 127
5.3.9.2 NetCDF Output Information . . . 127
5.3.9.3 max_minutes (pixel age) . . . 128
5.3.9.4 swap_endian . . . 128
5.3.9.5 write_pixel_age . . . 128

6 Tropical Cyclone Configuration Options **129**
6.1 Configuration Settings Common to Multiple Tools . . . 129
6.1.1 storm_id . . . 129
6.1.2 basin . . . 130
6.1.3 cyclone . . . 130
6.1.4 storm_name . . . 130
6.1.5 init_beg end inc exc . . . 131
6.1.6 valid_beg end inc exc . . . 131
6.1.7 init_hour . . . 132
6.1.8 lead_req . . . 132
6.1.9 version . . . 132
6.2 Settings Specific to Individual Tools . . . 132
6.2.1 TCPairsConfig_default . . . 132
6.2.1.1 model . . . 132
6.2.1.2 init_mask, valid_mask . . . 133
6.2.1.3 check_dup . . . 133
6.2.1.4 interp12 . . . 133
6.2.1.5 consensus . . . 134
6.2.1.6 lag_time . . . 134
6.2.1.7 best . . . 135
6.2.1.8 anly_track . . . 135
6.2.1.9 match_points . . . 136
6.2.1.10 dland_file . . . 136
6.2.1.11 watch_warn . . . 136
6.2.1.12 basin_map . . . 136
6.2.2 TCStatConfig_default . . . 137
6.2.2.1 amodel, bmodel . . . 137
6.2.2.2 init valid_hour lead req . . . 137
6.2.2.3 init_mask, valid_mask . . . 138
6.2.2.4 line_type . . . 138
6.2.2.5 track_watch_warn . . . 138
6.2.2.6 column_thresh_name_and_val . . . 139
6.2.2.7 column_str_name, column_str_val . . . 139
6.2.2.8 column_str_name val . . . 139
6.2.2.9 init_thresh_name, init_thresh_val . . . 140
6.2.2.10 init_str_name, init_str_val . . . 140
6.2.2.11 init_str_exc_name and _exc_val . . . 140
6.2.2.12 water_only . . . 141
6.2.2.13 rirw . . . 141
6.2.2.14 landfall beg end . . . 142
6.2.2.15 event_equal . . . 142
6.2.2.16 event_equal_lead . . . 143
6.2.2.17 out_int_mask . . . 143
6.2.2.18 out_valid_mask . . . 143
6.2.2.19 job . . . 143
6.2.2.19.1 -job filter . . . 144
6.2.2.19.2 -job summary . . . 145
6.2.2.19.3 -job rirw . . . 145

6.2.2.19.4 -job probrirw . . . 146
6.2.3 TCGenConfig_default . . . 147
6.2.3.1 init_freq . . . 147
6.2.3.2 lead_window . . . 147
6.2.3.3 min_duration . . . 147
6.2.3.4 fcst_genesis . . . 147
6.2.3.5 best_genesis . . . 148
6.2.3.6 oper_genesis . . . 148
6.2.3.7 filter . . . 148
6.2.3.8 desc . . . 148
6.2.3.9 model . . . 149
6.2.3.10 init_beg, init_end . . . 149
6.2.3.11 valid_beg, valid_end . . . 149
6.2.3.12 lead . . . 149
6.2.3.13 vx_mask . . . 149
6.2.3.14 dland_thresh . . . 149
6.2.3.15 genesis_window . . . 150
6.2.3.16 genesis_radius . . . 150
6.2.3.17 ci_alpha . . . 150
6.2.3.18 output_flag . . . 150

7 Re-Formatting of Point Observations 151
7.1 PB2NC Tool . . . 151
7.1.1 pb2nc Usage . . . 151
7.1.1.1 Required Arguments for pb2nc . . . 152
7.1.1.2 Optional Arguments for pb2nc . . . 152
7.1.2 pb2nc Configuration File . . . 153
7.1.3 pb2nc Output . . . 158
7.2 ASCII2NC Tool . . . 160
7.2.1 ascii2nc Usage . . . 161
7.2.1.1 Required Arguments for ascii2nc . . . 162
7.2.1.2 Optional Arguments for ascii2nc . . . 162
7.2.2 ascii2nc Configuration File . . . 163
7.2.3 ascii2nc Output . . . 164
7.3 MADIS2NC Tool . . . 164
7.3.1 madis2nc Usage . . . 164
7.3.1.1 Required Arguments for madis2nc . . . 165
7.3.1.2 Optional Arguments for madis2nc . . . 165
7.3.2 madis2nc Configuration File . . . 166
7.3.3 madis2nc Output . . . 167
7.4 LIDAR2NC Tool . . . 167
7.4.1 lidar2nc Usage . . . 167
7.4.1.1 Required Arguments for lidar2nc . . . 167
7.4.1.2 Optional Arguments for lidar2nc . . . 168
7.4.2 lidar2nc Output . . . 168
7.5 IODA2NC Tool . . . 169
7.5.1 ioda2nc Usage . . . 169
7.5.1.1 Required Arguments for ioda2nc . . . 169

7.5.1.2 Optional Arguments for ioda2nc . . . 169
7.5.2 ioda2nc Configuration File . . . 170
7.5.3 ioda2nc Output . . . 172
7.6 Point2Grid Tool . . . 172
7.6.1 point2grid Usage . . . 172
7.6.1.1 Required Arguments for point2grid . . . 172
7.6.1.2 Optional Arguments for point2grid . . . 173
7.6.2 point2grid Output . . . 175
7.6.3 point2grid Configuration File . . . 175
7.7 Point NetCDF to ASCII Python Utility . . . 176
7.8 IABP retrieval Python Utilities . . . 176

8 Re-Formatting of Gridded Fields 179
8.1 Pcp-Combine Tool . . . 179
8.1.1 pcp_combine Usage . . . 180
8.1.1.1 Required Arguments for the pcp_combine . . . 181
8.1.1.2 Optional Arguments for pcp_combine . . . 181
8.1.1.3 Required Arguments for the pcp_combine Sum Command . . . 181
8.1.1.4 Optional Arguments for pcp_combine Sum Command . . . 182
8.1.1.5 Required Arguments for the pcp_combine Derive Command . . . 182
8.1.1.6 Input Files for pcp_combine Add, Subtract, and Derive Commands . . . 182
8.1.2 pcp_combine Output . . . 184
8.2 Regrid-Data-Plane Tool . . . 184
8.2.1 regrid_data_plane Usage . . . 185
8.2.1.1 Required Arguments for regrid_data_plane . . . 185
8.2.1.2 Optional Arguments for regrid_data_plane . . . 185
8.2.2 Automated Regridding within Tools . . . 186
8.3 Shift-Data-Plane Tool . . . 186
8.3.1 shift_data_plane Usage . . . 187
8.3.1.1 Required Arguments for shift_data_plane . . . 187
8.3.1.2 Optional Arguments for shift_data_plane . . . 187
8.4 MODIS regrid Tool . . . 188
8.4.1 modis_regrid Usage . . . 188
8.4.1.1 Required Arguments for modis_regrid . . . 189
8.4.1.2 Optional Arguments for modis_regrid . . . 189
8.5 WWMCA Tool Documentation . . . 190
8.5.1 wwmca_plot Usage . . . 191
8.5.1.1 Required Arguments for wwmca_plot . . . 191
8.5.1.2 Optional Arguments for wwmca_plot . . . 191
8.5.2 wwmca_regrid Usage . . . 192
8.5.2.1 Required Arguments for wwmca_regrid . . . 193
8.5.2.2 Optional Arguments for wwmca_regrid . . . 193
8.5.3 wwmca_regrid Configuration File . . . 193

9 Gen-Ens-Prod Tool 195
9.1 Introduction . . . 195
9.2 Scientific and Statistical Aspects . . . 195
9.2.1 Ensemble Forecasts Derived from a Set of Deterministic Ensemble Members . . . 195

9.2.2 Climatology Data . . . 196
9.3 Practical Information . . . 196
9.3.1 gen_ens_prod Usage . . . 197
9.3.2 Required Arguments gen_ens_prod . . . 197
9.3.3 Optional Arguments for gen_ens_prod . . . 197
9.3.4 gen_ens_prod Configuration File . . . 198
9.3.5 gen_ens_prod Output . . . 202

10 Spatial Masking **203**
10.1 Gen-Vx-Mask Tool . . . 203
10.1.1 gen_vx_mask Usage . . . 203
10.1.1.1 Required Arguments for gen_vx_mask . . . 204
10.1.1.2 Optional Arguments for gen_vx_mask . . . 205
10.1.1.3 Types of Masking Available in gen_vx_mask . . . 206
10.1.1.4 Logic for gen_vx_mask . . . 208
10.1.1.5 Examples for gen_vx_mask . . . 208
10.2 Feature-Relative Methods . . . 209

11 Point-Stat Tool **211**
11.1 Introduction . . . 211
11.2 Scientific and Statistical Aspects . . . 211
11.2.1 Interpolation/Matching Methods . . . 211
11.2.2 HiRA Framework . . . 215
11.2.3 SEEPS Scores . . . 216
11.2.4 Statistical Measures . . . 217
11.2.4.1 Measures for Categorical Variables . . . 217
11.2.4.2 Measures for Continuous Variables . . . 217
11.2.4.3 Measures for Probabilistic Forecasts and Dichotomous Outcomes . . . 218
11.2.4.4 Measures for Comparison Against Climatology . . . 218
11.2.5 Statistical Confidence Intervals . . . 218
11.3 Practical Information . . . 221
11.3.1 point_stat Usage . . . 221
11.3.1.1 Required Arguments for point_stat . . . 221
11.3.1.2 Optional Arguments for point_stat . . . 222
11.3.2 point_stat Configuration File . . . 222
11.3.3 point_stat Output . . . 226

12 Pair-Stat Tool **243**
12.1 Introduction . . . 243
12.2 Scientific and Statistical Aspects . . . 243
12.3 Practical Information . . . 243
12.3.1 pair_stat Usage . . . 244
12.3.1.1 Required Arguments for pair_stat . . . 244
12.3.1.2 Optional Arguments for pair_stat . . . 244
12.3.2 pair_stat Configuration File . . . 245
12.3.3 pair_stat Output . . . 249

13 Grid-Stat Tool **251**

13.1 Introduction . . . 251
13.2 Scientific and Statistical Aspects . . . 251
13.2.1 Statistical Measures . . . 251
13.2.1.1 Measures for Categorical Variables . . . 252
13.2.1.2 Measures for Continuous Variables . . . 252
13.2.1.3 Measures for Probabilistic Forecasts and Dichotomous Outcomes . . . 252
13.2.1.4 Use of a Climatology Field for Comparative Verification . . . 253
13.2.1.5 Use of Analysis Fields for Verification . . . 253
13.2.2 Statistical Confidence Intervals . . . 253
13.2.3 Grid Weighting . . . 253
13.2.4 Neighborhood Methods . . . 253
13.2.5 SEEPS Scores . . . 254
13.2.6 Fourier Decomposition . . . 254
13.2.7 Gradient Statistics . . . 255
13.2.8 Distance Maps . . . 255
13.2.9 β and G_β . . . 258
13.3 Practical Information . . . 260
13.3.1 grid_stat Usage . . . 260
13.3.1.1 Required Arguments for grid_stat . . . 260
13.3.1.2 Optional Arguments for grid_stat . . . 261
13.3.2 grid_stat Configuration File . . . 262
13.3.3 grid_stat Output . . . 267

14 Ensemble-Stat Tool **275**
14.1 Introduction . . . 275
14.2 Scientific and Statistical Aspects . . . 275
14.2.1 HiRA Framework . . . 275
14.2.2 Ensemble Statistics . . . 276
14.2.3 Climatology Data . . . 277
14.2.4 Ensemble Observation Error . . . 278
14.3 Practical Information . . . 279
14.3.1 ensemble_stat Usage . . . 279
14.3.1.1 Required Arguments ensemble_stat . . . 279
14.3.1.2 Optional Arguments for ensemble_stat . . . 280
14.3.2 ensemble_stat Configuration File . . . 281
14.3.3 ensemble_stat Output . . . 287

15 Wavelet-Stat Tool **295**
15.1 Introduction . . . 295
15.2 Scientific and Statistical Aspects . . . 296
15.2.1 The Method . . . 296
15.2.2 The Spatial Domain Constraints . . . 305
15.2.3 Aggregation of Statistics on Multiple Cases . . . 306
15.3 Practical Information . . . 306
15.3.1 wavelet_stat Usage . . . 306
15.3.1.1 Required Arguments for wavelet_stat . . . 307
15.3.1.2 Optional Arguments for wavelet_stat . . . 307
15.3.2 wavelet_stat Configuration File . . . 307

15.3.3 wavelet_stat Output . . . 310

16 GSI Tools **315**
16.1 GSID2MPR Tool . . . 315
16.1.1 gsid2mpr Usage . . . 316
16.1.1.1 Required Arguments for gsid2mpr . . . 316
16.1.1.2 Optional Arguments for gsid2mpr . . . 316
16.1.2 gsid2mpr Output . . . 317
16.2 GSIDENS2ORANK Tool . . . 318
16.2.1 gsidens2orank Usage . . . 319
16.2.1.1 Required Arguments for gsidens2orank . . . 319
16.2.1.2 Optional Arguments for gsidens2orank . . . 319
16.2.2 gsidens2orank Output . . . 320

17 Stat-Analysis Tool **323**
17.1 Introduction . . . 323
17.2 Scientific and Statistical Aspects . . . 323
17.2.1 Filter STAT Lines . . . 324
17.2.2 Summary Statistics for Columns . . . 324
17.2.3 Aggregated Values from Multiple STAT Lines . . . 325
17.2.4 Aggregate STAT Lines and Produce Aggregated Statistics . . . 325
17.2.5 Skill Score Index . . . 326
17.2.6 GO Index . . . 327
17.2.7 CBS Index . . . 328
17.2.8 Ramp Events . . . 328
17.2.9 Wind Direction Statistics . . . 328
17.3 Practical Information . . . 329
17.3.1 stat_analysis Usage . . . 329
17.3.1.1 Required Arguments for stat_analysis . . . 330
17.3.1.2 Optional Arguments for stat_analysis . . . 330
17.3.2 stat_analysis Configuration File . . . 331
17.3.3 stat-analysis Tool Output . . . 339
17.3.3.1 Job: filter . . . 339
17.3.3.2 Job: summary . . . 339
17.3.3.3 Job: aggregate . . . 340
17.3.3.4 Job: aggregate_stat . . . 341
17.3.3.5 Job: ss_index, go_index, cbs_index . . . 341
17.3.3.6 Job: ramp . . . 342

18 Series-Analysis Tool **343**
18.1 Introduction . . . 343
18.2 Practical Information . . . 343
18.2.1 series_analysis Usage . . . 344
18.2.1.1 Required Arguments series_stat . . . 344
18.2.1.2 Optional Arguments for series_analysis . . . 344
18.2.2 series_analysis Output . . . 345
18.2.3 series_analysis Configuration File . . . 346

19 Grid-Diag Tool **349**
19.1 Introduction . . . 349
19.2 Practical Information . . . 349
19.2.1 grid_diag Usage . . . 349
19.2.1.1 Required Arguments for grid_diag . . . 350
19.2.1.2 Optional Arguments for grid_diag . . . 350
19.2.2 grid_diag Configuration File . . . 350
19.2.3 grid_diag Output File . . . 351

20 MODE Tool **353**
20.1 Introduction . . . 353
20.2 Scientific and Statistical Aspects . . . 354
20.2.1 Resolving Objects . . . 354
20.2.2 Attributes . . . 356
20.2.3 Fuzzy Logic . . . 356
20.2.4 Summary Statistics . . . 357
20.2.5 Multi-Variate MODE . . . 357
20.3 Practical Information . . . 359
20.3.1 mode Usage . . . 359
20.3.1.1 Required Arguments for mode . . . 360
20.3.1.2 Optional Arguments for mode . . . 360
20.3.2 mode Configuration File . . . 361
20.3.3 mode Output . . . 369

21 MODE-Analysis Tool **377**
21.1 Introduction . . . 377
21.2 Scientific and Statistical Aspects . . . 377
21.3 Practical Information . . . 378
21.3.1 mode_analysis Usage . . . 378
21.3.1.1 Required Arguments for mode_analysis: . . . 378
21.3.1.2 Optional Arguments for mode_analysis . . . 379
21.3.1.3 Analysis Options . . . 379
21.3.1.4 MODE Command Line Options . . . 379
21.3.1.5 Toggles . . . 379
21.3.1.6 Multiple-Set String Options . . . 380
21.3.1.7 Multiple-Set Integer Options . . . 381
21.3.1.8 Integer Max/Min Options . . . 381
21.3.1.9 Date/Time Max/Min Options . . . 382
21.3.1.10 Floating-Point Max/Min Options . . . 383
21.3.1.11 Miscellaneous Options . . . 386
21.3.2 mode_analysis Configuration File . . . 386
21.3.3 mode_analysis Output . . . 386

22 MODE Time Domain Tool **387**
22.1 Introduction . . . 387
22.1.1 Motivation . . . 387
22.2 Scientific and Statistical Aspects . . . 389
22.2.1 Attributes . . . 389

22.2.2 Convolution . . . 389
22.2.3 3D Single Attributes . . . 390
22.2.4 3D Pair Attributes . . . 392
22.2.5 2D Constant-Time Attributes . . . 393
22.2.6 Matching and Merging . . . 394
22.3 Practical Information . . . 396
22.3.1 MTD Input . . . 396
22.3.2 MTD Usage . . . 396
22.3.2.1 Required Arguments for mtd . . . 396
22.3.2.2 Optional Arguments for mtd . . . 397
22.3.3 MTD Configuration File . . . 397
22.3.4 mtd Output . . . 400

23 MET-TC Overview **405**
23.1 Introduction . . . 405
23.2 MET-TC Components . . . 405
23.3 Input Data Format . . . 406
23.4 Output Data Format . . . 408

24 TC-DLand Tool **409**
24.1 Introduction . . . 409
24.2 Input/Output Format . . . 409
24.3 Practical Information . . . 410
24.3.1 tc_dland Usage . . . 410
24.3.1.1 Required Arguments for tc_dland . . . 410
24.3.1.2 Optional Arguments for tc_dland . . . 410

25 TC-Pairs Tool **411**
25.1 Introduction . . . 411
25.2 Scientific and Statistical Aspects . . . 411
25.2.1 TC Diagnostics . . . 411
25.3 Practical Information . . . 413
25.3.1 tc_pairs Usage . . . 413
25.3.1.1 Required Arguments for tc_pairs . . . 413
25.3.1.2 Optional Arguments for tc_pairs . . . 414
25.3.2 tc_pairs Configuration File . . . 415
25.3.3 tc_pairs Output . . . 422

26 TC-Diag Tool **427**
26.1 Introduction . . . 427
26.2 Practical Information . . . 428
26.2.1 tc_diag Usage . . . 428
26.2.1.1 Required Arguments for tc_diag . . . 429
26.2.1.2 Optional Arguments for tc_diag . . . 429
26.2.2 tc_diag Configuration File . . . 429
26.2.2.1 Configuring Input Tracks and Time . . . 429
26.2.2.2 Configuring Domain Information . . . 430
26.2.2.3 Configuring Data Censoring and Conversion Options . . . 431

26.2.2.4 Configuring regridding options . . . 431
26.2.2.5 Configuring Fields, Levels, and Domains . . . 432
26.2.2.6 Configuring Vortex Removal Option . . . 432
26.2.2.7 Configuring Data Input and Output Options . . . 433
26.2.3 tc_diag Output . . . 433

27 TC-Stat Tool **439**
27.1 Introduction . . . 439
27.2 Statistical Aspects . . . 439
27.2.1 Filter TCST Lines . . . 439
27.2.2 Summary Statistics for Columns . . . 440
27.2.2.1 Frequency of Superior Performance . . . 440
27.2.2.2 Time-Series Independence . . . 440
27.2.3 Rapid Intensification/Weakening . . . 441
27.2.4 Probability of Rapid Intensification . . . 441
27.3 Practical Information . . . 442
27.3.1 tc_stat Usage . . . 442
27.3.1.1 Required Arguments for tc_stat . . . 442
27.3.1.2 Optional Arguments for tc_stat . . . 442
27.3.1.3 tc_stat Configuration File . . . 443
27.3.2 tc_stat Output . . . 449

28 TC-Gen Tool **453**
28.1 Introduction . . . 453
28.2 Statistical Aspects . . . 453
28.3 Practical Information . . . 454
28.3.1 tc_gen Usage . . . 454
28.3.1.1 Required Arguments for tc_gen . . . 455
28.3.1.2 Optional Arguments for tc_gen . . . 455
28.3.1.3 Scoring Logic . . . 456
28.3.2 tc_gen Configuration File . . . 458
28.3.3 tc_gen Output . . . 465

29 TC-RMW Tool **469**
29.1 Introduction . . . 469
29.2 Practical Information . . . 469
29.2.1 tc_rmw Usage . . . 469
29.2.1.1 Required Arguments for tc_rmw . . . 470
29.2.1.2 Optional Arguments for tc_rmw . . . 470
29.2.2 tc_rmw Configuration File . . . 470
29.2.3 tc_rmw Output File . . . 472

30 RMW-Analysis Tool **473**
30.1 Introduction . . . 473
30.2 Practical Information . . . 473
30.2.1 rmw_analysis Usage . . . 473
30.2.1.1 Required Arguments for rmw_analysis . . . 474
30.2.1.2 Optional Arguments for rmw_analysis . . . 474

30.2.2 rmw_analysis Configuration File . . . 474
30.2.3 rmw_analysis Output File . . . 476

31 Plotting and Graphics Support **477**
31.1 Plotting Utilities . . . 477
31.1.1 plot_point_obs Usage . . . 477
31.1.1.1 Required Arguments for plot_point_obs . . . 478
31.1.1.2 Optional Arguments for plot_point_obs . . . 478
31.1.2 plot_point_obs Configuration File . . . 479
31.1.3 plot_data_plane Usage . . . 482
31.1.3.1 Required Arguments for plot_data_plane . . . 482
31.1.3.2 Optional Arguments for plot_data_plane . . . 482
31.1.4 plot_mode_field Usage . . . 483
31.1.4.1 Required Arguments for plot_mode_field . . . 483
31.1.4.2 Optional Arguments for plot_mode_field . . . 483
31.2 Examples of Plotting MET Output . . . 485
31.2.1 Grid-Stat Tool Examples . . . 485
31.2.2 MODE Tool Examples . . . 485
31.2.3 TC-Stat Tool Example . . . 488

32 References **491**

33 Appendix A FAQs & How do I . . . ? **501**
33.1 Frequently Asked Questions . . . 501
33.1.1 File-IO . . . 501
33.1.1.1 Q. How do I improve the speed of MET tools using Gen-Vx-Mask? . . . 501
33.1.1.2 Q. How do I use map_data? . . . 501
33.1.1.3 Q. How can I understand the number of matched pairs? . . . 502
33.1.1.4 Q. What types of NetCDF files can MET read? . . . 503
33.1.1.5 Q. How do I choose a time slice in a NetCDF file? . . . 504
33.1.1.6 Q. How do I use the UNIX time conversion? . . . 504
33.1.1.7 Q. Does MET use a fixed-width output format for its ASCII output files? . . . 505
33.1.1.8 Q. Do the ASCII output files created by MET use scientific notation? . . . 505
33.1.2 Gen-Vx-Mask . . . 505
33.1.2.1 Q. I have a list of stations to use for verification. I also have a poly region defined. If I specify both of these should the result be a union of them? . . . 505
33.1.2.2 Q. How do I define a masking region with a GFS file? . . . 506
33.1.3 Grid-Stat . . . 507
33.1.3.1 Q. How do I define a complex masking region? . . . 507
33.1.3.2 Q. How do I use neighborhood methods to compute fraction skill score? . . . 508
33.1.3.3 Q. Is an example of verifying forecast probabilities? . . . 508
33.1.3.4 Q. What is an example of using Grid-Stat with regridding and masking turned on? . . . 509
33.1.3.5 Q. How do I use one mask for the forecast field and a different mask for the observation field? . . . 510
33.1.4 Pcp-Combine . . . 511
33.1.4.1 Q. How do I add and subtract with Pcp-Combine? . . . 511

33.1.4.2 Q. How do I combine 12-hour accumulated precipitation from two different initialization times? . . . 512
33.1.4.3 Q. How do I correct a precipitation time range? . . . 513
33.1.4.4 Q. How do I use Pcp-Combine as a pass-through to simply reformat from GRIB to NetCDF or to change output variable name? . . . 514
33.1.4.5 Q. How do I use "-pcprx" to run a project faster? . . . 514
33.1.4.6 Q. How do I enter the time format correctly? . . . 515
33.1.4.7 Q. How do I use Pcp-Combine when my GRIB data doesn't have the appropriate accumulation interval time range indicator? . . . 515
33.1.4.8 Q. How do I use "-sum", "-add", and "-subtract" to achieve the same accumulation interval? . . . 516
33.1.4.9 Q. What is the difference between "-sum" vs. "-add"? . . . 517
33.1.4.10 Q. How do I select a specific GRIB record? . . . 518
33.1.5 Plot-Data-Plane . . . 518
33.1.5.1 Q. How do I inspect Gen-Vx-Mask output? . . . 518
33.1.5.2 Q. How do I specify the GRIB version? . . . 518
33.1.5.3 Q. How do I test the variable naming convention? (Record number example.) 519
33.1.5.4 Q. How do I compute and verify wind speed? . . . 519
33.1.6 Stat-Analysis . . . 520
33.1.6.1 Q. How does '-aggregate_stat' work? . . . 520
33.1.6.2 Q. What is the best way to average the FSS scores within several days or even several months using 'Aggregate to Average Scores'? . . . 521
33.1.6.3 Q. How do I use '-by' to capture unique entries? . . . 521
33.1.6.4 Q. How do I use '-filter' to refine my output? . . . 522
33.1.6.5 Q. How do I use the "-by" flag to stratify results? . . . 522
33.1.6.6 Q. How do I speed up run times? . . . 523
33.1.7 TC-Stat . . . 523
33.1.7.1 Q. How do I use the "-by" flag to stratify results? . . . 523
33.1.7.2 Q. How do I use rapid intensification verification? . . . 524
33.1.8 Utilities . . . 524
33.1.8.1 Q. What would be an example of scripting to call MET? . . . 524
33.1.8.2 Q. How do I convert TRMM data files? . . . 525
33.1.8.3 Q. How do I convert a PostScript to png? . . . 526
33.1.8.4 Q. How does pairwise differences using plot_tcmpr.R work? . . . 526
33.1.9 Miscellaneous . . . 527
33.1.9.1 Q. Regrid-Data-Plane - How do I define a LatLon grid? . . . 527
33.1.9.2 Q. Pre-processing - How do I use wgrib2, pcp_combine regrid and reformat to format NetCDF files? . . . 527
33.1.9.3 Q. TC-Pairs - How do I get rid of WARNING: TrackInfo Using Specify Model Suffix? . . . 528
33.1.9.4 Q. Why is the grid upside down? . . . 529
33.1.9.5 Q. Why was the MET written largely in C++ instead of FORTRAN? . . . 530
33.1.9.6 Q. How does MET differ from the previously mentioned existing verification packages? . . . 530
33.1.9.7 Q. Will the MET work on data in native model coordinates? . . . 530
33.1.9.8 Q. How do I get help if my questions are not answered in the User's Guide? . 530
33.1.9.9 Q. What graphical features does MET provide? . . . 531
33.1.9.10 Q. How do I find the version of the tool I am using? . . . 531

33.1.9.11Q. What are MET's conventions for latitude, longitude, azimuth and bearing angles? . . . 531
33.2 Troubleshooting . . . 532
33.2.1 MET Won't Compile . . . 532
33.2.2 BUFRLIB Errors During MET Installation . . . 532
33.2.3 Command Line Double Quotes . . . 533
33.2.4 Environment Variable Settings . . . 533
33.2.5 NetCDF Install Issues . . . 533
33.2.6 Error While Loading Shared Libraries . . . 534
33.2.7 General Troubleshooting . . . 534
33.3 Where to Get Help . . . 534
33.4 How to Contribute Code . . . 535

34 Appendix B Map Projections, Grids, and Polylines 537
34.1 Map Projections . . . 537
34.2 Grid Specification Strings . . . 537
34.2.1 Lambert Conformal Grid . . . 538
34.2.2 Lambert Azimuthal Equal Area Grid . . . 538
34.2.3 Polar Stereographic Grid . . . 538
34.2.4 Lat/Lon Grid . . . 538
34.2.5 Rotated Lat/Lon Grid . . . 539
34.2.6 Mercator Grid . . . 539
34.2.7 Gaussian Grid . . . 539
34.2.8 Range/Azimuth Grid . . . 539
34.2.9 Semi Lat/Lon Grid . . . 540
34.3 Named Grids . . . 540
34.4 Polylines for NCEP Regions . . . 540

35 Appendix C Verification Measures 543
35.1 Which statistics are the same, but with different names? . . . 543
35.2 MET Verification Measures for Categorical (Dichotomous) Variables . . . 544
35.2.1 TOTAL . . . 545
35.2.2 Base Rate . . . 545
35.2.3 Mean Forecast . . . 545
35.2.4 Accuracy . . . 545
35.2.5 Frequency Bias . . . 545
35.2.6 H_RATE . . . 546
35.2.7 Probability of Detection (POD) . . . 546
35.2.8 Probability of False Detection (POFD) . . . 546
35.2.9 Probability of Detection of the Non-Event (PODn) . . . 546
35.2.10 False Alarm Ratio (FAR) . . . 547
35.2.11 Critical Success Index (CSI) . . . 547
35.2.12 Gilbert Skill Score (GSS) . . . 547
35.2.13 Hanssen-Kuipers Discriminant (HK) . . . 547
35.2.14 Heidke Skill Score (HSS) . . . 548
35.2.15 Heidke Skill Score - Expected Correct (HSS_EC) . . . 548
35.2.16 Odds Ratio (OR) . . . 548
35.2.17 Logarithm of the Odds Ratio (LODDS) . . . 549

35.2.18 Odds Ratio Skill Score (ORSS) . . . 549
35.2.19 Extreme Dependency Score (EDS) . . . 549
35.2.20 Extreme Dependency Index (EDI) . . . 549
35.2.21 Symmetric Extreme Dependency Score (SEDS) . . . 550
35.2.22 Symmetric Extremal Dependency Index (SEDI) . . . 550
35.2.23 Bias-Adjusted Gilbert Skill Score (BAGSS) . . . 550
35.2.24 Economic Cost Loss Relative Value (ECLV) . . . 550
35.2.25 Stable Equitable Error in Probability Space (SEEPS) . . . 551
35.3 MET Verification Measures for Continuous Variables . . . 551
35.3.1 Mean Forecast . . . 552
35.3.2 Mean Observation . . . 552
35.3.3 Forecast Standard Deviation . . . 552
35.3.4 Observation Standard Deviation . . . 552
35.3.5 Pearson Correlation Coefficient . . . 552
35.3.6 Spearman Rank Correlation Coefficient (ρ_s) . . . 553
35.3.7 Kendall's Tau Statistic (τ) . . . 553
35.3.8 Mean Error (ME) . . . 554
35.3.9 Mean Error Squared (ME2) . . . 554
35.3.10 Multiplicative Bias . . . 554
35.3.11 Mean-Squared Error (MSE) . . . 554
35.3.12 Root-Mean-Squared Error (RMSE) . . . 554
35.3.13 Scatter Index (SI) . . . 555
35.3.14 Standard Deviation of the Error . . . 555
35.3.15 Bias-Corrected MSE . . . 555
35.3.16 Mean Absolute Error (MAE) . . . 555
35.3.17 InterQuartile Range of the Errors (IQR) . . . 555
35.3.18 Median Absolute Deviation (MAD) . . . 556
35.3.19 Mean Squared Error Skill Score . . . 556
35.3.20 Root-Mean-Squared Forecast Anomaly . . . 556
35.3.21 Root-Mean-Squared Observation Anomaly . . . 556
35.3.22 Percentiles of the Errors . . . 556
35.3.23 Anomaly Correlation Coefficient . . . 557
35.3.24 Partial Sums Lines (SL1L2, SAL1L2, VL1L2, VAL1L2) . . . 557
35.3.25 Scalar L1 and L2 Values . . . 558
35.3.26 Scalar Anomaly L1 and L2 Values . . . 558
35.3.27 Vector L1 and L2 Values . . . 559
35.3.28 Vector Anomaly L1 and L2 Values . . . 560
35.3.29 Gradient Values . . . 560
35.4 MET Verification Measures for Probabilistic Forecasts . . . 562
35.4.1 Reliability . . . 562
35.4.2 Resolution . . . 563
35.4.3 Uncertainty . . . 563
35.4.4 Brier Score . . . 563
35.4.5 Brier Skill Score (BSS) . . . 564
35.4.6 OY_TP - Observed Yes Total Proportion . . . 564
35.4.7 ON_TP - Observed No Total Proportion . . . 564
35.4.8 Calibration . . . 564
35.4.9 Refinement . . . 565

35.4.10 Likelihood . . . 565
35.4.11 Base Rate . . . 565
35.4.12 Reliability Diagram . . . 565
35.4.13 Receiver Operating Characteristic . . . 567
35.4.14 Area Under the ROC Curve (AUC) . . . 568
35.5 MET Verification Measures for Ensemble Forecasts . . . 569
35.5.1 RPS . . . 569
35.5.2 CRPS . . . 569
35.5.3 Ensemble Mean Absolute Difference . . . 570
35.5.4 CRPS Skill Score . . . 571
35.5.5 Bias Ratio . . . 571
35.5.6 IGN . . . 571
35.5.7 PIT . . . 572
35.5.8 Observation Error Logarithmic Scoring Rules . . . 572
35.5.9 RANK . . . 573
35.5.10 SPREAD . . . 573
35.6 MET Verification Measures for Neighborhood Methods . . . 573
35.6.1 Fractions Brier Score . . . 574
35.6.2 Fractions Skill Score . . . 574
35.6.3 Asymptotic Fractions Skill Score . . . 575
35.6.4 Uniform Fractions Skill Score . . . 575
35.6.5 Forecast Rate . . . 575
35.6.6 Observation Rate . . . 575
35.7 MET Verification Measures for Distance Map Methods . . . 575
35.7.1 Baddeley's Δ Metric and Hausdorff Distance . . . 576
35.7.2 Mean-Error Distance . . . 576
35.7.3 Pratt's Figure of Merit . . . 577
35.7.4 Zhu's Measure . . . 577
35.7.5 G and G_β . . . 578
35.8 Calculating Percentiles . . . 578

36 Appendix D Confidence Intervals 581

37 Appendix E WWMCA Tools 585

38 Appendix F Python Embedding 589
38.1 Introduction . . . 589
38.2 Compiling MET for Python Embedding . . . 589
38.3 Controlling Which Python MET Uses When Running . . . 590
38.4 Data Structures Supported by Python Embedding . . . 591
38.4.1 Python Embedding for 2D Gridded Dataplanes . . . 592
38.4.1.1 Python Script Requirements for 2D Gridded Dataplanes . . . 592
38.4.1.2 Attributes for 2D Gridded Dataplanes . . . 592
38.4.1.3 Running Python Embedding for 2D Gridded Dataplanes . . . 596
38.4.1.4 Special Case for Gen-Ens-Prod, Ensemble-Stat, Series-Analysis, and MTD . . . 597
38.4.1.5 Examples of Python Embedding for 2D Gridded Dataplanes . . . 598
38.4.2 Python Embedding for Point Observations . . . 599
38.4.2.1 Python Script Requirements for Point Observations . . . 599

38.4.2.2 Running Python Embedding for Point Observations . . . 600
38.4.2.3 Examples of Python Embedding for Point Observations . . . 601
38.4.3 Python Embedding for STAT Data . . . 601
38.4.3.1 Python Script Requirements for MPR Data . . . 602
38.4.3.2 Running Python Embedding for MPR Data . . . 603
38.5 MET Python Package . . . 603

39 Appendix G Vectors and Vector Statistics 605

40 Appendix H Unstructured Grids 611
40.1 Introduction . . . 611
40.2 Unstructured Grid Files . . . 611
40.3 Required Unstructured Grid Metadata . . . 612
40.4 Unstructured Grid Configuration Entries . . . 612
40.4.1 ugrid_dataset . . . 612
40.4.2 ugrid_max_distance_km . . . 613
40.4.3 ugrid_coordinates_file . . . 613
40.4.4 ugrid_metadata_map . . . 613
40.5 Unstructured Grid Limitations . . . 613

Foreword: A note to MET users

This User's guide is provided as an aid to users of the Model Evaluation Tools (MET). MET is a set of verification tools developed by the Developmental Testbed Center (DTC) for use by the numerical weather prediction community to help them assess and evaluate the performance of numerical weather predictions. It is also the core component of the unified METplus verification framework. More details about METplus can be found on the METplus website.

It is important to note here that MET is an evolving software package. This documentation describes the 12.2.2 release dated 2026-08-28. Previous releases of MET have occurred each year since 2008. Intermediate releases may include bug fixes. MET is also able to accept new modules contributed by the community. If you have code you would like to contribute, we will gladly consider your contribution. Please create a post in the METplus GitHub Discussions Forum. We will then determine the maturity of the new verification method and coordinate the inclusion of the new module in a future version.

Model Evaluation Tools (MET) TERMS OF USE - IMPORTANT!

Copyright 2026, UCAR/NCAR, NOAA, CSU/CIRA, and CU/CIRES Licensed under the Apache License, Version 2.0 (the "License"); You may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Citations

The citation for this User's Guide should be:

Soh, H., L. Goodrich, B. Brown, R. Bullock, J. Halley Gotway, K. Newman, J. Opatz, T. Jensen, J. Prestopnik, 2026: The MET Version 12.2.2 User's Guide. Developmental Testbed Center. Available at: https://github.com/dtcenter/MET/releases

Acknowledgments

We thank the National Science Foundation (NSF) along with three organizations within the National Oceanic and Atmospheric Administration (NOAA): 1) Office of Atmospheric Research (OAR); 2) Next Generation Global Prediction System project (NGGPS); and 3) United State Weather Research Program (USWRP), the United States Air Force (USAF), and the United States Department of Energy (DOE) for their support of this work. Funding for the development of MET-TC is from the NOAA's Hurricane Forecast Improvement Project (HFIP) through the Developmental Testbed Center (DTC). Funding for the expansion of capability to address many methods pertinent to global and climate simulations was provided by NOAA's Next Generation Global Prediction System (NGGPS) and NSF Earth System Model 2 (EaSM2) projects. We would like to thank James Franklin at the National Hurricane Center (NHC) for his insight into the original development of the existing NHC verification software. Thanks also go to the staff at the Developmental Testbed Center for their help, advice, and many types of support. We released METv1.0 in January 2008 and would not have made a decade of cutting-edge verification support without those who participated in the original MET planning workshops and the now dis-banded verification advisory group (Mike Baldwin, Matthew Sittel, Elizabeth Ebert, Geoff DiMego, Chris Davis, and Jason Knievel).

The National Center for Atmospheric Research (NCAR) is sponsored by NSF. The DTC is sponsored by the National Oceanic and Atmospheric Administration (NOAA), the United States Air Force, and the National Science Foundation (NSF). NCAR is sponsored by the National Science Foundation (NSF).

Chapter 1

Overview of MET

1.1 Purpose and Organization of the User's Guide

The goal of this User's Guide is to provide basic information for users of the Model Evaluation Tools (MET) to enable them to apply MET to their datasets and evaluation studies. MET was originally designed for application to the post-processed output of the Weather Research and Forecasting (WRF) model. However, MET may also be used for the evaluation of forecasts from other models or applications, including the Unified Forecast System (UFS), and the System for Integrated Modeling of the Atmosphere (SIMA) if certain file format definitions (described in this document) are followed.

The MET User's Guide is organized as follows. Section 1 provides an overview of MET and its components. Section 3 contains basic information about how to get started with MET - including system requirements, required software (and how to obtain it), how to download MET, and information about compilers, libraries, and how to build the code. Section 4 - Section 10 focuses on the data needed to run MET, including formats for forecasts, observations, and output. These sections also document the reformatting and masking tools available in MET. Section 11 - Section 16 focuses on the main statistics modules contained in MET, including the Point-Stat, Grid-Stat, Ensemble-Stat, Wavelet-Stat and GSI Diagnostic Tools. These sections include an introduction to the statistical verification methodologies utilized by the tools, followed by a section containing practical information, such as how to set up configuration files and the format of the output. Section 17 and Section 18 focus on the analysis modules, Stat-Analysis and Series-Analysis, which aggregate the output statistics from the other tools across multiple cases. Section 20 - Section 22 describes a suite of object-based tools, including MODE, MODE-Analysis, and MODE-TD. Section 23 - Section 30 describes tools focused on tropical cyclones, including MET-TC Overview, TC-DLand, TC-Diag, TC-Pairs, TC-Stat, TC-Gen, TC-RMW and RMW-Analysis. Finally, Section 31 includes plotting tools included in the MET release for checking and visualizing data, as well as some additional tools and information for plotting MET results. The appendices provide further useful information, including answers to some typical questions (Appendix A, Section 33) and links and information about map projections, grids, and polylines (Appendix B, Section 34). Appendix C, Section 35 and Appendix D, Section 36 provide more information about the verification measures and confidence intervals that are provided by MET. Sample code that can be used to perform analyses on the output of MET and create particular types of plots of verification results is posted on the MET website). Note that the MET development group also accepts contributed analysis and plotting scripts which may be posted on the MET website for use by the community. It should be noted there are References (Section 32) in this User's Guide as well.

The remainder of this section includes information about the context for MET development, as well as information on the design principles used in developing MET. In addition, this section includes an overview of the MET package and its specific modules.

1.2 The Developmental Testbed Center (DTC)

MET has been developed, and will be maintained and enhanced, by the Developmental Testbed Center (DTC). The main goal of the DTC is to serve as a bridge between operations and research, to facilitate the activities of these two important components of the numerical weather prediction (NWP) community. The DTC provides an environment that is functionally equivalent to the operational environment in which the research community can test model enhancements; the operational community benefits from DTC testing and evaluation of models before new models are implemented operationally. MET serves both the research and operational communities in this way - offering capabilities for researchers to test their own enhancements to models and providing a capability for the DTC to evaluate the strengths and weaknesses of advances in NWP prior to operational implementation.

The MET package is available to DTC staff, visitors, and collaborators, as well as both the US and International modeling community, for testing and evaluation of new model capabilities, applications in new environments, and so on. It is also the core component of the unified METplus verification framework. METplus details can be found on the METplus webpage.

1.3 MET Goals and Design Philosophy

The primary goal of MET development is to provide a state-of-the-art verification package to the NWP community. By "state-of-the-art" we mean that MET will incorporate newly developed and advanced verification methodologies, including new methods for diagnostic and spatial verification and new techniques provided by the verification and modeling communities. MET also utilizes and replicates the capabilities of existing systems for verification of NWP forecasts. For example, the MET package replicates existing National Center for Environmental Prediction (NCEP) operational verification capabilities (e.g., I/O, methods, statistics, data types). MET development will take into account the needs of the NWP community - including operational centers and the research and development community. Some of the MET capabilities include traditional verification approaches for standard surface and upper air variables (e.g., Equitable Threat Score, Mean Squared Error), confidence intervals for verification measures, and spatial forecast verification methods. In the future, MET will include additional state-of-the-art and new methodologies.

The MET package has been designed to be modular and adaptable. For example, individual modules can be applied without running the entire set of tools. New tools can easily be added to the MET package due to this modular design. In addition, the tools can readily be incorporated into a larger "system" that may include a database as well as more sophisticated input/output and user interfaces. Currently, the MET package is a set of tools that can easily be applied by any user on their own computer platform. A suite of Python scripts for low-level automation of verification workflows and plotting has been developed to assist users with setting up their MET-based verification. It is called METplus and may be obtained on the METplus GitHub repository.

The MET code and documentation is maintained by the DTC in Boulder, Colorado. The MET package is freely available to the modeling, verification, and operational communities, including universities, governments,

the private sector, and operational modeling and prediction centers.

1.4 MET Components

The major components of the MET package are represented in Figure 1.1. The main stages represented are input, reformatting, plotting, intermediate output, statistical analyses, and output and aggregation/analysis. Each of these stages is described further in later sections. For example, the input and output formats are discussed in Section 4 as well as in the sections associated with each of the statistics modules. MET input files are represented on the far left.

The reformatting stage of MET consists of several tools which perform a variety of functions. The ASCII2NC, PB2NC, MADIS2NC, LIDAR2NC, and IODA2NC tools read a variety of point observation input file formats and, optionally, derive time summaries for each observing location. They all write to a common NetCDF point observation file format which can be read by the other MET tools. The Point2Grid tool reads that common NetCDF point observation file format or observations provided via Python and interpolates the point data onto a user-specified grid. The Regrid-Data-Plane, Shift-Data-Plane, MODIS-Regrid, and WWMCA-Regrid tools read a variety of gridded input file formats and interpolate user-requested input fields to a user-defined output grid. While the MET statistics tools can interpolate many input file formats in-memory and on-the-fly, manually regridding upstream is sometimes useful. The Pcp-Combine tool adds, subtracts, or derives fields across multiple time steps. It is often run to accumulate precipitation amounts into a user-specified time interval - if a user would like to verify over a different time interval than is included in their forecast or observational dataset. The Gen-Vx-Mask tool provides a variety of methods for creating bitmapped masking areas. Those masks can then be used to efficiently limit verification to the interior of a user-specified region in the downstream statistics tools. The Gen-Ens-Prod tool derives basic ensemble products (mean, spread, probabilities) from multiple gridded input ensemble members. The GSI tools reformat binary GSI diagnostic data to be read by the Stat-Analysis tool.

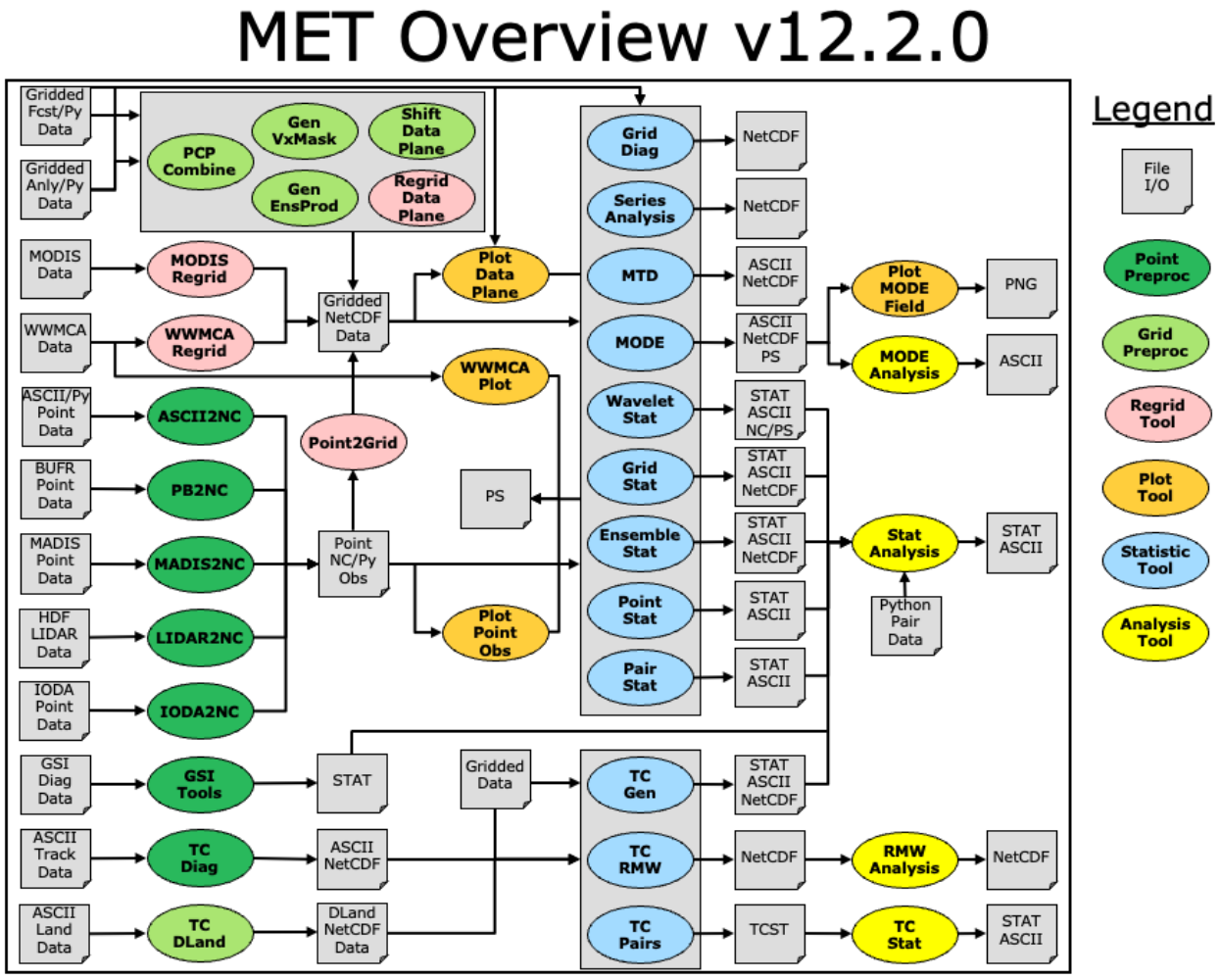

Figure 1.1: Basic representation of current MET structure and modules. Gray areas represent input and output files. Dark green areas represent point preprocessing tools. Light green areas represent grid preprocessing tools. Pink areas represent regridding tools. Orange areas represent plotting tools. Blue areas represent statistical tools. Yellow areas represent aggregation and analysis tools.

Several optional plotting utilities are provided to assist users in checking their output from the data preprocessing step. Plot-Point-Obs creates a postscript plot showing the locations of point observations. This can be quite useful for assessing whether the latitude and longitude of observation stations was specified correctly. Plot-Data-Plane produces a similar plot for gridded data. For users of the MODE object based verification methods, the Plot-MODE-Field utility will create graphics of the MODE object output. Finally, WWMCA-Plot produces a plot of the raw WWMCA data file.

The main statistical analysis components of the current version of MET are: Point-Stat, Pair-Stat, Grid-Stat, Series-Analysis, Ensemble-Stat, MODE, MODE-TD (MTD), Grid-Diag, and Wavelet-Stat.

The Point-Stat tool is used for grid-to-point verification, or verification of a gridded forecast field against point observations (i.e., surface observing stations, ACARS, rawinsondes, and other observation types that could be described as a point observation). The point observations are read from the common NetCDF point observation file format or are supplied via Python. In addition to providing traditional forecast ver-

ification scores for both continuous and categorical variables, confidence intervals are also produced using parametric and non-parametric methods. Confidence intervals take into account the uncertainty associated with verification statistics due to sampling variability and limitations in sample size. These intervals provide more meaningful information about forecast performance. For example, confidence intervals allow credible comparisons of performance between two models when a limited number of model runs is available.

The Pair-Stat tool computes a subset of the statistics and measures computed by the Point-Stat tool. However, it reads forecast and observation data that has already been paired, such as matched pair output from Point-Stat or through Python embedding and paired data from IODA files produced by the JEDI data assimilation system.

Sometimes it may be useful to verify a forecast against gridded fields (e.g., Stage IV precipitation analyses). The Grid-Stat tool produces traditional verification statistics when a gridded field is used as the observational dataset. Like the Point-Stat tool, the Grid-Stat tool also produces confidence intervals. The Grid-Stat tool also includes "neighborhood" spatial methods, such as the Fractional Skill Score (*Roberts and Lean, 2008* (page 498)). These methods are discussed in *Ebert (2008)* (page 494). The Grid-Stat tool accumulates statistics over the entire domain.

Users wishing to accumulate statistics over a time, height, or other series separately for each grid location should use the Series-Analysis tool. Series-Analysis can read any gridded matched pair data produced by the other MET tools and accumulate them, keeping each spatial location separate. Maps of these statistics can be useful for diagnosing spatial differences in forecast quality.

Ensemble-Stat compares ensemble member data to gridded analyses and/or point observations and computes measures of ensemble characteristics. The ensemble characteristics include ensemble mean and spread information, computation of rank and probability integral transform (PIT) histograms, the points for the receiver operator characteristic (ROC) and reliability diagrams, and ranked probabilities scores (RPS) and the continuous version (CRPS). When categorical thresholds are specified, Ensemble-Stat derives ensemble relative frequencies and verifies them as probability forecasts against the gridded analyses and/or point observations provided. Note that the ensemble post-processing provided in prior versions of this tool has moved to Gen-Ens-Prod.

The MODE (Method for Object-based Diagnostic Evaluation) tool also uses gridded fields as observational datasets. However, unlike the Grid-Stat tool, which applies traditional forecast verification techniques, MODE applies the object-based spatial verification technique described in *Davis et al. (2006a,b)* (page 493) and *Brown et al. (2007)* (page 492). This technique was developed in response to the "double penalty" problem in forecast verification. A forecast missed by even a small distance is effectively penalized twice by standard categorical verification scores: once for missing the event and a second time for producing a false alarm of the event elsewhere. As an alternative, MODE defines objects in both the forecast and observation fields. The objects in the forecast and observation fields are then matched and compared to one another. Applying this technique also provides diagnostic verification information that is difficult or even impossible to obtain using traditional verification measures. For example, the MODE tool can provide information about errors in location, size, and intensity.

The MODE-TD tool extends object-based analysis from two-dimensional forecasts and observations to include the time dimension. In addition to the two dimensional information provided by MODE, MODE-TD can be used to examine even more features including displacement in time, and duration and speed of moving areas of interest.

The Grid-Diag tool produces multivariate probability density functions (PDFs) that may be used either for exploring the relationship between two fields, or for the computation of percentiles generated from the

sample for use with percentile thresholding. The output from this tool requires post-processing by METplus or user-provided utilities.

The Wavelet-Stat tool decomposes two-dimensional forecasts and observations according to the Intensity-Scale verification technique described by *Casati et al. (2004)* (page 493). There are many types of spatial verification approaches and the Intensity-Scale technique belongs to the scale-decomposition (or scale-separation) verification approaches. The spatial scale components are obtained by applying a wavelet transformation to the forecast and observation fields. The resulting scale-decomposition measures error, bias and skill of the forecast on each spatial scale. Information is provided on the scale dependency of the error and skill, on the no-skill to skill transition scale, and on the ability of the forecast to reproduce the observed scale structure. The Wavelet-Stat tool is primarily used for precipitation fields. However, the tool can be applied to other variables, such as cloud fraction.

Results from the statistical analysis stage are output in ASCII, NetCDF and Postscript formats. The Point-Stat, Grid-Stat, Wavelet-Stat, and Ensemble-Stat tools create STAT (statistics) files which are tabular ASCII files ending with a ".stat" suffix. The STAT output files consist of multiple line types, each containing a different set of related statistics. The columns preceding the LINE_TYPE column are common to all lines. However, the number and contents of the remaining columns vary by line type.

The Stat-Analysis and MODE-Analysis tools aggregate the output statistics from the previous steps across multiple cases. The Stat-Analysis tool reads the STAT output of Point-Stat, Grid-Stat, Ensemble-Stat, and Wavelet-Stat and can be used to filter the STAT data and produce aggregated continuous and categorical statistics. Stat-Analysis also reads matched pair data (i.e. MPR line type) via python embedding. The MODE-Analysis tool reads the ASCII output of the MODE tool and can be used to produce summary information about object location, size, and intensity (as well as other object characteristics) across one or more cases.

Tropical cyclone forecasts and observations are quite different than numerical model forecasts, and thus they have their own set of tools. These consist of TC-DLand, TC-Diag, TC-Pairs, TC-Stat, TC-Gen, TC-RMW, and RMW-Analysis. The TC-DLand module calculates the distance to land from all locations on a specified grid. This information can be used in later modules to eliminate tropical cyclones that are over land from being included in the statistics. TC-Diag converts gridded model output into cylindrical coordinates for each storm location, calls Python scripts to compute storm-relative diagnostics, and writes ASCII output to be read by TC-Pairs. TC-Pairs matches up tropical cyclone forecasts and observations and writes all output to a file. In TC-Stat, these forecast / observation pairs are analyzed according to user preference to produce statistics. TC-Gen evaluates the performance of Tropical Cyclone genesis forecast using contingency table counts and statistics. TC-RMW performs a coordinate transformation for gridded model or analysis fields centered on the current storm location. RMW-Analysis filters and aggregates the output of TC-RMW across multiple cases.

The following sections of this MET User's Guide contain usage statements for each tool, which may be viewed if you type the name of the tool. Alternatively, the user can also type the name of the tool followed by **-help** to obtain the usage statement. Each tool also has a **-version** command line option associated with it so that the user can determine what version of the tool they are using.

1.5 Future Development Plans

MET is an evolving verification software package. New capabilities are provided in controlled, successive releases using sematic version numbering. Please refer to the issues listed in the MET GitHub repository to see our development priorities for upcoming releases.

Bugs and user-identified problems are documented as GitHub issues as they are found. Typically, bug fixes are provided in the next bugfix release for the current official realease as well as the next official release. Specific details about each bug can be found in the body and/or comments of the corresponding GitHub issue.

1.6 User Support

MET is the statistical component of the larger METplus system for which user support is provided through the METplus GitHub Discussions Forum, as described in the METplus User's Guide. Additional information about MET are provided on the MET web page.

Note: **-help** and **-version** command line options are available for all of the MET tools, and the command returns a good status of 0. Typing the name of the tool with no command line options also produces the usage statement, although the command returns a non-zero status of 1.

1.7 Fortify and SonarQube

Requirements from various government agencies that use MET have resulted in our code being analyzed by both the Fortify and SonarQube static source code analysis tools. Fortify and SonarQube analyze source code to identify for security risks, memory leaks, uninitialized variables, and other such weaknesses and bad coding practices. They categorize issue as low priority, high priority, or critical, and report these issues back to the developers for them to address. The goal is to drive the counts of both high priority and critical issues down to zero.

The MET developers are pleased to report that Fortify reports zero critical issues in the MET code. Users of the MET tools who work in high security environments can rest assured about the possibility of security risks when using MET, since the quality of the code has now been vetted by unbiased third-party experts. The MET developers continue using Fortify routinely to ensure that the critical counts remain at zero and to further reduce the counts for lower priority issues.

Chapter 2

MET Release Information

2.1 MET Release Notes

When applicable, release notes are followed by the GitHub issue number which describes the bugfix, enhancement, or new feature (MET GitHub issues). Important issues are listed **in bold** for emphasis.

2.1.1 MET Version 12.2.2 Release Notes (20260828)

Bugfixes

- Fix OpenMP 4.5 compilation errors from the GNU 8.5.0 compiler (#3359).
- Fix Grid-Stat to correct the timing information of the SEEPS data written to the NetCDF matched pairs output file (#3362).
- Fix NetCDF CF convention to support 'false_easting' and 'false_northing' for lambert conformal projections (#3374).
- Correctly handle invalid latitude values from an MPAS file (#3381).
- Refine RPS library code to use centered probability bins (#3396).
- Enable MPAS UGRID support to parse forecast lead time from diag files (#3410).
- Fix Ensemble-Stat to skip observations when the observation error table lookup fails (#3429).

2.1.2 MET Version 12.2.1 Release Notes (20260227)

Bugfixes

- Fix support for the "set_attr_grid" config option when defining the verification domain (#3293).
- Fix dependency checks and compile flag defaults in "compile_MET_all.sh" (#3317).
- Fix ASCII2NC to handle a wider range of NDBC bad data values (#3342).
- Fix support for the "file_type" option in the PCP-Combine "sum" command (#3353).

METbaseimage testing environment

- Update METbaseimage to Python 3.14 (METbaseimage#61).

Note: MET version 12.2.1 is built upon METbaseimage version 3.4.9 in which the Python version was upgraded from 3.12 to 3.14. This change only impacts MET's Docker development and testing environment. MET version 12.1.2 should continue working well with Python version 3.12.

2.1.3 MET Version 12.2.0 Release Notes (20251113)

Enhancements

- Extend time range support with NetCDF-CF files (#2836).
- Enhance ASCII2NC to recursively search input directories for specific file types for use with ISMN data (#3148).
- Enhance RMW-Analysis to filter individual track points rather than entire tracks (#3168).
- Enhance madis2nc so that it writes variable names (#3172).
- **Enhance Point-Stat and Ensemble-Stat to apply orographic corrections when verifying surface temperatures and heights (e.g. cloud base) against point observations** (#3174).
- Enhance Pair-Stat to support data censoring and conversion configuration options (#3186).
- Add CVE scanning to the release-docker-images.yml workflow (#3198).
- **Support multiple vertical level variables for CF NetCDF input** (#3226).
- Add new pre-defined grid definitions to support the computation of CBS scores (#3232).
- Refine default lapse rates for orographic corrections (#3258).
- Refine handling of missing data for orographic corrections (#3270).
- Enhance the MET tools to return consistent exit codes (#3278).

Bugfixes

- Bugfix: Fix compilation on non-GCC compilers by replacing #include <bits/stdc++.h> (#3175).
- Bugfix: Fix the AW_MEAN regridding method when OMP_NUM_THREADS is set very large (#3206).
- Bugfix: Fix STAT-Analysis handling of zero-length vectors when aggregating MPR inputs and writing WDIR or VCNT outputs (#3210).
- Bugfix: Python embedding does not work for MET version 12.1.0 with Python version 3.10 (#3219).
- Bugfix: Fix segfault when reading climatological data from a CF-compliant NetCDF file fails (#3235).
- Update the logic to apply "set_attr_grid" before "ShiftRight" (#3255).
- Fix ASCII2NC hang when run with an empty input file (#3266).

Repository, build, and test

- Confirm that the METplus difference logic handles MTD output file (#977).
- Switch MET's R-based differencing logic over to using the Python-based METplus implementation (#2718).

METbaseimage testing environment

- Add logic to regenerate previously tagged versions based on a schedule (METbaseimage#18).
- Add a new GHA development workflow and enhance workflows to scan Docker images for CVEs #33 (METbaseimage#33).
- METbaseimage: Address Critical CVEs from Grype scans of the Docker images (METplus-Internal#61).
- Enhance the "Create Release Docker Images" METplus workflows to determine and build the most recent bugfix versions (METplus-Internal#64).

2.2 MET Upgrade Instructions

This section summarizes and highlights important changes to MET since version 12.1.0, including:

- Adding new or modifying existing **configuration file options**.
- Modifying existing **output file formats**, such as new or modified ASCII or NetCDF file formats.
- Changing existing **output data values** generated by MET, typically due to fixing bugs that were computing incorrect output.
- Any other relevant details needed to upgrade from the MET version 12.1.0 to 12.2.0.

2.2.1 MET Version 12.2.0 Upgrade Instructions

Configuration file changes

MET version 12.2.0 adds, modifies, or removes the following configuration options:

- Ensemble-Stat configuration:
  - Adds support for **land_mask** and **topo_mask** configuration dictionaries previously only supported in Point-Stat.
- Ensemble-Stat and Point-Stat configurations:
  - Adds new **lapse_rate_correction** configuration dictionary to use topography differences to correct forecast or observation values of surface temperature.
  - Adds new **msl_agl_correction** configuration dictionary to convert between height values defined relative to mean-sea-level or above-ground-level.

Output format changes - NONE

MET version 12.2.0 adds or modifies the following output file formats:

- None

Output data changes - NONE

MET version 12.2.0 modifies existing output data values in the following ways:

- None

Additional upgrade instructions - NONE

Recommendations when upgrading to MET version 12.2.0:

- None

Chapter 3

Installation

3.1 Introduction

This section is meant to provide guidance on installing MET. It assumes a beginner user to MET is compiling MET from scratch and will step through the installation process, including listing dependent libraries and how to install them. Installation will require an understanding of the environment and hardware that MET is being installed on as it has options that are dependent on compiler usage, modulefiles, etc.

There are multiple supported methods for installing MET: using a provided compilation script, Docker instances, and Apptainer instances. All of these methods will be described below. The recommended method is using the provided *Using the compile_MET_all.sh script* (page 18).

Some organizations have access to precompiled versions of MET on shared systems, making the need for self-installation unnecessary. Users should check the METplus Downloads page for the latest coordinated release's Existing Builds page for existing MET installations before continuing.

3.2 Required External Libraries

MET is dependent on several external libraries to function properly. The required libraries are listed below:

- BUFRLIB for reading PrepBufr Observation files
- NetCDF4 in C and CXX, for intermediate and output file formats
- HDF5 is required to support NetCDF 4. HDF5 should be built with zlib
- GSL GNU Scientific Library Developer's Version for computing confidence intervals (use **GSL-1.11** for **PGI** compilers)
- Proj Library used to instantiate grids within MET

The following libraries are conditionally required, depending on the intended verification use and compiler language:

- GRIB2C Library (with dependencies Z, PNG, JASPER), if compiling GRIB2 support
- Python Libraries, if compiling support for Python embedding

- ecKit Library, if compiling support for unstructured grids
- ATLAS Library, if compiling support for unstructured grids
- HDF4 library if compiling the MODIS-Regrid or lidar2nc tool
- HDF-EOS2 library if compiling the MODIS-Regrid or lidar2nc tool
- Cairo library if compiling the MODE-Graphics tool
- FreeType library if compiling the MODE-Graphics tool
- f2c library for interfacing between Fortran and C (**Not required for most compilers**)

Users can take advantage of the compilation script to download and install all of the libraries automatically, both required and conditionally required *Using the compile_MET_all.sh script* (page 18).

3.3 Suggested External Utilities

The following utilities have been used with success by other METplus users in their verification processes. They are not required for MET to function, but depending on the user's intended verification needs, they may be of use:

- copygb utility for re-gridding GRIB version 1 data
- wgrib2 utility for re-gridding GRIB version 2 data
- Integrated Data Viewer (IDV) for displaying gridded data, including GRIB and NetCDF
- ncview utility for viewing gridded NetCDF data (e.g. the output of pcp_combine)

3.4 Using the compile_MET_all.sh script

The **compile_MET_all.sh** script is designed to install MET and, if desired, all of the external library dependencies required for running the system on various platforms. There are some required environment variables that need to be set before running the script and some optional environment variables, both of which are described below. The script relies on a **tar_files.tgz** file, which contains all of the required and optional library packages for MET but not MET itself. A separate command will be used to pull down the latest version of MET in **tar_files.tgz** format from GitHub, which the script will then install.

To begin, create and change to a directory where the latest version of MET will be installed. Assuming that the following guidance uses "/d1" as the parent directory, a suggested format is a path to a "met" directory, followed by the version number subdirectory (e.g. /d1/met/12.2.0). Next, download the compile_MET_all.sh script and tar_files.tgz file and place both of these files in the new directory. These files are available either through using the hyperlinks provided or by entering the following commands in the terminal while in the directory MET will be installed in:

```
wget https://raw.githubusercontent.com/dtcenter/MET/main_v12.2/internal/scripts/installation/
↪compile_MET_all.sh
wget https://dtcenter.ucar.edu/dfiles/code/METplus/MET/installation/tar_files.tgz
```

The tar files will need to be extracted in the MET installation directory:

```
tar -zxf tar_files.tgz
```

To make the compilation script into an executable, change the permissions to the following:

```
chmod 775 compile_MET_all.sh
```

Now change directories to the one that was created from expanding the tar files:

```
cd tar_files
```

The next step will be to identify and download the latest MET release as a tar file (e.g. v12.2.0.tar.gz) and place it in the *tar_files* directory. The file is available from the MET line under the "RECOMMENDED - COMPONENTS" section on the METplus website or by using a wget command while in the *tar_files* directory:

```
wget https://github.com/dtcenter/MET/archive/refs/tags/v12.2.0.tar.gz
```

3.4.1 Environment Variables to Run Script

Before running the compilation script, there are five environment variables that are required: **TEST_BASE**, **COMPILER**, **MET_SUBDIR**, **MET_TARBALL**, and **USE_MODULES**.

If compiling support for Python embedding, the script will need the following additional environment variables: **MET_PYTHON**, **MET_PYTHON_CC**, and **MET_PYTHON_LD**.

All of these environment variables are discussed in further detail in the Environment Variable Descriptions section below. An easy way to set these environment variables is in an environment configuration file (for example, **install_met_env.<machine_name>**). An example environment configuration file to start with (**install_met_env.generic_gnu**), as well as the environment configuration files used on HPCs at NCAR and NOAA, can be found in the MET GitHub repository in the scripts/installation/config directory.

Note: Starting with MET-12.0.0, C++17 is the default C++ standard for MET due to the requirements of its dependent libraries. However, MET itself only makes use of C++11 features.

The ATLAS library (conditionally required for MET, if support for unstructured grids is desired) versions 0.33.0 and later requires compiler support for the C++17 standard.

At this time, users with systems that do not yet support the C++17 standard, can still compile MET with an older C++ standard, using an older version of ATLAS, by adding the MET_CXX_STANDARD variable to the environment configuration file as described in the **OPTIONAL** section below.

3.4.2 Environment Variable Descriptions

REQUIRED

TEST_BASE – Format is */d1/met/12.2.0*. This is the MET installation directory that was created the beginning of, Section 3.4 and contains the **compile_MET_all.sh** script, **tar_files.tgz**, and the *tar_files* directory from the untar command.

COMPILER – Format is *compiler_version* (e.g. gnu_8.3.0). For the GNU family of compilers, use "gnu"; for the Intel family of compilers, use "intel", "intel-classic", "intel-oneapi", "ics", "ips", or "PrgEnv-intel", depending on the system. If using an Intel compiler, users that have also set the **USE_MODULES** environment variable to TRUE should review the additional information below for proper configuration file setup. In the past, support was provided for the PGI family of compilers through "pgi". However, this compiler option is no longer actively tested.

MET_SUBDIR – Format is */d1/met/12.2.0*. This is the location where the top-level MET subdirectory will be installed and is often set equivalent to **TEST_BASE** (e.g. ${TEST_BASE}).

MET_TARBALL – Format is *v12.2.0.tar.gz*. This is the name of the downloaded MET tarball.

USE_MODULES – Format is *TRUE* or *FALSE*. Set to FALSE if using a machine that does not use modulefiles; set to TRUE if using a machine that does use modulefiles. For more information on modulefiles, visit the Wikipedia page. If the **USE_MODULES** setting is set to true and the compiler is an Intel compiler, please review the additional information below for proper configuration file setup.

PYTHON_MODULE - Format is *PythonModuleName_version* (e.g. python_3.10.4). This environment variable is only required if **USE_MODULES** = TRUE. To set properly, list the Python module to load followed by an underscore and version number. For example, setting **PYTHON_MODULE** =python_3.10.4 will cause the script to run "module load python/3.10.4".

ADDITIONAL SETTINGS FOR INTEL COMPILER USERS WITH THE USE_MODULES SETTING

It is necessary for the user to specify (in the install_met_env.<machine> config file) the following environment variables if using the Intel compilers:

For non-oneAPI Intel compilers:

export FC=ifort
export F77=ifort
export F90=ifort
export CC=icc
export CXX=icpc

For oneAPI Intel compilers:

export FC=ifx

export F77=ifx
export F90=ifx
export CC=icx
export CXX=icpx

This is due to the machines allowing users to load a module but not setting these environment variables as expected, leading to failed installations. For user convenience, additional generic configuration files have been created that include these settings. Users with a classic Intel compiler are encouraged to use the install_met_env.generic_intel_classic configuration file, and users with a oneAPI Intel compiler should use the install_met_env.generic_intel_oneapi configuration file.

REQUIRED, IF COMPILING PYTHON EMBEDDING

MET_PYTHON – Format is */usr/local/python3*. This is the location containing the bin, include, lib, and share directories for Python.

MET_PYTHON_CC - Format is -I followed by the directory containing the Python include files (e.g. -I/usr/local/python3/include/python3.10). This information may be obtained by running `python3-config --cflags`; however, this command can, on certain systems, provide too much information.

MET_PYTHON_LD - Format is -L followed by the directory containing the Python library files then a space, then -l followed by the necessary Python libraries to link to (e.g. -L/usr/local/python3/lib/\ -lpython3.10\ -lpthread\ -ldl\ -lutil\ -lm). The backslashes are necessary in the example shown because of the spaces, which will be recognized as the end of the value unless preceded by the "\" character. Alternatively, a user can provide the value in quotations (e.g. export MET_PYTHON_LD="-L/usr/local/python3/lib/ -lpython3.10 -lpthread -ldl -lutil -lm"). This information may be obtained by running `python3-config --ldflags --embed`; however, this command can, on certain systems, provide too much information.

OPTIONAL

export MAKE_ARGS="-j #" – If there is a need to install external libraries, or to attempt to speed up the MET compilation process, this environmental setting can be added to the environment configuration file. Replace the # with the number of cores to use (as an integer) or simply specify "export MAKE_ARGS=-j" with no integer argument to start as many processes in parallel as possible. Note that Docker has trouble compiling without a specified value of cores to use. The automated MET testing scripts in the Docker environment have been successful with a value of 5 (e.g. export MAKE_ARGS="-j 5").

export MET_CXX_STANDARD - Specify the version of the supported C++ standard. Values may be 11, 14, or 17. The default value is 17. (e.g. export MET_CXX_STANDARD=11)

3.4.3 External Library Handling in compile_MET_all.sh

IF THE USER WANTS TO HAVE THE COMPILATION SCRIPT COMPILE THE LIBRARY DEPENDENCIES

The **compile_MET_all.sh** script will compile and install MET and its *Required External Libraries* (page 17), if needed. Note that if these libraries are already installed somewhere on the system, MET will call and use the libraries that were installed by the script.

IF THE USER ALREADY HAS THE LIBRARY DEPENDENCIES INSTALLED

If the required external library dependencies have already been installed and don't need to be reinstalled, or if compiling MET on a machine that uses modulefiles and the user would like to make use of the existing dependent libraries on that machine, there are more environment variables that need to be set to let MET know where those library and header files are. The following environment variables need to be added to the environment configuration file:

Feature	Configuration Option	Environment Variables
Always Required		MET_BUFRLIB, BUFRLIB_NAME, MET_PROJ, MET_HDF5, MET_NETCDF, MET_GSL
Optional GRIB2 Support	`--enable-all` or `--enable-grib2`	MET_GRIB2CLIB, MET_GRIB2CINC, GRIB2CLIB_NAME, LIB_JASPER, LIB_PNG, LIB_Z
Optional Python Support	`--enable-all` or `--enable-python`	MET_PYTHON_BIN_EXE, MET_PYTHON_CC, MET_PYTHON_LD
Optional Unstructured Grid Support	`--enable-all` or `--enable-ugrid`	MET_ATLAS, MET_ECKIT
Optional LIDAR2NC Support	`--enable-all` or `--enable-lidar2nc`	MET_HDF
Optional MODIS Support	`--enable-all` or `--enable-modis`	MET_HDF, MET_HDFEOS
Optional MODE Graphics Support	`--enable-all` or `--enable-mode_graphics`	MET_CAIRO, MET_FREETYPE
Optional Profiler Support	`--enable-profiler`	

Generally speaking, for each library there is a set of three environment variables that can describe the locations: **$MET_<lib>**, **$MET_<lib>INC** and **$MET_<lib>LIB**.

The $MET_<lib> environment variable can be used if the external library is installed such that there is a main directory which has a subdirectory called *lib* containing the library files and another subdirectory called *include* containing the include files.

Alternatively, the $MET_<lib>INC and $MET_<lib>LIB environment variables are used if the library and include files for an external library are installed in separate locations. In this case, both environment variables must be specified and the associated $MET_<lib> variable will be ignored.

FINAL NOTE ON EXTERNAL LIBRARIES

For users wishing to run the Plot-MODE-Field tool, the Ghostscript font data must be downloaded and the **MET_FONT_DIR** environment variable in the **install_met_env.<machine_name>** file should point to the directory containing those fonts.

3.4.4 Executing the compile_MET_all.sh script

With the proper files downloaded and the environment configuration file set to the particular system's needs, MET is ready for installation. The screenshot below shows the contents of the installation directory followed by the tar_files subdirectory at this step on the machine 'hera'.

```
/contrib/met/12.2.0$ ls
compile_MET_all.sh  install_met_env.hera  tar_files

/contrib/met/12.2.0$ ls tar_files
HDF-EOS2.16v1.00.tar.Z        eckit-1.24.4.tar.gz           netcdf-4.7.4.tar.gz
HDF4.2r3.tar.gz               freetype-2.11.0.tar.gz        netcdf-cxx4-4.3.1.tar.gz
atlas-0.30.0.tar.gz           g2clib-1.6.4.tar.gz           pixman-0.40.0.tar.gz
atlas-0.35.0.tar.gz           gsl-1.11.tar.gz               proj-7.1.0.tar.gz
bufr_v11.6.0.tar.gz           gsl-2.7.1.tar.gz              sqlite-autoconf-3430100.tar.gz
cairo-1.16.0.tar.xz           hdf5-1.12.2.tar.gz            tiff-4.6.0.tar.gz
ecbuild-3.5.0.tar.gz          jasper-2.0.25.tar.gz          zlib-1.2.11.tar.gz
ecbuild-3.7.0.tar.gz          jpegsrc.v9e.tar.gz
eckit-1.20.2.tar.gz           libpng-1.6.37.tar.gz
```

Simply enter the following into the terminal to execute the script:

```
./compile_MET_all.sh install_met_env.<machine_name>
```

The screenshot below shows the contents of the installation directory after installation:

```
/contrib/met/12.2.0$ ls
MET-12.2.0  bin  compile_MET_all.sh  external_libs  install_met_env.hera  share  tar_files
```

To confirm that MET was installed successfully, run the following command from the installation directory to check for errors in the test file:

```
grep -i error MET-12.2.0/met.make_test.log
```

If no errors are returned, the installation was successful. Due to the highly variable nature of hardware systems, users may encounter issues during the installation process that result in MET not being installed. If this occurs please first recheck that the location of all the necessary data files and scripts is correct. Next, recheck the environment variables in the environment configuration file and ensure there are no spelling errors or improperly set variables. After these checks are complete, run the script again.

If there are still errors, users still have options to obtain a successful MET installation. Check the FAQ section of the User's Guide on topics relevant to installation. Next, review previously asked questions on the

installation topic in GitHub Discussions. Users are welcome to post any questions they might have that have not been asked. Finally, consider one of the remaining installation methods for MET, as these may prove more successful.

3.5 Using Docker for Running MET

Docker is a system that seeks to eliminate some of the complexities associated with downloading various software and any library dependencies it might have by allowing users to run inside a preset container. Instead of using a hard copy of an application, Docker allows users to pull images of the application and run those within the Docker environment. This is beneficial to both developers (who no longer have to design with every possible system environment in mind) and users (who can skip tracking down system environment settings and meet with success faster) alike.

MET has numerous version images for Docker users and continues to be released as images at the same interval as system releases. While the advantages of Docker can make it an appealing installation route for first time users, it does require privileged user access that will result in an unsuccessful installation if not available. Please ensure the user has high system access (e.g. admin access) before attempting this method.

3.5.1 Installing Docker

To begin, download and install the correct version of Docker for the intended system. The Docker installation webpage should detect what system is accessing the webpage and auto select the appropriate version. If a different version is required, select the correct version from the dropdown option. Follow Docker's instructions for a successful installation.

3.5.2 Loading the Latest Docker Image of MET

Once the installation of Docker has been confirmed to be successful, all that's needed to run MET is to download the latest image of MET in Docker. To accomplish that, use the pull command, with the latest MET version number, for example:

```
docker pull dtcenter/met:12.2.0
```

Omitting the version number will result in an error due to Docker's behavior of attempting to retrieve an image with the "latest" tag, which MET no longer uses.

3.5.3 Running the Docker version of MET

All that is left to do is launch a shell in the Docker container. This is accomplished with the command:

```
docker run -it --rm dtcenter/met /bin/bash
```

Note that the "–rm" command was added to automatically remove the container created from the image once exiting Docker. Simply remove this command if the container should persist after exiting. If there is an

error during this run command, try adding the latest MET version number the same way the latest image of MET was pulled:

```
docker run -it --rm dtcenter/met:12.2.0 /bin/bash
```

If the usage MET via Docker images was successful, it is highly recommended to move on to using the METplus wrappers of the tools, which have their own Docker image. Instructions for obtaining that image are in the METplus Wrappers User's Guide.

3.6 Using Apptainer for Running MET

Similar to Docker, Apptainer (formerly Singularity) removes some of the complexities associated with downloading various library dependencies and runs inside a preset container. Apptainer is incredibly flexible and was designed to function on High Performance Computing (HPC) systems. It can utilize Container Library and Docker images, meaning users can benefit from the Docker images that already exist for MET.

Perhaps the biggest benefit of using Apptainer (aside from its agnostic platform availability) is its nonrequirement of root permissions. This can be one of the only ways users operating on large-scale, shared computing resources can access MET. That, plus the relatively simple installation of Apptainer and retrieval of Docker images, should help any users experiencing difficulties with MET installation using previous methods achieve success.

3.6.1 Installing Apptainer

To begin, download and install the correct version of Apptainer for the intended system. The method of installing from code is outlined in Apptainer's INSTALL.md file on their GitHub page. If users require an alternate method to install Apptainer, the Admin guide will provide further details.

3.6.2 Loading the Latest MET Image

Similar to Docker, Apptainer will build the container based off of the MET image in a single command. To accomplish this, Apptainer's "Swiss army knife" `build` command is used. Use the the latest MET version number in conjunction with `build` to make the container:

```
singularity build met-12.2.0.sif docker://dtcenter/met:12.2.0
```

3.6.3 Running the MET Container

The container is now ready for usage! Simply use the `exec` command to invoke the MET container, along with the appropriate MET command line usage:

```
singularity exec met-12.2.0.sif plot_data_plane /home/data/fcst_006.grb2 image_output.ps␣
↪'name="TMP"; level="Z0";'
```

3.6.4 Stopping the Apptainer Instance

Once work is complete within the instance, the `stop` command can be used to end the instance. This command will need to be used otherwise the instance will continue to run in the background:

```
singularity instance stop /path/to/container/met-12.2.0.sif met-12.2.0
```

Now that MET is successfully installed, it is highly recommended to next install the METplus wrappers to take full advantage of Python integration. Users can also proceed to the Tutorial and run through the examples that only utilize the MET processes (METplus wrapper applications and commands will not work unless METplus wrappers are also installed).

Chapter 4

MET Data I/O

Data must often be preprocessed prior to using it for verification. Several MET tools exist for this purpose. In addition to preprocessing observations, some plotting utilities for data checking are also provided and described at the end of this section. Both the input and output file formats are described in this section. Section 4.1 and Section 4.2 are primarily concerned with re-formatting input files into the intermediate files required by some MET modules. These steps are represented by the first three columns in the MET flowchart depicted in Figure 1.1. Output data formats are described in Section 4.3. Common configuration files options are described in Section 4.5. Description of software modules used to reformat the data may now be found in Section 7 and Section 8.

4.1 Input Data Formats

The MET package can handle multiple gridded input data formats: GRIB version 1, GRIB version 2, and NetCDF files following the Climate and Forecast (CF) conventions, containing raw WRF output, containing WRF output post-processed using wrf_interp, or produced by the MET tools themselves. MET supports standard NCEP, USAF, UKMet Office and ECMWF GRIB tables along with custom, user-defined GRIB tables and the extended PDS including ensemble member metadata. See Section 4.5 for more information. Point observation files may be supplied in either PrepBUFR, ASCII, or MADIS format. Note that MET does not require the Unified Post-Processor to be used, but does require that the input GRIB data be on a standard, de-staggered grid on pressure or regular levels in the vertical. While the Grid-Stat, Wavelet-Stat, MODE, and MTD tools can be run on a gridded field at virtually any level, the Point-Stat tool can only be used to verify forecasts at the surface or on pressure or height levels. MET does not interpolate between native model vertical levels.

When comparing two gridded fields with the Grid-Stat, Wavelet-Stat, Ensemble-Stat, MODE, MTD, or Series-Analysis tools, the input model and observation datasets must be on the same grid. MET will regrid files according to user specified options. Alternatively, outside of MET, the copygb and wgrib2 utilities are recommended for re-gridding GRIB1 and GRIB2 files, respectively. To preserve characteristics of the observations, it is generally preferred to re-grid the model data to the observation grid, rather than vice versa.

Input point observation files in PrepBUFR format are available through NCEP. The PrepBUFR observation files contain a wide variety of point-based observation types in a single file in a standard format. However, some users may wish to use observations not included in the standard PrepBUFR files. For this reason, prior to performing the verification step in the Point-Stat tool, the PrepBUFR file is reformatted with the PB2NC

tool. In this step, the user can select various ways of stratifying the observation data spatially, temporally, and by type. The remaining observations are reformatted into an intermediate NetCDF file. The ASCII2NC tool may be used to convert ASCII point observations that are not available in the PrepBUFR files into this common NetCDF point observation format. Several other MET tools, described below, are also provided to reformat point observations into this common NetCDF point observation format prior to passing them as input to the Point-Stat or Ensemble-Stat verification tools.

Tropical cyclone forecasts and observations are typically provided in a specific ATCF (Automated Tropical Cyclone Forecasting) ASCII format, in A-deck, B-deck, and E-deck files.

4.1.1 ASCII File Lists

Several MET tools read multiple gridded input data files in a single run. For example, multiple gridded input files can define a time series of data for Series-Analysis and MTD or multiple ensemble members for Gen-Ens-Prod and Ensemble-Stat. Generally, users have two options for defining that list of input files:

1. Directly on the command line, for a relatively small number of files.
2. In a single ASCII file containing a list of file names of arbitrary length.

ASCII file lists consist of a white-space separated list of paths to the input files. While relative paths may work, users are encouraged to provide full paths for more consistent behavior. Note that environment variables can also be used in the file lists.

The file list elements can be separated by spaces, tabs, or newlines, but not commas. Users are encouraged to add the optional **file_list** keyword as the first element of each list to clearly identify it as such. When **file_list** is the first item, no validation of the file existence is performed. When **file_list** is not present, MET checks whether each input file actually exists on the file system and errors out if it encounters too many missing input files.

While validating file existence is useful for standard input file formats, it can cause problems when providing a list of arguments for a Python embedding script since a **file_list** used for Python embedding can contain optional command line arguments in addition to file names on each line. Please see the description of *MET_PYTHON_INPUT_ARG* (page 597) for additional details.

Here is an example ASCII file list for three input files, each listed on a separate line:

```
file_list
/path/to/file1
/path/to/file2
/path/to/file3
```

4.1.2 Requirements for CF Compliant NetCDF

The MET tools use following attributes and variables for input "CF Compliant NetCDF data".

1. The global attribute "Conventions".
2. The "standard_name" and "units" attributes for coordinate variables. The "axis" attribute ("T" or "time") must exist as the time variable if the "standard_name" attribute does not exist.
3. The "coordinates" attribute for the data variables. It contains the coordinate variable names.
4. The "grid_mapping" attribute for the data variables for projections and the matching grid mapping variable (optional for latitude_longitude and rotated_latitude_longitude projections).
5. The gridded data should be evenly spaced horizontally and vertically.
6. (Optional) the "forecast_reference_time" variable for init_time.
7. (Optional) the "vertical coordinate" variable must include a supported units attribute to allow range selection.

MET processes the CF-Compliant gridded NetCDF files with the projection information. The CF-Compliant NetCDF is defined by the global attribute "Conventions" whose value begins with "CF-" ("CF-<Version_number>"). The global attribute "Conventions" is mandatory. MET accepts the variation of this attribute ("conventions" and "CONVENTIONS"). The value should be started with "CF-" and followed by the version number. MET accepts the attribute value that begins with "CF " ("CF" and a space instead of a hyphen) or "COARDS".

The grid mapping variable contains the projection information. The grid mapping variable can be found by looking at the variable attribute "grid_mapping" from the data variables. The "standard_name" attribute is used to filter out the coordinate variables like time, latitude, and longitude variables. The value of the "grid_mapping" attribute is the name of the grid mapping variable. Four projections are supported with grid mapping variables: latitude_longitude, lambert_conformal_conic, polar_stereographic, and geostationary. In case of the latitude_longitude projection, the latitude and longitude variable names should be the same as the dimension names and the "units" attribute should be valid.

Here are examples for the grid mapping variable ("edr" is the data variable):

Example 1: grid mapping for latitude_longitude projection

```
float edr(time, z, lat, lon) ;
        edr:units = "m^(2/3) s^-1" ;
        edr:long_name = "Median eddy dissipation rate" ;
        edr:coordinates = "lat lon" ;
        edr:_FillValue = -9999.f ;
        edr:grid_mapping = "grid_mapping" ;
int grid_mapping ;
        grid_mapping:grid_mapping_name = "latitude_longitude" ;
        grid_mapping:semi_major_axis = 6371000. ;
        grid_mapping:inverse_flattening = 0 ;
```

Example 2: grid mapping for lambert_conformal_conic projection

```
float edr(time, z, y, x) ;
        edr:units = "m^(2/3) s^-1" ;
        edr:long_name = "Eddy dissipation rate" ;
        edr:coordinates = "lat lon" ;
        edr:_FillValue = -9999.f ;
        edr:grid_mapping = "grid_mapping" ;
int grid_mapping ;
        grid_mapping:grid_mapping_name = "lambert_conformal_conic" ;
        grid_mapping:standard_parallel = 25. ;
        grid_mapping:longitude_of_central_meridian = -95. ;
        grid_mapping:latitude_of_projection_origin = 25. ;
        grid_mapping:false_easting = 0 ;
        grid_mapping:false_northing = 0 ;
        grid_mapping:GRIB_earth_shape = "spherical" ;
        grid_mapping:GRIB_earth_shape_code = 0 ;
```

When the grid mapping variable is not available, MET can detect either a latitude_longitude or rotated_latitude_longitude projection. It detects the latitude_longitude projection in the following order:

1. the lat/lon projection from the dimensions
2. the lat/lon projection from the "coordinates" attribute from the data variable
3. the lat/lon projection from the latitude and longitude variables by the "standard_name" attribute

MET is looking for variables with the same name as the dimension and checking the "units" attribute to find the latitude and longitude variables. The valid "units" strings are listed in the table below. MET accepts the variable "tlat" and "tlon" if the dimension names are "nlat" and "nlon".

If there are no latitude and longitude variables from dimensions, MET gets coordinate variable names from the "coordinates" attribute. The matching coordinate variables should have the proper "units" attribute.

MET gets the time, latitude, and longitude variables by looking at the standard name: "time", "latitude", and "longitude" as the last option.

MET gets the valid time from the time variable and the "forecast_reference_time" variable for the init_time. If the time variable does not exist, it can come from the file name. MET supports only two cases:

1. TRMM_3B42_3hourly_filename (3B42.<yyyymmdd>.<hh>.7.G3.nc)
2. TRMM_3B42_daily_filename (3B42_daily.<yyyy>.<mm>.<dd>.7.G3.nc)

For rotated_latitude_longitude projections, MET detects the projection using the following logic:

1. Looking for a variable that has the same name as a dimension.
2. If a variable is found, checking to make sure the variable has a standard name attribute and that that attribute is not empty.
3. Checking to see if the standard name attribute is called grid_latitude for latitude variables and grid_longitude for the longitude variable.

The latitude and longitude variables must be one dimensional and with their size matching the corresponding dimension for latitude_longitude and rotated_latitude_longitude grids.

Table 4.1: Valid strings for the "units" attribute.

time	latitude	longitude
"seconds since YYYY-MM-DD HH:MM:SS", "minutes since YYYY-MM-DD HH:MM:SS", "hours since YYYY-MM-DD HH:MM:SS", "days since YYYY-MM-DD HH:MM:SS", "months since YYYY-MM-DD HH:MM:SS", "years since YYYY-MM-DD HH:MM:SS", Accepts "Y", "YY", "YYY", "M", "D", "HH", and "HH:MM". "HH:MM:SS" is optional	"degrees_north", "degree_north", "degree_N", "degrees_N", "degreeN", "degreesN"	"degrees_east", "degree_east", "degree_E", "degrees_E", "degreeE", "degreesE"

4.1.3 Performance with NetCDF Input Data

There is no limitation on the NetCDF file size. The size of the data variables matters more than the file size. The NetCDF API loads the metadata first upon opening the NetCDF file. It's similar for accessing data variables. There are two API calls: getting the metadata and getting the actual data. The memory is allocated and consumed at the second API call (getting the actual data).

The dimensions of the data variables matter. MET requests the NetCDF data needs based on: 1) loading and processing a data plane, and 2) loading and processing the next data plane. This means an extra step for slicing with one more dimension in the NetCDF input data. The performance is quite different if the compression is enabled with high resolution data. NetCDF does compression per variable. The variables can have different compression levels (0 to 9). A value of 0 means no compression, and 9 is the highest level of compression possible. The number for decompression is the same between one more and one less dimension NetCDF input files (combined VS separated). The difference is the amount of data to be decompressed which requires more memory. For example, let's assume the time dimension is 30. NetCDF data with one less dimension (no time dimension) does decompression 30 times for nx by ny dataset. NetCDF with one more dimension does compression 30 times for 30 by nx by ny dataset and slicing for target time offset. So it's better to have multiple NetCDF files with one less dimension than a big file with bigger variable data if compressed. If the compression is not enabled, the file size will be much bigger requiring more disk space.

4.2 Intermediate Data Formats

MET uses NetCDF as an intermediate file format. The MET tools which write gridded output files write to a common gridded NetCDF file format. The MET tools which write point output files write to a common point observation NetCDF file format.

4.3 Output Data Formats

The MET package currently produces output in the following basic file formats: STAT files, ASCII files, NetCDF files, PostScript plots, and png plots from the Plot-Mode-Field utility.

The STAT format consists of tabular ASCII data that can be easily read by many analysis tools and software packages. MET produces STAT output for the Grid-Stat, Point-Stat, Ensemble-Stat, Wavelet-Stat, and TC-Gen tools. STAT is a specialized ASCII format containing one record on each line. However, a single STAT file will typically contain multiple line types. Several header columns at the beginning of each line remain the same for each line type. However, the remaining columns after the header change for each line type. STAT files can be difficult for a human to read as the quantities represented for many columns of data change from line to line.

For this reason, ASCII output is also available as an alternative for these tools. The ASCII files contain exactly the same output as the STAT files but each STAT line type is grouped into a single ASCII file with a column header row making the output more human-readable. The configuration files control which line types are output and whether or not the optional ASCII files are generated.

The MODE tool creates two ASCII output files as well (although they are not in a STAT format). It generates an ASCII file containing contingency table counts and statistics comparing the model and observation fields being compared. The MODE tool also generates a second ASCII file containing all of the attributes for the single objects and pairs of objects. Each line in this file contains the same number of columns, and those columns not applicable to a given line type contain fill data. Similarly, the MTD tool writes one ASCII output file for 2D objects attributes and four ASCII output files for 3D object attributes.

The TC-Pairs and TC-Stat utilities produce ASCII output, similar in style to the STAT files, but with TC relevant fields.

Many of the tools generate gridded NetCDF output. Generally, this output acts as input to other MET tools or plotting programs. The point observation preprocessing tools produce NetCDF output as input to the statistics tools. Full details of the contents of the NetCDF files is found in Section 4.4 below.

The MODE, Wavelet-Stat and plotting tools produce PostScript plots summarizing the spatial approach used in the verification. The PostScript plots are generated using internal libraries and do not depend on an external plotting package. The MODE plots contain several summary pages at the beginning, but the total number of pages will depend on the merging options chosen. Additional pages will be created if merging is performed using the double thresholding or fuzzy engine merging techniques for the forecast and observation fields. The number of pages in the Wavelet-Stat plots depend on the number of masking tiles used and the dimension of those tiles. The first summary page is followed by plots for the wavelet decomposition of the forecast and observation fields. The generation of these PostScript output files can be disabled using command line options.

Users can use the optional plotting utilities Plot-Data-Plane, Plot-Point-Obs, and Plot-Mode-Field to produce graphics showing forecast, observation, and MODE object files.

4.4 Data Format Summary

The following is a summary of the input and output formats for each of the tools currently in MET. The output listed is the maximum number of possible output files. Generally, the type of output files generated can be controlled by the configuration files and/or the command line options:

1. **PB2NC Tool**
    - **Input**: PrepBUFR point observation file(s) and one configuration file.
    - **Output**: One NetCDF file containing the observations that have been retained.
2. **ASCII2NC Tool**
    - **Input**: ASCII point observation file(s) that has (have) been formatted as expected, and optional configuration file.
    - **Output**: One NetCDF file containing the reformatted observations.
3. **MADIS2NC Tool**
    - **Input**: MADIS point observation file(s) in NetCDF format.
    - **Output**: One NetCDF file containing the reformatted observations.
4. **LIDAR2NC Tool**
    - **Input**: One CALIPSO satellite HDF file.
    - **Output**: One NetCDF file containing the reformatted observations.
5. **IODA2NC Tool**
    - **Input**: IODA observation file(s) in NetCDF format.
    - **Output**: One NetCDF file containing the reformatted observations.
6. **Point2Grid Tool**
    - **Input**: One NetCDF file in the common point observation format.
    - **Output**: One NetCDF file containing a gridded representation of the point observations.
7. **Pcp-Combine Tool**
    - **Input**: Two or more gridded model or observation files (in GRIB format for "sum" command, or any gridded file for "add", "subtract", and "derive" commands) containing data (often accumulated precipitation) to be combined.
    - **Output**: One NetCDF file containing output for the requested operation(s).
8. **Regrid-Data-Plane Tool**
    - **Input**: One gridded model or observation field and one gridded field to provide grid specification if desired.
    - **Output**: One NetCDF file containing the regridded data field(s).
9. **Shift-Data-Plane Tool**

- **Input**: One gridded model or observation field.
- **Output**: One NetCDF file containing the shifted data field.

10. **MODIS-Regrid Tool**
    - **Input**: One gridded model or observation field and one gridded field to provide grid specification.
    - **Output**: One NetCDF file containing the regridded data field.
11. **Gen-VX-Mask Tool**
    - **Input**: One gridded model or observation file and one file defining the masking region (varies based on masking type).
    - **Output**: One NetCDF file containing a bitmap for the resulting masking region.
12. **Point-Stat Tool**
    - **Input**: One gridded model file, at least one NetCDF file in the common point observation format, and one configuration file.
    - **Output**: One STAT file containing all of the requested line types and several ASCII files for each line type requested.
13. **Grid-Stat Tool**
    - **Input**: One gridded model file, one gridded observation file, and one configuration file.
    - **Output**: One STAT file containing all of the requested line types, several ASCII files for each line type requested, and one NetCDF file containing the matched pair data and difference field for each verification region and variable type/level being verified.
14. **Ensemble Stat Tool**
    - **Input**: An arbitrary number of gridded model files, one or more gridded and/or point observation files, and one configuration file. Point and gridded observations are both accepted.
    - **Output**: One NetCDF file containing requested ensemble forecast information. If observations are provided, one STAT file containing all requested line types, several ASCII files for each line type requested, and one NetCDF file containing gridded observation ranks.
15. **Wavelet-Stat Tool**
    - **Input**: One gridded model file, one gridded observation file, and one configuration file.
    - **Output**: One STAT file containing the "ISC" line type, one ASCII file containing intensity-scale information and statistics, one NetCDF file containing information about the wavelet decomposition of forecast and observed fields and their differences, and one PostScript file containing plots and summaries of the intensity-scale verification.
16. **GSID2MPR Tool**
    - **Input**: One or more binary GSI diagnostic files (conventional or radiance) to be reformatted.
    - **Output**: One ASCII file in matched pair (MPR) format.
17. **GSID2ORANK Tool**
    - **Input**: One or more binary GSI diagnostic files (conventional or radiance) to be reformatted.

- **Output**: One ASCII file in observation rank (ORANK) format.

18. **Stat-Analysis Tool**

    - **Input**: One or more STAT files output from the Point-Stat, Grid-Stat, Ensemble Stat, Wavelet-Stat, or TC-Gen tools and, optionally, one configuration file containing specifications for the analysis job(s) to be run on the STAT data.
    - **Output**: ASCII output of the analysis jobs is printed to the screen unless redirected to a file using the "-out" option or redirected to a STAT output file using the "-out_stat" option.

19. **Series-Analysis Tool**

    - **Input**: An arbitrary number of gridded model files and gridded observation files and one configuration file.
    - **Output**: One NetCDF file containing requested output statistics on the same grid as the input files.

20. **Grid-Diag Tool**

    - **Input**: An arbitrary number of gridded data files and one configuration file.
    - **Output**: One NetCDF file containing individual and joint histograms of the requested data.

21. **MODE Tool**

    - **Input**: One gridded model file, one gridded observation file, and one or two configuration files.
    - **Output**: One ASCII file containing contingency table counts and statistics, one ASCII file containing single and pair object attribute values, one NetCDF file containing object indices for the gridded simple and cluster object fields, and one PostScript plot containing a summary of the features-based verification performed.

22. **MODE-Analysis Tool**

    - **Input**: One or more MODE object statistics files from the MODE tool and, optionally, one configuration file containing specification for the analysis job(s) to be run on the object data.
    - **Output**: ASCII output of the analysis jobs will be printed to the screen unless redirected to a file using the "-out" option.

23. **MODE-TD Tool**

    - **Input**: Two or more gridded model files, two or more gridded observation files, and one configuration file.
    - **Output**: One ASCII file containing 2D object attributes, four ASCII files containing 3D object attributes, and one NetCDF file containing object indices for the gridded simple and cluster object fields.

24. **TC-DLand Tool**

    - **Input**: One or more files containing the longitude (Degrees East) and latitude (Degrees North) of all the coastlines and islands considered to be a significant landmass.
    - **Output**: One NetCDF format file containing a gridded field representing the distance to the nearest coastline or island, as specified in the input file.

25. **TC-Pairs Tool**

    - **Input**: At least one A-deck or E-deck file and one B-deck ATCF format file containing output from a tropical cyclone tracker and one configuration file. The A-deck files contain forecast tracks, the E-deck files contain forecast probabilities, and the B-deck files are typically the NHC Best Track Analysis but could also be any ATCF format reference.
    - **Output**: ASCII output with the suffix .tcst.

26. **TC-Stat Tool**

    - **Input**: One or more TCSTAT output files output from the TC-Pairs tool and, optionally, one configuration file containing specifications for the analysis job(s) to be run on the TCSTAT data.
    - **Output**: ASCII output of the analysis jobs will be printed to the screen unless redirected to a file using the "-out" option.

27. **TC-Gen Tool**

    - **Input**: One or more Tropical Cyclone genesis format files, one or more verifying operational and BEST track files in ATCF format, and one configuration file.
    - **Output**: One STAT file containing all of the requested line types, several ASCII files for each line type requested, and one gridded NetCDF file containing counts of track points.

28. **TC-RMW Tool**

    - **Input**: One or more gridded data files, one ATCF track file defining the storm location, and one configuration file.
    - **Output**: One gridded NetCDF file containing the requested model fields transformed into cylindrical coordinates.

29. **RMW-Analysis Tool**

    - **Input**: One or more NetCDF output files from the TC-RMW tool and one configuration file.
    - **Output**: One NetCDF file for results aggregated across the filtered set of input files.

30. **Plot-Point-Obs Tool**

    - **Input**: One NetCDF file containing point observation from the ASCII2NC, PB2NC, MADIS2NC, or LIDAR2NC tool.
    - **Output**: One postscript file containing a plot of the requested field.

31. **Plot-Data-Plane Tool**

    - **Input**: One gridded data file to be plotted.
    - **Output**: One postscript file containing a plot of the requested field.

32. **Plot-MODE-Field Tool**

    - **Input**: One or more MODE output files to be used for plotting and one configuration file.
    - **Output**: One PNG file with the requested MODE objects plotted. Options for objects include raw, simple or cluster and forecast or observed objects.

33. **GIS-Util Tools**

- **Input**: ESRI shape files ending in .dbf, .shp, or .shx.
- **Output**: ASCII description of their contents printed to the screen.

4.5 Configuration File Details

Part of the strength of MET is the leveraging of capability across tools. There are several configuration options that are common to many of the tools.

Many of the MET tools use a configuration file to set parameters. This prevents the command line from becoming too long and cumbersome and makes the output easier to duplicate.

The configuration file details are described in *Configuration File Overview* (page 41) and *Tropical Cyclone Configuration Options* (page 129).

Chapter 5

Configuration File Overview

The configuration files that control many of the MET tools contain formatted ASCII text. This format has been updated for MET version 12.2.2 and continues to be used in subsequent releases.

Settings common to multiple tools are described in the top part of this file and settings specific to individual tools are described beneath the common settings. Please refer to the MET User's Guide for more details about the settings if necessary.

A configuration file entry is an entry name, followed by an equal sign (=), followed by an entry value, and is terminated by a semicolon (;). The configuration file itself is one large dictionary consisting of entries, some of which are dictionaries themselves.

The configuration file language supports the following data types:

- Dictionary:
  - Grouping of one or more entries enclosed by curly braces {}.
- Array:
  - List of one or more entries enclosed by square braces [].
  - Array elements are separated by commas.
- String:
  - A character string enclosed by double quotation marks "".
- Integer:
  - A numeric integer value.
- Float:
  - A numeric float value.
- Boolean:
  - A boolean value (TRUE or FALSE).
- Threshold:
  - A threshold type (<, <=, ==, !-, >=, or >) followed by a numeric value.

  - The threshold type may also be specified using two letter abbreviations (lt, le, eq, ne, ge, gt).
  - Multiple thresholds may be combined by specifying the logic type of AND (&&) or OR (||). For example, ">=5&&<=10" defines the numbers between 5 and 10 and "==1||==2" defines numbers exactly equal to 1 or 2.

- Percentile Thresholds:
  - A threshold type (<, <=, ==, !=, >=, or >), followed by a percentile type description (SFP, SOP, SFCP, SOCP, USP, FCDP, OCDP, or FBIAS), followed by a numeric value, typically between 0 and 100.
  - Note that the two letter threshold type abbreviations (lt, le, eq, ne, ge, gt) are not supported for percentile thresholds.
  - Thresholds may be defined as percentiles of the data being processed in several places:
    * In Point-Stat and Grid-Stat when setting "cat_thresh", "wind_thresh" and "cnt_thresh".
    * In Wavelet-Stat when setting "cat_thresh".
    * In MODE when setting "conv_thresh" and "merge_thresh".
    * In Ensemble-Stat when setting "obs_thresh".
    * When using the "censor_thresh" config option.
    * In the Stat-Analysis "-out_fcst_thresh" and "-out_obs_thresh" job command options.
    * In the Gen-Vx-Mask "-thresh" command line option.
  - The following percentile threshold types are supported:
    * SFP for a percentile of the sample forecast values. e.g. ">SFP33.3" means greater than the 33.3-rd forecast percentile.
    * SOP for a percentile of the sample observation values. e.g. ">SOP75" means greater than the 75-th observation percentile.
    * SFCP for a percentile of the sample forecast climatology values. e.g. ">SFCP90" means greater than the 90-th forecast climatology percentile.
    * SOCP for a percentile of the sample observation climatology values. e.g. ">SOCP90" means greater than the 90-th observation climatology percentile. For backward compatibility, the "SCP" threshold type is processed the same as "SOCP".
    * USP for a user-specified percentile threshold. e.g. "<USP90(2.5)" means less than the 90-th percentile values which the user has already determined to be 2.5 outside of MET.
    * ==FBIAS for a user-specified frequency bias value. e.g. "==FBIAS1" to automatically debias the data, "==FBIAS0.9" to select a low-bias threshold, or "==FBIAS1.1" to select a high-bias threshold. This option must be used in conjunction with a simple threshold in the other field. For example, when "obs.cat_thresh = >5.0" and "fcst.cat_thresh = ==FBIAS1;", MET applies the >5.0 threshold to the observations and then chooses a forecast threshold which results in a frequency bias of 1. The frequency bias can be any float value > 0.0.
    * FCDP for forecast climatological distribution percentile thresholds. These thresholds require that the forecast climatological mean and standard deviation be defined using

the "climo_mean" and "climo_stdev" config file options, respectively. The categorical (cat_thresh), conditional (cnt_thresh), or wind speed (wind_thresh) thresholds can be defined relative to the climatological distribution at each point. Therefore, the actual numeric threshold applied can change for each point. e.g. ">FCDP50" means greater than the 50-th percentile of the climatological distribution for each point.

* OCDP for observation climatological distribution percentile thresholds. The "OCDP" threshold logic matches the "FCDP" logic described above. However these thresholds are defined using the observation climatological mean and standard deviation rather than the forecast climatological data. For backward compatibility, the "CDP" threshold type is processed the same as "OCDP".

– When percentile thresholds of type SFP, SOP, SFCP, SOCP, FCDP, or OCDP are requested for continuous filtering thresholds (cnt_thresh), wind speed thresholds (wind_thresh), or observation filtering thresholds (obs_thresh in ensemble_stat), the following special logic is applied. Percentile thresholds of type equality are automatically converted to percentile bins which span the values from 0 to 100. For example, ==OCDP25 is automatically expanded to 4 percentile bins: >=OCDP0&&<OCDP25,>=OCDP25&&<OCDP50,>=OCDP50&&<OCDP75,>=OCDP75&&<=OCDP100

– When sample percentile thresholds of type SFP, SOP, SFCP, SOCP, or FBIAS are requested, MET recomputes the actual percentile that the threshold represents. If the requested percentile and actual percentile differ by more than 5%, a warning message is printed. This may occur when the sample size is small or the data values are not truly continuous.

– When percentile thresholds of type SFP, SOP, SFCP, SOCP, or USP are used, the actual threshold value is appended to the FCST_THRESH and OBS_THRESH output columns. For example, if the 90-th percentile of the current set of forecast values is 3.5, then the requested threshold "<=SFP90" is written to the output as "<=SFP90(3.5)".

– When parsing FCST_THRESH and OBS_THRESH columns, the Stat-Analysis tool ignores the actual percentile values listed in parentheses.

Note: Prior to MET version 12.0.0, forecast climatological inputs were not supported. The observation climatological inputs were used to process threshold types named SCP and CDP.

For backward compatibility, the SCP threshold type is processed the same as SOCP and CDP the same as OCDP.

Users are encouraged to replace the deprecated SCP and CDP threshold types with the updated SOCP and OCDP types, respectively.

- Piecewise-Linear Function (currently used only by MODE):
  – A list of (x, y) points enclosed in parenthesis ().
  – The (x, y) points are *NOT* separated by commas.
- User-defined function of a single variable:
  – Left side is a function name followed by variable name in parenthesis.
  – Right side is an equation which includes basic math functions (+,-,*,/), built-in functions (listed below), or other user-defined functions.

  - Built-in functions include: sin, cos, tan, sind, cosd, tand, asin, acos, atan, asind, acosd, atand, atan2, atan2d, arg, argd, log, exp, log10, exp10, sqrt, abs, min, max, mod, floor, ceil, step, nint, sign

The context of a configuration entry matters. If an entry cannot be found in the expected dictionary, the MET tools recursively search for that entry in the parent dictionaries, all the way up to the top-level configuration file dictionary. If you'd like to apply the same setting across all cases, you can simply specify it once at the top-level. Alternatively, you can specify a setting at the appropriate dictionary level to have finer control over the behavior.

In order to make the configuration files more readable, several descriptive integer types have been defined in the ConfigConstants file. These integer names may be used on the right-hand side for many configuration file entries.

Each of the configurable MET tools expects a certain set of configuration entries. Examples of the MET configuration files can be found in *data/config* and *scripts/config*.

When you pass a configuration file to a MET tool, the tool actually parses up to four different configuration files in the following order:

1. Reads *share/met/config/ConfigConstants* to define constants.
2. If the tool produces PostScript output, it reads *share/met/config/ConfigMapData* to define the map data to be plotted.
3. Reads the default configuration file for the tool from *share/met/config*.
4. Reads the user-specified configuration file from the command line.

Many of the entries from step (3) are overwritten by the user-specified entries from step (4). Therefore, the configuration file you pass in on the command line really only needs to contain entries that differ from the defaults.

Any of the configuration entries may be overwritten by the user-specified configuration file. For example, the map data to be plotted may be included in the user-specified configuration file and override the default settings defined in the *share/met/config/ConfigMapData* file.

The configuration file language supports the use of environment variables. They are specified as ${ENV_VAR}, where ENV_VAR is the name of the environment variable. When scripting up many calls to the MET tools, you may find it convenient to use them. For example, when applying the same configuration to the output from multiple models, consider defining the model name as an environment variable which the controlling script sets prior to verifying the output of each model. Setting MODEL to that environment variable enables you to use one configuration file rather than maintaining many very similar ones.

An error in the syntax of a configuration file will result in an error from the MET tool stating the location of the parsing error.

5.1 Runtime Environment Variables

5.1.1 User-Specified Environment Variables

When editing configuration files, environment variables may be used for setting the configurable parameters if convenient. The configuration file parser expands environment variables to their full value before proceeding. Within the configuration file, environment variables must be specified in the form **${VAR_NAME}**.

For example, using an environment variable to set the message_type (see below) parameter to use ADPUPA and ADPSFC message types might consist of the following.

Setting the environment variable in a Bash Shell:

```
export MSG_TYP='"ADPUPA", "ADPSFC"'
```

Referencing that environment variable inside a MET configuration file:

```
message_type = [ ${MSG_TYP} ];
```

In addition to supporting user-specified environment variables within configuration files, the environment variables listed below have special meaning if set at runtime.

5.1.2 MET_AIRNOW_STATIONS

The MET_AIRNOW_STATIONS environment variable can be used to specify a file that will override the default file. If set, it should be the full path to the file. The default table can be found in the installed *share/met/table_files/airnow_monitoring_site_locations_v2.dat*. This file contains ascii column data that allows lookups of latitude, longitude, and elevation for all AirNow stations based on stationId and/or AqSid.

Additional information and updated site locations can be found at the EPA AirNow website. While some monitoring stations are permanent, others are temporary, and theirs locations can change. When running the ASCII2NC tool with the `-format airnowhourly` option, users should download the **Monitoring_Site_Locations_V2.dat** data file for the date being processed and set the MET_AIRNOW_STATIONS environment variable to define its location.

5.1.3 MET_NDBC_STATIONS

The MET_NDBC_STATIONS environment variable can be used to specify a file that will override the default file. If set it should be a full path to the file. The default table can be found in the installed *share/met/table_files/ndbc_stations.xml*. This file contains XML content for all stations that allows lookups of latitude, longitude, and, in some cases, elevation for all stations based on stationId.

This set of stations comes from 2 online sources: the active stations website and the complete stations website.

As these lists can change as a function of time, a script can be run to pull down the contents of both websites and merge any changes with the existing stations file content, creating an updated stations file locally. The MET_NDBC_STATIONS environment variable can be then set to refer to this newer stations file. Also, the MET development team will periodically run this script and update *share/met/table_files/ndbc_stations.xml*.

To run this utility:

```
build_ndbc_stations_from_web.py <-d> <-p> <-o OUTPUT_FILE> <-e EXISTING_FILE>

Usage: build_ndbc_stations_from_web.py [options]
Options:
  -h, --help            show this help message and exit
  -d, --diagnostic      Rerun using downlaoded files, skipping download step (optional,␣
↪default: False)
  -p, --prune           Prune files that are no longer online (optional, default: False)
  -o OUT_FILE, --out=OUT_FILE
                        Save the text into the named file (optional, default: merged.txt)
  -e EXISTING_FILE, --existing=EXISTING_FILE
                        Save the text into the named file (optional, default: ../../../data/
↪table_files/ndbc_stations.xml)
```

Note: The downloaded files are written to a subdirectory ndbc_temp_data which can be deleted once the final output file is created.

5.1.4 MET_BASE

The MET_BASE variable is defined in the code at compilation time as the path to the MET shared data. These are things like the default configuration files, common polygons and color scales. MET_BASE may be used in the MET configuration files when specifying paths and the appropriate path will be substituted in. If MET_BASE is defined as an environment variable, its value will be used instead of the one defined at compilation time.

5.1.5 MET_OBS_ERROR_TABLE

The MET_OBS_ERROR_TABLE environment variable can be set to specify the location of an ASCII file defining observation error information. The default table can be found in the installed *share/met/table_files/obs_error_table.txt*. This observation error logic is applied in Ensemble-Stat to perturb ensemble member values and/or define observation bias corrections.

When processing point and gridded observations, Ensemble-Stat searches the table to find the entry defining the observation error information. The table consists of 15 columns and includes a header row defining each column. The special string ALL is interpreted as a wildcard in these files. The first 6 columns (OBS_VAR, MESSAGE_TYPE, PB_REPORT_TYPE, IN_REPORT_TYPE, INSTRUMENT_TYPE, and STATION_ID) may be set to a comma-separated list of strings to be matched. In addition, the strings in the OBS_VAR column are interpreted as regular expressions when searching for a match. For example, setting the OBS_VAR column to 'APCP_[0-9]+' would match observations for both APCP_03 and APCP_24. The HGT_RANGE, VAL_RANGE, and PRS_RANGE columns should either be set to ALL or BEG,END where BEG and END specify the range of values to be used. The INST_BIAS_SCALE and INST_BIAS_OFFSET columns define instrument bias adjustments which are applied to the observation values. The DIST_TYPE and DIST_PARM columns define the distribution from which random perturbations should be drawn and applied to the ensemble

member values. See the obs_error description below for details on the supported error distributions. The last two columns, MIN and MAX, define the bounds for the valid range of the bias-corrected observation values and randomly perturbed ensemble member values. Values less than MIN are reset to the minimum value and values greater than MAX are reset to the maximum value. A value of NA indicates that the variable is unbounded.

If the observation error table contains no entries for the variable being evaluated, MET prints a warning message and disables the observation error logic. If the table does contain entries for the variable, but the lookup for an individual observation fails, that observation is excluded from the analysis. This situation would occur, for example, when the observation value falls outside the range of values in the VAL_RANGE column.

5.1.6 MET_GRIB_TABLES

The MET_GRIB_TABLES environment variable can be set to specify the location of custom GRIB tables. It can either be set to a specific file name or to a directory containing custom GRIB tables files. These file names must begin with a "grib1" or "grib2" prefix and end with a ".txt" suffix. Their format must match the format used by the default MET GRIB table files, described below. The custom GRIB tables are read prior to the default tables and their settings take precedence.

At runtime, the MET tools read default GRIB tables from the installed *share/met/table_files* directory, and their file formats are described below:

GRIB1 table files begin with "grib1" prefix and end with a ".txt" suffix. The first line of the file must contain GRIB1. The following lines consist of 4 integers followed by 3 strings:

Column 1: GRIB code (e.g. 11 for temperature)
Column 2: parameter table version number
Column 3: center id (e.g. 07 for US Weather Service- National Met. Center)
Column 4: subcenter id
Column 5: variable name
Column 6: variable description
Column 7: units

References: | Office Note 388 GRIB1 | A Guide to the Code Form FM 92-IX Ext. GRIB Edition 1

GRIB2 table files begin with "grib2" prefix and end with a ".txt" suffix. The first line of the file must contain GRIB2. The following lines consist of 8 integers followed by 3 strings.

Column 1: Section 0 Discipline
Column 2: Section 1 Master Tables Version Number
Column 3: Section 1 Master Tables Version Number, low range of tables
Column 4: Section 1 Master Table Version Number, high range of tables
Column 5: Section 1 originating center

Column 6: Local Tables Version Number
Column 7: Section 4 Template 4.0 Parameter category
Column 8: Section 4 Template 4.0 Parameter number
Column 9: variable name
Column 10: variable description
Column 11: units

References: | NCEP WMO GRIB2 Documentation

5.1.7 OMP_NUM_THREADS

5.1.7.1 Introduction

There are a number of different ways of parallelizing code. OpenMP offers parallelism within a single shared-memory workstation or supercomputer node. The programmer writes OpenMP directives into the code to parallelize particular code regions.

When a parallelized code region is reached, which we shall hereafter call a parallel region, a number of threads are spawned and work is shared among them. Running on different cores, this reduces the execution time. At the end of the parallel region, the code returns to single-thread execution.

The number of code regions parallelized in both the MET application and library code was significantly increased for version 12.1.0. Generally, parallelism has been applied to compute-intensive algorithms that loop over grid dimensions. Further expanded use of parallelism is planned for future versions of MET.

Due to the broad application of OpenMP, nearly all MET applications benefit from it. However, initializing OpenMP threads does incur some overhead cost. Typically the runtime benefit dramatically outweighs the setup cost. Generally, more threads produces faster runtimes, but that is not always the case. The optimal number of threads for any single run of a MET tool is data dependent.

5.1.7.2 Setting the number of threads

The number of threads is controlled by the OMP_NUM_THREADS environment variable. For example, on a quad core machine, the user might choose to run on 4 threads:

```
export OMP_NUM_THREADS=4
```

Alternatively, the variable may be specified as a prefix to the executable itself. For example:

```
OMP_NUM_THREADS=4 <exec>
```

Since OpenMP is initialized by all MET applications, they print common log messages about the runtime settings. If OMP_NUM_THREADS is set, the number of threads is reported:

```
DEBUG 2: OpenMP running on 16 thread(s).
```

The case where this variable remains unset is handled inside the code, which defaults to a single thread. A log message invites users to set it:

```
DEBUG 2: Defaulting unset OMP_NUM_THREADS to use 1 of 40 available threads.
         Recommend setting OMP_NUM_THREADS for faster runtimes.
DEBUG 2: OpenMP running on 1 thread(s).
```

There are choices when deciding how many threads to use. To perform a single run as fast as possible, it would likely be appropriate to use as many threads as there are (physical) cores available on the specific system. However, it is not a cast-iron guarantee that more threads will always translate into more speed. In theory, there is a chance that running across multiple non-uniform memory access (NUMA) regions may carry negative performance impacts. This has not been observed in practice, however.

A lower thread count is appropriate when time-to-solution is not so critical, because cores remain idle when the code is not inside a parallel region. Fewer threads typically means better resource utilization.

5.1.7.3 Thread Binding

It is normally beneficial to bind threads to particular cores, sometimes called *affinitization*. There are a few reasons for this, but at the very least it guarantees that threads remain evenly distributed across the available cores. Otherwise, the operating system may migrate threads between cores during a run.

OpenMP provides some environment variables to handle this: `OMP_PLACES` and `OMP_PROC_BIND`. We anticipate that the effect of setting only `OMP_PROC_BIND=true` would be neutral-to-positive.

However, there are sometimes compiler-specific environment variables. Instead, thread affinitization is sometimes handled by MPI launchers, since OpenMP is often used in MPI codes to reduce intra-node communications.

Where code is running in a production context, it is worth being familiar with the binding / affinitization method on the particular system and building it into any relevant scripting.

5.1.8 MET_KEEP_TEMP_FILE

The MET_KEEP_TEMP_FILE environment variable can be set to control the runtime behavior of the MET tools. The MET tools write temporary files in several places in the application and library code. By default, those temporary files are deleted when they are no longer needed. However it can be useful for development, testing, and debugging to keep them for further inspection. Setting this environment variable to a value of `yes` or `true` instructs the MET tools to retain temporary files instead of deleting them.

Note that doing so may fill up the temporary directory. It is the responsiblity of the user to monitor the temporary directory usage and remove temporary files that are no longer needed.

When running with this option, users are advised to refer to section Section 5.2.5 and write temporary files to a personal location rather than the default shared `/tmp` directory.

5.1.9 MET_PYTHON_DEBUG

The MET_PYTHON_DEBUG environment variable can be set to enable debugging log messages related to Python embedding. These log messages are disabled by default. The environment variable can be set to a value of `all` for all log messages, `dataplane` for log messages when reading gridded data, or `point` for log messages when reading point data.

5.1.10 MET_PYTHON_TMP_FORMAT

The MET_PYTHON_TMP_FORMAT environment variable defines whether temporary files for Python embedding should be written as NetCDF files or using JSON/NumPy serialization. By default, they are written using JSON for attributes and NumPy serialization for data to avoid NetCDF library conflicts between MET and Python. Setting this environment variable to `netcdf` enables the use of temporary NetCDF files instead.

5.1.11 MET_USE_WRF_SUBGRID

The MET_USE_WRF_SUBGRID environment variable controls how the grid information is read from WRF files. Some WRF files contain fields that are on a subgrid, which contain more grid points and require a computation to determine the d_km value. MET reads the grid information from a file before any fields are read and assumes that there is one grid definition per file. To avoid a complicated refactor to read the grid information after a field is read, this environment variable is used to tell MET that it should create the grid using the subgrid dimensions. If the environment variable is set to any value, the subgrid values will be read.

5.2 Settings Common to Multiple Tools

5.2.1 exit_on_warning

The "exit_on_warning" entry in ConfigConstants may be set to true or false. If set to true and a MET tool encounters a warning, it will immediately exit with bad status after writing the warning message.

```
exit_on_warning = FALSE;
```

5.2.2 time_offset_warning

The "time_offset_warning" entry in ConfigConstants defines an allowable offset in seconds to silence time differences warning messages. Several MET tools check the timestamps of the datasets being compared and print a warning message if they differ. Increasing this option from its default value of 0 seconds enables datasets that are close in time to be compared without triggering a warning. If the absolute value of a non-zero time difference is less than or equal to this setting, a debug log message is written instead of a warning.

```
time_offset_warning = 0;
```

5.2.3 nc_compression

The "nc_compression" entry in ConfigConstants defines the compression level for the NetCDF variables. Setting this option in the config file of one of the tools overrides the default value set in ConfigConstants. The environment variable MET_NC_COMPRESS overrides the compression level from configuration file. The command line argument "-compress n" for some tools overrides it. The range is 0 to 9.

- 0 is to disable the compression.
- 1 to 9: Lower number is faster, higher number for smaller files.

WARNING: Selecting a high compression level may slow down the reading and writing of NetCDF files within MET significantly.

```
nc_compression = 0;
```

5.2.4 output_precision

The "output_precision" entry in ConfigConstants defines the precision (number of significant decimal places) to be written to the ASCII output files. Setting this option in the config file of one of the tools will override the default value set in ConfigConstants.

```
output_precision = 5;
```

5.2.5 tmp_dir

The "tmp_dir" entry in ConfigConstants defines the directory for the temporary files. The directory must exist and be writable. The environment variable MET_TMP_DIR overrides the default value at the configuration file. Some tools override the temporary directory by the command line argument "-tmp_dir <directory_name>".

```
tmp_dir = "/tmp";
```

A description of the use of temporary files in MET can be found in `Contributor's Guide Section %s`.

5.2.6 message_type_group_map

The "message_type_group_map" entry is an array of dictionaries, each containing a "key" string and "val" string. This defines a mapping of message type group names to a comma-separated list of values. This map is defined in the config files for PB2NC, IODA2NC, Point-Stat, and Ensemble-Stat. Modify this map to define sets of message types that should be processed together as a group.

Beyond defining groups of message types, the following entries have special meaning:

- The "SURFACE" entry defines message types for which surface verification logic should be applied.
- The "LANDSF" and "WATERSF" entries differentiate between point observations over land and water, as described in "land_mask" below.

- The "LAPSERT" entry defines the message types for which surface temperature lapse rate correction logic should be applied, as described in "lapse_rate_correction" below.
- The "MSLAGL" entry defines the message types for which MSL/AGL height conversion logic should be applied, as described in "msl_agl_conversion" below.

If not defined, the default values listed below are used.

```
message_type_group_map = [
   { key = "SURFACE"; val = "ADPSFC,SFCSHP,MSONET";               },
   { key = "ANYAIR";  val = "AIRCAR,AIRCFT";                      },
   { key = "ANYSFC";  val = "ADPSFC,SFCSHP,ADPUPA,PROFLR,MSONET"; },
   { key = "ONLYSF";  val = "ADPSFC,SFCSHP";                      },
   { key = "LANDSF";  val = "ADPSFC,MSONET";                      },
   { key = "WATERSF"; val = "SFCSHP";                             },
   { key = "LAPSERT"; val = "ADPSFC,MSONET";                      },
   { key = "MSLAGL";  val = "";                                   }
];
```

5.2.7 obtype_as_group_val_flag

The "obtype_as_group_val_flag" entry is a boolean that controls how the OBTYPE header column is populated for message type groups defined in "message_type_group_map". If set to TRUE and when writing matched pair line types (MPR, SEEPS_MPR, and ORANK), write OBTYPE as the group map *value*, i.e. the input message type for each individual observation. If set to FALSE (default) and for all other line types, write OBTYPE as the group map key, i.e. the name of the message type group.

For example, if FALSE, write the OBTYPE column in the MPR line type as the "ANYAIR" message type group name. If TRUE, write OBTYPE as "AIRCAR" or "AIRCFT", based on the input message type of each point observation.

```
obtyp_as_group_val_flag = FALSE;
```

5.2.8 message_type_map

The "message_type_map" entry is an array of dictionaries, each containing a "key" string and "val" string. This defines a mapping of input strings to output message types. This mapping is applied in ASCII2NC when converting input little_r report types to output message types. This mapping is also supported in PBN2NC as a way of renaming input PREPBUFR message types.

```
message_type_map = [
   { key = "FM-12 SYNOP";  val = "ADPSFC"; },
   { key = "FM-13 SHIP";   val = "SFCSHP"; },
   { key = "FM-15 METAR";  val = "ADPSFC"; },
   { key = "FM-18 BUOY";   val = "SFCSHP"; },
   { key = "FM-281 QSCAT"; val = "ASCATW"; },
   { key = "FM-32 PILOT";  val = "ADPUPA"; },
```

(continues on next page)

(continued from previous page)

```
   { key = "FM-35 TEMP";   val = "ADPUPA"; },
   { key = "FM-88 SATOB";  val = "SATWND"; },
   { key = "FM-97 ACARS";  val = "AIRCFT"; }
];
```

5.2.9 model

The "model" entry specifies a name for the model being verified. This name is written to the MODEL column of the ASCII output generated. If you're verifying multiple models, you should choose descriptive model names (no whitespace) to distinguish between their output. e.g. model = "GFS";

```
model = "FCST";
```

5.2.10 desc

The "desc" entry specifies a user-specified description for each verification task. This string is written to the DESC column of the ASCII output generated. It may be set separately in each "obs.field" verification task entry or simply once at the top level of the configuration file. If you're verifying the same field multiple times with different quality control flags, you should choose description strings (no whitespace) to distinguish between their output. e.g. desc = "QC_9";

```
desc = "NA";
```

5.2.11 obtype

The "obtype" entry specifies a name to describe the type of verifying gridded observation used. This name is written to the OBTYPE column in the ASCII output generated. If you're using multiple types of verifying observations, you should choose a descriptive name (no whitespace) to distinguish between their output. When verifying against point observations the point observation message type value is written to the OBTYPE column. Otherwise, the configuration file obtype value is written.

```
obtype = "ANALYS";
```

5.2.12 regrid

The "regrid" entry is a dictionary containing information about how to handle input gridded data files. The "regrid" entry specifies regridding logic using the following entries:

5.2.12.1 to_grid

The "to_grid" entry may be set to NONE, FCST, OBS, a named grid, the path to a gridded data file defining the grid, or an explicit grid specification string.

- to_grid = NONE; To disable regridding.
- to_grid = FCST; To regrid observations to the forecast grid.
- to_grid = OBS; To regrid forecasts to the observation grid.
- to_grid = "G218"; To regrid both to a named grid.
- to_grid = "path"; To regrid both to a grid defined by a file.
- to_grid = "spec"; To define a grid specification string, as described in *Appendix B Map Projections, Grids, and Polylines* (page 537).

5.2.12.2 vld_thresh

The "vld_thresh" entry specifies a proportion between 0 and 1 to define the required ratio of valid data points. When regridding, compute a ratio of the number of valid data points to the total number of points in the neighborhood. If that ratio is less than this threshold, write bad data for the current point.

5.2.12.3 method

The "method" entry defines the regridding method to be used.

- Valid regridding methods:
  - MIN for the minimum value
  - MAX for the maximum value
  - MEDIAN for the median value
  - UW_MEAN for the unweighted average value
  - DW_MEAN for the distance-weighted average value (weight = distance ^ -2)
  - AW_MEAN for an area-weighted mean when regridding from high to low resolution grids (width = 1)
  - LS_FIT for a least-squares fit
  - BILIN for bilinear interpolation (width = 2)
  - NEAREST for the nearest grid point (width = 1)
  - BUDGET for the mass-conserving budget interpolation
    * The budget interpolation method is often used for precipitation in order to roughly conserve global averages. However it is computationally intensive and relatively slow. To compute the interpolated value for each point of the target grid, a higher resolution 5x5 mesh with 0.2 grid box spacing is centered on the point and bilinear interpolation is performed for each

of those 25 lat/lon locations. The budget interpolation value is computed as the average of those 25 bilinear interpolation values, assuming enough valid data is present to meet the "vld_thresh" threshold.

- FORCE to compare gridded data directly with no interpolation as long as the grid x and y dimensions match.
- UPPER_LEFT for the upper left grid point (width = 1)
- UPPER_RIGHT for the upper right grid point (width = 1)
- LOWER_RIGHT for the lower right grid point (width = 1)
- LOWER_LEFT for the lower left grid point (width = 1)
- MAXGAUSS to compute the maximum value in the neighborhood and apply a Gaussian smoother to the result

The BEST, GEOG_MATCH, and HIRA options are not valid for regridding.

5.2.12.4 width

The "width" entry specifies a regridding width, when applicable. `width = 4;` To regrid using a 4x4 box or circle with diameter 4.

5.2.12.5 shape

The "shape" entry defines the shape of the neighborhood. Valid values are `SQUARE` or `CIRCLE`

5.2.12.6 gaussian_dx

The "gaussian_dx" entry specifies a delta distance for Gaussian smoothing. The default is 81.271. Ignored if not Gaussian method.

5.2.12.7 gaussian_radius

The "gaussian_radius" entry defines the radius of influence for Gaussian smoothing. The default is 120. Ignored if not Gaussian method.

Note: The "gaussian_dx" and "gaussian_radius" settings must be in the same units, such as kilometers or degress. Their ratio (sigma = gaussian_radius / gaussian_dx) determines the Guassian weighting function.

5.2.12.8 convert, censor_thresh, and censor_val

The "convert", "censor_thresh", and "censor_val" entries are described below. When specified, these operations are applied to the output of the regridding step. The conversion operation is applied first, followed by the censoring operation. Note that these operations are limited in scope. They are only applied if defined within the regrid dictionary itself. Settings defined at higher levels of config file context are not applied.

```
regrid = {
   to_grid         = NONE;
   method          = NEAREST;
   width           = 1;
   vld_thresh      = 0.5;
   shape           = SQUARE;
   gaussian_dx     = 81.271;
   gaussian_radius = 120;
   convert(x)      = x;
   censor_thresh   = [];
   censor_val      = [];
}
```

5.2.13 fcst

The "fcst" entry is a dictionary containing information about the field(s) to be verified. This dictionary may include the following entries:

5.2.13.1 field

The "field" entry is an array of dictionaries, each specifying a verification task. Each of these dictionaries may include:

5.2.13.1.1 field.name

The "name" entry specifies a name for the field. Setting "name" is file-format specific. See *File-format Specific Settings for the "field" Entry* (page 62).

5.2.13.1.2 field.level

The "level" entry specifies level information for the field. Setting "level" is file-format specific. See *File-format Specific Settings for the "field" Entry* (page 62).

5.2.13.1.3 field.prob

The "prob" entry in the forecast dictionary defines probability information. It may either be set as a boolean (i.e. TRUE or FALSE) or as a dictionary defining probabilistic field information.

When set as a boolean to TRUE, it indicates that the "fcst.field" data should be treated as probabilities. For example, when verifying the probabilistic NetCDF output of Gen-Ens-Prod for an ensemble of size 10, one could configure the Grid-Stat or Point-Stat tools as follows:

```
fcst = {
   field = [ { name       = "APCP_24_A24_ENS_FREQ_gt0.0";
               level      = "(*,*)";
               cat_thresh = ==10;
               prob       = TRUE; } ];
   }
```

Setting "prob = TRUE" indicates that the "APCP_24_A24_ENS_FREQ_gt0.0" data should be processed as probabilities. Setting "cat_thresh = ==10" indicates that these probabilities are derived from an ensemble with 10 members and 11 probability bins should be defined, each centered on the value n/10 for n = 0, 1, ... 10.

When set as a dictionary, it defines the probabilistic field to be used. For example, when verifying GRIB files containing probabilistic data, one could configure the Grid-Stat or Point-Stat tools as follows:

```
fcst = {
   field = [ { name       = "PROB"; level = "A24";
               prob       = { name = "APCP"; thresh_lo = 2.54; }
               cat_thresh = ==0.25; },
             { name       = "PROB"; level = "P850";
               prob       = { name = "TMP"; thresh_hi = 273; }
               cat_thresh = ==0.1; } ];
}
```

The example above selects two probabilistic fields. In both, "name" is set to "PROB", the GRIB abbreviation for probabilities. The "level" entry defines the level information (i.e. "A24" for a 24-hour accumulation and "P850" for 850mb). The "prob" dictionary defines the event for which the probability is defined. The "thresh_lo" (i.e. APCP > 2.54) and/or "thresh_hi" (i.e. TMP < 273) entries are used to define the event threshold(s).

Probability fields should contain values in the range [0, 1] or [0, 100]. However, when MET encounters a probability field with a range [0, 100], it will automatically rescale it to be [0, 1] before applying the probabilistic verification methods.

Probabilistic statistics in MET are derived from an Nx2 probabilistic contingency table. The N-dimension is determined by the number of probability bins requested. The "cat_thresh" configuration option defines the number of and size of these probabibility bins. The bins must include the full range of possible probability values, [0, 1]. Since selecting bins of equal width is common, shorthand notation is provided to do so. The following options are supported.

- `cat_thresh = [ ==0.25 ];` specifies an equal probability bin width of 0.25 and defines 4 bins between

the values 0, 0.25, 0.5, 0.75, and 1.0. The ==p threshold may be set to any probability bin width greater than 0 and less than 1.

- `cat_thresh = [ ==10 ];` specifies probability bins for an ensemble of size 10 and defines 11 bins between the values -0.05, 0.05, 0.15, ..., 0.95, and 1.05. Note that each bin is centered on the probability value n/10, for n = 0 to 10. The ==n threshold may be set to any integer number of ensemble members greater than 1 to define n+1 probability bins.
- `cat_thresh = [ >=0, >=0.5, >=0.75, >=1.0 ];` explicitly specifies the probability thresholds and defines 3 bins of unequal width between the values 0, 0.5, 0.75, and 1.0. By convention, the greater-than-or-equal-to (">=" or "ge") inequality type is required.

5.2.13.1.4 field.prob_as_scalar

Set "prob_as_scalar = TRUE" to override the processing of probability data. When the "prob" entry is set as a dictionary to define the field of interest, setting "prob_as_scalar = TRUE" indicates that this data should be processed as regular scalars rather than probabilities. For example, this option can be used to compute traditional 2x2 contingency tables and neighborhood verification statistics for probability data. It can also be used to compare two probability fields directly. When this flag is set, probability values are automatically rescaled from the range [0, 100] to [0, 1].

5.2.13.1.5 field.convert

The "convert" entry is a user-defined function of a single variable for processing input data values. Any input values that are not bad data are replaced by the value of this function. The convert function is applied prior to regridding or thresholding. This function may include any of the built-in math functions (e.g. sqrt, log10) described above. Several standard unit conversion functions are already defined in *data/config/ConfigConstants*. Examples of user-defined conversion functions include:

```
convert(x) = 2*x;
convert(x) = x^2;
convert(a) = log10(a);
convert(a) = a^10;
convert(t) = max(1, sqrt(abs(t)));
convert(x) = K_to_C(x); where K_to_C(x) is defined in
                        ConfigConstants
```

5.2.13.1.6 field.censor_thresh and field.censor_val

The "censor_thresh" entry is an array of thresholds to be applied to the input data. The "censor_val" entry is an array of numbers and must be the same length as "censor_thresh". These arguments must appear together in the correct format (threshold and number). For each censor threshold, any input values meeting the threshold criteria will be reset to the corresponding censor value. An empty list indicates that no censoring should be performed. The censoring logic is applied prior to any regridding but after the convert function. All statistics are computed on the censored data. These entries may be used to apply quality control logic by

resetting data outside of an expected range to the bad data value of -9999. These entries are not indicated in the metadata of any output files, but the user can set the "desc" entry accordingly.

Examples of user-defined data censoring operations include:

```
censor_thresh = [ >12000 ];
censor_val    = [  12000 ];
```

5.2.13.1.7 field.mpr_column and field.mpr_thresh

The "mpr_column" and "mpr_thresh" entries are arrays of strings and thresholds to specify which matched pairs should be included in the statistics. These options apply to the Point-Stat and Grid-Stat tools. They are parsed seperately for each "obs.field" array entry. The "mpr_column" strings specify MPR column names (FCST, OBS, CLIMO_MEAN, CLIMO_STDEV, or CLIMO_CDF), differences of columns (FCST-OBS), or the absolute value of those differences (ABS(FCST-OBS)). The number of "mpr_thresh" thresholds must match the number of "mpr_column" entries, and the n-th threshold is applied to the n-th column. Any matched pairs which do not meet any of the specified thresholds are excluded from the analysis. For example, the following settings exclude matched pairs where the observation value differs from the forecast or climatological mean values by more than 10:

```
mpr_column = [ "ABS(OBS-FCST)", "ABS(OBS-CLIMO_MEAN)" ];
mpr_thresh = [ <=10, <=10 ];
```

5.2.13.1.8 field.cat_thresh

The "cat_thresh" entry is an array of thresholds to be used when computing categorical statistics.

5.2.13.1.9 field.cnt_thresh

The "cnt_thresh" entry is an array of thresholds for filtering data prior to computing continuous statistics and partial sums.

5.2.13.1.10 field.cnt_logic

The "cnt_logic" entry may be set to UNION, INTERSECTION, or SYMDIFF and controls the logic for how the forecast and observed cnt_thresh settings are combined when filtering matched pairs of forecast and observed values.

5.2.13.1.11 Override and Correct Metadata

Several configuration options are provided to override and correct the metadata read from the input file. The supported options are listed below:

```
// Data attributes
set_attr_name      = "string";
set_attr_level     = "string";
set_attr_units     = "string";
set_attr_long_name = "string";

// Time attributes
set_attr_init  = "YYYYMMDD[_HH[MMSS]]";
set_attr_valid = "YYYYMMDD[_HH[MMSS]]";
set_attr_lead  = "HH[MMSS]";
set_attr_accum = "HH[MMSS]";

// Grid definition (must match the actual data dimensions)
set_attr_grid  = "named grid or grid specification string";

// Flags
is_precipitation     = boolean;
is_specific_humidity = boolean;
is_u_wind            = boolean;
is_v_wind            = boolean;
is_grid_relative     = boolean;
is_wind_speed        = boolean;
is_wind_direction    = boolean;
is_prob              = boolean;
```

5.2.13.2 file_type

The "file_type" entry specifies the input gridded data file type rather than letting the code determine it. MET determines the file type by checking for known suffixes and examining the file contents. Use this option to override the code's choice. The valid file_type values are listed the "data/config/ConfigConstants" file and are described below. This entry should be defined within the "fcst" and/or "obs" dictionaries. For example:

```
fcst = {
   file_type = GRIB1;         GRIB version 1
   file_type = GRIB2;         GRIB version 2
   file_type = NETCDF_MET;    NetCDF created by another MET tool
   file_type = NETCDF_WRF;    NetCDF WRF output.
   file_type = NETCDF_PINT;   NetCDF created by running the p_interp
                              or wrf_interp utility on WRF output.
                              May be used to read unstaggered raw WRF
                              NetCDF output at the surface or a
```

(continues on next page)

(continued from previous page)

```
                              single model level.
   file_type = NETCDF_NCCF;   NetCDF following the Climate Forecast
                              (CF) convention.
   file_type = NETCDF_UGRID;  NetCDF containing data on an
                              unstructured grid.
   file_type = PYTHON_NUMPY;  Run a Python script to load data into
                              a NumPy array.
   file_type = PYTHON_XARRAY; Run a Python script to load data into
                              an xarray object.
}
```

5.2.13.3 wind_thresh

The "wind_thresh" entry is an array of thresholds used to filter wind speed values when computing VL1L2 vector partial sums. Only those U/V pairs that meet this wind speed criteria will be included in the sums. Setting this threshold to NA will result in all U/V pairs being used.

5.2.13.4 wind_logic

The "wind_logic" entry may be set to UNION, INTERSECTION, or SYMDIFF and controls the logic for how the forecast and observed wind_thresh settings are combined when filtering matched pairs of forecast and observed wind speeds.

5.2.13.5 eclv_points

The "eclv_points" entry specifies the economic cost/loss ratio points to be evaluated. For each cost/loss ratio specified, the relative value will be computed and written to the ECLV output line. This entry may either be specified as an array of numbers between 0 and 1 or as a single number. For an array, each array entry will be evaluated. For a single number, all evenly spaced points between 0 and 1 will be evaluated, where eclv_points defines the spacing. Cost/loss values are omitted for ratios of 0.0 and 1.0 since they are undefined.

5.2.13.6 init_time

The "init_time" entry specifies the initialization time in YYYYMMDD[_HH[MMSS]] format. This entry can be included in the "fcst" entry as shown below or included in the "field" entry if the user would like to use different initialization times for different fields.

5.2.13.7 valid_time

The "valid_time" entry specifies the valid time in YYYYMMDD[_HH[MMSS]] format. This entry can be included in the "fcst" entry as shown below or included in the "field" entry if the user would like to use different valid times for different fields.

5.2.13.8 lead_time

The "lead_time" entry specifies the lead time in HH[MMSS] format. This entry can be included in the "fcst" entry as shown below or included in the "field" entry if the user would like to use different lead times for different fields.

Note: It is only necessary to use the "init_time", "valid_time", and/or "lead_time" settings when verifying a file containing data for multiple output times. For example, to verify a GRIB file containing data for many lead times, you could use "lead_time" to specify the record to be verified.

5.2.13.9 File-format Specific Settings for the "field" Entry

5.2.13.9.1 GRIB1 and GRIB2

- For custom GRIB tables, see note about MET_GRIB_TABLES.
- The "name" entry specifies a GRIB code number or abbreviation.
  - GRIB1 Product Definition Section
  - GRIB2 Product Definition Section
- The "level" entry specifies a level type and value:
  - ANNN for accumulation interval NNN
  - ZNNN for vertical level NNN
  - ZNNN-NNN for a range of vertical levels
  - PNNN for pressure level NNN in hPa
  - PNNN-NNN for a range of pressure levels in hPa
  - LNNN for a generic level type
  - RNNN for a specific GRIB record number
- The "GRIB_lvl_typ" entry is an integer specifying the level type.
- **The "GRIB_lvl_val1" and "GRIB_lvl_val2" entries are floats specifying** the first and second level values.
- The "GRIB_ens" entry is a string specifying NCEP's usage of the extended PDS for ensembles. Set to "hi_res_ctl", "low_res_ctl", "+n", or "-n", for the n-th ensemble member.

- The GRIB1_ptv entry is an integer specifying the GRIB1 parameter table version number.
- The GRIB1_code entry is an integer specifying the GRIB1 code (wgrib kpds5 value).
- The GRIB1_center is an integer specifying the originating center.
- The GRIB1_subcenter is an integer specifying the originating subcenter.
- The GRIB1_tri is an integer specifying the time range indicator.
- The GRIB2_mtab is an integer specifying the master table number.
- The GRIB2_ltab is an integer specifying the local table number.
- The GRIB2_disc is an integer specifying the GRIB2 discipline code.
- The GRIB2_parm_cat is an integer specifying the parameter category code.
- The GRIB2_parm is an integer specifying the parameter code.
- The GRIB2_pdt is an integer specifying the product definition template (Table 4.0).
- The GRIB2_process is an integer specifying the generating process (Table 4.3).
- The GRIB2_cntr is an integer specifying the originating center.
- The GRIB2_ens_type is an integer specifying the ensemble type (Table 4.6).
- The GRIB2_der_type is an integer specifying the derived product type (Table 4.7).
- The GRIB2_stat_type is an integer specifying the statistical processing type (Table 4.10).
- The GRIB2_perc_val is an integer specifying the requested percentile value (0 to 100) to be used. This applies only to GRIB2 product definition templates 4.6 and 4.10.
- The GRIB2_aerosol_type is an integer specifying the aerosol type (Table 4.233). This applies only to GRIB2 product definition templates 4.46 and 4.48.
- The GRIB2_aerosol_interval_type is an integer specifying the aerosol size interval (Table 4.91). This applies only to GRIB2 product definition templates 4.46 and 4.48.
- The GRIB2_aerosol_size_lower and "GRIB2_aerosol_size_upper" are doubles specifying the endpoints of the aerosol size interval. These applies only to GRIB2 product defintion templates 4.46 and 4.48.
- The GRIB2_ipdtmpl_index and GRIB2_ipdtmpl_val entries are arrays of integers which specify the product description template values to be used. The indices are 0-based. For example, use the following to request a GRIB2 record whose 9-th and 27-th product description template values are 1 and 2, respectively:

  GRIB2_ipdtmpl_index=[8, 26]; GRIB2_ipdtmpl_val=[1, 2];

5.2.13.9.2 NetCDF

From MET tools, CF-compliant, p_interp, and wrf_interp:

- The "name" entry specifies the NetCDF variable name.
- The "level" entry specifies the dimensions to be used:
  - (i,...,j,*,*) for a single field, where i,...,j specifies fixed dimension values and , specifies the two dimensions for the gridded field. @ specifies the vertical level value or time value instead of offset, (i,...,@NNN,*,*). For example:

```
field = [
     {
       name       = "QVAPOR";
       level      = "(0,5,*,*)";
     },
     {
       name       = "TMP_P850_ENS_MEAN";
       level      = [ "(*,*)" ];
     }
   ];

field = [
     {
       name       = "QVAPOR";
       level      = "(@20220601_1200,@850,*,*)";
     },
     {
       name       = "TMP_P850_ENS_MEAN";
       level      = [ "(*,*)" ];
     }
   ];
```

5.2.13.9.3 Python

Using PYTHON_NUMPY or PYTHON_XARRAY:

- The Python interface for MET is described in Appendix F of the MET User's Guide.
- Two methods for specifying the Python command and input file name are supported. For tools which read a single gridded forecast and/or observation file, both options work. However, only the second option is supported for tools which read multiple gridded data files, such as Ensemble-Stat, Series-Analysis, and MTD.

Option 1:

- On the command line, replace the path to the input gridded data file with the constant string PYTHON_NUMPY or PYTHON_XARRAY.

- Specify the configuration "name" entry as the Python command to be executed to read the data.
- The "level" entry is not required for Python.

  For example:

```
field = [
  { name = "read_ascii_numpy.py data/python/fcst.txt FCST"; }
];
```

Option 2:

- On the command line, leave the path to the input gridded data as is.
- Set the configuration "file_type" entry to the constant PYTHON_NUMPY or PYTHON_XARRAY.
- Specify the configuration "name" entry as the Python command to be executed to read the data, but replace the input gridded data file with the constant MET_PYTHON_INPUT_ARG.
- The "level" entry is not required for Python.

  For example:

```
file_type = PYTHON_NUMPY;
field     = [
  { name = "read_ascii_numpy.py MET_PYTHON_INPUT_ARG FCST"; }
];
```

```
fcst = {
   censor_thresh = [];
   censor_val    = [];
   cnt_thresh    = [ NA ];
   cnt_logic     = UNION;
   wind_thresh   = [ NA ];
   wind_logic    = UNION;
   eclv_points   = 0.05;
   message_type  = [ "ADPSFC" ];
   init_time     = "20120619_12";
   valid_time    = "20120620_00";
   lead_time     = "12";

   field = [
      {
        name       = "APCP";
        level      = [ "A03" ];
        cat_thresh = [ >0.0, >=5.0 ];
      }
   ];
}
```

5.2.14 obs

The "obs" entry specifies the same type of information as "fcst", but for the observation data. It will often be set to the same things as "fcst", as shown in the example below. However, when comparing forecast and observation files of different format types, this entry will need to be set in a non-trivial way. The length of the "obs.field" array must match the length of the "fcst.field" array. For example:

```
obs = fcst;
```

or

```
fcst = {
  censor_thresh = [];
  censor_val    = [];
  cnt_thresh    = [ NA ];
  cnt_logic     = UNION;
  wind_thresh   = [ NA ];
  wind_logic    = UNION;

  field = [
     {
        name       = "PWAT";
        level      = [ "L0" ];
        cat_thresh = [ >2.5 ];
     }
   ];
}


obs = {
  censor_thresh = [];
  censor_val    = [];
  mpr_column    = [];
  mpr_thresh    = [];
  cnt_thresh    = [ NA ];
  cnt_logic     = UNION;
  wind_thresh   = [ NA ];
  wind_logic    = UNION;

  field = [
     {
        name       = "IWV";
        level      = [ "L0" ];
        cat_thresh = [ >25.0 ];
     }
   ];
}
```

5.2.14.1 message_type

The "message_type" entry is an array of point observation message types to be used. This only applies to the tools that verify against point observations. This may be specified once at the top-level "obs" dictionary or separately for each "field" array element. In the example shown above, this is specified in the "fcst" dictionary and copied to "obs".

Simplified vertical level matching logic is applied for surface message types. Observations for the following message types are assumed to be at the surface, as defined by the default message_type_group_map: ADPSFC, SFCSHP, MSONET

The "message_type" would be placed in the "field" array element if more than one "message_type" entry is desired within the config file. For example:

```
fcst = {
    censor_thresh = [];
    censor_val    = [];
    cnt_thresh    = [ NA ];
    cnt_logic     = UNION;
    wind_thresh   = [ NA ];
    wind_logic    = UNION;

    field = [
       {
         message_type = [ "ADPUPA" ];
         sid_inc      = [];
         sid_exc      = [];
         name         = "TMP";
         level        = [ "P250", "P500", "P700", "P850", "P1000" ];
         cat_thresh   = [ <=273.0 ];
       },
       {
         message_type = [ "ADPSFC" ];
         sid_inc      = [];
         sid_exc      = [ "KDEN", "KDET" ];
         name         = "TMP";
         level        = [ "Z2" ];
         cat_thresh   = [ <=273.0 ];
       }
    ];
  }
```

5.2.14.2 sid_inc and sid_exc

The "sid_inc" entry is an array of station ID groups indicating which station ID's should be included in the verification task. If specified, only those station ID's appearing in the list will be included. Note that filtering by station ID may also be accomplished using the "mask.sid" option. However, when using the "sid_inc" option, statistics are reported separately for each masking region.

The "sid_exc" entry is an array of station ID groups indicating which station ID's should be excluded from the verification task.

Each element in the "sid_inc" and "sid_exc" arrays is either the name of a single station ID or the full path to a station ID group file name. A station ID group file consists of a name for the group followed by a list of station ID's. All of the station ID's indicated will be concatenated into one long list of station ID's to be included or excluded.

As with "message_type" above, the "sid_inc" and "sid_exc" settings can be placed in the in the "field" array element to control which station ID's are included or excluded for each verification task.

```
obs = fcst;
```

5.2.15 climo_mean

The "climo_mean" dictionary specifies climatology mean data to be read by the Grid-Stat, Point-Stat, Ensemble-Stat, and Series-Analysis tools. It can be set inside the "fcst" and "obs" dictionaries to specify separate forecast and observation climatology data or once at the top-level configuration file context to use the same data for both. It consists of several entries defining the climatology file names and fields to be used.

5.2.15.1 file_names

The "file_names" entry specifies one or more file names containing the gridded climatology data to be used.

5.2.15.2 field

The "field" entry is an array of dictionaries, specified the same way as those in the "fcst" and "obs" dictionaries. If the array has length zero, not climatology data will be read and all climatology statistics will be written as missing data. Otherwise, the array length must match the length of "field" in the "fcst" and "obs" dictionaries.

5.2.15.3 regrid

The "regrid" dictionary defines how the climatology data should be regridded to the verification domain.

5.2.15.4 time_interp_method

The "time_interp_method" entry specifies how the climatology data should be interpolated in time to the forecast valid time:

- NEAREST for data closest in time
- UW_MEAN for average of data before and after
- DW_MEAN for linear interpolation in time of data before and after

5.2.15.5 day_interval

The "day_interval" entry is an integer specifying the spacing in days of the climatology data. Use 31 for monthly data or 1 for daily data. Use "NA" if the timing of the climatology data should not be checked.

5.2.15.6 hour_interval

The "hour_interval" entry is an integer specifying the spacing in hours of the climatology data for each day. This should be set between 0 and 24, with 6 and 12 being common choices. Use "NA" if the timing of the climatology data should not be checked.

Note: As of MET version 11.0.0, the "day_interval" and "hour_interval" entries replace the "match_month", "match_day", and "time_step" entries, which are now deprecated.

```
climo_mean = {

   file_name = [ "/path/to/climatological/mean/files" ];
   field     = [];

   regrid = {
      method     = NEAREST;
      width      = 1;
      vld_thresh = 0.5;
   }

   time_interp_method = DW_MEAN;
   day_interval       = 31;
   hour_interval      = 6;
}
```

5.2.16 climo_stdev

The "climo_stdev" dictionary specifies climatology standard deviation data to be read by the Grid-Stat, Point-Stat, Ensemble-Stat, and Series-Analysis tools. It can be set inside the "fcst" and "obs" dictionaries to specify separate forecast and observation climatology data or once at the top-level configuration file context to use the same data for both. The "climo_mean" and "climo_stdev" data define the climatological distribution for each grid point, assuming normality. These climatological distributions are used in two ways:

(1) To define climatological distribution percentiles thresholds (FCDP and OCDP) which can be used as categorical (cat_thresh), continuous (cnt_thresh), or wind speed (wind_thresh) thresholds.

(2) To subset matched pairs into climatological bins based on where the observation value falls within the observation climatological distribution. See the "climo_cdf" dictionary. Note that only the observation climatology data is used for this purpose, not the forecast climatology data.

This dictionary is identical to the "climo_mean" dictionary described above but points to files containing climatological standard deviation values rather than means. In the example below, this dictionary is set by copying over the "climo_mean" setting and then updating the "file_name" entry.

```
climo_stdev = climo_mean;
climo_stdev = {
   file_name = [ "/path/to/climatological/standard/deviation/files" ];
}
```

Prior to MET version 12.0.0, forecast climatological inputs were not supported. If the "climo_mean" and "climo_stdev" dictionaries are defined at the top-level configuration file context, the same data is used for both the forecast and observation climatologies. To specify separate forecast and observation climatologies, define "climo_mean" and "climo_stdev" inside the "fcst" and "obs" dictionaries, as shown below.

```
fcst = {
  field = [ ... ];
  climo_mean = { ... };
  climo_stdev = { ... };
}

obs = {
  field = [ ... ];
  climo_mean = { ... };
  climo_stdev = { ... };
}
```

5.2.17 climo_cdf

The "climo_cdf" dictionary specifies how the the observation climatological mean ("climo_mean") and standard deviation ("climo_stdev") data are used to evaluate model performance relative to where the observation value falls within the observation climatological distribution. It can be set inside the "obs" dictionary or at the top-level configuration file context. This dictionary consists of the following entries:

(1) The "cdf_bins" entry defines the climatological bins either as an integer or an array of floats between 0 and 1.

(2) The "center_bins" entry may be set to TRUE or FALSE.

(3) The "write_bins" entry may be set to TRUE or FALSE.

(4) The "direct_prob" entry may be set to TRUE or FALSE.

MET uses the observation climatological mean and standard deviation to construct a normal PDF at each observation location. The total area under the PDF is 1, and the climatological CDF value is computed as the area of the PDF to the left of the observation value. Since the CDF is a value between 0 and 1, the CDF bins must span that same range.

When "cdf_bins" is set to an array of floats, they explicitly define the climatological bins. The array must begin with 0.0 and end with 1.0. For example:

```
cdf_bins = [ 0.0, 0.10, 0.25, 0.75, 0.90, 1.0 ];
```

When "cdf_bins" is set to an integer, it defines the number of bins to be used. The "center_bins" flag indicates whether or not the bins should be centered on 0.5. An odd number of bins can be centered or uncentered while an even number of bins can only be uncentered. For example:

```
4 uncentered bins (cdf_bins = 4; center_bins = FALSE;) yields:
  0.0, 0.25, 0.50, 0.75, 1.0
5 uncentered bins (cdf_bins = 5; center_bins = FALSE;) yields:
  0.0, 0.2, 0.4, 0.6, 0.8, 0.9, 1.0
5   centered bins (cdf_bins = 5; center_bins = TRUE;) yields:
  0.0, 0.125, 0.375, 0.625, 0.875, 1.0
```

When multiple climatological bins are used for Point-Stat and Grid-Stat, statistics are computed separately for each bin, and the average of the statistics across those bins is written to the output. When "write_bins" is true, the statistics for each bin are also written to the output. The bin number is appended to the contents of the VX_MASK output column.

Setting the number of bins to 1 effectively disables this logic by grouping all pairs into a single climatological bin.

```
climo_cdf = {
   cdf_bins    = 11;    or an array of floats
   center_bins = TRUE;  or FALSE
   write_bins  = FALSE; or TRUE
   direct_prob = FALSE; or TRUE
}
```

5.2.18 climate_data

When specifying observation climatology data to evaluate probability forecasts, either supply a probabilistic observation "climo_mean" field or non-probabilistic "climo_mean" and "climo_stdev" fields from which a normal approximation of the observation climatological probabilities should be derived.

When the observation "climo_mean" is set to a probability field with a range of [0, 1] and "climo_stdev" is unset, the MET tools use the "climo_mean" probability values directly to compute Brier Skill Score (BSS).

When the observation "climo_mean" and "climo_stdev" are both set to non-probability fields, the MET tools use the mean, standard deviation, and observation event threshold to derive a normal approximation of the observation climatological probabilities.

The "direct_prob" option controls the derivation logic. When "direct_prob" is true, the climatological probability is computed directly from the climatological distribution at each point as the area to the left of the event threshold value. For greater-than or greater-than-or-equal-to thresholds, 1.0 minus the area is used. When "direct_prob" is false, the "cdf_bins" values are sampled from climatological distribution. The probability is computed as the proportion of those samples which meet the threshold criteria. In this way, the number of bins impacts the resolution of the climatological probabilities. These derived probability values are used to compute the climatological Brier Score and Brier Skill Score.

5.2.19 seeps_p1_thresh

The "seeps_p1_thresh" option controls the threshold of p1 (probability of being dry) values. The default setting is >=0.1&&<=0.85.

```
seeps_p1_thresh = >=0.1&&<=0.85;
```

5.2.20 seeps_grid_climo_name

The "seeps_grid_climo_name" option sets the location and the filename of the SEEPS climo file for the gridded data. The default setting is the empty string. It should be configured by the user. It can be overridden by the environment variable, MET_SEEPS_GRID_CLIMO_NAME.

```
seeps_grid_climo_name = "";
```

5.2.21 seeps_point_climo_name

The "seeps_point_climo_name" option controls the location and the filename of the SEEPS climo file for the point data. The default setting is the empty string which does not override the default location and name. It can be overridden by the environment variable, MET_SEEPS_POINT_CLIMO_NAME.

```
seeps_point_climo_name = "";
```

5.2.22 mask_missing_flag

The "mask_missing_flag" entry specifies how missing data should be handled in the Wavelet-Stat and MODE tools:

- NONE to perform no masking of missing data
- FCST to mask the forecast field with missing observation data
- OBS to mask the observation field with missing forecast data
- BOTH to mask both fields with missing data from the other

```
mask_missing_flag = BOTH;
```

5.2.23 obs_window

The "obs_window" entry is a dictionary specifying a beginning ("beg" entry) and ending ("end" entry) time offset values in seconds. It defines the time window over which observations are retained for scoring. These time offsets are defined relative to a reference time t, as [t+beg, t+end]. In PB2NC, the reference time is the PREPBUFR files center time. In Point-Stat and Ensemble-Stat, the reference time is the forecast valid time.

```
obs_window = {
   beg = -5400;
   end =  5400;
}
```

5.2.24 mask

The "mask" entry is a dictionary that specifies the verification masking regions to be used when computing statistics. Each mask defines a geographic extent, and any matched pairs falling inside that area will be used in the computation of statistics. Masking regions may be specified in the following ways:

5.2.24.1 grid

The "grid" entry is an array of named grids. It contains a comma-separated list of pre-defined NCEP grids over which to perform verification. An empty list indicates that no masking grids should be used. The standard NCEP grids are named "GNNN" where NNN indicates the three digit grid number. Supplying a value of "FULL" indicates that the verification should be performed over the entire grid on which the data resides. See: ON388 - TABLE B, GRID IDENTIFICATION (PDS Octet 7), MASTER LIST OF NCEP STORAGE GRIDS, GRIB Edition 1 (FM92). The "grid" entry can be the gridded data file defining grid.

5.2.24.2 poly

The "poly" entry contains a comma-separated list of files that define verification masking regions. These masking regions may be specified in three ways:

1. An ASCII polyline file containing lat/lon points defining the mask polygon.
2. The NetCDF output of the Gen-Vx-Mask tool.
3. Any gridded data file followed by a configuration string describing the data to be read and an optional threshold to be applied to that data.

These three options are described below:

- Option 1 - ASCII polyline file:

  If providing an ASCII file containing the lat/lon points defining the mask polygon, the file must contain a name for the region followed by the latitude (degrees north) and longitude (degrees east) for each vertex of the polygon. The values are separated by whitespace (e.g. spaces or newlines), and the first and last polygon points are connected. The general form is "poly_name lat1 lon1 lat2 lon2... latn lonn". Here is an example of a rectangle consisting of 4 points:

  Listing 5.1: ASCII Rectangle Polygon Mask

```
RECTANGLE
25  -120
55  -120
55   -70
25   -70
```

  Several masking polygons used by NCEP are predefined in the installed *share/met/poly* directory. Creating a new polygon is as simple as creating a text file with a name for the polygon followed by the lat/lon points which define its boundary. Adding a new masking polygon requires no code changes and no recompiling. Internally, the lat/lon polygon points are converted into x/y values in the grid. The lat/lon values for the observation points are also converted into x/y grid coordinates. The computations performed to check whether the observation point falls within the polygon defined is done in x/y grid space.

```
mask = { poly = [ "share/met/poly/CONUS.poly" ]; }
```

- Option 2 - Gen-Vx-Mask output:

  The NetCDF output of the gen_vx_mask tool. Please see Section 10 for more details.

```
mask = { poly = [ "/path/to/gen_vx_mask_output.nc" ]; }
```

- Option 3 - Any gridded data file:

  Any gridded data file that MET can read may be used to define a verification masking region. Users must specify a description of the field to be used from the input file and, optionally, may specify a threshold to be applied to that field. Once this threshold is applied, any grid point where the resulting field is 0, the mask is turned off. Any grid point where it is non-zero, the mask is turned on.

```
mask = { poly = [ "/path/to/sample.grib {name = \"TMP\"; level = \"Z2\";} >273" ]; }
```

Note: The syntax for the Option 3 is complicated since it includes quotes embedded within another quoted string. Any such embedded quotes must be escaped using a preceeding backslash character.

5.2.24.3 sid and llpnt

The "sid" entry is an array of strings which define groups of observation station ID's over which to compute statistics. Each station ID string can be followed by an optional numeric weight enclosed in parenethesis and used by the "point_weight_flag" configuration option. Each entry in the "sid" "array is either a filename or a comma-separated list.

- For an ASCII filename, the strings contained within it are whitespace-separated. The first string is the mask "name" and the remaining strings are the station ID's to be used.
- For a comma-separated list, optionally use a colon to specify a name. For "MY_LIST:SID1(WGT1),SID2(WGT2)", name = MY_LIST which consists of two station ID's (SID1 and SID2) and optional numeric weights (WGT1 and WGT2).
- For a comma-separated list of length one with no name specified, the mask "name" and value are both set to the single station ID string. For "SID1", name = SID1 and value = SID1.
- For a comma-separated list of length greater than one with no name specified, the name is set to MASK_SID and the values are the station ID's to be used. For "SID1,SID2", name = MASK_SID and values = SID1 and SID2.
- The "name" of the station ID mask is written to the VX_MASK column of the MET output files.

The "llpnt" entry is either a single dictionary or an array of dictionaries. Each dictionary contains three entries, the "name" for the masking region, "lat_thresh", and "lon_thresh". The latitude and longitude thresholds are applied directly to the point observation latitude and longitude values. Only observations whose latitude and longitude values meet this threshold criteria are used. A threshold set to "NA" always evaluates to true.

The masking logic for processing point observations in Point-Stat and Ensemble-Stat fall into two cateogries. The "sid" and "llpnt" options apply directly to the point observations. Only those observations for the specified station id's are included in the "sid" masks. Only those observations meeting the latitude and longitude threshold criteria are included in the "llpnt" masks.

The "grid" and "poly" mask options are applied to the grid points of the verification domain. Each grid point is determined to be inside or outside the masking region. When processing point observations, their latitude and longitude values are rounded to the nearest grid point of the verification domain. If the nearest grid point is inside the mask, that point observation is included in the mask.

```
mask = {
   grid    = [ "FULL" ];
   poly    = [ "MET_BASE/poly/LMV.poly",
               "MET_BASE/out/gen_vx_mask/CONUS_poly.nc",
```

(continues on next page)

(continued from previous page)

```
              "MET_BASE/sample_fcst/2005080700/wrfprs_ruc13_12.tm00_G212 \
              {name = \"TMP\"; level = \"Z2\";} >273"
            ];
   sid     = [ "CONUS.stations" ];
   llpnt   = [ { name       = "LAT30TO40";
                 lat_thresh = >=30&&<=40;
                 lon_thresh = NA; },
               { name       = "BOX";
                 lat_thresh = >=20&&<=40;
                 lon_thresh = >=-110&&<=-90; } ];
}
```

5.2.25 ci_alpha

The "ci_alpha" entry is an array of floats specifying the values for alpha to be used when computing confidence intervals. Values of alpha must be between 0 and 1. The confidence interval computed is 1 minus the alpha value. Therefore, an alpha value of 0.05 corresponds to a 95% confidence interval.

```
ci_alpha = [ 0.05, 0.10 ];
```

5.2.26 boot

The "boot" entry defines the parameters to be used in calculation of bootstrap confidence intervals. The interval variable indicates what method should be used for computing bootstrap confidence intervals:

5.2.26.1 interval

The "interval" entry specifies the confidence interval method:

- BCA for the BCa (bias-corrected percentile) interval method is highly accurate but computationally intensive.
- PCTILE uses the percentile method which is somewhat less accurate but more efficient.

5.2.26.2 rep_prop

The "rep_prop" entry specifies a proportion between 0 and 1 to define the replicate sample size to be used when computing percentile intervals. The replicate sample size is set to boot_rep_prop * n, where n is the number of raw data points.

When computing bootstrap confidence intervals over n sets of matched pairs, the size of the subsample, m, may be chosen less than or equal to the size of the sample, n. This variable defines the size of m as a proportion relative to the size of n. A value of 1 indicates that the size of the subsample, m, should be equal to the size of the sample, n.

5.2.26.3 n_rep

The "n_rep" entry defines the number of subsamples that should be taken when computing bootstrap confidence intervals. This variable should be set large enough so that when confidence intervals are computed multiple times for the same set of data, the intervals do not change much. Setting this variable to zero disables the computation of bootstrap confidence intervals, which may be necessary to run MET in realtime or near-realtime over large domains since bootstrapping is computationally expensive. Setting this variable to 1000 indicates that bootstrap confidence interval should be computed over 1000 subsamples of the matched pairs.

5.2.26.4 rng

The "rng" entry defines the random number generator to be used in the computation of bootstrap confidence intervals. Subsamples are chosen at random from the full set of matched pairs. The randomness is determined by the random number generator specified. Users should refer to detailed documentation of the GNU Scientific Library for a listing of the random number generators available for use.

5.2.26.5 seed

The "seed" entry may be set to a specific value to make the computation of bootstrap confidence intervals fully repeatable. When left empty the random number generator seed is chosen automatically which will lead to slightly different bootstrap confidence intervals being computed each time the data is run. Specifying a value here ensures that the bootstrap confidence intervals will be reproducable over multiple runs on the same computing platform.

```
boot = {
   interval = PCTILE;
   rep_prop = 1.0;
   n_rep    = 0;
   rng      = "mt19937";
   seed     = "";
}
```

5.2.27 interp

The "interp" entry is a dictionary that specifies what interpolation or smoothing (for the Grid-Stat tool) methods should be applied. This dictionary may include the following entries:

5.2.27.1 field

The "field" entry specifies to which field(s) the interpolation method should be applied. This does not apply when doing point verification with the Point-Stat or Ensemble-Stat tools:

- FCST to interpolate/smooth the forecast field.
- OBS to interpolate/smooth the observation field.
- BOTH to interpolate/smooth both the forecast and the observation.

5.2.27.2 vld_thresh

The "vld_thresh" entry specifies a number between 0 and 1. When performing interpolation over some neighborhood of points the ratio of the number of valid data points to the total number of points in the neighborhood is computed. If that ratio is less than this threshold, the matched pair is discarded. Setting this threshold to 1, which is the default, requires that the entire neighborhood must contain valid data. This variable will typically come into play only along the boundaries of the verification region chosen.

5.2.27.3 shape

The "shape" entry may be set to SQUARE or CIRCLE to specify the shape of the smoothing area.

5.2.27.4 type

The "type" entry is an array of dictionaries, each specifying one or more interpolation methods and widths. Interpolation is performed over an N by N box centered on each point, where N is the width specified. Each of these dictionaries must include:

5.2.27.4.1 type.width

The "width" entry is an array of integers to specify the size of the interpolation area. The area is either a square or circle containing the observation point. The width value specifies the width of the square or diameter of the circle. A width value of 1 is interpreted as the nearest neighbor model grid point to the observation point. For squares, a width of 2 defines a 2 x 2 box of grid points around the observation point (the 4 closest model grid points), while a width of 3 defines a 3 x 3 box of grid points around the observation point, and so on. For odd widths in grid-to-point comparisons (i.e. Point-Stat), the interpolation area is centered on the model grid point closest to the observation point. For grid-to-grid comparisons (i.e. Grid-Stat), the width must be odd.

5.2.27.4.2 type.method

The "method" entry is an array of interpolation procedures to be applied to the points in the box:

- MIN for the minimum value
- MAX for the maximum value
- MEDIAN for the median value
- UW_MEAN for the unweighted average value
- DW_MEAN for the distance-weighted average value where weight = distance^-2
- LS_FIT for a least-squares fit
- BILIN for bilinear interpolation (width = 2)
- NEAREST for the nearest grid point (width = 1)
- BEST for the value closest to the observation
- UPPER_LEFT for the upper left grid point (width = 1)
- UPPER_RIGHT for the upper right grid point (width = 1)
- LOWER_RIGHT for the lower right grid point (width = 1)
- LOWER_LEFT for the lower left grid point (width = 1)
- GAUSSIAN for the Gaussian kernel
- MAXGAUSS for the maximum value followed by a Gaussian smoother
- GEOG_MATCH for the nearest grid point where the land/sea mask and geography criteria are satisfied
- HIRA for all neighborhood points to define a spatial ensemble (only in Ensemble-Stat)

The BUDGET, FORCE, GAUSSIAN, and MAXGAUSS methods are not valid for interpolating to point locations. For grid-to-grid comparisons, the only valid smoothing methods are MIN, MAX, MEDIAN, UW_MEAN, and GAUSSIAN, and MAXGAUSS.

If multiple "method" and "width" options are specified, all possible permutations of their values are applied.

```
interp = {
   field      = BOTH;
   vld_thresh = 1.0;
   shape      = SQUARE;

   type = [
      {
         method = [ NEAREST ];
         width  = [ 1 ];
      }
```

(continues on next page)

(continued from previous page)

```
   ];
}
```

5.2.28 land_mask

The "land_mask" dictionary defines the land/sea mask field used when verifying at the surface. The "flag" entry enables/disables this logic. If the "file_name" entry is empty, the land/sea mask field is assumed to exist in the input forecast file. Otherwise, the specified file(s) are searched for the data specified in the "field" entry. The "regrid" settings specify how this field should be regridded to the verification domain. Lastly, the "thresh" entry is the threshold which defines land (threshold is true) and water (threshold is false).

The "message_type_group_map" dictionary entries for "LANDSF" and "WATERSF" specify comma-separated lists of message types whose observations exist on land or water, respectively. For point observations whose message type appears in the "LANDSF" entry, only interpolate using forecast grid points where land = TRUE. For point observations whose message type appears in the "WATERSF" entry, only interpolate using forecast grids points where land = FALSE. By default, "ADPSFC" and "MSONET" message types exist over land while the "SFCSHP" message type exists over water.

```
message_type_group_map = [
  ...
  { key = "LANDSF";  val = "ADPSFC,MSONET"; },
  { key = "WATERSF"; val = "SFCSHP";        },
  ...
];
```

The "topo_mask.flag", "topo_mask.use_obs_thresh", and "topo_mask.interp_fcst_thresh" The "land_mask.flag" entry may be set separately in each "obs.field" entry.

```
land_mask = {
   flag      = FALSE;
   file_name = [];
   field     = { name = "LAND"; level = "L0"; }
   regrid    = { method = NEAREST; width = 1; }
   thresh    = eq1;
}
```

5.2.29 topo_mask

The "topo_mask" dictionary defines the model topography field used when verifying at the surface. The flag entry enables/disables this logic. Only use point observations where the model topography minus station elevation elevation difference meets the "use_obs_thresh" threshold entry. For the observations kept, when interpolating forecast data to the observation location, only use forecast grid points where the topo minus station difference meets the "interp_fcst_thresh" threshold entry. If the "file_name" is left empty, the topography data is assumed to exist in the input forecast file(s). Otherwise, the specified file(s) are searched for the data specified in the "field" entry. The "regrid" settings specify how this field should be

regridded to the verification domain. The "interp" settings specify how this field should be interpolated to point observation locations.

The "message_type_group_map" dictionary entry for "SURFACE" specifies a comma-separated list of message types whose observations exist at the surface. The topography filtering logic is only applied to observations whose message type appears in this list. By default, the topography filtering logic is applied to observations for message types "ADPSFC", "SFCSHP", and "MSONET".

```
message_type_group_map = [
  ...
  { key = "SURFACE"; val = "ADPSFC,SFCSHP,MSONET"; },
  ...
];
```

The "topo_mask.flag", "topo_mask.use_obs_thresh", and "topo_mask.interp_fcst_thresh" entries can be set separately in each "obs.field" entry.

```
topo_mask = {
   flag               = FALSE;
   file_name          = [];
   field              = { name = "TOPO"; level = "L0"; }
   regrid             = { method = BILIN; width = 2; }
   interp             = { method = BILIN; width = 2; }
   use_obs_thresh     = ge-100&&le100;
   interp_fcst_thresh = ge-50&&le50;
}
```

5.2.30 lapse_rate_correction

The "lapse_rate_correction" dictionary defines how model topography data should be used to correct forecast or observation surface temperature values. The model topography data specified in the "topo_mask" dictionary is also used for this correction. The "apply_to" option can be set to "NONE" (default) to disable this correction logic, "FCST" to correct the forecast value, or "OBS" to correct the observation value, but not "BOTH". The "value" entry is a constant number that defines the lapse rate for correcting temperatures based on the difference of the model topography height and station elevation. By default, "value" is set to bad data and must be explicitly defined when "apply_to" is not set to NONE. The lapse rate correction fails when the model topography or observation height values are bad data and that matched pair is excluded from the computation of statistics.

The "message_type_group_map" dictionary entry for "LAPSERT" specifies a comma-separated list of message types for which this correction should be applied. By default, the correction logic is applied to observations for message types "ADPSFC" and "MSONET". If set to an empty list, the correction would be applied for all message types.

```
message_type_group_map = [
  ...
  { key = "LAPSERT"; val = "ADPSFC,MSONET"; },
```

(continues on next page)

(continued from previous page)

```
  ...
];
```

The environmental lapse rate values for 2-meter temperature and dewpoint temperature used by the World Meteorological Organization (WMO) can be found in "ConfigConstants".

```
WMO_TMP2M_LAPSE_RATE_K_per_M = 0.0065;
WMO_DPT2M_LAPSE_RATE_K_per_M = 0.0012;
```

Note that the lapse rate value depends on the units of temperature and elevation and the user is responsibile for defining it appropriately.

All "lapse_rate_correction" entries can be set separately in each "obs.field" entry.

```
lapse_rate_correction = {
  apply_to = NONE;
  value    = NA;
}
```

Run with verbosity level 8 or higher (`-v 8`) to see detailed log messages about lapse rate corrections.

5.2.31 msl_agl_conversion

The "msl_agl_conversion" dictionary defines how model topography heights and/or observation station elevations should be used to convert forecast and/or observation heights between mean-sea-level (MSL) and above-ground-level (AGL). The model topography data specified in the "topo_mask" dictionary is also used for this conversion. The "apply_to" option can be set to "NONE" (default) to disable this conversion logic, "FCST" to convert the forecast value, "OBS" to convert the observation value, or "BOTH" to convert both the forecast and observation values. The "apply_from" option can be set to "FCST" to convert using the model topography height, "OBS" to convert using the station elevation, or "BOTH" to convert the forecast value using the model topography height and convert the observation value using the station elevation. The "thresh" option specifies the valid range of values to be converted. Values not meeting this threshold criteria are left unchanged. The default threshold of "NA" always evaulates to true, but it can be set to avoid converting flag values. For example, set "thresh = ne99999;" to avoid converting a cloud base height flag value of 99999 which may indicate clear sky. The "msl_to_agl" entry is a boolean. When "TRUE", the elevation correction is subtracted from the height value to convert from MSL to AGL. When "FALSE", the elevation correction is added to the height value to convert from AGL to MSL. The MSL/AGL conversion fails when the model topography or observation height values are bad data and that matched pair is excluded from the computation of statistics.

The "message_type_group_map" dictionary entry for "MSLAGL" specifies a comma-separated list of message types for which this conversion should be applied. By default, the conversion logic is applied to observations for all message types since it is set to an empty list.

```
message_type_group_map = [
  ...
  { key = "MSLAGL"; val = ""; },
```

(continues on next page)

(continued from previous page)

```
  ...
];
```

All "msl_agl_conversion" entries can be set separately in each "obs.field" entry.

```
msl_agl_conversion = {
  apply_to   = NONE;
  apply_from = OBS;
  thresh     = NA;
  msl_to_agl = TRUE;
}
```

Run with verbosity level 8 or higher (`-v 8`) to see detailed log messages about MSL/AGL conversions.

5.2.32 hira

The "hira" entry is a dictionary that is very similar to the "interp" and "nbrhd" entries. It specifies information for applying the High Resolution Assessment (HiRA) verification logic in Point-Stat. HiRA is analogous to neighborhood verification but for point observations. The HiRA logic interprets the forecast values surrounding each point observation as an ensemble forecast. These ensemble values are processed in two ways. First, the ensemble continuous statistics (ECNT) and ranked probability score (RPS) line types are computed directly from the ensemble values. Second, for each categorical threshold specified, a fractional coverage value is computed as the ratio of the nearby forecast values that meet the threshold criteria. Point-Stat evaluates those fractional coverage values as if they were a probability forecast. When applying HiRA, users should enable the matched pair (MPR), probabilistic (PCT, PSTD, PJC, or PRC), or ensemble statistics (ECNT or PRS) line types in the output_flag dictionary. The number of probabilistic HiRA output lines is determined by the number of categorical forecast thresholds and HiRA neighborhood widths chosen. This dictionary may include the following entries:

5.2.32.1 flag

The "flag" entry is a boolean which toggles "hira" on (TRUE) and off (FALSE).

5.2.32.2 width

The "width" entry specifies the neighborhood size. Since HiRA applies to point observations, the width may be even or odd.

5.2.32.3 vld_thresh

The "vld_thresh" entry is as described above.

5.2.32.4 cov_thresh

The "cov_thresh" entry is an array of probabilistic thresholds used to populate the Nx2 probabilistic contingency table written to the PCT output line and used for computing probabilistic statistics.

5.2.32.5 shape

The "shape" entry defines the shape of the neighborhood. Valid values are `SQUARE` or `CIRCLE`.

5.2.32.6 prob_cat_thresh

The "prob_cat_thresh" entry defines the thresholds which define ensemble probabilities from which to compute the ranked probability score output. If left empty but climatology data is provided, the climo_cdf thresholds will be used instead. If left empty but no climatology data is provided, the obs.cat_thresh thresholds will be used instead.

```
hira = {
   flag            = FALSE;
   width           = [ 2, 3, 4, 5 ];
   vld_thresh      = 1.0;
   cov_thresh      = [ ==0.25 ];
   shape           = SQUARE;
   prob_cat_thresh = [];
}
```

5.2.33 gradient

The "gradient" entry is a dictionary which specifies the number and size of gradients to be computed and applies to both Grid-Stat and Series-Analysis. The "dx" and "dy" entries specify the size of the gradients in grid units in the X and Y dimensions, respectively. "dx" and "dy" are arrays of integers (positive or negative) which must have the same length, and the GRAD output line type will be computed separately for each entry. When computing gradients, the value at the (x, y) grid point is replaced by the value at the (x+dx, y+dy) grid point minus the value at (x, y). This configuration option may be set separately in each "obs.field" entry.

```
gradient = {
   dx = [ 1 ];
   dy = [ 1 ];
 }
```

5.2.34 output_flag

The "output_flag" entry is a dictionary that specifies what verification methods should be applied to the input data. Options exist for each output line type from the MET tools. Each line type may be set to one of:

- NONE to skip the corresponding verification method
- STAT to write the verification output only to the ".stat" output file
- BOTH to write to the ".stat" output file as well the optional "_type.txt" file, a more readable ASCII file sorted by line type.

```
output_flag = {
   fho    = NONE;  Forecast, Hit, Observation Rates
   ctc    = NONE;  Contingency Table Counts
   cts    = NONE;  Contingency Table Statistics
   mctc   = NONE;  Multi-category Contingency Table Counts
   mcts   = NONE;  Multi-category Contingency Table Statistics
   cnt    = NONE;  Continuous Statistics
   sl1l2  = NONE;  Scalar L1L2 Partial Sums
   sal1l2 = NONE;  Scalar Anomaly L1L2 Partial Sums when climatological data
                   is supplied
   vl1l2  = NONE;  Vector L1L2 Partial Sums
   val1l2 = NONE;  Vector Anomaly L1L2 Partial Sums when climatological data
                   is supplied
   pct    = NONE;  Contingency Table Counts for Probabilistic Forecasts
   pstd   = NONE;  Contingency Table Statistics for Probabilistic Forecasts
                   with Dichotomous outcomes
   pjc    = NONE;  Joint and Conditional Factorization for Probabilistic
                   Forecasts
   prc    = NONE;  Receiver Operating Characteristic for Probabilistic
                   Forecasts
   eclv   = NONE;  Economic Cost/Loss Value derived from CTC and PCT lines
   mpr    = NONE;  Matched Pair Data
   nbrctc = NONE;  Neighborhood Contingency Table Counts
   nbrcts = NONE;  Neighborhood Contingency Table Statistics
   nbrcnt = NONE;  Neighborhood Continuous Statistics
   isc    = NONE;  Intensity-Scale
   ecnt   = NONE;  Ensemble Continuous Statistics
   rps    = NONE;  Ranked Probability Score Statistics
   rhist  = NONE;  Rank Histogram
   phist  = NONE;  Probability Integral Transform Histogram
   orank  = NONE;  Observation Rank
   ssvar  = NONE;  Spread Skill Variance
   grad   = NONE;  Gradient statistics (S1 score)
}
```

5.2.35 nc_pairs_flag

The "nc_pairs_flag" can be set either to a boolean value or a dictionary in either Grid-Stat, Wavelet-Stat or MODE. The dictionary (with slightly different entries for the various tools . . . see the default config files) has individual boolean settings turning on or off the writing out of the various fields in the netcdf output file for the tool. Setting all dictionary entries to false means the netcdf file will not be generated.

"nc_pairs_flag" can also be set to a boolean value. In this case, a value of true means to just accept the default settings (which will turn on the output of all the different fields). A value of false means no netcdf output will be generated.

```
nc_pairs_flag = {
   latlon       = TRUE;
   raw          = TRUE;
   diff         = TRUE;
   climo        = TRUE;
   climo_cdp    = FALSE;
   weight       = FALSE;
   nbrhd        = FALSE;
   fourier      = FALSE;
   gradient     = FALSE;
   distance_map = FLASE;
   apply_mask   = TRUE;
}
```

5.2.36 nc_pairs_var_name

The "nc_pairs_var_name" entry specifies a string for each verification task in Grid-Stat. This string is parsed from each "obs.field" dictionary entry and is used to construct variable names for the NetCDF matched pairs output file. The default value of an empty string indicates that the "name" and "level" strings of the input data should be used. If the input data "level" string changes for each run of Grid-Stat, using this option to define a constant string may make downstream processing more convenient.

For example:

nc_pairs_var_name = "TMP";

```
nc_pairs_var_name = "";
```

5.2.37 nc_pairs_var_suffix

The "nc_pairs_var_suffix" entry is similar to the "nc_pairs_var_name" entry described above. It is also parsed from each "obs.field" dictionary entry. However, it defines a suffix to be appended to the output variable name. This enables the output variable names to be made unique. For example, when verifying height for multiple level types but all with the same level value, use this option to customize the output variable names.

For example:

nc_pairs_var_suffix = "TROPO"; (for the tropopause height)
nc_pairs_var_suffix = "FREEZING"; (for the freezing level height)

Note: Prior to MET version 9.0.0, this option was named "nc_pairs_var_str",' which is now deprecated.

```
nc_pairs_var_suffix = "";
```

5.2.38 ps_plot_flag

The "ps_plot_flag" entry is a boolean value for Wavelet-Stat and MODE indicating whether a PostScript plot should be generated summarizing the verification.

```
ps_plot_flag = TRUE;
```

5.2.39 grid_weight_flag

The "grid_weight_flag" specifies how grid weighting should be applied during the computation of contingency tables (CTC, MCTC, PCT, and NBRCTC), partial sums (SL1L2, SAL1L2, VL1L2, and VAL1L2), and statistics (CNT, CTS, MCTS, PSTD, PRC, PJC, ECLV, NBRCNT, and NBRCTS). It is meant to account for grid box area distortion and is often applied to global Lat/Lon grids. It is only applied for grid-to-grid verification in Grid-Stat and Ensemble-Stat and is not applied for grid-to-point verification, which is controlled by the "point_weight_flag" option. It can only be defined once at the highest level of config file context and applies to all verification tasks for that run.

Three grid weighting options are currently supported:

- NONE to disable grid weighting using a constant weight of 1.0 (default).
- COS_LAT to define the weight as the cosine of the grid point latitude. This an approximation for grid box area used by NCEP and WMO.
- AREA to define the weight as the true area of the grid box (km ^ 2).

If requested in the config file, the raw grid weights can be written to the NetCDF output from Grid-Stat and Ensemble-Stat.

When computing partial sums and continuous statistics, the weights are first normalized by dividing by the sum of the weights for the current masking region. When computing contingency tables and deriving statistics, each contingency table cell contains the sum of the weights of the matching grid points rather than the integer count of those grid points. Statistics are derived using these sums of weights rather than the raw counts.

When no grid weighting is requested (**NONE**), contingency tables are populated using a default constant weight of 1.0 and the corresponding cells are written to the output as integer counts for consistency with earlier versions of MET.

Note: The FHO line type is not compatible with grid weighting. If requested with grid weighting enabled, Grid-Stat prints a warning message and automatically disables the FHO line type. Users are advised to request the CTC line type instead.

```
grid_weight_flag = NONE;
```

5.2.40 point_weight_flag

The "point_weight_flag" is similar to the "grid_weight_flag", described above, but applies to grid-to-point verification in Point-Stat and Ensemble-Stat. It is not applied for grid-to-grid verification which is controlled by the "grid_weight_flag" option. It can only be defined once at the highest level of config file context and applies to all verification tasks for that run.

While only one point weighting option is currently supported, additional methods are planned for future versions:

- NONE to disable point weighting using a constant weight of 1.0 (default).
- SID to use the weights defined by the station ID masking configuration option, "mask.sid".

```
point_weight_flag = NONE;
```

5.2.41 hss_ec_value

The "hss_ec_value" entry is a floating point number used in the computation of the HSS_EC statistic in the CTS and MCTS line types. It specifies the expected correct (EC) rate by chance for multi-category contingency tables. If set to its default value of NA, it will automatically be replaced with 1.0 divided by the CTC or MCTC table dimension. For example, for a 2x2 CTC table, the default hss_ec_value is 1.0 / 2 = 0.5. For a 4x4 MCTC table, the default hss_ec_value is 1.0 / 4 = 0.25.

If set, it must greater than or equal to 0.0 and less than 1.0. A value of 0.0 produces an HSS_EC statistic equal to the Accuracy statistic.

```
hss_ec_value = NA;
```

5.2.42 rank_corr_flag

The "rank_corr_flag" entry is a boolean to indicate whether Kendall's Tau and Spearman's Rank Correlation Coefficients (in the CNT line type) should be computed. Computing them over large datasets is computationally intensive and slows down the runtime significantly.

```
rank_corr_flag = FALSE;
```

5.2.43 duplicate_flag

The "duplicate_flag" entry specifies how to handle duplicate point observations in Point-Stat and Ensemble-Stat:

- NONE to use all point observations (legacy behavior)
- UNIQUE only use a single observation if two or more observations match. Matching observations are determined if they contain identical latitude, longitude, level, elevation, and time information. They may contain different observation values or station IDs

The reporting mechanism for this feature can be activated by specifying a verbosity level of three or higher. The report will show information about where duplicates were detected and which observations were used in those cases.

```
duplicate_flag = NONE;
```

5.2.44 obs_summary

The "obs_summary" entry specifies how to compute statistics on observations that appear at a single location (lat,lon,level,elev) in Point-Stat and Ensemble-Stat. Eight techniques are currently supported:

- NONE to use all point observations (legacy behavior)
- NEAREST use only the observation that has the valid time closest to the forecast valid time
- MIN use only the observation that has the lowest value
- MAX use only the observation that has the highest value
- UW_MEAN compute an unweighted mean of the observations
- DW_MEAN compute a weighted mean of the observations based on the time of the observation
- MEDIAN use the median observation
- PERC use the Nth percentile observation where N = obs_perc_value

The reporting mechanism for this feature can be activated by specifying a verbosity level of three or higher. The report will show information about where duplicates were detected and which observations were used in those cases.

```
obs_summary = NONE;
```

5.2.45 obs_perc_value

Percentile value to use when obs_summary = PERC

```
obs_perc_value = 50;
```

5.2.46 obs_quality_inc

The "obs_quality_inc" entry specifies the quality flag values that are to be retained and used for verification. An empty list signifies that all point observations should be used, regardless of their quality flag value. The quality flag values will vary depending on the original source of the observations. The quality flag values to retain should be specified as an array of strings, even if the values themselves are numeric. Note "obs_quality_inc" replaces the older option "obs_quality".

```
obs_quality_inc = [ "1", "2", "3", "9" ];
```

5.2.47 obs_quality_exc

The "obs_quality_exc" entry specifies the quality flag values that are to be ignored and not used for verification. An empty list signifies that all point observations should be used, regardless of their quality flag value. The quality flag values will vary depending on the original source of the observations. The quality flag values to ignore should be specified as an array of strings, even if the values themselves are numeric.

```
obs_quality_exc = [ "1", "2", "3", "9" ];
```

5.2.48 met_data_dir

The "met_data_dir" entry specifies the location of the internal MET data sub-directory which contains data files used when generating plots. It should be set to the installed *share/met* directory so the MET tools can locate the static data files they need at run time.

```
met_data_dir = "MET_BASE";
```

5.2.49 many_plots

The "fcst_raw_plot" entry is a dictionary used by Wavelet-Stat and MODE containing colortable plotting information for the plotting of the raw forecast field:

5.2.49.1 color_table

The "color_table" entry specifies the location and name of the colortable file to be used.

5.2.49.2 plot_min and plot_max

The "plot_min" and "plot_max" entries specify the range of data values. If they are both set to 0, the MET tools will automatically rescale the colortable to the range of values present in the data. If they are not both set to 0, the MET tools will rescale the colortable using their values.

5.2.49.3 colorbar_flag

When applicable, the "colorbar_flag" enables the creation of a colorbar for this plot.

```
fcst_raw_plot = {
   color_table   = "MET_BASE/colortables/met_default.ctable";
   plot_min      = 0.0;
   plot_max      = 0.0;
   colorbar_flag = TRUE;
}
```

The "obs_raw_plot", "wvlt_plot", and "object_plot" entries are dictionaries similar to the "fcst_raw_plot" described above.

5.2.50 output_prefix

The "output_prefix" entry specifies a string to be included in the output file name. The MET statistics tools construct output file names that include the tool name and timing information. You can use this setting to modify the output file name and avoid naming conflicts for multiple runs of the same tool.

```
output_prefix  = "";
```

5.2.51 version

The "version" entry specifies the version number of the configuration file. The configuration file version number should match the version number of the MET code being run. This value should generally not be modified.

```
version = "VN.N";
```

5.2.52 time_summary

This feature was implemented to allow additional processing of observations with high temporal resolution. The "flag" entry toggles the "time_summary" on (TRUE) and off (FALSE). Obs may be summarized across the user specified time period defined by the "beg" and "end" entries. The "step" entry defines the time between intervals in seconds. The "width" entry specifies the summary interval in seconds. It may either be set as an integer number of seconds for a centered time interval or a dictionary with beginning and ending time offsets in seconds.

For example:

```
beg = "00";
end = "235959";
step = 300;
width = 600;
width = { beg = -300; end = 300; }
```

This example does a 10-minute time summary every 5 minutes throughout the day. The first interval will be from 23:55:00 the previous day through 00:04:59 of the current day. The second interval will be from 0:00:00 through 00:09:59. And so on.

The two "width" settings listed above are equivalent. Both define a centered 10-minute time interval. Use the "beg" and "end" entries to define uncentered time intervals. The following example requests observations for one hour prior:

```
width = { beg = -3600; end = 0; }
```

The summaries will only be calculated for the specified GRIB codes or observation variable ("obs_var") names.

When determining which observations fall within a time interval, data for the beginning timestamp is included while data for the ending timestamp is excluded. Users may need to adjust the "beg" and "end" settings in the "width" dictionary to include the desired observations in each time interval.

The supported time summaries are "min" (minimum), "max" (maximum), "range", "mean", "stdev" (standard deviation), "median", "sum", and "p##" (percentile, with the desired percentile value specified in place of ##).

The "vld_freq" and "vld_thresh" options may be used to require that a certain ratio of observations must be present and contain valid data within the time window in order for a summary value to be computed. The "vld_freq" entry defines the expected observation frequency in seconds. For example, when summarizing 1-minute data (vld_freq = 60) over a 30 minute time window, setting "vld_thresh = 0.5" requires that at least 15 of the 30 expected observations be present and valid for a summary value to be written. The default "vld_thresh = 0.0" setting will skip over this logic.

When using the "sum" option, users should specify "vld_thresh = 1.0" to avoid missing data values from affecting the resulting sum value.

The variable names are saved to NetCDF file if they are given instead of grib_codes which are not available for non GRIB input. The "obs_var" option was added and works like "grib_code" option (string value VS. int value). They are inclusive (union). All variables are included if both options are empty. Note: grib_code 11 is equivalent to obs_var "TMP".

```
time_summary = {
  flag = FALSE;
  beg = "000000";
  end = "235959";
  step = 300;
  width = 600;
  width = { beg = -300; end = 300; }
  grib_code = [ 11, 204, 211 ];
  obs_var   = [];
  type = [ "min", "max", "range", "mean", "stdev", "median", "p80" ];
  vld_freq = 0;
  vld_thresh = 0.0;
}
```

5.3 Settings Specific to Individual Tools

5.3.1 GenEnsProdConfig_default

5.3.1.1 ens

The "ens" entry is a dictionary that specifies the fields for which ensemble products should be generated. This is very similar to the "fcst" and "obs" entries. This dictionary may include the following entries:

5.3.1.1.1 ens.censor_thresh and ens.censor_val

The "censor_thresh" and "censor_val" entries are described above.

5.3.1.1.2 ens.ens_thresh

The "ens_thresh" entry specifies a proportion between 0 and 1 to define the required ratio of valid input ensemble member files. If the ratio of valid input ensemble files to expected ones is too low, the tool will error out.

5.3.1.1.3 ens.vld_thresh

The "vld_thresh" entry specifies a proportion between 0 and 1 to define the required ratio of valid data points. When computing ensemble products, if the ratio of valid data values is too low, the ensemble product will be set to bad data for that point.

5.3.1.1.4 ens.field

The "field" entry is as described above. However, in this case, the cat_thresh entry is used for calculating probabilities of exceeding the given threshold. In the default shown below, the probability of accumulated precipitation > 0.0 mm and > 5.0 mm will be calculated from the member accumulated precipitation fields and stored as an ensemble field.

```
ens = {
   censor_thresh = [];
   censor_val    = [];
   ens_thresh    = 1.0;
   vld_thresh    = 1.0;

   field = [
      {
         name       = "APCP";
         level      = "A03";
         cat_thresh = [ >0.0, >=5.0 ];
      }
   ];
}
```

5.3.1.2 nbrhd_prob

The nbrhd_prob dictionary defines the neighborhoods used to compute NEP and NMEP output. The neighborhood shape is a SQUARE or CIRCLE centered on the current point, and the width array specifies the width of the square or diameter of the circle as an odd integer. The vld_thresh entry is a number between 0 and 1 specifying the required ratio of valid data in the neighborhood for an output value to be computed.

If ensemble_flag.nep is set to TRUE, NEP output is created for each combination of the categorical threshold (cat_thresh) and neighborhood width specified.

```
nbrhd_prob = {
   width      = [ 5 ];
   shape      = CIRCLE;
   vld_thresh = 0.0;
}
```

5.3.1.3 nmep_smooth

Similar to the interp dictionary, the nmep_smooth dictionary includes a type array of dictionaries to define one or more methods for smoothing the NMEP data. Setting the interpolation method to nearest neighbor (NEAREST) effectively disables this smoothing step.

If ensemble_flag.nmep is set to TRUE, NMEP output is created for each combination of the categorical threshold (cat_thresh), neighborhood width (nbrhd_prob.width), and smoothing method (nmep_smooth.type) specified.

```
nmep_smooth = {
   vld_thresh      = 0.0;
   shape           = CIRCLE;
   gaussian_dx     = 81.27;
   gaussian_radius = 120;
   type = [
      {
         method = GAUSSIAN;
         width  = 1;
      }
   ];
}
```

5.3.1.4 ensemble_flag

The "ensemble_flag" entry is a dictionary of boolean value indicating which ensemble products should be generated:

- "latlon" for a grid of the Latitude and Longitude fields
- "mean" for the simple ensemble mean
- "stdev" for the ensemble standard deviation
- "minus" for the mean minus one standard deviation
- "plus" for the mean plus one standard deviation
- "min" for the ensemble minimum
- "max" for the ensemble maximum
- "range" for the range of ensemble values
- "vld_count" for the number of valid ensemble members
- "frequency" for the ensemble relative frequency meeting a threshold
- "nep" for the neighborhood ensemble probability
- "nmep" for the neighborhood maximum ensemble probability
- "rank" to write the rank for the gridded observation field to separate NetCDF output file.

- "weight" to write the grid weights specified in grid_weight_flag to the rank NetCDF output file.

```
ensemble_flag = {
  latlon     = TRUE;
   mean      = TRUE;
   stdev     = TRUE;
   minus     = TRUE;
   plus      = TRUE;
   min       = TRUE;
   max       = TRUE;
   range     = TRUE;
   vld_count = TRUE;
   frequency = TRUE;
   nep       = FALSE;
   nmep      = FALSE;
   rank      = TRUE;
   weight    = FALSE;
}
```

5.3.2 EnsembleStatConfig_default

5.3.2.1 fcst, obs

The fcst and obs entries define the fields for which Ensemble-Stat should compute rank histograms, probability integral transform histograms, spread-skill variance, relative position histograms, economic value, and other statistics.

5.3.2.1.1 obs.ens_ssvar_bin_size

The "ens_ssvar_bin_size" entry sets the width of the variance bins. Smaller bin sizes provide the user with more flexibility in how data are binned during analysis. The actual variance of the ensemble data will determine the number of bins written to the SSVAR output lines.

5.3.2.1.2 obs.ens_phist_bin_size

The "ens_phist_bin_size" is set to a value between 0 and 1. The number of bins for the probability integral transform histogram in the PHIST line type is defined as the ceiling of 1.0 / ens_phist_bin_size. For example, a bin size of 0.05 results in 20 PHIST bins.

5.3.2.1.3 fcst.prob_cat_thresh and obs.prob_cat_thresh

The "prob_cat_thresh" entry is an array of thresholds to be applied in the computation of the ranked probability score. If left empty, but climatology data is provided, the climo_cdf thresholds will be used instead.

```
fcst = {
   message_type       = [ "ADPUPA" ];
   ens_ssvar_bin_size = 1;
   ens_phist_bin_size = 0.05;
   prob_cat_thresh    = [];

   field = [
      {
         name  = "APCP";
         level = [ "A03" ];
      }
   ];
}
```

5.3.2.2 nc_var_str

The "nc_var_str" entry specifies a string for each Ensemble-Stat verification task. It is parsed from each "obs.field" dictionary entry and is used to customize the variable names written to the NetCDF output file. The default is an empty string, meaning that no customization is applied to the output variable names. When the Ensemble-Stat config file contains two fields with the same name and level value, this entry is used to make the resulting variable names unique. e.g. nc_var_str = "MIN";

```
nc_var_str = "";
```

5.3.2.3 obs_thresh

The "obs_thresh" entry is an array of thresholds for filtering observation values prior to applying ensemble verification logic. They specify the values to be included in the verification, not excluded. The default setting of NA, which always evaluates to true, means that all observations should be used. Verification output will be computed separately for each threshold specified. This option may be set separately for each obs.field entry.

```
obs_thresh = [ NA ];
```

5.3.2.4 skip_const

Setting "skip_const" to true tells Ensemble-Stat to exclude pairs where all the ensemble members and the observation have a constant value. For example, exclude points with zero precipitation amounts from all output line types. This option may be set separately for each obs.field entry. When set to false, constant points are included and the observation rank is chosen at random.

```
skip_const = FALSE;
```

5.3.2.5 obs_error

Observation error options:

5.3.2.5.1 obs_error.flag

The "flag" entry toggles the observation error logic on (TRUE) and off (FALSE). When flag is TRUE, random observation error perturbations are applied to the ensemble member values. No perturbation is applied to the observation values but the bias scale and offset values, if specified, are applied.

5.3.2.5.2 obs_error.dist_type

Set "dist_type" to NONE to use the observation error table instead. May be set separately in each "obs.field" entry. The obs_error dictionary controls how observation error information should be handled. Observation error information can either be specified directly in the configuration file or by parsing information from an external table file. By default, the *MET_BASE/data/table_files/obs_error_table.txt* file is read but this may be overridden by setting the $MET_OBS_ERROR_TABLE environment variable at runtime.

The "dist_type" entry may be set to NONE, NORMAL, EXPONENTIAL, CHISQUARED, GAMMA, UNIFORM, or BETA. The default value of NONE indicates that the observation error table file should be used rather than the configuration file settings.

5.3.2.5.3 obs_error.dist_parm

The "dist_parm" entry is an array of length 1 or 2 specifying the parameters for the distribution selected in dist_type. The NORMAL, EXPONENTIAL, and CHISQUARED distributions are defined by a single parameter. The GAMMA, UNIFORM, and BETA distributions are defined by two parameters. See the GNU Scientific Library Reference Manual for more information on these distributions.

5.3.2.5.4 obs_error.inst_bias_scale and obs_error.inst_bias_offset

The "inst_bias_scale" and "inst_bias_offset" entries specify bias scale and offset values that should be applied to observation values prior to perturbing them. These entries enable bias-correction on the fly.

Defining the observation error information in the configuration file is convenient but limited. If defined this way, the random perturbations for all points in the current verification task are drawn from the same distribution. Specifying an observation error table file instead (by setting dist_type = NONE;) provides much finer control, enabling the user to define observation error distribution information and bias-correction logic separately for each observation variable name, message type, PREPBUFR report type, input report type, instrument type, station ID, range of heights, range of pressure levels, and range of values.

```
obs_error = {
   flag             = FALSE;   TRUE or FALSE
   dist_type        = NONE;    Distribution type
   dist_parm        = [];      Distribution parameters
   inst_bias_scale  = 1.0;     Instrument bias scale adjustment
   inst_bias_offset = 0.0;     Instrument bias offset adjustment
   min              = NA;      Valid range of data
   max              = NA;
}
```

5.3.2.6 rng

See: Random Number Generator Performance used for random assignment of ranks when they are tied.

```
rng = {
   type = "mt19937";
   seed = "";
}
```

5.3.3 MODEAnalysisConfig_default

MODE line options are used to create filters that determine which MODE output lines are read in and processed. The MODE line options are numerous. They fall into seven categories: toggles, multiple set string options, multiple set integer options, integer max/min options, date/time max/min options, floating-point max/min options, and miscellaneous options. **In order to be applied, the options must be uncommented (i.e. remove the "//" marks) before running.** These options are described in subsequent sections. Please note that this configuration file is processed differently than the other config files.

5.3.3.1 Toggles

The MODE line options described in this section are shown in pairs. These toggles represent parameters that can have only one (or none) of two values. Any of these toggles may be left unspecified. However, if neither option for toggle is indicated, the analysis will produce results that combine data from both toggles. This may produce unintended results.

This toggle indicates whether forecast or observed lines should be used for analysis.

```
fcst      = FALSE;
obs       = FALSE;
```

This toggle indicates whether single object or object pair lines should be used.

```
single    = FALSE;
pair      = FALSE;
```

This toggle indicates whether simple object or object cluster object lines should be used.

```
simple    = FALSE;
cluster   = FALSE;
```

This toggle indicates whether matched or unmatched object lines should be used.

```
matched   = FALSE;
unmatched = FALSE;
```

5.3.3.2 Multiple-set String Options

The following options set various string attributes. They can be set multiple times on the command line but must be separated by spaces. Each of these options must be indicated as a string. String values that include spaces may be used by enclosing the string in quotation marks.

This options specifies which model to use

```
// model    = [];
```

These two options specify thresholds for forecast and observations objects to be used in the analysis, respectively.

```
// fcst_thr = [];
// obs_thr  = [];
```

These options indicate the names of variables to be used in the analysis for forecast and observed fields.

```
// fcst_var = [];
// obs_var = [];
```

These options indicate vertical levels for forecast and observed fields to be used in the analysis.

```
// fcst_lev = [];
// obs_lev = [];
```

5.3.3.3 Multiple-set Integer Options

The following options set various integer attributes. Each of the following options may only be indicated as an integer.

These options are integers of the form HH[MMSS] specifying the lead_time.

```
// fcst_lead       = [];
//obs_lead       = [];
```

These options are integers of the form HH[MMSS] specifying the valid hour.

```
// fcst_valid_hour = [];
// obs_valid_hour = [];
```

These options are integers of the form HH[MMSS] specifying the model initialization hour.

```
// fcst_init_hour  = [];
// obs_init_hour  = [];
```

These options are integers of the form HHMMSS specifying the accumulation time.

```
// fcst_accum      = [];
// obs_accum      = [];
```

These options indicate the convolution radius used for forecast of observed objects, respectively.

```
// fcst_rad        = [];
// obs_rad        = [];
```

5.3.3.4 Integer max/min Options

These options set limits on various integer attributes. Leaving a maximum value unset means no upper limit is imposed on the value of the attribute. The option works similarly for minimum values.

These options are used to indicate minimum/maximum values for the area attribute to be used in the analysis.

```
// area_min             = 0;
// area_max             = 0;
```

These options are used to indicate minimum/maximum values accepted for the area thresh. The area thresh is the area of the raw field inside the object that meets the threshold criteria.

```
// area_thresh_min       = 0;
// area_thresh_max       = 0;
```

These options refer to the minimum/maximum values accepted for the intersection area attribute.

```
// intersection_area_min = 0;
// intersection_area_max = 0;
```

These options refer to the minimum/maximum union area values accepted for analysis.

```
// union_area_min        = 0;
// union_area_max        = 0;
```

These options refer to the minimum/maximum values for symmetric difference for objects to be used in the analysis.

```
// symmetric_diff_min    = 0;
// symmetric_diff_max    = 0;
```

5.3.3.5 Date/time max/min Options

These options set limits on various date/time attributes. The values can be specified in one of three ways: First, the options may be indicated by a string of the form YYYMMDD_HHMMSS. This specifies a complete calendar date and time. Second, they may be indicated by a string of the form YYYYMMMDD_HH. Here, the minutes and seconds are assumed to be zero. The third way of indicating date/time attributes is by a string of the form YYYMMDD. Here, hours, minutes, and seconds are assumed to be zero.

These options indicate minimum/maximum values for the forecast valid time.

```
// fcst_valid_min = "";
// fcst_valid_max = "";
```

These options indicate minimum/maximum values for the observation valid time.

```
// obs_valid_min  = "";
// obs_valid_max  = "";
```

These options indicate minimum/maximum values for the forecast initialization time.

```
// fcst_init_min  = "";
// fcst_init_max  = "";
```

These options indicate minimum/maximum values for the observation initialization time.

```
// obs_init_min   = "";
// obs_init_max   = "";
```

5.3.3.6 Floating-point max/min Options

Setting limits on various floating-point attributes. One may specify these as integers (i.e., without a decimal point), if desired. The following pairs of options indicate minimum and maximum values for each MODE attribute that can be described as a floating- point number. Please refer to *Attributes* (page 356) in the MET User's Guide for a description of these attributes.

```
// centroid_x_min                = 0.0;
// centroid_x_max                = 0.0;

// centroid_y_min                = 0.0;
// centroid_y_max                = 0.0;

// centroid_lat_min              = 0.0;
// centroid_lat_max              = 0.0;

// centroid_lon_min              = 0.0;
// centroid_lon_max              = 0.0;

// axis_ang_min                  = 0.0;
// axis_ang_max                  = 0.0;

// length_min                    = 0.0;
// length_max                    = 0.0;

// width_min                     = 0.0;
// width_max                     = 0.0;

// aspect_ratio_min              = 0.0;
// aspect_ratio_max              = 0.0;

// curvature_min                 = 0.0;
// curvature_max                 = 0.0;

// curvature_x_min               = 0.0;
// curvature_x_max               = 0.0;

// curvature_y_min               = 0.0;
// curvature_y_max               = 0.0;

// complexity_min                = 0.0;
// complexity_max                = 0.0;

// intensity_10_min              = 0.0;
// intensity_10_max              = 0.0;

// intensity_25_min              = 0.0;
```

(continues on next page)

(continued from previous page)

```
// intensity_25_max               = 0.0;

// intensity_50_min               = 0.0;
// intensity_50_max               = 0.0;

// intensity_75_min               = 0.0;
// intensity_75_max               = 0.0;

// intensity_90_min               = 0.0;
// intensity_90_max               = 0.0;

// intensity_user_min             = 0.0;
// intensity_user_max             = 0.0;

// intensity_sum_min              = 0.0;
// intensity_sum_max              = 0.0;

// centroid_dist_min              = 0.0;
// centroid_dist_max              = 0.0;

// boundary_dist_min              = 0.0;
// boundary_dist_max              = 0.0;

// convex_hull_dist_min           = 0.0;
// convex_hull_dist_max           = 0.0;

// angle_diff_min                 = 0.0;
// angle_diff_max                 = 0.0;

// area_ratio_min                 = 0.0;
// area_ratio_max                 = 0.0;

// intersection_over_area_min     = 0.0;
// intersection_over_area_max     = 0.0;

// complexity_ratio_min           = 0.0;
// complexity_ratio_max           = 0.0;

// percentile_intensity_ratio_min = 0.0;
// percentile_intensity_ratio_max = 0.0;

// interest_min                   = 0.0;
// interest_max                   = 0.0;
```

5.3.4 MODEConfig_default

5.3.4.1 quilt

The "quilt" entry is a boolean to indicate whether all permutations of convolution radii and thresholds should be run. If set to false, the number of forecast and observation convolution radii and thresholds must all match. One configuration of MODE will be run for each group of settings in those lists. If set to true, the number of forecast and observation convolution radii must match and the number of forecast and observation convolution thresholds must match. For N radii and M thresholds, NxM configurations of MODE will be run.

```
quilt = false;
```

5.3.4.2 fcst, obs

The object definition settings for MODE are contained within the "fcst" and "obs" entries:

5.3.4.2.1 fcst/obs.censor_thresh and fcst/obs.censor_val

The "censor_thresh" and "censor_val" entries are described above. The entries replace the previously supported "raw_thresh" entry.

5.3.4.2.2 fcst/obs.conv_radius

The "conv_radius" entry specifies the convolution radius in grid squares. The larger the convolution radius, the smoother the objects. Multiple convolution radii may be specified as an array. For example:

```
conv_radius = [ 5, 10, 15 ];
```

5.3.4.2.3 fcst/obs.conv_thresh

The "conv_thresh" entry specifies the convolution threshold used to define MODE objects. The lower the threshold, the larger the objects. Multiple convolution thresholds may be specified as an array. For example:

```
conv_thresh = [ >=5.0, >=10.0, >=15.0 ];
```

5.3.4.2.4 fcst/obs.vld_thresh

The "vld_thresh" entry is described above.

5.3.4.2.5 fcst/obs.filter_attr_name and fcst/obs.filter_attr_thresh

The "filter_attr_name" and "filter_attr_thresh" entries are arrays of the same length which specify object filtering criteria. By default, no object filtering criteria is defined.

The "filter_attr_name" entry is an array of strings specifying the MODE output header column names for the object attributes of interest, such as "AREA", "LENGTH", "WIDTH", and "INTENSITY_50". In addition, "ASPECT_RATIO" specifies the aspect ratio (width/length), "INTENSITY_101" specifies the mean intensity value, and "INTENSITY_102" specifies the sum of the intensity values.

The "filter_attr_thresh" entry is an array of thresholds for the object attributes. Any simple objects not meeting all of these filtering criteria are discarded.

Note that the "area_thresh" and "inten_perc_thresh" entries form earlier versions of MODE are replaced by these options and are now deprecated.

5.3.4.2.6 fcst/obs.merge_thresh

The "merge_thresh" entry specifies a lower convolution threshold used when the double-threshold merging method is applied. The number of merge thresholds must match the number of convolution thresholds. Multiple merge thresholds may be specified as an array. For example:

```
merge_thresh = [ >=1.0, >=2.0, >=3.0 ];
```

5.3.4.2.7 fcst/obs.merge_flag

The "merge_flag" entry specifies the merging methods to be applied:

- NONE for no merging
- THRESH for the double-threshold merging method. Merge objects that would be part of the same object at the lower threshold.
- ENGINE for the fuzzy logic approach comparing the field to itself
- BOTH for both the double-threshold and engine merging methods

```
fcst = {
   field = {
      name  = "APCP";
      level = "A03";
   }
```

(continues on next page)

(continued from previous page)

```
   censor_thresh      = [];
   censor_val         = [];
   conv_radius        = 60.0/grid_res; in grid squares
   conv_thresh        = >=5.0;
   vld_thresh         = 0.5;
   filter_attr_name   = [];
   filter_attr_thresh = [];
   merge_thresh       = >=1.25;
   merge_flag         = THRESH;
}
```

5.3.4.3 grid_res

The "grid_res" entry is the nominal spacing for each grid square in kilometers. The variable is not used directly in the code, but subsequent variables in the configuration files are defined in terms of it. Therefore, setting the appropriately will help ensure that appropriate default values are used for these variables.

```
grid_res = 4;
```

5.3.4.4 match_flag

The "match_flag" entry specifies the matching method to be applied:

- NONE for no matching between forecast and observation objects
- MERGE_BOTH for matching allowing additional merging in both fields. If two objects in one field match the same object in the other field, those two objects are merged.
- MERGE_FCST for matching allowing only additional forecast merging
- NO_MERGE for matching with no additional merging in either field

```
match_flag = MERGE_BOTH;
```

5.3.4.5 max_centroid_dist

The "max_centroid_dist" entry specifies the maximum allowable distance in grid squares between the centroids of objects for them to be compared. Setting this to a reasonable value speeds up the runtime enabling MODE to skip unreasonable object comparisons.

```
max_centroid_dist = 800.0/grid_res;
```

5.3.4.6 weight

The weight variables control how much weight is assigned to each pairwise attribute when computing a total interest value for object pairs. The weights need not sum to any particular value but must be non-negative. When the total interest value is computed, the weighted sum is normalized by the sum of the weights listed.

```
weight = {
   centroid_dist    = 2.0;
   boundary_dist    = 4.0;
   convex_hull_dist = 0.0;
   angle_diff       = 1.0;
   area_ratio       = 1.0;
   int_area_ratio   = 2.0;
   complexity_ratio = 0.0;
   inten_perc_ratio = 0.0;
   inten_perc_value = 50;
}
```

5.3.4.7 interest_function

The set of interest function variables listed define which values are of interest for each pairwise attribute measured. The interest functions may be defined as a piecewise linear function or as an algebraic expression. A piecewise linear function is defined by specifying the corner points of its graph. An algebraic function may be defined in terms of several built-in mathematical functions.

```
interest_function = {

   centroid_dist = (
      (            0.0, 1.0 )
      (  60.0/grid_res, 1.0 )
      ( 600.0/grid_res, 0.0 )
   );

   boundary_dist = (
      (            0.0, 1.0 )
      ( 400.0/grid_res, 0.0 )
   );

   convex_hull_dist = (
      (            0.0, 1.0 )
      ( 400.0/grid_res, 0.0 )
   );

   angle_diff = (
      (  0.0, 1.0 )
      ( 30.0, 1.0 )
```

(continues on next page)

(continued from previous page)

```
      ( 90.0, 0.0 )
   );

   corner   = 0.8;
   ratio_if = (
      (    0.0, 0.0 )
      ( corner, 1.0 )
      (    1.0, 1.0 )
   );

   area_ratio = ratio_if;

   int_area_ratio = (
      ( 0.00, 0.00 )
      ( 0.10, 0.50 )
      ( 0.25, 1.00 )
      ( 1.00, 1.00 )
   );

   complexity_ratio = ratio_if;

   inten_perc_ratio = ratio_if;
}
```

5.3.4.8 total_interest_thresh

The total_interest_thresh variable should be set between 0 and 1. This threshold is applied to the total interest values computed for each pair of objects and is used in determining matches.

```
total_interest_thresh = 0.7;
```

5.3.4.9 print_interest_thresh

The print_interest_thresh variable determines which pairs of object attributes will be written to the output object attribute ASCII file. The user may choose to set the print_interest_thresh to the same value as the total_interest_thresh, meaning that only object pairs that actually match are written to the output file. When set to zero, all object pair attributes will be written as long as the distance between the object centroids is less than the max_centroid_dist variable.

```
print_interest_thresh = 0.0;
```

5.3.4.10 plot_valid_flag

When applied, the plot_valid_flag variable indicates that only the region containing valid data after masking is applied should be plotted. TRUE indicates the entire domain should be plotted; FALSE indicates only the region containing valid data after masking should be plotted.

```
plot_valid_flag = FALSE;
```

5.3.4.11 plot_gcarc_flag

When applied, the plot_gcarc_flag variable indicates that the edges of polylines should be plotted using great circle arcs as opposed to straight lines in the grid.

```
plot_gcarc_flag = FALSE;
```

5.3.4.12 ct_stats_flag

The ct_stats_flag can be set to TRUE or FALSE to produce additional output, in the form of contingency table counts and statistics.

```
ct_stats_flag = TRUE;
```

5.3.4.13 shift_right

When MODE is run on global grids, this parameter specifies how many grid squares to shift the grid to the right. MODE does not currently connect objects from one side of a global grid to the other, potentially causing objects straddling that longitude to be cut in half. Shifting the grid by some amount enables the user to control where that longitude cut line occurs. This option provides a very specialized case of automated regridding. The much more flexible "regrid" option may be used instead.

```
shift_right = 0;
```

5.3.5 PB2NCConfig_default

The PB2NC tool filters out observations from PREPBUFR or BUFR files using the following criteria:

(1) by message type: supply a list of PREPBUFR message types to retain

(2) by station id: supply a list of observation stations to retain

(3) by valid time: supply the beginning and ending time offset values in the obs_window entry described above.

(4) by location: use the "mask" entry described below to supply either an NCEP masking grid, a masking lat/lon polygon or a file to a mask lat/lon polygon

(5) by elevation: supply min/max elevation values

(6) by report type: supply a list of report types to retain using pb_report_type and in_report_type entries described below

(7) by instrument type: supply a list of instrument type to retain

(8) by vertical level: supply beg/end vertical levels using the level_range entry described below

(9) by variable type: supply a list of observation variable types to retain using the obs_bufr_var entry described below

(10) by quality mark: supply a quality mark threshold

(11) Flag to retain values for all quality marks, or just the first quality mark (highest): use the event_stack_flag described below

(12) by data level category: supply a list of category types to retain.

0 - Surface level (mass reports only)

1 - Mandatory level (upper-air profile reports)

2 - Significant temperature level (upper-air profile reports)

2 - Significant temperature and winds-by-pressure level (future combined mass and wind upper-air reports)

3 - Winds-by-pressure level (upper-air profile reports)

4 - Winds-by-height level (upper-air profile reports)

5 - Tropopause level (upper-air profile reports)

6 - Reports on a single level (e.g., aircraft, satellite-wind, surface wind, precipitable water retrievals, etc.)

7 - Auxiliary levels generated via interpolation from spanning levels (upper-air profile reports)

5.3.5.1 message_type

In the PB2NC tool, the "message_type" entry is an array of message types to be retained. An empty list indicates that all should be retained.

List of valid message types:
"ADPUPA", "AIRCAR", "AIRCFT", "ADPSFC", "ERS1DA", "GOESND", "GPSIPW",
"MSONET", "PROFLR", "QKSWND", "RASSDA", "SATEMP",
"SATWND", "SFCBOG", "SFCSHP", "SPSSMI", "SYNDAT", "VADWND"

For example:

message_type[] = ["ADPUPA", "AIRCAR"];

Current Table A Entries in PREPBUFR mnemonic table

```
message_type = [];
```

5.3.5.2 station_id

The "station_id" entry is an array of station ids to be retained or the filename which contains station ids. An array of station ids contains a comma-separated list. An empty list indicates that all stations should be retained.

For example: station_id = ["KDEN"];

```
station_id = [];
```

5.3.5.3 elevation_range

The "elevation_range" entry is a dictionary which contains "beg" and "end" entries specifying the range of observing locations elevations to be retained.

```
elevation_range = {
   beg =  -1000;
   end = 100000;
}
```

5.3.5.4 pb_report_type

The "pb_report_type" entry is an array of PREPBUFR report types to be retained. The numeric "pb_report_type" entry allows for further stratification within message types. An empty list indicates that all should be retained.

See: Code table for PREPBUFR report types used by Regional NAM GSI analyses

For example:

Report Type 120 is for message type ADPUPA but is only RAWINSONDE
Report Type 132 is for message type ADPUPA but is only FLIGHT-LEVEL RECON
and PROFILE DROPSONDE

```
pb_report_type  = [];
```

5.3.5.5 in_report_type

The "in_report_type" entry is an array of input report type values to be retained. The numeric "in_report_type" entry provides additional stratification of observations. An empty list indicates that all should be retained.

See: Code table for input report types

For example:

Input Report Type 11 Fixed land RAOB and PIBAL by block and station number
Input Report Type 12 Fixed land RAOB and PIBAL by call letters

```
in_report_type = [];
```

5.3.5.6 instrument_type

The "instrument_type" entry is an array of instrument types to be retained. An empty list indicates that all should be retained.

```
instrument_type = [];
```

5.3.5.7 level_range

The "level_range" entry is a dictionary which contains "beg" and "end" entries specifying the range of vertical levels (1 to 255) to be retained.

```
level_range = {
   beg = 1;
   end = 255;
}
```

5.3.5.8 level_category

The "level_category" entry is an array of integers specifying which level categories should be retained:

0 = Surface level (mass reports only)

1 = Mandatory level (upper-air profile reports)

2 = Significant temperature level (upper-air profile reports)

2 = Significant temperature and winds-by-pressure level (future combined mass and wind upper-air reports)

3 = Winds-by-pressure level (upper-air profile reports)

4 = Winds-by-height level (upper-air profile reports)

5 = Tropopause level (upper-air profile reports)

6 = Reports on a single level (For example: aircraft, satellite-wind, surface wind, precipitable water retrievals, etc.)

7 = Auxiliary levels generated via interpolation from spanning levels (upper-air profile reports)

An empty list indicates that all should be retained.

See: Current Table A Entries in PREPBUFR mnemonic table

```
level_category = [];
```

5.3.5.9 obs_bufr_var

The "obs_bufr_var" entry is an array of strings containing BUFR variable names to be retained or derived. This replaces the "obs_grib_code" setting from earlier versions of MET. Run PB2NC on your data with the "-index" command line option to see the list of available observation variables.

Observation variables that can be derived begin with "D_":

D_DPT for Dew point Temperature in K
D_WDIR for Wind Direction
D_WIND for Wind Speed in m/s
D_RH for Relative Humidity in %
D_MIXR for Humidity Mixing Ratio in kg/kg
D_PRMSL for Pressure Reduced to Mean Sea Level in Pa

```
obs_bufr_var = [ "QOB", "TOB", "ZOB", "UOB", "VOB" ];
```

5.3.5.10 obs_bufr_map

Mapping of input BUFR variable names to output variables names. The default PREPBUFR map, obs_prepbufr_map, is appended to this map. Users may choose to rename BUFR variables to match the naming convention of the forecast the observation is used to verify.

```
obs_bufr_map = [];
```

5.3.5.11 obs_prepbufr_map

Default mapping for PREPBUFR. Replace input BUFR variable names with GRIB abbreviations in the output. This default map is appended to obs_bufr_map. This should not typically be overridden. This default mapping provides backward compatibility for earlier versions of MET which wrote GRIB abbreviations to the output.

```
obs_prepbufr_map = [
   { key = "POB";     val = "PRES";  },
   { key = "QOB";     val = "SPFH";  },
   { key = "TOB";     val = "TMP";   },
   { key = "ZOB";     val = "HGT";   },
   { key = "UOB";     val = "UGRD";  },
   { key = "VOB";     val = "VGRD";  },
   { key = "D_DPT";   val = "DPT";   },
   { key = "D_WDIR";  val = "WDIR";  },
   { key = "D_WIND";  val = "WIND";  },
   { key = "D_RH";    val = "RH";    },
   { key = "D_MIXR";  val = "MIXR";  },
   { key = "D_PRMSL"; val = "PRMSL"; }
];
```

5.3.5.12 quality_mark_thresh

The "quality_mark_thresh" entry specifies the maximum quality mark value to be retained. Observations with a quality mark LESS THAN OR EQUAL TO this threshold will be retained, while observations with a quality mark GREATER THAN this threshold will be discarded.

See Code table for observation quality markers

```
quality_mark_thresh = 2;
```

5.3.5.13 event_stack_flag

The "event_stack_flag" entry is set to TOP or BOTTOM to specify whether observations should be drawn from the top of the event stack (most quality controlled) or the bottom of the event stack (most raw).

```
event_stack_flag = TOP;
```

5.3.6 SeriesAnalysisConfig_default

5.3.6.1 block_size

Computation may be memory intensive, especially for large grids. The "block_size" entry sets the number of grid points to be processed concurrently (i.e. in one pass through a time series). Smaller values require less memory but increase the number of passes through the data. If set less than or equal to 0, it is automatically reset to the number of grid points, and they are all processed concurrently.

```
block_size = 1024;
```

5.3.6.2 vld_thresh

Ratio of valid matched pairs to total length of series for a grid point. If valid threshold is exceeded at that grid point the statistics are computed and stored. If not, a bad data flag is stored. The default setting requires all data in the series to be valid.

```
vld_thresh = 1.0;
```

5.3.6.3 output_stats

Statistical output types need to be specified explicitly. Refer to User's Guide for available output types. To keep output file size reasonable, it is recommended to process a few output types at a time, especially if the grid is large.

```
output_stats = {
   fho    = [];
   ctc    = [];
   cts    = [];
   mctc   = [];
   mcts   = [];
   cnt    = [ "RMSE", "FBAR", "OBAR" ];
   sl1l2  = [];
   pct    = [];
   pstd   = [];
   pjc    = [];
   prc    = [];
}
```

5.3.7 STATAnalysisConfig_default

5.3.7.1 jobs

The "jobs" entry is an array of STAT-Analysis jobs to be performed. Each element in the array contains the specifications for a single analysis job to be performed. The format for an analysis job is as follows:

-job job_name
OPTIONAL ARGS

The following types of jobs are supported:

5.3.7.1.1 -job filter

To filter out the STAT lines matching the job filtering criteria specified below and using the optional arguments below. The output STAT lines are written to the file specified using the "-dump_row" argument.

Required Args: -dump_row

Optional Args:

```
-set_hdr column_name value
   May be used multiple times to override data written to the
   output dump_row file.
```

5.3.7.1.2 -job summary

To compute summary information for a set of statistics. The summary output includes the mean, standard deviation, percentiles (0th, 10th, 25th, 50th, 75th, 90th, and 100th), range, and inter-quartile range. Also included are columns summarizing the computation of WMO mean values. Both unweighted and weighted mean values are reported, and they are computed using three types of logic:

- simple arithmetic mean (default)
- square root of the mean of the statistic squared (applied to columns listed in "wmo_sqrt_stats")
- apply fisher transform (applied to columns listed in "wmo_fisher_stats")

The columns of data to be summarized are specified in one of two ways:

- Specify the -line_type option once and specify one or more column names.
- Format the -column option as LINE_TYPE:COLUMN.

Use the -derive job command option to automatically derive statistics on the fly from input contingency tables and partial sums.

Use the -column_union TRUE/FALSE job command option to compute summary statistics across the union of input columns rather than processing them separately.

For TCStat, the "-column" argument may be set to:

- "TRACK" for track, along-track, and cross-track errors.
- "WIND" for all wind radius errors.
- "TI" for track and maximum wind intensity errors.
- "AC" for along-track and cross-track errors.
- "XY" for x-track and y-track errors.
- "col" for a specific column name.
- "col1-col2" for a difference of two columns.
- "ABS(col or col1-col2)" for the absolute value.

Required Args: -line_type, -column

Optional Args:

```
-by column_name
   To specify case information.
-out_alpha
   To override the default alpha value.
-derive
   To derive statistics on the fly.
-column_union
   To summarize multiple columns.
```

5.3.7.1.3 -job aggregate

To aggregate the STAT data for the STAT line type specified using the "-line_type" argument. The output of the job will be in the same format as the input line type specified. The following line types may be aggregated:

```
-line_type FHO, CTC, MCTC,
           SL1L2, SAL1L2, VL1L2, VAL1L2,
           PCT, NBRCNT, NBRCTC, GRAD,
           ISC, ECNT, RPS, RHIST, PHIST, RELP, SSVAR
```

Required Args: -line_type

5.3.7.1.4 -job aggregate_stat

To aggregate the STAT data for the STAT line type specified using the "-line_type" argument. The output of the job will be the line type specified using the "-out_line_type" argument. The valid combinations of "-line_type" and "-out_line_type" are listed below.

```
-line_type FHO, CTC,      -out_line_type CTS, ECLV
-line_type MCTC           -out_line_type MCTS
-line_type SL1L2, SAL1L2, -out_line_type CNT
-line_type VL1L2          -out_line_type VCNT
-line_type VL1L2, VAL1L2, -out_line_type WDIR (wind direction)
-line_type PCT,           -out_line_type PSTD, PJC, PRC, ECLV
-line_type NBRCTC,        -out_line_type NBRCTS
-line_type ORANK,         -out_line_type ECNT, RPS, RHIST, PHIST,
                                         RELP, SSVAR
-line_type MPR,           -out_line_type FHO, CTC, CTS,
                                         MCTC, MCTS, CNT,
                                         SL1L2, SAL1L2,
                                         VL1L2, VCNT,
                                         PCT, PSTD, PJC, PRC, ECLV,
                                         WDIR (wind direction)
```

Required Args: -line_type, -out_line_type

Additional Required Args for -line_type MPR:

```
-out_thresh or -out_fcst_thresh and -out_obs_thresh
   When -out_line_type FHO, CTC, CTS, MCTC, MCTS,
                        PCT, PSTD, PJC, PRC
```

Additional Optional Args for -line_type MPR:

```
-mask_grid, -mask_poly, -mask_sid
-out_thresh or -out_fcst_thresh and -out_obs_thresh
-out_cnt_logic
-out_wind_thresh or -out_fcst_wind_thresh and
-out_obs_wind_thresh
-out_wind_logic
When -out_line_type WDIR
```

Additional Optional Arg for:

```
-line_type ORANK -out_line_type PHIST, SSVAR ...
-out_bin_size
```

Additional Optional Args for:

```
-out_line_type ECLV ...
-out_eclv_points
```

5.3.7.1.5 -job ss_index

The skill score index job can be configured to compute a weighted average of skill scores derived from a configurable set of variables, levels, lead times, and statistics. The skill score index is computed using two models, a forecast model and a reference model. For each statistic in the index, a skill score is computed as:

SS = 1 - (S[model]*S[model])/(S[reference]*S[reference])

Where S is the statistic.

Next, a weighted average is computed over all the skill scores.

Lastly, an index value is computed as:

Index = sqrt(1/(1-SS[avg]))

Where SS[avg] is the weighted average of skill scores.

Required Args:

```
Exactly 2 entries for -model, the forecast model and reference
For each term of the index:
-fcst_var, -fcst_lev, -fcst_lead, -line_type, -column, -weight
Where -line_type is CNT or CTS and -column is the statistic.
Optionally, specify other filters for each term, -fcst_thresh.
```

5.3.7.1.6 -job go_index

The GO Index is a special case of the skill score index consisting of a predefined set of variables, levels, lead times, statistics, and weights.

For lead times of 12, 24, 36, and 48 hours, it contains RMSE for:

```
- Wind Speed at the surface(b), 850(a), 400(a), 250(a) mb
- Dew point Temperature at the surface(b), 850(b), 700(b), 400(b) mB
- Temperature at the surface(b), 400(a) mB
- Height at 400(a) mB
- Sea Level Pressure(b)
Where (a) means weights of 4, 3, 2, 1 for the lead times, and
      (b) means weights of 8, 6, 4, 2 for the lead times.
```

Required Args: None

5.3.7.1.7 -job ramp

The ramp job operates on a time-series of forecast and observed values and is analogous to the RIRW (Rapid Intensification and Weakening) job supported by the tc_stat tool. The amount of change from one time to the next is computed for forecast and observed values. Those changes are thresholded to define events which are used to populate a 2x2 contingency table.

Required Args:

```
-ramp_thresh (-ramp_thresh_fcst or -ramp_thresh_obs)
   For DYDT, threshold for the amount of change required to
   define an event.
   For SWING, threshold the slope.
-swing_width val
   Required for the swinging door algorithm width.
```

Optional Args:

```
-ramp_type str
   Overrides the default ramp definition algorithm to be used.
   May be set to DYDT (default) or SWING for the swinging door
   algorithm.
-line_type str
   Overrides the default input line type, MPR.
-out_line_type str
   Overrides the default output line types of CTC and CTS.
   Set to CTC,CTS,MPR for all possible output types.
-column fcst_column,obs_column
   Overrides the default forecast and observation columns
   to be used, FCST and OBS.
-ramp_time HH[MMSS] (-ramp_time_fcst or -ramp_time_obs)
   Overrides the default ramp time interval, 1 hour.
-ramp_exact true/false (-ramp_exact_fcst or -ramp_exact_obs)
   Defines ramps using an exact change (true, default) or maximum
   change in the time window (false).
-ramp_window width in HH[MMSS] format
-ramp_window beg end in HH[MMSS] format
   Defines a search time window when attempting to convert misses
   to hits and false alarms to correct negatives. Use 1 argument
   to define a symmetric time window or 2 for an asymmetric
   window. Default window is 0 0, requiring an exact match.
```

Job command FILTERING options to further refine the STAT data:

Each optional argument may be used in the job specification multiple times unless otherwise indicated. When multiple optional arguments of the same type are indicated, the analysis will be performed over their union:

```
"-model            name"
"-fcst_lead        HHMMSS"
"-obs_lead         HHMMSS"
"-fcst_valid_beg   YYYYMMDD[_HH[MMSS]]" (use once)
"-fcst_valid_end   YYYYMMDD[_HH[MMSS]]" (use once)
"-obs_valid_beg    YYYYMMDD[_HH[MMSS]]" (use once)
"-obs_valid_end    YYYYMMDD[_HH[MMSS]]" (use once)
"-fcst_init_beg    YYYYMMDD[_HH[MMSS]]" (use once)
"-fcst_init_end    YYYYMMDD[_HH[MMSS]]" (use once)
"-obs_init_beg     YYYYMMDD[_HH[MMSS]]" (use once)
"-obs_init_end     YYYYMMDD[_HH[MMSS]]" (use once)
"-fcst_init_hour   HH[MMSS]"
"-obs_init_hour    HH[MMSS]"
"-fcst_valid_hour" HH[MMSS]
"-obs_valid_hour"  HH[MMSS]
"-fcst_var         name"
"-obs_var          name"
"-fcst_lev         name"
"-obs_lev          name"
"-obtype           name"
"-vx_mask          name"
"-interp_mthd      name"
"-interp_pnts      n"
"-fcst_thresh      t"
"-obs_thresh       t"
"-cov_thresh       t"
"-thresh_logic     UNION, or, ||
                   INTERSECTION, and, &&
                   SYMDIFF, symdiff, *
"-alpha            a"
"-line_type        type"
"-column           name"
"-weight           value"
```

Job command FILTERING options that may be used only when -line_type has been listed once. These options take two arguments: the name of the data column to be used and the min, max, or exact value for that column. If multiple column eq/min/max/str/exc options are listed, the job will be performed on their intersection:

```
"-column_min     col_name value"     e.g. -column_min BASER 0.02
"-column_max     col_name value"
"-column_eq      col_name value"
"-column_thresh  col_name threshold" e.g. -column_thresh FCST '>273'
"-column_str     col_name string" separate multiple filtering strings
                                  with commas
"-column_str_exc col_name string" separate multiple filtering strings
                                  with commas
```

Job command options to DEFINE the analysis job. Unless otherwise noted, these options may only be used ONCE per analysis job:

```
"-dump_row        path"
```

```
"-mask_grid       name"
"-mask_poly       file"
"-mask_sid        file|list" see description of "sid" entry above
```

```
"-out_line_type   name"
"-out_thresh      value" sets both -out_fcst_thresh and -out_obs_thresh
"-out_fcst_thresh value" multiple for multi-category contingency tables
                         and probabilistic forecasts
"-out_obs_thresh  value" multiple for multi-category contingency tables
"-out_cnt_logic   value"
```

```
"-out_wind_thresh      value"
"-out_fcst_wind_thresh value"
"-out_obs_wind_thresh  value"
"-out_wind_logic       value"
```

```
"-out_bin_size    value"
```

```
"-out_eclv_points value" see description of "eclv_points" config file
                         entry
```

```
"-out_alpha       value"
```

```
"-boot_interval   value"
"-boot_rep_prop   value"
"-n_boot_rep      value"
"-boot_rng        value"
"-boot_seed       value"
```

```
"-hss_ec_value    value"
"-rank_corr_flag  value"
"-vif_flag        value"
```

```
-out_stat path
   To write a .stat output file for aggregate and aggregate_stat jobs
   including the .stat header columns. Multiple input values for each
   header column are written to the output as a comma-separated list
   of unique values.

-set_hdr col_name value
```

(continues on next page)

(continued from previous page)

```
   May be used multiple times to explicity specify what should be
   written to the header columns of the output .stat file for
   aggregate and aggregate_stat jobs or output dump_row file
     for filter jobs.
```

When using the "-by" job command option, you may reference those columns in the "-set_hdr" job command options. For example, when computing statistics separately for each station, write the station ID string to the VX_MASK column of the output .stat output file:

```
-job aggregate_stat -line_type MPR -out_line_type CNT \
-by OBS_SID -set_hdr VX_MASK OBS_SID -stat_out out.stat
When using mulitple "-by" options, use "CASE" to reference the full string:
-by FCST_VAR,OBS_SID -set_hdr DESC CASE -stat_out out.stat
```

```
jobs = [
   "-job filter         -line_type SL1L2 -vx_mask DTC165 \
    -dump_row  job_filter_SL1L2.stat",
   "-job summary        -line_type CNT   -alpha 0.050 -fcst_var TMP \
    -dump_row job_summary_ME.stat -column ME",
   "-job aggregate      -line_type SL1L2 -vx_mask DTC165 -vx_mask DTC166 \
    -fcst_var TMP -dump_row job_aggregate_SL1L2_dump.stat  \
    -out_stat job_aggregate_SL1L2_out.stat \
    -set_hdr VX_MASK CONUS",
   "-job aggregate_stat -line_type SL1L2 -out_line_type CNT -vx_mask DTC165  \
    -vx_mask DTC166 -fcst_var TMP \
    -dump_row  job_aggregate_stat_SL1L2_CNT_in.stat",
   "-job aggregate_stat -line_type MPR   -out_line_type CNT -vx_mask DTC165 \
    -vx_mask DTC166 -fcat_var TMP -dump_row job_aggregate_stat_MPR_CNT_in.stat",
   "-job aggregate      -line_type CTC   -fcst_thresh <300.000 -vx_mask DTC165 \
    -vx_mask DTC166 -fcst_var TMP -dump_row job_aggregate_CTC_in.stat",
   "-job aggregate_stat -line_type CTC   -out_line_type CTS \
    -fcst_thresh <300.000 -vx_mask DTC165 -vx_mask DTC166 -fcst_var TMP \
    -dump_row job_aggregate_stat_CTC_CTS_in.stat",
   "-job aggregate      -line_type MCTC  -column_eq N_CAT 4 -vx_mask DTC165 \
    -vx_mask DTC166 -fcst_var APCP_24 -dump_row job_aggregate_MCTC_in.stat",
   "-job aggregate_stat -line_type MCTC  -out_line_type MCTS \
    -column_eq N_CAT 4 -vx_mask DTC165 -vx_mask DTC166 -fcst_var APCP_24 \
    -dump_row job_aggregate_stat_MCTC_MCTS_in.stat",
   "-job aggregate      -line_type PCT   -vx_mask DTC165 -vx_mask DTC166 \
    -dump_row job_aggregate_PCT_in.stat",
   "-job aggregate_stat -line_type PCT   -out_line_type PSTD -vx_mask DTC165 \
    -vx_mask DTC166 -dump_row job_aggregate_stat_PCT_PSTD_in.stat",
   "-job aggregate      -line_type ISC   -fcst_thresh >0.000 -vx_mask TILE_TOT \
    -fcst_var APCP_12 -dump_row job_aggregate_ISC_in.stat",
   "-job aggregate      -line_type RHIST -obtype MC_PCP -vx_mask HUC4_1605 \
    -vx_mask HUC4_1803 -dump_row job_aggregate_RHIST_in.stat",
```

(continues on next page)

(continued from previous page)

```
   "-job aggregate      -line_type SSVAR -obtype MC_PCP -vx_mask HUC4_1605 \
    -vx_mask HUC4_1803 -dump_row job_aggregate_SSVAR_in.stat",
   "-job aggregate_stat -line_type ORANK -out_line_type RHIST -obtype ADPSFC \
    -vx_mask HUC4_1605 -vx_mask HUC4_1803 \
    -dump_row job_aggregate_stat_ORANK_RHIST_in.stat"
];
```

5.3.7.2 wmo_sqrt_stats and wmo_fisher_stats

The "wmo_sqrt_stats" and "wmo_fisher_stats" entries are arrays of strings to control the logic used to compute the WMO mean value for the summary job type, as described in the *:ref:-job summary* section. Each entry is a line type followed by a statistic column name. Statistics computed using the default arithmetic mean method do not need to be listed.

```
wmo_sqrt_stats   = [ "CNT:FSTDEV",  "CNT:OSTDEV",  "CNT:ESTDEV",
                     "CNT:RMSE",    "CNT:RMSFA",   "CNT:RMSOA",
                     "VCNT:FS_RMS", "VCNT:OS_RMS", "VCNT:RMSVE",
                     "VCNT:FSTDEV", "VCNT:OSTDEV" ];

wmo_fisher_stats = [ "CNT:PR_CORR", "CNT:SP_CORR",
                     "CNT:KT_CORR", "CNT:ANOM_CORR",
                     "CNT:ANOM_CORR_UNCNTR" ];
```

5.3.7.3 vif_flag

The "vif_flag" entry is a boolean to indicate whether a variance inflation factor should be computed when aggregating a time series of contingency table counts or partial sums. The VIF is used to adjust the normal confidence intervals computed for the aggregated statistics.

```
vif_flag = FALSE;
```

5.3.8 WaveletStatConfig_default

5.3.8.1 grid_decomp_flag

The "grid_decomp_flag" entry specifies how the grid should be decomposed in Wavelet-Stat into dyadic (2^n x 2^n) tiles:

- AUTO to tile the input data using tiles of dimension n by n where n is the largest integer power of 2 less than the smallest dimension of the input data. Center as many tiles as possible with no overlap.
- TILE to use the tile definition specified below.
- PAD to pad the input data out to the nearest integer power of 2.

```
grid_decomp_flag = AUTO;
```

5.3.8.2 tile

The "tile" entry is a dictionary that specifies how tiles should be defined in Wavelet-Stat when the "grid_decomp_flag" is set to TILE:

5.3.8.2.1 tile.width

The "width" entry specifies the dimension for all tiles and must be an integer power of 2.

5.3.8.2.2 tile.location

The "location" entry is an array of dictionaries where each element consists of an "x_ll" and "y_ll" entry specifying the lower-left (x,y) coordinates of the tile.

```
tile = {
   width = 0;
   location = [
      {
         x_ll = 0;
         y_ll = 0;
      }
   ];
}
```

5.3.8.3 wavelet

The "wavelet" entry is a dictionary in Wavelet-Stat that specifies how the wavelet decomposition should be performed:

5.3.8.3.1 wavelet.type

The "type" entry specifies which wavelet should be used.

5.3.8.3.2 wavelet.member

The "member" entry specifies the wavelet shape. See: Discrete Wavelet Transforms (DWT) initialization

5.3.8.3.3 Valid Combinations of type and member

- HAAR for Haar wavelet (member = 2)
- HAAR_CNTR for Centered-Haar wavelet (member = 2)
- DAUB for Daubechies wavelet (member = 4, 6, 8, 10, 12, 14, 16, 18, 20)
- DAUB_CNTR for Centered-Daubechies wavelet (member = 4, 6, 8, 10, 12, 14, 16, 18, 20)
- BSPLINE for Bspline wavelet (member = 103, 105, 202, 204, 206, 208, 301, 303, 305, 307, 309)
- BSPLINE_CNTR for Centered-Bspline wavelet (member = 103, 105, 202, 204, 206, 208, 301, 303, 305, 307, 309)

```
wavelet = {
   type   = HAAR;
   member = 2;
}
```

5.3.8.4 obs_raw_wvlt_object_plots

The "obs_raw_plot", "wvlt_plot", and "object_plot" entries are dictionaries similar to the "fcst_raw_plot" described in the "Settings common to multiple tools" section.

5.3.9 WWMCARegridConfig_default

5.3.9.1 to_grid

Please see the description of the "to_grid" entry in the "regrid" dictionary at *to_grid* (page 54).

5.3.9.2 NetCDF Output Information

Supply the NetCDF output information. For example:

```
variable_name = "Cloud_Pct";
units         = "percent";
long_name     = "cloud cover percent";
level         = "SFC";
```

```
variable_name = "";
units         = "";
long_name     = "";
level         = "";
```

5.3.9.3 max_minutes (pixel age)

Maximum pixel age in minutes

```
max_minutes = 120;
```

5.3.9.4 swap_endian

The WWMCA pixel age data is stored in binary data files in 4-byte blocks. The swap_endian option indicates whether the endian-ness of the data should be swapped after reading.

```
swap_endian = TRUE;
```

5.3.9.5 write_pixel_age

By default, wwmca_regrid writes the cloud percent data specified on the command line to the output file. This option writes the pixel age data, in minutes, to the output file instead.

```
write_pixel_age = FALSE;
```

Chapter 6

Tropical Cyclone Configuration Options

See Section 5 for a description of the configuration file syntax.

6.1 Configuration Settings Common to Multiple Tools

6.1.1 storm_id

Specify a comma-separated list of storm id's to be used:

2-letter basin, 2-digit cyclone number, 4-digit year

An empty list indicates that all should be used.
For example:

storm_id = ["AL092011"];

This may also be set using basin, cyclone, and timing information below.

```
storm_id = [];
```

6.1.2 basin

Specify a comma-separated list of basins to be used. Expected format is a 2-letter basin identifier. An empty list indicates that all should be used.

Valid basins: "WP", "IO", "SH", "CP", "EP", "AL", "SL"

For example:

basin = ["AL", "EP"];

```
basin = [];
```

6.1.3 cyclone

Specify a comma-separated list of cyclone numbers (01-99) to be used. An empty list indicates that all should be used.

For example:

cyclone = ["01", "02", "03"];

```
cyclone = [];
```

6.1.4 storm_name

Specify a comma-separated list of storm names to be used. An empty list indicates that all should be used.

For example:

storm_name = ["KATRINA"];

```
storm_name = [];
```

6.1.5 init_beg end inc exc

Specify a model initialization time window in YYYYMMDD[_HH[MMSS]] format or provide a list of specific initialization times to include (inc) or exclude (exc). Tracks whose initial time meets the specified criteria will be used. An empty string indicates that all times should be used.

In TC-Stat, the **-init_beg**, **-init_end**, **init_inc** and **-int_exc** job command options can be used to further refine these selections.

For example:

init_beg = "20100101";
init_end = "20101231";
init_inc = ["20101231_06"];
init_exc = ["20101231_00"];

```
init_beg = "";
init_end = "";
init_inc = [];
init_exc = [];
```

6.1.6 valid_beg end inc exc

Specify a model valid time window in YYYYMMDD[_HH[MMSS]] format or provide a list of specific valid times to include (inc) or exclude (exc). If a time window is specified, only tracks for which all points are contained within the window will be used. If valid times to include or exclude are specified, tracks will be subset down to the points which meet that criteria. Empty begin/end time strings and empty include/exclude lists indicate that all valid times should be used.

In TC-Stat, the **-valid_beg**, **-valid_end**, **valid_inc** and **-valid_exc** job command options can be used to further refine these selections.

For example:

valid_beg = "20100101";
valid_end = "20101231_12";
valid_inc = ["20101231_06"];
valid_exc = ["20101231_00"];

```
valid_beg = "";
valid_end = "";
valid_inc = [];
valid_exc = [];
```

6.1.7 init_hour

Specify a comma-separated list of model initialization hours to be used in HH[MMSS] format. An empty list indicates that all hours should be used.

For example:

init_hour = ["00", "06", "12", "18"];

```
init_hour = [];
```

6.1.8 lead_req

Specify the required lead time in HH[MMSS] format. Tracks that contain all of these required times will be used. If a track has additional lead times, it will be retained. An empty list indicates that no lead times are required to determine which tracks are to be used; all lead times will be used.

```
lead_req  = [];
```

6.1.9 version

Indicate the version number for the contents of this configuration file. The value should generally not be modified.

```
version = "VN.N";
```

6.2 Settings Specific to Individual Tools

6.2.1 TCPairsConfig_default

6.2.1.1 model

The "model" entry specifies an array of model names to be verified. If verifying multiple models, choose descriptive model names (no whitespace) to distinguish between their output.

For example:

model = ["AHW4", "AHWI"];

```
model  = [];
```

6.2.1.2 init_mask, valid_mask

Specify spatial masking regions to be applied. These masks can be specified as lat/lon polyline files, the NetCDF output of the gen_vx_mask tool, or any gridded data file directly supported by MET. Tracks whose initial location (i.e. lead time of 0) falls within the "init_mask" region will be used. Generally, tracks for which all locations fall within the "valid_mask" region will be used. However, in RMW-Analysis, only data for track points falling inside the "valid_mask" region will be used.

For example:

init_mask = "MET_BASE/poly/EAST.poly";
valid_mask = "/path/to/gen_vx_mask_output.nc";

```
init_mask  = "";
valid_mask = "";
```

6.2.1.3 check_dup

Specify whether the code should check for duplicate ATCF lines when building tracks. Setting this to FALSE makes the parsing of tracks quicker.

For example:

check_dup = FALSE;

```
check_dup = FALSE;
```

6.2.1.4 interp12

Specify whether special processing should be performed for interpolated model names ending in 'I' (e.g. AHWI). Search for corresponding tracks whose model name ends in '2' (e.g. AHW2) and apply the following logic:

- NONE to do nothing.
- FILL to create a copy of '2' track and rename it as 'I' only when the 'I' track does not already exist.
- REPLACE to create a copy of the '2' track and rename it as 'I' in all cases, replacing any 'I' tracks that may already exist.

```
interp12 = REPLACE;
```

6.2.1.5 consensus

Specify how consensus forecasts should be defined:

name = consensus model name
members = array of consensus member model names
required = array of TRUE/FALSE for each member if empty, default is FALSE
min_req = minimum number of members required for the consensus

For example:

```
consensus = [
      {
            name = "CON1";
            members = [ "MOD1", "MOD2", "MOD3" ];
            required = [ TRUE, FALSE, FALSE ];
            min_req = 2;
      }
];
```

```
consensus = [];
```

6.2.1.6 lag_time

Specify a comma-separated list of forecast lag times to be used in HH[MMSS] format. For each ADECK track identified, a lagged track will be derived for each entry listed.

For example:

```
lag_time = [ "06", "12" ];
```

```
lag_time = [];
```

6.2.1.7 best

Specify comma-separated lists of CLIPER/SHIFOR baseline forecasts to be derived from the BEST and operational tracks, as defined by the best_technique and oper_technique settings.

Derived from BEST tracks:
BCLP, BCS5, BCD5, BCLA
Derived from OPER tracks:
OCLP, OCS5, OCD5, OCDT

For example:

best_technique = ["BEST"];

```
best_technique = [ "BEST" ];
best_baseline  = [];
oper_technique = [ "CARQ" ];
oper_baseline  = [];
```

6.2.1.8 anly_track

Analysis tracks consist of multiple track points with a lead time of zero for the same storm. An analysis track may be generated by running model analysis fields through a tracking algorithm. Specify which datasets should be searched for analysis track data by setting this to NONE, ADECK, BDECK, or BOTH. Use BOTH to create pairs using two different analysis tracks.

For example:

anly_track = BDECK;

```
anly_track = BDECK;
```

6.2.1.9 match_points

Specify whether only those track points common to both the ADECK and BDECK tracks should be written out.

For example:

match_points = FALSE;

```
match_points = FALSE;
```

6.2.1.10 dland_file

Specify the NetCDF output of the gen_dland tool containing a gridded representation of the minimum distance to land.

```
dland_file = "MET_BASE/tc_data/dland_nw_hem_tenth_degree.nc";
```

6.2.1.11 watch_warn

Specify watch/warning information. Specify an ASCII file containing watch/warning information to be used. At each track point, the most severe watch/warning status in effect, if any, will be written to the output. Also specify a time offset in seconds to be added to each watch/warning time processed. NHC applies watch/warning information to all track points occurring 4 hours (-14400 second) prior to the watch/warning time.

```
watch_warn = {
   file_name   = "MET_BASE/tc_data/wwpts_us.txt";
   time_offset = -14400;
}
```

6.2.1.12 basin_map

The basin_map entry defines a mapping of input names to output values. Whenever the basin string matches "key" in the input ATCF files, it is replaced with "val". This map can be used to modify basin names to make them consistent across the ATCF input files.

Many global modeling centers use ATCF basin identifiers based on region (e.g., "SP" for South Pacific Ocean, etc.), however the best track data provided by the Joint Typhoon Warning Center (JTWC) use just one basin identifier "SH" for all of the Southern Hemisphere basins. Additionally, some modeling centers may report basin identifiers separately for the Bay of Bengal (BB) and Arabian Sea (AB) whereas JTWC uses "IO".

The basin mapping allows MET to map the basin identifiers to the expected values without having to modify your data. For example, the first entry in the list below indicates that any data entries for "SI" will be

matched as if they were "SH". In this manner, all verification results for the Southern Hemisphere basins will be reported together as one basin.

An empty list indicates that no basin mapping should be used. Use this if you are not using JTWC best tracks and you would like to match explicitly by basin or sub-basin. Note that if your model data and best track do not use the same basin identifier conventions, using an empty list for this parameter will result in missed matches.

```
basin_map = [
   { key = "SI"; val = "SH"; },
   { key = "SP"; val = "SH"; },
   { key = "AU"; val = "SH"; },
   { key = "AB"; val = "IO"; },
   { key = "BB"; val = "IO"; }
];
```

6.2.2 TCStatConfig_default

6.2.2.1 amodel, bmodel

Stratify by the AMODEL or BMODEL columns. Specify comma-separated lists of model names to be used for all analyses performed. May add to this list using the "-amodel" and "-bmodel" job command options.

For example:

amodel = ["AHW4"];
bmodel = ["BEST"];

```
amodel = [];
bmodel = [];
```

6.2.2.2 init valid_hour lead req

Stratify by the initialization and valid hours and lead time. Specify a comma-separated list of initialization hours, valid hours, and lead times in HH[MMSS] format. May add using the "-init_hour", "-valid_hour", "-lead", and "-lead_req" job command options.

For example:

init_hour = ["00"];
valid_hour = ["12"];
lead = ["24", "36"];
lead_req = ["72", "84", "96", "108"];

```
init_hour  = [];
valid_hour = [];
lead       = [];
lead_req   = [];
```

6.2.2.3 init_mask, valid_mask

Stratify by the contents of the INIT_MASK and VALID_MASK columns. Specify a comma-separated list of strings for these options. May add using the "-init_mask" and "-valid_mask" job command options.

For example:

init_mask = ["AL_BASIN", "EP_BASIN"];

```
init_mask  = [];
valid_mask = [];
```

6.2.2.4 line_type

Stratify by the LINE_TYPE column. May add using the "-line_type" job command option.

For example:

line_type = ["TCMPR"];

```
line_type = [];
```

6.2.2.5 track_watch_warn

Stratify by checking the watch/warning status for each track point common to both the ADECK and BDECK tracks. If the watch/warning status of any of the track points appears in the list, retain the entire track. Individual watch/warning status by point may be specified using the -column_str options below, but this option filters by the track maximum. May add using the "-track_watch_warn" job command option. The value "ALL" matches HUWARN, TSWARN, HUWATCH, and TSWATCH.

For example:

track_watch_warn = ["HUWATCH", "HUWARN"];

```
track_watch_warn = [];
```

6.2.2.6 column_thresh_name_and_val

Stratify by applying thresholds to numeric data columns. Specify a comma-separated list of columns names and thresholds to be applied. May add using the "-column_thresh name thresh" job command options.

For example:

column_thresh_name = ["ADLAND", "BDLAND"];
column_thresh_val = [>200, >200];

```
column_thresh_name = [];
column_thresh_val  = [];
```

6.2.2.7 column_str_name, column_str_val

Stratify by performing string matching on non-numeric data columns. Specify a comma-separated list of columns names and values to be included in the analysis. May add using the "-column_str name string" job command options.

For example:

column_str_name = ["LEVEL", "LEVEL"];
column_str_val = ["HU", "TS"];

```
column_str_name = [];
column_str_val  = [];
```

6.2.2.8 column_str_name val

Stratify by performing string matching on non-numeric data columns. Specify a comma-separated list of columns names and values to be excluded from the analysis. May add using the "-column_str_exc name string" job command options.

For example:

column_str_exc_name = ["LEVEL"];
column_str_exc_val = ["TD"];

```
column_str_exc_name = [];
column_str_exc_val  = [];
```

6.2.2.9 init_thresh_name, init_thresh_val

Just like the column_thresh options above, but apply the threshold only when lead = 0. If lead = 0 value does not meet the threshold, discard the entire track. May add using the "-init_thresh name thresh" job command options.

For example:

init_thresh_name = ["ADLAND"];
init_thresh_val = [>200];

```
init_thresh_name = [];
init_thresh_val  = [];
```

6.2.2.10 init_str_name, init_str_val

Just like the column_str options above, but apply the string matching only when lead = 0. If lead = 0 string does not match, discard the entire track. May add using the "-init_str name thresh" job command options.

For example:

init_str_name = ["LEVEL"];
init_str_val = ["HU"];

```
init_str_name = [];
init_str_val  = [];
```

6.2.2.11 init_str_exc_name and _exc_val

Just like the column_str_exc options above, but apply the string matching only when lead = 0. If lead = 0 string does match, discard the entire track. May add using the "-init_str_exc name thresh" job command options.

For example:

init_str_exc_name = ["LEVEL"];

init_str_exc_val = ["HU"];

```
init_str_exc_name = [];
init_str_exc_val  = [];
```

6.2.2.12 water_only

Stratify by the ADECK and BDECK distances to land. Once either the ADECK or BDECK track encounters land, discard the remainder of the track.

For example:

water_only = FALSE;

```
water_only = FALSE;
```

6.2.2.13 rirw

Specify whether only those track points for which rapid intensification or weakening of the maximum wind speed occurred in the previous time step should be retained.

The NHC considers a 24-hour change >=30 kts to constitute rapid intensification or weakening.

May modify using the following job command options:

"-rirw_track"
"-rirw_time" for both or "-rirw_time_adeck" and "-rirw_time_bdeck"
"-rirw_exact" for both or "-rirw_exact_adeck" and "-rirw_exact_bdeck"
"-rirw_thresh" for both or "-rirw_thresh_adeck" and "-rirw_thresh_bdeck"

```
rirw = {
   track  = NONE;       Specify which track types to search (NONE, ADECK,
                        BDECK, or BOTH)
   adeck = {
      time   = "24";    Rapid intensification/weakening time period in HHMMSS
                        format.
      exact  = TRUE;    Use the exact or maximum intensity difference over the
                        time period.
      thresh = >=30.0;  Threshold for the intensity change.
   }
```

(continues on next page)

(continued from previous page)

```
   bdeck = adeck;       Copy settings to the BDECK or specify different logic.
}
```

6.2.2.14 landfall beg end

Specify whether only those track points occurring near landfall should be retained, and define the landfall retention window as a time string in HH[MMSS] format (or as an integer number of seconds) offset from the landfall time. Landfall is defined as the last BDECK track point before the distance to land switches from positive to 0 or negative.

May modify using the "-landfall_window" job command option, which automatically sets -landfall to TRUE.

The "-landfall_window" job command option takes 1 or 2 arguments in HH[MMSS] format. Use 1 argument to define a symmetric time window. For example, "-landfall_window 06" defines the time window +/- 6 hours around the landfall time. Use 2 arguments to define an asymmetric time window. For example, "-landfall_window 00 12" defines the time window from the landfall event to 12 hours after.

For example:

landfall = FALSE;
landfall_beg = "-24"; (24 hours prior to landfall)
landfall_end = "00";

```
landfall     = FALSE;
landfall_beg = "-24";
landfall_end = "00";
```

6.2.2.15 event_equal

Specify whether only those cases common to all models in the dataset should be retained. May modify using the "-event_equal" job command option.

For example:

event_equal = FALSE;

```
event_equal = FALSE;
```

6.2.2.16 event_equal_lead

Specify lead times that must be present for a track to be included in the event equalization logic.

```
event_equal_lead = [ "12", "24", "36" ];
```

6.2.2.17 out_int_mask

Apply polyline masking logic to the location of the ADECK track at the initialization time. If it falls outside the mask, discard the entire track. May modify using the "-out_init_mask" job command option.

For example:

out_init_mask = "";

```
out_init_mask = "";
```

6.2.2.18 out_valid_mask

Apply polyline masking logic to the location of the ADECK track at the valid time. If it falls outside the mask, discard only the current track point. May modify using the "-out_valid_mask" job command option.

For example:

out_valid_mask = "";

```
out_valid_mask = "";
```

6.2.2.19 job

The "jobs" entry is an array of TCStat jobs to be performed. Each element in the array contains the specifications for a single analysis job to be performed. The format for an analysis job is as follows:

-job job_name
OPTIONAL ARGS

The following types of jobs are supported:

6.2.2.19.1 -job filter

To filter out the TCST lines matching the job filtering criteria specified above and using the optional arguments below. The output TCST lines are written to the file specified using the "-dump_row" argument.

Required Args: -dump_row

To further refine the TCST data: Each optional argument may be used in the job specification multiple times unless otherwise indicated. When multiple optional arguments of the same type are indicated, the analysis will be performed over their union.

```
"-amodel            name"
"-bmodel            name"
"-lead        HHMMSS"
"-valid_beg   YYYYMMDD[_HH[MMSS]]" (use once)
"-valid_end   YYYYMMDD[_HH[MMSS]]" (use once)
"-valid_inc   YYYYMMDD[_HH[MMSS]]" (use once)
"-valid_exc   YYYYMMDD[_HH[MMSS]]" (use once)
"-init_beg    YYYYMMDD[_HH[MMSS]]" (use once)
"-init_end    YYYYMMDD[_HH[MMSS]]" (use once)
"-init_inc    YYYYMMDD[_HH[MMSS]]" (use once)
"-init_exc    YYYYMMDD[_HH[MMSS]]" (use once)
"-init_hour   HH[MMSS]"
"-valid_hour  HH[MMSS]
"-init_mask          name"
"-valid_mask         name"
"-line_type          name"
"-track_watch_warn   name"
"-column_thresh      name thresh"
"-column_str         name string"
"-column_str_exc     name string"
"-init_thresh        name thresh"
"-init_str           name string"
"-init_str_exc       name string"
```

Additional filtering options that may be used only when -line_type has been listed only once. These options take two arguments: the name of the data column to be used and the min, max, or exact value for that column. If multiple column eq/min/max/str options are listed, the job will be performed on their intersection:

```
"-column_min     col_name value" For example: -column_min TK_ERR 100.00
"-column_max     col_name value"
"-column_eq      col_name value"
"-column_str     col_name string" separate multiple filtering strings
                                  with commas
"-column_str_exc col_name string" separate multiple filtering strings
                                  with commas
```

Required Args: -dump_row

6.2.2.19.2 -job summary

To compute the mean, standard deviation, and percentiles (0th, 10th, 25th, 50th, 75th, 90th, and 100th) for the statistic specified using the "-line_type" and "-column" arguments. For TCStat, the "-column" argument may be set to:

- TRACK for track, along-track, and cross-track errors.
- WIND for all wind radius errors.
- TI for track and maximum wind intensity errors.
- AC for along-track and cross-track errors.
- XY for x-track and y-track errors.
- "col" for a specific column name.
- "col1-col2" for a difference of two columns.
- "ABS(col or col1-col2)" for the absolute value.

Use the -column_union TRUE/FALSE job command option to compute summary statistics across the union of input columns rather than processing them separately.

Required Args: -line_type, -column

Optional Args:

```
-by column_name to specify case information
-out_alpha to override default alpha value
-column_union to summarize multiple columns
```

6.2.2.19.3 -job rirw

To define rapid intensification/weakening contingency table using the ADECK and BDECK RI/RW settings and the matching time window and output contingency table counts and statistics.

Optional Args:

```
-rirw_window width in HH[MMSS] format to define a symmetric time window
-rirw_window beg end in HH[MMSS] format to define an asymmetric time window
 Default search time window is 0 0, requiring exact match
-rirw_time or -rirw_time_adeck and -rirw_time_bdeck to override defaults
-rirw_exact or -rirw_exact_adeck and -rirw_exact_bdeck to override defaults
-rirw_thresh or -rirw_thresh_adeck and -rirw_thresh_bdeck to override defaults
-by column_name to specify case information
-out_alpha to override default alpha value
-out_line_type to specify output line types (CTC, CTS, and MPR)
```

Note that the "-dump_row path" option results in 4 files being created:

path_FY_OY.tcst, path_FY_ON.tcst, path_FN_OY.tcst, and path_FN_ON.tcst, containing the TCST lines that were hits, false alarms, misses, and correct negatives, respectively. These files may be used as input for additional TC-Stat analysis.

6.2.2.19.4 -job probrirw

To define an Nx2 probabilistic contingency table by reading the PROBRIRW line type, binning the forecast probabilities, and writing output probabilistic counts and statistics.

Required Args:

```
-probrirw_thresh to define the forecast probabilities to be
   evaluated (For example: -probrirw_thresh 30)
```

Optional Args:

```
-probrirw_exact TRUE/FALSE to verify against the exact (for example:
   BDELTA column) or maximum (for example: BDELTA_MAX column) intensity
   change in the BEST track
-probrirw_bdelta_thresh to define BEST track change event
   threshold (For example: -probrirw_bdelta_thresh >=30)
-probrirw_prob_thresh to define output probability thresholds
   (for example: -probrirw_prob_thresh ==0.1)
-by column_name to specify case information
-out_alpha to override default alpha value
-out_line_type to specify output line types (PCT, PSTD, PRC, and PJC)
```

For the PROBRIRW line type, PROBRIRW_PROB is a derived column name. For example, the following options select 30 kt probabilities and match probability values greater than 0:

-probrirw_thresh 30 -column_thresh PROBRIRW_PROB >0

For example:

jobs = [
"-job filter -amodel AHW4 -dumprow ./tc_filter_job.tcst",
"-job filter -column_min TK_ERR 100.000
-dumprow ./tc_filter_job.tcst",
"-job summary -line_type TCMPR -column AC
-dumprow ./tc_summary_job.tcst",
"-job rirw -amodel AHW4 -dump_row ./tc_rirw_job"]

```
jobs = [];
```

6.2.3 TCGenConfig_default

6.2.3.1 init_freq

Model initialization frequency in hours, starting at 0.

```
init_freq = 6;
```

6.2.3.2 lead_window

Lead times in hours to be searched for genesis events.

```
lead_window = {
   beg = 24;
   end = 120;
}
```

6.2.3.3 min_duration

Minimum track duration for genesis event in hours.

```
min_duration = 12;
```

6.2.3.4 fcst_genesis

Forecast genesis event criteria. Defined as tracks reaching the specified intensity category, maximum wind speed threshold, and minimum sea-level pressure threshold. The forecast genesis time is the valid time of the first track point where all of these criteria are met.

```
fcst_genesis = {
   vmax_thresh = NA;
   mslp_thresh = NA;
}
```

6.2.3.5 best_genesis

BEST track genesis event criteria. Defined as tracks reaching the specified intensity category, maximum wind speed threshold, and minimum sea-level pressure threshold. The BEST track genesis time is the valid time of the first track point where all of these criteria are met.

```
best_genesis = {
   technique   = "BEST";
   category    = [ "TD", "TS" ];
   vmax_thresh = NA;
   mslp_thresh = NA;
}
```

6.2.3.6 oper_genesis

Operational track genesis event criteria. Defined as tracks reaching the specified intensity category, maximum wind speed threshold, and minimum sea-level pressure threshold. The operational track genesis time is valid time of the first track point where all of these criteria are met.

```
oper_genesis = {
   technique   = "CARQ";
   category    = [ "DB", "LO", "WV" ];
   vmax_thresh = NA;
   mslp_thresh = NA;
}
```

6.2.3.7 filter

Filter is an array of dictionaries containing the track filtering options listed below. If empty, a single filter is defined using the top-level settings.

```
filter = [];
```

6.2.3.8 desc

Description written to output DESC column

```
desc = "NA";
```

6.2.3.9 model

Forecast ATCF ID's If empty, all ATCF ID's found will be processed. Statistics will be generated separately for each ATCF ID.

```
model = [];
```

6.2.3.10 init_beg, init_end

Forecast and operational initialization time window, as strings in YYYYMMDD[_HH[MMSS]] format

```
init_beg = "";
init_end = "";
```

6.2.3.11 valid_beg, valid_end

Forecast, BEST, and operational valid time window, as strings in YYYYMMDD[_HH[MMSS]] format

```
valid_beg = "";
valid_end = "";
```

6.2.3.12 lead

Forecast and operational lead times, as strings in HH[MMSS] format

```
lead = [];
```

6.2.3.13 vx_mask

Spatial masking region (path to gridded data file or polyline file)

```
vx_mask = "";
```

6.2.3.14 dland_thresh

Distance to land threshold

```
dland_thresh = NA;
```

6.2.3.15 genesis_window

Genesis matching time window, in hours relative to the forecast genesis time

```
genesis_window = {
   beg = -24;
   end =  24;
}
```

6.2.3.16 genesis_radius

Genesis matching search radius in km.

```
genesis_radius = 300;
```

6.2.3.17 ci_alpha

Confidence interval alpha value

```
ci_alpha = 0.05;
```

6.2.3.18 output_flag

Statistical output types

```
output_flag = {
   fho    = NONE;
   ctc    = BOTH;
   cts    = BOTH;
   pct    = NONE;
   pstd   = NONE;
   pjc    = NONE;
   prc    = NONE;
   genmpr = NONE;
}
```

Chapter 7

Re-Formatting of Point Observations

There are several formats of point observations that may be preprocessed using the suite of reformatting tools in MET. These include PrepBUFR data from NCEP, SURFRAD data from NOAA, AERONET data from NASA, MADIS data from NOAA, little_r from WRF simulations, and user-defined data in a generic ASCII format. These steps are represented by the first columns in the MET flowchart depicted in Section 1. The software tools used to reformat point data are described in this section.

7.1 PB2NC Tool

This section describes how to configure and run the PB2NC tool. The PB2NC tool is used to stratify the contents of an input PrepBUFR point observation file and reformat it into NetCDF format for use by other MET tools. The PB2NC tool must be run on the input PrepBUFR point observation file prior to performing verification with the MET statistics tools.

7.1.1 pb2nc Usage

The usage statement for the PB2NC tool is shown below:

```
Usage: pb2nc
       prepbufr_file
       netcdf_file
       config_file
       [-pbfile PrepBUFR_file]
       [-valid_beg time]
       [-valid_end time]
       [-nmsg n]
       [-dump path]
       [-index]
       [-log file]
       [-v level]
       [-compress level]
```

pb2nc has both required and optional arguments.

7.1.1.1 Required Arguments for pb2nc

1. The **prepbufr_file** argument is the input PrepBUFR file to be processed.
2. The **netcdf_file** argument is the output NetCDF file to be written.
3. The **config_file** argument is the configuration file to be used. The contents of the configuration file are discussed below.

7.1.1.2 Optional Arguments for pb2nc

1. The **-pbfile prepbufr_file** option is used to pass additional input PrepBUFR files.
2. The **-valid_beg** time option in YYYYMMDD[_HH[MMSS]] format sets the beginning of the retention time window.
3. The **-valid_end** time option in YYYYMMDD[_HH[MMSS]] format sets the end of the retention time window.
4. The **-nmsg num_messages** option may be used for testing purposes. This argument indicates that only the first "num_messages" PrepBUFR messages should be processed rather than the whole file. This option is provided to speed up testing because running the PB2NC tool can take a few minutes for each file. Most users will not need this option.
5. The **-dump path** option may be used to dump the entire contents of the PrepBUFR file to several ASCII files written to the directory specified by "path". The user may use this option to view a human-readable version of the input PrepBUFR file, although writing the contents to ASCII files can be slow.
6. The **-index** option shows the available variables with valid data from the BUFR input. It collects the available variable list from BUFR input and checks the existence of valid data and directs the variable names with valid data to the screen. The NetCDF output won't be generated.
7. The **-log** file option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
8. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.
9. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the pb2nc calling sequence is shown below:

```
pb2nc sample_pb.blk \
sample_pb.nc \
PB2NCConfig
```

In this example, the PB2NC tool will process the input **sample_pb.blk** file applying the configuration specified in the **PB2NCConfig** file and write the output to a file named **sample_pb.nc**.

7.1.2 pb2nc Configuration File

The default configuration file for the PB2NC tool named **PB2NCConfig_default** can be found in the installed *share/met/config* directory. The version used for the installation test cases is available in *scripts/config*. It is recommended that users make a copy of configuration files prior to modifying their contents.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
obs_window = { beg  = -5400; end  = 5400; }
mask       = { grid = "";    poly = "";   }
tmp_dir    = "/tmp";
version    = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5. The use of temporary files in PB2NC is described in `Contributor's Guide Section %s`.

```
message_type = [];
```

Each PrepBUFR message is tagged with one of eighteen message types as listed in the Section 5 file. The **message_type** refers to the type of observation from which the observation value (or 'report') was derived. The user may specify a comma-separated list of message types to be retained. Providing an empty list indicates that all message types should be retained.

```
message_type_map = [ { key = "AIRCAR"; val = "AIRCAR_PROFILES"; } ];
```

The **message_type_map** entry is an array of dictionaries, each containing a **key** string and **val** string. This defines a mapping of input PrepBUFR message types to output message types. This provides a method for renaming input PrepBUFR message types.

```
message_type_group_map = [
   { key = "SURFACE"; val = "ADPSFC,SFCSHP,MSONET";               },
   { key = "ANYAIR";  val = "AIRCAR,AIRCFT";                      },
   { key = "ANYSFC";  val = "ADPSFC,SFCSHP,ADPUPA,PROFLR,MSONET"; },
   { key = "ONLYSF";  val = "ADPSFC,SFCSHP";                      }
];
```

The **message_type_group_map** entry is an array of dictionaries, each containing a **key** string and **val** string. This defines a mapping of message type group names to a comma-separated list of values. This map is defined in the config files for PB2NC, Point-Stat, or Ensemble-Stat. Modify this map to define sets of message types that should be processed together as a group. The **SURFACE** entry must be present to define message types for which surface verification logic should be applied.

```
station_id = [];
```

Each PrepBUFR message has a station identification string associated with it. The user may specify a comma-separated list of station IDs to be retained. Providing an empty list indicates that messages from all station IDs will be retained. It can be a file name containing a list of stations.

```
elevation_range = { beg = -1000; end = 100000; }
```

The **beg** and **end** variables are used to stratify the elevation (in meters) of the observations to be retained. The range shown above is set to -1000 to 100000 meters, which essentially retains every observation.

```
pb_report_type  = [];
in_report_type  = [];
instrument_type = [];
```

The **pb_report_type, in_report_type**, and **instrument_type** variables are used to specify comma-separated lists of PrepBUFR report types, input report types, and instrument types to be retained, respectively. If left empty, all PrepBUFR report types, input report types, and instrument types will be retained. See the following for more details:

Code table for PrepBUFR report types used by Regional NAM GSI analyses.

PrepBUFR Code table for input report types.

```
level_range    = { beg = 1; end = 255; }
level_category = [];
```

The **beg** and **end** variables are used to stratify the model level of observations to be retained. The range shown above is 1 to 255.

The **level_category** variable is used to specify a comma-separated list of PrepBUFR data level categories to retain. An empty string indicates that all level categories should be retained. Accepted values and their meanings are described in Table 7.1. See the following for more details:

PrepBUFR mnemonic table.

Table 7.1: Values for the level_category option.

Level category value	Description
0	Surface level
1	Mandatory level
2	Significant temperature level
3	Winds-by-pressure level
4	Winds-by-height level
5	Tropopause level
6	Reports on a single level
7	Auxiliary levels generated via interpolation from spanning levels

```
obs_bufr_var = [ 'QOB', 'TOB', 'ZOB', 'UOB', 'VOB' ];
```

Each PrepBUFR message will likely contain multiple observation variables. The **obs_bufr_var** variable is used to specify which observation variables should be retained or derived. The observation variable names are retrieved from the BUFR table embedded within the file. Users can run PB2NC with the **-index** command line argument to list out the variable names present in the file, and those names can be listed in this setting. If the list is empty, all BUFR variables present in the file are retained. This setting replaces the deprecated **obs_grib_code**.

The example **obs_bufr_var** setting above retains observations of QOB, TOB, ZOB, UOB, and VOB for specific humidity, temperature, height, and the u and v components of winds. Observations of those types are reported at the corresponding POB pressure level. In addition, PB2NC can derive several other variables from these observations. By convention, all observations that are derivable are named with a **D_** prefix:

- **D_DPT** for dew point (from POB and QOB)
- **D_WDIR** for wind direction (from UOB and VOB)
- **D_WIND** for wind speed (from UOB and VOB)
- **D_RH** for relative humidity (from POB, QOB, and TOB)
- **D_MIXR** for mixing ratio (from QOB)
- **D_PRMSL** for pressure reduced to mean sea level (from POB, TOB, and ZOB)
- **D_PBL** for planetary boundary layer height (from POB, QOB, TOB, ZOB, UOB, and VOB)
- **D_CAPE** for convective available potential energy (from POB, QOB, and TOB)
- **D_MLCAPE** for mixed layer convective available potential energy (from POB, QOB, and TOB)

In BUFR, lower quality mark values indicate higher quality observations. The quality marks for derived observations are computed as the maximum of the quality marks for its components. For example, **D_DPT** derived from **POB** with quality mark 1 and **QOB** with quality mark 2 is assigned a quality mark value of 2. **D_PBL**, **D_CAPE**, and **D_MLCAPE** are derived using data from multiple vertical levels. Their quality marks are computed as the maximum of their components over all vertical levels.

```
obs_bufr_map = [
   { key = 'POB';      val = 'PRES';  },
   { key = 'QOB';      val = 'SPFH';  },
   { key = 'TOB';      val = 'TMP';   },
   { key = 'ZOB';      val = 'HGT';   },
   { key = 'UOB';      val = 'UGRD';  },
   { key = 'VOB';      val = 'VGRD';  },
   { key = 'D_DPT';    val = 'DPT';   },
   { key = 'D_WDIR';   val = 'WDIR';  },
   { key = 'D_WIND';   val = 'WIND';  },
   { key = 'D_RH';     val = 'RH';    },
   { key = 'D_MIXR';   val = 'MIXR';  },
   { key = 'D_PRMSL';  val = 'PRMSL'; },
   { key = 'D_PBL';    val = 'PBL';   },
   { key = 'D_CAPE';   val = 'CAPE';  }
   { key = 'D_MLCAPE'; val = 'MLCAPE';  }
];
```

The BUFR variable names are not shared with other forecast data. This map is used to convert the BUFR name to the common name, like GRIB2. It allows to share the configuration for forecast data with PB2NC observation data. If there is no mapping, the BUFR variable name will be saved to output NetCDF file.

```
quality_mark_thresh = 2;
```

Each observation has a quality mark value associated with it. The **quality_mark_thresh** is used to stratify out which quality marks will be retained. The value shown above indicates that only observations with quality marks less than or equal to 2 will be retained.

```
event_stack_flag = TOP;
```

A PrepBUFR message may contain duplicate observations with different quality mark values. The **event_stack_flag** indicates whether to use the observations at the top of the event stack (observation values have had more quality control processing applied) or the bottom of the event stack (observation values have had no quality control processing applied). The flag value of **TOP** listed above indicates the observations with the most amount of quality control processing should be used, the **BOTTOM** option uses the data closest to raw values.

```
time_summary = {
   flag       = FALSE;
   raw_data   = FALSE;
   beg        = "000000";
   end        = "235959";
   step       = 300;
```

(continues on next page)

(continued from previous page)

```
   width      = 600;
   // width   = { beg = -300; end = 300; }
   grib_code  = [];
   obs_var    = [ "TMP", "WDIR", "RH" ];
   type       = [ "min", "max", "range", "mean", "stdev", "median", "p80" ];
   vld_freq   = 0;
   vld_thresh = 0.0;
}
```

The **time_summary** dictionary enables additional processing for observations with high temporal resolution. The **flag** entry toggles the **time_summary** on (**TRUE**) and off (**FALSE**). If the **raw_data** flag is set to TRUE, then both the individual observation values and the derived time summary value will be written to the output. If FALSE, only the summary values are written. Observations may be summarized across the user specified time period defined by the **beg** and **end** entries in HHMMSS format. The **step** entry defines the time between intervals in seconds. The **width** entry specifies the summary interval in seconds. It may either be set as an integer number of seconds for a centered time interval or a dictionary with beginning and ending time offsets in seconds.

This example listed above does a 10-minute time summary (width = 600;) every 5 minutes (step = 300;) throughout the day (beg = "000000"; end = 235959";). The first interval will be from 23:55:00 the previous day through 00:04:59 of the current day. The second interval will be from 0:00:00 through 00:09:59. And so on.

The two **width** settings listed above are equivalent. Both define a centered 10-minute time interval. Use the **beg** and **end** entries to define uncentered time intervals. The following example requests observations for one hour prior:

```
width = { beg = -3600; end = 0; }
```

The summaries will only be calculated for the observations specified in the **grib_code** or **obs_var** entries. The **grib_code** entry is an array of integers while the **obs_var** entries is an array of strings. The supported summaries are **min** (minimum), **max** (maximum), **range, mean, stdev** (standard deviation), **median** and **p##** (percentile, with the desired percentile value specified in place of ##). If multiple summaries are selected in a single run, a string indicating the summary method applied will be appended to the output message type.

The **vld_freq** and **vld_thresh** entries specify the required ratio of valid data for an output time summary value to be computed. This option is only applied when these entries are set to non-zero values. The **vld_freq** entry specifies the expected frequency of observations in seconds. The width of the time window is divided by this frequency to compute the expected number of observations for the time window. The actual number of valid observations is divided by the expected number to compute the ratio of valid data. An output time summary value will only be written if that ratio is greater than or equal to the **vld_thresh** entry. Detailed information about which observations are excluded is provided at debug level 4.

The quality mark for time summaries is always reported by PB2NC as bad data. Time summaries are computed by several MET point pre-processing tools using common library code. While BUFR quality marks are integers, the quality flags for other point data formats (MADIS NetCDF, for example) are stored as strings. MET does not currently contain logic to determine which quality flag strings are better or worse. Note however that any point observation whose quality mark does not meet the **quality_mark_thresh** criteria is not

used in the computation of time summaries.

7.1.3 pb2nc Output

Each NetCDF file generated by the PB2NC tool contains the dimensions and variables shown in Table 7.2 and Table 7.3.

Table 7.2: NetCDF file dimensions for pb2nc output

NetCDF Dimension	Description
mxstr, mxstr2, mxstr3	Maximum string lengths (16, 40, and 80)
nobs	Number of PrepBUFR observations in the file (UNLIMITED)
nhdr, npbhdr	Number of PrepBUFR messages in the file (variable)
nhdr_typ, nhdr_sid, nhdr_vld	Number of unique header message type, station ID, and valid time strings (variable)
nobs_qty	Number of unique quality control strings (variable)
obs_var_num	Number of unique observation variable types (variable)

Table 7.3: NetCDF variables in pb2nc output

NetCDF Variable	Dimension	Description	Data Type
obs_qty	nobs	Integer value of the n_obs_qty dimension for the observation quality control string	Integer
obs_hid	nobs	Integer value of the nhdr dimension for the header arrays with which this observation is associated	Integer
obs_vid	nobs	Integer value of the obs_var_num dimension for the observation variable name, units, and description	Integer
obs_lvl	nobs	Floating point pressure level in hPa or accumulation interval	Float
obs_hgt	nobs	Floating point height in meters above sea level	Float
obs_val	nobs	Floating point observation value.	Float
hdr_typ	nhdr	Integer value of the nhdr_typ dimension for the message type string	Integer
hdr_sid	nhdr	Integer value of the nhdr_sid dimension for the station ID string	Integer
hdr_vld	nhdr	Integer value of the nhdr_vld dimension for the valid time string	Integer
hdr_lat, hdr_lon	nhdr	Floating point latitude in degrees north and longitude in degrees east	Float
hdr_elv	nhdr	Floating point elevation of observing station in meters above sea level	Float
hdr_prpt_typ	npbhdr	Integer PrepBUFR report type value	Integer
hdr_irpt_typ	npbhdr	Integer input report type value	Integer
hdr_inst_typ	npbhdr	Integer instrument type value	Integer
hdr_typ_table	nhdr_typ, mxstr2	Lookup table containing unique message type strings	String
hdr_sid_table	nhdr_sid, mxstr2	Lookup table containing unique station ID strings	String
hdr_vld_table	nhdr_vld, mxstr	Lookup table containing unique valid time strings in YYYYMMDD_HHMMSS UTC format	Datetime String
obs_qty_table	nobs_qty, mxstr	Lookup table containing unique quality control strings	String
obs_var	obs_var_num, mxstr	Lookup table containing unique observation variable names	String
obs_unit	obs_var_num, mxstr2	Lookup table containing a units string for the unique observation variable names in obs_var	String
obs_desc	obs_var_num, mxstr3	Lookup table containing a description string for the unique observation variable names in obs_var	String

7.2 ASCII2NC Tool

This section describes how to run the ASCII2NC tool. The ASCII2NC tool is used to reformat ASCII point observations into the NetCDF format expected by the Point-Stat tool. For those users wishing to verify against point observations that are not available in PrepBUFR format, the ASCII2NC tool provides a way of incorporating those observations into MET. If the ASCII2NC tool is used to perform a reformatting step, no configuration file is needed. However, for more complex processing, such as summarizing time series observations, a configuration file may be specified. For details on the configuration file options, see Section 5 and example configuration files distributed with the MET code.

While initial versions of the ASCII2NC tool only supported a simple 11 column ASCII point observation format, support for several additional formats has been added. It currently supports point observation data in the following formats:

- Default 11 column MET point observation format, as described in Table 7.4
- little_r format
- SURFace RADiation (SURFRAD) and Integrated Surface Irradiance Study (ISIS) formats
- Western Wind and Solar Integration Study (WWSIS) format. WWSIS data are available by request from National Renewable Energy Laboratory (NREL) in Boulder, CO.
- AirNow DailyData_v2, AirNow HourlyData, and AirNow HourlyAQObs formats. See the *MET_AIRNOW_STATIONS* (page 45) environment variable.
- National Data Buoy (NDBC) Standard Meteorological Data format. See the *MET_NDBC_STATIONS* (page 45) environment variable.
- International Soil Moisture Network (ISMN) Data format.
- International Arctic Buoy Programme (IABP) Data format.
- AErosol RObotic NEtwork (AERONET) versions 2 and 3 format.
- U.S. Climate Reference Network (USCRN) Quality Controlled Datasets, where the specific data format is determined by the prefix and suffix of the input file name.
- Python embedding of point observations, as described in Section 38.4.2. See example below in Section 7.2.2.

The default ASCII point observation format consists of one row of data per observation value. Each row of data consists of 11 columns as shown in Table 7.4.

Table 7.4: Input MET ascii2nc point observation format

Column	Name	Description
1	Message_Type	Text string containing the observation message type as described in the previous section on the PB2NC tool (max 40 characters).
2	Station_ID	Text string containing the station id (max 40 characters).
3	Valid_Time	Text string containing the observation valid time in YYYYMMDD_HHMMSS format.
4	Lat	Latitude in degrees north of the observing location.
5	Lon	Longitude in degrees east of the observation location.
6	Elevation	Elevation in msl of the observing location.
7	GRIB_Code or Variable_Name	Integer GRIB code value or variable name (max 40 characters) corresponding to this observation type.
8	Level	Pressure level in hPa or accumulation interval in hours for the observation value.
9	Height	Height in msl or agl of the observation value.
10	QC_String	Quality control value (max 16 characters).
11	Observation_Value	Observation value in units consistent with the GRIB code definition.

7.2.1 ascii2nc Usage

Once the ASCII point observations have been formatted as expected, the ASCII file is ready to be processed by the ASCII2NC tool. The usage statement for ASCII2NC tool is shown below:

```
Usage: ascii2nc
       input1 ... inputn
       netcdf_file
       [-inputrx reg_exp]
       [-format type]
       [-config file]
       [-valid_beg time]
       [-valid_end time]
       [-mask_grid string]
       [-mask_poly file]
       [-mask_sid file|list]
       [-log file]
       [-v level]
       [-compress level]
```

ascii2nc has two required arguments and can take several optional ones.

7.2.1.1 Required Arguments for ascii2nc

1. The **input1 .. inputn** argument defines one or more sources of formatted ASCII point observation files to be processed. Each input is the path to:
   - a regular file
   - an ASCII file list, as described in Section 4.1.1
   - a top-level diretory to be recursively searched for files matching the **-inputrx reg_exp**, described below

If using Python embedding with the **-format python** option, specify inputs as quoted strings containing the Python script to be run followed by any command line arguments for that script.

2. The **netcdf_file** argument is the NetCDF output file to be written.

7.2.1.2 Optional Arguments for ascii2nc

3. The **-inputrx reg_exp** option overrides the default regular expression (.*) when searching directories for input files. Only files matching this regular expression will be processed.
4. The **-format type** option may be set to "met_point", "little_r", "surfrad", "wwsis", "airnowhourlyaqobs", "airnowhourly", "airnowdaily_v2", "ndbc_standard", "ismn", "iabp", "aeronet", "aeronetv2", "aeronetv3", "uscrn", or "python". If passing in ISIS data, use the "surfrad" format flag.
5. The **-config file** option is the configuration file for generating time summaries.
6. The **-valid_beg** time option in YYYYMMDD[_HH[MMSS]] format sets the beginning of the retention time window.
7. The **-valid_end** time option in YYYYMMDD[_HH[MMSS]] format sets the end of the retention time window.
8. The **-mask_grid** string option is a named grid or a gridded data file to filter the point observations spatially.
9. The **-mask_poly** file option is a polyline masking file to filter the point observations spatially.
10. The **-mask_sid** file|list option is a station ID masking file or a comma-separated list of station ID's to filter the point observations spatially. See the description of the "sid" entry in Section 5.
11. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
12. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.
13. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression

level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the ascii2nc calling sequence is shown below:

```
ascii2nc sample_ascii_obs.txt \
sample_ascii_obs.nc
```

In this example, the ASCII2NC tool will reformat the input **sample_ascii_obs.txt file** into NetCDF format and write the output to a file named **sample_ascii_obs.nc**.

7.2.2 ascii2nc Configuration File

The default configuration file for the ASCII2NC tool named **Ascii2NcConfig_default** can be found in the installed *share/met/config* directory. It is recommended that users make a copy of this file prior to modifying its contents.

The ASCII2NC configuration file is optional and only necessary when defining time summaries or message type mapping for little_r data. The contents of the default ASCII2NC configuration file are described below.

```
version = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
time_summary = { ... }
```

The **time_summary** feature was implemented to allow additional processing of observations with high temporal resolution, such as SURFRAD data every 5 minutes. This option is described in Section 7.1.2.

```
message_type_map = [
   { key = "FM-12 SYNOP";  val = "ADPSFC"; },
   { key = "FM-13 SHIP";   val = "SFCSHP"; },
   { key = "FM-15 METAR";  val = "ADPSFC"; },
   { key = "FM-18 BUOY";   val = "SFCSHP"; },
   { key = "FM-281 QSCAT"; val = "ASCATW"; },
   { key = "FM-32 PILOT";  val = "ADPUPA"; },
   { key = "FM-35 TEMP";   val = "ADPUPA"; },
   { key = "FM-88 SATOB";  val = "SATWND"; },
   { key = "FM-97 ACARS";  val = "AIRCFT"; }
];
```

This entry is an array of dictionaries, each containing a **key** string and **val** string which define a mapping of input strings to output message types. This mapping is currently only applied when converting input little_r report types to output message types.

7.2.3 ascii2nc Output

The NetCDF output of the ASCII2NC tool is structured in the same way as the output of the PB2NC tool described in Section 7.1.3.

"obs_vid" variable is replaced with "obs_gc" when the GRIB code is given instead of the variable names. In this case, the global attribute "use_var_id" does not exist or set to false (use_var_id = "false" ;). Three variables (obs_var, obs_units, and obs_desc) related with variable names are not added.

7.3 MADIS2NC Tool

This section describes how to run the MADIS2NC tool. The MADIS2NC tool is used to reformat Meteorological Assimilation Data Ingest System (MADIS) point observations into the NetCDF format expected by the MET statistics tools. An optional configuration file controls the processing of the point observations. The MADIS2NC tool supports many of the MADIS data types, as listed in the usage statement below. Support for additional MADIS data types may be added in the future based on user feedback.

7.3.1 madis2nc Usage

The usage statement for the MADIS2NC tool is shown below:

```
Usage: madis2nc
       madis_file [madis_file2 ... madis_filen]
       out_file
       -type str
       [-config file]
       [-qc_dd list]
       [-lvl_dim list]
       [-rec_beg n]
       [-rec_end n]
       [-mask_grid string]
       [-mask_poly file]
       [-mask_sid file|list]
       [-log file]
       [-v level]
       [-compress level]
```

madis2nc has required arguments and can also take optional ones.

7.3.1.1 Required Arguments for madis2nc

1. The **madis_file** argument is one or more input MADIS point observation files to be processed.
2. The **out_file** argument is the NetCDF output file to be written.
3. The argument **-type str** is a type of MADIS observations (metar, raob, profiler, maritime, mesonet or acarsProfiles).

7.3.1.2 Optional Arguments for madis2nc

4. The **-config file** option specifies the configuration file to generate summaries of the fields in the ASCII files.
5. The **-qc_dd list** option specifies a comma-separated list of QC flag values to be accepted(Z,C,S,V,X,Q,K,G,B).
6. The **-lvl_dim list** option specifies a comma-separated list of vertical level dimensions to be processed.
7. To specify the exact records to be processed, the **-rec_beg n** specifies the index of the first MADIS record to process and **-rec_end n** specifies the index of the last MADIS record to process. Both are zero-based.
8. The **-mask_grid string** option specifies a named grid or a gridded data file for filtering the point observations spatially.
9. The **-mask_poly file** option defines a polyline masking file for filtering the point observations spatially.
10. The **-mask_sid file|list** option is a station ID masking file or a comma-separated list of station ID's for filtering the point observations spatially. See the description of the "sid" entry in Section 5.
11. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
12. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.
13. The **-compress level** option specifies the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the madis2nc calling sequence is shown below:

```
madis2nc sample_madis_obs.nc \
sample_madis_obs_met.nc -log madis.log -v 3
```

In this example, the MADIS2NC tool will reformat the input sample_madis_obs.nc file into NetCDF format and write the output to a file named sample_madis_obs_met.nc. Warnings and error messages will be written to the madis.log file, and the verbosity level of logging is three.

7.3.2 madis2nc Configuration File

The default configuration file for the MADIS2NC tool named **Madis2NcConfig_default** can be found in the installed *share/met/config* directory. It is recommended that users make a copy of this file prior to modifying its contents.

The MADIS2NC configuration file is optional and only necessary when defining time summaries. The contents of the default MADIS2NC configuration file are described below.

```
version = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
time_summary = { ... }
```

The **time_summary** dictionary is described in Section 7.1.2.

```
grib_var_map = [
   { key = "1"  ;   val =  "PRES,Pa"      ; },  // Station Pressure
   { key = "2"  ;   val = "PRMSL,Pa"      ; },  // Sea Level Pressure
   { key = "7"  ;   val =   "HGT,gpm"     ; },  // Height
   { key = "11" ;   val =   "TMP,K"       ; },  // Temperature
   { key = "15" ;   val =  "TMAX,K"       ; },  // Maximum Temperature
   { key = "16" ;   val =  "TMIN,K"       ; },  // Minimum Temperature
   { key = "17" ;   val =   "DPT,K"       ; },  // Dewpoint
   { key = "20" ;   val = "VISIB,W/m^2"   ; },  // Visibility
   { key = "31" ;   val =  "WDIR,deg"     ; },  // Wind Direction
   { key = "32" ;   val =  "WIND,m/s"     ; },  // Wind Speed
   { key = "33" ;   val =  "UGRD,m/s"     ; },  // Write U-component of wind
   { key = "34" ;   val =  "VGRD,m/s"     ; },  // Write V-component of wind
   { key = "52" ;   val =    "RH,%"       ; },  // Relative Humidity
   { key = "54" ;   val =  "PWAT,kg/m^2"  ; },  // Precipitable Water
   { key = "59" ;   val = "PRATE,kg/m^2/s"; },  // Precipitation Rate
   { key = "61" ;   val =  "APCP,kg/m^2"  ; },  // Precipitation
   { key = "66" ;   val =  "SNOD,m"       ; },  // Snow Cover
   { key = "80" ;   val =  "WTMP,K"       ; },  // Sea Surface Temperature
   { key = "85" ;   val = "TSOIL,K"       ; },  // Soil Temperature
   { key = "180";   val =  "GUST,m/s"     ; },  // Wind Gust
   { key = "250";   val =  "SWHR,K/s"     ; }   // Solar Radiation
];
```

The GRIB code mappings for variable names and units are defined in the **grib_var_map** dictionary within **Madis2NcConfig_default**. In this mapping, each key is a GRIB code, and the corresponding value is a pair: the first element is the variable name, and the second is the unit.

7.3.3 madis2nc Output

The NetCDF output of the MADIS2NC tool is structured in the same way as the output of the PB2NC tool described in Section 7.1.3.

"obs_vid" variable is replaced with "obs_gc" when the GRIB code is given instead of the variable names. In this case, the global attribute "use_var_id" does not exist or set to false (use_var_id = "false" ;). Three variables (obs_var, obs_units, and obs_desc) related with variable names are not added.

Starting from MET version 12.2.0, the GRIB codes were replaced with variable names which are defined in the **grib_var_map** dictionary within **Madis2NcConfig_default**.

7.4 LIDAR2NC Tool

The LIDAR2NC tool creates a NetCDF point observation file from a CALIPSO HDF data file. Not all of the data present in the CALIPSO file is reproduced in the output, however. Instead, the output focuses mostly on information about clouds (as opposed to aerosols) as seen by the satellite along its ground track.

7.4.1 lidar2nc Usage

The usage statement for LIDAR2NC tool is shown below:

```
Usage: lidar2nc
       lidar_file
       -out out_file
       [-log file]
       [-v level]
       [-compress level]
```

Unlike most of the MET tools, lidar2nc does not use a config file. Currently, the options needed to run lidar2nc are not complex enough to require one.

7.4.1.1 Required Arguments for lidar2nc

1. The **lidar_file** argument is the input HDF lidar data file to be processed. Currently, CALIPSO files are supported but support for additional file types will be added in future releases.
2. The **out_file** argument is the NetCDF output file to be written.

7.4.1.2 Optional Arguments for lidar2nc

3. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.

4. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

5. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

7.4.2 lidar2nc Output

Each observation type in the lidar2nc output is assigned a GRIB code. These are outlined in Table 7.5. GRIB codes were assigned to these fields arbitrarily, with GRIB codes in the 600s denoting individual bit fields taken from the feature classification flag field in the CALIPSO file.

We will not give a detailed description of each CALIPSO data product that lidar2nc reads. Users should refer to existing CALIPSO documentation for this information. We will, however, give some explanation of how the cloud layer base and top information is encoded in the lidar2nc NetCDF output file.

Layer_Base gives the elevation in meters above ground level of the cloud base for each cloud level at each observation location. Similarly, **Layer_Top** gives the elevation of the top of each cloud layer. Note that if there are multiple cloud layers at a particular location, then there will be more than one base (or top) given for that location. For convenience, **Min_Base** and **Max_Top** give, respectively, the base elevation for the bottom cloud layer, and the top elevation for the top cloud layer. For these data types, there will be only one value per observation location regardless of how many cloud layers there are at that location.

Table 7.5: lidar2nc GRIB codes and their meaning, units, and abbreviations

GRIB Code	Meaning	Units	Abbreviation
500	Number of Cloud Layers	NA	NLayers
501	Cloud Layer Base AGL	m	Layer_Base
502	Cloud Layer Top AGL	m	Layer_Top
503	Cloud Opacity	%	Opacity
504	CAD Score	NA	CAD_Score
505	Minimum Cloud Base AGL	m	Min_Base
506	Maximum Cloud Top AGL	m	Max_Top
600	Feature Type	NA	Feature_Type
601	Ice/Water Phase	NA	Ice_Water_Phase
602	Feature Sub-Type	NA	Feature_Sub_Type
603	Cloud/Aerosol/PSC Type QA	NA	Cloud_Aerosol_PSC_Type_QA
604	Horizontal Averaging	NA	Horizontal_Averaging

7.5 IODA2NC Tool

This section describes the IODA2NC tool which is used to reformat IODA (Interface for Observation Data Access) point observations from the Joint Center for Satellite Data Assimilation (JCSDA) into the NetCDF format expected by the MET statistics tools. An optional configuration file controls the processing of the point observations. The IODA2NC tool reads NetCDF point observation files created by the IODA Converters. Support for interfacing with data from IODA may be added in the future based on user feedback.

7.5.1 ioda2nc Usage

The usage statement for the IODA2NC tool is shown below:

```
Usage: ioda2nc
       ioda_file
       netcdf_file
       [-config config_file]
       [-obs_var var]
       [-iodafile ioda_file]
       [-valid_beg time]
       [-valid_end time]
       [-nmsg n]
       [-log file]
       [-v level]
       [-compress level]
```

ioda2nc has required arguments and can also take optional ones.

7.5.1.1 Required Arguments for ioda2nc

1. The **ioda_file** argument is an input IODA NetCDF point observation file to be processed.
2. The **netcdf_file** argument is the NetCDF output file to be written.

7.5.1.2 Optional Arguments for ioda2nc

3. The **-config config_file** is a IODA2NCConfig file to filter the point observations and define time summaries.
4. The **-obs_var var_list** setting is a comma-separated list of variables to be saved from input the input file (by defaults, saves "all").
5. The **-iodafile ioda_file** option specifies additional input IODA observation files to be processed.
6. The **-valid_beg time** and **-valid_end time** options in YYYYMMDD[_HH[MMSS]] format overrides the retention time window from the configuration file.
7. The **-nmsg n** indicates the number of IODA records to process.

8. The **-log** file option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.

9. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

10. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the ioda2nc calling sequence is shown below:

```
ioda2nc \
ioda.NC001007.2020031012.nc ioda2nc.2020031012.nc \
-config IODA2NCConfig -v 3 -lg run_ioda2nc.log
```

In this example, the IODA2NC tool will reformat the data in the input ioda.NC001007.2020031012.nc file and write the output to a file named ioda2nc.2020031012.nc. The data to be processed is specified by IODA2NCConfig, log messages will be written to the ioda2nc.log file, and the verbosity level is three.

7.5.2 ioda2nc Configuration File

The default configuration file for the IODA2NC tool named **IODA2NcConfig_default** can be found in the installed *share/met/config* directory. It is recommended that users make a copy of this file prior to modifying its contents.

The IODA2NC configuration file is optional and only necessary when defining filtering the input observations or defining time summaries. The contents of the default IODA2NC configuration file are described below.

```
obs_window = { beg  = -5400; end  = 5400; }
mask       = { grid = "";    poly = "";   }
tmp_dir    = "/tmp";
version    = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
message_type           = [];
message_type_group_map = [];
message_type_map       = [];
station_id             = [];
elevation_range        = { ... };
level_range            = { ... };
```

(continues on next page)

(continued from previous page)

```
obs_var                = [];
quality_mark_thresh    = 0;
time_summary           = { ... }
```

The configuration options listed above are supported by other point observation pre-processing tools and are described in Section 7.1.2.

```
obs_name_map = [];
```

This entry is an array of dictionaries, each containing a **key** string and **val** string which define a mapping of input IODA variable names to output variable names. The default IODA map, obs_var_map, is appended to this map.

```
metadata_map = [
   { key = "message_type"; val = "msg_type,station_ob"; },
   { key = "station_id";   val = "station_id,report_identifier"; },
   { key = "pressure";     val = "air_pressure,pressure"; },
   { key = "height";       val = "height,height_above_mean_sea_level"; },
   { key = "elevation";    val = "elevation,station_elevation"; },
   { key = "nlocs";        val = "Location"; }
];
```

This entry is an array of dictionaries, each containing a **key** string and **val** string which define a mapping of metadata for IODA data files. The "nlocs" is for the dimension name of the locations. The following key can be added: "nstring", "latitude" and "longitude".

```
obs_to_qc_map  = [
   { key = "wind_from_direction"; val = "eastward_wind,northward_wind"; },
   { key = "wind_speed";          val = "eastward_wind,northward_wind"; }
];
```

This entry is an array of dictionaries, each containing a **key** string and **val** string which define a mapping of QC variable name for IODA data files.

```
missing_thresh = [ <=-1e9, >=1e9, ==-9999 ];
```

The **missing_thresh** option is an array of thresholds. Any data values which meet any of these thresholds are interpreted as being bad, or missing, data.

7.5.3 ioda2nc Output

The NetCDF output of the IODA2NC tool is structured in the same way as the output of the PB2NC tool described in Section 7.1.3.

7.6 Point2Grid Tool

The Point2Grid tool reads point observations from a MET NetCDF point obseravtion file, via python embedding, or from GOES NetCDF input files (especially, Aerosol Optical Depth) and creates a gridded NetCDF file. Future development may add support for additional input types.

7.6.1 point2grid Usage

The usage statement for the Point2Grid tool is shown below:

```
Usage: point2grid
       input_filename
       to_grid
       output_filename
       -field string
       [-config file]
       [-goes_qc flags]
       [-adp adp_filename]
       [-method type]
       [-gaussian_dx n]
       [-gaussian_radius n]
       [-prob_cat_thresh string]
       [-vld_thresh n]
       [-name list]
       [-log file]
       [-v level]
       [-compress level]
```

7.6.1.1 Required Arguments for point2grid

1. The **input_filename** argument indicates the name of the input file to be processed. The input can be a MET NetCDF point observation file generated by other MET tools or a GOES NetCDF AOD dataset. Python embedding for point observations is also supported, as described in Section 38.4.2.

The MET point observation NetCDF file name as **input_filename** argument is equivalent with "PYTHON_NUMPY=MET_BASE/python/examples/read_met_point_obs.py netcdf_filename".

2. The **to_grid** argument defines the output grid as: (1) a named grid, (2) the path to a gridded data file, or (3) an explicit grid specification string.

3. The **output_filename** argument is the name of the output NetCDF file to be written.

4. The **-field** string argument is a string that defines the data to be regridded. It may be used multiple times. If **-adp** option is given (for GOES AOD data), the name consists with the variable name from the input data file and the variable name from ADP data file (for example, "AOD_Smoke" or "AOD_Dust": getting AOD variable from the input data and applying smoke or dust variable from ADP data file).

7.6.1.2 Optional Arguments for point2grid

5. The **-config** file option is the configuration file to be used.
6. The **-goes_qc** flags option specifies a comma-separated list of quality control (QC) flags, for example "0,1". Only used if grid_mapping is set to "goes_imager_projection" and the QC variable exists. Note that the older **-qc** option name is also supported.
7. The **-adp adp_filename** option provides an additional Aerosol Detection Product (ADP) information on aerosols, dust, and smoke. This option is ignored if the requested variable is not GOES AOD ("AOD_Dust" or "AOD_Smoke"). The gridded data is filtered by the presence of dust/smoke. If -goes_qc options are given, it's applied to QC of dust/smoke, too (First filtering with AOD QC values and the second filtering with dust/smoke QC values).
8. The **-method type** option specifies the regridding method. The default method is UW_MEAN.
9. The **-gaussian_dx n** option specifies the distance interval for Gaussian smoothing. The default is 81.271 km. Only used if the method is GAUSSIAN or MAXGAUSS.
10. The **-gaussian_radius** n option specifies the radius of influence for Gaussian interpolation. The default is 120. Only used if the method is GAUSSIAN or MAXGAUSS.
11. The **-prob_cat_thresh string** option sets the threshold to compute the probability of occurrence. The default is set to disabled. This option is relevant when calculating practically perfect forecasts.
12. The **-vld_thresh n** option sets the required ratio of valid data for regridding. The default is 0.5.
13. The **-name list** option specifies a comma-separated list of output variable names for each field specified.
14. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
15. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.
16. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

Only 4 interpolation methods are applied to the field variables; MIN/MAX/MEDIAN/UW_MEAN. The GAUSSIAN method is applied to the probability variable only. Unlike regrad_data_plane, MAX method is applied to the file variable and Gaussian method to the probability variable with the MAXGAUSS method. If the probability variable is not requested, MAXGAUSS method is the same as MAX method.

For the GOES-East and GOES-West data, computing the latitude and longitude pixel locations is time consuming. The computed lat/lon coordinates are saved to a temporary NetCDF file, as described in `Contributor's Guide Section %s`. This step can be skipped in subsequent runs if the path to the coordinate file is specified with the MET_GEOSTATIONARY_DATA environment variable. The mapping of GOES pixel locations to target grid cells is saved to the MET_TMP_DIR directory and read in subsequent runs to reduce execution time. Once this file is created, the MET_GEOSTATIONARY_DATA is ignored. The grid mapping file should be deleted manually in order to apply a new MET_GEOSTATIONARY_DATA environment variable or to regenerate the grid mapping file. An example of calling point2grid to process GOES-East AOD data is shown below:

```
point2grid \
   OR_ABI-L2-AODC-M3_G16_s20181341702215_e20181341704588_c20181341711418.nc \
   G212 \
   regrid_data_plane_GOES-16_AOD_TO_G212.nc \
   -field 'name="AOD"; level="(*,*)";' \
   -goes_qc 0,1,2 \
   -method MAX
```

When processing GOES data, the **-goes_qc** option may also be used to specify the acceptable quality control flag values. The example above regrids the GOES-East AOD values to NCEP Grid number 212 (which QC flags are high, medium, and low), writing to the output the maximum AOD value falling inside each grid box.

The grid name or the grid definition can be given with the -field option when the grid information is missing from the input NetCDF file for the latitude_longitude projection. The latitude and longitude variable names should be defined by the user, and the grid information from the set_attr_grid is ignored in this case except nx and ny.

```
point2grid \
   iceh.2018-01-03.c00.tlat_tlon.nc \
   G231 \
   point2grid_cice_to_G231.nc \
   -config Point2GridConfig_tlat_tlon \
   -field 'name="hi_d"; level="(0,*,*)"; set_attr_grid="latlon 1440 1080 -79.80672 60.28144␣
↪0.04 0.04";'
```

Listed below is an example of using Python embedding to pass point observations as input to point2grid:

```
point2grid \
   'PYTHON_NUMPY=MET_BASE/python/examples/read_met_point_obs.py ascii2nc_edr_hourly.20130827.
↪nc' \
   G212 \
   python_gridded_ascii_python.nc -config Point2GridConfig_edr \
   -field 'name="200"; level="*"; valid_time="20130827_205959";' \
   -method MAX
```

Please refer to Appendix F, Section 38 for more details about Python embedding in MET.

7.6.2 point2grid Output

The point2grid tool will output a gridded NetCDF file containing the following:

1. Latitude
2. Longitude
3. The variable specified in the -field string regridded to the grid defined in the **to_grid** argument.
4. The count field which represents the number of point observations that were included calculating the value of the variable at that grid cell.
5. The mask field which is a binary field representing the presence or lack thereof of point observations at that grid cell. A value of "1" indicates that there was at least one point observation within the bounds of that grid cell and a value of "0" indicates the lack of point observations at that grid cell.
6. The probability field which is the probability of the event defined by the **-prob_cat_thresh** command line option. The output variable name includes the threshold used to define the probability. Ranges from 0 to 1.
7. The probability mask field which is a binary field that represents whether or not there is probability data at that grid point. Can be either "0" or "1" with "0" meaning the probability value does not exist and a value of "1" meaning that the probability value does exist.

For MET observation input and CF complaint NetCDF input with 2D time variable: The latest observation time within the target grid is saved as the observation time. If the "valid_time" is configured at the configuration file, the valid_time from the configuration file is saved into the output file.

7.6.3 point2grid Configuration File

The default configuration file for the point2grid tool named **Point2GridConfig_default** can be found in the installed *share/met/config* directory. It is recommended that users make a copy of this file prior to modifying its contents.

The point2grid configuration file is optional and only necessary when defining the variable name instead of GRIB code or filtering by time. The contents of the default point2grid configuration file are described below.

```
obs_window = { beg = -5400; end =  5400; }
message_type    = [];
obs_quality_inc = [];
obs_quality_exc = [];
version = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
valid_time = "YYYYMMDD_HHMMSS";
```

This entry is a string to override the observation time into the output and to filter observation data by time.

```
var_name_map = [
   { key = "1";     val = "PRES"; },        // GRIB: Pressure
   { key = "2";     val = "PRMSL"; },       // GRIB: Pressure reduced to MSL
   { key = "7";     val = "HGT"; },         // GRIB: Geopotential height
   { key = "11";    val = "TMP"; },         // GRIB: Temperature
   { key = "15";    val = "TMAX"; },        // GRIB: Max Temperature
   ...
   { key = "lat_vname"; val = "NLAT"; },    // NetCDF latitude variable name
   { key = "lon_vname"; val = "NLON"; },    // NetCDF longitude varialbe name
   ...
]
```

This entry is an array of dictionaries, each containing a **GRIB code** string and matching **variable name** string which define a mapping of GRIB code to the output variable names. The latitude and longitude variables for NetCDF input can be overridden by the configurations. There are two special keys, **lat_vname** and **lon_vname**, are applied to the NetCDF input, not for a GRIB code.

7.7 Point NetCDF to ASCII Python Utility

As a tool for debugging, a utility script called print_pointnc2ascii.py is included for users. This script reads the MET point NetCDF file format and returns an ASCII representation to the screen, with either space or comma delimiting. Optionally, the user can request that the output be written to a file.

The script can be found at:

```
MET_BASE/python/utility/print_pointnc2ascii.py
```

For how to use the script, issue the command:

```
python3 MET_BASE/python/utility/print_pointnc2ascii.py -h
```

7.8 IABP retrieval Python Utilities

International Arctic Buoy Programme (IABP) Data is one of the data types supported by ascii2nc. A utility script that pulls all this data from the web and stores it locally, called get_iabp_from_web.py is included. This script accesses the appropriate webpage and downloads the ascii files for all buoys. It is straightforward, but can be time intensive as the archive of this data is extensive and files are downloaded one at a time.

The script can be found at:

```
MET_BASE/python/utility/get_iabp_from_web.py
```

For how to use the script, issue the command:

```
python3 MET_BASE/python/utility/get_iabp_from_web.py -h
```

Another IABP utility script is included for users, to be run after all files have been downloaded using get_iabp_from_web.py. This script examines all the files and lists those files that contain entries that fall within a user specified range of days. It is called find_iabp_in_timerange.py.

The script can be found at:

```
MET_BASE/python/utility/find_iabp_in_timerange.py
```

For how to use the script, issue the command:

```
python3 MET_BASE/python/utility/find_iabp_in_timerange.py -h
```

Chapter 8

Re-Formatting of Gridded Fields

Several MET tools exist for the purpose of reformatting gridded fields, and they are described in this section. These tools are represented by the reformatting column of MET flowchart depicted in Figure 1.1.

8.1 Pcp-Combine Tool

This section describes the Pcp-Combine tool which summarizes data across multiple input gridded data files and writes the results to a single NetCDF output file. It is often used to modify precipitation accumulation intervals in the forecast and/or observation datasets to make them comparable. However it can also be used to derive summary fields, such as daily min/max temperature or average precipitation rate.

The Pcp-Combine tool supports four types of commands ("sum", "add", "subtract", and "derive") which may be run on any gridded data files supported by MET.

1. The "sum" command is the default command and therefore specifying "-sum" on the command line is optional. Using the sum arguments described below, Pcp-Combine searches the input directories ("-pcpdir" option) for data that matches the requested time stamps and accumulation intervals. Pcp-Combine only considers files from the input data directory which match the specified regular expression ("-pcprx" option). While "sum" searches for matching data, all the other commands are run on the explicit set of input files specified.
2. The "add" command reads the requested data from the input data files and adds them together.
3. The "subtract" command reads the requested data from exactly two input files and computes their difference.
4. The "derive" command reads the requested data from the input data files and computes the requested summary fields.

By default, the Pcp-Combine tool processes data for **APCP**, the GRIB string for accumulated precipitation. When requesting data using time strings (i.e. [HH]MMSS), Pcp-Combine searches for accumulated precipitation for that accumulation interval. Alternatively, use the "-field" option to process fields other than **APCP** or for non-GRIB files. The "-field" option may be used multiple times to process multiple fields in a single run. Since the Pcp-Combine tool does not support automated regridding, all input data must be on the same grid. In general the input files should have the same initialization time unless the user has indicated that it

should ignore the initialization time for the "sum" command. The "subtract" command produces a warning when the input initialization times differ or the subtraction results in a negative accumulation interval.

8.1.1 pcp_combine Usage

The usage statement for the Pcp-Combine tool is shown below:

```
Usage: pcp_combine
       [-sum] sum_args |
       -add input_files |
       -subtract input_files |
       -derive stat_list input_files
       out_file
       [-field string]
       [-name list]
       [-input_thresh n]
       [-vld_thresh n]
       [-log file]
       [-v level]
       [-compress level]
```

The arguments to pcp_combine vary depending on the run command. Listed below are the arguments for the sum command:

```
SUM_ARGS:
      init_time
      in_accum
      valid_time
      out_accum
      out_file
      [-pcpdir path]
      [-pcprx reg_exp]
```

The add, subtract, and derive commands all require that the input files be explicitly listed:

```
INPUT_FILES:
       file_1 config_str_1 ... file_n config_str_n |
       file_1 ... file_n |
       file_list
```

8.1.1.1 Required Arguments for the pcp_combine

1. The Pcp-Combine tool must be run with exactly one run command (-sum, -add, -subtract, or -derive) with the corresponding additional arguments.
2. The **out_file** argument indicates the name for the NetCDF file to be written.

8.1.1.2 Optional Arguments for pcp_combine

3. The **-field string** option defines the data to be extracted from the input files. Use this option when processing fields other than **APCP** or non-GRIB files. It can be used multiple times and output will be created for each. In general, the field string should include the **name** and **level** of the requested data and be enclosed in single quotes. It is processed as an inline configuration file and may also include data filtering, censoring, and conversion options. For example, use **-field 'name="ACPCP"; level="A6"; convert(x)=x/25.4;'** to read 6-hourly accumulated convective precipitation from a GRIB file and convert from millimeters to inches.
4. The **-name list** option is a comma-separated list of output variable names which override the default choices. If specified, the number of names must match the number of variables to be written to the output file.
5. The **-input_thresh n** option overrides the default required ratio of valid input files. This option does not apply to the -subtract command where exactly two valid inputs are required. The default is 1.0.
6. The **-vld_thresh n** option overrides the default required ratio of valid data at each grid point for an output value to be written. The default is 1.0.
7. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
8. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.
9. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

8.1.1.3 Required Arguments for the pcp_combine Sum Command

1. The **init_time** argument, provided in YYYYMMDD[_HH[MMSS]] format, indicates the initialization time for model data to be summed. Only files found with this initialization time will be processed. If combining observation files, Stage II or Stage IV data for example, the initialization time is not applicable. Providing a string of all zeros (00000000_000000) indicates that all files, regardless of initialization time should be processed.

2. The **in_accum** argument, provided in HH[MMSS] format, indicates the accumulation interval of the model or observation gridded files to be processed. This value must be specified, since a model output file may contain multiple accumulation periods for precipitation in a single file. The argument indicates which accumulation period to extract.
3. The **valid_time** argument, in YYYYMMDD[_HH[MMSS]] format, indicates the desired valid time to which the accumulated precipitation is to be summed.
4. The **out_accum** argument, in HH[MMSS] format, indicates the desired total accumulation period to be summed.

8.1.1.4 Optional Arguments for pcp_combine Sum Command

5. The **-pcpdir path** option indicates the directories in which the input files reside. The contents of "**path**" will override the default setting. This option may be used multiple times and can accept multiple arguments, supporting the use of wildcards.
6. The **-pcprx reg_exp** option indicates the regular expression to be used in matching files in the search directories specified. The contents of "reg_exp" will override the default setting that matches all file names. If the search directories contain a large number of files, the user may specify that only a subset of those files be processed using a regular expression which will speed up the run time.

8.1.1.5 Required Arguments for the pcp_combine Derive Command

1. The "derive" run command must be followed by **stat_list** which is a comma-separated list of summary fields to be computed. The **stat_list** may be set to sum, min, max, range, mean, stdev, and vld_count for the sum, minimum, maximum, range (max-min), average, standard deviation, and valid data count fields, respectively.

8.1.1.6 Input Files for pcp_combine Add, Subtract, and Derive Commands

The input files for the add, subtract, and derive command can be specified in one of 3 ways:

1. Use **file_1 config_str_1 ... file_n config_str_n** to specify the full path to each input file followed by a description of the data to be read from it. The **config_str_i** argument describing the data can be a set to a time string in HH[MMSS] format for accumulated precipitation or a full configuration string. For example, use **'name="TMP"; level="P500";'** to process temperature at 500mb.
2. Use **file_1 ... file_n** to specify the list of input files to be processed on the command line. Rather than specifying a separate configuration string for each input file, the "-field" command line option is required to specify the data to be processed.
3. Use **file_list** to specify the name of an ASCII file which contains the paths for the gridded data files to be processed, as described in Section 4.1.1. As in the previous option, the "-field" command line option is required to specify the data to be processed.

An example of the pcp_combine calling sequence is presented below:

Example 1:

```
pcp_combine -sum \
20050807_000000 3 \
20050808_000000 24 \
sample_fcst.nc \
-pcpdir ../data/sample_fcst/2005080700
```

In Example 1, the Pcp-Combine tool will sum the values in model files initialized at 2005/08/07 00Z and containing 3-hourly accumulation intervals of precipitation. The requested valid time is 2005/08/08 00Z with a requested total accumulation interval of 24 hours. The output file is to be named sample_fcst.nc, and the Pcp-Combine tool is to search the directory indicated for the input files.

The Pcp-Combine tool will search for 8 files containing 3-hourly accumulation intervals which meet the criteria specified. It will write out a single NetCDF file containing that 24 hours of accumulation.

A second example of the pcp_combine calling sequence is presented below:

Example 2:

```
pcp_combine -sum \
00000000_000000 1 \
20050808_000000 24 \
sample_obs.nc \
-pcpdir ../data/sample_obs/ST2ml
```

Example 2 shows an example of using the Pcp-Combine tool to sum observation data. The **init_time** has been set to all zeros to indicate that when searching through the files in the precipitation directory, the initialization time should be ignored. The **in_accum** has been changed from 3 to 1 to indicate that the input observation files contain 1-hourly accumulations of precipitation. Lastly, **-pcpdir** provides a different directory to be searched for the input files.

The Pcp-Combine tool will search for 24 files containing 1-hourly accumulation intervals which meet the criteria specified. It will write out a single NetCDF file containing that 24 hours of accumulation.

Example 3:

```
pcp_combine -add input_pinterp.nc 'name="TT"; level="(0,*,*)";' tt_10.nc
```

This command would grab the first level of the TT variable from a pinterp NetCDF file and write it to the output tt_10.nc file.

Example 4:

```
pcp_combine -subtract 2022043018_48.grib2 'name="APCP"; level="A48";' 2022043018_36.grib2
↪'name="APCP"; level="A36";' sample_fcst.nc
```

The Pcp-Combine tool will subtract the 36 hour precipitation accumulations in the file 2022043018_36.grib2 (a 36hr forecast initialized at 2022-04-30 18Z) from the 48 hour accumulations in the file 2022043018_48.grib2 (a 48hr forecast from the same model cycle). This will produce the 12 hour accumulation amounts for the period in between the 36 and 48 hour forecasts. It will write out a single NetCDF file containing that 12 hours of accumulation.

8.1.2 pcp_combine Output

The output NetCDF files contain the requested accumulation intervals as well as information about the grid on which the data lie. That grid projection information will be parsed out and used by the MET statistics tools in subsequent steps. One may use NetCDF utilities such as ncdump or ncview to view the contents of the output file. Alternatively, the MET Plot-Data-Plane tool described in Section 31.1.3 may be run to create a PostScript image of the data.

Each NetCDF file generated by the Pcp-Combine tool contains the dimensions and variables shown in the following two tables.

Table 8.1: NetCDF file dimensions for pcp_combine output.

NetCDF dimension	Description
lat	Dimension of the latitude (i.e. Number of grid points in the North-South direction)
lon	Dimension of the longitude (i.e. Number of grid points in the East-West direction)

Table 8.2: NetCDF variables for pcp_combine output.

NetCDF variable	Dimension	Description	Data Type
lat	lat, lon	Latitude value for each point in the grid	Double
lon	lat, lon	Longitude value for each point in the grid	Double
Name and level of the requested data or value of the -name option.	lat, lon	Data value (i.e. accumulated precipitation) for each point in the grid. The name of the variable describes the name and level and any derivation logic that was applied.	Double

8.2 Regrid-Data-Plane Tool

This section contains a description of running the Regrid-Data-Plane tool. This tool may be run to read data from any gridded file MET supports, interpolate to a user-specified grid, and writes the field(s) out in NetCDF format. The user may specify the method of interpolation used for regridding as well as which fields to regrid. This tool is particularly useful when dealing with GRIB2 and NetCDF input files that need to be regridded. For GRIB1 files, it has also been tested for compatibility with the copygb regridding utility mentioned in Section 3.3.

8.2.1 regrid_data_plane Usage

The usage statement for the regrid_data_plane utility is shown below:

```
Usage: regrid_data_plane
       input_filename
       to_grid
       output_filename
       -field string
       [-method type]
       [-width n]
       [-gaussian_dx n]
       [-gaussian_radius n]
       [-shape type]
       [-vld_thresh n]
       [-name list]
       [-log file]
       [-v level]
       [-compress level]
```

8.2.1.1 Required Arguments for regrid_data_plane

1. The **input_filename** is the gridded data file to be read.
2. The **to_grid** defines the output grid as a named grid, the path to a gridded data file, or an explicit grid specification string.
3. The **output_filename** is the output NetCDF file to be written.
4. The **-field string** may be used multiple times to define the field(s) to be regridded.

8.2.1.2 Optional Arguments for regrid_data_plane

5. The **-method type** option overrides the default regridding method. Default is NEAREST.
6. The **-width n** option overrides the default regridding width. Default is 1. In case of MAXGAUSS method, the width should be the ratio between from_grid and to_grid (for example, 27 if from_grid is 3km and to_grid is 81.271km).
7. The **-gaussian_dx** option overrides the default delta distance for Gaussian smoothing. Default is 81.271. Ignored if not the MAXGAUSS method.
8. The **-gaussian_radius** option overrides the default radius of influence for Gaussian interpolation. Default is 120. Ignored if not the MAXGAUSS method.
9. The **-shape** option overrides the default interpolation shape. Default is SQUARE.
10. The **-vld_thresh n** option overrides the default required ratio of valid data for regridding. Default is 0.5.
11. The **-name list** specifies a comma-separated list of output variable names for each field specified.

12. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.

13. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

14. The **-compress level** option specifies the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

For more details on setting the **to_grid, -method, -width,** and **-vld_thresh** options, see the regrid entry in Section 5. An example of the regrid_data_plane calling sequence is shown below:

```
regrid_data_plane \
input.grb \
togrid.grb \
regridded.nc \
-field 'name="APCP"; level="A6";'
-field 'name="TMP";  level="Z2";' \
-field 'name="UGRD"; level="Z10";' \
-field 'name="VGRD"; level="Z10";' \
-field 'name="HGT";  level="P500";' \
-method BILIN -width 2 -v 1
```

In this example, the Regrid-Data-Plane tool will regrid data from the **input.grb** file to the grid on which the first record of the **togrid.grb** file resides using Bilinear Interpolation with a width of 2 and write the output in NetCDF format to a file named **regridded.nc**. The variables in **regridded.nc** will include 6-hour accumulated precipitation, 2m temperature, 10m U and V components of the wind, and the 500mb geopotential height.

8.2.2 Automated Regridding within Tools

While the Regrid-Data-Plane tool is useful as a stand-alone tool, the capability is also included to automatically **regrid** one or both fields in most of the MET tools that handle gridded data. See the regrid entry in Section 4.5 for a description of the configuration file entries that control automated regridding.

8.3 Shift-Data-Plane Tool

The Shift-Data-Plane tool performs a rigid shift of the entire grid based on user-defined specifications and writes the field(s) out in NetCDF format. This tool was originally designed to account for track error when comparing fields associated with tropical cyclones. The user specifies the latitude and longitude of the source and destination points to define the shift. Both points must fall within the domain and are used to define the X and Y direction grid unit shift. The shift is then applied to all grid points. The user may specify the method of interpolation and the field to be shifted. The effects of topography and land/water masks are ignored.

8.3.1 shift_data_plane Usage

The usage statement for the shift_data_plane utility is shown below:

```
Usage: shift_data_plane
       input_filename
       output_filename
       field_string
       -from lat lon
       -to lat lon
       [-method type]
       [-width n]
       [-shape SHAPE]
       [-log file]
       [-v level]
       [-compress level]
```

shift_data_plane has five required arguments and can also take optional ones.

8.3.1.1 Required Arguments for shift_data_plane

1. The **input_filename** is the gridded data file to be read.
2. The **output_filename** is the output NetCDF file to be written.
3. The **field_string** defines the data to be shifted from the input file.
4. The **-from lat lon** specifies the starting location within the domain to define the shift. Latitude and longitude are defined in degrees North and East, respectively.
5. The **-to lat lon** specifies the ending location within the domain to define the shift. Lat is deg N, Lon is deg E.

8.3.1.2 Optional Arguments for shift_data_plane

6. The **-method type** overrides the default regridding method. Default is NEAREST.
7. The **-width n** overrides the default regridding width. Default is 2.
8. The **-shape SHAPE** overrides the default interpolation shape. Default is SQUARE.
9. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
10. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.
11. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0

from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

For more details on setting the **-method** and **-width** options, see the **regrid** entry in Section 5. An example of the shift_data_plane calling sequence is shown below:

```
shift_data_plane \
nam.grib \
nam_shift_APCP_12.nc \
'name = "APCP"; level = "A12";' \
-from 38.6272  -90.1978 \
-to   40.1717 -105.1092 \
-v 2
```

In this example, the Shift-Data-Plane tool reads 12-hour accumulated precipitation from the **nam.grb** file, applies a rigid shift defined by (38.6272, -90.1978) to (40.1717, -105.1092) and writes the output in NetCDF format to a file named **nam_shift_APCP_12.nc**. These **-from** and **-to** locations result in a grid shift of -108.30 units in the x-direction and 16.67 units in the y-direction.

8.4 MODIS regrid Tool

This section contains a description of running the MODIS regrid tool. This tool may be run to create a NetCDF file for use in other MET tools from MODIS level 2 cloud product from NASA.

8.4.1 modis_regrid Usage

The usage statement for the modis_regrid utility is shown below:

```
Usage: modis_regrid
       -data_file path
       -field name
       -out path
       -scale value
       -offset value
       -fill value
       [-units text]
       [-compress level]
       modis_file
```

modis_regrid has some required arguments and can also take optional ones.

8.4.1.1 Required Arguments for modis_regrid

1. The **-data_file path** argument specifies the data files used to get the grid information.
2. The **-field name** argument specifies the name of the field to use in the MODIS data file.
3. The **-out path** argument specifies the name of the output NetCDF file.
4. The **-scale value** argument specifies the scale factor to be used on the raw MODIS values.
5. The **-offset value** argument specifies the offset value to be used on the raw MODIS values.
6. The **-fill value** argument specifies the bad data value in the MODIS data.
7. The **modis_file** argument is the name of the MODIS input file.

8.4.1.2 Optional Arguments for modis_regrid

8. The **-units text** option specifies the units string in the global attributes section of the output file.
9. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the modis_regrid calling sequence is shown below:

```
modis_regrid -field Cloud_Fraction \
-data_file grid_file \
-out t2.nc \
-units percent \
-scale 0.01 \
-offset 0 \
-fill 127 \
modis_file
```

In this example, the Modis-Regrid tool will process the Cloud_Fraction field from **modis_file** and write it out to the output NetCDF file t2.nc on the grid specified in grid_file using the appropriate scale, offset and fill values.

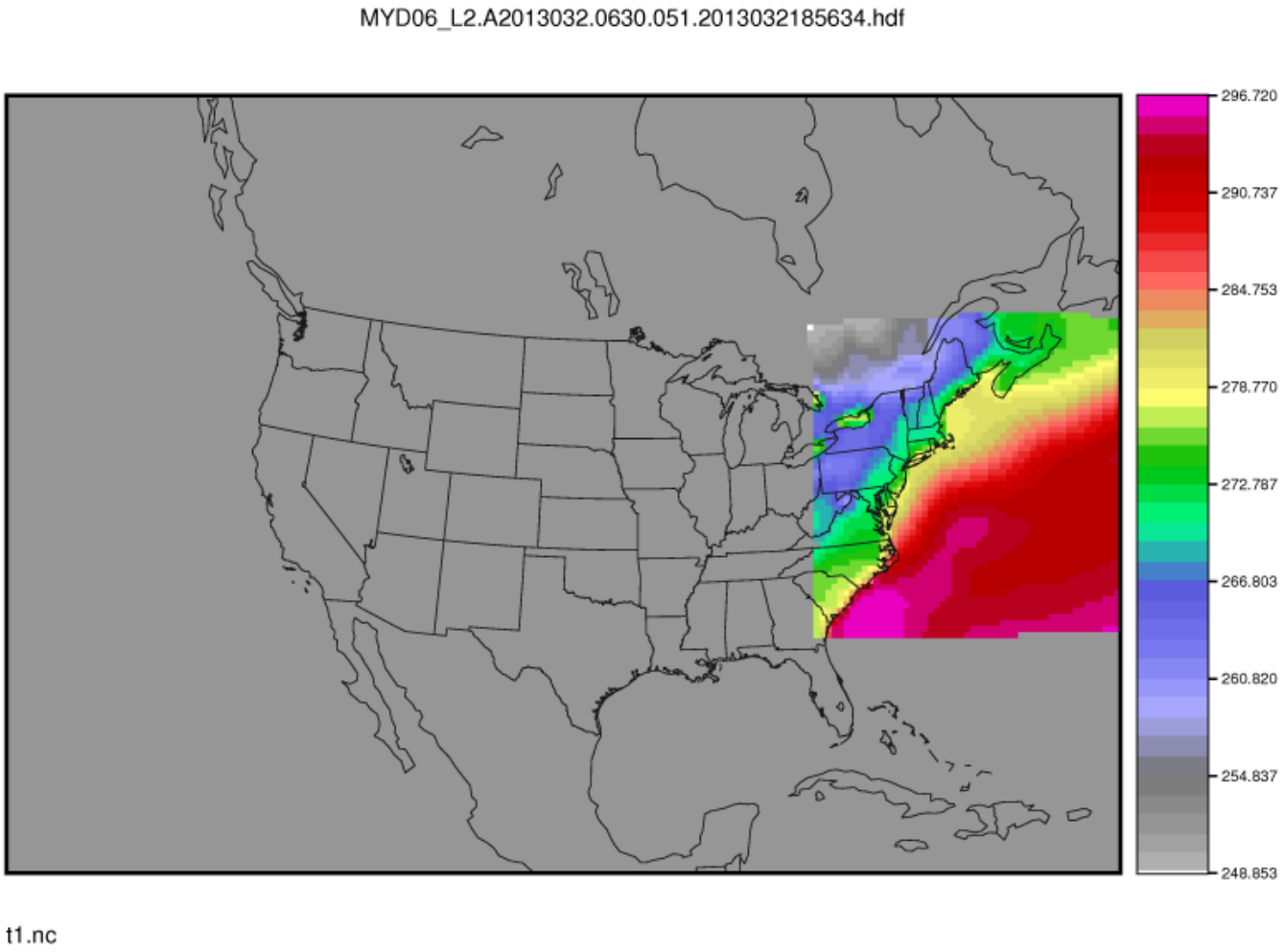

Figure 8.1: Example plot showing surface temperature from a MODIS file.

8.5 WWMCA Tool Documentation

There are two WWMCA tools available. The WWMCA-Plot tool makes a PostScript plot of one or more WWMCA cloud percent files and the WWMCA-Regrid tool regrids binary WWMCA data files and reformats them into NetCDF files that the other MET tools can read. The WWMCA-Regrid tool has been generalized to more broadly support any data stored in the WWMCA binary format.

The WWMCA tools attempt to parse timing and hemisphere information from the file names. They tokenize the filename using underscores (_) and dots (.) and examine each element which need be in no particular order. A string of 10 or more numbers is interpreted as the valid time in YYYYMMDDHH[MMSS] format. The string NH indicates the northern hemisphere while SH indicates the southern hemisphere. While WWMCA data is an analysis and has no forecast lead time, other datasets following this format may. Therefore, a string of 1 to 4 numbers is interpreted as the forecast lead time in hours. While parsing the filename provides default values for this timing information, they can be overridden by explicitly setting their values in the WWMCA-Regrid configuration file.

8.5.1 wwmca_plot Usage

The usage statement for the WWMCA-Plot tool is shown below:

```
Usage: wwmca_plot
       [-outdir path]
       [-max max_minutes]
       [-log file]
       [-v level]
       wwmca_cloud_pct_file_list
```

wmmca_plot has some required arguments and can also take optional ones.

8.5.1.1 Required Arguments for wwmca_plot

1. The **wwmca_cloud_pct_file_list** argument represents one or more WWMCA cloud percent files given on the command line. As with any command given to a UNIX shell, the user can use meta-characters as a shorthand way to specify many filenames. For each input file specified, one output PostScript plot will be created.

8.5.1.2 Optional Arguments for wwmca_plot

2. The **-outdir path** option specifies the directory where the output PostScript plots will be placed. If not specified, then the plots will be put in the current (working) directory.
3. The **-max minutes** option specifies the maximum pixel age in minutes to be plotted.
4. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
5. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

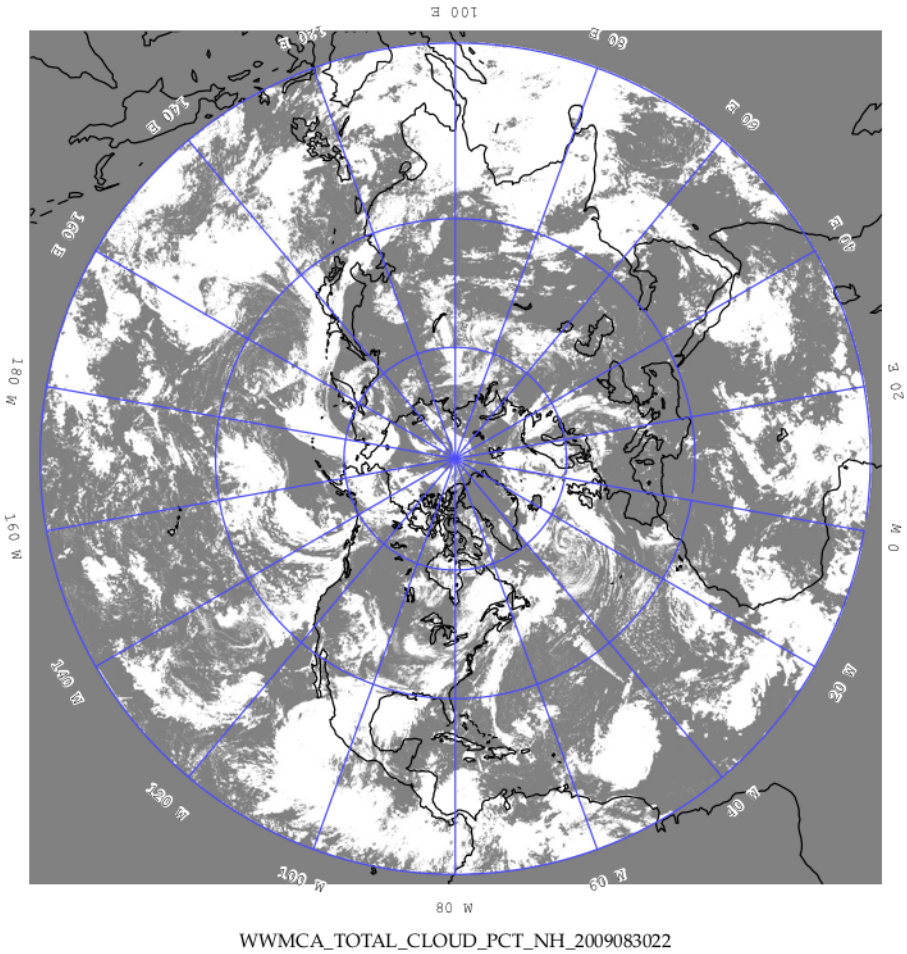

Figure 8.2: Example output of WWMCA-Plot tool.

8.5.2 wwmca_regrid Usage

The usage statement for the WWMCA-Regrid tool is shown below:

```
Usage: wwmca_regrid
       -out filename
       -config filename
       -nh filename [pt_filename]
       -sh filename [pt_filename]
       [-log file]
       [-v level]
       [-compress level]
```

wmmca_regrid has some required arguments and can also take optional ones.

8.5.2.1 Required Arguments for wwmca_regrid

1. The **-out filename** argument specifies the name of the output netCDF file.
2. The **-config filename** argument indicates the name of the configuration file to be used. The contents of the configuration file are discussed below.
3. The **-nh filename [pt_filename]** argument specifies the northern hemisphere WWMCA binary file and, optionally, may be followed by a binary pixel age file. This switch is required if the output grid includes any portion of the northern hemisphere.
4. The **-sh filename [pt_filename]** argument specifies the southern hemisphere WWMCA binary file and, optionally, may be followed by a binary pixel age file. This switch is required if the output grid includes any portion of the southern hemisphere.

8.5.2.2 Optional Arguments for wwmca_regrid

5. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
6. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.
7. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

In any regridding problem, there are two grids involved: the "From" grid, which is the grid the input data are on, and the "To" grid, which is the grid the data are to be moved onto. In **WWMCA-Regrid** the "From" grid is pre-defined by the hemisphere of the WWMCA binary files being processed. The "To" grid and corresponding regridding logic are specified using the **regrid** section of the configuration file. If the "To" grid is entirely confined to one hemisphere, then only the WWMCA data file for that hemisphere needs to be given. If the "To" grid or the interpolation box used straddles the equator, the data files for both hemispheres need to be given. Once the "To" grid is specified in the config file, the WWMCA-Regrid tool will know which input data files it needs and will complain if it is not given the right ones.

8.5.3 wwmca_regrid Configuration File

The default configuration file for the WWMCA-Regrid tool named **WWMCARegridConfig_default** can be found in the installed *share/met/config* directory. We encourage users to make a copy of this file prior to modifying its contents. The contents of the configuration file are described in the subsections below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
regrid = { ... }
```

See the **regrid entry** in Section 4.5 for a description of the configuration file entries that control regridding.

```
variable_name = "Cloud_Pct";
units         = "percent";
long_name     = "cloud cover percent";
level         = "SFC";
```

The settings listed above are strings which control the output netCDF variable name and specify attributes for that variable.

```
init_time  = "";
valid_time = "";
accum_time = "01";
```

The settings listed above are strings which specify the timing information for the data being processed. The accumulation time is specified in HH[MMSS] format and, by default, is set to a value of 1 hour. The initialization and valid time strings are specified in YYYYMMDD[_HH[MMSS]] format. However, by default they are set to empty strings. If empty, the timing information parsed from the filename will be used. If not empty, these values override the times parsed from the filename.

```
max_minutes     = 120;
swap_endian     = TRUE;
write_pixel_age = FALSE;
```

The settings listed above control the processing of the WWMCA pixel age data. This data is stored in binary data files in 4-byte blocks. The **swap_endian** option indicates whether the endian-ness of the data should be swapped after reading. The **max_minutes** option specifies a maximum allowed age for the cloud data in minutes. Any data values older than this value are set to bad data in the output. The **write_pixel_age** option writes the pixel age data, in minutes, to the output file instead of the cloud data.

Chapter 9

Gen-Ens-Prod Tool

9.1 Introduction

The Gen-Ens-Prod tool generates simple ensemble products (mean, spread, probability, etc) from gridded ensemble member input files. While it processes model inputs, it does not compare them to observations or compute statistics. However, the output products can be passed as input to the MET statistics tools for comparison against observations. Climatological mean and standard deviation data may also be provided to define thresholds based on the climatological distribution at each grid point.

Note: This ensemble product generation step was provided by the Ensemble-Stat tool in earlier versions of MET. The Gen-Ens-Prod tool replaces and extends that functionality. Users are strongly encouraged to migrate ensemble product generation from Ensemble-Stat to Gen-Ens-Prod, as new features will only be added to Gen-Ens-Prod and the existing Ensemble-Stat functionality will be deprecated in a future version.

9.2 Scientific and Statistical Aspects

9.2.1 Ensemble Forecasts Derived from a Set of Deterministic Ensemble Members

Ensemble forecasts are often created as a set of deterministic forecasts. The ensemble members are rarely used separately. Instead, they can be combined in various ways to produce a forecast. MET can combine the ensemble members into some type of summary forecast according to user specifications. Ensemble means are the most common, and can be paired with the ensemble variance or spread. Maximum, minimum and other summary values are also available, with details in the practical information section.

The -ctrl command line option specifies an input file for the ensemble control member. The fields specified in the configuration file are read from the control member file. Those fields are included in the computation of the ensemble mean and probabilities but excluded from the ensemble spread.

The ensemble relative frequency is the simplest method for turning a set of deterministic forecasts into something resembling a probability forecast. MET will create the ensemble relative frequency as the proportion of

ensemble members forecasting some event. For example, if 5 out of 10 ensemble members predict measurable precipitation at a grid location, then the ensemble relative frequency of precipitation will be $5/10 = 0.5$. If the ensemble relative frequency is calibrated (unlikely) then this could be thought of as a probability of precipitation.

The neighborhood ensemble probability (NEP) and neighborhood maximum ensemble probability (NMEP) methods are described in *Schwartz and Sobash (2017)* (page 499). They are an extension of the ensemble relative frequencies described above. The NEP value is computed by averaging the relative frequency of the event within the neighborhood over all ensemble members. The NMEP value is computed as the fraction of ensemble members for which the event is occurring somewhere within the surrounding neighborhood. The NMEP output is typically smoothed using a Gaussian kernel filter. The neighborhood sizes and smoothing options can be customized in the configuration file.

The Gen-Ens-Prod tool writes the gridded relative frequencies, NEP, and NMEP fields to a NetCDF output file. Probabilistic verification methods can then be applied to those fields by evaluating them with the Grid-Stat and/or Point-Stat tools.

9.2.2 Climatology Data

The ensemble relative frequencies derived by Gen-Ens-Prod are computed by applying threshold(s) to the input ensemble member data. Those thresholds can be simple and remain constant over the entire domain (e.g. >0) or can be defined relative to the climatological distribution at each grid point (e.g. >OCDP90, for exceeding the 90-th percentile of the observation climatology data provided).

To use climatological distribution percentile thresholds, users must specify the climatological mean ("climo_mean") and standard deviation ("climo_stdev") entries in the configuration file. With forecast climatology inputs, use forecast climatology distribution percentile thresholds (e.g. >FCDP90). With observation climatology inputs, use observation climatological distribution percentile thresholds instead (e.g. >OCDP90). However, Gen-Ens-Prod cannot actually determine the input climatology data source and both "FCDP" and "OCDP" threshold types will work.

9.3 Practical Information

This section contains information about configuring and running the Gen-Ens-Prod tool. The Gen-Ens-Prod tool writes a NetCDF output file containing the requested ensemble product fields for each input field specified. If provided, the climatology data files must be gridded. All input gridded model and climatology datasets must be on the same grid. However, users may leverage the automated regridding feature in MET if the desired output grid is specified in the configuration file.

9.3.1 gen_ens_prod Usage

The usage statement for the Ensemble Stat tool is shown below:

```
Usage: gen_ens_prod
       -ens file_1 ... file_n | file_list
       -out file
       -config file
       [-ctrl file]
       [-log file]
       [-v level]
```

gen_ens_prod has three required arguments and accepts several optional ones.

9.3.2 Required Arguments gen_ens_prod

1. The **-ens file_1 ... file_n | file_list** option specifies the ensemble member files or ASCII file list of file names to be used, as described in Section 4.1.1.
2. The **-out file** option specifies the NetCDF output file name to be written.
3. The **-config file** option is a **GenEnsProdConfig** file containing the desired configuration settings.

9.3.3 Optional Arguments for gen_ens_prod

4. The **-ctrl file** option specifies the input file for the ensemble control member. Data for this member is included in the computation of the ensemble mean, but excluded from the spread. The control file should not appear in the **-ens** list of ensemble member files (unless processing a single file that contains all ensemble members).
5. The **-log** file outputs log messages to the specified file.
6. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

An example of the gen_ens_prod calling sequence is shown below:

```
gen_ens_prod \
-ens sample_fcst/2009123112/*gep*/d01_2009123112_02400.grib \
-out out/gen_ens_prod/gen_ens_prod_20100101_120000V_ens.nc \
-config config/GenEnsProdConfig -v 2
```

In this example, the Gen-Ens-Prod tool derives products from the input ensemble members listed on the command line.

9.3.4 gen_ens_prod Configuration File

The default configuration file for the Gen-Ens-Prod tool named **GenEnsProdConfig_default** can be found in the installed *share/met/config* directory. Another version is located in *scripts/config*. We encourage users to make a copy of these files prior to modifying their contents. The contents of the configuration file are described in the subsections below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
model          = "FCST";
desc           = "NA";
regrid         = { ... }
censor_thresh  = [];
censor_val     = [];
nc_var_str     = "";
climo_mean     = { ... } // Corresponding to ens.field entries
climo_stdev    = { ... } // Corresponding to ens.field entries
rng            = { ... }
version        = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
ens = {
  ens_thresh = 1.0;
  vld_thresh = 1.0;
  field = [
    {
      name       = "APCP";
      level      = "A03";
      cat_thresh = [ >0.0, >=5.0 ];
    }
  ];
}
```

The **ens** dictionary defines which ensemble fields should be processed.

When summarizing the ensemble, compute a ratio of the number of valid ensemble fields to the total number of ensemble members. If this ratio is less than the **ens_thresh**, then quit with an error. This threshold must be between 0 and 1. Setting this threshold to 1 requires that all ensemble members input files exist and all requested data be present.

When summarizing the ensemble, for each grid point compute a ratio of the number of valid data values to the number of ensemble members. If that ratio is less than **vld_thresh**, write out bad data for that grid point. This threshold must be between 0 and 1. Setting this threshold to 1 requires that each grid point contain valid data for all ensemble members in order to compute ensemble product values for that grid point.

For each dictionary entry in the **field** array, give the name and vertical or accumulation level, plus one or more categorical thresholds in the **cat_thresh** entry. The formatting for threshold are described in Section 5. It is the user's responsibility to know the units for each model variable and choose appropriate threshold values. The thresholds are used to define ensemble relative frequencies. For example, a threshold of >=5 is used to define the proportion of ensemble members predicting precipitation of at least 5mm at each grid point.

```
ens_member_ids = [];
control_id = "";
```

The **ens_member_ids** array is only used if reading a single file that contains all ensemble members. It should contain a list of string identifiers that are substituted into the **ens** dictionary fields to determine which data to read from the file. The length of the array determines how many ensemble members will be processed for a given field. Each value in the array will replace the text **MET_ENS_MEMBER_ID**.

NetCDF Example:

```
ens = {
  field = [
    {
      name  = "fcst";
      level = "(MET_ENS_MEMBER_ID,0,*,*)";
    }
  ];
}
```

GRIB Example:

```
ens = {
  field = [
    {
      name     = "fcst";
      level    = "L0";
      GRIB_ens = "MET_ENS_MEMBER_ID";
    }
  ];
}
```

This replacement behavior can also be applied to climatology file name entry, in the climo_mean and climo_stdev dictionaries.

```
climo_mean = {
   file_name = ["/path/to/file/memberMET_ENS_MEMBER_ID-mean.nc"];
   }
```

This substitution method can only be used if **ens_member_ids** has at least one entry and the **normalize** option is set to **CLIMO_ANOM** or **CLIMO_STD_ANOM**.

control_id is a string that is substituted in the same way as the **ens_member_ids** values to read a control member. This value is only used when the **-ctrl** command line argument is used. The value should not be found in the **ens_member_ids** array.

```
normalize = NONE;
```

The **normalize** option defines if and how the input ensemble member data should be normalized. Options are provided to normalize relative to an external climatology, specified using the **climo_mean** and **climo_stdev** dictionaries, or relative to current ensemble forecast being processed. The anomaly is computed by subtracting the (climatological or ensemble) mean from each ensemble memeber. The standard anomaly is computed by dividing the anomaly by the (climatological or ensemble) standard deviation. Values for the **normalize** option are described below:

- **NONE** (default) to skip the normalization step and process the raw ensemble member data.
- **CLIMO_ANOM** to subtract the climatological mean field.
- **CLIMO_STD_ANOM** to subtract the climatological mean field and divide by the climatological standard deviation.
- **FCST_ANOM** to subtract the current ensemble mean field.
- **FCST_STD_ANOM** to subtract the current ensemble mean field and divide by the current ensemble standard deviation.

Note that the **normalize** option may be specified separately for each entry in the **ens.field** array.

```
nbrhd_prob = {
   width      = [ 5 ];
   shape      = CIRCLE;
   vld_thresh = 0.0;
}
```

The **nbrhd_prob** dictionary defines the neighborhoods used to compute NEP and NMEP output.

The neighborhood **shape** is a **SQUARE** or **CIRCLE** centered on the current point, and the **width** array specifies the width of the square or diameter of the circle as an odd integer. The **vld_thresh** entry is a number between 0 and 1 specifying the required ratio of valid data in the neighborhood for an output value to be computed.

If **ensemble_flag.nep** is set to TRUE, NEP output is created for each combination of the categorical threshold (**cat_thresh**) and neighborhood width specified.

```
nmep_smooth = {
   vld_thresh      = 0.0;
   shape           = CIRCLE;
   gaussian_dx     = 81.27;
   gaussian_radius = 120;
```

(continues on next page)

(continued from previous page)

```
   type = [
      {
        method = GAUSSIAN;
        width  = 1;
      }
  ];
}
```

Similar to the **interp** dictionary, the **nmep_smooth** dictionary includes a **type** array of dictionaries to define one or more methods for smoothing the NMEP data. Setting the interpolation method to nearest neighbor (**NEAREST**) effectively disables this smoothing step.

If **ensemble_flag.nmep** is set to TRUE, NMEP output is created for each combination of the categorical threshold (**cat_thresh**), neighborhood width (**nbrhd_prob.width**), and smoothing method(**nmep_smooth.type**) specified.

```
ensemble_flag = {
  latlon    = TRUE;
       mean      = TRUE;
       stdev     = TRUE;
       minus     = TRUE;
       plus      = TRUE;
       min       = TRUE;
       max       = TRUE;
       range     = TRUE;
       vld_count = TRUE;
       frequency = TRUE;
       nep       = FALSE;
       nmep      = FALSE;
       climo     = FALSE;
       climo_cdp = FALSE;
}
```

The **ensemble_flag** specifies which derived ensemble fields should be calculated and output. Setting the flag to TRUE produces output of the specified field, while FALSE produces no output for that field type. The flags correspond to the following output line types:

1. Grid Latitude and Longitude Fields
2. Ensemble Mean Field
3. Ensemble Standard Deviation Field
4. Ensemble Mean - One Standard Deviation Field
5. Ensemble Mean + One Standard Deviation Field
6. Ensemble Minimum Field

7. Ensemble Maximum Field
8. Ensemble Range Field
9. Ensemble Valid Data Count
10. Ensemble Relative Frequency (i.e. uncalibrate probability forecast) for each categorical threshold (**cat_thresh**) specified
11. Neighborhood Ensemble Probability for each categorical threshold (**cat_thresh**) and neighborhood width (**nbrhd_prob.width**) specified
12. Neighborhood Maximum Ensemble Probability for each categorical threshold (**cat_thresh**), neighborhood width (**nbrhd_prob.width**), and smoothing method (**nmep_smooth.type**) specified
13. Climatology mean (**climo_mean**) and standard deviation (**climo_stdev**) data regridded to the model domain
14. Climatological Distribution Percentile field for each FCDP or OCDP threshold specified

9.3.5 gen_ens_prod Output

The Gen-Ens-Prod tools writes a gridded NetCDF output file whose file name is specified using the -out command line option. The contents of that file depend on the contents of the **ens.field** array, the **ensemble_flag** options selected, and the presence of climatology data. The NetCDF variable names are self-describing and include the name/level of the field being processed, the type of ensemble product, and any relevant threshold information. If **nc_var_str** is defined for an **ens.field** array entry, that string is included in the corresponding NetCDF output variable names.

The Gen-Ens-Prod NetCDF output can be passed as input to the MET statistics tools, like Point-Stat and Grid-Stat, for futher processing and comparison against observations.

Chapter 10

Spatial Masking

Verification over a particular region or area of interest may be performed using "masking". Defining a masking region is simply selecting the desired set of grid points to be used. The Gen-Vx-Mask tool automates this process and replaces the Gen-Poly-Mask and Gen-Circle-Mask tools from previous releases. It may be run to create a bitmap verification masking region to be used by many of the statistical tools. This tool enables the user to generate a masking region once for a domain and apply it to many cases. It supports multiple methods for defining regional spatial masks, as described below. In addition, Gen-Vx-Mask can be run iteratively, passing the output from one run as input to the next, to combine multiple masking regions and define a complex area of interest.

10.1 Gen-Vx-Mask Tool

The Gen-Vx-Mask tool may be run to create a bitmap verification masking region to be used by the MET statistics tools. This tool enables the user to generate a masking region once for a domain and apply it to many cases. While the MET statistics tools can define some masking regions on the fly using pre-defined grids and polylines, doing so can be slow, especially for complex polylines containing hundreds of vertices. Using the Gen-Vx-Mask tool to create a bitmap masking region before running the other MET tools will make them run more efficiently.

10.1.1 gen_vx_mask Usage

The usage statement for the Gen-Vx-Mask tool is shown below:

```
Usage: gen_vx_mask
       input_grid
       mask_file
       out_file
       -type str
       [-input_field string]
       [-mask_field string]
       [-complement]
```

(continues on next page)

(continued from previous page)

```
      [-union | -intersection | -symdiff]
      [-thresh string]
      [-height n]
      [-width n]
      [-shapeno n]
      [-shape_str name string]
      [-value n]
      [-name string]
      [-log file]
      [-v level]
      [-compress level]
```

gen_vx_mask has four required arguments and can take optional ones. Note that **-type string** (masking type) was optional in prior versions but is now required.

10.1.1.1 Required Arguments for gen_vx_mask

1. The **input_grid** is a named grid, the path to a gridded data file, or an explicit grid specification string (see Section 34.2) which defines the grid for which a mask is to be defined. If set to a gen_vx_mask output file, automatically read mask data as the **input_field**.
2. The **mask_file** defines the masking information, see below.

- For "poly", "poly_xy", "box", "circle", and "track" masking, specify an ASCII Lat/Lon file. Refer to *Types of Masking Available in gen_vx_mask* (page 206) for details on how to construct the ASCII Lat/Lon file for each type of mask.
- For "grid" masking, specify a named grid, the path to a gridded data file, or an explicit grid specification.
- For "data" masking, specify a gridded data file.
- For "solar_alt", "solar_azi", and "solar_time" masking, specify a gridded data file or a time string in YYYYMMDD[_HH[MMSS]] UTC format.
- For "lat" and "lon" masking, no "mask_file" is needed, simply repeat "input_grid".
- For "shape" masking, specify a shapefile (suffix ".shp").

3. The **out_file** is the output NetCDF mask file to be written.
4. The **-type string** is a comma-separated list of masking types to be applied. The application will print an error message and exit if "-type string" is not specified at least once on the command line. Use multiple times for multiple mask types. See a list of supported masking types described below.

Note: While multiple **-type** mask types can be requested in a single run, all requested masking types must use the same **mask_file** setting.

10.1.1.2 Optional Arguments for gen_vx_mask

5. The **-input_field string** option initializes the "input_grid" with values from this field.
6. The **-mask_field string** option defines the field from "mask_file" to be used for "data" masking. Use multiple times for multiple mask types.
7. The **-complement** option can be used to compute the complement of the area defined by "mask_file".
8. The **-union | -intersection | -symdiff** options specify how to combine multiple binary masks. Applies to masks read from the "input_field" and those generated during the current run.
9. The **-thresh string** option is a comma-separated list of thresholds to be applied. Use multiple times for multiple mask types.
   - For "circle" and "track" masking, threshold the distance (km).
   - For "data" masking, threshold the values of "mask_field".
   - For "solar_alt" and "solar_azi" masking, threshold the computed solar values (deg).
   - For "solar_time" masking, threshold the solar time (hr).
   - For "lat" and "lon" masking, threshold the latitude and longitude values (deg).
10. The **-height n** and **-width n** options specify the dimensions in grid units for "box" masking.
11. The **-shapeno n** option is only used for shapefile masking. See the description of shapefile masking below.
12. The **-shape_str name string** option is only used for shapefile masking. See the description of shapefile masking below.
13. The **-value n** option overrides the default output mask data value (1).
14. The **-name string** option specifies the output variable name for the mask.
15. The **-log file** option writes log messages to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
16. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.
17. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

10.1.1.3 Types of Masking Available in gen_vx_mask

The Gen-Vx-Mask tool supports the following types of masking region definition selected using the **-type** command line option:

1. Polyline (**poly**) masking reads an input ASCII file containing Lat/Lon locations, connects the first and last points, and selects grid points whose Lat/Lon location falls inside that polyline in Lat/Lon space. This option is useful when defining geographic subregions of a domain.

2. Polyline XY (**poly_xy**) masking reads an input ASCII file containing Lat/Lon locations. It converts the polyline Lat/Lon locations into grid X/Y space and connects the first and last points. It selects grid points whose X/Y location falls inside that polyline in X/Y space. This option is useful when defining geographic subregions of a domain.

3. Box (**box**) masking reads an input ASCII file containing Lat/Lon locations and draws a box around each point. The height and width of the box is specified by the **-height** and **-width** command line options in grid units. For a square, only one of **-height** or **-width** needs to be used.

4. Circle (**circle**) masking reads an input ASCII file containing Lat/Lon locations and for each grid point, computes the minimum great-circle arc distance in kilometers to those points. If the **-thresh** command line option is not used, the minimum distance value for each grid point will be written to the output. If it is used, only those grid points whose minimum distance meets the threshold criteria will be selected. This option is useful when defining areas within a certain radius of radar locations.

5. Track (**track**) masking reads an input ASCII file containing Lat/Lon locations and for each grid point, computes the minimum great-circle arc distance in kilometers to the track defined by those points. The first and last track points are not connected. As with **circle** masking the output for each grid point depends on the use of the **-thresh** command line option. This option is useful when defining the area within a certain distance of a hurricane track.

6. Grid (**grid**) masking reads an input gridded data file, extracts the field specified using its grid definition, and selects grid points falling inside that grid. This option is useful when using a model nest to define the corresponding area of the parent domain.

7. Data (**data**) masking reads an input gridded data file, extracts the field specified using the **-mask_field** command line option, thresholds the data using the **-thresh** command line option, and selects grid points which meet that threshold criteria. The option is useful when thresholding topography to define a mask based on elevation or when threshold land use to extract a particular category.

8. Solar altitude (**solar_alt**) and solar azimuth (**solar_azi**) masking computes the solar altitude and azimuth values in degrees at each grid point for the time defined by the **mask_file** setting. **mask_file** may either be set to an explicit time string in YYYYMMDD[_HH[MMSS]] UTC format or to a gridded data file. If set to a gridded data file, the **-mask_field** command line option specifies the field of data whose valid time should be used. If the **-thresh** command line option is not used, the raw solar altitude or azimuth degrees for each grid point will be written to the output. If it is used, the resulting binary mask field will be written. This option is useful when defining a day/night mask.

9. Solar time (**solar_time**) masking computes the solar time in decimal hours at each grid point for the for the time defined by the **mask_file** setting, as described above. The solar hours of the day range from 0 to 24, with a value of 12 indicating solar noon. Note that solar time is based only on longitude. If the **-thresh** command line option is not used, the raw solar time hours will be written to the output.

10. Latitude (**lat**) and longitude (**lon**) masking computes the latitude and longitude value at each grid point. This logic only requires the definition of the grid, specified by the **input_file**. Technically, the **mask_file** is not needed, but a value must be specified for the command line to parse correctly. Users are advised to simply repeat the **input_file** setting twice. If the **-thresh** command line option is not used, the raw latitude or longitude values for each grid point will be written to the output. This option is useful when defining latitude or longitude bands over which to compute statistics.

11. Shapefile (**shape**) masking uses closed polygons taken from an ESRI shapefile to define the masking region. Gen-Vx-Mask reads the shapefile with the ".shp" suffix and extracts the latitude and longitudes of the vertices. The shapefile must consist of closed polygons rather than polylines, points, or any of the other data types that shapefiles support. When the **-shape_str** command line option is used, Gen-Vx-Mask also reads metadata from the corresponding dBASE file with the ".dbf" suffix.

    Shapefiles usually contain more than one polygon, and the user must select which of these shapes should be used. The **-shapeno n** and **-shape_str name string** command line options enable the user to select one or more polygons from the shapefile. For **-shape n**, **n** is a comma-separated list of integer shape indices to be used. Note that these values are zero-based. So the first polygon in the shapefile is shape number 0, the second polygon in the shapefile is shape number 1, etc. For example, **-shapeno 0,1,2** uses the first three shapes in the shapefile. When multiple shapes are specified, the mask is defined as their union. So all grid points falling inside at least one of the specified shapes are included in the mask.

    For the user's convenience, some utilities that perform human-readable screen dumps of shapefile contents are provided with MET. The **gis_dump_shp**, **gis_dump_shx**, and **gis_dump_dbf** tools enable the user to examine the contents of these shapefiles. In particular, the **gis_dump_dbf** tool prints the name and values of the metadata for each record. The **-shape_str** command line option filters the shapes using the attributes listed in the **gis_dump_dbf** output, and requires two arguments. The **name** argument is set to any valid shapefile attribute, and the **string** argument is a comma-separated list of values to be matched. An example of using **-shape_str** is **-shape_str CONTINENT Europe**, which will match all "CONTINENT" attribues that have the string "Europe" in them. Strings that contain embedded whitespace should be enclosed in single quotes. Also note that case insensitive matching is used. For example, when using a global country outline shapefile, **-shape_str NAME 'united kingdom,united states of america'** matches the "NAME" attributes that have both "United Kingdom" and "United States of America" in them. If **-shape_str** is used multiple times, only shapes matching all the named attributes will be used. For example, **-shape_str CONTINENT Europe -shape_str NAME Spain,Portugal** will only match shapes where the "CONTINENT" attrinute contains "Europe "and the "NAME" attribute contains "Spain" or "Portugal". If a user wishes, they can combine both the **-shape_str** and **-shapeno** options. In this case, the union of all matches from the shapefile will be used.

The polyline, polyline XY, box, circle, and track masking methods all read an ASCII file containing Lat/Lon locations. Those files must contain a string, which defines the name of the masking region, followed by a series of whitespace-separated latitude (degrees north) and longitude (degree east) values.

10.1.1.4 Logic for gen_vx_mask

The Gen-Vx-Mask tool performs three main steps, described below.

1. Determine the input grid definition.

- Read the **input_grid** to determine the grid over which the mask should be defined.
- By default, initialize the input field value at each grid point to zero.
- If the **-input_field** option was specified, initialize each input field value using the values from that field.
- If the **input_grid** is the output from a previous run of Gen-Vx-Mask, automatically initialize each input field value with the previously-generated mask value.

2. Process each of the requested masking regions.

- For each **-type** mask type option requested, process the **mask_file** setting.
- Read the **mask_file**, process it based on the **-type** setting (as described above), and define the masking region value for each grid point to specify whether or not it is included in the mask.
- By default, store the mask value as 1 unless the **-value** option was specified to override that default value.
- If the **-complement** option was specified, select the opposite of the masking area.
- Apply logic to combine the newly generated masking region with those defined by previous **-type** mask type options to create a **mask_field**.
  - By default, compute the **-union** of multiple masks, unless **-intersection** or **-symdiff** were specified to override this default.

3. Apply logic to combine the input field and current masking region and write the **out_file**.

- By default, the output value at each grid point is set to the value of current masking region if included in the mask, or the value of **input_field** if not included.
- If the **-union, -intersection**, or **-symdiff** option was specified, apply that logic to the input field and current masking region values at each grid point to determine the output value.
- Write the output value for each grid point to the **out_file**.

10.1.1.5 Examples for gen_vx_mask

An example of defining the northwest hemisphere of the earth, as defined by latitudes $>= 0$ and longitudes < 0, in a single run is shown below:

```
gen_vx_mask G004 G004 northwest_hemisphere.nc \
-type lat,lon -thresh ge0,lt0 \
-intersection -name nw_hemisphere
```

The Gen-Vx-Mask tool to be run iteratively on its own output using different **mask_file** settings to generate complex masking areas. The **-union, -intersection**, and **-symdiff** options control the logic for combining the

input field and current mask values at each grid point. For example, one could define a complex masking region by selecting grid points with an elevation greater than 1000 meters within a Contiguous United States geographic region by doing the following:

- Run Gen-Vx-Mask to apply data masking by thresholding a field of topography greater than 1000 meters.
- Run Gen-Vx-Mask a second time on the output from the first call and applying polyline masking to define the geographic area of interest. Use the **-intersection** option to only select grid points whose value is non-zero in both the input field and the current mask.

An example of this Gen-Vx-Mask calling sequence is shown below:

```
gen_vx_mask fcst.grib fcst.grib TOPO_mask.nc \
-type data \
-mask_field 'name="TOPO"; level="L0";' \
-thresh '>1000'

gen_vx_mask TOPO_mask.nc CONUS.poly TOPO_CONUS_mask.nc \
-type poly \
-intersection -name TOPO_CONUS_mask
```

Here, Gen-Vx-Mask uses the **data** masking type to read topography data (**TOPO**) from a GRIB file and thresholds the values **>1000** to define a topography mask. The second run of Gen-Vx-Mask uses the **poly** masking type to read the ASCII Lat/Lon file named **CONUS.poly** and select all grid points within that region to define a polyline mask. When reading its own output, Gen-Vx-Mask automatically reads the topography mask as the **input_field** and applies the **intersection** logic to combine it with the polyline mask, selecting grid points where both conditions are true. The resulting complex mask is written to the output NetCDF file named **TOPO_CONUS_mask.nc**.

10.2 Feature-Relative Methods

This section contains a description of several methods that may be used to perform feature-relative (or event -based) evaluation. The methodology pertains to examining the environment surrounding a particular feature or event such as a tropical, extra-tropical cyclone, convective cell, snow-band, etc. Several approaches are available for these types of investigations including applying masking described above (e.g. circle or box) or using the FORCE interpolation method in the regrid configuration option (see Section 5). These methods generally require additional scripting, including potentially storm-track identification, outside of MET to be paired with the features of the MET tools. METplus may be used to execute this type of analysis. Please refer to the METplus User's Guide.

Chapter 11

Point-Stat Tool

11.1 Introduction

The Point-Stat tool provides verification statistics for forecasts at observation points (as opposed to over gridded analyses). The Point-Stat tool matches gridded forecasts to point observation locations and supports several different interpolation options. The tool then computes continuous, categorical, spatial, and probabilistic verification statistics. The categorical and probabilistic statistics generally are derived by applying a threshold to the forecast and observation values. Confidence intervals - representing the uncertainty in the verification measures - are computed for the verification statistics.

Scientific and statistical aspects of the Point-Stat tool are discussed in the following section. Practical aspects of the Point-Stat tool are described in Section 11.3.

11.2 Scientific and Statistical Aspects

The statistical methods and measures computed by the Point-Stat tool are described briefly in this section. In addition, Section 11.2.1 discusses the various interpolation options available for matching the forecast grid point values to the observation points. The statistical measures computed by the Point-Stat tool are described briefly in Section 11.2.4 and in more detail in Appendix C, Section 35. Section 11.2.5 describes the methods for computing confidence intervals that are applied to some of the measures computed by the Point-Stat tool; more detail on confidence intervals is provided in Appendix D, Section 36.

11.2.1 Interpolation/Matching Methods

This section provides information about the various methods available in MET to match gridded model output to point observations. Matching in the vertical and horizontal are completed separately using different methods.

In the vertical, if forecasts and observations are at the same vertical level, then they are paired as-is. If any discrepancy exists between the vertical levels, then the forecasts are interpolated to the level of the observation. The vertical interpolation is done in the natural log of pressure coordinates, except for specific humidity, which is interpolated using the natural log of specific humidity in the natural log of pressure

coordinates. Vertical interpolation for heights above ground are done linear in height coordinates. When forecasts are for the surface, no interpolation is done. They are matched to observations with message types that are mapped to "SURFACE" in the **message_type_group_map** configuration option. By default, the surface message types include ADPSFC, SFCSHP, and MSONET. The regular expression is applied to the message type list at the message_type_group_map. The derived message types from the time summary ("ADPSFC_MIN_hhmmss" and "ADPSFC_MAX_hhmmss") are accepted as "ADPSFC".

To match forecasts and observations in the horizontal plane, the user can select from a number of methods described below. Many of these methods require the user to define the width of the forecast grid W, around each observation point P, that should be considered. In addition, the user can select the interpolation shape, either a SQUARE or a CIRCLE. For example, a square of width 2 defines the 2 x 2 set of grid points enclosing P, or simply the 4 grid points closest to P. A square of width of 3 defines a 3 x 3 square consisting of 9 grid points centered on the grid point closest to P. Figure 11.1 provides illustration. The point P denotes the observation location where the interpolated value is calculated. The interpolation width W, shown is five.

This section describes the options for interpolation in the horizontal.

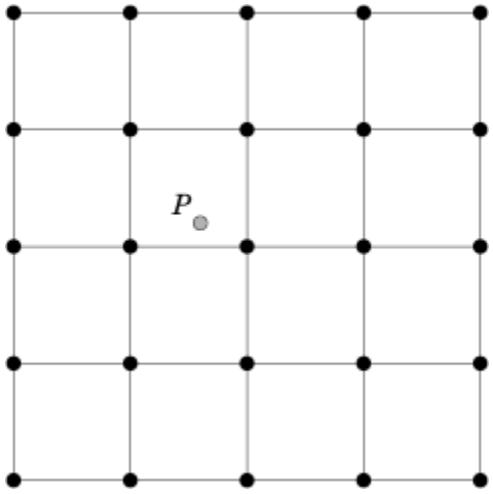

Figure 11.1: Diagram illustrating matching and interpolation methods used in MET. See text for explanation.

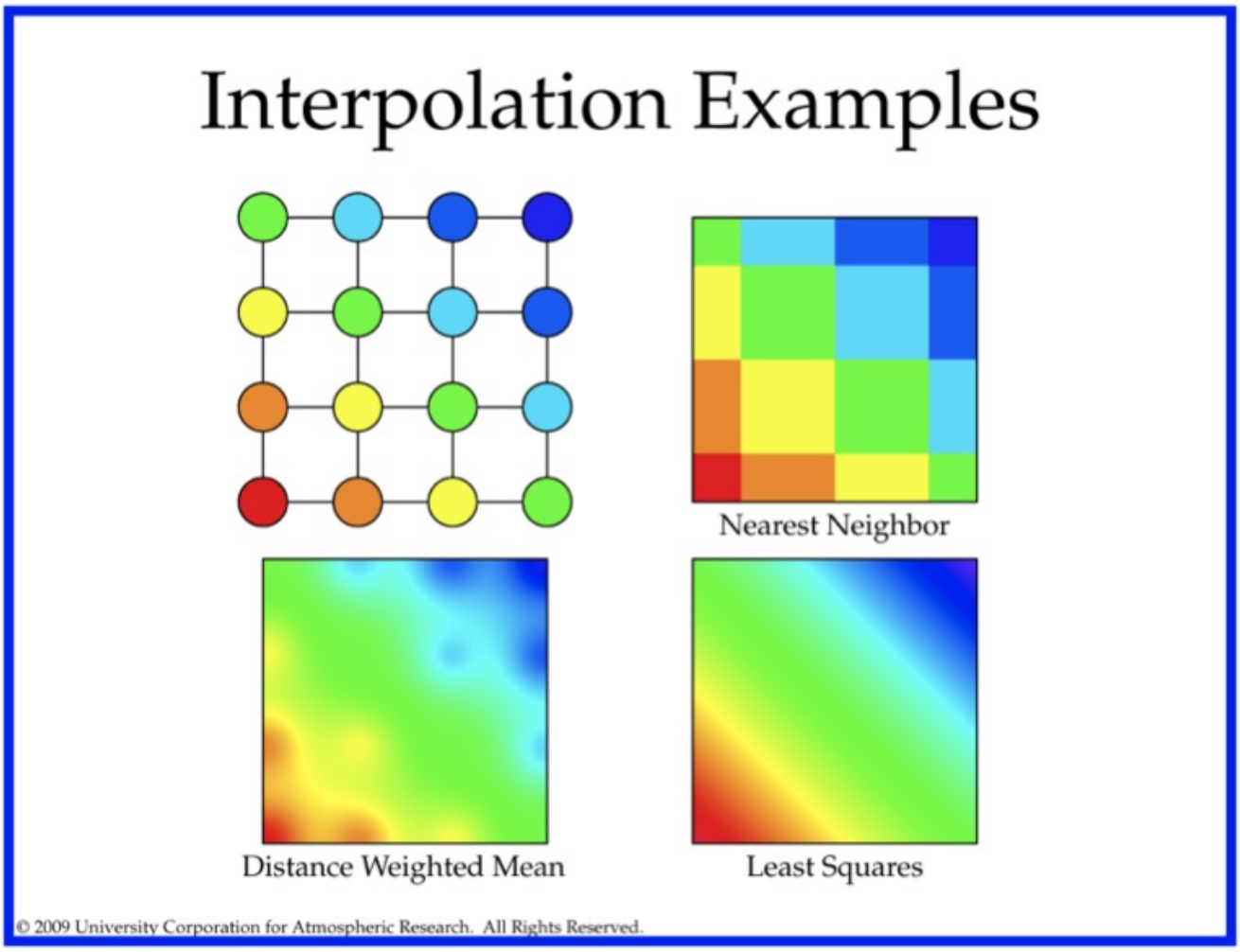

Figure 11.2: Illustration of some matching and interpolation methods used in MET. See text for explanation.

Nearest Neighbor

The forecast value at P is assigned the value at the nearest grid point. No interpolation is performed. Here, "nearest" means spatially closest in horizontal grid coordinates. This method is used by default when the interpolation width, W, is set to 1.

Geography Match

The forecast value at P is assigned the value at the nearest grid point in the interpolation area where the land/sea mask and topography criteria are satisfied.

Gaussian

The forecast value at P is a weighted sum of the values in the interpolation area. The weight given to each forecast point follows the Gaussian distribution with nearby points contributing more the far away points. The shape of the distribution is configured using sigma.

When used for regridding, with the **regrid** configuration option, or smoothing, with the **interp** configuration option in grid-to-grid comparisons, the Gaussian method is named **MAXGAUSS** and is implemented as a 2-step process. First, the data is regridded or smoothed using the maximum value interpolation method described below, where the **width** and **shape** define the interpolation area. Second, the Gaussian smoother, defined by the **gaussian_dx** and **gaussian_radius** configuration options, is applied.

Minimum value

The forecast value at P is the minimum of the values in the interpolation area.

Maximum value

The forecast value at P is the maximum of the values in the interpolation area.

Distance-weighted mean

The forecast value at P is a weighted sum of the values in the interpolation area. The weight given to each forecast point is the reciprocal of the square of the distance (in grid coordinates) from P. The weighted sum of forecast values is normalized by dividing by the sum of the weights.

Unweighted mean

This method is similar to the distance-weighted mean, except all the weights are equal to 1. The distance of any point from P is not considered.

Median

The forecast value at P is the median of the forecast values in the interpolation area.

Least-Squares Fit

To perform least squares interpolation of a gridded field at a location P, MET uses an **WxW** subgrid centered (as closely as possible) at P. Figure 11.1 shows the case where W = 5.

If we denote the horizontal coordinate in this subgrid by x, and vertical coordinate by y, then we can assign coordinates to the point P relative to this subgrid. These coordinates are chosen so that the center of the grid is. For example, in Figure 11.1, P has coordinates (-0.4, 0.2). Since the grid is centered near P, the coordinates of P should always be at most 0.5 in absolute value. At each of the vertices of the grid (indicated by black dots in the figure), we have data values. We would like to use these values to interpolate a value at P. We do this using least squares. If we denote the interpolated value by z, then we fit an expression of the form $z = \alpha(x) + \beta(y) + \gamma$ over the subgrid. The values of α, β, γ are calculated from the data values at the

vertices. Finally, the coordinates (**x,y**) of P are substituted into this expression to give z, our least squares interpolated data value at P.

Bilinear Interpolation

This method is performed using the four closest grid squares. The forecast values are interpolated linearly first in one dimension and then the other to the location of the observation.

Upper Left, Upper Right, Lower Left, Lower Right Interpolation

This method is performed using the four closest grid squares. The forecast values are interpolated to the specified grid point.

Best Interpolation

The forecast value at P is chosen as the grid point inside the interpolation area whose value most closely matches the observation value.

11.2.2 HiRA Framework

The Point-Stat tool has been enhanced to include the High Resolution Assessment (HiRA) verification logic (*Mittermaier, 2014* (page 497)). HiRA is analogous to neighborhood verification but for point observations. The HiRA logic interprets the forecast values surrounding each point observation as an ensemble forecast. These ensemble values are processed in three ways. First, the ensemble continuous statistics (ECNT), the observation rank statistics (ORANK) and the ranked probability score (RPS) line types are computed directly from the ensemble values. Second, for each categorical threshold specified, a fractional coverage value is computed as the ratio of the nearby forecast values that meet the threshold criteria. Point-Stat evaluates those fractional coverage values as if they were a probability forecast. When applying HiRA, users should enable the matched pair (MPR), probabilistic (PCT, PSTD, PJC, or PRC), continuous ensemble statistics (ECNT), observation rank statistics (ORANK) or ranked probability score (RPS) line types in the **output_flag** dictionary. The number of probabilistic HiRA output lines is determined by the number of categorical forecast thresholds and HiRA neighborhood widths chosen.

The HiRA framework provides a unique method for evaluating models in the neighborhood of point observations, allowing for some spatial and temporal uncertainty in the forecast and/or the observations. Additionally, the HiRA framework can be used to compare deterministic forecasts to ensemble forecasts. In MET, the neighborhood is a circle or square centered on the grid point closest to the observation location. An event is defined, then the proportion of points with events in the neighborhood is calculated. This proportion is treated as an ensemble probability, though it is likely to be uncalibrated.

Figure 11.3 shows a couple of examples of how the HiRA proportion is derived at a single model level using square neighborhoods. Events (in our case, model accretion values > 0) are separated from non-events (model accretion value $= 0$). Then, in each neighborhood, the total proportion of events is calculated. In the leftmost panel, four events exist in the 25 point neighborhood, making the HiRA proportion is 4/25 $= 0.16$. For the neighborhood of size 9 centered in that same panel, the HiRA proportion is 1/9. In the right panel, the size 25 neighborhood has HiRA proportion of 6/25, with the centered 9-point neighborhood

having a HiRA value of 2/9. To extend this method into 3-dimensions, all layers within the user-defined layer are also included in the calculation of the proportion in the same manner.

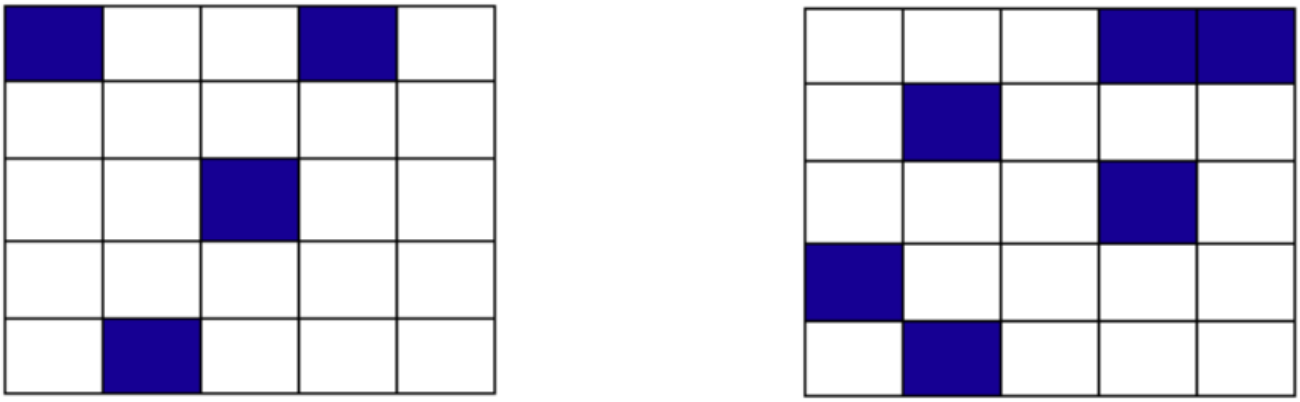

Figure 11.3: Example showing how HiRA proportions are calculated.

Often, the neighborhood size is chosen so that multiple models to be compared have approximately the same horizontal resolution. Then, standard metrics for probabilistic forecasts, such as Brier Score, can be used to compare those forecasts. HiRA was developed using surface observation stations so the neighborhood lies completely within the horizontal plane. With any type of upper air observation, the vertical neighborhood must also be defined.

11.2.3 SEEPS Scores

The Stable Equitable Error in Probability Space (SEEPS) was devised for monitoring global deterministic forecasts of precipitation against the WMO gauge network (*Rodwell et al., 2010* (page 498); *Haiden et al., 2012* (page 496)) and is a multi-category score which uses a climatology to account for local variations in behavior. Since the score uses probability space to define categories using the climatology, it can be aggregated over heterogeneous climate regions. Even though it was developed for use with precipitation forecasts, in principle it could be applied to any forecast parameter for which a sufficiently long time period of observations exists to create a suitable climatology. The computation of SEEPS for precipitation is only supported for now.

For use with precipitation, three categories are used, named 'dry', 'light' and 'heavy'. The 'dry' category is defined (using the WMO observing guidelines) with any accumulation (rounded to the nearest 0.1 millimeter) that is less than or equal to 0.2 mm. The remaining precipitation is divided into 'light' and 'heavy' categories whose thresholds are with respect to a climatology and thus location specific. The light precipitation is defined to occur twice as often as heavy precipitation.

When calculating a single SEEPS value over observing stations for a particular region, the scores should have a density weighting applied which accounts for uneven station distribution in the region of interest (see Section 9.1 in *Rodwell et al., 2010* (page 498)). This density weighting has not yet been implemented in MET. Global precipitation climatologies calculated from the WMO SYNOP records from 1980-2009 are supplied with the release. At the moment, a 24-hour climatology is available (valid at 00 UTC or 12 UTC), but in future a 6-hour climatology will become available.

11.2.4 Statistical Measures

The Point-Stat tool computes a wide variety of verification statistics. Broadly speaking, these statistics can be subdivided into statistics for categorical variables and statistics for continuous variables. The categories of measures are briefly described here; specific descriptions of the measures are provided in Appendix C, Section 35. Additional information can be found in *Wilks (2011)* (page 500) and *Jolliffe and Stephenson (2012)* (page 496), and at Collaboration for Australian Weather and Climate Research. Forecast Verification - Issues, Methods and FAQ web page.

In addition to these verification measures, the Point-Stat tool also computes partial sums and other FHO statistics that are produced by the NCEP verification system. These statistics are also described in Appendix C, Section 35.

11.2.4.1 Measures for Categorical Variables

Categorical verification statistics are used to evaluate forecasts that are in the form of a discrete set of categories rather than on a continuous scale. If the original forecast is continuous, the user may specify one or more thresholds in the configuration file to divide the continuous measure into categories. Currently, Point-Stat computes categorical statistics for variables in two or more categories. The special case of dichotomous (i.e., 2-category) variables has several types of statistics calculated from the resulting contingency table and are available in the CTS output line type. For multi-category variables, fewer statistics can be calculated so these are available separately, in line type MCTS. Categorical variables can be intrinsic (e.g., rain/no-rain) or they may be formed by applying one or more thresholds to a continuous variable (e.g., temperature < 273.15 K or cloud coverage percentages in 10% bins). See Appendix C, Section 35 for more information.

11.2.4.2 Measures for Continuous Variables

For continuous variables, many verification measures are based on the forecast error (i.e., f - o). However, it also is of interest to investigate characteristics of the forecasts, and the observations, as well as their relationship. These concepts are consistent with the general framework for verification outlined by *Murphy and Winkler (1987)* (page 497). The statistics produced by MET for continuous forecasts represent this philosophy of verification, which focuses on a variety of aspects of performance rather than a single measure. See Appendix C, Section 35 for specific information.

A user may wish to eliminate certain values of the forecasts from the calculation of statistics, a process referred to here as``'conditional verification". For example, a user may eliminate all temperatures above freezing and then calculate the error statistics only for those forecasts of below freezing temperatures. Another common example involves verification of wind forecasts. Since wind direction is indeterminate at very low wind speeds, the user may wish to set a minimum wind speed threshold prior to calculating error statistics for wind direction. The user may specify these thresholds in the configuration file to specify the conditional verification. Thresholds can be specified using the usual Fortran conventions (<, <=, ==, !-, >=, or >) followed by a numeric value. The threshold type may also be specified using two letter abbreviations (lt, le, eq, ne, ge, gt). Further, more complex thresholds can be achieved by defining multiple thresholds and using && or || to string together event definition logic. The forecast and observation threshold can be used together according to user preference by specifying one of: UNION, INTERSECTION, or SYMDIFF (symmetric difference).

11.2.4.3 Measures for Probabilistic Forecasts and Dichotomous Outcomes

For probabilistic forecasts, many verification measures are based on reliability, accuracy and bias. However, it also is of interest to investigate joint and conditional distributions of the forecasts and the observations, as in *Wilks (2011)* (page 500). See Appendix C, Section 35 for specific information.

Probabilistic forecast values are assumed to have a range of either 0 to 1 or 0 to 100. If the max data value is > 1, we assume the data range is 0 to 100, and divide all the values by 100. If the max data value is <= 1, then we use the values as is. Further, thresholds are applied to the probabilities with equality on the lower end. For example, with a forecast probability p, and thresholds t1 and t2, the range is defined as: t1 <= p < t2. The exception is for the highest set of thresholds, when the range includes 1: t1 <= p <= 1. To make configuration easier, in METv6.0, these probabilities may be specified in the configuration file as a list (>=0.00,>=0.25,>=0.50,>=0.75,>=1.00) or using shorthand notation (==0.25) for bins of equal width.

When the "prob" entry is set as a dictionary to define the field of interest, setting "prob_as_scalar = TRUE" indicates that this data should be processed as regular scalars rather than probabilities. For example, this option can be used to compute traditional 2x2 contingency tables and neighborhood verification statistics for probability data. It can also be used to compare two probability fields directly.

11.2.4.4 Measures for Comparison Against Climatology

For each of the types of statistics mentioned above (categorical, continuous, and probabilistic), it is possible to calculate measures of skill relative to climatology. MET will accept a climatology file provided by the user, and will evaluate it as a reference forecast. Further, anomalies, i.e. departures from average conditions, can be calculated. As with all other statistics, the available measures will depend on the nature of the forecast. Common statistics that use a climatological reference include: the mean squared error skill score (MSESS), the Anomaly Correlation (ANOM_CORR and ANOM_CORR_UNCNTR), scalar and vector anomalies (SAL1L2 and VAL1L2), continuous ranked probability skill score (CRPSS and CRPSS_EMP), Brier Skill Score (BSS) (*Wilks, 2011* (page 500); *Mason, 2004* (page 496)).

Often, the sample climatology is used as a reference by a skill score. The sample climatology is the average over all included observations and may be transparent to the user. This is the case in most categorical skill scores. The sample climatology will probably prove more difficult to improve upon than a long term climatology, since it will be from the same locations and time periods as the forecasts. This may mask legitimate forecast skill. However, a more general climatology, perhaps covering many years, is often easier to improve upon and is less likely to mask real forecast skill.

11.2.5 Statistical Confidence Intervals

A single summary score gives an indication of the forecast performance, but it is a single realization from a random process that neglects uncertainty in the score's estimate. That is, it is possible to obtain a good score, but it may be that the "good" score was achieved by chance and does not reflect the "true" score. Therefore, when interpreting results from a verification analysis, it is imperative to analyze the uncertainty in the realized scores. One good way to do this is to utilize confidence intervals. A confidence interval indicates that if the process were repeated many times, say 100, then the true score would fall within the interval $100(1-\alpha)\%$ of the time. Typical values of α are 0.01, 0.05, and 0.10. The Point-Stat tool allows the user to select one or more specific α-values to use.

For continuous fields (e.g., temperature), it is possible to estimate confidence intervals for some measures of forecast performance based on the assumption that the data, or their errors, are normally distributed. The Point-Stat tool computes confidence intervals for the following summary measures: forecast mean and standard deviation, observation mean and standard deviation, correlation, mean error, and the standard deviation of the error. In the case of the respective means, the central limit theorem suggests that the means are normally distributed, and this assumption leads to the usual $100(1-\alpha)\%$ confidence intervals for the mean. For the standard deviations of each field, one must be careful to check that the field of interest is normally distributed, as this assumption is necessary for the interpretation of the resulting confidence intervals.

For the measures relating the two fields (i.e., mean error, correlation and standard deviation of the errors), confidence intervals are based on either the joint distributions of the two fields (e.g., with correlation) or on a function of the two fields. For the correlation, the underlying assumption is that the two fields follow a bivariate normal distribution. In the case of the mean error and the standard deviation of the mean error, the assumption is that the errors are normally distributed, which for continuous variables, is usually a reasonable assumption, even for the standard deviation of the errors.

Bootstrap confidence intervals for any verification statistic are available in MET. Bootstrapping is a non-parametric statistical method for estimating parameters and uncertainty information. The idea is to obtain a sample of the verification statistic(s) of interest (e.g., bias, ETS, etc.) so that inferences can be made from this sample. The assumption is that the original sample of matched forecast-observation pairs is representative of the population. Several replicated samples are taken with replacement from this set of forecast-observation pairs of variables (e.g., precipitation, temperature, etc.), and the statistic(s) are calculated for each replicate. That is, given a set of n forecast-observation pairs, we draw values at random from these pairs, allowing the same pair to be drawn more than once, and the statistic(s) is (are) calculated for each replicated sample. This yields a sample of the statistic(s) based solely on the data without making any assumptions about the underlying distribution of the sample. It should be noted, however, that if the observed sample of matched pairs is dependent, then this dependence should be taken into account somehow. Currently, the confidence interval methods in MET do not take into account dependence, but future releases will support a robust method allowing for dependence in the original sample. More detailed information about the bootstrap algorithm is found in the Appendix D, Section 36. Note that MET writes temporary files whenever bootstrap confidence intervals are computed, as described in `Contributor's Guide Section %s`.

Confidence intervals can be calculated from the sample of verification statistics obtained through the bootstrap algorithm. The most intuitive method is to simply take the appropriate quantiles of the sample of statistic(s). For example, if one wants a 95% CI, then one would take the 2.5 and 97.5 percentiles of the resulting sample. This method is called the percentile method, and has some nice properties. However, if the original sample is biased and/or has non-constant variance, then it is well known that this interval is too optimistic. The most robust, accurate, and well-behaved way to obtain accurate CIs from bootstrapping is to use the bias corrected and adjusted percentile method (or BCa). If there is no bias, and the variance is constant, then this method will yield the usual percentile interval. The only drawback to the approach is that it is computationally intensive. Therefore, both the percentile and BCa methods are available in MET, with the considerably more efficient percentile method being the default.

The only other option associated with bootstrapping currently available in MET is to obtain replicated samples smaller than the original sample (i.e., to sample $m<n$ points at each replicate). Ordinarily, one should use $m=n$, and this is the default. However, there are cases where it is more appropriate to use a smaller value of m (e.g., when making inference about high percentiles of the original sample). See *Gilleland (2010)* (page 495) for more information and references about this topic.

MET provides parametric confidence intervals based on assumptions of normality for the following categorical statistics:

- Base Rate
- Forecast Mean
- Accuracy
- Probability of Detection
- Probability of Detection of the non-event
- Probability of False Detection
- False Alarm Ratio
- Critical Success Index
- Hanssen-Kuipers Discriminant
- Odds Ratio
- Log Odds Ratio
- Odds Ratio Skill Score
- Extreme Dependency Score
- Symmetric Extreme Dependency Score
- Extreme Dependency Index
- Symmetric Extremal Dependency Index

MET provides parametric confidence intervals based on assumptions of normality for the following continuous statistics:

- Forecast and Observation Means
- Forecast, Observation, and Error Standard Deviations
- Pearson Correlation Coefficient
- Mean Error

MET provides parametric confidence intervals based on assumptions of normality for the following probabilistic statistics:

- Brier Score
- Base Rate

MET provides non-parametric bootstrap confidence intervals for many categorical and continuous statistics. Kendall's Tau and Spearman's Rank correlation coefficients are the only exceptions. Computing bootstrap confidence intervals for these statistics would be computationally unrealistic.

For more information on confidence intervals pertaining to verification measures, see *Wilks (2011)* (page 500), *Jolliffe and Stephenson (2012)* (page 496), and Bradley (2008).

11.3 Practical Information

The Point-Stat tool is used to perform verification of a gridded model field using point observations. The gridded model field to be verified must be in one of the supported file formats. The point observations must be formatted as the NetCDF output of the point reformatting tools described in Section 7. The Point-Stat tool provides the capability of interpolating the gridded forecast data to the observation points using a variety of methods as described in Section 11.2.1. The Point-Stat tool computes a number of continuous statistics on the matched pair data as well as discrete statistics once the matched pair data have been thresholded.

If no matched pairs are found for a particular verification task, a report listing counts for reasons why the observations were not used is written to the log output at the default verbosity level of 2. If matched pairs are found, this report is written at verbosity level 3. Inspecting these rejection reason counts is the first step in determining why Point-Stat found no matched pairs. The order of the log messages matches the order in which the processing logic is applied. Start from the last log message and work your way up, considering each of the non-zero rejection reason counts. Verbosity level 9 prints a very detailed explanation about why each observation is used or skipped for each verification task.

11.3.1 point_stat Usage

The usage statement for the Point-Stat tool is shown below:

```
Usage: point_stat
       fcst_file
       obs_file
       config_file
       [-ugrid_config config_file]
       [-point_obs file]
       [-obs_valid_beg time]
       [-obs_valid_end time]
       [-outdir path]
       [-log file]
       [-v level]
```

point_stat has three required arguments and can take many optional ones.

11.3.1.1 Required Arguments for point_stat

1. The **fcst_file** argument names the gridded file in either GRIB or NetCDF containing the model data to be verified.
2. The **obs_file** argument indicates the MET NetCDF point observation file to be used for verifying the model. Python embedding for point observations is also supported, as described in Section 38.4.2.
3. The **config_file** argument indicates the name of the configuration file to be used. The contents of the configuration file are discussed below.

11.3.1.2 Optional Arguments for point_stat

5. The **-ugrid_config** option provides a way for a user to provide a separate config file with metadata about their UGRID.

5. The **-point_obs** file may be used to pass additional NetCDF point observation files to be used in the verification. Python embedding for point observations is also supported, as described in Section 38.4.2.

6. The **-obs_valid_beg** time option in YYYYMMDD[_HH[MMSS]] format sets the beginning of the observation matching time window, overriding the configuration file setting.

7. The **-obs_valid_end** time option in YYYYMMDD[_HH[MMSS]] format sets the end of the observation matching time window, overriding the configuration file setting.

8. The **-outdir path** indicates the directory where output files should be written.

9. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.

10. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

An example of the point_stat calling sequence is shown below:

```
point_stat sample_fcst.grb \
sample_pb.nc \
PointStatConfig
```

In this example, the Point-Stat tool evaluates the model data in the sample_fcst.grb GRIB file using the observations in the NetCDF output of PB2NC, sample_pb.nc, applying the configuration options specified in the **PointStatConfig file**.

11.3.2 point_stat Configuration File

The default configuration file for the Point-Stat tool named **PointStatConfig_default** can be found in the installed *share/met/config* directory. Another version is located in *scripts/config*. We encourage users to make a copy of these files prior to modifying their contents. The contents of the configuration file are described in the subsections below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
model                   = "FCST";
desc                    = "NA";
regrid                  = { ... }
climo_mean              = { ... }
climo_stdev             = { ... }
```

(continues on next page)

(continued from previous page)

```
climo_cdf                = { ... }
land_mask                = { ... }
topo_mask                = { ... }
lapse_rate_correction    = { ... }
msl_agl_conversion       = { ... }
obs_window               = { ... }
mask                     = { grid = [ "FULL" ];
                             poly = [];
                             sid = []; }
ci_alpha                 = [ 0.05 ];
boot                     = { ... }
interp                   = { ... }
censor_thresh            = [];
censor_val               = [];
mpr_column               = [];
mpr_thresh               = [];
eclv_points              = 0.05;
hss_ec_value             = NA;
rank_corr_flag           = TRUE;
sid_inc                  = [];
sid_exc                  = [];
duplicate_flag           = NONE;
obs_quality_inc          = [];
obs_quality_exc          = [];
obs_summary              = NONE;
obs_perc_value           = 50;
message_type_group_map   = [ ... ];
obtype_as_group_val_flag = FALSE;
point_weight_flag        = NONE;
tmp_dir                  = "/tmp";
output_prefix            = "";
version                  = "VN.N";
```

The configuration options listed above are common to multiple MET tools and are described in Section 5.

Setting up the **fcst** and **obs** dictionaries of the configuration file is described in Section 5. The following are some special considerations for the Point-Stat tool.

The **obs** dictionary looks very similar to the **fcst** dictionary. When the forecast and observation variables follow the same naming convention, one can easily copy over the forecast settings to the observation dictionary using **obs = fcst;**. However when verifying forecast data in NetCDF format or verifying against not-standard observation variables, users will need to specify the **fcst** and **obs** dictionaries separately. The number of fields specified in the **fcst** and **obs** dictionaries must match.

The **message_type** entry, defined in the **obs** dictionary, contains a comma-separated list of the message types to use for verification. At least one entry must be provided. The Point-Stat tool performs verification using observations for one message type at a time. See Table 1.a Current Table A Entries in PREPBUFR mnemonic

table for a list of the possible types. If using **obs = fcst;**, it can be defined in the forecast dictionary and the copied into the observation dictionary.

```
hira = {
   flag            = FALSE;
   width           = [ 2, 3, 4, 5 ]
   vld_thresh      = 1.0;
   cov_thresh      = [ ==0.25 ];
   shape           = SQUARE;
   prob_cat_thresh = [];
}
```

The **hira** dictionary that is very similar to the **interp** and **nbrhd** entries. It specifies information for applying the High Resolution Assessment (HiRA) verification logic described in section Section 11.2.2. The **flag** entry is a boolean which toggles HiRA on (**TRUE**) and off (**FALSE**). The **width** and **shape** entries define the neighborhood size and shape, respectively. Since HiRA applies to point observations, the width may be even or odd. The **vld_thresh** entry is the required ratio of valid data within the neighborhood to compute an output value. The **cov_thresh** entry is an array of probabilistic thresholds used to populate the Nx2 probabilistic contingency table written to the PCT output line and used for computing probabilistic statistics. The **prob_cat_thresh** entry defines the thresholds to be used in computing the ranked probability score in the RPS output line type. If left empty but climatology data is provided, the **climo_cdf** thresholds will be used instead of **prob_cat_thresh**.

```
output_flag = {
   fho    = BOTH;
   ctc    = BOTH;
   cts    = BOTH;
   mctc   = BOTH;
   mcts   = BOTH;
   cnt    = BOTH;
   sl1l2  = BOTH;
   sal1l2 = BOTH;
   vl1l2  = BOTH;
   vcnt   = BOTH;
   val1l2 = BOTH;
   pct    = BOTH;
   pstd   = BOTH;
   pjc    = BOTH;
   prc    = BOTH;
   ecnt   = BOTH;  // Only for HiRA
   orank  = BOTH;  // Only for HiRA
   rps    = BOTH;  // Only for HiRA
   eclv   = BOTH;
   mpr    = BOTH;
   seeps  = NONE;
```

(continues on next page)

(continued from previous page)

```
   seeps_mpr = NONE;
}
```

The **output_flag** array controls the type of output that the Point-Stat tool generates. Each flag corresponds to an output line type in the STAT file. Setting the flag to NONE indicates that the line type should not be generated. Setting the flag to STAT indicates that the line type should be written to the STAT file only. Setting the flag to BOTH indicates that the line type should be written to the STAT file as well as a separate ASCII file where the data is grouped by line type. The output flags correspond to the following output line types:

1. **FHO** for Forecast, Hit, Observation Rates
2. **CTC** for Contingency Table Counts
3. **CTS** for Contingency Table Statistics
4. **MCTC** for Multi-category Contingency Table Counts
5. **MCTS** for Multi-category Contingency Table Statistics
6. **CNT** for Continuous Statistics
7. **SL1L2** for Scalar L1L2 Partial Sums
8. **SAL1L2** for Scalar Anomaly L1L2 Partial Sums when climatological data is supplied
9. **VL1L2** for Vector L1L2 Partial Sums
10. **VAL1L2** for Vector Anomaly L1L2 Partial Sums when climatological data is supplied
11. **VCNT** for Vector Continuous Statistics
12. **PCT** for Contingency Table counts for Probabilistic forecasts
13. **PSTD** for contingency table Statistics for Probabilistic forecasts with Dichotomous outcomes
14. **PJC** for Joint and Conditional factorization for Probabilistic forecasts
15. **PRC** for Receiver Operating Characteristic for Probabilistic forecasts
16. **ECNT** for Ensemble Continuous Statistics is only computed for the HiRA methodology
17. **ORANK** for Ensemble Matched Pair Information when point observations are supplied for the HiRA methodology
18. **RPS** for Ranked Probability Score is only computed for the HiRA methodology
19. **ECLV** for Economic Cost/Loss Relative Value
20. **MPR** for Matched Pair data
21. **SEEPS** for averaged SEEPS (Stable Equitable Error in Probability Space) score
22. **SEEPS_MPR** for SEEPS score of Matched Pair data

Note that the FHO and CTC line types are easily derived from each other. Users are free to choose which measures are most desired. The output line types are described in more detail in Section 11.3.3.

Note that writing out matched pair data (MPR lines) for a large number of cases is generally not recommended. The MPR lines create very large output files and are only intended for use on a small set of cases.

If all line types corresponding to a particular verification method are set to NONE, the computation of those statistics will be skipped in the code and thus make the Point-Stat tool run more efficiently. For example, if FHO, CTC, and CTS are all set to NONE, the Point-Stat tool will skip the categorical verification step.

The default SEEPS climo file exists at MET_BASE/climo/seeps/PPT24_seepsweights.nc. It is configurable by using the configuration file (seeps_point_climo_name). It can be overridden by the environment variable, MET_SEEPS_POINT_CLIMO_NAME.

11.3.3 point_stat Output

point_stat produces output in STAT and, optionally, ASCII format. The ASCII output duplicates the STAT output but has the data organized by line type. The output files will be written to the default output directory or the directory specified using the "-outdir" command line option.

The output STAT file will be named using the following naming convention:

point_stat_PREFIX_HHMMSSL_YYYYMMDD_HHMMSSV.stat where PREFIX indicates the user-defined output prefix, HHMMSSL indicates the forecast lead time and YYYYMMDD_HHMMSS indicates the forecast valid time.

The output ASCII files are named similarly:

point_stat_PREFIX_HHMMSSL_YYYYMMDD_HHMMSSV_TYPE.txt where TYPE is one of mpr, fho, ctc, cts, cnt, mctc, mcts, pct, pstd, pjc, prc, ecnt, orank, rps, eclv, sl1l2, sal1l2, vl1l2, vcnt or val1l2 to indicate the line type it contains.

The first set of header columns are common to all of the output files generated by the Point-Stat tool. Tables describing the contents of the header columns and the contents of the additional columns for each line type are listed in the following tables. The ECNT line type is described in Table 14.2. The ORANK line type is described in Table 14.7. The RPS line type is described in Table 14.3.

Table 11.1: Header information for Point-Stat STAT output

Column Number	Header Column Name	Description	Data Type
1	VERSION	Version number	String
2	MODEL	User-provided text string designating model name	String
3	DESC	User-provided text string describing the verification task	String
4	FCST_LEAD	Forecast lead time in HHMMSS format	Time String
5	FCST_VALID_BEG	Forecast valid start time in YYYYMMDD_HHMMSS format	Datetime String
6	FCST_VALID_END	Forecast valid end time in YYYYMMDD_HHMMSS format	Datetime String
7	OBS_LEAD	Observation lead time in HHMMSS format	Time String
8	OBS_VALID_BEG	Observation valid start time in YYYYMMDD_HHMMSS format	Datetime String
9	OBS_VALID_END	Observation valid end time in YYYYMMDD_HHMMSS format	Datetime String
10	FCST_VAR	Model variable	String
11	FCST_UNITS	Units for model variable	String
12	FCST_LEV	Selected vertical level for forecast	String
13	OBS_VAR	Observation variable	String
14	OBS_UNITS	Units for observation variable	String
15	OBS_LEV	Selected vertical level for observations	String
16	OBTYPE	Observation message type selected	String
17	VX_MASK	Verifying masking region indicating the masking grid or polyline region applied	String
18	INTERP_MTHD	Interpolation method applied to forecasts	String
19	INTERP_PNTS	Number of points used in interpolation method	Integer
20	FCST_THRESH	The threshold applied to the forecast	Threshold String
21	OBS_THRESH	The threshold applied to the observations	Threshold String
22	COV_THRESH	NA in Point-Stat	Threshold String
23	ALPHA	Error percent value used in confidence intervals	Double
24	LINE_TYPE	Output line types are listed in tables Table 11.2 through Table 11.20.	String

Table 11.2: Format information for FHO (Forecast, Hit rate, Observation rate) output line type.

Column Number	FHO Column Name	Description	Data Type
24	FHO	Forecast, Hit, Observation line type	String
25	TOTAL	Total number of matched pairs	Integer
26	F_RATE	Forecast rate	Double
27	H_RATE	Hit rate	Double
28	O_RATE	Observation rate	Double

Table 11.3: Format information for CTC (Contingency Table Counts) output line type.

Column Number	CTC Column Name	Description	Data Type
24	CTC	Contingency Table Counts line type	String
25	TOTAL	Total number of matched pairs	Integer
26	FY_OY	Sum of weights for hits (forecast yes and observation yes)	Double
27	FY_ON	Sum of weights for false alarms (forecast yes and observation no)	Double
28	FN_OY	Sum of weights for misses (forecast no and observation yes)	Double
29	FN_ON	Sum of weights for correct negatives (forecast no and observation no)	Double
30	EC_VALUE	Expected correct rate, used for CTS HSS_EC	Double

Table 11.4: Format information for CTS (Contingency Table Statistics) output line type.

Column Number	CTS Column Name	Description	Data Type
24	CTS	Contingency Table Statistics line type	String
25	TOTAL	Total number of matched pairs	Integer
26-30	BASER, BASER_NCL, BASER_NCU, BASER_BCL, BASER_BCU	Base rate including normal and bootstrap upper and lower confidence limits	Double
31-35	FMEAN, FMEAN_NCL, FMEAN_NCU, FMEAN_BCL, FMEAN_BCU	Forecast mean including normal and bootstrap upper and lower confidence limits	Double
36-40	ACC, ACC_NCL, ACC_NCU, ACC_BCL, ACC_BCU	Accuracy including normal and bootstrap upper and lower confidence limits	Double
41-43	FBIAS, FBIAS_BCL, FBIAS_BCU	Frequency Bias including bootstrap upper and lower confidence limits	Double
44-48	PODY, PODY_NCL, PODY_NCU, PODY_BCL, PODY_BCU	Probability of detecting yes including normal and bootstrap upper and lower confidence limits	Double
49-53	PODN, PODN_NCL, PODN_NCU, PODN_BCL, PODN_BCU	Probability of detecting no including normal and bootstrap upper and lower confidence limits	Double
54-58	POFD, POFD_NCL, POFD_NCU, POFD_BCL, POFD_BCU	Probability of false detection including normal and bootstrap upper and lower confidence limits	Double
59-63	FAR, FAR_NCL, FAR_NCU, FAR_BCL, FAR_BCU	False alarm ratio including normal and bootstrap upper and lower confidence limits	Double
64-68	CSI, CSI_NCL, CSI_NCU, CSI_BCL, CSI_BCU	Critical Success Index including normal and bootstrap upper and lower confidence limits	Double
69-71	GSS, GSS_BCL, GSS_BCU	Gilbert Skill Score including bootstrap upper and lower confidence limits	Double
72-76	HK, HK_NCL, HK_NCU, HK_BCL, HK_BCU	Hanssen-Kuipers Discriminant including normal and bootstrap upper and lower confidence limits	Double
77-79	HSS, HSS_BCL, HSS_BCU	Heidke Skill Score including bootstrap upper and lower confidence limits	Double
80-84	ODDS, ODDS_NCL, ODDS_NCU, ODDS_BCL, ODDS_BCU	Odds Ratio including normal and bootstrap upper and lower confidence limits	Double
85-89	LODDS, LODDS_NCL, LODDS_NCU, LODDS_BCL, LODDS_BCU	Logarithm of the Odds Ratio including normal and bootstrap upper and lower confidence limits	Double
90-94	ORSS, ORSS _NCL, ORSS _NCU, ORSS _BCL, ORSS _BCU	Odds Ratio Skill Score including normal and bootstrap upper and lower confidence limits	Double
95-99	EDS, EDS _NCL, EDS _NCU, EDS _BCL, EDS _BCU	Extreme Dependency Score including normal and bootstrap upper and lower confidence limits	Double
100-104	SEDS, SEDS _NCL, SEDS _NCU, SEDS _BCL, SEDS _BCU	Symmetric Extreme Dependency Score including normal and bootstrap upper and lower confidence limits	Double
105-109	EDI, EDI _NCL, EDI _NCU, EDI _BCL, EDI _BCU	Extreme Dependency Index including normal and bootstrap upper and lower confidence limits	Double
111-113	SEDI, SEDI _NCL, SEDI _NCU, SEDI _BCL, SEDI _BCU	Symmetric Extremal Dependency Index including normal and bootstrap upper and lower confidence	Double

Table 11.5: Format information for CNT (Continuous Statistics) output line type.

Column Number	CNT Column Name	Description	Data Type
24	CNT	Continuous statistics line type	String
25	TOTAL	Total number of matched pairs	Integer
26-30	FBAR, FBAR_NCL, FBAR_NCU, FBAR_BCL, FBAR_BCU	Forecast mean including normal and bootstrap upper and lower confidence limits	Double
31-35	FSTDEV, FSTDEV_NCL, FSTDEV_NCU, FSTDEV_BCL, FSTDEV_BCU	Standard deviation of the forecasts including normal and bootstrap upper and lower confidence limits	Double
36-40	OBAR, OBAR_NCL, OBAR_NCU, OBAR_BCL, OBAR_BCU	Observation mean including normal and bootstrap upper and lower confidence limits	Double
41-45	OSTDEV, OSTDEV_NCL, OSTDEV_NCU, OSTDEV_BCL, OSTDEV_BCU	Standard deviation of the observations including normal and bootstrap upper and lower confidence limits	Double
46-50	PR_CORR, PR_CORR_NCL, PR_CORR_NCU, PR_CORR_BCL, PR_CORR_BCU	Pearson correlation coefficient including normal and bootstrap upper and lower confidence limits	Double
51	SP_CORR	Spearman's rank correlation coefficient	Double
52	KT_CORR	Kendall's tau statistic	Double
53	RANKS	Number of ranks used in computing Kendall's tau statistic	Integer
54	FRANK_TIES	Number of tied forecast ranks used in computing Kendall's tau statistic	Integer
55	ORANK_TIES	Number of tied observation ranks used in computing Kendall's tau statistic	Integer
56-60	ME, ME_NCL, ME_NCU, ME_BCL, ME_BCU	Mean error (F-O) including normal and bootstrap upper and lower confidence limits	Double
61-65	ESTDEV, ESTDEV_NCL, ESTDEV_NCU, ESTDEV_BCL, ESTDEV_BCU	Standard deviation of the error including normal and bootstrap upper and lower confidence limits	Double
66-68	MBIAS, MBIAS_BCL, MBIAS_BCU	Multiplicative bias including bootstrap upper and lower confidence limits	Double
69-71	MAE, MAE_BCL, MAE_BCU	Mean absolute error including bootstrap upper and lower confidence limits	Double
72-74	MSE, MSE_BCL, MSE_BCU	Mean squared error including bootstrap upper and lower confidence limits	Double
75-77	BCMSE, BCMSE_BCL, BCMSE_BCU	Bias-corrected mean squared error including bootstrap upper and lower con-	Double

Table 11.6: Format information for MCTC (Multi-category Contingency Table Count) output line type.

Column Number	MCTC Column Name	Description	Data Type
24	MCTC	Multi-category Contingency Table Counts line type	String
25	TOTAL	Total number of matched pairs	Integer
26	N_CAT	Dimension of the contingency table	Integer
27	Fi_Oj	Sum of weights for events in forecast category i and observation category j, with the observations incrementing first (repeated)	Double
*	EC_VALUE	Expected correct rate, used for MCTS HSS_EC	Double

Table 11.7: Format information for MCTS (Multi-category Contingency Table Statistics) output line type.

Column Number	MCTS Column Name	Description	Data Type
24	MCTS	Multi-category Contingency Table Statistics line type	String
25	TOTAL	Total number of matched pairs	Integer
26	N_CAT	The total number of categories in each dimension of the contingency table. So the total number of cells is N_CAT*N_CAT.	Integer
27-31	ACC, ACC_NCL, ACC_NCU, ACC_BCL, ACC_BCU	Accuracy, normal confidence limits and bootstrap confidence limits	Double
32-34	HK, HK_BCL, HK_BCU	Hanssen and Kuipers Discriminant and bootstrap confidence limits	Double
35-37	HSS, HSS_BCL, HSS_BCU	Heidke Skill Score and bootstrap confidence limits	Double
38-40	GER, GER_BCL, GER_BCU	Gerrity Score and bootstrap confidence limits	Double
41-43	HSS_EC, HSS_EC_BCL, HSS_EC_BCU	Heidke Skill Score with user-specific expected correct and bootstrap confidence limits	Double
44	EC_VALUE	Expected correct rate, used for MCTS HSS_EC	Double

Table 11.8: Format information for PCT (Contingency Table Counts for Probabilistic forecasts) output line type.

Column Number	PCT Column Name	Description	Data Type
24	PCT	Probability contingency table count line type	String
25	TOTAL	Total number of matched pairs	Integer
26	N_THRESH	Number of probability thresholds	Integer
27	THRESH_i	The ith probability threshold value (repeated)	Double
28	OY_i	Sum of weights for observation yes when forecast is between the ith and i+1th probability thresholds (repeated)	Double
29	ON_i	Sum of weights for observation no when forecast is between the ith and i+1th probability thresholds (repeated)	Double
*	THRESH_n	Last probability threshold value	Double

Table 11.9: Format information for PSTD (Contingency Table Statistics for Probabilistic forecasts) output line type

Column Number	PSTD Column Name	Description	Data Type
24	PSTD	Probabilistic statistics for dichotomous outcome line type	String
25	TOTAL	Total number of matched pairs	Integer
26	N_THRESH	Number of probability thresholds	Integer
27-29	BASER, BASER_NCL, BASER_NCU	The Base Rate, including normal upper and lower confidence limits	Double
30	RELIABILITY	Reliability	Double
31	RESOLUTION	Resolution	Double
32	UNCERTAINTY	Uncertainty	Double
33	ROC_AUC	Area under the receiver operating characteristic curve	Double
34-36	BRIER, BRIER_NCL, BRIER_NCU	Brier Score including normal upper and lower confidence limits	Double
37-39	BRIERCL, BRIERCL_NCL, BRIERCL_NCU	Climatological Brier Score including upper and lower normal confidence limits	Double
40	BSS	Brier Skill Score relative to external climatology	Double
41	BSS_SMPL	Brier Skill Score relative to sample climatology	Double
42	THRESH_i	The ith probability threshold value (repeated)	Double

Table 11.10: Format information for PJC (Joint and Conditional factorization for Probabilistic forecasts) output line type.

Column Number	PJC Column Name	Description	Data Type
24	PJC	Probabilistic Joint/Continuous line type	String
25	TOTAL	Total number of matched pairs	Integer
26	N_THRESH	Number of probability thresholds	Integer
27	THRESH_i	The ith probability threshold value (repeated)	Double
28	OY_TP_i	Number of observation yes when forecast is between the ith and i+1th probability thresholds as a proportion of the total OY (repeated)	Double
29	ON_TP_i	Number of observation no when forecast is between the ith and i+1th probability thresholds as a proportion of the total ON (repeated)	Double
30	CALIBRATION_i	Calibration when forecast is between the ith and i+1th probability thresholds (repeated)	Double
31	REFINEMENT_i	Refinement when forecast is between the ith and i+1th probability thresholds (repeated)	Double
32	LIKELIHOOD_i	Likelihood when forecast is between the ith and i+1th probability thresholds (repeated)	Double
33	BASER_i	Base rate when forecast is between the ith and i+1th probability thresholds (repeated)	Double
*	THRESH_n	Last probability threshold value	Double

Table 11.11: Format information for PRC (Receiver Operating Characteristic for Probabilistic forecasts) output line type.

Column Number	PRC Column Name	Description	Data Type
24	PRC	Probability ROC points line type	String
25	TOTAL	Total number of matched pairs	Integer
26	N_THRESH	Number of probability thresholds	Integer
27	THRESH_i	The ith probability threshold value (repeated)	Double
28	PODY_i	Probability of detecting yes when forecast is greater than the ith probability thresholds (repeated)	Double
29	POFD_i	Probability of false detection when forecast is greater than the ith probability thresholds (repeated)	Double
*	THRESH_n	Last probability threshold value	Double

Table 11.12: Format information for ECLV (Economic Cost/Loss Relative Value) output line type.

Column Number	ECLV Column Name	Description	Data Type
24	ECLV	Economic Cost/Loss Relative Value line type	String
25	TOTAL	Total number of matched pairs	Integer
26	BASER	Base rate	Double
27	VALUE_BASER	Economic value of the base rate	Double
28	N_PNT	Number of Cost/Loss ratios	Integer
29	CL_i	ith Cost/Loss ratio evaluated	Double
30	VALUE_i	Relative value for the ith Cost/Loss ratio	Double

Table 11.13: Format information for SL1L2 (Scalar Partial Sums) output line type.

Column Number	SL1L2 Column Name	Description	Data Type
24	SL1L2	Scalar L1L2 line type	String
25	TOTAL	Total number of matched pairs of forecast (f) and observation (o)	Integer
26	FBAR	Mean(f)	Double
27	OBAR	Mean(o)	Double
28	FOBAR	Mean(f*o)	Double
29	FFBAR	Mean(f^2)	Double
30	OOBAR	Mean(o^2)	Double
31	MAE	Mean(|f-o|)	Double

Table 11.14: Format information for SAL1L2 (Scalar Anomaly Partial Sums) output line type.

Column Number	SAL1L2 Column Name	Description	Data Type
24	SAL1L2	Scalar Anomaly L1L2 line type	String
25	TOTAL	Total number of matched pairs of forecast (f), observation (o), forecast climatology (cf), and observation climatology (co)	Integer
26	FABAR	Mean(f-cf)	Double
27	OABAR	Mean(o-co)	Double
28	FOABAR	Mean((f-cf)*(o-co))	Double
29	FFABAR	Mean($(f-cf)^2$)	Double
30	OOABAR	Mean($(o-co)^2$)	Double
31	MAE	Mean(|(f-cf)-(o-co)|)	Double

Table 11.15: Format information for VL1L2 (Vector Partial Sums) output line type.

Column Number	VL1L2 Column Name	Description	Data Type
24	VL1L2	Vector L1L2 line type	String
25	TOTAL	Total number of matched pairs of forecast winds (uf, vf) and observation winds (uo, vo)	Integer
26	UFBAR	Mean(uf)	Double
27	VFBAR	Mean(vf)	Double
28	UOBAR	Mean(uo)	Double
29	VOBAR	Mean(vo)	Double
30	UVFOBAR	Mean(uf*uo+vf*vo)	Double
31	UVFFBAR	Mean(uf^2+vf^2)	Double
32	UVOOBAR	Mean(uo^2+vo^2)	Double
33	F_SPEED_BAR	Mean forecast wind speed	Double
34	O_SPEED_BAR	Mean observed wind speed	Double
35	TOTAL_DIR	Total number of matched pairs for which both the forecast and observation wind directions are well-defined (i.e. non-zero vectors)	Double
36	DIR_ME	Mean wind direction difference, from -180 to 180 degrees	Double
37	DIR_MAE	Mean absolute wind direction difference	Double
38	DIR_MSE	Mean squared wind direction difference	Double

Table 11.16: Format information for VAL1L2 (Vector Anomaly Partial Sums) output line type.

Column Number	VAL1L2 Column Name	Description	Data Type
24	VAL1L2	Vector Anomaly L1L2 line type	String
25	TOTAL	Total number of matched pairs of forecast winds (uf, vf), observation winds (uo, vo), forecast climatology winds (ucf, vcf), and observation climatology winds (uco, vco)	Integer
26	UFABAR	Mean(uf-ucf)	Double
27	VFABAR	Mean(vf-vcf)	Double
28	UOABAR	Mean(uo-uco)	Double
29	VOABAR	Mean(vo-vco)	Double
30	UVFOABAR	Mean((uf-ucf)*(uo-uco)+(vf-vcf)*(vo-vco))	Double
31	UVFFABAR	Mean$((\text{uf-ucf})^2+(\text{vf-vcf})^2)$	Double
32	UVOOABAR	Mean$((\text{uo-uco})^2+(\text{vo-vco})^2)$	Double
33	FA_SPEED_BAR	Mean forecast wind speed anomaly	Double
34	OA_SPEED_BAR	Mean observed wind speed anomaly	Double
35	TOTAL_DIR	Total number of matched pairs for which the forecast, observation, forecast climatology, and observation climatology wind directions are well-defined (i.e. non-zero vectors)	Double
36	DIRA_ME	Mean wind direction anomaly difference, from -180 to 180 degrees	Double
37	DIRA_MAE	Mean absolute wind direction anomaly difference	Double
38	DIRA_MSE	Mean squared wind direction anomaly difference	Double

Table 11.17: Format information for VCNT (Vector Continuous Statistics) output line type. Note that the bootstrap confidence intervals columns ending with BCL and BCU are not currently calculated for this release of MET, but will be in future releases.

Column Numbers	VCNT Column Name	Description	Data Type
24	VCNT	Vector Continuous Statistics line type	String
25	TOTAL	Total number of data points	Integer
26-28	FBAR, FBAR_BCL, FBAR_BCU	Mean value of forecast wind speed including bootstrap upper and lower confidence limits	Double
29-31	OBAR, OBAR_BCL, OBAR_BCU	Mean value of observed wind speed including bootstrap upper and lower confidence limits	Double
32-34	FS_RMS, FS_RMS_BCL, FS_RMS_BCU	Root mean square forecast wind speed including bootstrap upper and lower confidence limits	Double
35-37	OS_RMS, OS_RMS_BCL, OS_RMS_BCU	Root mean square observed wind speed including bootstrap upper and lower confidence limits	Double
38-40	MSVE, MSVE_BCL, MSVE_BCU	Mean squared length of the vector difference between the forecast and observed winds including bootstrap upper and lower confidence limits	Double
41-43	RMSVE, RMSVE_BCL, RMSVE_BCU	Square root of MSVE including bootstrap upper and lower confidence limits	Double
45-46	FSTDEV, FSTDEV_BCL, FSTDEV_BCU	Standard deviation of the forecast wind speed including bootstrap upper and lower confidence limits	Double
47-49	OSTDEV, OSTDEV_BCL, OSTDEV_BCU	Standard deviation of the observed wind field including bootstrap upper and lower confidence limits	Double
50-52	FDIR, FDIR_BCL, FDIR_BCU	Direction of the average forecast wind vector including bootstrap upper and lower confidence limits	Double
53-55	ODIR, ODIR_BCL, ODIR_BCU	Direction of the average observed wind vector including bootstrap upper and lower confidence limits	Double
56-58	FBAR_SPEED, FBAR_SPEED_BCL, FBAR_SPEED_BCU	Length (speed) of the average forecast wind vector including bootstrap upper and lower confidence limits	Double
59-61	OBAR_SPEED, OBAR_SPEED_BCL, OBAR_SPEED_BCU	Length (speed) of the average observed wind vector including bootstrap upper and lower confidence limits	Double
62-64	VDIFF_SPEED, VDIFF_SPEED_BCL, VDIFF_SPEED_BCU	Length (speed) of the vector difference between the average forecast and average observed wind vectors including bootstrap upper and lower confidence limits	Double
65-67	VDIFF_DIR, VDIFF_DIR_BCL, VDIFF_DIR_BCU	Direction of the vector difference between the average forecast and average wind vectors including bootstrap upper and lower confidence limits	Double
68-70	SPEED_ERR, SPEED_ERR_BCL, SPEED_ERR_BCU	Difference between the length of the average forecast wind vector and the average observed wind vector (in the sense F - O) including bootstrap upper and lower confidence limits	Double
71-73	SPEED_ABSERR, SPEED_ABSERR_BCL, SPEED_ABSERR_BCU	Absolute value of SPEED_ERR including bootstrap upper and lower confidence limits	Double

Table 11.18: Format information for MPR (Matched Pair) output line type.

Column Number	MPR Column Name	Description	Data Type
24	MPR	Matched Pair line type	String
25	TOTAL	Total number of matched pairs	Integer
26	INDEX	Index for the current matched pair	Integer
27	OBS_SID	Station Identifier of observation	String
28	OBS_LAT	Latitude of the observation in degrees north	Double
29	OBS_LON	Longitude of the observation in degrees east	Double
30	OBS_LVL	Pressure level of the observation in hPa or accumulation interval in hours	Double
31	OBS_ELV	Elevation of the observation in meters above sea level	Double
32	FCST	Forecast value interpolated to the observation location	Double
33	OBS	Observation value	Double
34	OBS_QC	Quality control flag for observation	String
35	OBS_CLIMO_MEAN	Observation climatological mean value (named CLIMO_MEAN prior to met-12.0.0)	Double
36	OBS_CLIMO_STDEV	Observation climatological standard deviation value (named CLIMO_STDEV prior to met-12.0.0)	Double
37	OBS_CLIMO_CDF	Observation climatological cumulative distribution function value (named CLIMO_CDF prior to met-12.0.0)	Double
38	FCST_CLIMO_MEAN	Forecast climatological mean value	Double
39	FCST_CLIMO_STDEV	Forecast climatological standard deviation value	Double

Table 11.19: Format information for SEEPS (Stable Equitable Error in Probability Space) of MPR (Matched Pair) output line type.

Column Number	SEEPS_MPR Column Name	Description	Data Type
24	SEEPS_MPR	SEEPS Matched Pair line type	String
25	OBS_SID	Station Identifier of observation	String
26	OBS_LAT	Latitude of the observation in degrees north	Double
27	OBS_LON	Longitude of the observation in degrees east	Double
28	FCST	Forecast value interpolated to the observation location	Double
29	OBS	Observation value	Double
30	OBS_QC	Quality control flag for observation	String
31	FCST_CAT	Forecast category (dry, light, or heavy)	Integer
32	OBS_CAT	Observation category (dry, light, or heavy)	Integer
33	P1	Climo-derived probability value for this station (dry)	Double
34	P2	Climo-derived probability value for this station (dry + light)	Double
35	T1	Threshold 1 for P1 (dry)	Double
36	T2	Threshold 2 for P2 (dry + light)	Double
37	SEEPS	SEEPS (Stable Equitable Error in Probability Space) score	Double

Table 11.20: Format information for SEEPS (Stable Equitable Error in Probability Space) output line type.

Column Number	SEEPS Column Name	Description	Data Type
24	SEEPS	SEEPS line type	String
25	TOTAL	Total number of SEEPS matched pairs	Integer
26	ODFL	Counts multiplied by the weights for the observation dry, forecast light category	Double
27	ODFH	Counts multiplied by the weights for the observation dry, forecast heavy category	Double
28	OLFD	Counts multiplied by the weights for the observation light, forecast dry category	Double
29	OLFH	Counts multiplied by the weights for the observation light, forecast heavy category	Double
30	OHFD	Counts multiplied by the weights for the observation heavy, forecast dry category	Double
31	OHFL	Counts multiplied by the weights for the observation heavy, forecast light category	Double
32	PF1	Marginal probabilities of the forecast dry (FCST_CAT 0)	Double
33	PF2	Marginal probabilities of the forecast light (FCST_CAT 1)	Double
34	PF3	Marginal probabilities of the forecast heavy (FCST_CAT 2)	Double
35	PV1	Marginal probabilities of the observed dry (OBS_CAT 0)	Double
36	PV2	Marginal probabilities of the observed light (OBS_CAT 1)	Double
37	PV3	Marginal probabilities of the observed heavy (OBS_CAT 2)	Double
38	SEEPS	Averaged SEEPS (Stable Equitable Error in Probability Space) score	Double

The STAT output files described for point_stat may be used as inputs to the Stat-Analysis tool. For more information on using the Stat-Analysis tool to create stratifications and aggregations of the STAT files produced by point_stat, please see Section 17.

Chapter 12

Pair-Stat Tool

12.1 Introduction

The Pair-Stat tool provides verification statistics for forecast and observation data that has already been paired in time and space. While no smoothing, regridding, or interpolation methods apply to the forecast and observation pairs, the Pair-Stat tool filters and groups the pairs temporally and spatially. It then computes continuous, categorical, and probabilistic verification statistics. The categorical and probabilistic statistics are generally derived by applying a threshold to the forecast and observation values. Confidence intervals - representing the uncertainty in the verification measures - are computed for the verification statistics.

Scientific and statistical aspects of the Pair-Stat tool are discussed in the following section. Practical aspects of the Pair-Stat tool are described in Section 12.3.

12.2 Scientific and Statistical Aspects

The statistics and measures computed by the Pair-Stat tool are a subset of those computed by the Point-Stat tool which are described briefly in Section 11.2.4 and in more detail in Appendix C, Section 35. Additionally, Section 11.2.5 describes the methods for computing confidence intervals that are applied to some of the measures computed by the Pair-Stat tool; more detail on confidence intervals is provided in Appendix D, Section 36.

12.3 Practical Information

The Pair-Stat tool performs verification for forecast and observation data that has already been paired in time and space. The paired data is supplied via one of the supported input formats, including the *MET MPR Line Type* (page 239) written by the Point-Stat tool with the **-format mpr** command line option, through *Python embedding* (page 601) with the **-format python** command line option, and from IODA (Interface for Observation Data Access) files described in the *IODA2NC Tool* (page 169) with the **-format ioda** command line option. The Stat-Analysis tool also processes MPR data, so the functionality of Pair-Stat and Stat-Analysis overlap in this way. Based on configuration file settings, paired data for each verification task requested is extracted from one or more input files, subsetted temporally and spatially, and used to compute and write

a variety of statistics and measures. If forecast and/or observation climatology data is provided in the configuration file, it is interpolated to the location of each pair and used in the computation of statistics.

If no matched pairs are found for a particular verification task, no statistics are computed or written to the output.

12.3.1 pair_stat Usage

The usage statement for the Pair-Stat tool is shown below:

```
Usage: pair_stat
       -pairs file_1 ... file_n | file_list
       -format type
       -config config_file
       [-out base]
       [-log file]
       [-v level]
```

pair_stat has three required arguments and accepts optional ones.

12.3.1.1 Required Arguments for pair_stat

1. The **-pairs** argument defines one or more input files containing forecast/observation pairs. May be set as a list of file names (**file_1 ... file_n**) or as an ASCII file containing a list file names (**file_list**), as described in Section 4.1.1 (required).

   This option can be used multiple times but all inputs must follow the same **-format type**, described below.

   For **-format python**, the **-pairs file** defines the path to a Python embedding script to be run followed by any arguments to that script and enclosed in single or double quotes.

2. The **-format type** argument defines the input pairs file format and may be set to "mpr", "python", or "ioda" (required).

3. The **-config config_file** argument is a PairStatConfig file containing the desired configuration settings (required).

12.3.1.2 Optional Arguments for pair_stat

4. The **-out base** option overrides the default output file base (./pair_stat) (optional).

   Each output file begins with this output file base followed by the ".stat" or "_TYPE.txt" suffix, where TYPE is a specific output line type. Users should set **-out base** appropriately to avoid overwriting output generated by previous runs of the Pair-Stat tool.

5. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.

6. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

An example of the pair_stat calling sequence is shown below:

```
pair_stat \
-pairs point_stat_run1_mpr.txt point_stat_run2.stat \
-format mpr \
-config PairStatConfig
```

In this example, the Pair-Stat tool reads matched pair data (**-format mpr**) from **point_stat_run1_mpr.txt** and **point_stat_run2.stat** files and applies the configuration options specified in the **PairStatConfig** file.

12.3.2 pair_stat Configuration File

The default configuration file for the Pair-Stat tool named **PairStatConfig_default** can be found in the installed *share/met/config* directory. Users are encouraged to make a copy prior to modifying its contents. The configuration file options are described in the subsections below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
model             = "FCST";
desc              = "NA";
point_weight_flag = NONE;
tmp_dir           = "/tmp";
version           = "VN.N";
```

The configuration options listed above are common to multiple MET tools and are described in Section 5.

```
fcst = {
  pairs = [
    {
      name  = "TMP";
      level = [ "Z2" ];
    }
  ];
}

obs = {
  pairs = [
    {
      name = "TMP";
      level = [ "Z2" ];
```

(continues on next page)

(continued from previous page)

```
    }
  ];
}
```

The **fcst** and **obs** entries are dictionaries containing the **pairs** entry which is an array of dictionaries. The **fcst.pairs** and **obs.pairs** arrays must have the same non-zero length. The formatting of **fcst.pairs** is the same as **fcst.field**, described in Section 5.2.13. The **name** and **level** entries in each array entry vary based on the input file format:

- For the **mpr** and **python** formats, set **name** and **level** based on the desired values of the input *MET MPR Line Type* (page 239). Set **fcst.pairs.name** and **obs.pairs.name** to the desired values of the **FCST_VAR** and **OBS_VAR** columns, respectively. Set **fcst.pairs.level** and and **obs.pairs.level** to one or more desired values of the **FCST_LEV** and **OBS_LEV** columns, respectively. Only MPR lines whose variable names match those requested and whose level strings appear in the list of requested level strings will be used for that verification task.
- For the **ioda** format, the **name** entry specifies the input IODA NetCDF variable to be read. The **level** entry does not apply. IODA files contain multiple variables indexed by a point location dimension. Data read from the **fcst.pairs.name** and **obs.pairs.name** variables define the matched pairs for that verification task.

Note: IODA files typically use NetCDF4 groups, and the **name** entry should specify both the group and variable names, formatted as `name = "/GROUP_NAME/VARIABLE_NAME";` (e.g. `name = "/hofx/air_temperature";`).

```
convert(x)    = ...
censor_thresh = [];
censor_val    = [];
cat_thresh    = [ NA ];
cnt_thresh    = [ NA ];
cnt_logic     = UNION;
wind_thresh   = [ NA ];
wind_logic    = UNION;
```

The configuration options listed above are used to process the input paired data and define thresholds for filtering data and for categorical verification. They can be speficied separately for each verification task inside in each **fcst.pairs** or **obs.pairs** array entry. They are common to multiple MET tools and are described in Section 5.

```
mpr_column = [];
mpr_thresh = [];
```

The configuration options listed above filter the input paired data using numeric thresholds. They can be specified separately for each verification task inside each **obs.pairs** array entry. They are common to multiple MET tools and are described in Section 5.

```
mpr_str_inc = [];
mpr_str_exc = [];
```

The configuration options listed above filter the input paired data by specifying input column strings to be included or excluded. They can be specified separately for each verification task inside each **obs.pairs** array entry. The options define arrays of dictionaries with each dictionary containing a **key** and **val** entry. The **key** entry defines the input MPR column name to be filtered and the **val** entry defines a comma-separated list of values.

For example, the following settings *include* only MPR lines with the **OBTYPE** column set to **ADPUPA** and *exclude* any MPR lines with the **VX_MASK** column set to **DTC165** or **DTC166**.

```
mpr_str_inc = [ { key = "OBTYPE";  val = "ADPUPA";        } ];
mpr_str_exc = [ { key = "VX_MASK"; val = "DTC165,DTC166"; } ];
```

```
fcst_lead       = [];
obs_lead        = [];

fcst_valid_beg  = "";
fcst_valid_end  = "";
fcst_valid_inc  = [];
fcst_valid_exc  = [];
fcst_valid_hour = [];

obs_valid_beg   = "";
obs_valid_end   = "";
obs_valid_inc   = [];
obs_valid_exc   = [];
obs_valid_hour  = [];

fcst_init_beg   = "";
fcst_init_end   = "";
fcst_init_inc   = [];
fcst_init_exc   = [];
fcst_init_hour  = [];

obs_init_beg    = "";
obs_init_end    = "";
obs_init_inc    = [];
obs_init_exc    = [];
obs_init_hour   = [];
```

The configuration options listed above filter the input paired data temporally. They can be specified separately for each verificaiton task inside each **obs.pairs** array entry. They are also supported for the Stat-Analysis tool and are described in Section 17.3.2.

```
eclv_points     = 0.05;
hss_ec_value    = NA;
rank_corr_flag  = FALSE;
ci_alpha        = [ 0.05 ];
boot            = { interval = PCTILE; rep_prop = 1.0; n_rep = 1000;
                    rng = "mt19937"; seed = ""; }
seeps_p1_thresh = >=0.1&&<=0.85;
```

The configuration options listed above define verification logic options. They can be specified separately for each verification task inside each **obs.pairs** array entry. They are common to multiple MET tools and are described in Section 5.

```
climo_mean  = { ... }
climo_stdev = { ... }
climo_cdf   = { ... }
```

The configuration options listed above specify climatological input data. They can be specified separately inside the **fcst** and **obs** dictionaries. They are common to multiple MET tools and are described in Section 5.

The length of the **climo_mean.field** and **climo_stdev.field** arrays must match and can either be zero (for no climatological data) or match the **fcst.pairs** and **obs.pairs** array length.

Note: When interpolating climatological data to pair data locations, the grid on which the climatological data is defined is used with the **nearest neighbor** interpolation method and that cannot currently be overridden. A future enhancement may make these interpolation options configurable.

```
mask = {
  grid  = [ "FULL", "DTC165", "DTC166" ];
  poly  = [];
  sid   = [];
  llpnt = [];
}
```

The **mask** dictionary defines how the input paired data is aggregated spatially when computing statistics. For each verification task, output statistics will be computed for each masking region defined. This is common to multiple MET tools and are described in Section 5.

Note: The **mask.grid** and **mask.poly** options are currently defined relative to a reference grid. Since no grid applies to the input paired data, a **global 1/10 degree reference grid** is used by default and that grid cannot currently be overridden. A future enhancement may eliminate the use of a reference grid in this context.

```
output_flag = {
   fho    = BOTH;
   ctc    = BOTH;
   cts    = BOTH;
   mctc   = BOTH;
   mcts   = BOTH;
   cnt    = BOTH;
   sl1l2  = BOTH;
   sal1l2 = BOTH;
   vl1l2  = BOTH;
   vcnt   = BOTH;
   val1l2 = BOTH;
   pct    = BOTH;
   pstd   = BOTH;
   pjc    = BOTH;
   prc    = BOTH;
   eclv   = BOTH;
   mpr    = BOTH;
   seeps  = NONE;
   seeps_mpr = NONE;
}
```

The **output_flag** array controls the type of output that the Pair-Stat tool generates. These output types are a subset of those supported by the Point-Stat tool and are described in Section 11.3.2.

12.3.3 pair_stat Output

The Pair-Stat tool produces output in STAT and, optionally, ASCII format. The ASCII output duplicates the STAT output but has the data organized by line type. The output files names begin with **./pair_stat** or the string specified with the **-out base** command line option. Users should set the **-out base** command line option appropriately to avoid overwriting output generated by previous runs of the Pair-Stat tool.

The output STAT file is named by appending **.stat** to the output base string.

The output ASCII files are named by appending **_TYPE.txt** to the output base string for each output line TYPE set to **BOTH** in the **output_flag** configuration dictionary.

Chapter 13

Grid-Stat Tool

13.1 Introduction

The Grid-Stat tool functions in much the same way as the Point-Stat tool, except that the verification statistics it calculates are for a matched forecast-observation grid (as opposed to a set of observation points). Neither the forecast nor the observation grid needs to be identical to the final matched grid. If the forecast grid is different from the final matched grid, then forecast values are regridded (interpolated) to the final matched grid. The same procedure is followed for observations. No regridding is necessary if the forecast and observation grids are identical but remains optional. A smoothing operation may be performed on the forecast and observation fields prior to verification. All the matched forecast-observation grid points are used to compute the verification statistics. In addition to traditional verification approaches, the Grid-Stat tool includes Fourier decompositions, gradient statistics, distance metrics, and neighborhood methods, designed to examine forecast performance as a function of spatial scale.

Scientific and statistical aspects of the Grid-Stat tool are briefly described in this section. Practical aspects of the Grid-Stat tool are described in Section 13.3.

13.2 Scientific and Statistical Aspects

13.2.1 Statistical Measures

The Grid-Stat tool computes a wide variety of verification statistics. Broadly speaking, these statistics can be subdivided into three types of statistics: measures for categorical variables, measures for continuous variables, and measures for probabilistic forecasts. Further, when a climatology file is included, reference statistics for the forecasts compared to the climatology can be calculated. These categories of measures are briefly described here; specific descriptions of all measures are provided in Appendix C, Section 35. Additional information can be found in *Wilks (2011)* (page 500) and *Jolliffe and Stephenson (2012)* (page 496), and on the Collaboration for Australian Weather and Climate Research Forecast Verification - Issues, Methods and FAQ web page.

In addition to these verification measures, the Grid-Stat tool also computes partial sums and other FHO statistics that are produced by the NCEP verification system. These statistics are also described in Appendix C, Section 35.

13.2.1.1 Measures for Categorical Variables

Categorical verification statistics are used to evaluate forecasts that are in the form of a discrete set of categories rather than on a continuous scale. Grid-Stat computes both 2x2 and multi-category contingency tables and their associated statistics, similar to Point-Stat. See Appendix C, Section 35 for more information.

13.2.1.2 Measures for Continuous Variables

For continuous variables, many verification measures are based on the forecast error (i.e., f - o). However, it also is of interest to investigate characteristics of the forecasts, and the observations, as well as their relationship. These concepts are consistent with the general framework for verification outlined by *Murphy and Winkler (1987)* (page 497). The statistics produced by MET for continuous forecasts represent this philosophy of verification, which focuses on a variety of aspects of performance rather than a single measure. See Appendix C, Section 35 for specific information.

A user may wish to eliminate certain values of the forecasts from the calculation of statistics, a process referred to here as "conditional verification". For example, a user may eliminate all temperatures above freezing and then calculate the error statistics only for those forecasts of below freezing temperatures. Another common example involves verification of wind forecasts. Since wind direction is indeterminate at very low wind speeds, the user may wish to set a minimum wind speed threshold prior to calculating error statistics for wind direction. The user may specify these thresholds in the configuration file to specify the conditional verification. Thresholds can be specified using the usual Fortran conventions (<, <=, ==, !-, >=, or >) followed by a numeric value. The threshold type may also be specified using two letter abbreviations (lt, le, eq, ne, ge, gt). Further, more complex thresholds can be achieved by defining multiple thresholds and using && or || to string together event definition logic. The forecast and observation threshold can be used together according to user preference by specifying one of: UNION, INTERSECTION, or SYMDIFF (symmetric difference).

13.2.1.3 Measures for Probabilistic Forecasts and Dichotomous Outcomes

For probabilistic forecasts, many verification measures are based on reliability, accuracy and bias. However, it also is of interest to investigate joint and conditional distributions of the forecasts and the observations, as in *Wilks (2011)* (page 500). See Appendix C, Section 35 for specific information.

Probabilistic forecast values are assumed to have a range of either 0 to 1 or 0 to 100. If the max data value is > 1, we assume the data range is 0 to 100, and divide all the values by 100. If the max data value is <= 1, then we use the values as is. Further, thresholds are applied to the probabilities with equality on the lower end. For example, with a forecast probability p, and thresholds t1 and t2, the range is defined as: t1 <= p < t2. The exception is for the highest set of thresholds, when the range includes 1: t1 <= p <= 1. To make configuration easier, since METv6.0, these probabilities may be specified in the configuration file as a list (>0.00,>0.25,>0.50,>0.75,>1.00) or using shorthand notation (==0.25) for bins of equal width.

Since METv6.0, when the "prob" entry is set as a dictionary to define the field of interest, setting "prob_as_scalar = TRUE" indicates that this data should be processed as regular scalars rather than probabilities.For example, this option can be used to compute traditional 2x2 contingency tables and neighborhood verification statistics for probability data. It can also be used to compare two probability fields directly.

13.2.1.4 Use of a Climatology Field for Comparative Verification

The Grid-Stat tool allows evaluation of model forecasts compared with a user-supplied climatology. Prior to calculation of statistics, the climatology must be put on the same grid as the forecasts and observations. In particular, the anomaly correlation and mean squared error skill score provide a measure of the forecast skill versus the climatology. For more details about climatological comparisons and reference forecasts, see the relevant section in the Point-Stat Chapter: Section 11.2.4.4.

13.2.1.5 Use of Analysis Fields for Verification

The Grid-Stat tool allows evaluation of model forecasts using model analysis fields. However, users are cautioned that an analysis field is not independent of its parent model; for this reason verification of model output using an analysis field from the same model is generally not recommended and is not likely to yield meaningful information about model performance.

13.2.2 Statistical Confidence Intervals

The confidence intervals for the Grid-Stat tool are the same as those provided for the Point-Stat tool except that the scores are based on pairing grid points with grid points so that there are likely more values for each field making any assumptions based on the central limit theorem more likely to be valid. However, it should be noted that spatial (and temporal) correlations are not presently taken into account in the confidence interval calculations. Therefore, confidence intervals reported may be somewhat too narrow (e.g., *Efron, 2007* (page 494)). See Appendix D, Section 36 for details regarding confidence intervals provided by MET.

13.2.3 Grid Weighting

When computing continuous statistics on a regular large scale or global latitude-longitude grid, weighting may be applied in order to compensate for the meridian convergence toward higher latitudes. Grid square area weighting or weighting based on the cosine of the latitude are two configuration options in both point-stat and grid-stat. See Section 5 for more information.

13.2.4 Neighborhood Methods

MET also incorporates several neighborhood methods to give credit to forecasts that are close to the observations, but not necessarily exactly matched up in space. Also referred to as "fuzzy" verification methods, these methods do not just compare a single forecast at each grid point to a single observation at each grid point; they compare the forecasts and observations in a neighborhood surrounding the point of interest. With the neighborhood method, the user chooses a distance within which the forecast event can fall from the observed event and still be considered a hit. In MET this is implemented by defining a square search window around each grid point. Within the search window, the number of observed events is compared to the number of forecast events. In this way, credit is given to forecasts that are close to the observations without requiring a strict match between forecasted events and observed events at any particular grid point. The neighborhood methods allow the user to see how forecast skill varies with neighborhood size and can help determine the smallest neighborhood size that can be used to give sufficiently accurate forecasts.

There are several ways to present the results of the neighborhood approaches, such as the Fractions Skill Score (FSS) or the Fractions Brier Score (FBS). These scores are presented in Appendix C, Section 35. One can also simply up-scale the information on the forecast verification grid by smoothing or resampling within a specified neighborhood around each grid point and recalculate the traditional verification metrics on the coarser grid. The MET output includes traditional contingency table statistics for each threshold and neighborhood window size.

The user must specify several parameters in the grid_stat configuration file to utilize the neighborhood approach, such as the interpolation method, size of the smoothing window, and required fraction of valid data points within the smoothing window. For FSS-specific results, the user must specify the size of the neighborhood window, the required fraction of valid data points within the window, and the fractional coverage threshold from which the contingency tables are defined. These parameters are described further in the practical information section below.

13.2.5 SEEPS Scores

The Stable Equitable Error in Probability Space (SEEPS) was devised for monitoring global deterministic forecasts of precipitation against the WMO gauge network (*Rodwell et al., 2010* (page 498); *Haiden et al., 2012* (page 496)) and is a multi-category score which uses a climatology to account for local variations in behavior. Please see Point-Stat documentation Section 11.2.3 for more details.

The capability to calculate the SEEPS has also been added to Grid-Stat. This follows the method described in *North et al, 2022* (page 497), which uses the TRMM 3B42 v7 gridded satellite product for the climatological values and interpolates the forecast and observed products onto this grid for evaluation. A 24-hour TRMM climatology (valid at 00 UTC) constructed from data over the time period 1998-2015 is supplied with the release. Expansion of the capability to other fields will occur as well vetted examples and funding allow.

The gridded climatology required to compute SEEPS is not distributed as part of the code release and can be downloaded from Zenodo. The path to the file needs to be specified using MET_SEEPS_GRID_CLIMO_NAME.

13.2.6 Fourier Decomposition

The MET software will compute the full one-dimensional Fourier transform, then do a partial inverse transform based on the two user-defined wave numbers. These two wave numbers define a band pass filter in the Fourier domain. This process is conceptually similar to the operation of projecting onto subspace in linear algebra. If one were to sum up all possible wave numbers the result would be to simply reproduce the raw data.

Decomposition via Fourier transform allows the user to evaluate the model separately at each spatial frequency. As an example, the Fourier analysis allows users to examine the "dieoff", or reduction, in anomaly correlation of geopotential height at various levels for bands of waves. A band of low wave numbers, say 0 - 3, represent larger frequency components, while a band of higher wave numbers, for example 70 - 72, represent smaller frequency components. Generally, anomaly correlation should be higher for frequencies with low wave numbers than for frequencies with high wave numbers, hence the "dieoff".

Wavelets, and in particular the MET wavelet tool, can also be used to define a band pass filter (*Casati et al., 2004* (page 493); *Weniger et al., 2016* (page 499)). Both the Fourier and wavelet methods can be used to look at different spatial scales.

13.2.7 Gradient Statistics

The S1 score has been in historical use for verification of forecasts, particularly for variables such as pressure and geopotential height. This score compares differences between adjacent grid points in the forecast and observed fields. When the adjacent points in both forecast and observed fields exhibit the same differences, the S1 score will be the perfect value of 0. Larger differences will result in a larger score.

Differences are computed in both of the horizontal grid directions and is not a true mathematical gradient. Because the S1 score focuses on differences only, any bias in the forecast will not be measured. Further, the score depends on the domain and spacing of the grid, so can only be compared on forecasts with identical grids.

As described in *Ebert-Uphoff et al., 2024* (page 494), statistics based on the magnitude of the forecast and observed gradients are also provided. Similiar to the S1 score, the root-mean-squared error of the magnitude of the gradients and their divergence quantify the similarity in the texture of the fields, with 0 being a perfect score. These gradient-based statistics assess the difference in smoothness between the two fields but not the accuracy of the forecast.

13.2.8 Distance Maps

The following methods can all be computed efficiently by utilizing fast algorithms developed for calculating distance maps. A distance map results from calculating the shortest distance from every grid point, **s=(x,y)**, in the domain, **D**, to the nearest one-valued grid point. In each of the following, it is understood that they are calculated between event areas **A**, from one field and observation event areas **B** from another. If the measure is applied to a feature within a field, then the distance map is still calculated over the entire original domain. Some of the distance map statistics are computed over the entire distance map, while others use only parts of it.

Because these methods rely on the distance map, it is helpful to understand precisely what such maps do. Figure 13.1 demonstrates the path of the shortest distance to the nearest event point in the event area A marked by the gray rectangle in the diagram. Note that the arrows all point to a grid point on the boundary of the event area A as it would be a longer distance to any point in its interior. Figure 13.2 demonstrates the shortest distances from every grid point inside a second event area marked by the gray circle labeled B to the same event area A as in Figure 13.1. Note that all of the distances are to points on a small subsection (indicated by the yellow stretch) of the subset A.

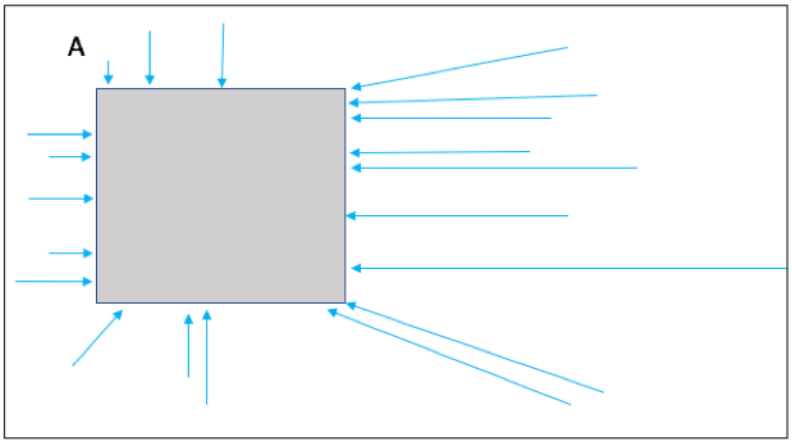

Figure 13.1: The above diagram depicts how a distance map is formed. From every grid point in the domain (depicted by the larger rectangle), the shortest distance from that grid to the nearest non-zero grid point (event; depicted by the gray rectangle labeled as A) is calculated (a sample of grid points with arrows indicate the path of the shortest distance with the length of the arrow equal to this distance. In a distance map, the value at each grid point is this distance. For example, grid points within the rectangle A will all have value zero in the distance map.

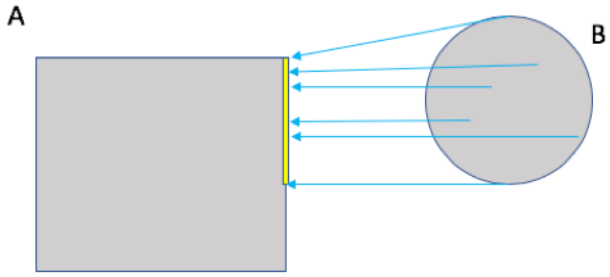

Figure 13.2: Diagram depicting the shortest distances from one event area to another. The yellow bar indicates the part of the event area A to where all of the shortest distances from B are calculated. That is, the shortest distances from every point inside the set B to the set A all point to a point along the yellow bar.

While Figure 13.1 and Figure 13.2 are helpful in illustrating the idea of a distance map, Figure 13.3 shows an actual distance map calculated for binary fields consisting of circular event areas, where one field has two circular event areas labeled A, and the second has one circular event area labeled B. Notice that the values of the distance map inside the event areas are all zero (dark blue) and the distances grow larger in the pattern of concentric circles around these event areas as grid cells move further away. Finally, Figure 13.4 depicts special situations from which the distance map measures to be discussed are calculated. In particular, the top left panel shows the absolute difference between the two distance maps presented in the bottom row of Figure 13.3. The top right panel shows the portion of the distance map for A that falls within the event area of B, and the bottom left depicts the portion of the distance map for B that falls within the event area A. That is, the first shows the shortest distances from every grid point in the set B to the nearest grid point in the event area A, and the latter shows the shortest distance from every grid point in A to the nearest grid point in B.

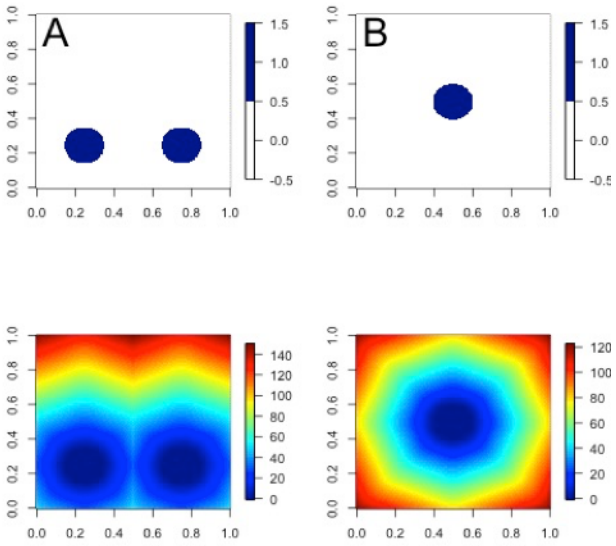

Figure 13.3: Binary fields (top) with event areas A (consisting of two circular event areas) and a second field with event area B (single circular area) with their respective distance maps (bottom).

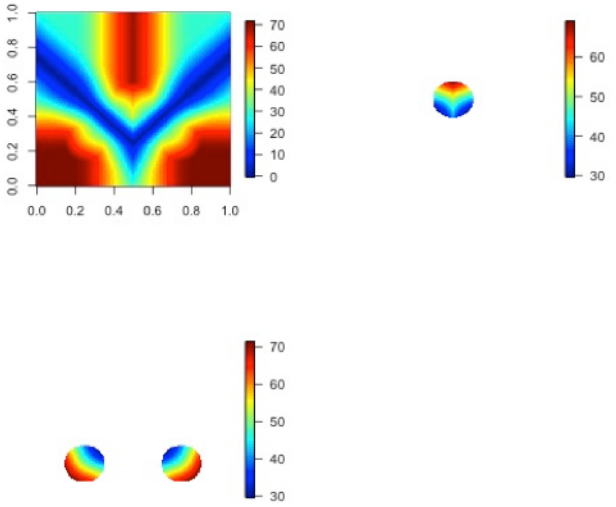

Figure 13.4: The absolute difference between the distance maps in the bottom row of Figure 13.3 (top left), the shortest distances from every grid point in B to the nearest grid point in A (top right), and the shortest distances from every grid point in A to the nearest grid points in B (bottom left). The latter two do not have axes in order to emphasize that the distances are now only considered from within the respective event sets. The top right graphic is the distance map of A conditioned on the presence of an event from B, and that in the bottom left is the distance map of B conditioned on the presence of an event from A.

The statistics derived from these distance maps are described in Appendix C, Section 35.7. To make fair comparisons, any grid point containing bad data in either the forecast or observation field is set to bad data in both fields. For each combination of input field and categorical threshold requested in the configuration file, Grid-Stat applies that threshold to define events in the forecast and observation fields and computes distance maps for those binary fields. Statistics for all requested masking regions are derived from those distance maps. Note that the distance maps are computed only once over the full verification domain, not separately for each masking region. Events occurring outside of a masking region can affect the distance map values inside that masking region and, therefore, can also affect the distance maps statistics for that region.

13.2.9 β and G_β

See Section 35.7.5 for the G and G_β equations.

G_β provides a summary measure of forecast quality for each user-defined threshold chosen. It falls into a range from zero to one where one is a perfect forecast and zero is considered to be a very poor forecast as determined by the user through the value of β. Values of G_β closer to one represent better forecasts and worse forecasts as it decreases toward zero. Although a particular value cannot be universally compared against any forecast, when applied with the same choice of β for the same variable and on the same domain, it is highly effective at ranking such forecasts.

G_β is sensitive to the choice of β, which depends on the (i) specific domain, (ii) variable, and (iii) user's needs. Smaller values make G_β more stringent and larger values make it more lenient. Figure 13.5 shows an example of applying G_β over a range of β values to a precipitation verification set where the binary fields are created by applying a threshold of $2.1mmh^{-1}$. Color choice and human bias can make it difficult to determine the quality of the forecast for a human observer looking at the raw images in the top row of the figure (*Ahijevych et al., 2009* (page 491)). The bottom left panel of the figure displays the differences in their binary fields, which highlights that the forecast captured the overall shapes of the observed rain areas but suffers from a spatial displacement error (perhaps really a timing error).

Whether or not the forecast from Figure 13.5 is "good" or not depends on the specific user. Is it sufficient that the forecast came as close as it did to the observation field? If the answer is yes for the user, then a higher choice of β, such as $N/2$, with N equal to the number of points in the domain, will correctly inform this user that it is a "good" forecast as it will lead to a G_β value near one. If the user requires the forecast to be much better aligned spatially with the observation field, then a lower choice, perhaps $\beta = N$, will correctly inform that the forecast suffers from spatial displacement errors that are too large for this user to be pleased. If the goal is to rank a series of ensemble forecasts, for example, then a choice of β that falls in the steep part of the curve shown in the lower right panel of the figure should be preferred, say somewhere between $\beta = N$ and $\beta = N^2/2$. Such a choice will ensure that each member is differentiated by the measure.

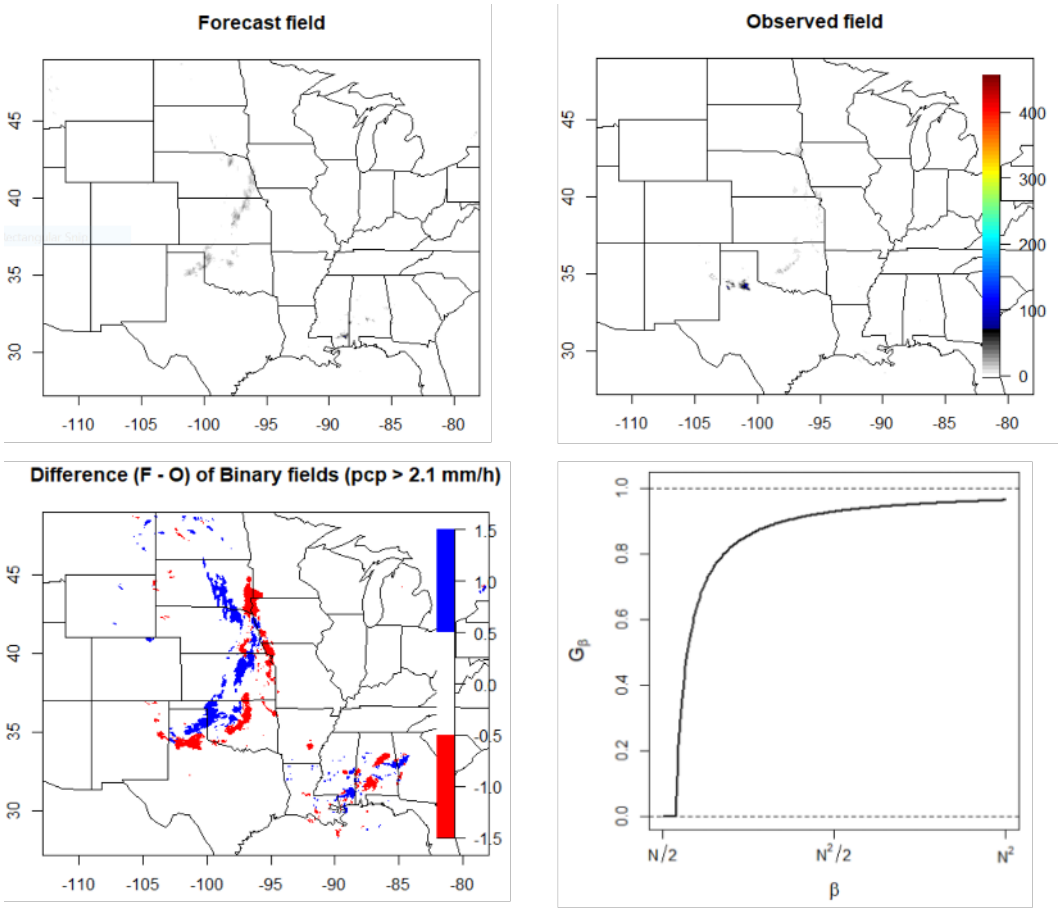

Figure 13.5: Top left is an example of an accumulated precipitation (mm/h) forecast with the corresponding observed field on the top right. Bottom left shows the difference in binary fields, where the binary fields are created by setting all values in the original fields that fall above $2.1mmh^{-1}$ to one and the rest to zero. Bottom right shows the results for G_β calculated on the binary fields using the threshold of $2.1mmh^{-1}$ over a range of choices for β.

In some cases, a user may be interested in a much higher threshold than $2.1mmh^{-1}$ of the above example. *Gilleland, 2021 (Fig. 4)* (page 495), for example, shows this same forecast using a threshold of $40mmh^{-1}$. Only a small area in Mississippi has such extreme rain predicted at this valid time; yet none was observed. Small spatial areas of extreme rain in the observed field, however, did occur in a location far away from Mississippi that was not predicted. Generally, for this type of verification, the Hausdorff metric is a good choice of measure. However, a small choice of β will provide similar results as the Hausdorff distance (*Gilleland, 2021* (page 495)). The user should think about the average size of storm areas and multiply this value by the displacement distance they are comfortable with in order to get a good initial choice for β, and may have to increase or decrease its value by trial-and-error using one or two example cases from their verification set.

Since G_β is so sensitive to the choice of β, which is defined relative to the number of points in the verification domain, G_β is only computed for the full verification domain. G_β is reported as a bad data value for any masking region subsets of the full verification domain.

13.3 Practical Information

This section contains information about configuring and running the Grid-Stat tool. The Grid-Stat tool verifies gridded model data using gridded observations. The input gridded model and observation datasets must be in one of the MET supported file formats. The requirement of having all gridded fields using the same grid specification was removed in METv5.1. There is a regrid option in the configuration file that allows the user to define the grid upon which the scores will be computed. The gridded observation data may be a gridded analysis based on observations such as Stage II or Stage IV data for verifying accumulated precipitation, or a model analysis field may be used.

The Grid-Stat tool provides the capability of verifying one or more model variables/levels using multiple thresholds for each model variable/level. The Grid-Stat tool performs no interpolation when the input model, observation, and climatology datasets must be on a common grid. MET will interpolate these files to a common grid if one is specified. The interpolation parameters may be used to perform a smoothing operation on the forecast field prior to verifying it to investigate how the scale of the forecast affects the verification statistics. The Grid-Stat tool computes a number of continuous statistics for the forecast minus observation differences, discrete statistics once the data have been thresholded, or statistics for probabilistic forecasts. All types of statistics can incorporate a climatological reference.

13.3.1 grid_stat Usage

The usage statement for the Grid-Stat tool is listed below:

```
Usage: grid_stat
       fcst_file
       obs_file
       config_file
       [-ugrid_config config_file]
       [-outdir path]
       [-log file]
       [-v level]
       [-compress level]
```

grid_stat has three required arguments and accepts several optional ones.

13.3.1.1 Required Arguments for grid_stat

1. The **fcst_file** argument indicates the gridded file containing the model data to be verified.
2. The **obs_file** argument indicates the gridded file containing the gridded observations to be used for the verification of the model.
3. The **config_file** argument indicates the name of the configuration file to be used. The contents of the configuration file are discussed below.

13.3.1.2 Optional Arguments for grid_stat

4. The **-ugrid_config** option provides a way for a user to provide a separate config file with metadata about their UGRID.
5. The **-outdir path** indicates the directory where output files should be written.
6. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
7. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.
8. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the grid_stat calling sequence is listed below:

Example 1:

```
grid_stat sample_fcst.grb \
sample_obs.grb \
GridStatConfig
```

In Example 1, the Grid-Stat tool will verify the model data in the sample_fcst.grb GRIB file using the observations in the sample_obs.grb GRIB file applying the configuration options specified in the **GridStatConfig** file.

A second example of the grid_stat calling sequence is listed below:

Example 2:

```
grid_stat sample_fcst.nc
sample_obs.nc
GridStatConfig
```

In the second example, the Grid-Stat tool will verify the model data in the sample_fcst.nc NetCDF output of pcp_combine, using the observations in the sample_obs.nc NetCDF output of pcp_combine, and applying the configuration options specified in the **GridStatConfig** file. Because the model and observation files contain only a single field of accumulated precipitation, the **GridStatConfig** file should be configured to specify that only accumulated precipitation be verified.

13.3.2 grid_stat Configuration File

The default configuration file for the Grid-Stat tool, named **GridStatConfig_default**, can be found in the installed *share/met/config* directory. Other versions of the configuration file are included in *scripts/config*. We recommend that users make a copy of the default (or other) configuration file prior to modifying it. The contents are described in more detail below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
model            = "FCST";
desc             = "NA";
obtype           = "ANALYS";
fcst             = { ... }
obs              = { ... }
regrid           = { ... }
climo_mean       = { ... }
climo_stdev      = { ... }
climo_cdf        = { ... }
mask             = { grid = [ "FULL" ]; poly = []; }
ci_alpha         = [ 0.05 ];
boot             = { interval = PCTILE; rep_prop = 1.0; n_rep = 1000;
                     rng = "mt19937"; seed = ""; }
interp           = { field = BOTH; vld_thresh = 1.0; shape = SQUARE;
                     type = [ { method = NEAREST; width = 1; } ]; }
censor_thresh    = [];
censor_val       = [];
mpr_column       = [];
mpr_thresh       = [];
eclv_points      = 0.05;
hss_ec_value     = NA;
rank_corr_flag   = TRUE;
gradient         = { dx = [ 1 ]; dy = [ 1 ]; }
grid_weight_flag = NONE;
tmp_dir          = "/tmp";
output_prefix    = "";
version          = "VN.N";
```

The configuration options listed above are common to multiple MET tools and are described in Section 5.

nbrhd (page 262)

```
nbrhd = {
   field      = BOTH;
   vld_thresh = 1.0;
   shape      = SQUARE;
```

(continues on next page)

(continued from previous page)

```
   width      = [ 1 ];
   cov_thresh = [ >=0.5 ];
 }
```

The **nbrhd** dictionary contains a list of values to be used in defining the neighborhood to be used when computing neighborhood verification statistics. The neighborhood **shape** is a **SQUARE** or **CIRCLE** centered on the current point, and the **width** value specifies the width of the square or diameter of the circle as an odd integer.

The **field** entry is set to **BOTH, FCST, OBS**, or **NONE** to indicate the fields to which the fractional coverage derivation logic should be applied. This should always be set to **BOTH** unless you have already computed the fractional coverage field(s) with numbers between 0 and 1 outside of MET.

The **vld_thresh** entry contains a number between 0 and 1. When performing neighborhood verification over some neighborhood of points the ratio of the number of valid data points to the total number of points in the neighborhood is computed. If that ratio is greater than this threshold, that value is included in the neighborhood verification. Setting this threshold to 1, which is the default, requires that the entire neighborhood must contain valid data. This variable will typically come into play only along the boundaries of the verification region chosen.

The **cov_thresh** entry contains a comma separated list of thresholds to be applied to the neighborhood coverage field. The coverage is the proportion of forecast points in the neighborhood that exceed the forecast threshold. For example, if 10 of the 25 forecast grid points contain values larger than a threshold of 2, then the coverage is 10/25 = 0.4. If the coverage threshold is set to 0.5, then this neighborhood is considered to be a "No" forecast.

---

fourier (page 263)

```
fourier = {
   wave_1d_beg = [ 0, 4, 10 ];
   wave_1d_end = [ 3, 9, 20 ];
}
```

The **fourier** entry is a dictionary which specifies the application of the Fourier decomposition method. It consists of two arrays of the same length which define the beginning and ending wave numbers to be included. If the arrays have length zero, no Fourier decomposition is applied. For each array entry, the requested Fourier decomposition is applied to the forecast and observation fields. The beginning and ending wave numbers are indicated in the MET ASCII output files by the INTERP_MTHD column (e.g. WV1_0-3 for waves 0 to 3 or WV1_10 for only wave 10). This 1-dimensional Fourier decomposition is computed along the Y-dimension only (i.e. the columns of data). It is applied to the forecast and observation fields as well as the climatological mean field, if specified. It is only defined when each grid point contains valid data. If any input field contains missing data, no Fourier decomposition is computed.

The available wave numbers start at 0 (the mean across each row of data) and end at (Nx+1)/2 (the finest level of detail), where Nx is the X-dimension of the verification grid:

- The **wave_1d_beg** entry is an array of integers specifying the first wave number to be included.
- The **wave_1d_end** entry is an array of integers specifying the last wave number to be included.

distance_map (page 264)

```
distance_map = {
   baddeley_p        = 2;
   baddeley_max_dist = NA;
   fom_alpha         = 0.1;
   zhu_weight        = 0.5;
   beta_value(n)     = n * n / 2.0;
}
```

The **distance_map** entry is a dictionary containing options related to the distance map statistics in the **DMAP** output line type. The **baddeley_p** entry is an integer specifying the exponent used in the Lp-norm when computing the Baddeley Δ metric. The **baddeley_max_dist** entry is a floating point number specifying the maximum allowable distance for each distance map. Any distances larger than this number will be reset to this constant. A value of **NA** indicates that no maximum distance value should be used. The **fom_alpha** entry is a floating point number specifying the scaling constant to be used when computing Pratt's Figure of Merit. The **zhu_weight** specifies a value between 0 and 1 to define the importance of the RMSE of the binary fields (i.e. amount of overlap) versus the mean-error distance (MED). The default value of 0.5 gives equal weighting. This configuration option may be set separately in each **obs.field** entry. The **beta_value** entry is defined as a function of n, where n is the total number of grid points in the full verification domain containing valid data in both the forecast and observation fields. The resulting beta_value is used to compute the G_β statistic. The default function, $N^2/2$, is recommended in *Gilleland, 2021* (page 495) but can be modified as needed.

```
output_flag = {
   fho    = BOTH;
   ctc    = BOTH;
   cts    = BOTH;
   mctc   = BOTH;
   mcts   = BOTH;
   cnt    = BOTH;
   sl1l2  = BOTH;
   sal1l2 = NONE;
   vl1l2  = BOTH;
   val1l2 = NONE;
   vcnt   = BOTH;
   pct    = BOTH;
   pstd   = BOTH;
   pjc    = BOTH;
   prc    = BOTH;
   eclv   = BOTH;
   nbrctc = BOTH;
   nbrcts = BOTH;
   nbrcnt = BOTH;
```

(continues on next page)

(continued from previous page)

```
   grad   = BOTH;
   dmap   = BOTH;
   seeps  = NONE;
}
```

The **output_flag** array controls the type of output that the Grid-Stat tool generates. Each flag corresponds to an output line type in the STAT file. Setting the flag to NONE indicates that the line type should not be generated. Setting the flag to STAT indicates that the line type should be written to the STAT file only. Setting the flag to BOTH indicates that the line type should be written to the STAT file as well as a separate ASCII file where the data are grouped by line type. These output flags correspond to the following types of output line types:

1. **FHO** for Forecast, Hit, Observation Rates
2. **CTC** for Contingency Table Counts
3. **CTS** for Contingency Table Statistics
4. **MCTC** for Multi-Category Contingency Table Counts
5. **MCTS** for Multi-Category Contingency Table Statistics
6. **CNT** for Continuous Statistics
7. **SL1L2** for Scalar L1L2 Partial Sums
8. **SAL1L2** for Scalar Anomaly L1L2 Partial Sums when climatological data is supplied
9. **VL1L2** for Vector L1L2 Partial Sums
10. **VAL1L2** for Vector Anomaly L1L2 Partial Sums when climatological data is supplied
11. **VCNT** for Vector Continuous Statistics
12. **PCT** for Contingency Table Counts for Probabilistic forecasts
13. **PSTD** for Contingency Table Statistics for Probabilistic forecasts
14. **PJC** for Joint and Conditional factorization for Probabilistic forecasts
15. **PRC** for Receiver Operating Characteristic for Probabilistic forecasts
16. **ECLV** for Cost/Loss Ratio Relative Value
17. **NBRCTC** for Neighborhood Contingency Table Counts
18. **NBRCTS** for Neighborhood Contingency Table Statistics
19. **NBRCNT** for Neighborhood Continuous Statistics
20. **GRAD** for Gradient Statistics
21. **DMAP** for Distance Map Statistics
22. **SEEPS** for SEEPS (Stable Equitable Error in Probability Space) score. It's described in Table 11.20. The SEEPS score of matched pair data is saved into the NetCDF.

Note that the first two line types are easily derived from one another. The user is free to choose which measure is most desired. The output line types are described in more detail in Section 13.3.3.

The SEEPS climo file is not distributed with MET tools because of the file size. It should be configured by using the configuration file (seeps_grid_climo_name). It can be overridden by the environment variable, MET_SEEPS_GRID_CLIMO_NAME.

```
nc_pairs_flag = {
   latlon       = TRUE;
   raw          = TRUE;
   diff         = TRUE;
   climo        = TRUE;
   climo_cdp    = TRUE;
   weight       = FALSE;
   nbrhd        = FALSE;
   gradient     = FALSE;
   distance_map = FALSE;
   apply_mask   = TRUE;
}
```

The **nc_pairs_flag** entry may either be set to a boolean value or a dictionary specifying which fields should be written. Setting it to TRUE indicates the output NetCDF matched pairs file should be created with all available output fields, while setting all to FALSE disables its creation. This is done regardless of if **output_flag** dictionary indicates any statistics should be computed. The **latlon, raw**, and **diff** entries control the creation of output variables for the latitude and longitude, the forecast and observed fields after they have been modified by any user-defined regridding, censoring, and conversion, and the forecast minus observation difference fields, respectively. The **climo, weight**, and **nbrhd** entries control the creation of output variables for the climatological mean and standard deviation fields, the grid area weights applied, and the fractional coverage fields computed for neighborhood verification methods. Setting these entries to TRUE indicates that they should be written, while setting them to FALSE disables their creation.

Setting the **climo_cdp** entry to TRUE enables the creation of an output variable for each climatological distribution percentile (FCDP or OCDP) threshold requested in the configuration file. Note that enabling **nbrhd** output may lead to very large output files. The **gradient** entry controls the creation of output variables for the FCST and OBS gradients in the grid-x and grid-y directions. The **distance_map** entry controls the creation of output variables for the FCST and OBS distance maps for each categorical threshold. The **apply_mask** entry controls whether to create the FCST, OBS, and DIFF output variables for all defined masking regions. Setting this to TRUE will create the FCST, OBS, and DIFF output variables for all defined masking regions. Setting this to FALSE will create the FCST, OBS, and DIFF output variables for only the FULL verification domain.

```
nc_pairs_var_name = "";
```

The **nc_pairs_var_name** entry specifies a string for each verification task. This string is parsed from each **obs.field** dictionary entry and is used to construct variable names for the NetCDF matched pairs output file. The default value of an empty string indicates that the **name** and **level** strings of the input data should be

used. If the input data **level** string changes for each run of Grid-Stat, using this option to define a constant string may make downstream processing more convenient.

```
nc_pairs_var_suffix = "";
```

The **nc_pairs_var_suffix** entry is similar to the **nc_pairs_var_name** entry. It is also parsed from each **obs.field** dictionary entry. However, it defines a suffix to be appended to the output variable name. This enables the output variable names to be made unique. For example, when verifying height for multiple level types but all with the same level value, use this option to customize the output variable names. This option was previously named **nc_pairs_var_str** which is now deprecated.

13.3.3 grid_stat Output

grid_stat produces output in STAT and, optionally, ASCII and NetCDF formats. The ASCII output duplicates the STAT output but has the data organized by line type. The output files are written to the default output directory or the directory specified by the -outdir command line option.

The output STAT file is named using the following naming convention:

grid_stat_PREFIX_HHMMSSL_YYYYMMDD_HHMMSSV.stat where PREFIX indicates the user-defined output prefix, HHMMSSL indicates the forecast lead time and YYYYMMDD_HHMMSSV indicates the forecast valid time.

The output ASCII files are named similarly:

grid_stat_PREFIX_HHMMSSL_YYYYMMDD_HHMMSSV_TYPE.txt where TYPE is one of fho, ctc, cts, mctc, mcts, cnt, sl1l2, vl1l2, vcnt, pct, pstd, pjc, prc, eclv, nbrctc, nbrcts, nbrcnt, dmap, or grad to indicate the line type it contains.

The format of the STAT and ASCII output of the Grid-Stat tool are the same as the format of the STAT and ASCII output of the Point-Stat tool with the exception of the five additional line types. Please refer to the tables in Section 11.3.3 for a description of the common output STAT and optional ASCII file line types. The formats of the five additional line types for grid_stat are explained in the following tables.

Table 13.1: Header information for Grid-Stat STAT output

Column Number	Header Column Name	Description	Data Type
1	VERSION	Version number	String
2	MODEL	User-provided text string designating model name	String
3	DESC	User-provided text string describing the verification task	String
4	FCST_LEAD	Forecast lead time in HHMMSS format	Time String
5	FCST_VALID_BEG	Forecast valid start time in YYYYMMDD_HHMMSS format	Datetime String
6	FCST_VALID_END	Forecast valid end time in YYYYMMDD_HHMMSS format	Datetime String
7	OBS_LEAD	Observation lead time in HHMMSS format	Time String
8	OBS_VALID_BEG	Observation valid start time in YYYYMMDD_HHMMSS format	Datetime String
9	OBS_VALID_END	Observation valid end time in YYYYMMDD_HHMMSS format	Datetime String
10	FCST_VAR	Model variable	String
11	FCST_UNITS	Units for model variable	String
12	FCST_LEV	Selected vertical level for forecast	String
13	OBS_VAR	Observation variable	String
14	OBS_UNITS	Units for observation variable	String
15	OBS_LEV	Selected vertical level for observations	String
16	OBTYPE	User-provided text string designating the observation type	String
17	VX_MASK	Verifying masking region indicating the masking grid or polyline region applied	String
18	INTERP_MTHD	Interpolation method applied to forecast field	String
19	INTERP_PNTS	Number of points used by interpolation method	Integer
20	FCST_THRESH	The threshold applied to the forecast	Threshold String
21	OBS_THRESH	The threshold applied to the observations	Threshold String
22	COV_THRESH	Proportion of observations in specified neighborhood which must exceed obs_thresh	Threshold String
23	ALPHA	Error percent value used in confidence intervals	Double
24	LINE_TYPE	Various line type options, refer to Section 11.3.3 and the tables below.	String

Table 13.2: Format information for NBRCTC (Neighborhood Contingency Table Counts) output line type

Column Number	NBRCTC Column Name	Description	Data Type
24	NBRCTC	Neighborhood Contingency Table Counts line type	String
25	TOTAL	Total number of matched pairs	Integer
26	FY_OY	Sum of weights for hits (forecast yes and observation yes)	Double
27	FY_ON	Sum of weights for false alarms (forecast yes and observation no)	Double
28	FN_OY	Sum of weights for misses (forecast no and observation yes)	Double
29	FN_ON	Sum of weights for correct negatives (forecast no and observation no)	Double

Table 13.3: Format information for NBRCTS (Neighborhood Contingency Table Statistics) output line type

Column Number	NBRCTS Column Name	Description	Data Type
24	NBRCTS	Neighborhood Contingency Table Statistics line type	String
25	TOTAL	Total number of matched pairs	Integer
26-30	BASER, BASER_NCL, BASER_NCU, BASER_BCL, BASER_BCU	Base rate including normal and bootstrap upper and lower confidence limits	Double
31-35	FMEAN, FMEAN_NCL, FMEAN_NCU, FMEAN_BCL, FMEAN_BCU	Forecast mean including normal and bootstrap upper and lower confidence limits	Double
36-40	ACC, ACC_NCL, ACC_NCU, ACC_BCL, ACC_BCU	Accuracy including normal and bootstrap upper and lower confidence limits	Double
41-43	FBIAS, FBIAS_BCL, FBIAS_BCU	Frequency Bias including bootstrap upper and lower confidence limits	Double
44-48	PODY, PODY_NCL, PODY_NCU, PODY_BCL, PODY_BCU	Probability of detecting yes including normal and bootstrap upper and lower confidence limits	Double
49-53	PODN, PODN_NCL, PODN_NCU, PODN_BCL, PODN_BCU	Probability of detecting no including normal and bootstrap upper and lower confidence limits	Double
54-58	POFD, POFD_NCL, POFD_NCU, POFD_BCL, POFD_BCU	Probability of false detection including normal and bootstrap upper and lower confidence limits	Double
59-63	FAR, FAR_NCL, FAR_NCU, FAR_BCL, FAR_BCU	False alarm ratio including normal and bootstrap upper and lower confidence limits	Double
64-68	CSI, CSI_NCL, CSI_NCU, CSI_BCL, CSI_BCU	Critical Success Index including normal and bootstrap upper and lower confidence limits	Double
69-71	GSS, GSS_BCL, GSS_BCU	Gilbert Skill Score including bootstrap upper and lower confidence limits	Double
72-76	HK, HK_NCL, HK_NCU, HK_BCL, HK_BCU	Hanssen-Kuipers Discriminant including normal and bootstrap upper and lower confidence limits	Double
77-79	HSS, HSS_BCL, HSS_BCU	Heidke Skill Score including bootstrap upper and lower confidence limits	Double
80-84	ODDS, ODDS_NCL, ODDS_NCU, ODDS_BCL, ODDS_BCU	Odds Ratio including normal and bootstrap upper and lower confidence limits	Double
85-89	LODDS, LODDS_NCL, LODDS_NCU, LODDS_BCL, LODDS_BCU	Logarithm of the Odds Ratio including normal and bootstrap upper and lower confidence limits	Double
90-94	ORSS, ORSS _NCL, ORSS _NCU, ORSS _BCL, ORSS _BCU	Odds Ratio Skill Score including normal and bootstrap upper and lower confidence limits	Double
95-99	EDS, EDS _NCL, EDS _NCU, EDS _BCL, EDS _BCU	Extreme Dependency Score including normal and bootstrap upper and lower confidence limits	Double
100-104	SEDS, SEDS _NCL, SEDS _NCU, SEDS _BCL SEDS _BCU	Symmetric Extreme Dependency Score including normal and bootstrap upper and lower confidence limits	Double
105-109	EDI, EDI _NCL, EDI _NCU, EDI _BCL, EDI _BCU	Extreme Dependency Index including normal and bootstrap upper and lower confidence limits	Double
110-114	SEDI, SEDI _NCL, SEDI _NCU, SEDI _BCL SEDI _BCU	Symmetric Extremal Dependency Index including normal and bootstrap upper and lower confidence	Double

Table 13.4: Format information for NBRCNT(Neighborhood Continuous Statistics) output line type

Column Number	NBRCNT Column Name	Description	Data Type
24	NBRCNT	Neighborhood Continuous statistics line type	String
25	TOTAL	Total number of matched pairs	Integer
26-28	FBS, FBS_BCL, FBS_BCU	Fractions Brier Score including bootstrap upper and lower confidence limits	Double
29-31	FSS, FSS_BCL, FSS_BCU	Fractions Skill Score including bootstrap upper and lower confidence limits	Double
32-34	AFSS, AFSS_BCL, AFSS_BCU	Asymptotic Fractions Skill Score including bootstrap upper and lower confidence limits	Double
35-37	UFSS, UFSS_BCL, UFSS_BCU	Uniform Fractions Skill Score including bootstrap upper and lower confidence limits	Double
38-40	F_RATE, F_RATE _BCL, F_RATE _BCU	Forecast event frequency including bootstrap upper and lower confidence limits	Double
41-43	O_RATE, O _RATE _BCL, O _RATE _BCU	Observed event frequency including bootstrap upper and lower confidence limits	Double

Table 13.5: Format information for GRAD (Gradient Statistics) output line type

Column Number	GRAD Column Name	Description	Data Type
24	GRAD	Gradient Statistics line type	String
25	TOTAL	Total number of matched pairs	Integer
26	FGBAR	Mean of absolute value of forecast gradients	Double
27	OGBAR	Mean of absolute value of observed gradients	Double
28	MGBAR	Mean of maximum of absolute values of forecast and observed gradients	Double
29	EGBAR	Mean of absolute value of forecast minus observed gradients	Double
30	S1	S1 score	Double
31	S1_OG	S1 score with respect to observed gradient	Double
32	FGOG_RATIO	Ratio of forecast and observed gradients	Double
33	DX	Gradient size in the X-direction	Integer
34	DY	Gradient size in the Y-direction	Integer
35	FGMAG	Magnitude of the forecast gradient when the X and Y-directions are interpreted as a vector	Double
36	OGMAG	Magnitude of the observed gradient when the X and Y-directions are intrepreted as a vector	Double
37	MAG_RMSE	Root mean squared difference of the forecast gradient magnitude minus the observed gradient magnitude	Double
38	LAPLACE_RMSE	Root mean squared difference of the sum of the forecast X and Y-gradients minus the sum of the observed X and Y-gradients	Double

Table 13.6: Format information for DMAP (Distance Map) output line type

Column Number	DMAP Column Name	Description	Data Type
24	DMAP	Distance Map line type	String
25	TOTAL	Total number of matched pairs	Integer
26	FY	Number of forecast events	Integer
27	OY	Number of observation events	Integer
28	FBIAS	Frequency Bias	Double
29	BADDELEY	Baddeley's Δ Metric	Double
30	HAUSDORFF	Hausdorff Distance	Double
31	MED_FO	Mean-error Distance from observation to forecast	Double
32	MED_OF	Mean-error Distance from forecast to observation	Double
33	MED_MIN	Minimum of MED_FO and MED_OF	Double
34	MED_MAX	Maximum of MED_FO and MED_OF	Double
35	MED_MEAN	Mean of MED_FO and MED_OF	Double
36	FOM_FO	Pratt's Figure of Merit from observation to forecast	Double
37	FOM_OF	Pratt's Figure of Merit from forecast to observation	Double
38	FOM_MIN	Minimum of FOM_FO and FOM_OF	Double
39	FOM_MAX	Maximum of FOM_FO and FOM_OF	Double
40	FOM_MEAN	Mean of FOM_FO and FOM_OF	Double
41	ZHU_FO	Zhu's Measure from observation to forecast	Double
42	ZHU_OF	Zhu's Measure from forecast to observation	Double
43	ZHU_MIN	Minimum of ZHU_FO and ZHU_OF	Double
44	ZHU_MAX	Maximum of ZHU_FO and ZHU_OF	Double
45	ZHU_MEAN	Mean of ZHU_FO and ZHU_OF	Double
46	G	G distance measure	Double
47	GBETA	G_β distance measure	Double
48	BETA_VALUE	Beta value used to compute G_β	Double

If requested using the **nc_pairs_flag** dictionary in the configuration file, a NetCDF file containing the matched pair and forecast minus observation difference fields for each combination of variable type/level and masking region applied will be generated. The contents of this file are determined by the contents of the nc_pairs_flag dictionary. The output NetCDF file is named similarly to the other output files: **grid_stat_PREFIX_ HHMMSSL_YYYYMMDD_HHMMSSV_pairs.nc**. Commonly available NetCDF utilities such as ncdump or ncview may be used to view the contents of the output file.

The output NetCDF file contains the dimensions and variables shown in Table 13.7 and Table 13.8.

Table 13.7: Dimensions defined in NetCDF matched pair output

NetCDF Dimension	Description
Lat	Dimension of the latitude (i.e. Number of grid points in the North-South direction)
Lon	Dimension of the longitude (i.e. Number of grid points in the East-West direction)

Table 13.8: A selection of variables that can appear in the NetCDF matched pair output

NetCDF Variable	Dimension	Description	Data Type
FCST_VAR_LVL_MASK _INTERP_MTHD _INTERP_PNTS	lat, lon	For each model variable (VAR), vertical level (LVL), masking region (MASK), and, if applicable, smoothing operation (INTERP_MTHD and INTERP_PNTS), the forecast value is listed for each point in the mask.	Float
OBS_VAR_LVL_MASK DIFF_FCSTVAR	lat, lon	For each model variable (VAR), vertical level (LVL), and masking region (MASK), the observation value is listed for each point in the mask.	Float
DIFF_FCSTVAR _FCSTLVL _OBSVAR _OBSLVL_MASK _INTERP_MTHD _INTERP_PNTS	lat, lon	For each model variable (VAR), vertical level (LVL), masking region (MASK), and, if applicable, smoothing operation (INTERP_MTHD and INTERP_PNTS), the difference (forecast - observation) is computed for each point in the mask.	Float
FCST_XGRAD_DX FCST_YGRAD_DX OBS_XGRAD_DY OBS_YGRAD_DY	lat, lon	List the gradient of the forecast and observation fields computed in the grid-x and grid-y directions where DX and DY indicate the gradient direction and size.	Float

The STAT output files described for grid_stat may be used as inputs to the Stat-Analysis tool. For more information on using the Stat-Analysis tool to create stratifications and aggregations of the STAT files produced by grid_stat, please see Section 17.

Chapter 14

Ensemble-Stat Tool

14.1 Introduction

The Ensemble-Stat tool verifies deterministic ensemble members against gridded and/or point observations. It computes ensemble statistics such as rank histograms, probability integral transform histograms, spread/skill variance, relative position and continuous ranked probability score. Climatological mean and standard deviation data may also be provided, and is used as a reference forecast in several of the output statistics. Finally, observation error perturbations can be included prior to calculation of statistics. Details about and equations for the statistics produced for ensembles are given in Appendix C, Section 35.5.

Note: Earlier versions of the Ensemble-Stat tool supported both ensemble product generation and ensemble verification. However, the ensemble product generation logic has moved to the *Gen-Ens-Prod Tool* (page 195), which replaces and extends that functionality. Ensemble product generation was removed from Ensemble-Stat in version 11.0.0.

14.2 Scientific and Statistical Aspects

14.2.1 HiRA Framework

The HiRA framework described in Section 11.2.2 is also supported in the Ensemble-Stat tool. That support is provided as an interpolation option via the **interp** dictionary. The interpolation dictionary defines how gridded model data is matched to each observation value. Most interpolation methods, such as **UW_MEAN** for the unweighted mean or **BILIN** for bilinear, compute a single value for each ensemble member. When the High Resolution Assessment (HiRA) interpolation method is chosen, as shown below, all of the nearby neighborhood points surrounding each observation from each member are used. Therefore, processing an N-member ensemble using a HiRA neighborhood of size M produces ensemble output with size N*M. This approach fully leverages information from all nearby grid points to evaluate the ensemble quality.

```
interp = {
  field      = BOTH;
```

(continues on next page)

(continued from previous page)

```
  vld_thresh = 1.0;
  shape      = SQUARE;

  type = [
     {
        method = HIRA;
        width  = 2;
        shape  = SQUARE;
     }
  ];
}
```

In this example, all four grid points of the 2x2 square surrounding each observation point are used to define the ensemble. Therefore, an N-member ensemble is evaluated as an ensemble of size Nx4.

14.2.2 Ensemble Statistics

Rank histograms and probability integral transform (PIT) histograms are used to determine if the distribution of ensemble values is the same as the distribution of observed values for any forecast field (*Hamill, 2001* (page 496)). The rank histogram is a tally of the rank of the observed value when placed in order with each of the ensemble values from the same location. If the distributions are identical, then the rank of the observation will be uniformly distributed. In other words, it will fall among the ensemble members randomly in equal likelihood. The PIT histogram applies this same concept, but transforms the actual rank into a probability to facilitate ensembles of differing sizes or with missing members.

Often, the goal of ensemble forecasting is to reproduce the distribution of observations using a set of many forecasts. In other words, the ensemble members represent the set of all possible outcomes. When this is true, the spread of the ensemble is identical to the error in the mean forecast. Though this rarely occurs in practice, the spread / skill relationship is still typically assessed for ensemble forecasts (*Barker, 1991* (page 491); *Buizza,1997* (page 492)). MET calculates the spread and skill in user defined categories according to *Eckel et al. (2012)* (page 494).

The relative position (RELP) is a count of the number of times each ensemble member is closest to the observation. For stochastic or randomly derived ensembles, this statistic is meaningless. For specified ensemble members, however, it can assist users in determining if any ensemble member is performing consistently better or worse than the others.

The ranked probability score (RPS) is included in the Ranked Probability Score (RPS) line type. It is the mean of the Brier scores computed from ensemble probabilities derived for each probability category threshold (prob_cat_thresh) specified in the configuration file. The continuous ranked probability score (CRPS) is the average the distance between the forecast (ensemble) cumulative distribution function and the observation cumulative distribution function. It is an analog of the Brier score, but for continuous forecast and observation fields. The CRPS statistic is computed using two methods: assuming a normal distribution defined by the ensemble mean and spread (*Gneiting et al., 2004* (page 495)) and using the empirical ensemble distribution (*Hersbach, 2000* (page 496)). The CRPS statistic using the empirical ensemble distribution can be adjusted (bias corrected) by subtracting 1/(2*m) times the mean absolute difference of the ensemble members, where m is the ensemble size. This is reported as a separate statistic called CRPS_EMP_FAIR. The

empirical CRPS and its fair version are included in the Ensemble Continuous Statistics (ECNT) line type, along with other statistics quantifying the ensemble spread and ensemble mean skill.

The Ensemble-Stat tool can derive ensemble relative frequencies and verify them as probability forecasts all in the same run. Note however that these simple ensemble relative frequencies are not actually calibrated probability forecasts. If probabilistic line types are requested (output_flag), this logic is applied to each pair of fields listed in the forecast (fcst) and observation (obs) dictionaries of the configuration file. Each probability category threshold (prob_cat_thresh) listed for the forecast field is applied to the input ensemble members to derive a relative frequency forecast. The probability category threshold (prob_cat_thresh) parsed from the corresponding observation entry is applied to the (gridded or point) observations to determine whether or not the event actually occurred. The paired ensemble relative frequencies and observation events are used to populate an Nx2 probabilistic contingency table. The dimension of that table is determined by the probability PCT threshold (prob_pct_thresh) configuration file option parsed from the forecast dictionary. All probabilistic output types requested are derived from this Nx2 table and written to the ascii output files. Note that the FCST_VAR name header column is automatically reset as "PROB({FCST_VAR}{THRESH})" where {FCST_VAR} is the current field being evaluated and {THRESH} is the threshold that was applied.

Note that if no probability category thresholds (prob_cat_thresh) are defined, but climatological mean and standard deviation data is provided along with climatological bins, climatological distribution percentile thresholds are automatically derived and used to compute probabilistic outputs.

14.2.3 Climatology Data

The Ensemble-Stat output includes at least three statistics computed relative to external climatology data. The climatology is defined by mean and standard deviation fields, and typically both are required in the computation of ensemble skill score statistics. MET assumes that the climatology follows a normal distribution, defined by the mean and standard deviation at each point.

When computing the CRPS skill score for (*Gneiting et al., 2004* (page 495)) the reference CRPS statistic is computed using the climatological mean and standard deviation directly. When computing the CRPS skill score for (*Hersbach, 2000* (page 496)) the reference CRPS statistic is computed by selecting equal-area-spaced values from the assumed normal climatological distribution. The number of points selected is determined by the *cdf_bins* setting in the *climo_cdf* dictionary. The reference CRPS is computed empirically from this ensemble of climatology values. If the number bins is set to 1, the climatological CRPS is computed using only the climatological mean value. In this way, the empirical CRPSS may be computed relative to a single model rather than a climatological distribution.

The climatological distribution is also used for the RPSS. The forecast RPS statistic is computed from a probabilistic contingency table in which the probabilities are derived from the ensemble member values. In a simliar fashion, the climatogical probability for each observed value is derived from the climatological distribution. The area of the distribution to the left of the observed value is interpreted as the climatological probability. These climatological probabilities are also evaluated using a probabilistic contingency table from which the reference RPS score is computed. The skill scores are derived by comparing the forecast statistic to the reference climatology statistic.

The Ensemble-Stat tool also allows the computation of RPS and RPSS utilizing an ensemble forecast in probabilistic space. This unique ability is for users with ensemble data that is formulated relative to climatology; it does not require the use of any climatological datasets, instead relying on the assumption that each ensemble member has a climatologically equal chance of occurring. Each ensemble member's field should contain

values in the range [0, 1] or [0, 100]. However, when MET encounters a probability field with a range [0, 100], it will automatically rescale it to be [0, 1]. The sum of all ensemble member fields should equal 1 (if range is [0, 1]) or 100 (if range is [0, 100]). When calculating RPS, the ensemble member field values are cumulatively summed in the order that they are evaluated by Ensemble-Stat. Each of these sums is then used to calculate a squared probability error relative to observations and their appropriate thresolds. Note that it is expected each observation point or observation gridpoint will indicate exactly one climatological probability bin where the observation occurred. The cumulative sum and squared probability errors are also calculated for climatology using an even, constant probability bin width that is equal to 1 divided by the number of ensemble members (ex. three ensemble members are evaluated with three climatology probability bins of width 0.333). The accompanying skill score is derived by comparing the forecast RPS to the reference climatology RPS.

14.2.4 Ensemble Observation Error

In an attempt to ameliorate the effect of observation errors on the verification of forecasts, a random perturbation approach has been implemented. A great deal of user flexibility has been built in, but the methods detailed in *Candille and Talagrand (2008)* (page 493) can be replicated using the appropriate options. Additional variations of the ignorance score that include observational uncertainty recommended by *Ferro, 2017* (page 494) are also provided.

Observation error information can be defined directly in the Ensemble-Stat configuration file or through a more flexible observation error lookup table. The user selects a distribution for the observation error, along with parameters for that distribution. Rescaling and bias correction can also be specified prior to the perturbation. Random draws from the distribution can then be added to either, or both of the forecast and observed fields, including ensemble members. Details about the effects of the choices on verification statistics should be considered, with many details provided in the literature (*e.g.* *Candille and Talagrand, 2008* (page 493); *Saetra et al., 2004* (page 498); *Santos and Ghelli, 2012* (page 498)). Generally, perturbation makes verification statistics better when applied to ensemble members, and worse when applied to the observations themselves.

Normal and uniform are common choices for the observation error distribution. The uniform distribution provides the benefit of being bounded on both sides, thus preventing the perturbation from taking on extreme values. Normal is the most common choice for observation error. However, the user should realize that with the very large samples typical in NWP, some large outliers will almost certainly be introduced with the perturbation. For variables that are bounded below by 0, and that may have inconsistent observation errors (e.g. larger errors with larger measurements), a lognormal distribution may be selected. Wind speeds and precipitation measurements are the most common of this type of NWP variable. The lognormal error perturbation prevents measurements of 0 from being perturbed, and applies larger perturbations when measurements are larger. This is often the desired behavior in these cases, but this distribution can also lead to some outliers being introduced in the perturbation step.

Observation errors differ according to instrument, temporal and spatial representation, and variable type. Unfortunately, many observation errors have not been examined or documented in the literature. Those that have usually lack information regarding their distributions and approximate parameters. Instead, a range or typical value of observation error is often reported and these are often used as an estimate of the standard deviation of some distribution. Where possible, it is recommended to use the appropriate type and size of perturbation for the observation to prevent spurious results.

14.3 Practical Information

This section contains information about configuring and running the Ensemble-Stat tool. The Ensemble-Stat tool creates or verifies gridded model data. For verification, this tool can accept either gridded or point observations. If provided, the climatology data files must be gridded. The input gridded model, observation, and climatology datasets must be on the same grid prior to calculation of any statistics, and in one of the MET supported gridded file formats. If gridded files are not on the same grid, MET will do the regridding for you if you specify the desired output grid. The point observations must be formatted as the NetCDF output of the point reformatting tools described in Section 7.

14.3.1 ensemble_stat Usage

The usage statement for the Ensemble Stat tool is shown below:

```
Usage: ensemble_stat
       n_ens file_1 ... file_n | file_list
       config_file
       [-grid_obs file]
       [-point_obs file]
       [-ens_mean file]
       [-ctrl file]
       [-obs_valid_beg time]
       [-obs_valid_end time]
       [-outdir path]
       [-log file]
       [-v level]
       [-compress level]
```

ensemble_stat has two required arguments and accepts several optional ones.

14.3.1.1 Required Arguments ensemble_stat

1. The **n_ens file_1 ... file_n | file_list** specifies either the number of ensemble members followed by a list of ensemble member file names or an ASCII file list of the file names to be used, as described in Section 4.1.1.
2. The **config_file** is an **EnsembleStatConfig** file containing the desired configuration settings.

14.3.1.2 Optional Arguments for ensemble_stat

3. To produce ensemble statistics using gridded observations, use the **-grid_obs file** option to specify a gridded observation file. This option may be used multiple times if your observations are in several files.

4. To produce ensemble statistics using point observations, use the **-point_obs file** option to specify a NetCDF point observation file. This option may be used multiple times if your observations are in several files. Python embedding for point observations is also supported, as described in Section 38.4.2.

5. To override the simple ensemble mean value of the input ensemble members for the ECNT, SSVAR, and ORANK line types, the **-ens_mean file** option specifies an ensemble mean model data file. This option replaces the **-ssvar_mean file** option from earlier versions of MET.

6. The **-ctrl file** option specifies an ensemble control member data file. The control member is included in the computation of the ensemble mean but excluded from the spread. The control file should not appear in the list of ensemble member files (unless processing a single file that contains all ensemble members).

7. To filter point observations by time, use **-obs_valid_beg time** in YYYYMMDD[_HH[MMSS]] format to set the beginning of the matching observation time window.

8. As above, use **-obs_valid_end time** in YYYYMMDD[_HH[MMSS]] format to set the end of the matching observation time window.

9. Specify the **-outdir path** option to override the default output directory (./).

10. The **-log** file outputs log messages to the specified file.

11. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

12. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the ensemble_stat calling sequence is shown below:

```
ensemble_stat \
6 sample_fcst/2009123112/*gep*/d01_2009123112_02400.grib \
config/EnsembleStatConfig \
-grid_obs sample_obs/ST4/ST4.2010010112.24h \
-point_obs out/ascii2nc/precip24_2010010112.nc \
-outdir out/ensemble_stat -v 2
```

In this example, the Ensemble-Stat tool will process six forecast files specified in the file list into an ensemble forecast. Observations in both point and grid format will be included, and be used to compute ensemble statistics separately. Ensemble Stat will create a NetCDF file containing requested ensemble fields and an output STAT file.

14.3.2 ensemble_stat Configuration File

The default configuration file for the Ensemble-Stat tool named **EnsembleStatConfig_default** can be found in the installed *share/met/config* directory. Another version is located in *scripts/config*. We encourage users to make a copy of these files prior to modifying their contents. Each configuration file (both the default and sample) contains many comments describing its contents. The contents of the configuration file are also described in the subsections below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
model                    = "FCST";
desc                     = "NA";
obtype                   = "ANALYS";
regrid                   = { ... }
climo_mean               = { ... }
climo_stdev              = { ... }
climo_cdf                = { ... }
land_mask                = { ... }
topo_mask                = { ... }
lapse_rate_correction    = { ... }
msl_agl_conversion       = { ... }
obs_window               = { ... }
mask                     = { grid = [ "FULL" ];
                             poly = [];
                             sid = []; }
ci_alpha                 = [ 0.05 ];
interp                   = { ... }
eclv_points              = [];
sid_inc                  = [];
sid_exc                  = [];
duplicate_flag           = NONE;
obs_quality_inc          = [];
obs_quality_exc          = [];
obs_summary              = NONE;
obs_perc_value           = 50;
message_type_group_map   = [...];
obtype_as_group_val_flag = FALSE;
grid_weight_flag         = NONE;
point_weight_flag        = NONE;
output_prefix            = "";
version                  = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

Note: The **HIRA** interpolation method is only supported in Ensemble-Stat.

Note: The "grid_weight_flag" and "point_weight_flag" options described in Section 5 define how matched pairs are weighted for grid-to-grid and grid-to-point verification in Ensemble-Stat. These weights currently only apply to the computation of probabilistic outputs (PCT, PSTD, PJC, and PRC) but no other Ensemble-Stat output line types.

When processing the **fcst** data, compute a ratio of the number of valid ensemble fields to the total number of ensemble members. If this ratio is less than the **ens_thresh**, then quit with an error. This threshold must be between 0 and 1. Setting this threshold to 1 will require that all ensemble members be present to be processed.

When processing the **fcst** data, for each grid point compute a ratio of the number of valid data values to the number of ensemble members. If that ratio is less than **vld_thresh**, write out bad data. This threshold must be between 0 and 1. Setting this threshold to 1 will require each grid point to contain valid data for all ensemble members.

For each **field** listed in the forecast field, give the name and vertical or accumulation level, plus one or more categorical thresholds. The thresholds are specified using symbols, as shown above. It is the user's responsibility to know the units for each model variable and to choose appropriate threshold values. The thresholds are used to define ensemble relative frequencies, e.g. a threshold of >=5 can be used to compute the proportion of ensemble members predicting precipitation of at least 5mm at each grid point.

```
is_prob = TRUE;
```

While **is_prob** is not a unique setting to the Ensemble-Stat tool, setting this boolean to **TRUE** in the **fcst** dictionary field does allow users the ability to pass ensemble member data in probabilistic space relative to a climatology. Doing so will disable the requirement for setting **prob_cat_thresh** in the **fcst** dictionary field but does require that the **obs** dictionary field contain a **prob_cat_thresh** array with the same number of inequality entries as ensemble members being evaluated. These inequalities must be monotonically increasing and of the same inequality type. Using this feature will disable all **output_flag** line type outputs requested other than the RPS. Additionally the RPS line type columns decomposing the RPS into Reliability, Resolution, and Uncertainty, as well as the RPSS relative to a sample climatology, will have entries of N/A.

```
ens_member_ids = [];
control_id = "";
```

The **ens_member_ids** array is only used if reading a single file that contains all ensemble members. It should contain a list of string identifiers that are substituted into the **fcst** dictionary field to determine which data to read from the file. The length of the array determines how many ensemble members will be processed for a given field. Each value in the array will replace the text **MET_ENS_MEMBER_ID**.

NetCDF Example:

```
fcst = {
  field = [
    {
      name  = "fcst";
      level = "(MET_ENS_MEMBER_ID,0,*,*)";
    }
  ];
}
```

GRIB Example:

```
fcst = {
  field = [
    {
      name     = "fcst";
      level    = "L0";
      GRIB_ens = "MET_ENS_MEMBER_ID";
    }
  ];
}
```

The **ens_member_ids** array can also be used to control the **name** field of the **fcst** dictionary. When used in this manner, Ensemble-Stat will use the first string of **ens_member_ids** to set the **FCST_VAR** column in line type output. Users may find it useful to override this behavior using **set_attr_name**:

```
fcst = {
  ens_thresh        = 0.75;
  vld_thresh        = 1.0;

  field = [
    {
      name          = "MET_ENS_MEMBER_ID";
      level         = "(*,*)";
      set_attr_name = "TMEAN_TERCILES";
    }
  ];
}
```

control_id is a string that is substituted in the same way as the **ens_member_ids** values to read a control member. This value is only used when the **-ctrl** command line argument is used. The value should not be found in the **ens_member_ids** array.

```
obs_thresh = [ NA ];
```

The **obs_thresh** entry is an array of thresholds for filtering observation values prior to applying ensemble verification logic. The default setting of **NA** means that no observations should be filtered out. Verification

output will be computed separately for each threshold specified. This option may be set separately for each **obs.field** entry.

```
skip_const = FALSE;
```

Setting **skip_const** to true tells Ensemble-Stat to exclude pairs where all the ensemble members and the observation have a constant value. For example, exclude points with zero precipitation amounts from all output line types. This option may be set separately for each **obs.field** entry. When set to false, constant points are and the observation rank is chosen at random.

```
ens_ssvar_bin_size = 1.0;
ens_phist_bin_size = 0.05;
```

Setting up the **fcst** and **obs** dictionaries of the configuration file is described in Section 5. The following are some special considerations for the Ensemble-Stat tool.

The **obs** dictionary looks very similar to the **fcst** dictionary. If verifying against point observations which are assigned GRIB1 codes, the observation section must be defined following GRIB1 conventions. When verifying GRIB1 forecast data, one can easily copy over the forecast settings to the observation dictionary using **obs = fcst;**. However, when verifying non-GRIB1 forecast data, users will need to specify the **fcst** and **obs** sections separately.

The **ens_ssvar_bin_size** and **ens_phist_bin_size** specify the width of the categorical bins used to accumulate frequencies for spread-skill-variance or probability integral transform statistics, respectively.

```
prob_cat_thresh = [];
prob_pct_thresh = [];
```

The **prob_cat_thresh** entry is an array of thresholds. It is applied both to the computation of the RPS line type as well as the when generating probabilistic output line types. Since these thresholds can change for each variable, they can be specified separately for each **fcst.field** entry. If left empty but climatological mean and standard deviation data is provided, the **climo_cdf** thresholds will be used instead. If no climatology data is provided, and the RPS output line type is requested, then the **prob_cat_thresh** array must be defined. When probabilistic output line types are requested, for each **prob_cat_thresh** threshold listed, ensemble relative frequencies are derived and verified against the point and/or gridded observations.

The **prob_pct_thresh** entry is an array of thresholds which define the Nx2 probabilistic contingency table used to evaluate probability forecasts. It can be specified separately for each **fcst.field** entry. These thresholds must span the range [0, 1]. A shorthand notation to create equal bin widths is provided. For example, the following setting creates 4 probability bins of width 0.25 from 0 to 1.

```
prob_pct_thresh = [ ==0.25 ];
```

```
obs_error = {
   flag             = FALSE;
   dist_type        = NONE;
   dist_parm        = [];
   inst_bias_scale  = 1.0;
   inst_bias_offset = 0.0;
}
```

The **obs_error** dictionary controls how observation error information should be handled. This dictionary may be set separately for each **obs.field** entry. Observation error information can either be specified directly in the configuration file or by parsing information from an external table file. By default, the **MET_BASE/share/met/table_files/obs_error_table.txt** file is read but this may be overridden by setting the **$MET_OBS_ERROR_TABLE** environment variable at runtime.

The **flag** entry toggles the observation error logic on (**TRUE**) and off (**FALSE**). When the **flag** is **TRUE**, random observation error perturbations are applied to the ensemble member values. No perturbation is applied to the observation values but the bias scale and offset values, if specified, are applied.

The **dist_type** entry may be set to **NONE, NORMAL, LOGNORMAL, EXPONENTIAL,CHISQUARED, GAMMA, UNIFORM**, or **BETA**. The default value of **NONE** indicates that the observation error table file should be used rather than the configuration file settings.

The **dist_parm** entry is an array of length 1 or 2 specifying the parameters for the distribution selected in **dist_type**. The **GAMMA, UNIFORM**, and **BETA** distributions are defined by two parameters, specified as a comma-separated list (a,b), whereas all other distributions are defined by a single parameter.

The **inst_bias_scale** and **inst_bias_offset** entries specify bias scale and offset values that should be applied to observation values prior to perturbing them. These entries enable bias-correction on the fly.

Defining the observation error information in the configuration file is convenient but limited. The random perturbations for all points in the current verification task are drawn from the same distribution. Specifying an observation error table file instead (by setting **dist_type = NONE;**) provides much finer control, enabling the user to define observation error distribution information and bias-correction logic separately for each observation variable name, message type, PrepBUFR report type, input report type, instrument type, station ID, range of heights, range of pressure levels, and range of values.

```
output_flag = {
   ecnt  = NONE;
   rps   = NONE;
   rhist = NONE;
   phist = NONE;
   orank = NONE;
   ssvar = NONE;
   relp  = NONE;
   pct   = NONE;
   pstd  = NONE;
   pjc   = NONE;
   prc   = NONE;
```

(continues on next page)

(continued from previous page)

```
   eclv  = NONE;
}
```

The **output_flag** array controls the type of output that is generated. Each flag corresponds to an output line type in the STAT file. Setting the flag to NONE indicates that the line type should not be generated. Setting the flag to STAT indicates that the line type should be written to the STAT file only. Setting the flag to BOTH indicates that the line type should be written to the STAT file as well as a separate ASCII file where the data is grouped by line type. The output flags correspond to the following output line types:

1. **ECNT** for Continuous Ensemble Statistics
2. **RPS** for Ranked Probability Score Statistics
3. **RHIST** for Ranked Histogram Counts
4. **PHIST** for Probability Integral Transform Histogram Counts
5. **ORANK** for Ensemble Matched Pair Information when point observations are supplied
6. **SSVAR** for Binned Spread/Skill Variance Information
7. **RELP** for Relative Position Counts
8. **PCT** for Contingency Table counts for derived ensemble relative frequencies
9. **PSTD** for Probabilistic statistics for dichotomous outcomes for derived ensemble relative frequencies
10. **PJC** for Joint and Conditional factorization for derived ensemble relative frequencies
11. **PRC** for Receiver Operating Characteristic for derived ensemble relative frequencies
12. **ECLV** for Economic Cost/Loss Relative Value for derived ensemble relative frequencies

---

```
nc_orank_flag = {
   latlon    = TRUE;
   mean      = TRUE;
   raw       = TRUE;
   rank      = TRUE;
   pit       = TRUE;
   vld_count = TRUE;
   weight    = FALSE;
}
```

The **nc_orank_flag** specifies which gridded verification output types should be written to the Observation Rank (**_orank.nc**) NetCDF file. This output file is only created when gridded observations have been provided with the -grid_obs command line option. Setting the flag to TRUE produces output of the specified field, while FALSE produces no output for that field type. The flags correspond to the following output line types:

1. Grid Latitude and Longitude Fields
2. Ensemble mean field

3. Raw observation values
4. Observation ranks
5. Observation probability-integral transform values
6. Ensemble valid data count
7. Grid area weight values

```
nc_var_str = "";
```

The **nc_var_str** entry specifies a string for each ensemble field and verification task. This string is parsed from each **ens.field** and **obs.field** dictionary entry and is used to customize the variable names written to theNetCDF output file. The default is an empty string, meaning that no customization is applied to the output variable names. When the Ensemble-Stat config file contains two fields with the same name and level value, this entry is used to make the resulting variable names unique.

```
rng = {
   type = "mt19937";
   seed = "";
   }
```

The **rng** group defines the random number generator **type** and **seed** to be used. In the case of a tie when determining the rank of an observation, the rank is randomly chosen from all available possibilities. The randomness is determined by the random number generator specified.

The **seed** variable may be set to a specific value to make the assignment of ranks fully repeatable. When left empty, as shown above, the random number generator seed is chosen automatically which will lead to slightly different bootstrap confidence intervals being computed each time the data is run.

Refer to the description of the **boot** entry in Section 5 for more details on the random number generator.

14.3.3 ensemble_stat Output

ensemble_stat can produce output in STAT, ASCII, and NetCDF formats. The ASCII output duplicates the STAT output but has the data organized by line type. The output files are written to the default output directory or the directory specified by the -outdir command line option.

The output STAT file is named using the following naming convention:

ensemble_stat_PREFIX_YYYYMMDD_HHMMSSV.stat where PREFIX indicates the user-defined output prefix and YYYYMMDD_HHMMSSV indicates the forecast valid time. Note that the forecast lead time is not included in the output file names since it would not be well-defined for time-lagged ensembles. When verifying multiple lead times for the same valid time, users should either write the output to separate directories or specify an output prefix to ensure unique file names.

The output ASCII files are named similarly:

ensemble_stat_PREFIX_YYYYMMDD_HHMMSSV_TYPE.txt where TYPE is one of elements of the **output_flag** configuration option to indicate the line type it contains.

When verification against gridded analyses is performed, Ensemble-Stat can produce output NetCDF files using the following naming convention:

ensemble_stat_PREFIX_YYYYMMDD_HHMMSSV_orank.nc contains gridded fields of observation ranks when the -grid_obs command line option is used. Its contents are specified by the **nc_orank_flag** configuration option.

The Ensemble-Stat tool can compute the following statistics for the fields specified in the fcst and obs dictionaries of the configuration file:

Continuous Ensemble Statistics

Ranked Histograms

Probability Integral Transform (PIT) Histograms

Relative Position Histograms

Spread/Skill Variance

Ensemble Matched Pair information

The format of the STAT and ASCII output of the Ensemble-Stat tool are described below.

Table 14.1: Header information for Ensemble-Stat STAT output

Column Number	Header Column Name	Description	Data Type
1	VERSION	Version number	String
2	MODEL	User-provided text string designating model name	String
3	DESC	User-provided text string describing the verification task	String
4	FCST_LEAD	Forecast lead time in HHMMSS format	Time String
5	FCST_VALID_BEG	Forecast valid start time in YYYYMMDD_HHMMSS format	Datetime String
6	FCST_VALID_END	Forecast valid end time in YYYYMMDD_HHMMSS format	Datetime String
7	OBS_LEAD	Observation lead time in HHMMSS format	Time String
8	OBS_VALID_BEG	Observation valid start time in YYYYMMDD_HHMMSS format	Datetime String
9	OBS_VALID_END	Observation valid end time in YYYYMMDD_HHMMSS format	Datetime String
10	FCST_VAR	Model variable	String
11	FCST_UNITS	Units for model variable	String
12	FCST_LEV	Selected vertical level for forecast	String
13	OBS_VAR	Observation variable	String
14	OBS_UNITS	Units for observation variable	String
15	OBS_LEV	Selected vertical level for observations	String
16	OBTYPE	Type of observation selected	String
17	VX_MASK	Verifying masking region indicating the masking grid or polyline region applied	String
18	INTERP_MTHD	Interpolation method applied to forecasts	String
19	INTERP_PNTS	Number of points used in interpolation method	Integer
20	FCST_THRESH	The threshold applied to the forecast	Threshold String
21	OBS_THRESH	The threshold applied to the observations	Threshold String
22	COV_THRESH	The minimum fraction of valid ensemble members required to calculate statistics	Threshold String
23	ALPHA	Error percent value used in confidence intervals	Double
24	LINE_TYPE	Output line types are listed in Table 14.2 through Table 14.8	String

Table 14.2: Format information for ECNT (Ensemble Continuous Statistics) output line type.

Column Number	ECNT Column Name	Description	Data Type
24	ECNT	Ensemble Continuous Statistics line type	String
25	TOTAL	Count of observations	Integer
26	N_ENS	Number of ensemble values	Integer
27	CRPS	The Continuous Ranked Probability Score (normal distribution)	Double
28	CRPSS	The Continuous Ranked Probability Skill Score (normal distribution)	Double
29	IGN	The Ignorance Score	Double
30	ME	The Mean Error of the ensemble mean (unperturbed or supplied)	Double
31	RMSE	The Root Mean Square Error of the ensemble mean (unperturbed or supplied)	Double
32	SPREAD	The square root of the mean of the variance of the unperturbed ensemble member values at each observation location	Double
33	ME_OERR	The Mean Error of the PERTURBED ensemble mean (e.g. with Observation Error)	Double
34	RMSE_OERR	The Root Mean Square Error of the PERTURBED ensemble mean (e.g. with Observation Error)	Double
35	SPREAD_OERR	The square root of the mean of the variance of the PERTURBED ensemble member values (e.g. with Observation Error) at each observation location	Double
36	SPREAD_PLUS_OERR	The square root of the sum of unperturbed ensemble variance and the observation error variance	Double
37	CRPSCL	Climatological Continuous Ranked Probability Score (normal distribution)	Double
38	CRPS_EMP	The Continuous Ranked Probability Score (empirical distribution)	Double
39	CRPSCL_EMP	Climatological Continuous Ranked Probability Score (empirical distribution)	Double
40	CRPSS_EMP	The Continuous Ranked Probability Skill Score (empirical distribution)	Double
41	CRPS_EMP_FAIR	The Continuous Ranked Probability Score (empirical distribution) adjusted by the mean absolute difference of the ensemble members	Double
42	SPREAD_MD	The pairwise Mean Absolute Difference of the unperturbed ensemble members	Double
43	MAE	The Mean Absolute Error of the ensemble mean (unperturbed or supplied)	Double
44	MAE_OERR	The Mean Absolute Error of the PERTURBED ensemble mean (e.g. with Observation Error)	Double
45	BIAS_RATIO	The Bias Ratio	Double
46	N_GE_OBS	The number of ensemble values greater than or equal to their observations	Integer
47	ME_GE_OBS	The Mean Error of the ensemble values greater than or equal to their observations	Double

Table 14.3: Format information for RPS (Ranked Probability Score) output line type.

Column Number	RPS Column Name	Description	Data Type
24	RPS	Ranked Probability Score line type	String
25	TOTAL	Count of observations	Integer
26	N_PROB	Number of probability thresholds (i.e. number of ensemble members in Ensemble-Stat)	Integer
27	RPS_REL	RPS Reliability, mean of the reliabilities for each RPS threshold	Double
28	RPS_RES	RPS Resolution, mean of the resolutions for each RPS threshold	Double
29	RPS_UNC	RPS Uncertainty, mean of the uncertainties for each RPS threshold	Double
30	RPS	Ranked Probability Score, mean of the Brier Scores for each RPS threshold OR mean of the climatology-based probabilistic ensemble members	Double
31	RPSS	Ranked Probability Skill Score relative to external climatology OR relative to a fixed climatology value of one divided by the number of ensemble members when using climatology-based probabilistic ensemble members	Double
32	RPSS_SMPL	Ranked Probability Skill Score relative to sample climatology	Double

Table 14.4: Format information for RHIST (Ranked Histogram) output line type.

Column Number	RHIST Column Name	Description	Data Type
24	RHIST	Ranked Histogram line type	String
25	TOTAL	Count of observations	Integer
26	N_RANK	Number of possible ranks for observation	Integer
27	RANK_i	Count of observations with the ith rank (repeated)	Integer

Table 14.5: Format information for PHIST (Probability Integral Transform Histogram) output line type.

Column Number	PHIST Column Name	Description	Data Type
24	PHIST	Probability Integral Transform line type	String
25	TOTAL	Count of observations	Integer
26	BIN_SIZE	Probability interval width	Double
27	N_BIN	Total number of probability intervals	Integer
28	BIN_i	Count of observations in the ith probability bin (repeated)	Integer

Table 14.6: Format information for RELP (Relative Position) output line type.

Column Number	RELP Column Name	Description	Data Type
24	RELP	Relative Position line type	String
25	TOTAL	Count of observations	Integer
26	N_ENS	Number of ensemble members	Integer
27	RELP_i	Number of times the ith ensemble member's value was closest to the observation (repeated). When n members tie, 1/n is assigned to each member.	Double

Table 14.7: Format information for ORANK (Observation Rank) output line type.

Column Number	ORANK Column Name	Description	Data Type
24	ORANK	Observation Rank line type	String
25	TOTAL	Count of observations	Integer
26	INDEX	Index for the current ensemble pair	Integer
27	OBS_SID	Station Identifier	String
28	OBS_LAT	Latitude of the observation	Double
29	OBS_LON	Longitude of the observation	Double
30	OBS_LVL	Level of the observation	Double
31	OBS_ELV	Elevation of the observation	Double
32	OBS	Value of the observation	Double
33	PIT	Probability Integral Transform	Double
34	RANK	Rank of the observation	Integer
35	N_ENS_VLD	Number of valid ensemble values	Integer
36	N_ENS	Number of ensemble values	Integer
37	ENS_i	Value of the ith ensemble member (repeated)	Double
Last-9	OBS_QC	Quality control string for the observation	String
Last-8	ENS_MEAN	The unperturbed ensemble mean value	Double
Last-7	OBS_CLIMO_MEAN	Observation climatological mean value (named CLIMO_MEAN prior to met-12.0.0)	Double
Last-6	SPREAD	The spread (standard deviation) of the unperturbed ensemble member values	Double
Last-5	ENS_MEAN_OERR	The PERTURBED ensemble mean (e.g. with Observation Error)	Double
Last-4	SPREAD_OERR	The spread (standard deviation) of the PERTURBED ensemble member values (e.g. with Observation Error)	Double
Last-3	SPREAD_PLUS_OERR	The square root of the sum of the unperturbed ensemble variance and the observation error variance	Double
Last-2	OBS_CLIMO_STDEV	Observation climatological standard deviation value (named CLIMO_STDEV prior to met-12.0.0)	Double
Last-1	FCST_CLIMO_MEAN	Forecast climatological mean value	Double
Last	FCST_CLIMO_STDEV	Forecast climatological standard deviation value	Double

Table 14.8: Format information for SSVAR (Spread/Skill Variance) output line type.

Column Number	SSVAR Column Name	Description	Data Type
24	SSVAR	Spread/Skill Variance line type	String
25	TOTAL	Count of observations	Integer
26	N_BIN	Number of bins for current forecast run	Integer
27	BIN_i	Index of the current bin	Integer
28	BIN_N	Number of points in bin i	Integer
29	VAR_MIN	Minimum variance	Double
30	VAR_MAX	Maximum variance	Double
31	VAR_MEAN	Average variance	Double
32	FBAR	Average forecast value	Double
33	OBAR	Average observed value	Double
34	FOBAR	Average product of forecast and observation	Double
35	FFBAR	Average of forecast squared	Double
36	OOBAR	Average of observation squared	Double
37-38	FBAR_NCL, FBAR_NCU	Mean forecast normal upper and lower confidence limits	Double
39-41	FSTDEV, FSTDEV_NCL, FSTDEV_NCU	Standard deviation of the error including normal upper and lower confidence limits	Double
42-43	OBAR_NCL, OBAR_NCU	Mean observation normal upper and lower confidence limits	Double
44-46	OSTDEV, OSTDEV_NCL, OSTDEV_NCU	Standard deviation of the error including normal upper and lower confidence limits	Double
47-49	PR_CORR, PR_CORR_NCL, PR_CORR_NCU	Pearson correlation coefficient including normal upper and lower confidence limits	Double
50-52	ME, ME_NCL, ME_NCU	Mean error including normal upper and lower confidence limits	Double
53-55	ESTDEV, ESTDEV_NCL, ESTDEV_NCU	Standard deviation of the error including normal upper and lower confidence limits	Double
56	MBIAS	Magnitude bias	Double
57	MSE	Mean squared error	Double
58	BCMSE	Bias corrected root mean squared error	Double
59	RMSE	Root mean squared error	Double

Chapter 15

Wavelet-Stat Tool

15.1 Introduction

The Wavelet-Stat tool decomposes two-dimensional forecasts and observations according to intensity and scale. This section describes the Wavelet-Stat tool, which enables users to apply the Intensity-Scale verification technique described by *Casati et al. (2004)* (page 493).

The Intensity-Scale technique is one of the recently developed verification approaches that focus on verification of forecasts defined over spatial domains. Spatial verification approaches, as opposed to point-by-point verification approaches, aim to account for the presence of features and for the coherent spatial structure characterizing meteorological fields. Since these approaches account for the intrinsic spatial correlation existing between nearby grid-points, they do not suffer from point-by-point comparison related verification issues, such as double penalties. Spatial verification approaches aim to account for the observation and forecast time-space uncertainties, and aim to provide feedback on the forecast error in physical terms.

The Intensity-Scale verification technique, as most of the spatial verification approaches, compares a forecast field to an observation field. To apply the Intensity-Scale verification approach, observations need to be defined over the same spatial domain of the forecast to be verified.

Within the spatial verification approaches, the Intensity-Scale technique belongs to the scale-decomposition (or scale-separation) verification approaches. The scale-decomposition approaches enable users to perform the verification on different spatial scales. Weather phenomena on different scales (e.g. frontal systems versus convective showers) are often driven by different physical processes. Verification on different spatial scales can therefore provide deeper insights into model performance at simulating these different processes.

The spatial scale components are obtained usually by applying a single band spatial filter to the forecast and observation fields (e.g. Fourier, Wavelets). The scale-decomposition approaches measure error, bias and skill of the forecast on each different scale component. The scale-decomposition approaches therefore provide feedback on the scale dependency of the error and skill, on the no-skill to skill transition scale, and on the capability of the forecast of reproducing the observed scale structure.

The Intensity-Scale technique evaluates the forecast skill as a function of the intensity values and of the spatial scale of the error. The scale components are obtained by applying a two dimensional Haar wavelet filter. Note that wavelets, because of their locality, are suitable for representing discontinuous fields characterized by few sparse non-zero features, such as precipitation. Moreover, the technique is based on a categorical approach, which is a robust and resistant approach, suitable for non-normally distributed variables, such as

precipitation. The intensity-scale technique was specifically designed to cope with the difficult characteristics of precipitation fields, and for the verification of spatial precipitation forecasts. However, the intensity-scale technique can also be applied to verify other variables, such as cloud fraction.

15.2 Scientific and Statistical Aspects

15.2.1 The Method

Casati et al. (2004) (page 493) applied the Intensity-Scale verification to preprocessed and re-calibrated (unbiased) data. The preprocessing was aimed to mainly normalize the data, and defined categorical thresholds so that each categorical bin had a similar sample size. The recalibration was performed to eliminate the forecast bias. Preprocessing and recalibration are not strictly necessary for the application of the Intensity-Scale technique. The MET Intensity-Scale Tool does not perform either, and applies the Intensity-Scale approach to biased forecasts, for categorical thresholds defined by the user.

The Intensity Scale approach can be summarized in the following 5 steps:

1. For each threshold, the forecast and observation fields are transformed into binary fields: where the grid-point precipitation value meets the threshold criteria it is assigned 1, where the threshold criteria are not met it is assigned 0. This can also be done with no thresholds indicated at all and in that case the grid-point values are not transformed to binary fields and instead the raw data is used as is for statistics. Figure 15.1 illustrates an example of a forecast and observation fields, and their corresponding binary fields for a threshold of 1mm/h. This case shows an intense storm of the scale of 160 km displaced almost its entire length. The displacement error is clearly visible from the binary field difference and the contingency table image obtained for the same threshold Table 15.1.

2. The binary forecast and observation fields obtained from the thresholding are then decomposed into the sum of components on different scales, by using a 2D Haar wavelet filter (Figure 15.3). Note that the scale components are fields, and their sum adds up to the original binary field. For a forecast defined over square domain of $2^n \times 2^n$ grid-points, the scale components are **n+1: n** mother wavelet components + the largest father wavelet (or scale-function) component. The **n** mother wavelet components have resolution equal to **1, 2, 4, ...** 2^{n-1} grid-points. The largest father wavelet component is a constant field over the $2^n \times 2^n$ grid-point domain with value equal to the field mean.

Note that the wavelet transform is a linear operator: this implies that the difference of the spatial scale components of the binary forecast and observation fields (Figure 15.3) are equal to the spatial scale components of the difference of the binary forecast and observation fields (Figure 15.2), and these scale components also add up to the original binary field difference (Figure 15.1). The intensity-scale technique considers thus the spatial scale of the error. For the case illustrated (Figure 15.1 and Figure 15.2) note the large error associated at the scale of 160 km, due the storm, 160km displaced almost its entire length.

Note also that the means of the binary forecast and observation fields (i.e. their largest father wavelet components) are equal to the proportion of forecast and observed events above the threshold, **(a+b)/n** and **(a+c)/n**, evaluated from the contingency table counts (Table 15.1) obtained from the original forecast and observation fields by thresholding with the same threshold used to obtain the binary forecast and observation fields. This relation is intuitive when observing forecast and observation binary fields and their corresponding contingency table image (Figure 15.1). The comparison of the largest father wavelet component of binary forecast and observation fields therefore provides feedback on the whole field bias.

3. For each threshold (**t**) and for each scale component (**j**) of the binary forecast and observation, the Mean Squared Error (MSE) is then evaluated (Figure 15.4). The error is usually large for small thresholds, and decreases as the threshold increases. This behavior is partially artificial, and occurs because the smaller the threshold the more events will exceed it, and therefore the larger would be the error, since the error tends to be proportional to the amount of events in the binary fields. The artificial effect can be diminished by normalization: because of the wavelet orthogonal properties, the sum of the MSE of the scale components is equal to the MSE of the original binary fields: $MSE(t) = jMSE(t, j)$. Therefore, the percentage that the MSE for each scale contributes to the total MSE may be computed: for a given threshold, **t**, $MSE\%(t, j) = MSE(t, j)/MSE(t)$. The MSE% does not exhibit the threshold dependency, and usually shows small errors on large scales and large errors on small scales, with the largest error associated to the smallest scale and highest threshold. For the NIMROD case illustrated, note the large error at 160 km and between the thresholds of and 4 mm/h, due to the storm, 160km displaced almost its entire length.

Note that the MSE of the original binary fields is equal to the proportion of the counts of misses (**c/n**) and false alarms (**b/n**) for the contingency table (Table 15.1) obtained from the original forecast and observation fields by thresholding with the same threshold used to obtain the binary forecast and observation fields: $MSE(t) = (b + c)/n$. This relation is intuitive when comparing the forecast and observation binary field difference and their corresponding contingency table image (Table 15.1).

4. The MSE for the random binary forecast and observation fields is estimated by $MSE(t)random = FBI * Br * (1 - Br) + Br * (1 - FBI * Br)$, where $FBI = (a + b)/(a + c)$ is the frequency bias index and $Br = (a + c)/n$ is the sample climatology from the contingency table (Table 15.1) obtained from the original forecast and observation fields by thresholding with the same threshold used to obtain the binary forecast and observation fields. This formula follows by considering the *Murphy and Winkler (1987)* (page 497) framework, applying the Bayes' theorem to express the joint probabilities **b/n** and **c/n** as product of the marginal and conditional probability (e.g. *Jolliffe and Stephenson, 2012* (page 496); *Wilks, 2010* (page 500)), and then noticing that for a random forecast the conditional probability is equal to the unconditional one, so that **b/n** and **c/n** are equal to the product of the corresponding marginal probabilities solely.

5. For each threshold (**t**) and scale component (**j**), the skill score based on the MSE of binary forecast and observation scale components is evaluated (Figure 15.5). The standard skill score definition as in *Jolliffe and Stephenson (2012)* (page 496) or *Wilks (2010)* (page 500) is used, and random chance is used as reference forecast. The MSE for the random binary forecast is equipartitioned on the **n+1** scales to evaluate the skill score: $SS(t, j) = 1 - MSE(t, j) * (n + 1)/MSE(t)random$

The Intensity-Scale (IS) skill score evaluates the forecast skill as a function of the precipitation intensity and of the spatial scale of the error. Positive values of the IS skill score are associated with a skillful forecast, whereas negative values are associated with no skill. Usually large scales exhibit positive skill (large scale events, such as fronts, are well predicted), whereas small scales exhibit negative skill (small scale events, such as convective showers, are less predictable), and the smallest scale and highest thresholds exhibit the worst skill. For the NIMROD case illustrated note the negative skill associated with the 160 km scale, for the thresholds to 4 mm/h, due to the 160 km storm displaced almost its entire length.

Table 15.1: 2x2 contingency table in terms of counts. The n_{ij} values in the table represent the counts in each forecast-observation category, where **i** represents the forecast and **j** represents the observations.

Forecast	Observation		Total
	o = 1 (e.g., "Yes")	o = 0 (e.g., "No")	
f = 1 (e.g., "Yes")	Hits = **a**	False Alarms = **b**	**a+b**
f = 0 (e.g., "No")	Misses = **c**	Correct rejections = **d**	**c+d**
Total	**a+c**	**b+d**	**a+b+c+d**

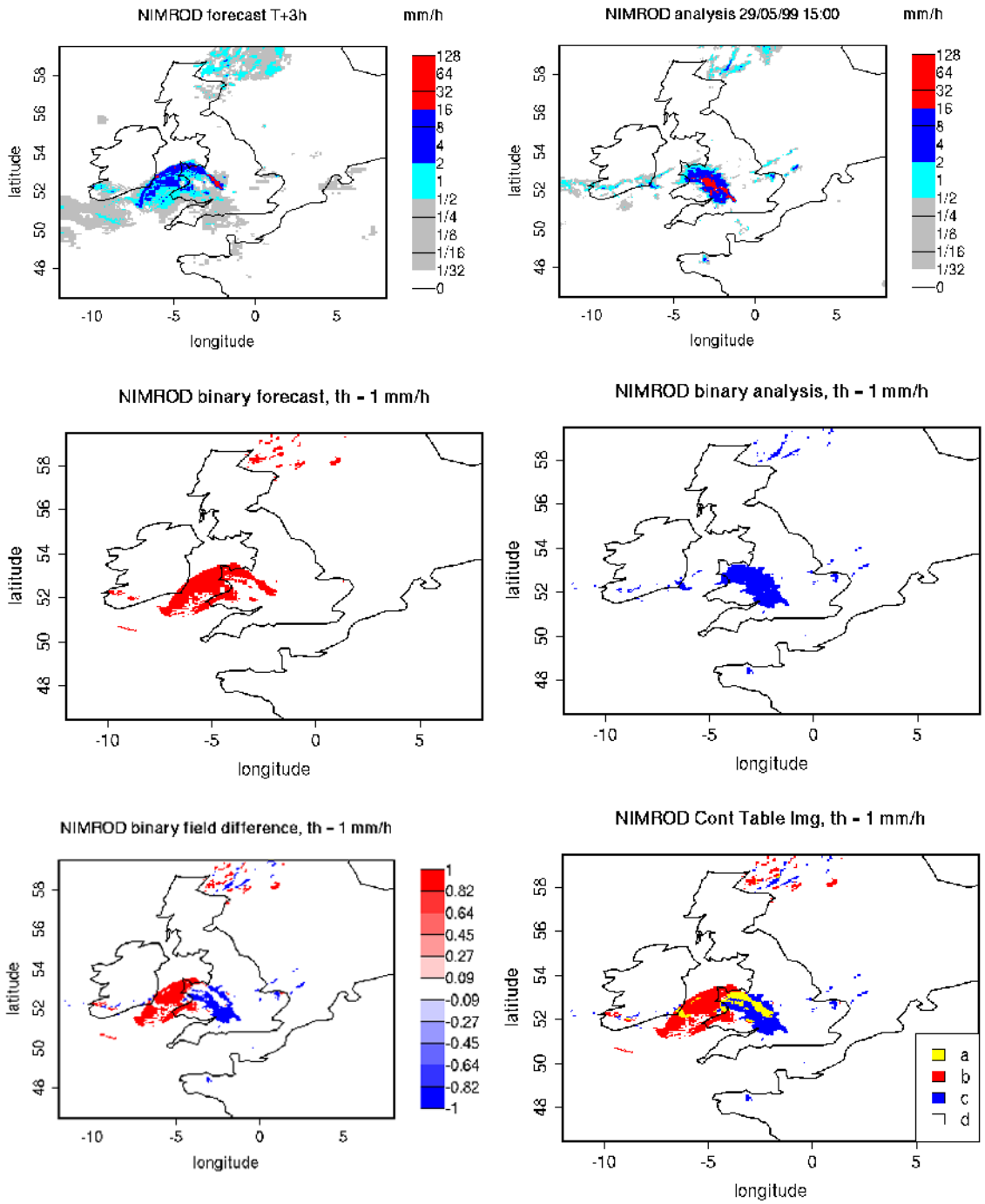

Figure 15.1: NIMROD 3h lead-time forecast and corresponding verifying analysis field (precipitation rate in mm/h, valid the 05/29/99 at 15:00 UTC); forecast and analysis binary fields obtained for a threshold of 1mm/h, the binary field difference has their corresponding Contingency Table Image (see Table 15.1). The forecast shows a storm of 160 km displaced almost its entire length.

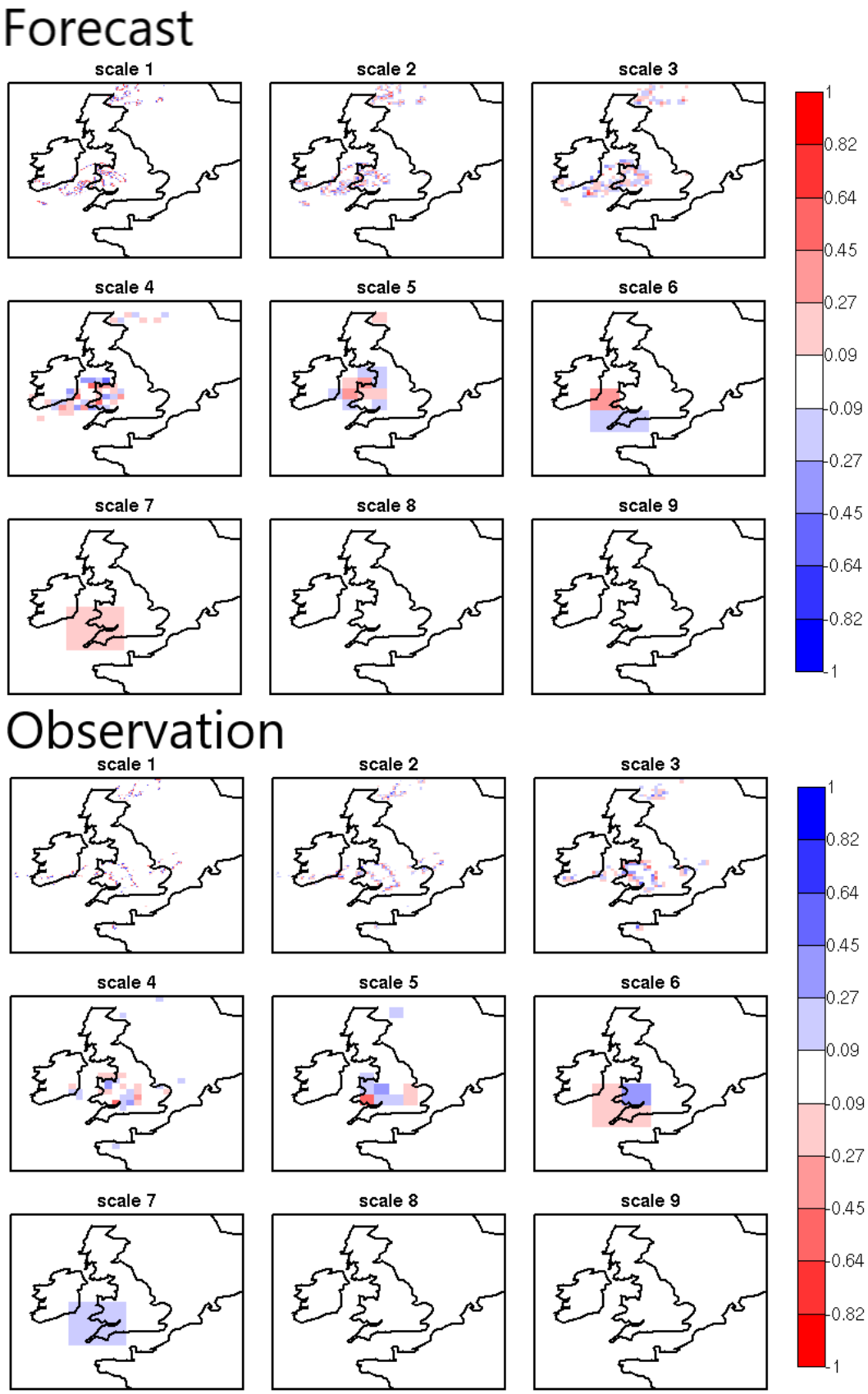

Figure 15.2: NIMROD binary forecast (top) and binary analysis (bottom) spatial scale components obtained by a 2D Haar wavelet transform (th=1 mm/h). Scales 1 to 8 refer to mother wavelet components (5, 10, 20, 40, 80, 160, 320, 640 km resolution); scale 9 refers to the largest father wavelet component (1280 km resolution).

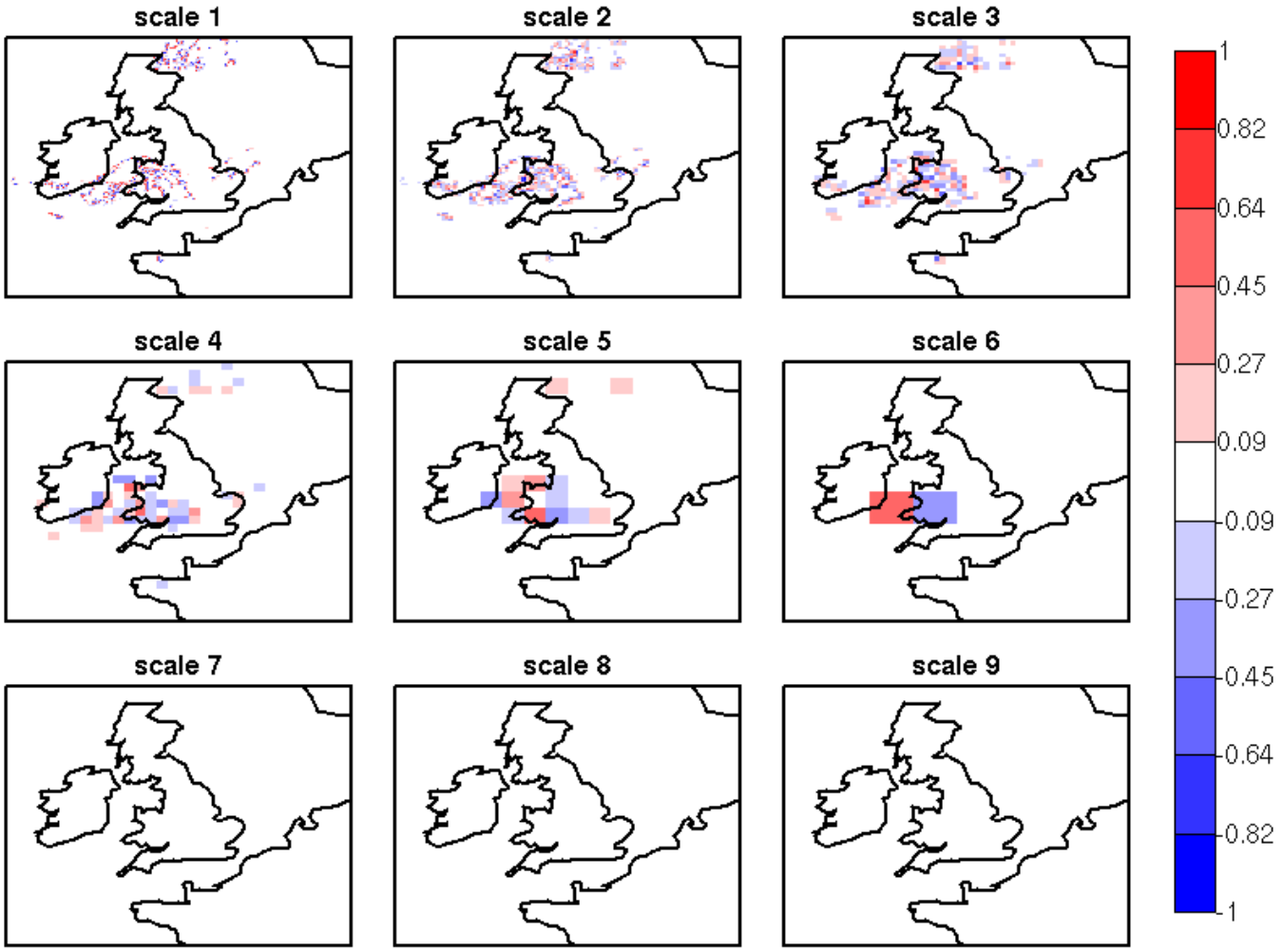

Figure 15.3: NIMROD binary field difference spatial scale components obtained by a 2D Haar wavelet transform (th=1 mm/h). Scales 1 to 8 refer to mother wavelet components (5, 10, 20, 40, 80, 160, 320, 640 km resolution); scale 9 refers to the largest father wavelet component (1280 km resolution). Note the large error at the scale 6 = 160 km, due to the storm, 160 km displaced almost of its entire length.

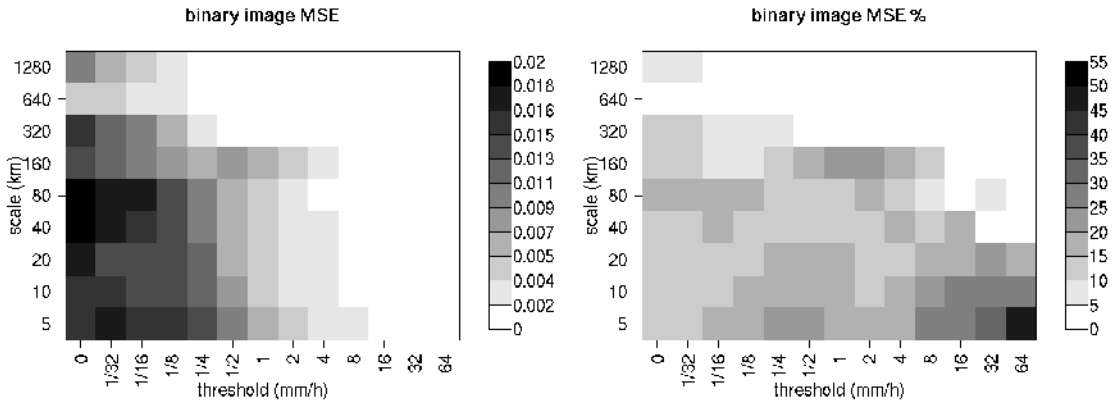

Figure 15.4: MSE and MSE % for the NIMROD binary forecast and analysis spatial scale components. In the MSE%, note the large error associated with the scale 6 = 160 km, for the thresholds ½ to 4 mm/h, associated with the displaced storm.

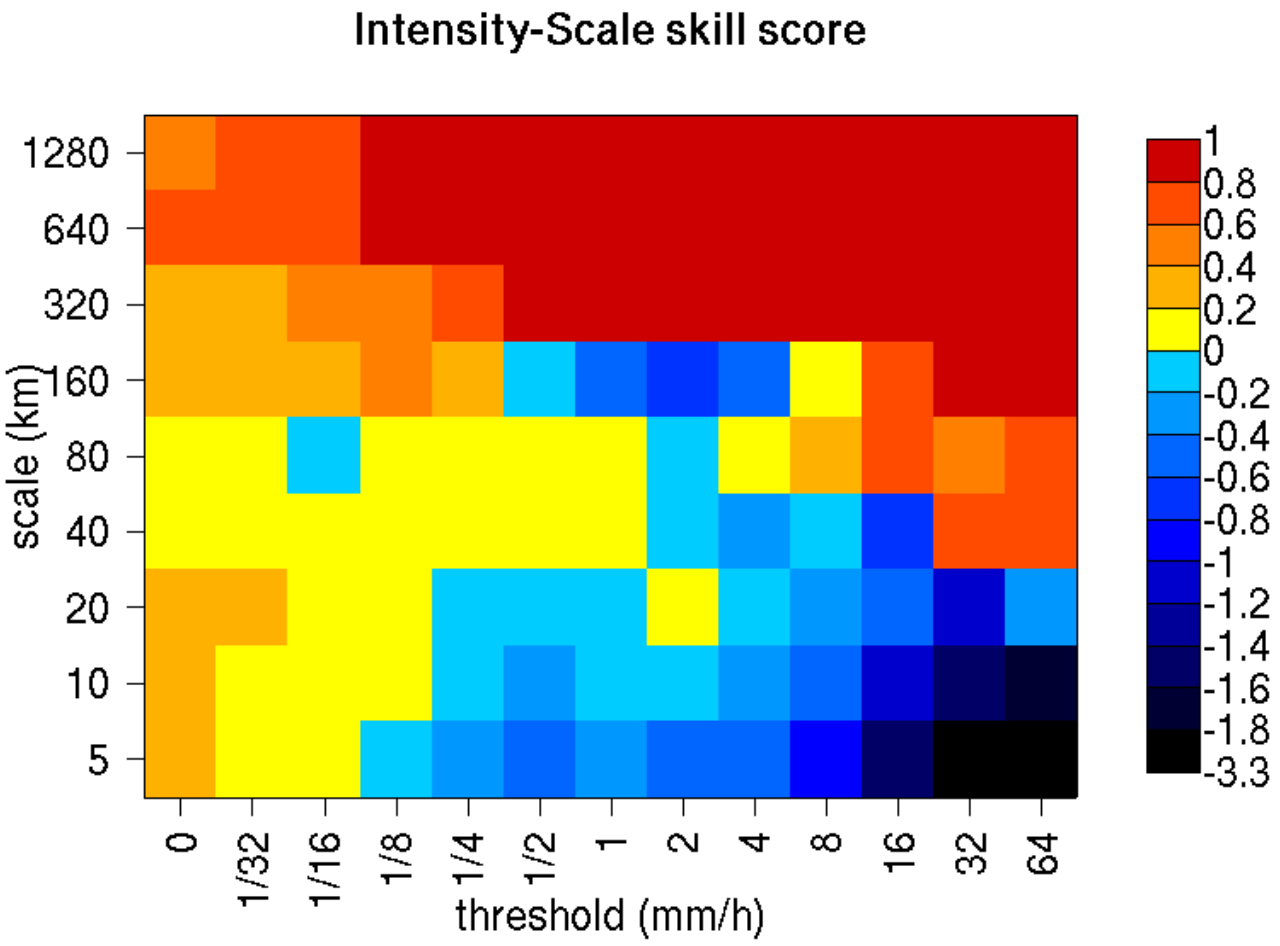

Figure 15.5: Intensity-Scale skill score for the NIMROD forecast and analysis shown in Figure 15.1. The skill score is a function of the intensity of the precipitation rate and spatial scale of the error. Note the negative skill associated with the scale 6 = 160 km, for the thresholds to 4 mm/h, associated with the displaced storm.

In addition to the MSE and the SS, the energy squared is also evaluated, for each threshold and scale (Figure 15.6). The energy squared of a field X is the average of the squared values: $En2(X) = \sum_i x_i^2$. The energy squared provides feedback on the amount of events present in the forecast and observation fields for each scale, for a given threshold. Usually, small thresholds are associated with a large energy, since many events exceed the threshold. Large thresholds are associated with a small energy, since few events exceed the threshold. Comparison of the forecast and observed squared energy provide feedback on the bias on different scales, for each threshold.

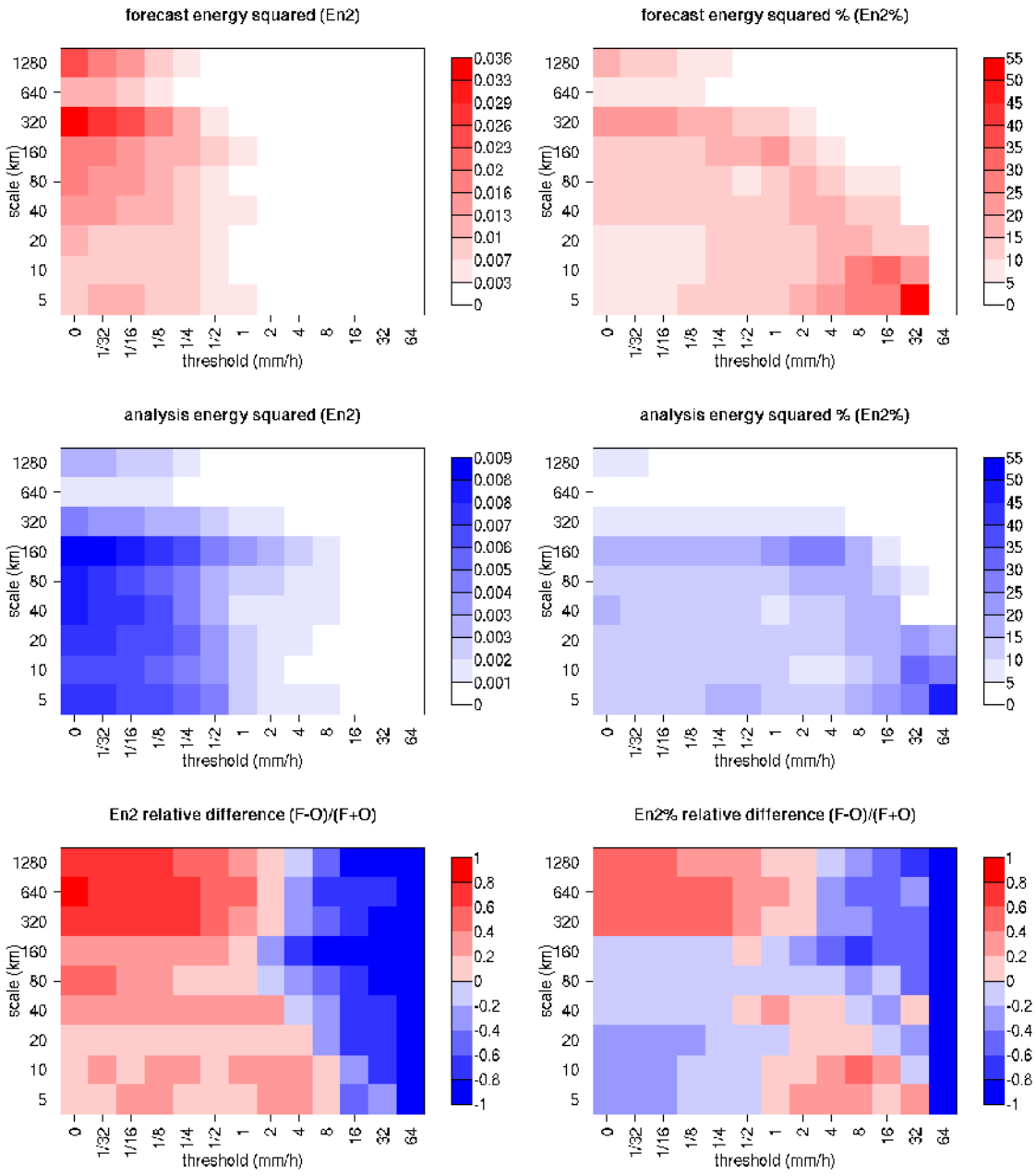

Figure 15.6: Energy squared and energy squared percentages, for each threshold and sale, for the NIMROD forecast and analysis, and forecast and analysis En2 and En2% relative differences.

The En2 bias for each threshold and scale is assessed by the En2 relative difference, equal to the difference between forecast and observed squared energies normalized by their sum: $En2(F) - En2(O)]/[En2(F) + En2(O)]$. Since defined in such a fashion, the En2 relative difference accounts for the difference between forecast and observation squared energies relative to their magnitude, and it is sensitive therefore to the ratio of the forecast and observed squared energies. The En2 relative difference ranges between -1 and 1, positive values indicate over-forecast and negative values indicate under-forecast. For the NIMROD case

illustrated the forecast exhibits over-forecast for small thresholds, quite pronounced on the large scales, and under-forecast for high thresholds.

As for the MSE, the sum of the energy of the scale components is equal to the energy of the original binary field: $En2(t) = j\ En2(t,j)$. Therefore, the percentage that the En2 for each scale contributes the total En2 may be computed: for a given threshold, **t**, $En2\%(t,j) = En2(t,j)/En2(t)$. Usually, for precipitation fields, low thresholds exhibit most of the energy percentage on large scales (and less percentage on the small scales), since low thresholds are associated with large scale features, such as fronts. On the other hand, for higher thresholds, the energy percentage is usually larger on small scales, since intense events are associated with small scales features, such as convective cells or showers. The comparison of the forecast and observation squared energy percentages provides feedback on how the events are distributed across the scales, and enables the comparison of forecast and observation scale structure.

For the NIMROD case illustrated, the scale structure is assessed again by the relative difference, but calculated of the squared energy percentages. For small thresholds the forecast overestimates the number of large scale events and underestimates the number of small scale events, in proportion to the total number of events. On the other hand, for larger thresholds the forecast underestimates the number of large scale events and overestimates the number of small scale events, again in proportion to the total number of events. Overall it appears that the forecast overestimates the percentage of events associated with high occurrence, and underestimates the percentage of events associated with low occurrence. The En2% for the 64 mm/h thresholds is homogeneously underestimated for all the scales, since the forecast does not have any event exceeding this threshold.

Note that the energy squared of the observation binary field is identical to the sample climatology $Br = (a+c)/n$. Similarly, the energy squared of the forecast binary field is equal to $(a+b)/n$. The ratio of the squared energies of the forecast and observation binary fields is equal to the $FBI = (a+b)/(a+c)$, for the contingency table (Table 15.1) obtained from the original forecast and observation fields by thresholding with the same threshold used to obtain the binary forecast and observation fields.

15.2.2 The Spatial Domain Constraints

The Intensity-Scale technique is constrained by the fact that orthogonal wavelets (discrete wavelet transforms) are usually performed dyadic domains, square domains of $2^n \times 2^n$ grid-points. The Wavelet-Stat tool handles this issue based on settings in the configuration file by defining tiles of dimensions $2^n \times 2^n$ over the input domain in the following ways:

1. User-Defined Tiling: The user may define one or more tiles of size $2^n \times 2^n$ over their domain to be applied. This is done by selecting the grid coordinates for the lower-left corner of the tile(s) and the tile dimension to be used. If the user specifies more than one tile, the Intensity-Scale method will be applied to each tile separately. At the end, the results will automatically be aggregated across all the tiles and written out with the results for each of the individual tiles. Users are encouraged to select tiles which consist entirely of valid data.

2. Automated Tiling: This tiling method is essentially the same as the user-defined tiling method listed above except that the tool automatically selects the location and size of the tile(s) to be applied. It figures out the maximum tile of dimension $2^n \times 2^n$ that fits within the domain and places the tile at the center of the domain. For domains that are very elongated in one direction, it defines as many of these tiles as possible that fit within the domain.

3. Padding: If the domain size is only slightly smaller than $2^n \times 2^n$, for certain variables (e.g. precipitation), it is advisable to expand the domain out to $2^n \times 2^n$ grid-points by adding extra rows and/or columns of fill data. For precipitation variables, a fill value of zero is used. For continuous variables, such as temperature, the fill value is defined as the mean of the valid data in the rest of the field. A drawback to the padding method is the introduction of artificial data into the original field. Padding should only be used when a very small number of rows and/or columns need to be added.

15.2.3 Aggregation of Statistics on Multiple Cases

The Stat-Analysis tool aggregates the intensity scale technique results. Since the results are scale-dependent, it is sensible to aggregate results from multiple model runs (e.g. daily runs for a season) on the same spatial domain, so that the scale components for each singular case will be the same number, and the domain, if not a square domain of $2^n \times 2^n$ grid-points, will be treated in the same fashion. Similarly, the intensity thresholds for each run should all be the same.

The MSE and forecast and observation squared energy for each scale and thresholds are aggregated simply with a weighted average, where weights are proportional to the number of grid-points used in each single run to evaluate the statistics. If the same domain is always used (and it should) the weights result all the same, and the weighted averaging is a simple mean. For each threshold, the aggregated Br is equal to the aggregated squared energy of the binary observation field, and the aggregated FBI is obtained as the ratio of the aggregated squared energies of the forecast and observation binary fields. From aggregated Br and FBI, the MSErandom for the aggregated runs can be evaluated using the same formula as for the single run. Finally, the Intensity-Scale Skill Score is evaluated by using the aggregated statistics within the same formula used for the single case.

15.3 Practical Information

The following sections describe the usage statement, required arguments and optional arguments for the Stat-Analysis tool.

15.3.1 wavelet_stat Usage

The usage statement for the Wavelet-Stat tool is shown below:

```
Usage: wavelet_stat
       fcst_file
       obs_file
       config_file
       [-outdir path]
       [-log file]
       [-v level]
       [-compress level]
```

wavelet_stat has three required arguments and accepts several optional ones.

15.3.1.1 Required Arguments for wavelet_stat

1. The **fcst_file** argument is the gridded file containing the model data to be verified.
2. The **obs_file** argument is the gridded file containing the observations to be used.
3. The **config_file** argument is the configuration file to be used. The contents of the configuration file are discussed below.

15.3.1.2 Optional Arguments for wavelet_stat

4. The **-outdir path** indicates the directory where output files should be written.
5. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
6. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.
7. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the wavelet_stat calling sequence is listed below:

```
wavelet_stat \
sample_fcst.grb \
sample_obs.grb \
WaveletStatConfig
```

In the example, the Wavelet-Stat tool will verify the model data in the **sample_fcst.grb** GRIB file using the observations in the **sample_obs.grb** GRIB file applying the configuration options specified in the **WaveletStatConfig** file.

15.3.2 wavelet_stat Configuration File

The default configuration file for the Wavelet-Stat tool, **WaveletStatConfig_default**, can be found in the installed *share/met/config* directory. Another version of the configuration file is provided in *scripts/config*. We recommend that users make a copy of the default (or other) configuration file prior to modifying it. The contents are described in more detail below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
model             = "FCST";
desc              = "NA";
obtype            = "ANALYS";
fcst              = { ... }
obs               = { ... }
regrid            = { ... }
mask_missing_flag = NONE;
met_data_dir      = "MET_BASE";
ps_plot_flag      = TRUE;
fcst_raw_plot     = { color_table = "MET_BASE/colortables/met_default.ctable";
                      plot_min = 0.0; plot_max = 0.0; }
obs_raw_plot      = { ... }
wvlt_plot         = { ... }
output_prefix     = "";
version           = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
// Empty list of thresholds
cat_thresh = [];

// Or explicitly set the NA threshold type
cat_thresh = [>0.0, >=5.0, NA];
```

The **cat_thresh** option defines an array of thresholds for each field defined in the fcst and obs dictionaries. The number of forecast and observation categorical thresholds must match. If set to an empty list, the thresholds will not be applied (no binary masking) and all the raw grid-point values will be used for downstream statistics.

If the array of thresholds is an empty list, the application will set the threshold to NA internally and skip applying the thresholds. If the threshold is set to NA explicitly in the list, the application will also skip applying the threshold.

Since the application has the ability to loop through multiple thresholds (for multiple fields), a user can include NA in the list of thresholds to produce statistics for the raw data values for the given field.

```
grid_decomp_flag = AUTO;

tile = {
   width    = 0;
   location = [ { x_ll = 0; y_ll = 0; } ];
}
```

The **grid_decomp_flag** variable specifies how tiling should be performed:

- **AUTO** indicates that the automated-tiling should be done.

- **TILE** indicates that the user-defined tiles should be applied.
- **PAD** indicated that the data should be padded out to the nearest dimension of $2^n \times 2^n$

The **width** and **location** variables allow users to manually define the tiles of dimension they would like to apply. The x_ll and y_ll variables specify the location of one or more lower-left tile grid (x, y) points.

```
wavelet = {
   type   = HAAR;
   member = 2;
}
```

The **wavelet_flag** and **wavelet_k** variables specify the type and shape of the wavelet to be used for the scale decomposition. The *Casati et al. (2004)* (page 493) method uses a Haar wavelet which is a good choice for discontinuous fields like precipitation. However, users may choose to apply any wavelet family/shape that is available in the GNU Scientific Library. Values for the **wavelet_flag** variable, and associated choices for k, are described below:

- **HAAR** for the Haar wavelet (member = 2).
- **HAAR_CNTR** for the Centered-Haar wavelet (member = 2).
- **DAUB** for the Daubechies wavelet (member = 4, 6, 8, 10, 12, 14, 16, 18, 20).
- **DAUB_CNTR** for the Centered-Daubechies wavelet (member = 4, 6, 8, 10, 12, 14, 16, 18, 20).
- **BSPLINE** for the Bspline wavelet (member = 103, 105, 202, 204, 206, 208, 301, 303, 305, 307, 309).
- **BSPLINE_CNTR** for the Centered-Bspline wavelet (member = 103, 105, 202, 204, 206, 208, 301, 303, 305, 307, 309).

```
output_flag = {
   isc = BOTH;
}
```

The **output_flag** array controls the type of output that the Wavelet-Stat tool generates. This flag is set similarly to the output flag of the other MET tools, with possible values of NONE, STAT, and BOTH. The ISC line type is the only one available for Intensity-Scale STAT lines.

```
nc_pairs_flag = {
   latlon = TRUE;
   raw    = TRUE;
}
```

The nc_pairs_flag is described in Section 13.3.2

15.3.3 wavelet_stat Output

wavelet_stat produces output in STAT and, optionally, ASCII and NetCDF and PostScript formats. The ASCII output duplicates the STAT output but has the data organized by line type. While the Wavelet-Stat tool currently only outputs one STAT line type, additional line types may be added in future releases. The output files are written to the default output directory or the directory specified by the -outdir command line option.

The output STAT file is named using the following naming convention:

wavelet_stat_PREFIX_HHMMSSL_YYYYMMDD_HHMMSSV.stat where PREFIX indicates the user-defined output prefix, HHMMSS indicates the forecast lead time, and YYYYMMDD_HHMMSS indicates the forecast valid time.

The output ASCII files are named similarly:

wavelet_stat_PREFIX_HHMMSSL_YYYYMMDD_HHMMSSV_TYPE.txt where TYPE is isc to indicate that this is an intensity-scale line type.

The format of the STAT and ASCII output of the Wavelet-Stat tool is similar to the format of the STAT and ASCII output of the Point-Stat tool. Please refer to the tables in Section 11.3.3 for a description of the common output for STAT files types. The information contained in the STAT and isc files are identical. However, for consistency with the STAT files produced by other tools, the STAT file will only have names for the header columns. The isc file contains names for all columns. The format of the ISC line type is explained in the following table.

Table 15.2: Header information for Wavelet-Stat STAT output

Column Number	Header Column Name	Description	Data Type
1	VERSION	Version number	String
2	MODEL	User-provided text string designating model name	String
3	DESC	User-provided text string describing the verification task	String
4	FCST_LEAD	Forecast lead time in HHMMSS format	Time String
5	FCST_VALID_BEG	Forecast valid start time in YYYYMMDD_HHMMSS format	Datetime String
6	FCST_VALID_END	Forecast valid end time in YYYYMMDD_HHMMSS format	Datetime String
7	OBS_LEAD	Observation lead time in HHMMSS format	Time String
8	OBS_VALID_BEG	Observation valid start time in YYYYMMDD_HHMMSS format	Datetime String
9	OBS_VALID_END	Observation valid end time in YYYYMMDD_HHMMSS format	Datetime String
10	FCST_VAR	Model variable	String
11	FCST_UNITS	Units for model variable	String
12	FCST_LEV	Selected vertical level for forecast	String
13	OBS_VAR	Observation variable	String
14	OBS_UNITS	Units for observation variable	String
15	OBS_LEV	Selected vertical level for observations	String
16	OBTYPE	User-provided text string designating the observation type	String
17	VX_MASK	Verifying masking region indicating the masking grid or polyline region applied	String
18	INTERP_MTHD	NA in Wavelet-Stat	String
19	INTERP_PNTS	NA in Wavelet-Stat	Integer
20	FCST_THRESH	The threshold applied to the forecast	Threshold String
21	OBS_THRESH	The threshold applied to the observations	Threshold String
22	COV_THRESH	NA in Wavelet-Stat	Threshold String
23	ALPHA	NA in Wavelet-Stat	Double
24	LINE_TYPE	See table below.	String

Table 15.3: Format information for the ISC (Intensity-Scale) output line type.

Column Number	ISC Column Name	Description	Data Type
24	ISC	Intensity-Scale line type	String
25	TOTAL	The number of grid points (forecast locations) used	Integer
26	TILE_DIM	The dimensions of the tile	Integer
27	TILE_XLL	Horizontal coordinate of the lower left corner of the tile	Integer
28	TILE_YLL	Vertical coordinate of the lower left corner of the tile	Integer
29	NSCALE	Total number of scales used in decomposition	Integer
30	ISCALE	The scale at which all information following applies	Integer
31	MSE	Mean squared error for this scale	Double
32	ISC	The intensity scale skill score	Double
33	FENERGY	Forecast energy squared for this scale	Double
34	OENERGY	Observed energy squared for this scale	Double
35	BASER	The base rate (not scale dependent)	Double
36	FBIAS	The frequency bias	Double

The **Wavelet-Stat** tool creates a NetCDF output file containing the raw and decomposed values for the forecast, observation, and difference fields for each combination of variable and threshold value.

The dimensions and variables included in the wavelet_stat NetCDF files are described in Tables Table 15.4 and Table 15.5.

Table 15.4: Dimensions defined in NetCDF output.

NetCDF Dimension	Description
x	Dimension of the tile which equals 2^n
y	Dimension of the tile which equals 2^n
scale	Dimension for the number of scales. This is set to **n+2**, where 2^n is the tile dimension. The 2 extra scales are for the binary image and the wavelet averaged over the whole tile.
tile	Dimension for the number of tiles used

Table 15.5: Variables defined in NetCDF output.

NetCDF Variable	Dimension	Description	Data Type
FCST_FIELD_LEVEL_RAW	tile, x, y	Raw values for the forecast field specified by "FIELD_LEVEL"	Float
OBS_FIELD_LEVEL_RAW	tile, x, y	Raw values for the observation field specified by "FIELD_LEVEL"	Float
DIFF_FIELD_LEVEL_RAW	tile, x, y	Raw values for the difference field (**f-o**) specified by "FIELD_LEVEL"	Float
FCST_FIELD_LEVEL_THRESH	thresh, scale, x, y	Wavelet scale-decomposition of the forecast field specified by "FIELD_LEVEL_THRESH"	Float
OBS_FIELD_LEVEL_THRESH	thresh, scale, x, y	Wavelet scale-decomposition of the observation field specified by "FIELD_LEVEL_THRESH"	Float

Lastly, the **Wavelet-Stat** tool creates a PostScript plot summarizing the scale-decomposition approach used in the verification. The PostScript plot is generated using internal libraries and does not depend on an external plotting package. The generation of this PostScript output can be disabled using the **ps_plot_flag** configuration file option.

The PostScript plot begins with one summary page illustrating the tiling method that was applied to the domain. The remaining pages depict the Intensity-Scale method that was applied. For each combination of field, tile, and threshold, the binary difference field (**f-o**) is plotted followed by the difference field for each decomposed scale. Underneath each difference plot, the statistics applicable to that scale are listed. Examples of the PostScript plots can be obtained by running the example cases provided with the MET tarball.

Chapter 16

GSI Tools

Gridpoint Statistical Interpolation (GSI) diagnostic files are binary files written out from the data assimilation code before the first and after each outer loop. The files contain useful information about how a single observation was used in the analysis by providing details such as the innovation (O-B), observation values, observation error, adjusted observation error, and quality control information.

For more detail on generating GSI diagnostic files and their contents, see the GSI User's Guide.

When MET reads GSI diagnostic files, the innovation (O-B; generated prior to the first outer loop) or analysis increment (O-A; generated after the final outer loop) is split into separate values for the observation (OBS) and the forecast (FCST), where the forecast value corresponds to the background (O-B) or analysis (O-A).

MET includes two tools for processing GSI diagnostic files. The GSID2MPR tool reformats individual GSI diagnostic files into the MET matched pair (MPR) format, similar to the output of the Point-Stat tool. The GSIDENS2ORANK tool processes an ensemble of GSI diagnostic files and reformats them into the MET observation rank (ORANK) line type, similar to the output of the Ensemble-Stat tool. The output of both tools may be passed to the Stat-Analysis tool to compute a wide variety of continuous, categorical, and ensemble statistics.

16.1 GSID2MPR Tool

This section describes how to run the GSID2MPR tool. The GSID2MPR tool reformats one or more GSI diagnostic files into an ASCII matched pair (MPR) format, similar to the MPR output of the Point-Stat tool. The output MPR data may be passed to the Stat-Analysis tool to compute a wide variety of continuous or categorical statistics.

16.1.1 gsid2mpr Usage

The usage statement for the GSID2MPR tool is shown below:

```
Usage: gsid2mpr
       gsi_file_1 [gsi_file_2 ... gsi_file_n]
       [-swap]
       [-no_check_dup]
       [-channel n]
       [-set_hdr col_name value]
       [-suffix string]
       [-outdir path]
       [-log file]
       [-v level]
```

gsid2mpr has one required argument and accepts several optional ones.

16.1.1.1 Required Arguments for gsid2mpr

1. The **gsi_file_1 [gsi_file2 ... gsi_file_n]** argument indicates the GSI diagnostic files (conventional or radiance) to be reformatted.

16.1.1.2 Optional Arguments for gsid2mpr

2. The **-swap** option switches the endianness when reading the input binary files.
3. The **-no_check_dup** option disables the checking for duplicate matched pairs which slows down the tool considerably for large files.
4. The **-channel n** option overrides the default processing of all radiance channels with the values of a comma-separated list.
5. The **-set_hdr col_name value** option specifies what should be written to the output header columns.
6. The **-suffix string** option overrides the default output filename suffix (.stat).
7. The **-outdir path** option overrides the default output directory (./).
8. The **-log file** option outputs log messages to the specified file.
9. The **-v level** option overrides the default level of logging (2).

An example of the gsid2mpr calling sequence is shown below:

```
gsid2mpr diag_conv_ges.mem001 \
-set_hdr MODEL GSI_MEM001 \
-outdir out
```

In this example, the GSID2MPR tool will process a single input file named **diag_conv_ges.mem001** file, set the output **MODEL** header column to **GSI_MEM001**, and write output to the **out** directory. The output file is named the same as the input file but a **.stat** suffix is added to indicate its format.

16.1.2 gsid2mpr Output

The GSID2MPR tool performs a simple reformatting step and thus requires no configuration file. It can read both conventional and radiance binary GSI diagnostic files. Support for additional GSI diagnostic file type may be added in future releases. Conventional files are determined by the presence of the string **conv** in the filename. Files that are not conventional are assumed to contain radiance data. Multiple files of either type may be passed in a single call to the GSID2MPR tool. For each input file, an output file will be generated containing the corresponding matched pair data.

The GSID2MPR tool writes the same set of MPR output columns for the conventional and radiance data types. However, it also writes additional columns at the end of the MPR line which depend on the input file type. Those additional columns are described in the following tables.

Table 16.1: Format information for GSI Diagnostic Conventional MPR (Matched Pair) output line type.

Column Number	Column Name	Description	Data Type
1-37		Standard MPR columns described in Table 11.18	Varies
38	OBS_PRS	Model pressure value at the observation height (hPa)	Double
39	OBS_ERR_IN	PrepBUFR inverse observation error	Double
40	OBS_ERR_ADJ	read_PrepBUFR inverse observation error	Double
41	OBS_ERR_FIN	Final inverse observation error	Double
42	PREP_USE	read_PrepBUFR usage	Integer
43	ANLY_USE	Analysis usage (1 for yes, -1 for no)	Integer
44	SETUP_QC	Setup quality control	Integer
45	QC_WGHT	Non-linear quality control relative weight	Double

Table 16.2: Format information for GSI Diagnostic Radiance MPR (Matched Pair) output line type.

Column Number	Column Name	Description
1-37		Standard MPR columns described in Table 11.18
38	CHAN_USE	Channel used (1 for yes, -1 for no)
39	SCAN_POS	Sensor scan position
40	SAT_ZNTH	Satellite zenith angle (degrees)
41	SAT_AZMTH	Satellite azimuth angle (degrees)
42	SUN_ZNTH	Solar zenith angle (degrees)
43	SUN_AZMTH	Solar azimuth angle (degrees)
44	SUN_GLNT	Sun glint angle (degrees)
45	FRAC_WTR	Fractional coverage by water
46	FRAC_LND	Fractional coverage by land
47	FRAC_ICE	Fractional coverage by ice
48	FRAC_SNW	Fractional coverage by snow
49	SFC_TWTR	Surface temperature over water (K)
50	SFC_TLND	Surface temperature over land (K)
51	SFC_TICE	Surface temperature over ice (K)
52	SFC_TSNW	Surface temperature over snow (K)

Table 16.2 – continued from previous page

Column Number	Column Name	Description
53	TSOIL	Soil temperature (K)
54	SOILM	Soil moisture
55	LAND_TYPE	Surface land type
56	FRAC_VEG	Vegetation fraction
57	SNW_DPTH	Snow depth
58	SFC_WIND	Surface wind speed (m/s)
59	FRAC_CLD CLD_LWC	Cloud fraction (%) Cloud liquid water (kg/m**2) (microwave only)
60	CTOP_PRS TC_PWAT	Cloud top pressure (hPa) Total column precip. water (km/m**2) (microw
61	TFND	Foundation temperature: Tr
62	TWARM	Diurnal warming: d(Tw) at depth zob
63	TCOOL	Sub-layer cooling: d(Tc) at depth zob
64	TZFND	d(Tz)/d(Tr)
65	OBS_ERR	Inverse observation error
66	FCST_NOBC	Brightness temperature with no bias correction (K)
67	SFC_EMIS	Surface emissivity
68	STABILITY	Stability index
69	PRS_MAX_WGT	Pressure of the maximum weighing function

The gsid2mpr output may be passed to the Stat-Analysis tool to derive additional statistics. In particular, users should consider running the **aggregate_stat** job type to read MPR lines and compute partial sums (SL1L2), continuous statistics (CNT), contingency table counts (CTC), or contingency table statistics (CTS). Stat-Analysis has been enhanced to parse any extra columns found at the end of the input lines. Users can filter the values in those extra columns using the **-column_thresh**, **-column_str**, and **-column_str_exc** job command options.

An example of the Stat-Analysis calling sequence is shown below:

```
stat_analysis -lookin diag_conv_ges.mem001.stat \
-job aggregate_stat -line_type MPR -out_line_type CNT \
-fcst_var t -column_thresh ANLY_USE eq1
```

In this example, the Stat-Analysis tool will read MPR lines from the input file named **diag_conv_ges.mem001.stat**, retain only those lines where the **FCST_VAR** column indicates temperature (**t**) and where the **ANLY_USE** column has a value of 1.0, and derive continuous statistics.

16.2 GSIDENS2ORANK Tool

This section describes how to run the GSIDENS2ORANK tool. The GSIDENS2ORANK tool processes an ensemble of GSI diagnostic files and reformats them into the MET observation rank (ORANK) line type, similar to the output of the Ensemble-Stat tool. The ORANK line type contains ensemble matched pair information and is analogous to the MPR line type for a deterministic model. The output ORANK data may be passed to the Stat-Analysis tool to compute ensemble statistics.

16.2.1 gsidens2orank Usage

The usage statement for the GSIDENS2ORANK tool is shown below:

```
Usage: gsidens2orank
       file_1 ... file_n | file_list
       -out path
       [-ens_mean path]
       [-swap]
       [-rng_name str]
       [-rng_seed str]
       [-set_hdr col_name value]
       [-log file]
       [-v level]
```

gsidens2orank has two required arguments and accepts several optional ones.

16.2.1.1 Required Arguments for gsidens2orank

1. The **file_1 ... file_n | file_list** argument specifies the ensemble binary GSI diagnostic files to be reformatted or ASCII file list of file names to be used, as described in Section 4.1.1.
2. The **-out path** argument specifies the name of the output **.stat** file.

16.2.1.2 Optional Arguments for gsidens2orank

3. The **-ens_mean path** option is the ensemble mean binary GSI diagnostic file.
4. The **-swap** option switches the endianness when reading the input binary files.
5. The **-channel n** option overrides the default processing of all radiance channels with a comma-separated list.
6. The **-rng_name str** option overrides the default random number generator name (mt19937).
7. The **-rng_seed str** option overrides the default random number generator seed.
8. The **-set_hdr col_name value** option specifies what should be written to the output header columns.
9. The **-log file** option outputs log messages to the specified file.
10. The **-v level** option overrides the default level of logging (2).

An example of the gsidens2orank calling sequence is shown below:

```
gsidens2orank diag_conv_ges.mem* \
-ens_mean diag_conv_ges.ensmean \
-out diag_conv_ges_ens_mean_orank.txt
```

In this example, the GSIDENS2ORANK tool will process all of the ensemble members whose file name **matches diag_conv_ges.mem*,** write output to the file named **diag_conv_ges_ens_mean_orank.txt**, and

populate the output **ENS_MEAN** column with the values found in the **diag_conv_ges.ensmean** file rather than computing the ensemble mean values from the ensemble members on the fly.

16.2.2 gsidens2orank Output

The GSIDENS2ORANK tool performs a simple reformatting step and thus requires no configuration file. The multiple files passed to it are interpreted as members of the same ensemble. Therefore, each call to the tool processes exactly one ensemble. All input ensemble GSI diagnostic files must be of the same type. Mixing conventional and radiance files together will result in a runtime error. The GSIDENS2ORANK tool processes each ensemble member and keeps track of the observations it encounters. It constructs a list of the ensemble values corresponding to each observation and writes an output ORANK line listing the observation value, its rank, and all the ensemble values. The random number generator is used by the GSIDENS2ORANK tool to randomly assign a rank value in the case of ties.

The GSID2MPR tool writes the same set of ORANK output columns for the conventional and radiance data types. However, it also writes additional columns at the end of the ORANK line which depend on the input file type. The extra columns are limited to quantities which remain constant over all the ensemble members and are therefore largely a subset of the extra columns written by the GSID2MPR tool. Those additional columns are described in the following tables.

Table 16.3: Format information for GSI Diagnostic Conventional ORANK (Observation Rank) output line type.

Column Number	Column Name	Description	Data Type
1-?		Standard ORANK columns described in Table 14.7	Varies
Last-2	N_USE	Number of members with ANLY_USE = 1	Integer
Last-1	PREP_USE	read_PrepBUFR usage	Integer
Last	SETUP_QC	Setup quality control	Integer

Table 16.4: Format information for GSI Diagnostic Radiance ORANK (Observation Rank) output line type.

Column Number	Column Name	Description	Data Type
1-?		Standard ORANK columns described in Table 14.7	Varies
Last-24	N_USE	Number of members with OBS_QC = 0	Integer
Last-23	CHAN_USE	Channel used (1 for yes, -1 for no)	Integer
Last-22	SCAN_POS	Sensor scan position	Integer
Last-21	SAT_ZNTH	Satellite zenith angle (degrees)	Double
Last-20	SAT_AZMTH	Satellite azimuth angle (degrees)	Double
Last-19	SUN_ZNTH	Solar zenith angle (degrees)	Double
Last-18	SUN_AZMTH	Solar azimuth angle (degrees)	Double
Last-17	SUN_GLNT	Sun glint angle (degrees)	Double
Last-16	FRAC_WTR	Fractional coverage by water	Double
Last-15	FRAC_LND	Fractional coverage by land	Double
Last-14	FRAC_ICE	Fractional coverage by ice	Double
Last-13	FRAC_SNW	Fractional coverage by snow	Double
Last-12	SFC_TWTR	Surface temperature over water (K)	Double
Last-11	SFC_TLND	Surface temperature over land (K)	Double
Last-10	SFC_TICE	Surface temperature over ice (K)	Double
Last-9	SFC_TSNW	Surface temperature over snow (K)	Double
Last-8	TSOIL	Soil temperature (K)	Double
Last-7	SOILM	Soil moisture	Double
Last-6	LAND_TYPE	Surface land type	Integer
Last-5	FRAC_VEG	Vegetation fraction	Double
Last-4	SNW_DPTH	Snow depth	Double
Last-3	TFND	Foundation temperature: Tr	Double
Last-2	TWARM	Diurnal warming: d(Tw) at depth zob	Double
Last-1	TCOOL	Sub-layer cooling: d(Tc) at depth zob	Double
Last	TZFND	d(Tz)/d(Tr)	Double

The gsidens2orank output may be passed to the Stat-Analysis tool to derive additional statistics. In particular, users should consider running the **aggregate_stat** job type to read ORANK lines and ranked histograms (RHIST), probability integral transform histograms (PHIST), and spread-skill variance output (SSVAR). Stat-Analysis has been enhanced to parse any extra columns found at the end of the input lines. Users can filter the values in those extra columns using the **-column_thresh**, **-column_str**, and **-column_str_exc** job command options.

An example of the Stat-Analysis calling sequence is shown below:

```
stat_analysis -lookin diag_conv_ges_ens_mean_orank.txt \
-job aggregate_stat -line_type ORANK -out_line_type RHIST \
-by fcst_var -column_thresh N_USE eq20
```

In this example, the Stat-Analysis tool will read ORANK lines from **diag_conv_ges_ens_mean_orank.txt**, retain only those lines where the **N_USE** column indicates that all 20 ensemble members were used, and write ranked histogram (RHIST) output lines for each unique value of encountered in the **FCST_VAR** column.

Chapter 17

Stat-Analysis Tool

17.1 Introduction

The Stat-Analysis tool ties together results from the Point-Stat, Grid-Stat, Ensemble-Stat, Wavelet-Stat, and TC-Gen tools by providing summary statistical information and a way to filter their STAT output files. It processes the STAT output created by the other MET tools in a variety of ways which are described in this section.

MET version 9.0 adds support for the passing matched pair data (MPR) into Stat-Analysis using a Python script with the "-lookin python ..." option. An example of running Stat-Analysis with Python embedding can be found in Appendix F, Section 38.

17.2 Scientific and Statistical Aspects

The Stat-Analysis tool can perform a variety of analyses, and each type of analysis is called a "job". The job types include the ability to (i) aggregate results over a user-specified time; (ii) stratify statistics based on time of day, model initialization time, lead-time, model run identifier, output filename, or wavelet decomposition scale; and (iii) compute specific verification indices such as the GO Index[1] and wind direction statistics. Future functionality may include information about time-trends and/or calculations based on climatology (e.g., anomaly correlation). This section summarizes the capabilities of the supported Stat-Analysis jobs.

[1] The GO Index is a summary measure for NWP models that is used by the US Air Force. It combines verification statistics for several forecast variables and lead times.

17.2.1 Filter STAT Lines

The Stat-Analysis "filter" job simply filters out specific STAT lines based on user-specified search criteria. All of the STAT lines that are retained from one or many files are written to a single output file. The output file for filtered STAT lines must be specified using the **-dump_row** job command option.

17.2.2 Summary Statistics for Columns

The Stat-Analysis "summary" job produces summary information for columns of data. After the user specifies the column(s) of interest and any other relevant search criteria, summary information is produced from values in those column(s) of data. The summary statistics produced are: mean, standard deviation, minimum, maximum, the 10th, 25th, 50th, 75th, and 90th percentiles, the interquartile range, the range, and both weighted and unweighted means using the logic prescribed by the World Meteorological Organization (WMO).

Confidence intervals are computed for the mean and standard deviation of the column of data. For the mean, the confidence interval is computed two ways - based on an assumption of normality and also using the bootstrap method. For the standard deviation, the confidence interval is computed using the bootstrap method. In this application of the bootstrap method, the values in the column of data being summarized are resampled, and for each replicated sample, the mean and standard deviation are computed.

The columns to be summarized can be specified in one of two ways. Use the **-line_type** option exactly once to specify a single input line type and use the **-column** option one or more times to select the columns of data to be summarized. Alternatively, use the **-column** option one or more times formatting the entries as **LINE_TYPE:COLUMN**. For example, the RMSE column from the CNT line type can be selected using **-line_type CNT -column RMSE** or using **-column CNT:RMSE**. With the second option, columns from multiple input line types may be selected. For example, **-column CNT:RMSE,CNT:MAE,CTS:CSI** select two CNT columns and one CTS column.

The WMO mean values are computed in one of three ways, as determined by the configuration file settings for **wmo_sqrt_stats** and **wmo_fisher_stats**. The statistics listed in the first option are square roots. When computing WMO means, the input values are first squared, then averaged, and the square root of the average value is reported. The statistics listed in the second option are correlations to which the Fisher transformation is applied. For any statistic not listed, the WMO mean is computed as a simple arithmetic mean. The **WMO_TYPE** output column indicates the method applied (**SQRT, FISHER**, or **MEAN**). The **WMO_MEAN** and **WMO_WEIGHTED_MEAN** columns contain the unweighted and weighted means, respectively. The value listed in the **TOTAL** column of each input line is used as the weight.

The **-derive** job command option can be used to perform the derivation of statistics on the fly from input partial sums and contingency table counts. When enabled, SL1L2 and SAL1L2 input lines are converted to CNT statistics, VL1L2 input lines are converted to VCNT statistics, and CTC lines are converted to CTS statistics. Users should take care with this option. If the data passed to this job contains both partial sums and derived statistics, using the **-derive** option will effectively cause the statistics to be double counted. Use the **-line_type** job command option to filter the data passed to Stat-Analysis jobs.

17.2.3 Aggregated Values from Multiple STAT Lines

The Stat-Analysis "aggregate" job aggregates values from multiple STAT lines of the same type. The user may specify the specific line type of interest and any other relevant search criteria. The Stat-Analysis tool then creates sums of each of the values in all lines matching the search criteria. The aggregated data are output as the same line type as the user specified. The STAT line types which may be aggregated in this way are the contingency table (FHO, CTC, PCT, MCTC, NBRCTC), partial sums (SL1L2, SAL1L2, VL1L2, and VAL1L2), and other (ISC, ECNT, RPS, RHIST, PHIST, RELP, NBRCNT, SSVAR, GRAD, and SEEPS) line types.

17.2.4 Aggregate STAT Lines and Produce Aggregated Statistics

The Stat-Analysis "aggregate-stat" job aggregates multiple STAT lines of the same type together and produces relevant statistics from the aggregated line. This may be done in the same manner listed above in Section 17.2.3. However, rather than writing out the aggregated STAT line itself, the relevant statistics generated from that aggregated line are provided in the output. Specifically, if a contingency table line type (FHO, CTC, PCT, MCTC, or NBRCTC) has been aggregated, contingency table statistics (CTS, ECLV, PSTD, MCTS, or NBRCTS) line types can be computed. If a partial sums line type (SL1L2 or SAL1L2) has been aggregated, the continuous statistics (CNT) line type can be computed. If a vector partial sums line type (VL1L2 or VAL1L2) has been aggregated, the vector continuous statistics (VCNT) line type can be computed. For ensembles, the ORANK line type can be accumulated into ECNT, RPS, RHIST, PHIST, RELP, or SSVAR output. If a SEEPS matched pair line type (SEEPS_MPR) has been aggregated, the aggregated SEEPS line type (SEEPS) can be computed. If the matched pair line type (MPR) has been aggregated, may output line types (FHO, CTC, CTS, CNT, MCTC, MCTS, SL1L2, SAL1L2, VL1L2, VCNT, WDIR, PCT, PSTD, PJC, PRC, or ECLV) can be computed. Multiple output line types may be specified for each "aggregate-stat" job, as long as each output is derivable from the input.

When aggregating the matched pair line type (MPR), additional required job command options are determined by the requested output line type(s). For example, the "-out_thresh" (or "-out_fcst_thresh" and "-out_obs_thresh" options) are required to compute contingnecy table counts (FHO, CTC) or statistics (CTS). Those same job command options can also specify filtering thresholds when computing continuous partial sums (SL1L2, SAL1L2) or statistics (CNT). Output is written for each threshold specified.

When aggregating the matched pair line type (MPR) and computing an output contingency table statistics (CTS) or continuous statistics (CNT) line type, the bootstrapping method can be applied to compute confidence intervals. The bootstrapping method is applied here in the same way that it is applied in the statistics tools. For a set of n matched forecast-observation pairs, the matched pairs are resampled with replacement many times. For each replicated sample, the corresponding statistics are computed. The confidence intervals are derived from the statistics computed for each replicated sample.

17.2.5 Skill Score Index

The Stat-Analysis "ss_index", "go_index", and "cbs_index" jobs calculate skill score indices by weighting scores for meteorological fields at different levels and lead times. Pre-defined configuration files are provided for the GO Index and CBS Index which are special cases of the highly configurable skill score index job.

In general, a skill score index is computed over several terms and the number and definition of those terms is configurable. It is computed by aggregating the output from earlier runs of the Point-Stat and/or Grid-Stat tools over one or more cases. When configuring a skill score index job, the following requirements apply:

1. Exactly two models names must be chosen. The first is interpreted as the forecast model and the second is the reference model, against which the performance of the forecast should be measured. Specify this with the "model" configuration file entry or using the "-model" job command option.
2. The forecast variable name, level, lead time, line type, column, and weight options must be specified. If the value remains constant for all the terms, set it to an array of length one. If the value changes for at least one term, specify an array entry for each term. Specify these with the "fcst_var", "fcst_lev", "lead_time", "line_type", "column", and "weight" configuration file entries, respectively, or use the corresponding job command options.
3. While these line types are required, additional options may be provided for each term, including the observation type ("obtype"), verification region ("vx_mask"), and interpolation method ("interp_mthd"). Specify each as single value or provide a value for each term.
4. Only the SL1L2 and CTC input line types are supported, and the input Point-Stat and/or Grid-Stat output must contain these line types.
5. For the SL1L2 line type, set the "column" entry to the CNT output column that contains the statistic of interest (e.g. RMSE for root-mean-squared-error). Note, only those continuous statistics that are derivable from SL1L2 lines can be used.
6. For the CTC line type, set the "column" entry to the CTS output column that contains the statistic of intereest (e.g. PODY for probability of detecting yes). Note, consider specifying the "fcst_thresh" for the CTC line type.

For each term, all matching SL1L2 (or CTC) input lines are aggregated separately for the forecast and reference models. The requested statistic ("column") is derived from the aggregated partial sums or counts. For each term, a skill score is defined as:

$$ss = 1.0 - \frac{s_{fcst}^2}{s_{ref}^2}$$

Where s_{fcst} and s_{ref} are the aggregated forecast and reference statistics, respectively. Next, a weighted average is computed from the skill scores for each term:

$$ss_{avg} = \frac{1}{n} \sum_{i=1}^{n} w_i * ss_i$$

Where, w_i and ss_i are the weight and skill score for each term and n is the number of terms. Finally, the

skill score index is computed as:

$$index = \sqrt{\frac{1.0}{1.0 - ss_{avg}}}$$

A value greater than 1.0 indicates that the forecast model outperforms the reference, while a value less than 1.0 indicates that the reference outperforms the forecast.

The default skill score index name (SS_INDEX) can be overridden using the "ss_index_name" option in the configuration file. The pre-defined configuration files for the GO Index and CBS Index use "GO_INDEX" and "CBS_INDEX", respectively.

When running a skill score index job using the "-out_stat" job command option, a .stat output file is written containing the skill score index (SSIDX) output line type. If the "-by" job command option is specified, the skill score index will be computed separately for each unique combination of values found in the column(s) specified. For example, "-by FCST_INIT_BEG,VX_MASK" runs the job separately for each combination of model initialization time and verification region found in the input. Note that increasing the Stat-Analysis verbosity level (-v 3) on the command line prints detailed information about each skill score index term.

17.2.6 GO Index

The "go_index" job is a special case of the "ss_index" job, described in Section 17.2.5. The GO Index is a weighted average of 48 skill scores of RMSE statistics for wind speed, dew point temperature, temperature, height, and pressure at several levels in the atmosphere. The variables, levels, and lead times included in the index are listed in Table 17.1 and are defined by the default "STATAnalysisConfig_GO_Index" configuration file. The partial sums (SL1L2 lines in the STAT output) for each of these variables at each level and lead time must have been computed in a previous step. The Stat-Analysis tool then uses the weights in Table 17.1 to compute values for the GO Index.

Table 17.1: Variables, levels, and weights used to compute the GO Index.

Variable	Level	Weights by Lead time			
		12 h	24 h	36 h	48 h
Wind speed	250 hPa	4	3	2	1
	400 hPa	4	3	2	1
	850 hPa	4	3	2	1
	Surface	8	6	4	2
Dew point temperature	400 hPa	8	6	4	2
	700 hPa	8	6	4	2
	850 hPa	8	6	4	2
	Surface	8	6	4	2
Temperature	400 hPa	4	3	2	1
	Surface	8	6	4	2
Height	400 hPa	4	3	2	1
Pressure	Mean sea level	8	6	4	2

17.2.7 CBS Index

The "cbs_index" job is a special case of the "ss_index" job, described in Section 17.2.5. The CBS Index is a weighted average of 40 skill scores of RMSE statistics for mean sea level pressure, height, and wind speed at multiple levels computed over the northern hemisphere, southern hemisphere and the tropics. The variables, levels, lead times, and regions included in the index are listed in Table 17.2 and are defined by the default "STATAnalysisConfig_CBS_Index" configuration file. The partial sums (SL1L2 lines in the STAT output) for each of these variables for each level, lead time, and masking region must have been computed in a previous step. The Stat-Analysis tool then uses the weights in Table 17.2 to compute values for the CBS Index.

Table 17.2: Variables, levels, and weights used to compute the CBS Index for 24, 48, 72, 96 and 120 hour lead times.

Variable	Level	Weights by Region		
		North Hem	Tropics	South Hem
Pressure	Mean sea level	6.4	x	3.2
Height	500 hPa	2.4	x	1.2
Wind speed	250 hPa	2.4	1.2	1.2
	850 hPa	x	2.0	x

17.2.8 Ramp Events

The Stat-Analysis "ramp" job identifies ramp events (large increases or decreases in values over a time window) in both the forecast and observation data. It categorizes these events as hits, misses, false alarms, or correct negatives by applying a configurable matching time window and computes the corresponding categorical statistics.

17.2.9 Wind Direction Statistics

The Stat-Analysis "aggregate_stat" job can read vector partial sums and derive wind direction error statistics (WDIR). The vector partial sums (VL1L2 or VAL1L2) or matched pairs (MPR) for the UGRD and VGRD must have been computed in a previous step, i.e. by Point-Stat or Grid-Stat tools. This job computes an average forecast wind direction and an average observed wind direction along with their difference. The output is in degrees. In Point-Stat and Grid-Stat, the UGRD and VGRD can be verified using thresholds on their values or on the calculated wind speed. If thresholds have been applied, the wind direction statistics are calculated for each threshold.

The first step in verifying wind direction is running the Grid-Stat and/or Point-Stat tools to verify each forecast of interest and generate the VL1L2 or MPR line(s). When running these tools, please note:

1. To generate VL1L2 or MPR lines, the user must request the verification of both the U-component and V-component of wind at the same vertical levels.
2. To generate VL1L2 or MPR lines, the user must set the "output_flag" to indicate that the VL1L2 or MPR line should be computed and written out.
3. The user may select one or more spatial verification regions over which to accumulate the statistics.

4. The user may select one or more wind speed thresholds to be applied to the U and V wind components when computing the VL1L2 lines. It may be useful to investigate the performance of wind forecasts using multiple wind speed thresholds. For MPR line types, the wind speed threshold can be applied when computing the MPR lines, or the MPR output may be filtered afterwards by the Stat-Analysis tool.

Once the appropriate lines have been generated for each verification time of interest, the user may run the Stat-Analysis tool to analyze them. The Stat-Analysis job "aggregate_stat", along with the "-output_line_type WDIR" option, reads all of the input lines and computes statistics about the wind direction. When running this job the user is encouraged to use the many Stat-Analysis options to filter the input lines down to the set of lines of interest. The output of the wind direction analysis job consists of two lines with wind direction statistics computed in two slightly different ways. The two output lines begin with "ROW_MEAN_WDIR" and "AGGR_WDIR", and the computations are described below:

1. For the "ROW_MEAN_WDIR" line, each of the input VL1L2 lines is treated separately and given equal weight. The mean forecast wind direction, mean observation wind direction, and the associated error are computed for each of these lines. Then the means are computed across all of these forecast wind directions, observation wind directions, and their errors.

2. For the "AGGR_WDIR" line, the input VL1L2 lines are first aggregated into a single line of partial sums where the weight for each line is determined by the number of points it represents. From this aggregated line, the mean forecast wind direction, observation wind direction, and the associated error are computed and written out.

Note that wind direction is undefined for zero length vectors. Any inputs with zero length forecast or observation vectors are automatically excluded from the "ROW_MEAN_WDIR" output line since the direction error is also undefined. However, those inputs are included in the aggregated statistics in the "AGGR_WDIR" output line. For this reason, the "TOTAL" column value, indicating the number of inputs used, may larger in "AGGR_WDIR" than "ROW_MEAN_WDIR", from which zero length vectors have been excluded. Users can also specify the "-out_wind_thresh gt0 -out_wind_logic INTERSECTION" job command options to explicitly exclude all pairs containing zero length vectors from the analysis. In that case, the "ROW_MEAN_WDIR" and "AGGR_WDIR" "TOTAL" column values would match.

17.3 Practical Information

The following sections describe the usage statement, required arguments and optional arguments for the Stat-Analysis tool.

17.3.1 stat_analysis Usage

The usage statement for the Stat-Analysis tool is shown below:

```
Usage: stat_analysis
       -lookin path
       [-out file]
       [-tmp_dir path]
       [-log file]
```

(continues on next page)

(continued from previous page)

```
      [-v level]
      [-config config_file] | [JOB COMMAND LINE]
```

stat_analysis has two required arguments and accepts several optional ones.

In the usage statement for the Stat-Analysis tool, some additional terminology is introduced. In the Stat-Analysis tool, the term "job" refers to a set of tasks to be performed after applying user-specified options (i.e., "filters"). The filters are used to pare down a collection of output from the MET statistics tools to only those lines that are desired for the analysis. The job and its filters together comprise the "job command line". The "job command line" may be specified either on the command line to run a single analysis job or within the configuration file to run multiple analysis jobs at the same time. If jobs are specified in both the configuration file and the command line, only the jobs indicated in the configuration file will be run. The various jobs types are described in Table 17.3 and the filtering options are described in Section 17.3.2.

17.3.1.1 Required Arguments for stat_analysis

1. The **-lookin path** specifies the name of a directory to be searched recursively for STAT files (ending in ".stat") or any explicit file name with any suffix (such as "_ctc.txt") to be read. This option may be used multiple times to specify multiple directories and/or files to be read. If "-lookin python" is used, it must be followed by a Python embedding script and any command line arguments it takes. Python embedding can be used to pass **only** matched pair (MPR) lines as input to Stat-Analysis.
2. Either a configuration file must be specified with the **-config** option, or a **JOB COMMAND LINE** must be denoted. The **JOB COMMAND LINE** is described in Section 17.3.2

17.3.1.2 Optional Arguments for stat_analysis

3. The **-config config_file** specifies the configuration file to be used. The contents of the configuration file are discussed below.
4. The **-out file** option indicates the file to which output data should be written. If this option is not used, the output is directed to standard output.
5. The **-tmp_dir path** option selects the directory for writing out temporary files.
6. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
7. The **-v level** indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

An example of the stat_analysis calling sequence is shown below.

```
stat_analysis -lookin ../out/point_stat \
-config STATAnalysisConfig
```

In this example, the Stat-Analysis tool will search for valid STAT lines located in the *../out/point_stat* directory that meet the options specified in the configuration file, *config/STATAnalysisConfig*.

17.3.2 stat_analysis Configuration File

The default configuration file for the Stat-Analysis tool named **STATAnalysisConfig_default** can be found in the installed *share/met/config* directory. The version used for the example run in Section 3 is also available in *scripts/config*. Like the other configuration files described in this document, it is recommended that users make a copy of these files prior to modifying their contents.

The configuration file for the Stat-Analysis tool is optional. Users may find it more convenient initially to run Stat-Analysis jobs on the command line specifying job command options directly. Once the user has a set of or more jobs they would like to run routinely on the output of the MET statistics tools, they may find grouping those jobs together into a configuration file to be more convenient.

Most of the user-specified parameters listed in the Stat-Analysis configuration file are used to filter the ASCII statistical output from the MET statistics tools down to a desired subset of lines over which statistics are to be computed. Only output that meets all of the parameters specified in the Stat-Analysis configuration file will be retained.

The Stat-Analysis tool actually performs a two step process when reading input data. First, it stores the filtering information in the defined top section of the configuration file. It applies that filtering criteria when reading the input STAT data and writes the filtered data out to a temporary file, as described in `Contributor's Guide Section %s`. Second, each job defined in the **jobs** entry reads data from that temporary file and performs the task defined for the job. After all jobs have run, the Stat-Analysis tool deletes the temporary file.

This two step process enables the Stat-Analysis tool to run more efficiently when many jobs are defined in the configuration file. If only operating on a small subset of the input data, the common filtering criteria can be applied once rather than re-applying it for each job. In general, filtering criteria common to all tasks defined in the **jobs** entry should be moved to the top section of the configuration file.

As described above, filtering options specified in the first section of the configuration file will be applied to every task in the **jobs** entry. However, if an individual job specifies a particular option that was specified above, it will be applied for that job. For example, if the **model[]** option is set at the top to ["Run 1", "Run2"], but a job in the joblist sets the **-model** option as "Run1", that job will be performed only on "Run1" data. Also note that environment variables may be used when editing configuration files, as described in the Section 7.1.2 for the PB2NC tool.

```
boot           = { interval = PCTILE; rep_prop = 1.0; n_rep = 1000;
                   rng = "mt19937"; seed = ""; }
hss_ec_value   = NA;
rank_corr_flag = TRUE;
tmp_dir        = "/tmp";
version        = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
model = [];
```

The user may specify a comma-separated list of model names to be used for all analyses performed. The names must be in double quotation marks. If multiple models are listed, the analyses will be performed on their union. These selections may be further refined by using the "**-model**" option within the job command lines.

```
desc = [];
```

The user may specify a comma-separated list of description strings to be used for all analyses performed. The names must be in double quotation marks. If multiple description strings are listed, the analyses will be performed on their union. These selections may be further refined by using the "**-desc**" option within the job command lines.

```
fcst_lead = [];
obs_lead  = [];
```

The user may specify a comma-separated list of forecast and observation lead times in HH[MMSS] format to be used for any analyses to be performed. If multiple times are listed, the analyses will be performed on their union. These selections may be further refined by using the "**-fcst_lead**" and "**-obs_lead**" options within the job command lines.

```
fcst_valid_beg  = "";
fcst_valid_end  = "";
fcst_valid_inc  = [];
fcst_valid_exc  = [];
fcst_valid_hour = [];

obs_valid_beg   = "";
obs_valid_end   = "";
obs_valid_inc   = [];
obs_valid_exc   = [];
obs_valid_hour  = [];
```

The user may filter data based on its valid time. The fcst/obs_valid_beg and fcst/obs_valid_end options are strings in YYYYMMDD[_HH[MMSS]] format which define retention time windows for all analyses to be performed. The analyses are performed on all data whose valid time falls within these windows. If left as empty strings, no valid time window filtering is applied.

The fcst/obs_valid_hour options are arrays of strings in HH format which define the valid hour(s) of the data to be used. If specified, only data whose valid hour appears in the list of hours is used. The fcst/obs_valid_inc/exc options are arrays of strings in YYYYMMDD[_HH[MMSS]] format which explicitly define the valid times for data to be included or excluded from all analyses.

These selections may be further refined by using the **"-fcst_valid_beg"**, **"-fcst_valid_end"**, **"-fcst_valid_inc"**, **"-fcst_valid_exc"**, **"-fcst_valid_hour"**, **"-obs_valid_beg"**, **"-obs_valid_end"**, **"-obs_valid_inc"**, **"-obs_valid_exc"**, and **"-obs_valid_hour"** options within the job command line.

```
fcst_init_beg  = "";
fcst_init_end  = "";
fcst_init_inc  = [];
fcst_init_exc  = [];
fcst_init_hour = [];

obs_init_beg   = "";
obs_init_end   = "";
obs_init_inc   = [];
obs_init_exc   = [];
obs_init_hour  = [];
```

These time filtering options are the same as described above but applied to initialization times rather than valid times. These selections may be further refined by using the **"-fcst_init_beg"**, **"-fcst_init_end"**, **"-fcst_init_inc"**, **"-fcst_init_exc"**, **"-fcst_init_hour"**," **"-obs_init_beg"**, **"-obs_init_end"**, **"-obs_init_inc"**, **"-obs_init_exc"** and **"-obs_init_hour"** options within the job command line.

```
fcst_var = [];
obs_var  = [];
```

The user may specify a comma-separated list of forecast and observation variable types to be used for any analyses to be performed. If multiple variable types are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-fcst_var"** and **"-obs_var"** options within the job command lines.

```
fcst_units = [];
obs_units  = [];
```

The user may specify a comma-separated list of forecast and observation units to be used for any analyses to be performed. If multiple units are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-fcst_units"** and **"-obs_units"** options within the job command lines.

```
fcst_lev = [];
obs_lev  = [];
```

The user may specify a comma-separated list of forecast and observation level types to be used for any analyses to be performed. If multiple level types are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-fcst_lev"** and **"-obs_lev"** options within the job command lines.

```
obtype = [];
```

The user may specify a comma-separated list of observation types to be used for all analyses. If multiple observation types are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-obtype"** option within the job command line.

```
vx_mask = [];
```

The user may specify a comma-separated list of verification masking regions to be used for all analyses. If multiple verification masking regions are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-vx_mask"** option within the job command line.

```
interp_mthd = [];
```

The user may specify a comma-separated list of interpolation methods to be used for all analyses. If multiple interpolation methods are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-interp_mthd"** option within the job command line.

```
interp_pnts = [];
```

The user may specify a comma-separated list of interpolation points to be used for all analyses. If multiple interpolation points are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-interp_pnts"** option within the job command line.

```
fcst_thresh = [];
obs_thresh  = [];
cov_thresh  = [];
```

The user may specify comma-separated lists of forecast, observation, and coverage thresholds to be used for any analyses to be performed. If multiple thresholds are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-fcst_thresh"**, **"-obs_thresh"**, and **"-cov_thresh"** options within the job command lines.

```
alpha = [];
```

The user may specify a comma-separated list alpha confidence values to be used for all analyses. If alpha values are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-alpha"** option within the job command line.

```
line_type = [];
```

The user may specify a comma-separated list of line types to be used for all analyses. If multiple line types are listed, the analyses will be performed on their union. These selections may be further refined by using the **"-line_type"** option within the job command line.

```
column = [];
weight = [];
```

The column and weight entries are used to define a skill score index. They can either be set to a constant value of length one or specify a separate value for each term of the index.

```
ss_index_name       = "SS_INDEX";
ss_index_vld_thresh = 1.0;
```

The ss_index_name and ss_index_vld_thresh options are used to define a skill score index. The ss_index_name entry is a string which defines the output name for the current skill score index configuration. The ss_index_vld_thresh entry is a number between 0.0 and 1.0 that defines the required ratio of valid terms. If the ratio of valid skill score index terms to the total is less than than this number, no output is written for that case. The default value of 1.0 indicates that all terms are required.

```
jobs = [
 "-job filter -dump_row ./filter_job.stat"
];
```

The user may specify one or more analysis jobs to be performed on the STAT lines that remain after applying the filtering parameters listed above. Each entry in the joblist contains the task and additional filtering options for a single analysis to be performed. The format for an analysis job is as follows:

-job_name REQUIRED and OPTIONAL ARGUMENTS

All possible tasks for **job_name** are listed in Table 17.3.

Table 17.3: Description of components of the job command lines for the Stat-Analysis tool.Variables, levels, and weights used to compute the GO Index.

Job Name	Job commandDescription	Required Arguments
filter	Filters out the statistics lines based on applying options* (See note below table)	-dump_row
summary	Computes the mean, standard deviation, percentiles (min, 10th, 25th, 50th, 75th, 90th, and max), interquartile range, range, wmo_mean, and wmo_weighted_mean	-line_type -column
aggregate	Aggregates the statistics output, computing the statistic specified for the entire collection of valid lines	-line_type
aggregate_stat	Aggregates the statistics output, and converts the input line type to the output line type specified	-line_type -out_line_type
ss_index	Calculates a user-defined Skill Score index as described in section Section 17.2.5.	-model forecast -model reference
go_index	Calculates the GO Index as described in section Section 17.2.6.	-model forecast -model reference
cbs_index	Calculates the CBS Index as described in section Section 17.2.7.	-model forecast -model reference
ramp	Defines a ramp event on a time-series of forecast and observed values. The amount of change from one time to the next is computed for forecast and observed values. Those changes are thresholded to define events which are used to populate a 2x2 contingency table.	-ramp_type -ramp_thresh -out_line_type -column -ramp_time -ramp_exact -ramp_window

```
out_alpha = 0.05;
```

This entry specifies the alpha value to be used when computing confidence intervals for output statistics. It is similar to the **ci_alpha** entry described in Section 5.

```
wmo_sqrt_stats = [ "CNT:FSTDEV",  "CNT:OSTDEV",  "CNT:ESTDEV",
                   "CNT:RMSE",    "CNT:RMSFA",   "CNT:RMSOA",
                   "VCNT:FS_RMS", "VCNT:OS_RMS", "VCNT:RMSVE",
                   "VCNT:FSTDEV", "VCNT:OSTDEV" ];

wmo_fisher_stats = [ "CNT:PR_CORR", "CNT:SP_CORR",
                     "CNT:KT_CORR", "CNT:ANOM_CORR", "CNT:ANOM_CORR_UNCNTR" ];
```

These entries specify lists of statistics in the form LINE_TYPE:COLUMN to which the various WMO mean logic types should be applied for the summary job type.

```
vif_flag = FALSE;
```

The variance inflation factor (VIF) flag indicates whether to apply a first order variance inflation when calculating normal confidence intervals for an aggregated time series of contingency table counts or partial sums. The VIF adjusts the variance estimate for the lower effective sample size caused by autocorrelation of the statistics through time. A value of **FALSE** will not compute confidence intervals using the VIF. A value of **TRUE** will include the VIF, resulting in a slightly wider normal confidence interval.

The Stat-Analysis tool supports several additional job command options which may be specified either on the command line when running a single job or within the **jobs** entry within the configuration file. These additional options are described below:

```
-by col_name
```

This job command option is extremely useful. It can be used multiple times to specify a list of STAT header column names. When reading each input line, the Stat-Analysis tool concatenates together the entries in the specified columns and keeps track of the unique cases. It applies the logic defined for that job to each unique subset of data. For example, if your output was run over many different model names and masking regions, specify **-by MODEL,VX_MASK** to get output for each unique combination rather than having to run many very similar jobs.

```
-column_min     col_name value
-column_max     col_name value
-column_eq      col_name value
-column_thresh  col_name thresh
-column_str     col_name string
-column_str_exc col_name string
```

The column filtering options may be used when the **-line_type** has been set to a single value. These options take two arguments, the name of the data column to be used followed by a value, string, or threshold to be applied. If multiple column_min/max/eq/thresh/str options are listed, the job will be performed on their intersection. Each input line is only retained if its value meets the numeric filtering criteria defined, matches one of the strings defined by the **-column_str** option, or does not match any of the string defined by the **-column_str_exc** option. Multiple filtering strings may be listed using commas. Defining thresholds in MET is described in Section 5.

```
-dump_row file
```

Each analysis job is performed over a subset of the input data. Filtering the input data down to a desired subset is often an iterative process. The **-dump_row** option may be used for each job to specify the name of an output file to which the exact subset of data used for that job will be written. When initially constructing Stat-Analysis jobs, users are strongly encouraged to use the option and check its contents to ensure that the analysis was actually done over the intended subset.

```
-out_line_type name
```

This option specifies the desired output line type(s) for the **aggregate_stat** job type.

```
-out_stat file
-set_hdr  col_name string
```

The Stat-Analysis tool writes its output to either the log file or the file specified using the **-out** command line option. However the **aggregate** and **aggregate_stat** jobs create STAT output lines and the standard output written lacks the full set of STAT header columns. The **-out_stat** job command option may be used for these jobs to specify the name of an output file to which full STAT output lines should be written. When the **-out_stat** job command option is used for **aggregate** and **aggregate_stat** jobs the output is sent to the **-out_stat** file instead of the log or **-out** file.

Jobs will often combine output with multiple entries in the header columns. For example, a job may aggregate output with three different values in the **VX_MASK** column, such as "mask1", "mask2", and "mask3". The output **VX_MASK** column will contain the unique values encountered concatenated together with commas: "mask1,mask2,mask3". Alternatively, the **-set_hdr** option may be used to specify what should be written to the output header columns, such as "-set_hdr VX_MASK all_three_masks". When **-set_hdr** is specified for **filter** jobs, it controls what is written to the **-dump_row** output file.

When using the "-out_stat" option to create a .stat output file and stratifying results using one or more "-by" job command options, those columns may be referenced in the "-set_hdr" option. When using multiple "-by" options, use "CASE" to reference the full case information string:

```
-job aggregate_stat -line_type MPR -out_line_type CNT -by FCST_VAR,OBS_SID \
-set_hdr VX_MASK OBS_SID -set_hdr DESC CASE
```

The example above reads MPR lines, stratifies the data by forecast variable name and station ID, and writes the output for each case to a .stat output file. When creating the .stat file, write the full case information to the DESC output column and the station ID to the VX_MASK column.

```
-mask_grid name
-mask_poly file
-mask_sid  file|list
```

When processing input MPR lines, these options may be used to define a masking grid, polyline, or list of station ID's to filter the matched pair data geographically prior to computing statistics. The **-mask_sid** option is a station ID masking file or a comma-separated list of station ID's for filtering the matched pairs spatially. See the description of the "sid" entry in Section 5.

```
-out_fcst_thresh thresh
-out_obs_thresh  thresh
-out_thresh      thresh
-out_cnt_logic   string
```

When processing input MPR lines, these options are used to define the forecast, observation, or both thresholds to be applied when computing statistics. For categorical output line types (FHO, CTC, CTS, MCTC, MCTS) these define the categorical thresholds. For continuous output line types (SL1L2, SAL1L2, CNT), these define the continuous filtering thresholds and **-out_cnt_logic** defines how the forecast and observed logic should be combined.

```
-out_fcst_wind_thresh thresh
-out_obs_wind_thresh  thresh
-out_wind_thresh      thresh
-out_wind_logic       string
```

These job command options are analogous to the options listed above but apply when processing input MPR lines and deriving wind direction statistics.

```
-out_bin_size value
```

When processing input ORANK lines and writing output RHIST or PHIST lines, this option defines the output histogram bin width to be used.

17.3.3 stat-analysis Tool Output

The output generated by the Stat-Analysis tool contains statistics produced by the analysis. It also records information about the analysis job that produced the output for each line. Generally, the output is printed to the screen. However, it can be redirected to an output file using the "**-out**" option. The format of output from each STAT job command is described below.

The "**-by column**" job command option may be used to run the same job multiple times on unique subsets of data. Specify the "**-by column**" option one or more times to define a search key, and that job will be run once for each unique search key found. For example, use "-by VX_MASK" to run the same job for multiple masking regions, and output will be generated for each unique masking region found. Use "-by VX_MASK -by FCST_LEAD" to generate output for each unique combination of masking region and lead time.

17.3.3.1 Job: filter

This job command finds and filters STAT lines down to those meeting criteria specified by the filter's options. The filtered STAT lines are written to a file specified by the "**-dump_row**" option.

The output of this job is the same STAT format described in sections Section 11.3.3, Section 13.3.3, and Section 15.3.3.

17.3.3.2 Job: summary

This job produces summary statistics for the column name and line type specified by the "**-column**" and "**-line_type**" options. The output of this job type consists of three lines. The first line contains "**JOB_LIST**", followed by a colon, then the filtering and job definition parameters used for this job. The second line contains "**COL_NAME**", followed by a colon, then the column names for the data in the next line. The third line contains the word "**SUMMARY**", followed by a colon, then the total, mean with confidence intervals, standard deviation with confidence intervals, minimum value, percentiles (10th, 25th, 50th, 75th, and 90th), the maximum value, the interquartile range, the range, and WMO mean information. The output columns are shown in Table 17.4 below.

Table 17.4: Columnar output of "summary" job output from the Stat-Analysis tool.

Column Number	Description
1	SUMMARY: (job type)
2	Total
3-7	Mean including normal and bootstrap upper and lower confidence limits
8-10	Standard deviation including bootstrap upper and lower confidence limits
11	Minimum value
12	10th percentile
13	25th percentile
14	Median (50th percentile)
15	75th percentile
16	90th percentile
17	Maximum value
18	Interquartile range (75th - 25th percentile)
19	Range (Maximum - Minimum)
20	WMO Mean type
21	WMO Unweighted Mean value
22	WMO Weighted Mean value

17.3.3.3 Job: aggregate

This job aggregates output from the STAT line type specified using the "**-line_type**" argument. The output of this job type is in the same format as the line type specified (see Section 11.3.3, Section 13.3.3, and Section 15.3.3). Again the output consists of three lines. The first line contains "**JOB_LIST**", as described above. The second line contains "**COL_NAME**", followed by a colon, then the column names for the line type selected. The third line contains the name of the line type selected followed by the statistics for that line type.

The STAT line types which may be aggregated in this way are the contingency table (FHO, CTC, PCT, MCTC, NBRCTC), partial sums (SL1L2, SAL1L2, VL1L2, and VAL1L2), and other (ISC, ECNT, RPS, RHIST, PHIST, RELP, NBRCNT, SSVAR, and GRAD) line types.

17.3.3.4 Job: aggregate_stat

This job is similar to the "**aggregate**" job listed above, however the format of its output is determined by the "**-out_line_type**" argument. Again the output consists of three lines for "**JOB_LIST**", "**COL_NAME**", and the name of the output STAT line, as described above. Valid combinations of the "**-line_type**" and "**-out_line_type**" arguments are listed in Table 17.5 below.

Table 17.5: Valid combinations of "-line_type" and "-out_line_type" arguments for the "aggregate_stat" job.

Input Line Type	Output Line Type
FHO or CTC	CTS
MCTC	MCTS
SL1L2 or SAL1L2	CNT
VL1L2 or VAL1L2	WDIR (wind direction), VCNT
PCT	PSTD, PJC, PRC
NBRCTC	NBRCTS
ORANK	RHIST, PHIST, RELP, SSVAR
MPR	CNT, SL1L2, SAL1L2, WDIR
MPR	FHO, CTC, CTS, MCTC, MCTS, PCT, PSTD, PJC, or PRC (must specify "**-out_fcst_thresh**" and "**-out_obs_thresh**" arguments)

17.3.3.5 Job: ss_index, go_index, cbs_index

While the inputs for the "ss_index", "go_index", and "cbs_index" jobs may vary, the output is the same. By default, the job output is written to the screen or to a "-out" file, if specified. If the "-out_stat" job command option is specified, a STAT output file is written containing the skill score index (SSIDX) output line type.

The SSIDX line type consists of the common STAT header columns described in Table 11.1 followed by the columns described below. In general, when multiple input header strings are encountered, the output is reported as a comma-separated list of the unique values. The "-set_hdr" job command option can be used to override any of the output header strings (e.g. "-set_hdr VX_MASK MANY" sets the output VX_MASK column to "MANY"). Special logic applied to some of the STAT header columns are also described below.

Table 17.6: Format information for the SSIDX (Skill Score Index) output line type.

Column Number	SSIDX Column Name	Description	Data Type
4	FCST_LEAD	Maximum input forecast lead time	Time String
5	FCST_VALID_BEG	Minimum input forecast valid start time	Datetime String
6	FCST_VALID_END	Maximum input forecast valid end time	Datetime String
7	OBS_LEAD	Maximum input observation lead time	Time String
8	OBS_VALID_BEG	Minimum input observation valid start time	Datetime String
9	OBS_VALID_END	Maximum input observation valid end time	Datetime String
10	FCST_VAR	Skill score index name from the "ss_index_name" option	String
11	OBS_VAR	Skill score index name from the "ss_index_name" option	String
24	SSIDX	Skill score index line type	String
25	FCST_MODEL	Forecast model name	String
26	REF_MODEL	Reference model name	String
27	N_INIT	Number of unique input model initialization times	Integer
28	N_TERM	Number of skill score index terms	Integer
29	N_VLD	Number of terms for which a valid skill score was computed	Integer
30	SS_INDEX	Skill score index value	Double

17.3.3.6 Job: ramp

The ramp job operates on a time-series of forecast and observed values and is analogous to the RIRW (Rapid Intensification and Weakening) job described in Section 27.3.2. The amount of change from one time to the next is computed for forecast and observed values. Those changes are thresholded to define events which are used to populate a 2x2 contingency table.

See Section 5 for a detailed description of the job command options available for ramp job type.

The default output for this job is contingency table counts and statistics (**-out_line_type CTC,CTS**). Matched pair information may also be output by requesting MPR output (**-out_line_type CTC,CTS,MPR**).

Chapter 18

Series-Analysis Tool

18.1 Introduction

The Series-Analysis Tool accumulates statistics separately for each horizontal grid location over a series. Often, this series is over time or height, though any type of series is possible. This differs from the Grid-Stat tool in that Grid-Stat verifies all grid locations together as a group. Thus, the Series-Analysis Tool can be used to find verification information specific to certain locations or see how model performance varies over the domain.

18.2 Practical Information

This Series-Analysis tool performs verification of gridded model fields using matching gridded observation fields. It computes a variety of user-selected statistics. These statistics are a subset of those produced by the Grid-Stat tool, with options for statistic types, thresholds, and conditional verification options as discussed in Section 13. However, these statistics are computed separately for each grid location and accumulated over some series such as time or height, rather than accumulated over the whole domain for a single time or height as is done by Grid-Stat.

This tool computes statistics for exactly one series each time it is run. Multiple series may be processed by running the tool multiple times. The length of the series to be processed is determined by the first of the following that is greater than one: the number of forecast fields in the configuration file, the number of observation fields in the configuration file, the number of input forecast files, the number of input observation files. Several examples of defining series are described below.

To define a time series of forecasts where the valid time changes for each time step, set the forecast and observation fields in the configuration file to single values and pass the tool multiple forecast and observation files. The tool will loop over the forecast files, extract the specified field from each, and then search the observation files for a matching record with the same valid time.

To define a time series of forecasts that all have the same valid time, set the forecast and observation fields in the configuration file to single values. Pass the tool multiple forecast files and a single observation file containing the verifying observations. The tool will loop over the forecast files, extract the specified field from each, and then retrieve the verifying observations.

To define a series of vertical levels all contained in a single input file, set the forecast and observation fields to a list of the vertical levels to be used. Pass the tool single forecast and observation files containing the vertical level data. The tool will loop over the forecast field entries, extract that field from the input forecast file, and then search the observation file for a matching record.

18.2.1 series_analysis Usage

The usage statement for the Series-Analysis tool is shown below:

```
Usage: series_analysis
       -fcst  file_1 ... file_n | file_list
       -obs   file_1 ... file_n | file_list
       [-both file_1 ... file_n | file_list]
       [-aggr file]
       [-paired]
       -out file
       -config file
       [-log file]
       [-v level]
       [-compress level]
```

series_analysis has four required arguments and accepts several optional ones.

18.2.1.1 Required Arguments series_stat

1. The **-fcst file_1 ... file_n | file_list** option specifies the gridded forecast files or ASCII file list of file names to be used, as described in Section 4.1.1.
2. The **-obs file_1 ... file_n | file_list** option specifies the gridded observation files or ASCII file list of file names to be used, as described in Section 4.1.1.
3. The **-out file** is the NetCDF output file containing computed statistics.
4. The **-config file** is a Series-Analysis Configuration file containing the desired settings.

18.2.1.2 Optional Arguments for series_analysis

5. To set both the forecast and observations to the same set of files, use the optional **-both file_1 ... file_n | file_list** option to the same set of files. This is useful when reading the NetCDF matched pair output of the Grid-Stat tool which contains both forecast and observation data.
6. The **-aggr** option specifies the path to an existing Series-Analysis output file. When computing statistics for the input forecast and observation data, Series-Analysis aggregates the partial sums (SL1L2, SAL1L2 line types) and contingency table counts (CTC, MCTC, and PCT line types) with data provided in the aggregate file. This option enables Series-Analysis to run iteratively and update existing partial sums, counts, and statistics with new data.

Note: When the **-aggr** option is used, only statistics that are derivable from partial sums and contingency table counts can be requested. Runtimes are generally much slower when aggregating data since it requires many additional NetCDF variables containing the scalar partial sums and contingency table counts to be read and written.

7. The **-paired** option indicates that the **-fcst** and **-obs** file lists are already paired, meaning there is a one-to-one correspondence between the files in those lists. This option affects how missing data is handled. When **-paired** is not used, missing or incomplete files result in a runtime error with no output file being created. When **-paired** is used, missing or incomplete files result in a warning with output being created using the available data.
8. The **-log** file outputs log messages to the specified file.
9. The **-v** level overrides the default level of logging (2).
10. The **-compress** level option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the series_analysis calling sequence is shown below:

```
series_analysis \
-fcst   myfcstfilelist.txt \
-obs    myobsfilelist.txt \
-config SeriesAnalysisConfig \
-out    out/my_series_statistics.nc
```

In this example, the Series-Analysis tool will process the list of forecast and observation files specified in the text file lists into statistics for each grid location using settings specified in the configuration file. Series-Analysis will create an output NetCDF file containing requested statistics.

18.2.2 series_analysis Output

The Series-Analysis tool produces NetCDF files containing output statistics for each grid location from the input files. The details about the output statistics available from each output line type are detailed in Chapter 5 since they are also produced by the Grid-Stat Tool. A subset of these can be produced by this tool, with the most notable exceptions being the wind vector and neighborhood statistics. Users can inventory the contents of the Series-Analysis output files using the ncdump -h command to view header information. Additionally, ncview or the Plot-Data-Plane tool can be used to visualize the output. An example of Series-Analysis output is shown in Figure 18.1 below.

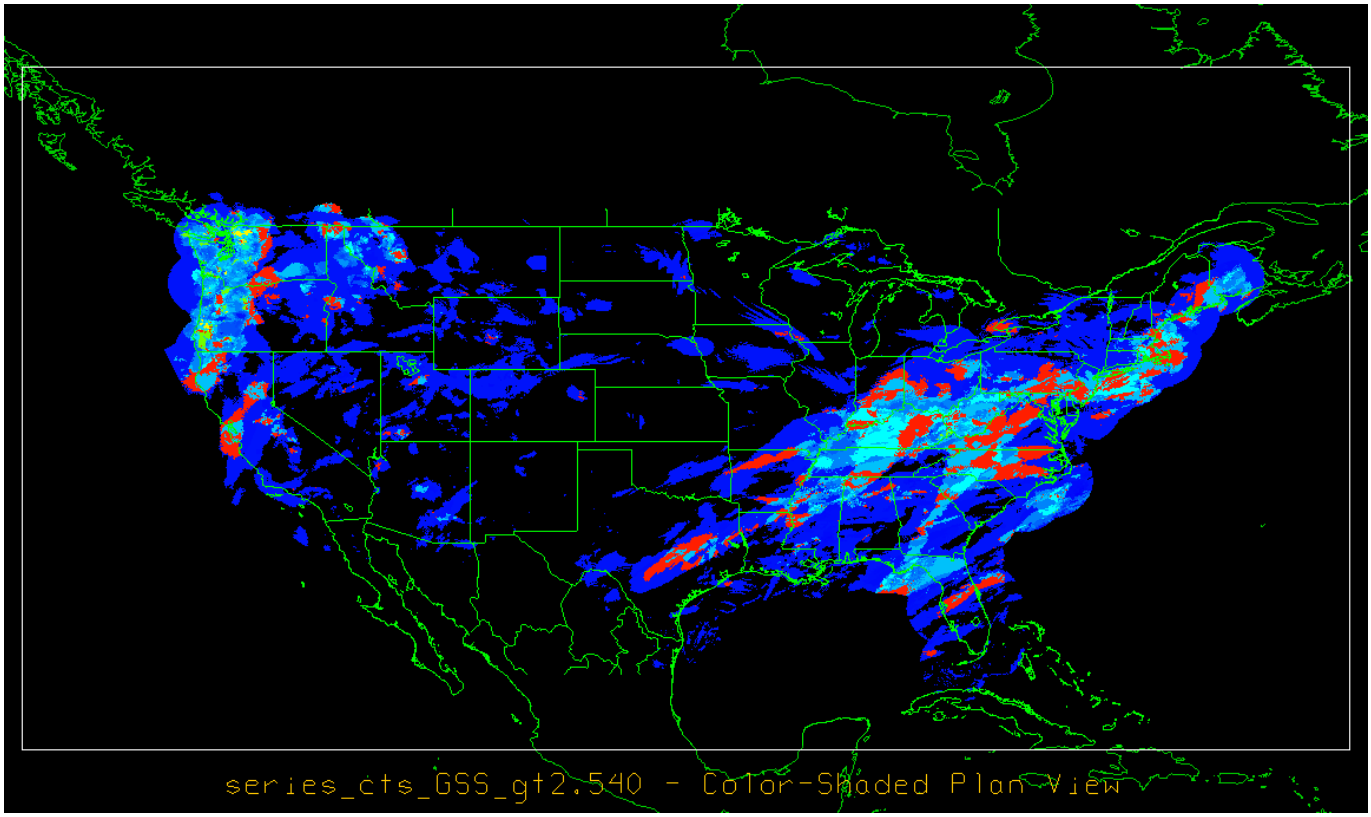

Figure 18.1: An example of the Gilbert Skill Score for precipitation forecasts at each grid location for a month of files.

18.2.3 series_analysis Configuration File

The default configuration file for the Series-Analysis tool named **SeriesAnalysisConfig_default** can be found in the installed *share/met/config* directory. The contents of the configuration file are described in the subsections below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
model          = "FCST";
desc           = "NA";
obtype         = "ANALYS";
regrid         = { ... }
fcst           = { ... }
obs            = { ... }
climo_mean     = { ... }
climo_stdev    = { ... }
ci_alpha       = [ 0.05 ];
boot           = { interval = PCTILE; rep_prop = 1.0; n_rep = 1000;
                   rng = "mt19937"; seed = ""; }
```

(continues on next page)

(continued from previous page)

```
mask           = { grid = ""; poly = ""; }
gradient       = { dx = [ 1 ]; dy = [ 1 ]; }
hss_ec_value   = NA;
rank_corr_flag = TRUE;
tmp_dir        = "/tmp";
version        = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
block_size = 1024;
```

Number of grid points to be processed concurrently. Set smaller to use less memory but increase the number of passes through the data. The amount of memory the Series-Analysis tool consumes is determined by the size of the grid, the length of the series, and the block_size entry defined above. The larger this entry is set the faster the tool will run, subject to the amount of memory available on the machine. If set less than or equal to 0, it is automatically reset to the number of grid points, and they are all processed concurrently.

```
vld_thresh = 1.0;
```

Ratio of valid matched pairs for the series of values at each grid point required to compute statistics. Set to a lower proportion to allow some missing values. Setting it to 1.0 requires that every data point be valid over the series to compute statistics.

```
output_stats = {
   fho    = [];
   ctc    = [];
   cts    = [];
   mctc   = [];
   mcts   = [];
   cnt    = ["RMSE", "FBAR", "OBAR"];
   sl1l2  = [];
   sal1l2 = [];
   pct    = [];
   pstd   = [];
   pjc    = [];
   prc    = [];
   grad   = [];
}
```

The output_stats array controls the type of output that the Series-Analysis tool generates. Each flag corresponds to an output line type in the STAT file and is used to specify the comma-separated list of statistics to be computed. Use the column names from the tables listed below to specify the statistics. The output flags correspond to the following types of output line types:

1. FHO for Forecast, Hit, Observation Rates (See Table 11.2)
2. CTC for Contingency Table Counts (See Table 11.3)
3. CTS for Contingency Table Statistics (See Table 11.4)
4. MCTC for Multi-Category Contingency Table Counts (See Table 11.6)
5. MCTS for Multi-Category Contingency Table Statistics (See Table 11.7)
6. CNT for Continuous Statistics (See Table 11.5)
7. SL1L2 for Scalar L1L2 Partial Sums (See Table 11.13)
8. SAL1L2 for Scalar Anomaly L1L2 Partial Sums climatological data is supplied (See Table 11.14)
9. PCT for Contingency Table Counts for Probabilistic forecasts (See Table 11.8)
10. PSTD for Contingency Table Statistics for Probabilistic forecasts (See Table 11.9)
11. PJC for Joint and Conditional factorization for Probabilistic forecasts (See Table 11.10)
12. PRC for Receiver Operating Characteristic for Probabilistic forecasts (See Table 11.11)
13. GRAD for Gradient Statistics (See Table 13.5)

Note: When the -input option is used, all partial sum and contingency table count columns are required to aggregate statistics across multiple runs. To facilitate this, the output_stats entries for the CTC, SL1L2, SAL1L2, PCT, and GRAD line types can be set to "ALL" to indicate that all available columns for those line types should be written.

Chapter 19

Grid-Diag Tool

19.1 Introduction

The Grid-Diag tool creates histograms (probability distributions when normalized) for an arbitrary collection of data fields and levels. Joint histograms will be created for all possible pairs of variables. Masks can be used to subset the data fields spatially. The histograms are accumulated over a time series of input data files, similar to Series-Analysis.

19.2 Practical Information

19.2.1 grid_diag Usage

The following sections describe the usage statement, required arguments, and optional arguments for **grid_diag**.

```
Usage: grid_diag
       -data file_1 ... file_n | file_list
       -out file
       -config file
       [-log file]
       [-v level]
       [-compress level]
```

Note: The "-data" option can be used once to read all fields from each input file or once for each field to be processed.

---

grid_diag has required arguments and can accept several optional arguments.

19.2.1.1 Required Arguments for grid_diag

1. The **-data file_1 ... file_n | file_list** options specify the gridded data files or an ASCII file containing a list of file names to be used, as described in Section 4.1.1.

When **-data** is used once, all fields are read from each input file. When used multiple times, it must match the number of fields to be processed. In this case the first field in the config data field list is read from the files designated by the first **-data**, the second field in the field list is read from files designated by the second **-data**, and so forth. All files within each set must be of the same file type, but the file types of each set may differ. A typical use case for this option is for the first **-data** to specify forecast data files and the second **-data** the observation data files.

2. The **-out** argument is the NetCDF output file.

3. The **-config file** is the configuration file to be used. The contents of the configuration file are discussed below.

19.2.1.2 Optional Arguments for grid_diag

4. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.

5. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

6. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

19.2.2 grid_diag Configuration File

The default configuration file for the Grid-Diag tool named **GridDiagConfig_default** can be found in the installed *share/met/config/* directory. It is encouraged for users to copy these default files before modifying their contents. The contents of the configuration file are described in the subsections below.

```
desc          = "GFS";
regrid        = { ... }
censor_thresh = [];
censor_val    = [];
mask          = { grid = ""; poly = ""; }
version       = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
data = {
 field = [
      {
         name   = "APCP";
         level  = ["L0"];
         n_bins = 30;
         range  = [0, 12];
      },
      {
         name   = "PWAT";
         level  = ["L0"];
         n_bins = 35;
         range  = [35, 70];
      }
   ];
}
```

The **name** and **level** entries in the **data** dictionary define the data to be processed. The **n_bins** parameter specifies the number of histogram bins for that variable, and the **range** parameter the lower and upper bounds of the histogram. The interval length is the upper and lower difference divided by **n_bins**. Each bin is inclusive on the left side and exclusive on the right, such as [a,b).

Grid-Diag prints a warning message if the actual range of data values falls outside the range defined for that variable in the configuration file. Any data values less than the configured range are counted in the first bin, while values greater than the configured range are counted in the last bin.

19.2.3 grid_diag Output File

The NetCDF file has a dimension for each of the specified data variable and level combinations, e.g. APCP_L0 and PWAT_L0. The bin minimum, midpoint, and maximum values are indicated with an _min, _mid, or _max appended to the variable/level.

For each variable/level combination in the data dictionary, a corresponding histogram will be written to the NetCDF output file. For example, hist_APCP_L0 and hist_PWAT_L0 are the counts of all data values falling within the bin. Data values below the minimum or above the maximum are included in the lowest and highest bins, respectively. A warning message is printed when the range of the data falls outside the range defined in the configuration file. In addition to 1D histograms, 2D histograms for all variable/level pairs are written. For example, hist_APCP_L0_PWAT_L0 is the joint histogram for those two variables/levels. The output variables for grid_size, mask_size, and n_series specify the number of points in the grid, the number of grid points in the mask, and the number of files that were processed, respectively. The range of the initialization, valid, and lead times processed is written to the global attributes.

Chapter 20

MODE Tool

20.1 Introduction

This section provides a description of the Method for Object-Based Diagnostic Evaluation (MODE) tool, which was developed at the Research Applications Laboratory, NCAR/Boulder, USA. More information about MODE can be found in *Davis et al. (2006a,b)* (page 493), *Brown et al. (2007)* (page 492) and *Bullock et al. (2016)* (page 493).

MODE was developed in response to a need for verification methods that can provide diagnostic information that is more directly useful and meaningful than the information that can be obtained from traditional verification approaches, especially in application to high-resolution NWP output. The MODE approach was originally developed for application to spatial precipitation forecasts, but it can also be applied to other fields with coherent spatial structures (*e.g.*, clouds, convection). MODE is only one of a number of different approaches that have been developed in recent years to meet these needs. In the future, we expect that the MET package will include additional methods. References for many of these methods are provided at the MesoVict website.

MODE may be used in a generalized way to compare any two fields. For simplicity, field_1 may be thought of in this section as "the forecast", while field_2 may be thought of as "the observation", which is usually a gridded analysis of some sort. The convention of field_1/field_2 is also used in Table 20.2. MODE resolves objects in both the forecast and observed fields. These objects mimic what humans would call "regions of interest". Object attributes are calculated and compared, and are used to associate ("merge") objects within a single field, as well as to "match" objects between the forecast and observed fields. Finally, summary statistics describing the objects and object pairs are produced. These statistics can be used to identify correlations and differences among the objects, leading to insights concerning forecast strengths and weaknesses.

20.2 Scientific and Statistical Aspects

The methods used by the MODE tool to identify and match forecast and observed objects are briefly described in this section.

20.2.1 Resolving Objects

The process used for resolving objects in a raw data field is called *convolution thresholding*. The raw data field is first convolved with a simple filter function as follows:

$$C(x,y) = \sum_{u,v} \phi(u,v)\ f(x-u, y-v)$$

In this formula, f is the raw data field, ϕ is the filter function, and C is the resulting convolved field. The variables (x,y) and (u,v) are grid coordinates. The filter function ϕ is a simple circular filter determined by a radius of influence R , and a height H :

$$\phi(x,y) = \begin{cases} H & \text{if } x^2+y^2 \leq R^2 \\ 0 & \text{otherwise.} \end{cases} \tag{20.1}$$

The parameters R and H are not independent. They are related by the requirement that the integral of ϕ over the grid be unity:

$$\pi R^2 H = 1.$$

Thus, the radius of influence R is the only tunable parameter in the convolution process. Once R is chosen, H is determined by the above equation.

Once the convolved field C is in hand, it is thresholded to create a mask field M :

$$M(x,y) = \begin{cases} 1 & \text{if } C(x,y) \geq T \\ 0 & \text{otherwise.} \end{cases} \tag{20.2}$$

where T is the threshold. The objects are the connected regions where $M = 1$. Finally, the raw data are restored to object interiors to obtain the object field F :

$$F(x,y) = M(x,y) f(x,y).$$

Thus, two parameters - the radius of influence R, and the threshold T - control the entire process of resolving objects in the raw data field.

An example of the steps involved in resolving objects is shown in Figure 20.1. It shows a "raw" precipitation field, where the vertical coordinate represents the precipitation amount. Part (b) shows the convolved field, and part (c) shows the masked field obtained after the threshold is applied. Finally, Figure 20.1 shows the objects once the original precipitation values have been restored to the interiors of the objects.

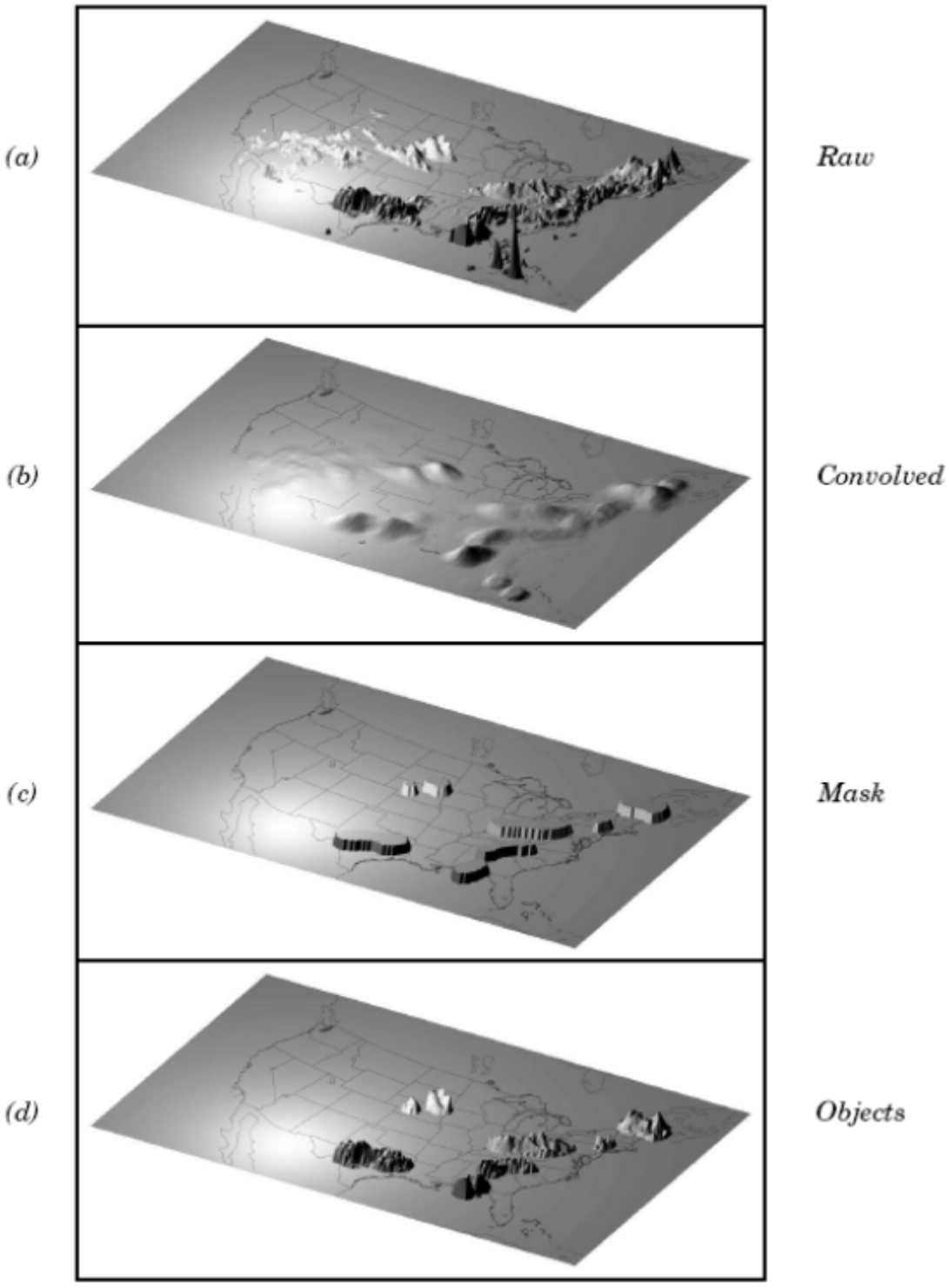

Figure 20.1: Example of an application of the MODE object identification process to a model precipitation field.

20.2.2 Attributes

Object attributes are defined both for single objects and for object pairs. One of the objects in a pair is from the forecast field and the other is taken from the observed field.

Area is simply a count of the number of grid squares an object occupies. If desired, a true area (say, in km^2) can be obtained by adding up the true areas of all the grid squares inside an object, but in practice this is seldom necessary.

Moments are used in the calculation of several object attributes. If we define $\xi(x, y)$ to be 1 for points (x, y) inside our object, and zero for points outside, then the first-order moments, S_x and S_y, are defined as

$$S_x = \sum_{x,y} x\xi(x, y) \text{ and } S_y = \sum_{x,y} y\xi(x, y)$$

Higher order moments are similarly defined and are used in the calculation of some of the other attributes. For example, the **centroid** is a kind of geometric center of an object, and can be calculated from first moments. It allows one to assign a single point location to what may be a large, extended object.

Axis Angle, denoted by θ, is calculated from the second-order moments. It gives information on the orientation or "tilt" of an object. **Curvature** is another attribute that uses moments in its calculation, specifically, third-order moments.

Aspect Ratio is computed by fitting a rectangle around an object. The rectangle is aligned so that it has the same axis angle as the object, and the length and width are chosen so as to just enclose the object. We make no claim that the rectangle so obtained is the smallest possible rectangle enclosing the given object. However, this rectangle is much easier to calculate than a smaller enclosing rectangle and serves our purposes just as well. Once the rectangle is determined, the aspect ratio of the object is defined to be the width of the fitted rectangle divided by its length.

Another object attribute defined by MODE is **complexity**. Complexity is defined by comparing the area of an object to the area of its convex hull.

All the attributes discussed so far are defined for single objects. Once these are determined, they can be used to calculate attributes for pairs of objects. One example is the **centroid difference**. This measure is simply the (vector) difference between the centroids of the two objects. Another example is the **angle difference**. This is the difference between the axis angles.

Several area measures are also used for pair attributes. **Union Area** is the total area that is in either one (or both) of the two objects. **Intersection Area** is the area that is inside both objects simultaneously. **Symmetric Difference** is the area inside at least one object, but not inside both.

20.2.3 Fuzzy Logic

Once object attributes $\alpha_1, \alpha_2, \ldots, \alpha_n$ are estimated, some of them are used as input to a fuzzy logic engine that performs the matching and merging steps. **Merging** refers to grouping together objects in a single field, while **matching** refers to grouping together objects in different fields, typically the forecast and observed fields. Interest maps I_i are applied to the individual attributes α_i to convert them into interest values, which range from zero (representing no interest) to one (high interest). For example, the default interest map for centroid difference is one for small distances, and falls to zero as the distance increases. For other attributes (*e.g.*, intersection area), low values indicate low interest, and high values indicate more interest.

The next step is to define confidence maps C_i for each attribute. These maps (again with values ranging from zero to one) reflect how confident we are in the calculated value of an attribute. The confidence maps generally are functions of the entire attribute vector $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, in contrast to the interest maps, where each I_i is a function only of α_i. To see why this is necessary, imagine an electronic anemometer that outputs a stream of numerical values of wind speed and direction. It is typically the case for such devices that when the wind speed becomes small enough, the wind direction is poorly resolved. The wind must be at least strong enough to overcome friction and turn the anemometer. Thus, in this case, our confidence in one attribute (wind direction) is dependent on the value of another attribute (wind speed). In MODE, all of the confidence maps except the map for axis angle are set to a constant value of 1. The axis angle confidence map is a function of aspect ratio, with values near one having low confidence, and values far from one having high confidence.

Next, scalar weights $\boldsymbol{w}_i$ are assigned to each attribute, representing an empirical judgment regarding the relative importance of the various attributes. As an example, the initial development of MODE, centroid distance was weighted more heavily than other attributes, because the location of storm systems close to each other in space seemed to be a strong indication (stronger than that given by any other attribute) that they were related.

Finally, all these ingredients are collected into a single number called the total interest, $\boldsymbol{T}$, given by:

$$T(\alpha) = \frac{\sum_i w_i C_i(\alpha) I_i(\alpha_i)}{\sum_i w_i C_i(\alpha)}$$

This total interest value is then thresholded, and pairs of objects that have total interest values above the threshold are merged (if they are in the same field) or matched (if they are in different fields).

Another merging method is available in MODE, which can be used instead of, or along with, the fuzzy logic based merging just described. Recall that the convolved field is thresholded to produce the mask field. A second (lower) threshold can be specified so that objects that are separated at the higher threshold but joined at the lower threshold are merged.

20.2.4 Summary Statistics

Once MODE has been run, summary statistics are written to an output file. These files contain information about all single and cluster objects and their attributes. Total interest for object pairs is also output, as are percentiles of intensity inside the objects. The output file is in a simple flat ASCII tabular format (with one header line) and thus should be easily readable by just about any programming language, scripting language, or statistics package. Refer to Section 20.3.3 for lists of the statistics included in the MODE output files. Example scripts will be posted on the MET website in the future.

20.2.5 Multi-Variate MODE

Traditionally, MODE defines objects by smoothing and thresholding data from a single input field. MET version 10.1.0 extends MODE by adding the option to define objects using multiple input fields.

As described in Section 20.3.2, the **field** entry in the forecast and observation dictionaries define the input data to be processed. If **field** is defined as a dictionary, the traditional method for running MODE is invoked, where objects are defined using a single input field. If **field** is defined as an array of dictionaries, each specifying a different input field, then the multi-variate MODE logic is invoked and requires the **multivar_logic**

configuration entry to be set. Traditional MODE is run once for each input field to define objects for that field. Note that the object definition criteria can be defined separately for each field array entry. The objects from each input field are combined into forecast and observation data *super* objects

The **multivar_logic** configuration entry, described in Section 20.3.2, defines the boolean logic for combining objects from multiple fields into *super* objects. It can be defined once to apply to both the forecast and observation dictionaries if the field array lengths are the same, or defined separately within each dictionary. If defined separately within each dictionary, the field array lengths do not need to be the same for the forecast and observations. Note that the multi-variate MODE forecast and observation input fields and combination logic do not need to match.

The **multivar_intensity_compare_fcst** and **multivar_intensity_compare_obs** configuration entries, described in Section 20.3.2, define the field array indexes for which to optionally compare intensities for individual input fields when the input is masked to non-missing only inside the *super* objects and are required to be the same length. For example, if **multivar_intensity_compare_fcst = [1, 2];** and **multivar_intensity_compare_obs = [2, 3];**, then index 1 (2) of the forecast field array will be compared with index 2 (3) of the observation field array. If an intensity comparision is requested, the corresponding pair of fields (fcst and obs) are masked to non-missing inside the fcst and obs super objects, and traditional mode is run on that pair of masked inputs producing uniquely named outputs. If **multivar_intensity_compare_fcst** and **multivar_intensity_compare_obs** are empty, the forecast and observation *super* objects are written to NetCDF, text, and postscript output files in the standard mode output format, but with no intensity information.

When regridding to the FCST or OBS field (e.g. to_grid = FCST), the first field of the field array is used from the forecast and observation field dictionaries, respectively. All regridding is then done to that grid. Other regrid options described in *regrid* (page 53) can also be used as normal.

"file_type" can be set independently for each input in multivariate mode. If not set for an input, MET uses file names and file content to determine the type.

In multivariate mode, with quilt=**FALSE**, for all inputs the number of forecast and observation convolution radii and thresholds must all match. One configuration of MODE will be run for each group of settings in those lists.

In multivariate mode, with quilt=**TRUE**, for all inputs the number of forecast and observation convolution radii must match and the number of forecast and observation convolution thresholds must match. When each input has N radii and M thresholds, NxM configurations of MODE will be run.

When setting a threshold to a percentile, some choices require both an observation input and a forecast input. When this is the case, it's assumed the indices match, so for example if forecast input 1 has such a percentile setting, then observation input 1 will be used to compute the percentile. Percentiles in which this will happen are:

- SFP in an observation input. * The matching forecast input will be used to determine the threshold. e.g. ">SFP33.3" in the 2nd observation input means greater than 33.3-rd percentile of the 2nd forecast input will be used as the threshold for that observation input.
- SOP in a forecast input. * The matching observation input will be used to determine the threshold. e.g. ">SOP33.3" in the 2nd forecast input means greater than 33.3-rd percentile of the 2nd observation input will be used as the threshold for that forecast input.
- "==FBIAS" in an observation input. * e.g. "==FBIAS1" in an observation input to automatically de-bias the data, using a simple threshold in the matching forecast input. For example, when obser-

vation input 3 has "==FBIAS1", and forecast input 3 has ">5.0", MET applies the >5.0 threshold to the forecast and then chooses an observation threshold which results in a frequency bias of 1. The frequency bias can be any float value > 0.0.

- "==FBIAS" in a forecast input. * e.g. "==FBIAS1" in a forecast input to automatically de-bias the data, using a simple threshold in the matching observation input. For example, when forecast input 2 has "==FBIAS1", and observation input 2 has ">5.0", MET applies the >5.0 threshold to the observation and then chooses a forecast threshold which results in a frequency bias of 1. The frequency bias can be any float value > 0.0.

20.3 Practical Information

This section contains a description of how MODE can be configured and run. The MODE tool is used to perform a features-based verification of gridded model data using gridded observations. The input gridded model and observation datasets must be in one of the MET supported gridded file formats. If the input datasets are not already on a common grid, MODE can interpolate them to a common grid. The regrid option in the configuration file enables the user to specify the grid upon which the scores will be computed. The gridded analysis data may be based on observations, such as Stage II or Stage IV data for verifying accumulated precipitation, or a model analysis field may be used. However, users are cautioned that it is generally unwise to verify model output using an analysis field produced by the same model.

MODE provides the capability to select a single model variable/level from which to derive objects to be analyzed. MODE was developed and tested using accumulated precipitation. However, the code has been generalized to allow the use of any gridded model and observation field. Based on the options specified in the configuration file, MODE will define a set of simple objects in the model and observation fields. It will then compute an interest value for each pair of objects across the fields using a fuzzy engine approach. Those interest values are thresholded, and any pairs of objects above the threshold will be matched/merged. Through the configuration file, MODE offers a wide range of flexibility in how the objects are defined, processed, matched, and merged.

20.3.1 mode Usage

The usage statement for the MODE tool is listed below:

```
Usage: mode
       fcst_file
       obs_file
       config_file
       [-config_merge merge_config_file]
       [-outdir path]
       [-log file]
       [-v level]
       [-compress level]
```

The MODE tool has three required arguments and can accept several optional arguments.

20.3.1.1 Required Arguments for mode

1. The **fcst_file** argument indicates the gridded file containing the model field to be verified.
2. The **obs_file** argument indicates the gridded file containing the gridded observations to be used for the verification of the model.
3. The **config_file** argument indicates the name of the configuration file to be used. The contents of the configuration file are discussed below.

20.3.1.2 Optional Arguments for mode

4. The **-config_merge merge_config_file** option indicates the name of a second configuration file to be used when performing fuzzy engine merging by comparing the model or observation field to itself. The MODE tool provides the capability of performing merging within a single field by comparing the field to itself. Interest values are computed for each object and all of its neighbors. If an object and its neighbor have an interest value above some threshold, they are merged. The **merge_config_file** controls the settings of the fuzzy engine used to perform this merging step. If a **merge_config_file** is not provided, the configuration specified by the config_file in the previous argument will be used.
5. The **-outdir path** option indicates the directory where output files should be written.
6. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
7. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.
8. The **-compress level** option indicates the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. The value of "level" will override the default setting of 0 from the configuration file or the environment variable MET_NC_COMPRESS. Setting the compression level to 0 will make no compression for the NetCDF output. Lower number is for fast compression and higher number is for better compression.

An example of the MODE calling sequence is listed below:

Example 1:

```
mode sample_fcst.grb \
sample_obs.grb \
MODEConfig_grb
```

In Example 1, the MODE tool will verify the model data in the sample_fcst.grb GRIB file using the observations in the sample_obs.grb GRIB file applying the configuration options specified in the **MODEConfig_grb** file.

A second example of the MODE calling sequence is presented below:

Example 2:

```
mode sample_fcst.nc \
sample_obs.nc \
MODEConfig_nc
```

In Example 2, the MODE tool will verify the model data in the sample_fcst.nc NetCDF output of pcp_combine using the observations in the sample_obs.nc NetCDF output of pcp_combine, using the configuration options specified in the **MODEConfig_nc** file. Since the model and observation files contain only a single field of accumulated precipitation, the **MODEConfig_nc** file should specify that accumulated precipitation be verified.

20.3.2 mode Configuration File

The default configuration file for the MODE tool, **MODEConfig_default**, can be found in the installed *share/met/config* directory. Another version of the configuration file is provided in *scripts/config*. We encourage users to make a copy of the configuration files prior to modifying their contents. Descriptions of **MODEConfig_default** and the required variables for any MODE configuration file are also provided below. While the configuration file contains many entries, most users will only need to change a few for their use. Specific options are described in the following subsections.

A second default configuration file for the multivar MODE option, **MODEMultivarConfig_default**, is also found in the installed *share/met/config* directory. We encourage users to make a copy of this default configuration file when setting up a multivar configuration prior to modifying content. The two default config files **MODEConfig_default** and **MODEMultivarConfig_default** are similar, with **MODEMultivarConfig_default** having example multivar specific content.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
model          = "FCST";
desc           = "NA";
obtype         = "ANALYS";
regrid         = { ... }
met_data_dir   = "MET_BASE";
output_prefix  = "";
version        = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
grid_res = 4;
```

The **grid_res** entry is the nominal spacing for each grid square in kilometers. This entry is not used directly in the code, but subsequent entries in the configuration file are defined in terms of it. Therefore, setting this appropriately will help ensure that appropriate default values are used for these entries.

```
quilt = FALSE;
```

The **quilt** entry indicates whether all permutations of convolution radii and thresholds should be run.

- If **FALSE**, the number of forecast and observation convolution radii and thresholds must all match. One configuration of MODE will be run for each group of settings in those lists.
- If **TRUE**, the number of forecast and observation convolution radii must match and the number of forecast and observation convolution thresholds must match. For N radii and M thresholds, NxM configurations of MODE will be run.

```
multivar_logic = "#1 && #2 && #3";
```

The **multivar_logic** entry appears only in the **MODEMultivarConfig_default** file. This option applies to running multi-variate MODE by setting **field** to an array of dictionaries to define multiple input fields. Objects are defined separately for each input field based on the configuration settings specified for each field array entry. The **multivar_logic** entry is a string which defines how objects for each field are combined into a final *super* object. The objects for each field are referred to as '#N' where N is the N-th field array entry. The '&&' and '||' strings define intersection and union logic, respectively. For example, "#1 && #2" is the intersection of the objects from the first and second fields. "(#1 && #2) || #3" is the union of that intersection with the objects from the third field.

The **multivar_logic** entry is parsed separately from the **fcst** and **obs** dictionaries and can be defined differently in each. It does not require the same number of fields in the forecast and observation arrays.

```
multivar_intensity_compare_fcst = [1,2];
multivar_intensity_compare_obs = [2,3];
```

The **multivar_intensity_compare_fcst** and **multivar_intensity_compare_obs** entries appear only in the **MODEMultivarConfig_default** file. These entries define an index in the field arrays to be compared for forecast and observation intensities and must be the same length. For example, in the above example, forecast field 1 will be compared to observation field 2 for computing intensity attribute statistics. If the **multivar_intensity_compare_fcst** and **multivar_intensity_compare_obs** are empty, traditional mode output is created for the super objects, but with no intensity information.

```
multivar_name = "Super";
```

The **multivar_name** entry appears only in the **MODEMultivarConfig_default** file. This option is used only when the multivar option is enabled, and only when **multivar_intensity_compare_fcst** and **multivar_intensity_compare_obs** are empty. It can be thought of as an identifier for the multivariate super object. It shows up in output files names and content. It can be set separately for forecasts and observations or as a common value for both.

```
multivar_level = "LO";
```

The **multivar_level** entry appears only in the **MODEMultivarConfig_default** file. This option is used only when the multivar option is enabled, and only when **multivar_intensity_compare_fcst** and **multivar_intensity_compare_obs** are empty. It is the identifier for the multivariate super object as regards level. It shows up in output files names and content. If not set the default value is "NA". It can be set separately for forecasts and observations, or as a common value for both.

```
fcst = {
   field = {
      name = "APCP";
      level = "A03";
   }
   censor_thresh      = [];
   censor_val         = [];
   conv_radius        = 60.0/grid_res; // in grid squares
   conv_thresh        = >=5.0;
   vld_thresh         = 0.5;
   filter_attr_name   = [];
   filter_attr_thresh = [];
   merge_thresh       = >=1.25;
   merge_flag         = THRESH;
}
obs = fcst;
```

The **field** entries in the forecast and observation dictionaries specify the model and observation variables and level to be compared. See a more complete description of them in Section 5. In the above example, the forecast settings are copied into the observation dictionary using **obs = fcst;.**

When **field** is set to an array of dictionaries rather than a single one, the multi-variate MODE logic is invoked. Please see Section 20.2.5 for a description of that logic.

The **censor_thresh** and **censor_val** entries are used to censor the raw data as described in Section 5. Their functionality replaces the **raw_thresh** entry, which is deprecated in met-6.1. Prior to defining objects, it is recommended that the raw fields should be made to look similar to each other. For example, if the model only predicts values for a variable above some threshold, the observations should be thresholded at that same level. The censor thresholds can be specified using symbols. By default, no censor thresholding is applied.

The **conv_radius** entry defines the radius of the circular convolution applied to smooth the raw fields. The radii are specified in terms of grid units. The default convolution radii are defined in terms of the previously defined **grid_res** entry. Multiple convolution radii may be specified as an array (e.g. **conv_radius = [5, 10, 15];**).

The **conv_thresh** entry specifies the threshold values to be applied to the convolved field to define objects. By default, objects are defined using a convolution threshold of 5.0. Multiple convolution thresholds may be specified as an array (e.g. **conv_thresh = [>=5.0, >=10.0, >=15.0];**).

Multiple convolution radii and thresholds are processed using the logic defined by the **quilt** entry. The logic specific to multivariate mode is described in the multivariate mode section above.

The **vld_thresh** entry must be set between 0 and 1. When performing the circular convolution step if the proportion of bad data values in the convolution area is greater than or equal to this threshold, the resulting convolved value will be bad data. If the proportion is less than this threshold, the convolution will be performed on only the valid data. By default, the **vld_thresh** is set to 0.5.

The **filter_attr_name** and **filter_attr_thresh** entries are arrays of the same length which specify object filtering criteria. By default, no object filtering criteria is defined.

The **filter_attr_name** entry is an array of strings specifying the MODE output header column names for the object attributes of interest, such as **AREA, LENGTH, WIDTH**, and **INTENSITY_50**. In addition, **ASPECT_RATIO** specifies the aspect ratio (width/length), **INTENSITY_101** specifies the mean intensity value, and **INTENSITY_102** specifies the sum of the intensity values.

The **filter_attr_thresh** entry is an array of thresholds for these object attributes. Any simple objects not meeting all of the filtering criteria are discarded.

Note that the **area_thresh** and **inten_perc_thresh** entries from earlier versions of MODE are replaced by these options and are now deprecated.

The **merge_thresh** entry is used to define larger objects for use in merging the original objects. It defines the threshold value used in the double thresholding merging technique. Note that in order to use this merging technique, it must be requested for both the forecast and observation fields. These thresholds should be chosen to define larger objects that fully contain the originally defined objects. For example, for objects defined as >=5.0, a merge threshold of >=2.5 will define larger objects that fully contain the original objects. Any two original objects contained within the same larger object will be merged. By default, the merge thresholds are set to be greater than or equal to 1.25. Multiple merge thresholds may be specified as an array (e.g. **merge_thresh = [>=1.0, >=2.0, >=3.0];**). The number of **merge_thresh** entries must match the number of **conv_thresh** entries.

The **merge_flag** entry controls what type of merging techniques will be applied to the objects defined in each field.

- **NONE** indicates that no merging should be applied.
- **THRESH** indicates that the double thresholding merging technique should be applied.
- **ENGINE** indicates that objects in each field should be merged by comparing the objects to themselves using a fuzzy engine approach.
- **BOTH** indicates that both techniques should be used.

By default, the double thresholding **THRESH** merging technique is applied in single variable mode. The merging defaults to **NONE** with multivariate mode.

```
mask_missing_flag = NONE;
```

The **mask_missing_flag** entry specifies how missing data in the raw model and observation fields will be treated.

- **NONE** indicates no additional processing is to be done.

- **FCST** indicates missing data in the observation field should be used to mask the forecast field.
- **OBS** indicates missing data in the forecast field should be used to mask the observation field.
- **BOTH** indicates masking should be performed in both directions (i.e., mask the forecast field with the observation field and vice-versa).

Prior to defining objects, it is recommended that the raw fields be made to look similar to each other by assigning a value of BOTH to this parameter. However, by default no masking is performed.

```
match_flag = MERGE_BOTH;
```

The **match_flag** entry controls how matching will be performed when comparing objects from the forecast field to objects from the observation field. An interest value is computed for each possible pair of forecast/observation objects. The interest values are then thresholded to define which objects match. If two objects in one field happen to match the same object in the other field, then those two objects could be merged. The **match_flag** entry controls what type of merging is allowed in this context.

- **NONE** indicates that no matching should be performed between the fields at all.
- **MERGE_BOTH** indicates that additional merging is allowed in both fields.
- **MERGE_FCST** indicates that additional merging is allowed only in the forecast field.
- **NO_MERGE** indicates that no additional merging is allowed in either field, meaning that each object will match at most one object in the other field.

By default, additional merging is allowed in both fields.

```
max_centroid_dist = 800/grid_res;
```

Computing the attributes for all possible pairs of objects can take some time depending on the numbers of objects. The **max_centroid_dist** entry is used to specify how far apart objects should be in order to conclude that they have no chance of matching. No pairwise attributes are computed for pairs of objects whose centroids are farther away than this distance, defined in terms of grid units. Setting this entry to a reasonable value will improve the execution time of the MODE tool. By default, the maximum centroid distance is defined in terms of the previously defined **grid_res** entry.

```
mask = {
   grid = "";
   grid_flag = NONE; // Apply to NONE, FCST, OBS, or BOTH
   poly = "";
   poly_flag = NONE; // Apply to NONE, FCST, OBS, or BOTH
}
```

Defining a **grid** and **poly** masking region is described in Section 5. Applying a masking region when running MODE sets all grid points falling outside of that region to missing data, effectively limiting the area of which objects should be defined.

The **grid_flag** and **poly_flag** entries specify how the grid and polyline masking should be applied:

- **NONE** indicates that the masking grid should not be applied.
- **FCST** indicates that the masking grid should be applied to the forecast field.
- **OBS** indicates that the masking grid should be applied to the observation field.
- **BOTH** indicates that the masking grid should be applied to both fields.

By default, no masking grid or polyline is applied.

```
weight = {
   centroid_dist    = 2.0;
   boundary_dist    = 4.0;
   convex_hull_dist = 0.0;
   angle_diff       = 1.0;
   aspect_diff      = 0.0;
   area_ratio       = 1.0;
   int_area_ratio   = 2.0;
   curvature_ratio  = 0.0;
   complexity_ratio = 0.0;
   inten_perc_ratio = 0.0;
   inten_perc_value = 50;
}
```

The **weight** entries listed above control how much weight is assigned to each pairwise attribute when computing a total interest value for object pairs. The weights listed above correspond to the **centroid distance** between the objects, the **boundary distance** (or minimum distance), the **convex hull distance** (or minimum distance between the convex hulls of the objects), the **orientation angle** difference, the **aspect ratio** difference, the **object area ratio** (minimum area divided by maximum area), the **intersection divided by the minimum object area ratio**, the **curvature ratio**, the **complexity ratio**, and the **intensity ratio**. The weights need not sum to any particular value. When the total interest value is computed, the weighted sum is normalized by the sum of the weights listed above.

The **inten_perc_value** entry corresponds to the **inten_perc_ratio**. The **inten_perc_value** should be set between 0 and 102 to define which percentile of intensity should be compared for pairs of objects. 101 and 102 specify the intensity mean and sum, respectively. By default, the 50th percentile, or median value, is chosen.

```
interest_function = {
   centroid_dist      = ( ... );
   boundary_dist      = ( ... );
   convex_hull_dist   = ( ... );
   angle_diff         = ( ... );
   aspect_diff        = ( ... );
   corner             = 0.8;
   ratio_if           = ( ( 0.0, 0.0 )
                          ( corner, 1.0 )
```

(continues on next page)

(continued from previous page)

```
                        ( 1.0, 1.0 ) );
   area_ratio         = ratio_if;
   int_area_ratio     = ( ... );
   curvature_ratio    = ratio_if;
   complexity_ratio   = ratio_if;
   inten_perc_ratio   = ratio_if;
}
```

The interest function entries listed above define which values are of interest for each pairwise attribute measured. Each interest function is defined as a piecewise linear function by specifying the corner points of its graph. The range of each function must be within **0** and **1**. Including (x, y) points with y-values outside this range results in a runtime error. See Section 21.2 for how interest values are used by the fuzzy logic engine. By default, many of these functions are defined in terms of the previously defined **grid_res** entry.

```
total_interest_thresh = 0.7;
```

The **total_interest_thresh** entry should be set between **0** and **1**. This threshold is applied to the total interest values computed for each pair of objects. Object pairs that have an interest value that is above this threshold will be matched, while those with an interest value that is below this threshold will remain unmatched. Increasing the threshold will decrease the number of matches while decreasing the threshold will increase the number of matches. By default, the total interest threshold is set to 0.7.

```
print_interest_thresh = 0.0;
```

The **print_interest_thresh** entry determines which pairs of object attributes will be written to the output object attribute ASCII file. The user may choose to set the **print_interest_thresh** to the same value as the **total_interest_thresh**, meaning that only object pairs that actually match are written to the output file. By default, the print interest threshold is set to zero, meaning that all object pair attributes will be written as long as the distance between the object centroids is less than the **max_centroid_dist** entry.

```
fcst_raw_plot = {
   color_table = "MET_BASE/colortables/met_default.ctable";
   plot_min = 0.0;
   plot_max = 0.0;
}
obs_raw_plot = {
   color_table = "MET_BASE/colortables/met_default.ctable";
   plot_min = 0.0;
   plot_max = 0.0;
}
object_plot = {
   color_table = "MET_BASE/colortables/mode_obj.ctable";
}
```

Specifying dictionaries to define the **color_table, plot_min**, and **plot_max** entries are described in Section 5.

The MODE tool generates a color bar to represent the contents of the colortable that was used to plot a field of data. The number of entries in the color bar matches the number of entries in the color table. The values defined for each color in the color table are also plotted next to the color bar.

```
plot_valid_flag = FALSE;
```

When applied, the **plot_valid_flag entry** indicates that only the region containing valid data after masking is applied should be plotted.

- **FALSE** indicates the entire domain should be plotted.
- **TRUE** indicates only the region containing valid data after masking should be plotted.

The default value of this flag is FALSE.

```
plot_gcarc_flag = FALSE;
```

When applied, the **plot_gcarc_flag** entry indicates that the edges of polylines should be plotted using great circle arcs as opposed to straight lines in the grid. The default value of this flag is FALSE.

```
ps_plot_flag  = TRUE;
ct_stats_flag = TRUE;
```

These flags can be set to TRUE or FALSE to produce additional output, in the form of PostScript plots and contingency table counts and statistics, respectively.

```
nc_pairs_flag = {
   latlon     = TRUE;
   raw        = TRUE;
   object_raw = TRUE;
   object_id  = TRUE;
   cluster_id = TRUE;
   polylines  = TRUE;
}
```

Each component of the pairs information in the NetCDF file can be turned on or off. The old syntax is still supported: **TRUE** means accept the defaults, **FALSE** means no NetCDF output is generated. NetCDF output can also be turned off by setting all the individual dictionary flags to false.

The nc_pairs_flag is described in Section 13.3.2

```
shift_right = 0;
```

When MODE is run on global grids, this parameter specifies how many grid squares to shift the grid to the right. MODE does not currently connect objects from one side of a global grid to the other, potentially causing objects straddling the "cut" longitude to be separated into two objects. Shifting the grid by integer number of grid units enables the user to control where that longitude cut line occurs.

20.3.3 mode Output

MODE produces output in ASCII, NetCDF, and PostScript formats.

ASCII output

The MODE tool creates two ASCII output files. The first ASCII file contains contingency table counts and statistics for comparing the forecast and observation fields. This file consists of 4 lines. The first is a header line containing column names. The second line contains data comparing the two raw fields after any masking of bad data or based on a grid or lat/lon polygon has been applied. The third contains data comparing the two fields after any raw thresholds have been applied. The fourth, and last, line contains data comparing the derived object fields scored using traditional measures.

Table 20.1: Format of MODE CTS output file.

Column Number	CTS Column Name	Description
1	VERSION	Version number
2	MODEL	User-provided text string giving model name
3	N_VALID	Number of valid data points
4	GRID_RES	User-provided nominal grid resolution
5	DESC	User-provided text string describing the verification task
6	FCST_LEAD	Forecast lead time in HHMMSS format
7	FCST_VALID	Forecast valid start time in YYYYMMDD_HHMMSS format
8	FCST_ACCUM	Forecast accumulation time in HHMMSS format
9	OBS_LEAD	Observation lead time in HHMMSS format; when field2 is actually an obser
10	OBS_VALID	Observation valid start time in YYYYMMDD_HHMMSS format
11	OBS_ACCUM	Observation accumulation time in HHMMSS format
12	FCST_RAD	Forecast convolution radius in grid units
13	FCST_THR	Forecast convolution threshold
14	OBS_RAD	Observation convolution radius in grid units
15	OBS_THR	Observation convolution threshold
16	FCST_VAR	Forecast variable
17	FCST_UNITS	Units for model variable
18	FCST_LEV	Forecast vertical level
19	OBS_VAR	Observation variable
20	OBS_UNITS	Units for observation variable
21	OBS_LEV	Observation vertical level
22	OBTYPE	User-provided observation type
23	FIELD	Field type for this line:* RAW for the raw input fields * OBJECT for the reso

Table 20.1 – continued from previous page

Column Number	CTS Column Name	Description
24	TOTAL	Total number of matched pairs
25	FY_OY	Number of hits (forecast yes and observation yes)
26	FY_ON	Number of false alarms (forecast yes and observation no)
27	FN_OY	Number of misses (forecast no and observation yes)
28	FN_ON	Number of correct negatives (forecast no and observation no)
29	BASER	Base rate
30	FMEAN	Forecast mean
31	ACC	Accuracy
32	FBIAS	Frequency Bias
33	PODY	Probability of detecting yes
34	PODN	Probability of detecting no
35	POFD	Probability of false detection
36	FAR	False alarm ratio
37	CSI	Critical Success Index
38	GSS	Gilbert Skill Score
39	HK	Hanssen-Kuipers Discriminant
40	HSS	Heidke Skill Score
41	ODDS	Odds Ratio
42	LODDS	Logarithm of the Odds Ratio
43	ORSS	Odds Ratio Skill Score
44	EDS	Extreme Dependency Score
45	SEDS	Symmetric Extreme Dependency Score
46	EDI	Extreme Dependency Index
47	SEDI	Symmetric Extremal Dependency Index
48	BAGSS	Bias-Adjusted Gilbert Skill Score

This first file uses the following naming convention:

mode_PREFIX_FCST_VAR_LVL_vs_OBS_VAR_LVL_HHMMSSL_YYYYMMDD_HHMMSSV_HHMMSSA_cts.txt

where *PREFIX* indicates the user-defined output prefix, *FCST_VAR_LVL* is the forecast variable and vertical level being used, *OBS_VAR_LVL* is the observation variable and vertical level being used, *HHMMSSL* indicates the forecast lead time, *YYYYMMDD_HHMMSSV* indicates the forecast valid time, and *HHMMSSA* indicates the accumulation period. The {tt cts} string stands for contingency table statistics. The generation of this file can be disabled using the *ct_stats_flag* option in the configuration file. This CTS output file differs somewhat from the CTS output of the Point-Stat and Grid-Stat tools. The columns of this output file are summarized in Table 20.1.

The second ASCII file the MODE tool generates contains all of the attributes for simple objects, the merged cluster objects, and pairs of objects. Each line in this file contains the same number of columns, though those columns not applicable to a given line contain fill data. The first row of every MODE object attribute file is a header containing the column names. The number of lines in this file depends on the number of objects defined. This file contains lines of 6 types that are indicated by the contents of the **OBJECT_ID** column. The **OBJECT_ID** can take the following 6 forms: **FNN, ONN, FNNN_ONNN, CFNNN, CONNN, CFNNN_CONNN**. In each case, **NNN** is a three-digit number indicating the object index. While all lines have the first 18 header columns in common, these 6 forms for **OBJECT_ID** can be divided into two types - one

for single objects and one for pairs of objects. The single object lines **(FNN, ONN, CFNNN**, and **CONNN)** contain valid data in columns 19-39 and fill data in columns 40-51. The object pair lines **(FNNN_ONNN** and **CFNNN_CONNN)** contain valid data in columns 40-51 and fill data in columns 19-39.

These object identifiers are described in Table 20.2.

Table 20.2: Object identifier descriptions for MODE object attribute output file.

Object identifier (object_id)	Valid Data Columns	Description of valid data
FNNN, ONNN	1-18,19-39	Attributes for simple forecast, observation objects
FNNN_ ONNN	1-18, 40-51	Attributes for pairs of simple forecast and observation objects
CFNNN, CONNN	1-18,19-39	Attributes for merged cluster objects in forecast, observation fields
CFNNN_ CONNN	1-18, 40-51	Attributes for pairs of forecast and observation cluster objects

A note on terminology: a cluster (referred to as "composite" in earlier versions) object need not necessarily consist of more than one simple object. A cluster object is by definition any set of one or more objects in one field which match a set of one or more objects in the other field. When a single simple forecast object matches a single simple observation object, they are each considered to be cluster objects as well.

The contents of the columns in this ASCII file are summarized in Table 20.3.

Ta

Column Number	Object Column Name	Description
1	VERSION	Version number
2	MODEL	User-provided text string designating model name
3	N_VALID	Number of valid data points
4	GRID_RES	User-provided nominal grid resolution
5	DESC	User-provided text string describing the verification task
6	FCST_LEAD	Forecast lead time in HHMMSS format
7	FCST_VALID	Forecast valid start time in YYYYMMDD_HHMMSS format
8	FCST_ACCUM	Forecast accumulation time in HHMMSS format
9	OBS_LEAD	Observation lead time in HHMMSS format; when field2 is a
10	OBS_VALID	Observation valid start time in YYYYMMDD_HHMMSS form
11	OBS_ACCUM	Observation accumulation time in HHMMSS format
12	FCST_RAD	Forecast convolution radius in grid squares
13	FCST_THR	Forecast convolution threshold
14	OBS_RAD	Observation convolution radius in grid squares
15	OBS_THR	Observation convolution threshold
16	FCST_VAR	Forecast variable
17	FCST_UNITS	Units for forecast variable
18	FCST_LEV	Forecast vertical level
19	OBS_VAR	Observation variable

Column Number	Object Column Name	Description
20	OBS_UNITS	Units for observation variable
21	OBS_LEV	Observation vertical level
22	OBTYPE	User-provided observation type
23	OBJECT_ID	Object numbered from 1 to the number of objects in each fie
24	OBJECT_CAT	Object category indicating to which cluster object it belongs
25-26	CENTROID_X, _Y	Location of the centroid (in grid units)
27-28	CENTROID_LAT, _LON	Location of the centroid (in lat/lon degrees)
29	AXIS_ANG	Object axis angle (in degrees)
30	LENGTH	Length of the enclosing rectangle (in grid units)
31	WIDTH	Width of the enclosing rectangle (in grid units)
32	AREA	Object area (in grid squares)
33	AREA_THRESH	Area of the object containing data values in the raw field tha
34	CURVATURE	Radius of curvature of the object defined in terms of third or
35-36	CURVATURE_X, _Y	Center of curvature (in grid coordinates)
37	COMPLEXITY	Ratio of the difference between the area of an object and the
38-42	INTENSITY_10, _25, _50, _75, _90	10th, 25th, 50th, 75th, and 90th percentiles of intensity of t
43	INTENSITY_NN	The percentile of intensity chosen for use in the PERCENTIL
44	INTENSITY_SUM	Sum of the intensities of the raw field within the object (var
45	CENTROID_DIST	Distance between two objects centroids (in grid units)
46	BOUNDARY_DIST	Minimum distance between the boundaries of two objects (i
47	CONVEX_HULL _DIST	Minimum distance between the convex hulls of two objects
48	ANGLE_DIFF	Difference between the axis angles of two objects (in degree
49	ASPECT_DIFF	Absolute value of the difference between the aspect ratios o
50	AREA_RATIO	The forecast object area divided by the observation object ar
51	INTERSECTION _AREA	Intersection area of two objects (in grid squares)
52	UNION_AREA	Union area of two objects (in grid squares)
53	SYMMETRIC_DIFF	Symmetric difference of two objects (in grid squares)
54	INTERSECTION _OVER_AREA	Ratio of intersection area to the lesser of the forecast and ob
55	CURVATURE _RATIO	Ratio of the curvature of two objects defined as the lesser of
56	COMPLEXITY _RATIO	Ratio of complexities of two objects defined as the lesser of
57	PERCENTILE _INTENSITY _RATIO	Ratio of the nth percentile (INTENSITY_NN column) of inte
58	INTEREST	Total interest value computed for a pair of simple objects (u

NetCDF Output

The MODE tool creates a NetCDF output file containing the object fields that are defined. The NetCDF file contains gridded fields including indices for the simple forecast objects, indices for the simple observation objects, indices for the matched cluster forecast objects, and indices for the matched cluster observation objects. The NetCDF file also contains lat/lon and x/y data for the vertices of the polygons for the boundaries of the simple forecast and observation objects. The generation of this file can be disabled using the **nc_pairs_flag** configuration file option.

The dimensions and variables included in the mode NetCDF files are described in Table 20.4 and Table 20.5.

Table 20.4: NetCDF dimensions for MODE output.

NetCDF Dimension	Description
lat	Dimension of the latitude (i.e. Number of grid points in the North-South direction)
lon	Dimension of the longitude (i.e. Number of grid points in the East-West direction)
fcst_thresh_length	Number of thresholds applied to the forecast
obs_thresh_length	Number of thresholds applied to the observations
fcst_simp	Number of simple forecast objects
fcst_simp_bdy	Number of points used to define the boundaries of all of the simple forecast objects
fcst_simp_hull	Number of points used to define the hull of all of the simple forecast objects
obs_simp	Number of simple observation objects
obs_simp_bdy	Number of points used to define the boundaries of all of the simple observation objects
obs_simp_hull	Number of points used to define the hull of all of the simple observation objects
fcst_clus	Number of forecast clusters
fcst_clus_hull	Number of points used to define the hull of all of the cluster forecast objects
obs_clus	Number of observed clusters
obs_clus_hull	Number of points used to define the hull of all of the cluster observation objects

Table 20.5: Variables contained in MODE NetCDF output.

NetCDF Variable	Dimension	Description	Data Type
lat	lat, lon	Latitude	Float
lon	lat, lon	Longitude	Float
fcst_raw	lat, lon	Forecast raw values	Float
fcst_obj_raw	lat, lon	Forecast Object Raw Values	Float
fcst_obj_id	lat, lon	Simple forecast object id number for each grid point	Integer
fcst_clus_id	lat, lon	Cluster forecast object id number for each grid point	Integer
obs_raw	lat, lon	Observation Raw Values	Float
obs_obj_raw	lat, lon	Observation Object Raw Values	Float
obs_obj_id	-	Simple observation object id number for each grid point	Integer
obs_clus_id	-	Cluster observation object id number for each grid point	Integer
fcst_conv_radius	-	Forecast convolution radius	Integer
obs_conv_radius	-	Observation convolution radius	Integer
fcst_conv _threshold	-	Forecast convolution threshold	String
obs_conv _threshold	-	Observation convolution threshold	String
n_fcst_simp	-	Number of simple forecast objects	Integer
n_obs_simp	-	Number of simple observation objects	Integer
n_clus	-	Number of cluster objects	Integer
fcst_simp_bdy _start	fcst_simp	Forecast Simple Boundary Starting Index	Integer
fcst_simp_bdy _npts	fcst_simp	Number of Forecast Simple Boundary Points	Integer
fcst_simp_bdy _lat	fcst_simp_bdy	Forecast Simple Boundary Latitude	Float
fcst_simp_bdy _lon	fcst_simp_bdy	Forecast Simple Boundary Longitude	Float
fcst_simp_bdy_x	fcst_simp_bdy	Forecast Simple Boundary X-Coordinate	Integer

continues on next page

Table 20.5 – continued from previous page

NetCDF Variable	Dimension	Description	Data Type
fcst_simp_bdy_y	fcst_simp_bdy	Forecast Simple Boundary Y-Coordinate	Integer
fcst_simp_hull _start	fcst_simp	Forecast Simple Convex Hull Starting Index	Integer
fcst_simp_hull _npts	fcst_simp	Number of Forecast Simple Convex Hull Points	Integer
fcst_simp_hull _lat	fcst_simp_hull	Forecast Simple Convex Hull Point Latitude	Float
fcst_simp_hull _lon	fcst_simp_hull	Forecast Simple Convex Hull Point Longitude	Float
fcst_simp_hull_x	fcst_simp_hull	Forecast Simple Convex Hull Point X-Coordinate	Integer
fcst_simp_hull_y	fcst_simp_hull	Forecast Simple Convex Hull Point Y-Coordinate	Integer
obs_simp_bdy _start	obs_simp	Observation Simple Boundary Starting Index	Integer
obs_simp_bdy _npts	obs_simp	Number of Observation Simple Boundary Points	Integer
obs_simp_bdy _lat	obs_simp_bdy	Observation Simple Boundary Point Latitude	Float
obs_simp_bdy _lon	obs_simp_bdy	Observation Simple Boundary Point Longitude	Float
obs_simp_bdy_x	obs_simp_bdy	Observation Simple Boundary Point X-Coordinate	Integer
obs_simp_bdy_y	obs_simp_bdy	Observation Simple Boundary Point Y-Coordinate	Integer
obs_simp_hull _start	obs_simp	Observation Simple Convex Hull Starting Index	Integer
obs_simp_hull _npts	obs_simp	Number of Observation Simple Convex Hull Points	Integer
obs_simp_hull _lat	obs_simp_hull	Observation Simple Convex Hull Point Latitude	Float
obs_simp_hull _lon	obs_simp_hull	Observation Simple Convex Hull Point Longitude	Float
obs_simp_hull_x	obs_simp_hull	Observation Simple Convex Hull Point X-Coordinate	Integer
obs_simp_hull_y	obs_simp_hull	Observation Simple Convex Hull Point Y-Coordinate	Integer
fcst_clus_hull _start	fcst_clus	Forecast Cluster Convex Hull Starting Index	Integer
fcst_clus_hull _npts	fcst_clus	Number of Forecast Cluster Convex Hull Points	Integer
fcst_clus_hull _lat	fcst_clus_hull	Forecast Cluster Convex Hull Point Latitude	Float
fcst_clus_hull _lon	fcst_clus_hull	Forecast Cluster Convex Hull Point Longitude	Float
fcst_clus_hull_x	fcst_clus_hull	Forecast Cluster Convex Hull Point X-Coordinate	Integer
fcst_clus_hull_y	fcst_clus_hull	Forecast Cluster Convex Hull Point Y-Coordinate	Integer
obs_clus_hull _start	obs_clus	Observation Cluster Convex Hull Starting Index	Integer
obs_clus_hull _npts	obs_clus	Number of Observation Cluster Convex Hull Points	Integer
obs_clus_hull _lat	obs_clus_hull	Observation Cluster Convex Hull Point Latitude	Float
obs_clus_hull _lon	obs_clus_hull	Observation Cluster Convex Hull Point Longitude	Float
obs_clus_hull_x	obs_clus_hull	Observation Cluster Convex Hull Point X-Coordinate	Integer
obs_clus_hull_y	obs_clus_hull	Observation Cluster Convex Hull Point Y-Coordinate	Integer

Postscript File

Lastly, the MODE tool creates a PostScript plot summarizing the features-based approach used in the verification. The PostScript plot is generated using internal libraries and does not depend on an external plotting package. The generation of this PostScript output can be disabled using the **ps_plot_flag** configuration file option.

The PostScript plot will contain 5 summary pages at a minimum, but the number of pages will depend on the merging options chosen. Additional pages will be created if merging is performed using the double thresholding or fuzzy engine merging techniques for the forecast and/or observation fields. Examples of the PostScript plots can be obtained by running the example cases provided with the MET tarball.

The first page of PostScript output contains a great deal of summary information. Six tiles of images provide thumbnail images of the raw fields, matched/merged object fields, and object index fields for the forecast

and observation grids. In the matched/merged object fields, matching colors of objects across fields indicate that the corresponding objects match, while within a single field, black outlines indicate merging. Note that objects that are colored royal blue are unmatched. Along the bottom of the page, the criteria used for object definition and matching/merging are listed. Along the right side of the page, total interest values for pairs of simple objects are listed in sorted order. The numbers in this list correspond to the object indices shown in the object index plots.

The second and third pages of the PostScript output file display enlargements of the forecast and observation raw and object fields, respectively. The fourth page displays the forecast object with the outlines of the observation objects overlaid, and vice versa. The fifth page contains summary information about the pairs of matched cluster objects.

If the double threshold merging or the fuzzy engine merging techniques have been applied, the output from those steps is summarized on additional pages.

Chapter 21

MODE-Analysis Tool

21.1 Introduction

Users may wish to summarize multiple ASCII files produced by MODE across many cases. The MODE output files contain many output columns making it very difficult to interpret the results by simply browsing the files. Furthermore, for particular applications some data fields in the MODE output files may not be of interest. The MODE-Analysis tool provides a simple way to compute basic summary statistics and filtering capabilities for these files. Users who are not proficient at writing scripts can use the tool directly, and even those using their own scripts can use this tool as a filter, to extract only the MODE output lines that are relevant for their application.

21.2 Scientific and Statistical Aspects

The MODE-Analysis tool operates in two modes, called "summary" and "bycase". In summary mode, the user specifies on the command line the MODE output columns of interest as well as filtering criteria that determine which input lines should be used. For example, a user may be interested in forecast object areas, but only if the object was matched, and only if the object centroid is inside a particular region. The summary statistics generated for each specified column of data are the minimum, maximum, mean, standard deviation, and the 10th, 25th, 50th, 75th and 90th percentiles. In addition, the user may specify a "dump"' file: the individual MODE lines used to produce the statistics will be written to this file. This option provides the user with a filtering capability. The dump file will consist only of lines that match the specified criteria.

The other option for operating the analysis tool is "bycase". Given initial and final values for forecast lead time, the tool will output, for each valid time in the interval, the matched area, unmatched area, and the number of forecast and observed objects that were matched or unmatched. For the areas, the user can specify forecast or observed objects, and also simple or cluster objects. A dump file may also be specified in this mode.

21.3 Practical Information

The MODE-Analysis tool reads lines from MODE ASCII output files and applies filtering and computes basic statistics on the object attribute values. For each job type, filter parameters can be set to determine which MODE output lines are used. The following sections describe the mode_analysis usage statement, required arguments, and optional arguments.

21.3.1 mode_analysis Usage

The usage statement for the MODE-Analysis tool is shown below:

```
Usage: mode_analysis
       -lookin path
       -summary | -bycase
       [-column name]
       [-dump_row filename]
       [-out filename]
       [-log file]
       [-v level]
       [-help]
       [MODE FILE LIST]
       [-config config_file] | [MODE LINE OPTIONS]
```

The MODE-Analysis tool has two required arguments and can accept several optional arguments.

21.3.1.1 Required Arguments for mode_analysis:

1. The **-lookin path** specifies the name of a specific STAT file (any file ending in .stat) or the name of a directory where the Stat-Analysis tool will search for STAT files. This option may be used multiple times to specify multiple locations.
2. The MODE-Analysis tool can perform two basic types of jobs **-summary** or **-bycase**. Exactly one of these job types must be specified.

Specifying **-summary** will produce summary statistics for the MODE output column specified. For this job type, a column name (or column number) must be specified using the **-column** option. Column names are not case sensitive. The column names are the same as described in Section 20.3.3. More information about this option is provided in subsequent sections.

Specifying **-bycase** will produce a table of metrics for each case undergoing analysis. Any columns specified are ignored for this option.

21.3.1.2 Optional Arguments for mode_analysis

3. The mode_analysis options are described in the following section. These are divided into sub-sections describing the analysis options and mode line options.

21.3.1.3 Analysis Options

The general analysis options described below provide a way for the user to indicate configuration files to be used, where to write lines used to perform the analysis, and over which fields to generate statistics.

```
-config filename
```

This option gives the name of a configuration file to be read. The contents of the configuration file are described in Section 21.3.2.

```
-dump_row filename
```

Any MODE lines kept from the input files are written to *filename*.

```
-column column
```

Specifies which columns in the MODE output files to generate statistics for. Fields may be indicated by name (case insensitive) or column number (beginning at one). This option can be repeated to specify multiple columns.

21.3.1.4 MODE Command Line Options

MODE command line options are used to create filters that determine which of the MODE output lines that are read in, are kept. The MODE line options are numerous. They fall into seven categories: toggles, multiple set string options, multiple set integer options, integer max/min options, date/time max/min options, floating-point max/min options, and miscellaneous options. These options are described here.

21.3.1.5 Toggles

The MODE line options described in this section are shown in pairs. These toggles represent parameters that can have only one (or none) of two values. Any of these toggles may be left unspecified. However, if neither option for each toggle is indicated, the analysis will produce results that combine data from both toggles. This may produce unintended results.

```
-fcst | -obs
```

This toggle indicates whether forecast or observed lines should be used for analysis.

```
-single | -pair
```

This toggle indicates whether single object or object pair lines should be used.

```
-simple | -cluster
```

This toggle indicates whether simple object or cluster object lines should be used.

```
-matched | -unmatched
```

This toggle indicates whether matched or unmatched object lines should be used.

21.3.1.6 Multiple-Set String Options

The following options set various string attributes. They can be set multiple times on the command line but must be separated by spaces. Each of these options must be indicated as a string. String values that include spaces may be used by enclosing the string in quotation marks.

```
-model value
```

This option specifies which model to use; value must be a string.

```
-fcst_thr value
-obs_thr  value
```

These two options specify thresholds for forecast and observation objects to be used in the analysis, respectively.

```
-fcst_var value
-obs_var  value
```

These options indicate the names of variables to be used in the analysis for forecast and observed fields.

```
-fcst_units value
-obs_units  value
```

These options indicate the units to be used in the analysis for forecast and observed fields.

```
-fcst_lev value
-obs_lev  value
```

These options indicate vertical levels for forecast and observed fields to be used in the analysis.

21.3.1.7 Multiple-Set Integer Options

The following options set various integer attributes. They can be set multiple times on the command line but must be separated by spaces. Each of the following options may only be indicated as an integer.

```
-fcst_lead value
-obs_lead  value
```

These options are integers of the form HH[MMSS] specifying an (hour-minute-second) lead time.

```
-fcst_accum value
-obs_accum  value
```

These options are integers of the form HHMMSS specifying an (hour-minute-second) accumulation time.

```
-fcst_rad value
-obs_rad  value
```

These options indicate the convolution radius used for forecast or observed objects, respectively.

21.3.1.8 Integer Max/Min Options

These options set limits on various integer attributes. Leaving a maximum value unset means no upper limit is imposed on the value of the attribute. The option works similarly for minimum values.

```
-area_min value
-area_max value
```

These options are used to indicate minimum/maximum values for the area attribute to be used in the analysis.

```
-area_filter_min value
-area_filter_max value
```

These options are used to indicate minimum/maximum values accepted for the area filter. The area filter refers to the number of non-zero values of the raw data found within the object.

```
-area_thresh_min value
-area_thresh_max value
```

These options are used to indicate minimum/maximum values accepted for the area thresh. The area thresh refers to the number of values of the raw data found within the object that meet the object definition threshold criteria used.

```
-intersection_area_min value
-intersection_area_max value
```

These options refer to the minimum/maximum values accepted for the intersection area attribute.

```
-union_area_min value
-union_area_max value
```

These options refer to the minimum/maximum union area values accepted for analysis.

```
-symmetric_diff_min value
-symmetric_diff_max value
```

These options refer to the minimum/maximum values for symmetric difference for objects to be used in the analysis.

21.3.1.9 Date/Time Max/Min Options

These options set limits on various date/time attributes. The values can be specified in one of three ways:

First, the options may be indicated by a string of the form YYYYMMDD_HHMMSS. This specifies a complete calendar date and time.

Second, they may be indicated by a string of the form YYYYMMDD_HH. Here, the minutes and seconds are assumed to be zero.

The third way of indicating date/time attributes is by a string of the form YYYYMMDD. Here, hours, minutes and seconds are assumed to be zero.

```
-fcst_valid_min YYYYMMDD[_HH[MMSS]]
-fcst_valid_max YYYYMMDD[_HH[MMSS]]
-obs_valid_min  YYYYMMDD[_HH[MMSS]]
-obs_valid_max  YYYYMMDD[_HH[MMSS]]
```

These options indicate minimum/maximum values for the forecast and observation valid times.

```
-fcst_init_min YYYYMMDD[_HH[MMSS]]
-fcst_init_max YYYYMMDD[_HH[MMSS]]
-obs_init_min  YYYYMMDD[_HH[MMSS]]
-obs_init_max  YYYYMMDD[_HH[MMSS]]
```

These two options indicate minimum/maximum values for forecast and observation initialization times.

21.3.1.10 Floating-Point Max/Min Options

Setting limits on various floating-point attributes. One may specify these as integers (i.e., without a decimal point), if desired. The following pairs of options indicate minimum and maximum values for each MODE attribute that can be described as a floating-point number. Please refer to Section 20.3.3 for a description of these attributes as needed.

```
-centroid_x_min value
-centroid_x_max value
```

```
-centroid_y_min value
-centroid_y_max value
```

```
-centroid_lat_min value
-centroid_lat_max value
```

```
-centroid_lon_min value
-centroid_lon_max value
```

```
-axis_ang_min value
-axis_ang_max value
```

```
-length_min value
-length_max value
```

```
-width_min value
-width_max value
```

```
-curvature_min value
-curvature_max value
```

```
-curvature_x_min value
-curvature_x_max value
```

```
-curvature_y_min value
-curvature_y_max value
```

```
-complexity_min value
-complexity_max value
```

```
-intensity_10_min value
-intensity_10_max value
```

```
-intensity_25_min value
-intensity_25_max value
```

```
-intensity_50_min value
-intensity_50_max value
```

```
-intensity_75_min value
-intensity_75_max value
```

```
-intensity_90_min value
-intensity_90_max value
```

```
-intensity_user_min value
-intensity_user_max value
```

```
-intensity_sum_min value
-intensity_sum_max value
```

```
-centroid_dist_min value
-centroid_dist_max value
```

```
-boundary_dist_min value
-boundary_dist_max value
```

```
-convex_hull_dist_min value
-convex_hull_dist_max value
```

```
-angle_diff_min value
-angle_diff_max value
```

```
-aspect_diff_min value
-aspect_diff_max value
```

```
-area_ratio_min value
-area_ratio_max value
```

```
-intersection_over_area_min value
-intersection_over_area_max value
```

```
-curvature_ratio_min value
-curvature_ratio_max value
```

```
-complexity_ratio_min value
-complexity_ratio_max value
```

```
-percentile_intensity_ratio_min value
-percentile_intensity_ratio_max value
```

```
-interest_min value
-interest_max value
```

21.3.1.11 Miscellaneous Options

These options are used to indicate parameters that did not fall into any of the previous categories.

```
-mask_poly filename
```

This option indicates the name of a polygon mask file to be used for filtering. The format for these files is the same as that of the polyline files for the other MET tools.

```
-help
```

This option prints the usage message.

21.3.2 mode_analysis Configuration File

To use the MODE-Analysis tool, the user must un-comment the options in the configuration file to apply them and comment out unwanted options. The options in the configuration file for the MODE-Analysis tools are the same as the MODE command line options described in Section 21.3.1.

The parameters that are set in the configuration file either add to or override parameters that are set on the command line. For the "set string" and "set integer type" options enclosed in brackets, the values specified in the configuration file are added to any values set on the command line. For the "toggle" and "min/max type" options, the values specified in the configuration file override those set on the command line.

21.3.3 mode_analysis Output

The output of the MODE-Analysis tool is a self-describing tabular format written to standard output. The length and contents of the table vary depending on whether **-summary** or **-bycase** is selected. The contents also change for **-summary** depending on the number of columns specified by the user.

Chapter 22

MODE Time Domain Tool

22.1 Introduction

22.1.1 Motivation

MODE Time Domain (MTD) is an extension of the MODE object-based approach to verification. In addition to incorporating spatial information, MTD utilizes the time dimension to get at temporal aspects of forecast verification. Since the two spatial dimensions of traditional meteorological forecasts are retained in addition to the time dimension, the method is inherently three dimensional. Given that, however, the overall methodology has deliberately been kept as similar as possible to that of traditional MODE.

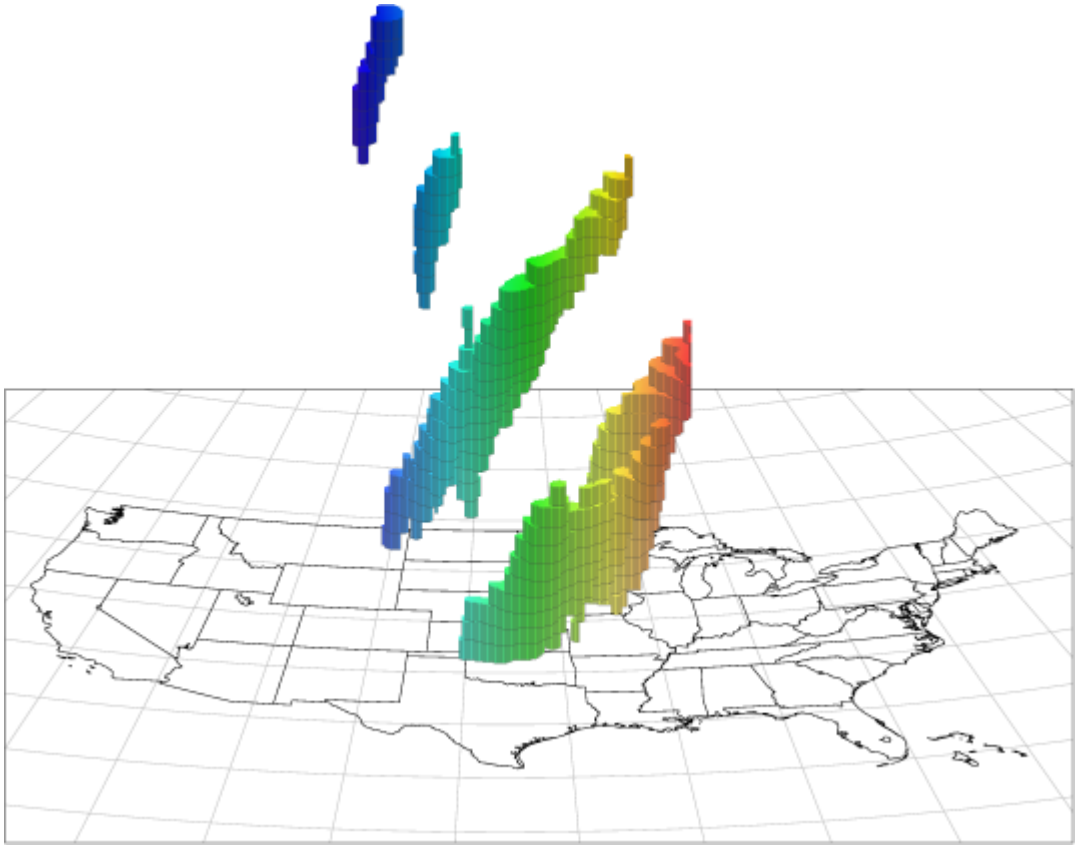

Figure 22.1: MTD Spacetime Objects

A plot of some MTD precipitation objects is shown over the United States in Figure 22.1. The colors indicate longitude, with red in the east moving through the spectrum to blue in the west. Time increases vertically in this plot (and in most of the spacetime diagrams in this users' guide). A few things are worthy of note in this figure. First, the tendency of storm systems to move from west to east over time shows up clearly. Second, tracking of storm objects over time is easily done: if we want to know if a storm at one time is a later version of a storm at an earlier time, we need only see if they are part of the same 3D spacetime object. Lastly, storms splitting up or merging over time are handled easily by this method.

The 2D (or traditional) MODE approach to object-base verification enabled users to analyze forecasts in terms of location errors, intensity errors and shape, size and orientation errors. MTD retains all of that capability, and adds new classes of forecast errors involving time information: speed and direction errors, buildup and decay errors, and timing and duration errors. This opens up new ways of analyzing forecast quality.

In the past, many MET users have performed separate MODE runs at a series of forecast valid times and analyzed the resulting object attributes, matches and merges as functions of time in an effort to incorporate temporal information in assessments of forecast quality. MTD was developed as a way to address this need in a more systematic way. Most of the information obtained from such multiple coordinated MODE runs can be obtained more simply from MTD.

At first glance, the addition of a third dimension would seem to entail no difficulties other than increased memory and processing requirements to handle the three-dimensional datasets and objects, and that would

indeed be largely true of an extension of MODE that used three spatial dimensions. In fact, the implementation of MTD entailed both conceptual difficulties (mostly due to the fact that there is no distance function in spacetime, so some MODE attributes, such as centroid distance, no longer even made sense), and engineering difficulties brought on by the need to redesign several core MODE algorithms for speed. It is planned that in the future some of these improved algorithms will be incorporated into MODE.

In this section, we will assume that the reader has a basic familiarity with traditional MODE, its internal operation, (convolution thresholding, fuzzy logic matching and merging) and its output. We will not review these things here. Instead, we will point out differences in MTD from the way traditional MODE does things when they come up. This release is a beta version of MTD, intended mostly to encourage users to experiment with it and give us feedback and suggestions to be used in a more robust MTD release in the future.

22.2 Scientific and Statistical Aspects

22.2.1 Attributes

Object attributes are, for the most part, calculated in much the same way in MTD as they are in MODE, although the fact that one of the dimensions is non-spatial introduces a few quirks. Several of the object attributes that traditional MODE calculates assume that distances, angles and areas can be calculated in grid coordinates via the usual Euclidean/Cartesian methods. That is no longer the case in spacetime, since there is no distance function (more precisely, no *metric*) there. Given two points in this spacetime, say (x_1, y_1, t_1) and (x_2, y_2, t_2), there is no way to measure their separation with a single nonnegative number in a physically meaningful way. If all three of our dimensions were spatial, there would be no difficulties.

This means that some care must be taken both in determining how to generalize the calculation of a geometric attribute to three-dimensional spacetime, and also in interpreting the attributes even in the case where the generalization is straightforward.

22.2.2 Convolution

As in MODE, MTD applies a convolution filter to the raw data as a preliminary step in resolving the field into objects. The convolution step in MTD differs in several respects from that performed in MODE, however.

First, MTD typically reads in several planes of data for each data field-one plane for each time step, and there is really no limit to the number of time steps. So MTD is convolving much more data than it would be if it were simply analyzing a 2D data field. Secondly, MTD convolves in time as well as space, which again increases the amount of data needing to be processed. The net effect of all this is to greatly increase the time needed to perform the convolution step.

Because of this, the developers decided to make several changes in the way convolution was performed in MTD. Most of the differences come from the need to make the convolution step as fast as possible.

The most basic change is to use a square convolution filter rather than the circular one that MODE uses. The overall "size" of the filter is still determined by one parameter (denoted R, as in MODE), but this should not be thought of as a radius. Instead, the size of the square is $(2R + 1) \times (2R + 1)$, as shown in Figure 22.2.

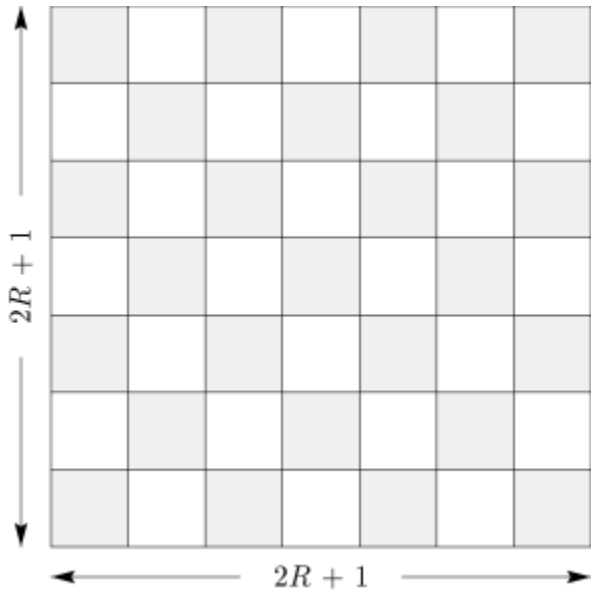

Figure 22.2: Convolution Region

Another change is that we do not allow any bad data in the convolution square. In MODE, the user may specify what percentage of bad data in the convolution region is permissible, and it will rescale the value of the filter accordingly for each data point. For the sake of speed, MTD requires that there be no bad data in the convolution region. If any bad data exists in the region, the convolved value there is set to a bad data flag.

22.2.3 3D Single Attributes

MTD calculates several 3D attributes for single objects. The object could come from either the forecast field or the observed field.

A 3D spacetime **centroid** $(\bar{x}, \bar{y}, \bar{t})$ is calculated. There are no statistical overtones here. The number $\bar{x}$, for example, is just the average value of the x coordinate over the object.

The vector **velocity** (v_x, v_y) is obtained by fitting a line to a 3D object. The requirement for fitting the line is to minimize the sum of the squares of the spatial distances from each point of the object to the line to be minimized. (We can't measure distances in spacetime but at each fixed time t we can measure purely spatial distances.) See Figure 22.3 for an illustration, where the solid line is the fitted axis, and the inclination of the axis from the vertical is a measure of object speed. Thus, from this velocity we get the **speed** and **direction** of movement of the object. As in MODE, where spatial separation is in units of the grid resolution, so here in MTD the unit of length is the grid resolution, and the unit of time is whatever the time separation between the input files is. Speed and velocity are thus in grid units per time unit.

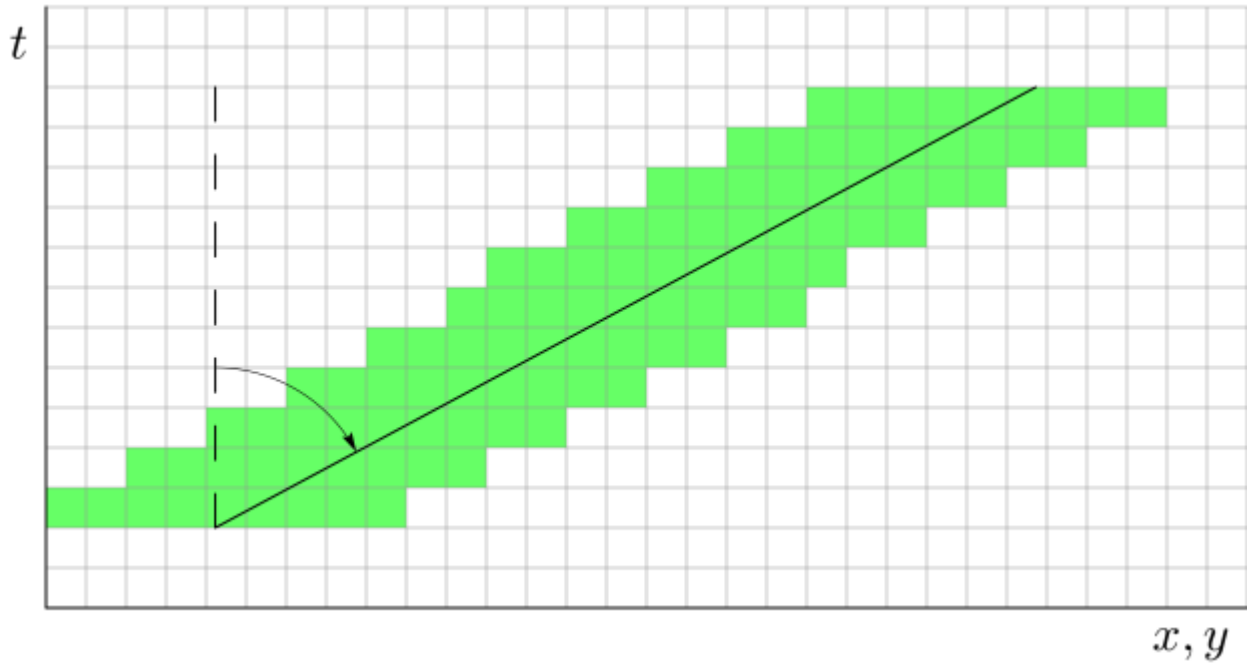

Figure 22.3: Velocity

The spatial orientation of an object (what traditional MODE calls the **axis angle** of an object) is gotten by fitting a plane to an object. As with the case of velocity, our optimization criterion is that the sum of the squares of the spatial distances from each point of the object to the plane be minimized.

Figure 22.4 gives some idea of the reason for fitting a plane, rather than a line, as MODE does. On the left in the figure, we see an object (in blue shaped like an "A") at several time steps moving through the grid. For simplicity, the object is not rotating as it moves (though of course real objects can certainly do this). At each time step, the 2D MODE spatial axis of the object is indicated by the red line. In the center of the figure, we see the same thing, just with more time steps. And on the right, even more time steps. We see that the axis lines at each time step sweep out a plane in three dimensions, shown in red on the right. This plane is the same one that MTD would calculate for this 3D object to determine its spatial orientation, *i.e.,* axis angle. Indeed, for the special case of an object that is not moving at all, the MTD calculation of axis angle reduces to the same one that traditional MODE uses, as it should.

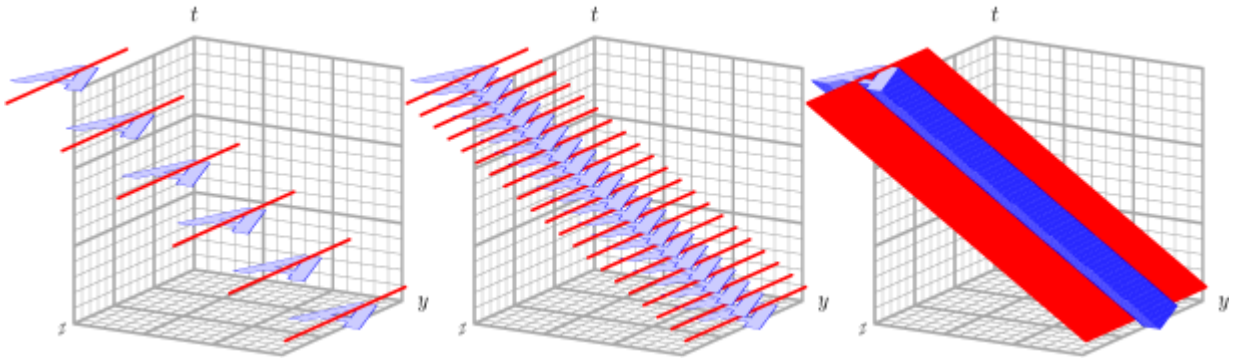

Figure 22.4: 3D axis

A simple integer count of the number of grid squares in an object for all of it's lifetime gives the **volume** of the object. Remember that while we're working in three dimensions, one of the dimensions is non-spatial, so one should not attempt to convert this to a volume in, e.g., km^3.

The **start time** and **end time** of an object are attributes as well. These are integers reflecting at which time step an object starts and ends. These values are zero-based, so for example, if an object comes into existence at the 3^{rd} time step and lasts until the 9^{th} time step, then the start time and end time will be listed as 2 and 8, respectively. Note that this object has a lifetime of 7 time steps, not 6.

Centroid distance traveled is the total great circle distance, in kilometers, traveled by the 2D spatial centroid over the lifetime of the object. In other words, at each time t for which the 3D object exists, the set of points in the object also have that value of t will together form a 2D spatial object. That 2D object will have a spatial centroid, which will move around as t varies. This attribute represents this total 2D centroid movement over time.

Finally, MTD calculates several **intensity percentiles** of the raw data values inside each object. Not all of the attributes are purely geometrical.

22.2.4 3D Pair Attributes

The next category of spatial attributes is for pairs of objects - one of the pair coming from the collection of forecast objects, the other coming from the observation objects.

Note: whenever a pair attribute is described below as a *delta*, that means it's a simple difference of two single-object attributes. The difference is always taken as "forecast minus observed".

The **spatial centroid distance** is the purely spatial part of the centroid separation of two objects. If one centroid is at $(\bar{x}_1, \bar{y}_1, \bar{t}_1)$ and the other is at $(\bar{x}_2, \bar{y}_2, \bar{t}_2)$ then the distance is calculated as

$$\sqrt{(\bar{x_1} - \bar{x_2})^2 + (\bar{y_1} - \bar{y_2})^2}$$

The **time centroid delta** is the difference between the time coordinates of the centroid. Since this is a simple difference, it can be either positive or negative.

The **axis difference** is smaller of the two angles that the two spatial axis planes make with each other. Figure 22.5 shows the idea. In the figure, the axis angle would be reported as angle α, not angle β.

Speed delta and **direction difference** are obtained from the velocity vectors of the two objects. Speed delta is the difference in the lengths of the vectors, and direction difference is the angle that the two vectors make with each other.

Volume ratio is the volume of the forecast object divided by the volume of the observed object. Note that any 3D object must necessarily have a nonzero volume, so there's no chance of zeros in the denominator.

Start time delta and **end time delta** are the differences in the corresponding time steps associated with the two objects and are computed as "forecast minus obs".

Intersection volume measures the overlap of two objects. If the two objects do not overlap, then this will be zero.

Duration difference is the difference in the lifetimes of the two objects constituting the pair, in the sense of "forecast minus obs". For example, if the forecast object of the pair has a lifetime of 5 time steps, and the

observed object has a lifetime of 3 time steps, then this attribute has the value 2. Note that we do not take absolute values of the difference, so this attribute can be positive, negative, or zero.

Finally, the **total interest** gives the result of the fuzzy-logic matching and merging calculation for this pair of objects. Note that this is provided only for simple objects, not for clusters.

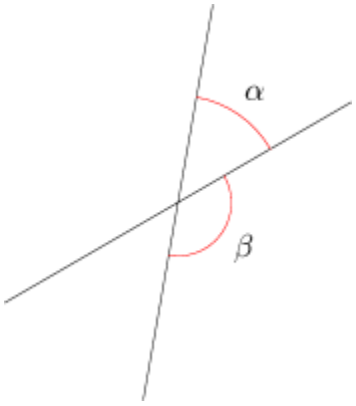

Figure 22.5: Axis Angle Difference

22.2.5 2D Constant-Time Attributes

The final category of object attributes calculated by MTD are two-dimensional spatial attributes for horizontal (*i.e.*, constant-time) slices of a spacetime object. This is so that the behavior of these attributes over time can be examined. These 2D constant-time attributes are written out for both simple and cluster objects.

For example, in our earlier discussion relating to Figure 22.4, we mentioned that for simplicity, the object in the figure was not allowed to rotate as it moved. But what if the object (a hurricane, for example) is rotating over time? In that case, it's probably not meaningful to assign a single spatial orientation to the object over its entire lifetime. If we had a spatial axis angle at each time, however, then we could fit a model such as $\theta = \theta_0 + \omega t$ to the angles and test the goodness of fit.

For such reasons, having 2D spatial attributes (as in MODE) for each object at each time step can be useful. The list of the 2D attributes calculated is:

◦ Centroid (x, y)

◦ Centroid latitude and longitude

◦ Area

◦ Axis Angle

22.2.6 Matching and Merging

Matching and merging operations in MTD are done in a simpler fashion than in MODE. In order to understand this operation, it is necessary to discuss some very basic notions of graph theory.

A **graph** is a finite set of **vertices** (also called **nodes**) and **edges**, with each edge connecting two vertices. Conceptually, it is enough for our purposes to think of vertices as points and edges as lines connecting them. See Figure 22.6 for an illustration. In the figure we see a collection of 11 nodes, indicated by the small circles, together with some edges indicated by straight line segments. A **path** is a sequence of vertices $(v_1, v_2, \ldots, v_n)$ such that for each $1 \leq i < n$ there is an edge connecting v_i to v_{i+1}. For example, in Figure 22.6, there is no edge connecting vertices #6 and #7, but there is a path connecting them. In illustrations, graph vertices are often labelled with identifying information, such as the numbers in Figure 22.6.

If we consider two distinct nodes in a graph to be related if there is a path connecting them, then it's easy to see that this defines an equivalence relation on the set of nodes, partitioning the graph into equivalence classes. Any node, such as #10 in Figure 22.6, that has no edges emanating from it is in a class by itself.

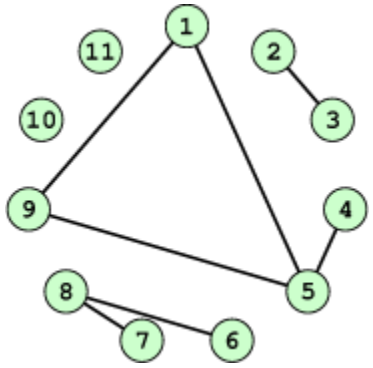

Figure 22.6: Basic Graph Example

We have barely scratched the surface of the enormous subject of graph theory, but this will suffice for our purposes. How does MTD use graphs? Essentially the simple forecast and observed objects become nodes in a graph. Each pair of objects that have sufficiently high total interest (as determined by the fuzzy logic engine) generates an edge connecting the two corresponding nodes in the graph. The graph is then partitioned into equivalence classes using path connectivity (as explained above), and the resulting equivalence classes determine the matches and merges.

An example will hopefully make this clear. In parts (a) and (b) of Figure 22.7 we indicate the objects in the forecast and observed field for this simple example. We have used 2D rather than 3D objects in this example for simplicity. Also, to help distinguish the objects in each field, the forecast objects are labelled by numbers and the observed object by letters. Each forecast and each observed object become nodes in a graph as indicated in part (c) of the figure.

For the purposes of this example, suppose that the MTD fuzzy engine reports that observed simple object B and forecast simple object 4 together have a total interest higher than the total interest threshold specified in the config file. Also, observed simple object C and forecast simple object 4 have high enough interest to pass the threshold. Furthermore, forecast simple objects 2 and 3 both have sufficiently high interest when paired with observed simple object A.

These four pairings result in the 4 edges in the graph shown by the solid lines in part (c) of the figure.

Partitioning this graph into equivalence classes results in the three sets indicated in part (d) of the figure. These three sets are the cluster objects determined by MTD. In this example, forecast objects 2 and 3 are merged into forecast cluster object #1 which is matched to observed cluster object #1, consisting of observed object A. (As in MODE, a cluster object may contain multiple simple objects, but may also consist of a single simple object.) Essentially, forecast simple objects 2 and 3 are merged because there is a path connecting them in the graph. This is indicated by the dashed line in the graph.

Continuing this example, forecast cluster object #2 (consisting only of forecast simple object 4) is matched to observed cluster object #2 (consisting of observed simple objects B and C). Again, the merging of observed simple objects is indicated by the dashed line in the graph.

Forecast cluster object #3 consists solely of forecast simple object 1. It is not matched to any observed cluster object. Alternatively, one may take the viewpoint that forecast simple object 1 ended up not participating in the matching and merging process; it is not merged with anything, it is not matched with anything. Essentially it represents a false alarm.

To summarize: Any forecast simple objects that find themselves in the same equivalence class are merged. Similarly, any observed objects in the same class are merged. Any forecast and observed objects in the same class are matched.

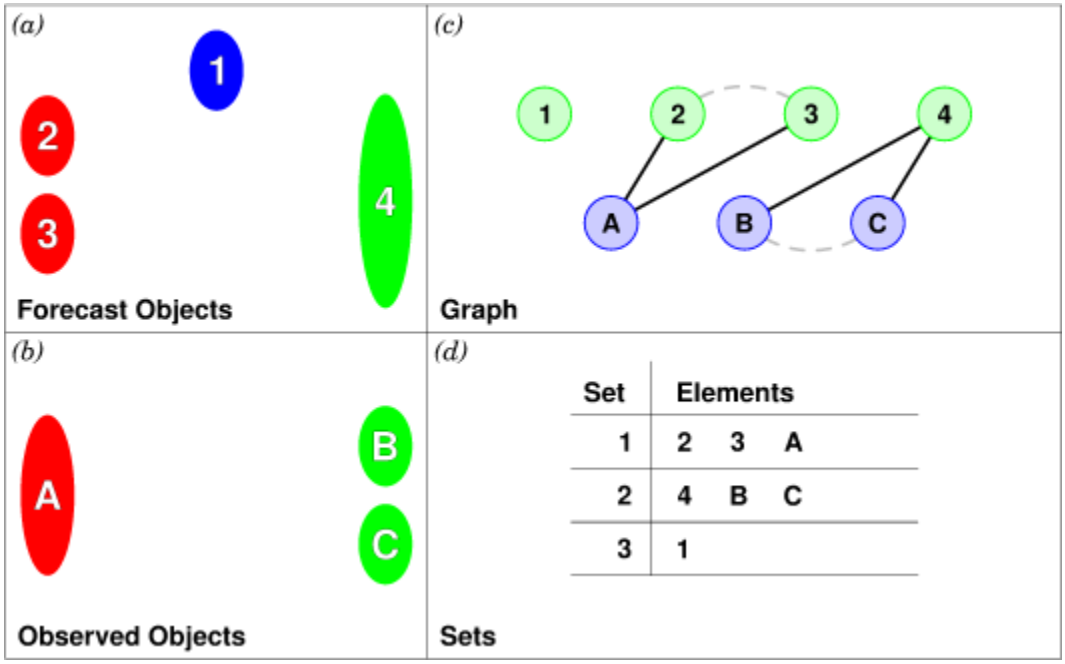

Figure 22.7: Match & Merge Example

22.3 Practical Information

22.3.1 MTD Input

The formats for two-dimensional data files used as input to MTD are the same ones supported by most of the MET tools. Generally speaking, if MODE can use a forecast or observation data file as input, then that file can also be used by MTD. The only difference is that while MODE takes only one forecast and one observed data file as input, MTD takes a series of files.

As shown in the next section, filenames for each time used must be given. Thus, for example, if MTD is being used for verification over a period of 24 hours, and the data file valid times are separated by one hour, then a total of 48 filenames must be specified on the MTD command line - 24 filenames for the forecast files, and 24 for the observation files. Further, the filenames must be given in order of increasing valid time. Many users will prefer to write scripts to automate this, rather than type in a lengthy command line by hand.

22.3.2 MTD Usage

The usage statement for the MODE-TD tool is listed below: The command line switches may be given in any order.

```
Usage: mtd
       -fcst    file_1 ... file_n | file_list
       -obs     file_1 ... file_n | file_list
       [-single  file_1 ... file_n | file_list]
       -config  config_file
       [-outdir path]
       [-log    file]
       [-v      level]
```

The MODE-TD tool has three required arguments and can accept several optional arguments.

22.3.2.1 Required Arguments for mtd

1. The **-fcst file_1 ... file_n | file_list** option specifies the gridded forecast files or ASCII file list of file names to be used, as described in Section 4.1.1. The files should have equally-spaced intervals of valid time.

2. The **-obs file_1 ... file_n | file_list** option specifies the gridded observation files or ASCII file list of file names to be used, as described in Section 4.1.1. The files should have equally-spaced intervals of valid time. This valid time spacing should be the same as for the forecast files.

3. **-config config_file** gives the path to a local configuration file that is specific to this particular run of MTD. The default MTD configuration file will be read first, followed by this one. Thus, only configuration options that are different from the default settings need be specified. Options set in this file will override any corresponding options set in the default configuration file.

22.3.2.2 Optional Arguments for mtd

4. The **-single file_1 ... file_n | file_list** option may be used instead of the **-fcst** and **-obs** command line options to define objects in a single field.

Note: When the **-single** command line option is used, data specified in the **fcst** configuration file entry is read from those input files.

---

5. **-log file** gives the name of a file where a log of this MTD run will be written. All output that appears on the screen during a MTD run will be duplicated in the log file.
6. **-v level** gives the verbosity level. As with the **-log** option described above, this option is present in most of the MET tools. Increasing this value causes more diagnostic output to be written to the screen (and also to the log file, if one has been specified).
7. **-outdir path** gives the name of the directory into which MTD will write its output files. If not specified, then MTD will write its output into the current directory.

An example of the mtd calling sequence is listed below:

```
mtd -fcst fcst_files/*.grb \
 -obs obs_files/*.grb \
 -config MTDConfig_default \
 -outdir out_dir/mtd \
 -v 1
```

In this example, the MODE-TD tool will read in a list of forecast GRIB files in the fcst_files directory and similarly spaced observation GRIB files in the obs_files directory. It uses a configuration file called **MTDConfig_default** and writes the output to the *out_dir/mtd* directory.

22.3.3 MTD Configuration File

The default configuration file for the MODE tool, **MODEConfig_default**, can be found in the installed *share/met/config* directory. Another version of the configuration file is provided in *scripts/config*. We encourage users to make a copy of the configuration files prior to modifying their contents.Most of the entries in the MTD configuration file should be familiar from the corresponding file for MODE. This initial beta release of MTD does not offer all the tunable options that MODE has accumulated over the years, however. In this section, we will not bother to repeat explanations of config file details that are exactly the same as those in MODE; we will only explain those elements that are different from MODE, and those that are unique to MTD.

```
model          = "FCST";
desc           = "NA";
obtype         = "ANALYS";
regrid         = { ... }
met_data_dir   = "MET_BASE";
```

(continues on next page)

(continued from previous page)

```
output_prefix  = "";
version        = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5.

```
grid_res = 4;
fcst = {
   field = {
      name  = "APCP";
      level = "A03";
   }
   conv_time_window  = { beg = -1; end = 1; }
   conv_radius       = 60.0/grid_res; // in grid squares
   conv_thresh       = >=5.0;
}
obs = fcst;
total_interest_thresh = 0.7;
```

The configuration options listed above are common to many MODE and are described in Section 20.3.2.

The **conv_time_window** entry is a dictionary defining how much smoothing in time should be done. The **beg** and **end** entries are integers defining how many time steps should be used before and after the current time. The default setting of **beg = -1; end = 1;** uses one time step before and after. Setting them both to 0 effectively disables smoothing in time.

```
inten_perc_value = 99;
```

The **inten_perc_value** entry is an integer between 0 and 100 which specifies a requested intensity percentile value. By default, MTD writes 5 output columns for the 10th, 25th, 50th, 75th, and 90th percentile of object intensities. The percentile value specified here indicates which percentile should be written to the 6th output column.

```
min_volume = 2000;
```

The **min_volume** entry tells MTD to throw away objects whose "volume" (as described elsewhere in this section) is smaller than the given value. Spacetime objects whose volume is less than this will not participate in the matching and merging process, and no attribute information will be written to the ASCII output files. The default value is 10,000. If this seems rather large, consider the following example: Suppose the user is running MTD on a 600×400 grid, using 24 time steps. Then the volume of the whole data field is $600 \times 400 \times 24 = 5{,}760{,}000$ cells. An object of volume 10,000 represents only $10{,}000/5{,}760{,}000 = 1/576$ of the total data field. Setting **min_volume** too small will typically produce a very large number of small objects, slowing down the MTD run and increasing the size of the output files.The configuration options listed above are common to many MODE and are described in Section 20.3.2.

```
weight = {
   space_centroid_dist  = 1.0;
   time_centroid_delta  = 1.0;
   speed_delta          = 1.0;
   direction_diff       = 1.0;
   volume_ratio         = 1.0;
   axis_angle_diff      = 1.0;
   start_time_delta     = 1.0;
   end_time_delta       = 1.0;
}
```

The **weight** entries listed above control how much weight is assigned to each pairwise attribute when computing a total interest value for object pairs. See Table 22.4 for a description of each weight option. When the total interest value is computed, the weighted sum is normalized by the sum of the weights listed above.

```
interest_function = {
   space_centroid_dist = ( ... );
   time_centroid_delta = ( ... );
   speed_delta         = ( ... );
   direction_diff      = ( ... );
   volume_ratio        = ( ... );
   axis_angle_diff     = ( ... );
   start_time_delta    = ( ... );
   end_time_delta      = ( ... );
};
```

The **interest_function** entries listed above control how much weight is assigned to each pairwise attribute when computing a total interest value for object pairs. See Table 22.4 for a description of each weight option. The interest functions may be defined as a piecewise linear function or as an algebraic expression. A piecewise linear function is defined by specifying the corner points of its graph. An algebraic function may be defined in terms of several built-in mathematical functions. See Section 20.2 for how interest values are used by the fuzzy logic engine. By default, many of these functions are defined in terms of the previously defined **grid_res** entry.

```
nc_output = {
   latlon     = true;
   raw        = true;
   object_id  = true;
   cluster_id = true;
};
```

The **nc_output** dictionary contains a collection of boolean flags controlling which fields are written to the NetCDF output file. **latlon** controls the output of a pair of 2D fields giving the latitude and longitude of each grid point. The **raw** entry controls the output of the raw input data for the MTD run. These will be 3D

fields, one for the forecast data and one for the observation data. Finally, the **object_id** and **cluster_id** flags control the output of the object numbers and cluster numbers for the objects. This is similar to MODE.

```
txt_output = {
   attributes_2d = true;
   attributes_3d = true;
};
```

The **txt_output** dictionary also contains a collection of boolean flags, in this case controlling the output of ASCII attribute files. The **attributes_2d** flag controls the output of the 2D object attributes for constant-time slices of 3D objects, while the **attributes_3d** flag controls the output of single and pair 3D spacetime object attributes.

22.3.4 mtd Output

MTD creates several output files after each run in ASCII and NetCDF formats. There are text files giving 2D and 3D attributes of spacetime objects and information on matches and merges, as well as a NetCDF file giving the objects themselves, in case any further or specialized analysis of the objects needs to be done.

MODE, along with several other of the MET tools (wavelet_stat for example, and a few others), provides PostScript-based graphics output to help visualize the output. Unfortunately, no similar graphics capabilities are provided with MTD, mainly because of the complexity of producing 3D plots. This should not discourage the user from making their own plots, however. There is enough information in the various output files created by MTD to make a wide variety of plots. Highly motivated users who write their own plotting scripts are encouraged to submit them to the user-contributed code area of the MET website. Due credit will be given, and others will benefit from their creations.

ASCII output

Five ASCII output files are created:

- Single attributes for 3D simple objects
- Single attributes for 3D cluster objects
- Pair attributes for 3D simple objects
- Pair attributes for 3D cluster objects
- 2D spatial attributes for single simple objects for each time index of their existence.

Each ASCII file is laid out in tabular format, with the first line consisting of text strings giving names for each column. The first 15 columns of each file are identical, and give information on timestamps, model names, and the convolution radius and threshold used for the forecast and observation input data.

These columns are explained in Table 22.1. Each file contains additional columns that come after these. Columns for 2D constant-time attributes are shown in Table 22.2. Columns for 3D single and pair attributes are shown in Table 22.3 and Table 22.4 respectively.

The contents of the OBJECT_ID and OBJECT_CAT columns identify the objects using the same logic as the MODE tool. In these columns, the F and O prefixes are used to indicate simple forecast and observation

objects, respectively. Similarly, the CF and CO prefixes indicate cluster forecast and observation objects, respectively. Each prefix is followed by a 3-digit number, using leading zeros, to indicate the object number (as in **F001, O001, CF001**, or **CO000**). Pairs of objects are indicated by listing the forecast object information followed by the observation object information, separated by an underscore (as in **F001_O001** or **CF001_CO001**). The OBJECT_ID column indicates the single object or pair of objects being described in that line. The OBJECT_CAT column indicates the cluster or pair of clusters to which these object(s) belong. A simple object that is not part of a cluster is assigned a cluster number of zero (as in **CF000** or **CO000**). When pairs of objects belong to the same matching cluster, the OBJECT_CAT column indicates the matching cluster number (as in **CF001_CO001**). When they do not, the OBJECT_CAT column is set to **CF000_CO000**.

Table 22.1: Text Header Columns

Column	Name	Description	Data Type
1	VERSION	Version number	String
2	MODEL	User-provided text string giving model name	String
3	DESC	User-provided text string describing the verification task	String
4	FCST_LEAD	Forecast lead time in HHMMSS format	Time String
5	FCST_VALID	Forecast valid time in YYYYMMDD_HHMMSS format	Datetime String
6	OBS_LEAD	Observation lead time in HHMMSS format	Time String
7	OBS_VALID	Observation valid time in YYYYMMDD_HHMMSS format	Datetime String
8	T_DELTA	Time separation between input data files in HHMMSS format	Time String
9	FCST_T_BEG	Forecast time convolution begin offset	Integer
10	FCST_T_END	Forecast time convolution end offset	Integer
11	FCST_RAD	Forecast convolution radius in grid units	Integer
12	FCST_THR	Forecast convolution threshold	Threshold String
13	OBS_T_BEG	Observation time convolution begin offset	Integer
14	OBS_T_END	Observation time convolution end offset	Integer
15	OBS_RAD	Observation convolution radius in grid units	Integer
16	OBS_THR	Observation convolution threshold	Threshold String
17	FCST_VAR	Forecast variable	String
18	FCST_UNITS	Units for forecast variable	String
19	FCST_LEV	Forecast vertical level	String
20	OBS_VAR	Observation variable	String
21	OBS_UNITS	Units for observation variable	String
22	OBS_LEV	Observation vertical level	String

Table 22.2: 2D Object Attributes

Column Number	2D Column Name	Description	Data Type
23	OBJECT_ID	Object number	String
24	OBJECT_CAT	Object category	String
25	TIME_INDEX	Time index of slice	Integer
26	AREA	2D cross-sectional area (in grid squares)	Integer
27-28	CENTROID_X,_Y	Location of centroid (in grid units)	Double
29-30	CENTROID_LAT,_LON	Location of centroid (in lat/lon degrees)	Double
31	AXIS_ANG	Angle that the axis makes with the grid x direction (in degrees)	Double
32-36	INTENSITY_10,_25,_50,_75,_90	10th, 25th, 50th, 75th, and 90th percentile intensity in time slice (various units)	Double
37	INTENSITY_NN	User-specified percentile intensity in time slice (various units)	Double

Table 22.3: 3D Single Object Attributes

Column Number	3D Single Column Name	Description	Data Type
23	OBJECT_ID	Object number	String
24	OBJECT_CAT	Object category	String
25-26	CENTROID_X,_Y	Location of the centroid (in grid units)	Double
27	CENTROID_T	Time coordinate of centroid (in time steps)	Double
28-29	CENTROID_LAT,_LON	Location of the centroid (in lat/lon degrees)	Double
30	X_DOT	X component of object velocity (in grid units per time step)	Double
31	Y_DOT	Y component of object velocity (in grid units per time step)	Double
32	AXIS_ANG	Angle that the axis plane of an object makes with the grid x direction (in degrees)	Double
33	VOLUME	3D object volume (integer count of 3D spacetime cells)	Integer
34	START_TIME	Object start time (in time steps)	Integer
35	END_TIME	Object end time (in time steps)	Integer
36	CDIST_TRAVELLED	Total great circle distance travelled by the 2D spatial centroid over the lifetime of the 3D object (in kilometers)	Double
37-41	INTENSITY_10,_25,_50,_75,_90	10th, 25th, 50th, 75th, and 90th percentile intensity in spacetime object (various units)	Double
42	INTENSITY_NN	User-specified percentile intensity in spacetime object (various units)	Double

Table 22.4: 3D Pair Object Attributes

Column Number	3D Pair Column Name	Description	Data Type
23	OBJECT_ID	Object number	String
24	OBJECT_CAT	Object category	String
25	SPACE_CENTROID_DIST	Cartesian distance between (x, y) coordinates of object spacetime centroids (in grid units)	Double
26	TIME_CENTROID_DELTA	Observation minus forecast difference in t temporal index of object spacetime centroid (in time steps)	Double
27	AXIS_DIFF	Angle between the spatial axis plane angles (in degrees, from 0 to 90)	Double
28	SPEED_DELTA	Forecast minus observation speed difference (in grid units per time step)	Double
29	DIRECTION_DIFF	Difference in object direction of movement (in degrees, from 0 to 180)	Double
30	VOLUME_RATIO	Forecast 3D object volume divided by observation 3D object volume (unitless)	Double
31	START_TIME_DELTA	Forecast minus observation starting time step (in time steps)	Integer
32	END_TIME_DELTA	Forecast minus observation ending time step (in time steps)	Integer
33	INTERSECTION_VOLUME	3D object intersection volume (integer count of 3D spacetime cells)	Integer
34	DURATION_DIFF	Forecast minus observation difference in object lifetimes (in time steps)	Integer
35	INTEREST	Total interest for this object pair (unitless)	Double

NetCDF File

MTD writes a NetCDF file containing various types of information as specified in the configuration file. The possible output data are:

- **Latitude** and **longitude** of all the points in the 2D grid. Useful for geolocating points or regions given by grid coordinates.
- **Raw data** from the input data files. This can be useful if the input data were grib format, since NetCDF is often easier to read.
- **Object ID** numbers, giving for each grid point the number of the simple object (if any) that covers that point. These numbers are one-based. A value of zero means that this point is not part of any object.
- **Cluster ID** numbers. As above, only for cluster objects rather than simple objects.

Chapter 23

MET-TC Overview

23.1 Introduction

The purpose of this User's Guide is to provide basic information to the users of the Model Evaluation Tools - Tropical Cyclone (MET-TC) to enable users to apply MET-TC to their tropical cyclone datasets and evaluation studies. MET-TC is intended for use with model forecasts run through a vortex tracking software or with operational model forecasts in Automated Tropical Cyclone Forecast (ATCF) file format.

The following sections provide an overview of MET-TC and its components, as well as basic information on the software build. The required input, including file format and the MET-TC are discussed followed by a description of the TC-DLand, TC-Diag, TC-Pairs, TC-Stat, TC-RMW, and RMW-Analysis tools. Each section covers the input, output and practical usage including a description of the configuration files. This is followed by a short overview of graphical utilities available within the MET-TC release.

23.2 MET-TC Components

The MET tools used in the verification of Tropical Cyclones are referred to as MET-TC. These tools are shown across the bottom of the flowchart in Figure 1.1. The MET-TC tools are described in more detail in later sections.

Tropical cyclone forecasts and observations are quite different than numerical model forecasts, and thus they have their own set of tools. These consist of TC-DLand, TC-Diag, TC-Pairs, TC-Stat, TC-Gen, TC-RMW, and RMW-Analysis. The TC-DLand module calculates the distance to land from all locations on a specified grid. This information can be used in later modules to eliminate tropical cyclones that are over land from being included in the statistics. TC-Diag converts gridded model output into cylindrical coordinates for each storm location, calls Python scripts to compute storm-relative diagnostics, and writes ASCII output to be read by TC-Pairs. TC-Pairs matches up tropical cyclone forecasts and observations and writes all output to a file. In TC-Stat, these forecast / observation pairs are analyzed according to user preference to produce statistics. TC-Gen evaluates the performance of Tropical Cyclone genesis forecast using contingency table counts and statistics. TC-RMW performs a coordinate transformation for gridded model or analysis fields centered on the current storm location. RMW-Analysis filters and aggregates the output of TC-RMW across multiple cases.

23.3 Input Data Format

This section discusses the input and output file formats expected and produced by MET-TC. When discussing the input data, it is expected that users have run model output through vortex tracking software in order to obtain position and intensity information in Automated Tropical Cyclone Forecasting System (ATCF) file format. Best track and aids files in Automated Tropical Cyclone Forecasting System (ATCF) format (hereafter referred to as ATCF format) are necessary for model data input into the TC-Pairs tool. The ATCF format was first developed at the Naval Oceanographic and Atmospheric Research Laboratory (NRL), and is currently used for the National Hurricane Center (NHC) operations. ATCF format must be adhered to in order for the MET-TC tools to properly parse the input data.

The ATCF file format includes a section with common fields:

BASIN, CY, YYYYMMDDHH, TECHNUM/MIN, TECH, TAU, LatN/S, LonE/W, VMAX, MSLP, TY, RAD, WINDCODE, RAD1, RAD2, RAD3, RAD4, POUTER, ROUTER, RMW, GUSTS, EYE, SUBREGION, MAXSEAS, INITIALS, DIR, SPEED, STORMNAME, DEPTH, SEAS, SEASCODE, SEAS1, SEAS2, SEAS3, SEAS4

BASIN: basin

CY: annual cyclone number: 1 - 99

YYYYMMDDHH: Warning Date-Time-Group.

TECHNUM/MIN: objective technique sorting number, minutes for best track: 00 - 99

TECH: acronym for each objective technique or CARQ or WRNG, BEST for best track

TAU: forecast period: -24 through 240 hours, 0 for best-track

LatN/S: Latitude for the date time group (DTG)

LonE/W: Longitude for the DTG

VMAX: Maximum sustained wind speed in knots

MSLP: Minimum sea level pressure, 850 - 1050 mb.

TY: Highest level of tropical cyclone development

RAD: Wind intensity for the radii defined in this record: 34, 50 or 64 kt.

WINDCODE: Radius code

RAD1: If full circle, radius of specified wind intensity, or radius of first quadrant wind intensity

RAD2: If full circle this field not used, or radius of 2nd quadrant wind intensity

RAD3: If full circle this field not used, or radius of 3rd quadrant wind intensity

RAD4: If full circle this field not used, or radius of 4th quadrant wind intensity

POUTER: pressure in millibars of the last closed isobar

ROUTER: radius of the last closed isobar

RMW: radius of max winds

GUSTS: gusts

EYE: eye diameter

SUBREGION: subregion

MAXSEAS: max seas

INITIALS: Forecaster's initials

DIR: storm direction

SPEED: storm speed

STORMNAME: literal storm name, number, NONAME or INVEST, or TCcyx

DEPTH: system depth

SEAS: Wave height for radii defined in SEAS1 - SEAS4

SEASCODE - Radius code

SEAS1: first quadrant seas radius as defined by SEASCODE

SEAS2: second quadrant seas radius as defined by SEASCODE

SEAS3: third quadrant seas radius as defined by SEASCODE

SEAS4: fourth quadrant seas radius as defined by SEASCODE

Of the above common fields in the ATCF file format, MET-TC requires the input file to have the first 8 comma-separated columns present. Although all 8 columns must exist, valid data in each field is not required. In order to ensure proper matching, unique data in the BASIN, CY, YYYYMMDDHH, and TAU fields should be present.

The TC-Pairs tool expects two input data sources in order to generate matched pairs and subsequent error statistics. The expected input for MET-TC is an ATCF format file from model output, or the operational aids files with the operational model output for the 'adeck' and the NHC best track analysis (BEST) for the 'bdeck'. The BEST is a subjectively smoothed representation of the storm's location and intensity over its lifetime. The track and intensity values are based on a retrospective assessment of all available observations of the storm.

The BEST is in ATCF file format and contains all the above listed common fields. Given the reference dataset is expected in ATCF file format, any second ATCF format file from model output or operational model output from the NHC aids files can be supplied as well. The expected use of the TC-Pairs tool is to generate matched pairs between model output and the BEST. Note that some of the columns in the TC-Pairs output are populated based on the BEST information (e.g. storm category), therefore use of a different baseline may reduce the available filtering options.

All operational model aids and the BEST can be obtained from the NHC ftp server.

Click here for detailed information on the ATCF format description and specifications.

If a user has gridded model output, the model data must be run through a vortex tracking algorithm in order to obtain the ATCF-formatted input that MET-TC requires. Many vortex tracking algorithms have been developed in order to obtain basic position, maximum wind, and minimum sea level pressure information from model forecasts. One vortex tracking algorithm that is supported and freely available is the GFDL vortex tracker package.

23.4 Output Data Format

The MET package produces output in four basic file formats: STAT files, ASCII files, NetCDF files, and PostScript plots. The MET-TC tool produces output in TCSTAT, which stands for Tropical Cyclone - STAT. This output format consists of tabular ASCII data that can be easily read by many analysis tools and software packages, making the output from MET-TC very versatile. Like STAT, TCSTAT is a specialized ASCII format containing one record on each line. Currently, the only line type available in MET-TC is TCMPR (Tropical Cyclone Matched Pairs). As more line types are included in future releases, all line types will be included in a single TCSTAT file. The TC-DLand, TC-Diag, TC-RMW, and RMW-Analysis tools also write NetCDF files containing a variety of output data types.

Chapter 24

TC-DLand Tool

24.1 Introduction

Many filtering criteria within the MET-TC tools depend on the distinction between when a storm is over land or water. The TC-dland tool was developed to aid in quickly parsing data for filter jobs that only verify over water, threshold verification based on distance to land, and exclusion of forecasts outside a specified time window of landfall. For each grid point in the user-specified grid, it computes the great circle arc distance to the nearest coast line. Compared to the simple Euclidean distances, great circle arc distances are more accurate but take considerably longer to compute. Grid points over water have distances greater than zero while points over land have distances less than zero.

While the TC-dland tool is available to be run, most users will find the precomputed distance to land files distributed with the release sufficient. Therefore, the typical user will not actually need to run this tool.

24.2 Input/Output Format

The input for the TC-dland tool is a file containing the longitude (degrees east) and latitude (degrees north) of all the coastlines and islands considered to be a significant landmass. The default input is to use all three land data files (**aland.dat, shland.dat, wland.dat**) found in the installed *share/met/tc_data/* directory. The use of all three files produces a global land data file. The **aland.dat** file contains the longitude and latitude distinctions used by NHC for the Atlantic and eastern North Pacific basins, the **shland.dat** contains longitude and latitude distinctions for the Southern Hemisphere (south Pacific and South Indian Ocean), and the **wland.dat** contains the remainder of the Northern Hemisphere (western North Pacific and North Indian Ocean). Users may supply their own input file in order to refine the definition of coastlines and a significant landmass.

The output file from TC-dland is a NetCDF format file containing a gridded field representing the distance to the nearest coastline or island, as specified in the input file. This file is used in the TC-Pairs tool to compute the distance from land for each track point in the adeck and bdeck. The precomputed distance to land (NetCDF output from TC-dland) files are available in the release. In the installed *share/met/tc_data* directory:

dland_nw_hem_tenth_degree.nc: TC-dland output from **aland.dat** using a 1/10th degree grid

dland_global_tenth_degree.nc: TC-dland output from all three land data files (global coverage) using a 1/10th degree grid.

24.3 Practical Information

This section briefly describes how to run tc_dland. The default grid is set to 1/10th degree Northwest (NW) hemispheric quadrant (over North America) grid.

24.3.1 tc_dland Usage

```
Usage: tc_dland
       out_file
       [-grid spec]
       [-noll]
       [-land file]
       [-log file]
       [-v level]
       [-compress level]
```

tc_dland has one required argument and accepts several optional ones.

24.3.1.1 Required Arguments for tc_dland

1. The **out_file** argument indicates the NetCDF output file containing the computed distances to land.

24.3.1.2 Optional Arguments for tc_dland

2. The **-grid spec** argument overrides the default grid (1/10th NH grid). Spec = **lat_ll lon_ll delta_lat delta_lon n_lat n_lon**
3. The **-noll** argument skips writing the lon/lat variables in the output NetCDF file to reduce the file size.
4. The **-land file** argument overwrites the default land data files (**aland.dat, shland.dat**, and **wland.dat**).
5. The **-log file** argument outputs log messages to the specified file.
6. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.
7. The **-compress level** option specifies the desired level of compression (deflate level) for NetCDF variables. The valid level is between 0 and 9. Setting the compression level to 0 will make no compression for the NetCDF output. Lower numbers result in minimal compression and faster I/O processing speed; higher numbers result in better compression, but at the expense of I/O processing speed.

Chapter 25

TC-Pairs Tool

25.1 Introduction

The TC-Pairs tool provides verification for tropical cyclone forecasts in ATCF file format. It matches an ATCF format tropical cyclone (TC) forecast with a second ATCF format reference TC dataset (most commonly the Best Track analysis). The TC-Pairs tool processes both track and intensity adeck data and probabilistic edeck data. The adeck matched pairs contain position errors, as well as values for each TC dataset, including wind speed, wind radii, minimum sea level pressure, and distance to land. The edeck matched pairs contain probabilistic forecast values and the verifying observation values. The pair generation can be subset based on user-defined filtering criteria. Practical aspects of the TC-Pairs tool are described in Section 25.3.

25.2 Scientific and Statistical Aspects

25.2.1 TC Diagnostics

TC diagnostics provide information about a TC's structure or its environment. Each TC diagnostic is a single-valued measure that corresponds to some aspect of the storm itself or the surrounding large-scale environment. TC diagnostics can be derived from observational analyses, model fields, or even satellite observations. Examples include:

- Inner core diagnostics provide information about the structure of the storm near the storm center. Examples include the intensity of the storm and the radius of maximum winds.
- Large scale diagnostics provide information about quantities that characterize its environment. Examples include environmental vertical wind shear, total precipitable water, relative humidity, convective instability, and the upper bound of intensity that a storm may be expected to achieve in its current environment. These diagnostics are typically derived from model fields as an average of the quantity of interest over either a circular area or an annulus centered on the storm center. Often, the storm center is taken to be the underlying model's storm center. In other cases, the diagnostics may be computed along some other specified track.
- Ocean-based diagnostics provide information about the sea's thermal characteristics in the vicinity of the storm center. Examples include the sea surface temperature, ocean heat content, and the depth of

warm water of a given temperature.

- Satellite-based diagnostics provide information about the storm structure as observed by geostationary satellite infrared imagery. Examples include information about the shape and extent of the cold-cirrus canopy of the TC and whether patterns are present that may portend intensification.

Diagnostics are critically important for training and running statistical-dynamical models that predict a TC's intensity or size. One of the most well-known diagnostics sets is that of the Statistical Hurricane Intensity Prediction Scheme (SHIPS), which supports predictions of TC intensity. A large 30-year development dataset of TC diagnostics has been retrospectively derived to support the training of the SHIPS intensity model as well as other related models such as the Logistic Growth Equation Model (LGEM), SHIPS Rapid Intensification Index (SHIPS-RII), and others. These diagnostics, called *lsdiag* for "large scale" environment, are computed using a *perfect prog* approach in which the diagnostics are computed on the reference model's verifying analyses to generate a set of time-dependent diagnostics from t=0 out to the desired maximum forecast lead time. This is repeated for each initialization, building up a full history of diagnostics for each storm. By using the subsequent verifying analysis for later lead times, the model is taken to be "perfect", removing the impact of model forecast errors. The resulting developmental dataset is ideal for training statistical-dynamical models such as SHIPS. To generate forecasts in real-time, the diagnostics are computed along a forecast track (often taken to be the National Hurricane Center's official forecast) using the fields of the underlying NWP model (e.g, the Global Forecast System, or GFS model). The resulting diagnostics are then used as *predictors* in models like SHIPS and LGEM to predict a TC's future intensity or probability of undergoing rapid intensification.

Beside their use in TC prediction, TC diagnostics can be very useful to forecasters to understand the forecast scenario. They are also useful to model developers for evaluation of model errors and understanding model performance under different environmental conditions. For instance, a modeler may wish to understand their model's track biases under conditions of high vertical wind shear. TC diagnostics can also be used to understand the sensitivity of the model's intensity predictions to oceanic conditions such as upwelling. The TC-Pairs tool allows filtering and subsetting based on the values of one or several TC diagnostics.

As of MET v11.0.0, two types of TC diagnostics are supported in TC-Pairs:

- SHIPS_DIAG_RT: Real-time SHIPS diagnostics computed from a NWP model such as the Global Forecast System (GFS) model along a GFS forecast track defined by a SHIPS-specific tracker. Note that these SHIPS-derived forecast tracks do not appear in the NHC adeck data.
- CIRA_DIAG_RT: Real-time model-based diagnostics computed along the model's predicted track.

Diagnostics from the SHIPS Development Datasets (SHIPS_DIAG_DEV) will be supported in a future release of MET.

A future version of MET will also allow the CIRA model diagnostics to be computed directly from model forecast fields. Until then, users may obtain the SHIPS diagnostics at the following locations:

- SHIPS_DIAG_DEV: https://rammb2.cira.colostate.edu/research/tropical-cyclones/ships/#DevelopmentalData
- SHIPS_DIAG_RT: https://ftp.nhc.noaa.gov/atcf/lsdiag/

25.3 Practical Information

This section describes how to configure and run the TC-Pairs tool. The TC-Pairs tool is used to match a tropical cyclone model forecast to a corresponding reference dataset. Both tropical cyclone forecast/reference data must be in ATCF format. Output from the TC-dland tool (NetCDF gridded distance file) is also a required input for the TC-Pairs tool. It is recommended to run tc_pairs on a storm-by-storm basis, rather than over multiple storms or seasons to avoid memory issues.

25.3.1 tc_pairs Usage

The usage statement for tc_pairs is shown below:

```
Usage: tc_pairs
       -adeck path and/or -edeck path
       -bdeck path
       -config file
       [-diag source path]
       [-out base]
       [-log file]
       [-v level]
```

tc_pairs has required arguments and can accept several optional arguments.

25.3.1.1 Required Arguments for tc_pairs

1. The **-adeck path** argument indicates the adeck TC-Pairs acceptable format data containing tropical cyclone model forecast (output from tracker) data to be verified. Acceptable data formats are limited to the standard ATCF format and the one column modified ATCF file, generated by running the tracker in genesis mode. It specifies the name of a TC-Pairs acceptable format file or top-level directory containing TC-Pairs acceptable format files ending in ".dat" to be processed. The **-adeck** or **-edeck** option must be used at least once.

2. The **-edeck path** argument indicates the edeck ATCF format data containing probabilistic track data to be verified. It specifies the name of an ATCF format file or top-level directory containing ATCF format files ending in ".dat" to be processed. The **-adeck** or **-edeck** option must be used at least once.

3. The **-bdeck path** argument indicates the TC-Pairs acceptable format data containing the tropical cyclone reference dataset to be used for verifying the adeck data. This data is typically the NHC Best Track Analysis, but could be any TC-Pairs acceptable formatted reference. The acceptable data formats for bdecks are the same as those for adecks. This argument specifies the name of a TC-Pairs acceptable format file or top-level directory containing TC-Pairs acceptable format files ending in ".dat" to be processed.

4. The **-config file** argument indicates the name of the configuration file to be used. The contents of the configuration file are discussed below.

25.3.1.2 Optional Arguments for tc_pairs

5. The **-diag source path** argument indicates the TC-Pairs acceptable format data containing the tropical cyclone diagnostics dataset corresponding to the adeck tracks. The **source** can be set to CIRA_DIAG_RT or SHIPS_DIAG_RT to indicate the input diagnostics data source. The **path** argument specifies the name of a TC-Pairs acceptable format file or top-level directory containing TC-Pairs acceptable format files ending in ".dat" to be processed. Support for additional diagnostic sources will be added in future releases.
6. The -**out base** argument indicates the path of the output file base. This argument overrides the default output file base (**./out_tcmpr**).
7. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.
8. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

This tool currently only supports the rapid intensification (**RI**) edeck probability type but support for additional edeck probability types will be added in future releases.

At least one **-adeck** or **-edeck** option must be specified. The **-adeck, -edeck**, and **-bdeck** options may optionally be followed with **suffix=string** to append that string to all model names found within that data source. This option may be useful when processing track data from two different sources which reuse the same model names.

An example of the tc_pairs calling sequence is shown below:

```
tc_pairs -adeck aal092010.dat -bdeck bal092010.dat -config TCPairsConfig
```

In this example, the TC-Pairs tool matches the model track (aal092010.dat) and the best track analysis (bal092010.dat) for the 9th Atlantic Basin storm in 2010. The track matching and subsequent error information is generated with configuration options specified in the **TCPairsConfig** file.

The TC-Pairs tool implements the following logic:

- Parse the adeck, edeck, and bdeck data files and store them as track objects.
- Parse diagnostics data files and add the requested diagnostics to the existing adeck track objects.
- Apply configuration file settings to filter the adeck, edeck, and bdeck track data down to a subset of interest.
- Apply configuration file settings to derive additional adeck track data, such as interpolated tracks, consensus tracks, time-lagged tracks, and statistical track and intensity models.
- For each adeck track that was parsed or derived, search for a matching bdeck track with the same basin and cyclone number and overlapping valid times. If not matching against the BEST track, also ensure that the model initialization times match.
- For each adeck/bdeck track pair, match up their track points in time, lookup distances to land, compute track location errors, and write an output TCMPR line for each track point.

- For each set of edeck probabilities that were parsed, search for a matching bdeck track.
- For each edeck/bdeck pair, write paired edeck probabilities and matching bdeck values to output PROBRIRW lines.

25.3.2 tc_pairs Configuration File

The default configuration file for the TC-Pairs tool named **TCPairsConfig_default** can be found in the installed *share/met/config/* directory. Users are encouraged to copy these default files before modifying their contents. The contents of the configuration file are described in the subsections below.

The contents of the tc_pairs configuration file are described below.

```
storm_id     = [];
basin        = [];
cyclone      = [];
storm_name   = [];
init_beg     = "";
init_end     = "";
init_inc     = [];
init_exc     = [];
valid_beg    = "";
valid_end    = "";
valid_inc    = [];
valid_exc    = [];
init_hour    = [];
init_mask    = "";
valid_mask   = "";
lead_req     = [];
match_points = TRUE;
version      = "VN.N";
```

The configuration options listed above are common to multiple MET tools and are described in Section 6.

```
model = [ "DSHP", "LGEM", "HWRF" ];
```

The **model** variable contains a list of comma-separated models to be used. Each model is identified with an ATCF TECH ID (normally four unique characters). This model identifier should match the model column in the ATCF format input file. An empty list indicates that all models in the input file(s) will be processed. Note that when reading ATCF track data, all instances of the string **AVN** are automatically replaced with **GFS**.

```
write_valid = [ "20101231_06" ];
```

The **write_valid** entry specifies a comma-separated list of valid time strings in YYYYMMDD[_HH[MMSS]] format for which output should be written. An empty list indicates that data for all valid times should be written. This option may be useful when verifying track forecasts in realtime. If evaluating performance for a single valid time, this option can limit the output to that time and skip output for earlier track points.

```
check_dup = FALSE;
```

The **check_dup** flag expects either TRUE and FALSE, indicating whether the code should check for duplicate ATCF lines when building tracks. Setting **check_dup** to TRUE will check for duplicated lines, and produce output information regarding the duplicate. Any duplicated ATCF line will not be processed in the tc_pairs output. Setting **check_dup** to FALSE, will still exclude tracks that decrease with time, and will overwrite repeated lines, but specific duplicate log information will not be output. Setting **check_dup** to FALSE will make parsing the track quicker.

```
interp12 = NONE;
```

The **interp12** flag expects the entry NONE, FILL, or REPLACE, indicating whether special processing should be performed for interpolated forecasts. The NONE option indicates no changes are made to the interpolated forecasts. The FILL and REPLACE (default) options determine when the 12-hour interpolated forecast (normally indicated with a "2" or "3" at the end of the ATCF ID) will be renamed with the 6-hour interpolated ATCF ID (normally indicated with the letter "I" at the end of the ATCF ID). The FILL option renames the 12-hour interpolated forecasts with the 6-hour interpolated forecast ATCF ID only when the 6-hour interpolated forecasts is missing (in the case of a 6-hour interpolated forecast which only occurs every 12-hours (e.g. EMXI, EGRI), the 6-hour interpolated forecasts will be "filled in" with the 12-hour interpolated forecasts in order to provide a record every 6-hours). The REPLACE option renames all 12-hour interpolated forecasts with the 6-hour interpolated forecasts ATCF ID regardless of whether the 6-hour interpolated forecast exists. The original 12-hour ATCF ID will also be retained in the output file (all modified ATCF entries will appear at the end of the TC-Pairs output file). This functionality expects both the 12-hour and 6-hour early (interpolated) ATCF IDs to be listed in the model field.

```
consensus = [
   {
      name          = "CON1";
      members       = [ "MOD1", "MOD2", "MOD3" ];
      required      = [   true,  false, false  ];
      min_req       = 2;
      diag_required = [   false, false, false  ];
      min_diag_req  = 0;
      write_members = TRUE;
   }
];
```

The **consensus** array allows users to derive consensus forecasts from any number of models. A consensus forecast is computed as the average intensity and location of the members which comprise it. TC-Pairs attempts to derive consensus forecasts for each unique storm ID and initialization time found in the input track data. Each array entry is a dictionary which defines the consensus name, membership, and requirements:

- The **name** field is a string defining the consensus model name to be written.
- The **members** field is a comma-separated array of model ID stings which define the members of the consensus.
- The **required** field is a comma-separated array of true/false values associated with each consensus member. If a member is designated as true, that member must be present in order for the consensus to be generated. If a member is false, the consensus will be generated regardless of whether or not the member is present. The required array can either be empty or have the same length as the members array. If empty, it defaults to all false.
- The **min_req** field is the number of members required in order for the consensus to be computed. The **required** and **min_req** field options are applied at each forecast lead time. If any member of the consensus has a non-valid position or intensity value, the consensus for that valid time will not be generated.
- Tropical cyclone diagnostics, if provided on the command line, are included in the computation of consensus tracks. The consensus diagnostics are computed as the mean of the diagnostics for the members. The **diag_required** and **min_diag_req** entries apply the same logic described above, but to the computation of each consensus diagnostic value rather than the consensus track location and intensity. If **diag_required** is missing or an empty list, it defaults to all false. If **min_diag_req** is missing, it defaults to 0.
- The **write_members** field is a boolean that indicates whether or not to write track output for the individual consensus members. If set to true, standard output will show up for all members. If set to false, output for the consensus members is excluded from the output, even if they are used to define other consensus tracks in the configuration file.

Users should take care to avoid filtering out track data for the consensus members with the **model** field, described above. Either set **model** to an empty list to process all input track data or include all of the consensus members in the **model** list. Use the **write_members** field, not the **model** field, to suppress track output for consensus members.

```
lag_time = [ "06", "12" ];
```

The **lag_time** field is a comma-separated list of forecast lag times to be used in HH[MMSS] format. For each adeck track identified, a lagged track will be derived for each entry. In the tc_pairs output, the original adeck record will be retained, with the lagged entry listed as the adeck name with "_LAG_HH" appended.

```
best_technique = [ "BEST" ];
best_baseline  = [ "BCLP", "BCD5", "BCLA" ];
```

The **best_technique** field specifies a comma-separated list of technique name(s) to be interpreted as BEST track data. The default value (BEST) should suffice for most users. The **best_baseline** field specifies a comma-separated list of CLIPER/SHIFOR baseline forecasts to be derived from the best tracks. Specifying multiple **best_technique** values and at least one **best_baseline** value results in a warning since the derived baseline forecast technique names may be used multiple times.

The following are valid baselines for the **best_baseline** field:

BTCLIP: Neumann original 3-day CLIPER in best track mode. Used for the Atlantic basin only. Specify model as BCLP.

BTCLIP5: 5-day CLIPER (*Aberson, 1998* (page 491))/SHIFOR (*DeMaria and Knaff, 2003* (page 496)) in best track mode for either Atlantic or eastern North Pacific basins. Specify model as BCS5.

BTCLIPA: Sim Aberson's recreation of Neumann original 3-day CLIPER in best-track mode. Used for Atlantic basin only. Specify model as BCLA.

```
oper_technique = [ "CARQ" ];
oper_baseline  = [ "OCLP", "OCS5", "OCD5" ];
```

The **oper_technique** field specifies a comma-separated list of technique name(s) to be interpreted as operational track data. The default value (CARQ) should suffice for most users. The **oper_baseline** field specifies a comma-separated list of CLIPER/SHIFOR baseline forecasts to be derived from the operational tracks. Specifying multiple **oper_technique** values and at least one **oper_baseline** value results in a warning since the derived baseline forecast technique names may be used multiple times.

The following are valid baselines for the **oper_baseline** field:

OCLIP: Merrill modified (operational) 3-day CLIPER run in operational mode. Used for Atlantic basin only. Specify model as OCLP.

OCLIP5: 5-day CLIPER (*Aberson, 1998* (page 491))/ SHIFOR (*DeMaria and Knaff, 2003* (page 496)) in operational mode, rerun using CARQ data. Specify model as OCS5.

OCLIPD5: 5-day CLIPER (*Aberson, 1998* (page 491))/ DECAY-SHIFOR (*DeMaria and Knaff, 2003* (page 496)). Specify model as OCD5.

```
anly_track = BDECK;
```

Analysis tracks consist of multiple track points with a lead time of zero for the same storm. An analysis track may be generated by running model analysis fields through a tracking algorithm. The **anly_track** field specifies which datasets should be searched for analysis track data and may be set to **NONE, ADECK, BDECK**, or **BOTH**. Use **BOTH** to create pairs using two different analysis tracks.

```
match_points = TRUE;
```

The **match_points** field specifies whether only those track points common to both the adeck and bdeck tracks should be written out. If **match_points** is selected as FALSE, the union of the adeck and bdeck tracks will be written out, with "NA" listed for unmatched data.

```
dland_file = "MET_BASE/tc_data/dland_global_tenth_degree.nc";
```

The **dland_file** string specifies the path of the NetCDF format file (default file: dland_global_tenth_degree.nc) to be used for the distance to land check in the tc_pairs code. This file is generated using tc_dland (default file provided in installed *share/met/tc_data* directory).

```
watch_warn = {
    file_name   = "MET_BASE/tc_data/wwpts_us.txt";
    time_offset = -14400;
 }
```

The **watch_warn** field specifies the file name and time applied offset to the **watch_warn** flag. The **file_name** string specifies the path of the watch/warning file to be used to determine when a watch or warning is in effect during the forecast initialization and verification times. The default file is named **wwpts_us.txt**, which is found in the installed *share/met/tc_data/* directory within the MET build. The **time_offset** string is the time window (in seconds) assigned to the watch/warning. Due to the non-uniform time watches and warnings are issued, a time window is assigned for which watch/warnings are included in the verification for each valid time. The default watch/warn file is static, and therefore may not include warned storms beyond the current MET code release date; therefore users may wish to create a post in the METplus GitHub Discussions Forum in order to obtain the most recent watch/warning file if the static file does not contain storms of interest.

```
diag_info_map = [
   {
      diag_source    = "CIRA_DIAG_RT";
      track_source   = "GFS";
      field_source   = "GFS_0p50";
      match_to_track = [];
      diag_name      = [];
   },
   {
      diag_source    = "SHIPS_DIAG_RT";
      track_source   = "SHIPS_TRK";
      field_source   = "GFS_0p50";
      match_to_track = [ "OFCL" ];
      diag_name      = [];
   }
];
```

A TCMPR line is written to the output for each track point. If diagnostics data is also defined for that track point, a TCDIAG line is written immediately after the corresponding TCMPR line. The contents of that TCDIAG line is determined by the **diag_info_map** entry.

The **diag_info_map** entries define how the diagnostics read with the **-diag** command line option should be used. Each array element is a dictionary consisting of entries for **diag_source**, **track_source**, **field_source**, **match_to_track**, and **diag_name**.

- The **diag_source** entry is one of the supported diagnostics data sources.

- The **track_source** entry is a string defining the ATCF ID of the track data used to define the locations at which diagnostics are computed. This string is written to the **TRACK_SOURCE** column of the TCDIAG output line.
- The **field_source** entry is a string describing the gridded model data from which the diagnostics are computed. This string is written to the **FIELD_SOURCE** column of the TCDIAG output line type.
- The **match_to_track** entry specifies a comma-separated list of strings defining the ATCF ID(s) of the tracks to which these diagnostic values should be matched. For the SHIPS_DIAG_RT source, this is required since it is the only way to associate diagnostics with track ATCF IDs. For the CIRA_DIAG_RT source, this is optional. If a non-zero list is provided, the diagnostics will be matched to tracks for the specified ATCF ID(s). If defined as an empty list (default), the ATCF ID will be extracted from each CIRA diagnostic file and used to match the diagnostics to track data.
- The **diag_name** entry specifies a comma-separated list of strings for the tropical cyclone diagnostics of interest. If a non-zero list of diagnostic names is specified, only those diagnostics appearing in the list are written to the TCDIAG output line type. If defined as an empty list (default), all diagnostics found in the input are written to the TCDIAG output lines.

```
diag_convert_map = [
   {
      diag_source = "CIRA_DIAG";
      key         = [ "(10C)", "(10KT)", "(10M/S)" ];
      convert(x)  = x / 10;
   },
   {
      diag_source = "SHIPS_DIAG";
      key         = [ "LAT",  "LON",  "CSST", "RSST", "DSST", "DSTA", "XDST", "XNST", "NSST",
↪ "NSTA",
                      "NTMX", "NTFR", "U200", "U20C", "V20C", "E000", "EPOS", "ENEG", "EPSS",
↪ "ENSS",
                      "T000", "TLAT", "TLON", "TWAC", "TWXC", "G150", "G200", "G250", "V000",
↪ "V850",
                      "V500", "V300", "SHDC", "SHGC", "T150", "T200", "T250", "SHRD", "SHRS",
↪ "SHRG",
                      "HE07", "HE05", "PW01", "PW02", "PW03", "PW04", "PW05", "PW06", "PW07",
↪ "PW08",
                      "PW09", "PW10", "PW11", "PW12", "PW13", "PW14", "PW15", "PW16", "PW17",
↪ "PW18",
                      "PW20", "PW21" ];
      convert(x)  = x / 10;
   },
   {
      diag_source = "SHIPS_DIAG";
      key         = [ "VVAV", "VMFX", "VVAC" ];
      convert(x)  = x / 100;
   },
```

(continues on next page)

(continued from previous page)

```
   {
       diag_source = "SHIPS_DIAG";
       key         = [ "TADV" ];
       convert(x)  = x / 1000000;
   },
   {
      diag_source = "SHIPS_DIAG";
      key         = [ "Z850", "D200", "TGRD", "DIVC" ];
      convert(x)  = x / 10000000;
   },
   {
      diag_source = "SHIPS_DIAG";
      key         = [ "PENC", "PENV" ];
      convert(x)  = x / 10 + 1000;
   }
];
```

The **diag_convert_map** entries define conversion functions to be applied to diagnostics data read with the **-diag** command line option. Each array element is a dictionary consisting of a **diag_source**, **key**, and **convert(x)** entry.

The **diag_source** entry is one of the supported diagnostics data sources. Partial string matching logic is applied, so **SHIPS_DIAG** entries are matched to both **SHIPS_DIAG_RT** and **SHIPS_DIAG_DEV** diagnostic sources. The **key** entry is an array of strings. The strings can specify diagnostic names or units, although units are only checked for **CIRA_DIAG** sources. If both the name and units are specified, the conversion function for the name takes precedence. The **convert(x)** entry is a function of one variable which defines how the diagnostic data should be converted. The defined function is applied to any diagnostic value whose name or units appears in the **key**.

```
basin_map = [
   { key = "SI"; val = "SH"; },
   { key = "SP"; val = "SH"; },
   { key = "AU"; val = "SH"; },
   { key = "AB"; val = "IO"; },
   { key = "BB"; val = "IO"; }
];
```

The **basin_map** entry defines a mapping of input names to output values. Whenever the basin string matches "key" in the input ATCF files, it is replaced with "val". This map can be used to modify basin names to make them consistent across the ATCF input files.

Many global modeling centers use ATCF basin identifiers based on region (e.g., 'SP' for South Pacific Ocean, etc.), however the best track data provided by the Joint Typhoon Warning Center (JTWC) use just one basin identifier 'SH' for all of the Southern Hemisphere basins. Additionally, some modeling centers may report basin identifiers separately for the Bay of Bengal (BB) and Arabian Sea (AB) whereas JTWC uses 'IO'.

The basin mapping allows MET to map the basin identifiers to the expected values without having to modify

your data. For example, the first entry in the list below indicates that any data entries for 'SI' will be matched as if they were 'SH'. In this manner, all verification results for the Southern Hemisphere basins will be reported together as one basin.

An empty list indicates that no basin mapping should be used. Use this if you are not using JTWC best tracks and you would like to match explicitly by basin or sub-basin. Note that if your model data and best track do not use the same basin identifier conventions, using an empty list for this parameter will result in missed matches.

25.3.3 tc_pairs Output

TC-Pairs produces output in TCST format. The default output file name can be overwritten using the -out file argument in the usage statement. The TCST file output from TC-Pairs may be used as input into the TC-Stat tool. The header column in the TC-Pairs output is described in Table 25.1.

Table 25.1: Header information for TC-Pairs TCST output

Column Number	Header Column Name	Description	Data Type
1	VERSION	Version number	String
2	AMODEL	User-provided text string designating the forecast ATCF ID	String
3	BMODEL	User-provided text string designating the reference ATCF ID	String
4	DESC	User-provided description text string	String
5	STORM_ID	BBCCYYYY designation of storm	String
6	BASIN	Basin (BB in STORM_ID)	String
7	CYCLONE	Cyclone number (CC in STORM_ID)	String
8	STORM_NAME	Name of Storm	String
9	INIT	Initialization time of forecast in YYYYMMDD_HHMMSS format	Datetime String
10	LEAD	Forecast lead time in HHMMSS format	Time String
11	VALID	Forecast valid time in YYYYMMDD_HHMMSS format	Datetime String
12	INIT_MASK	Initialization time masking grid applied	String
13	VALID_MASK	Valid time masking grid applied	String
14	LINE_TYPE	Output line types described below	String

Table 25.2: Format information for TCMPR (Tropical Cyc
Matched Pairs) output line type.

Column Number	TCMPR Column Name	Description
14	TCMPR	Tropical Cyclone Matched Pair line type
15	TOTAL	Total number of pairs in track
16	INDEX	Index of the current track pair
17	LEVEL	Level of storm classification

Table 25.2 – continued from previous page

Column Number	TCMPR Column Name	Description
18	WATCH_WARN	HU or TS watch or warning in effect
19	INITIALS	Forecaster initials
20	ALAT	Latitude position of adeck model
21	ALON	Longitude position of adeck model
22	BLAT	Latitude position of bdeck model
23	BLON	Longitude position of bdeck model
24	TK_ERR	Track error of adeck relative to bdeck (nm)
25	X_ERR	X component position error (nm)
26	Y_ERR	Y component position error (nm)
27	ALTK_ERR	Along track error (nm)
28	CRTK_ERR	Cross track error (nm)
29	ADLAND	adeck distance to land (nm)
30	BDLAND	bdeck distance to land (nm)
31	AMSLP	adeck mean sea level pressure
32	BMSLP	bdeck mean sea level pressure
33	AMAX_WIND	adeck maximum wind speed
34	BMAX_WIND	bdeck maximum wind speed
35, 36	A/BAL_WIND_34	a/bdeck 34-knot radius winds in full circle or the mean of the non-zero 3
37, 38	A/BNE_WIND_34	a/bdeck 34-knot radius winds in NE quadrant
39, 40	A/BSE_WIND_34	a/bdeck 34-knot radius winds in SE quadrant
41, 42	A/BSW_WIND_34	a/bdeck 34-knot radius winds in SW quadrant
43, 44	A/BNW_WIND_34	a/bdeck 34-knot radius winds in NW quadrant
45, 46	A/BAL_WIND_50	a/bdeck 50-knot radius winds in full circle or the mean of the non-zero 5
47, 48	A/BNE_WIND_50	a/bdeck 50-knot radius winds in NE quadrant
49, 50	A/BSE_WIND_50	a/bdeck 50-knot radius winds in SE quadrant
51, 52	A/BSW_WIND_50	a/bdeck 50-knot radius winds in SW quadrant
53, 54	A/BNW_WIND_50	a/bdeck 50-knot radius winds in NW quadrant
55, 56	A/BAL_WIND_64	a/bdeck 64-knot radius winds in full circle or the mean of the non-zero 6
57, 58	A/BNE_WIND_64	a/bdeck 64-knot radius winds in NE quadrant
59, 60	A/BSE_WIND_64	a/bdeck 64-knot radius winds in SE quadrant
61, 62	A/BSW_WIND_64	a/bdeck 64-knot radius winds in SW quadrant
63, 64	A/BNW_WIND_64	a/bdeck 64-knot radius winds in NW quadrant
65, 66	A/BRADP	pressure in millibars of the last closed isobar, 900 - 1050 mb
67, 68	A/BRRP	radius of the last closed isobar in nm, 0 - 9999 nm
69, 70	A/BMRD	radius of max winds, 0 - 999 nm
71, 72	A/BGUSTS	gusts, 0 through 995 kts
73, 74	A/BEYE	eye diameter, 0 through 999 nm
75, 76	A/BDIR	storm direction in compass coordinates, 0 - 359 degrees
77, 78	A/BSPEED	storm speed, 0 - 999 kts
79, 80	A/BDEPTH	system depth, D-deep, M-medium, S-shallow, X-unknown
81	NUM_MEMBERS	consensus variable: number of models (or ensemble members) that were
82	TRACK_SPREAD	consensus variable: the mean of the distances from the member location
83	TRACK_STDEV	consensus variable: the standard deviation of the distances from the mem

Table 25.2 – continued from previous page

Column Number	TCMPR Column Name	Description
84	MSLP_STDEV	consensus variable: the standard deviation of the member's mean sea lev
85	MAX_WIND_STDEV	consensus variable: the standard deviation of the member's maximum w

Table 25.3: Format information for TCDIAG (Tropical Cyclone Diagnostics) output line type.

Column Number	TCDIAG Column Name	Description	Data Type
14	TCDIAG	Tropical Cyclone Diagnostics line type	String
15	TOTAL	Total number of pairs in track	Integer
16	INDEX	Index of the current track pair	Integer
17	DIAG_SOURCE	Diagnostics data source indicated by the *-diag* command line option	String
18	TRACK_SOURCE	ATCF ID of the track data used to define the diagnostics	String
19	FIELD_SOURCE	Description of gridded field data source used to define the diagnostics	String
20	N_DIAG	Number of storm diagnostic name and value columns to follow	Integer
21	DIAG_i	Name of the of the ith storm diagnostic (repeated)	String
22	VALUE_i	Value of the ith storm diagnostic (repeated)	Double

Table 25.4: Format information for PROBRIRW (Probability of Rapid Intensification/Weakening) output line type.

Column Number	PROBRIRW Column Name	Description	Data Type
14	PROBRIRW	Probability of Rapid Intensification/Weakening line type	String
15	ALAT	Latitude position of edeck model	Double
16	ALON	Longitude position of edeck model	Double
17	BLAT	Latitude position of bdeck model	Double
18	BLON	Longitude position of bdeck model	Double
19	INITIALS	Forecaster initials	String
20	TK_ERR	Track error of adeck relative to bdeck (nm)	Double
21	X_ERR	X component position error (nm)	Double
22	Y_ERR	Y component position error (nm)	Double
23	ADLAND	adeck distance to land (nm)	Double
24	BDLAND	bdeck distance to land (nm)	Double
25	RI_BEG	Start of RI time window in HH format	Time String
26	RI_END	End of RI time window in HH format	Time String
27	RI_WINDOW	Width of RI time window in HH format	Time String
28	AWIND_END	Forecast maximum wind speed at RI end	Double
29	BWIND_BEG	Best track maximum wind speed at RI begin	Double
30	BWIND_END	Best track maximum wind speed at RI end	Double
31	BDELTA	Exact Best track wind speed change in RI window	Double
32	BDELTA_MAX	Maximum Best track wind speed change in RI window	Double
33	BLEVEL_BEG	Best track storm classification at RI begin	String
34	BLEVEL_END	Best track storm classification at RI end	String
35	N_THRESH	Number of probability thresholds	Integer
36	THRESH_i	The ith probability threshold value (repeated)	Double
37	PROB_i	The ith probability value (repeated)	Double

Chapter 26

TC-Diag Tool

26.1 Introduction

A diagnosis of the large-scale environment of tropical cyclones (TCs) is foundational for many prediction techniques, including statistical-dynamical forecast aids and techniques based on artificial intelligence. Such diagnostics can also be used by forecasters seeking to understand how a given model's forecast will pan out. Finally, TC diagnostics can be useful in verification to stratify the performance of models in different environmental regimes over a longer period of time, thereby providing useful insights on model biases or deficiencies for model developers and forecasters.

Originally developed for the Statistical Hurricane Intensity Prediction Scheme (SHIPS), and later as a stand-alone package called 'Model Diagnostics', by the Cooperative Institute for Research in the Atmosphere (CIRA), MET now integrates these capabilities into an extensible framework called the TC-Diag tool. This tool allows users to compute diagnostics for the large-scale environment of TCs using ATCF track and gridded model data inputs. The current version of the TC-Diag tool requires that the tracks and fields be self-consistent [i.e., the track should be the model's (or ensemble's) own predicted track(s)]. The reason is that the diagnostics are computed in a coordinate system centered on the model's moving model storm and the current version of the tool does not yet include vortex removal. If the track is not consistent with the underlying fields, the diagnostics output are unlikely to be useful because the model's simulated storm would contaminate the diagnostics calculations.

Note: A future version of the tool will include the capability to remove the model's own vortex, which will allow the user to specify any arbitrary track (such as the operational center's official forecast). Until then, users are advised that the track selected must be consistent with the model's predicted track.

TC-Diag is run once for each initialization time to produce diagnostics for each user-specified combination of TC tracks and model fields. The user provides track data (such as one or more ATCF a-deck track files), along with track filtering criteria as needed, to select one or more tracks to be processed. The user also provides gridded model data from which diagnostics should be computed. Gridded data can be provided for multiple concurrent storms, multiple models, and/or multiple domains (i.e. parent and nest) in a single run.

TC-Diag first determines the list of valid times that appear in any one of the tracks. For each valid time,

it processes all track points for that time. For each track point, it reads the gridded model fields requested in the configuration file and transforms the gridded data to a range-azimuth cylindrical coordinates grid, as described for the TC-RMW tool in Section 29. For each domain, it writes the range-azimuth data to a temporary NetCDF file, as described in `Contributor's Guide Section %s`.

Once the input data have been processed into the temporary NetCDF files, TC-Diag then calls one or more Python diagnostics scripts, as specified in the configuration file, to compute tropical cyclone diagnostic values. The computed diagnostics values are retrieved from the Python script and stored in memory.

After processing all valid times and all corresponding track points, the computed diagnostics are written to ASCII and/or NetCDF output files. If requested in the configuration file, the temporary range-azimuth cylindrical coordinates files are combined into a single NetCDF file and written to the output for each combination of model track and domain.

The default Python diagnostics scripts included with the MET release provide the standard set of CIRA diagnostics. However, users can copy/modify the logic in those scripts as they see fit to refine and/or add to the diagnostics computed.

Note: The Python diagnostics library in MET uses SciPy to compute certain diagnostics on a cylindrical grid, including wind shear, tangential winds, vorticity, and divergence. If SciPy is available in the version of Python that is run, those diagnostics will be computed. If SciPy is missing, TC-Diag will still run, but a fill value will be written for them.

26.2 Practical Information

26.2.1 tc_diag Usage

The following sections describe the usage statement, required arguments, and optional arguments for tc_diag.

```
Usage: tc_diag
       -data domain tech_id_list [ file_1 ... file_n | file_list ]
       -deck file
       -config file
       [-outdir path]
       [-log file]
       [-v level]
```

tc_diag has required arguments and can accept several optional arguments.

26.2.1.1 Required Arguments for tc_diag

1. The **-data domain tech_id_list [file_1 ... file_n | file_list]** option specifies a domain name, a comma-separated list of ATCF tech ID's, and a list of gridded data files or an ASCII file containing a list of files to be used, as described in Section 4.1.1. Specify **-data** once for each gridded data source.
2. The **-deck source** option is the ATCF format track data source.
3. The **-config file** option is the TCDiagConfig file to be used. The contents of the configuration file are discussed below.

26.2.1.2 Optional Arguments for tc_diag

4. The **-outdir path** option overrides the default output directory (current working directory) with the output directory path provided.
5. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no logfile.
6. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

26.2.2 tc_diag Configuration File

The default configuration file for the TC-Diag tool named **TCDiagConfig_default** can be found in the installed *share/met/config/* directory. Users are encouraged to copy these default files before modifying their contents. The contents of the configuration file are described in the subsections below.

26.2.2.1 Configuring Input Tracks and Time

```
model = [ "GFSO", "OFCL" ];
storm_id = "";
basin = "";
cyclone = "";
init_inc = "";
valid_beg = "";
valid_end = "";
valid_inc = [];
valid_exc = [];
valid_hour = [];
tmp_dir = "/tmp";
version = "VN.N";
```

The TC-Diag tool should be configured to filter the input track data (**-deck**) down to the subset of tracks that correspond to the gridded data files provided (**-data**). The filtered tracks should contain data for only *one initialization time* but may contain tracks for multiple models.

The configuration options listed above are used to filter the input track data down to those that should be processed in the current run. These options are common to multiple MET tools and are described in Section 6.

```
lead = [   "0",    "6",  "12",  "18",  "24",
          "30",   "36",  "42",  "48",  "54",
          "60",   "66",  "72",  "78",  "84",
          "90",   "96", "102", "108", "114",
          "120", "126" ];
```

The **lead** entry is an array of strings specifying lead times in HH[MMSS] format. By default, diagnostics are computed every 6 hours out to 126 hours. Lead times for which no track point or gridded model data exist produce a warning message and diagnostics set to a missing data value.

26.2.2.2 Configuring Domain Information

```
domain_info = [
   {
      domain         = "parent";
      n_range        = 150;
      n_azimuth      = 8;
      delta_range_km = 10.0;
      diag_script    = [ "MET_BASE/python/tc_diag/compute_tc_diag.py MET_BASE/python/tc_diag/
↪config/post_resample.yml MET_BASE/tc_data/v2023-04-07_gdland_table.dat" ];
      override_diags = [];
 },
   {
      domain         = "nest";
      n_range        = 150;
      n_azimuth      = 8;
      delta_range_km = 2.0;
      diag_script    = [ "MET_BASE/python/tc_diag/compute_tc_diag.py MET_BASE/python/tc_diag/
↪config/post_resample_nest.yml MET_BASE/tc_data/v2023-04-07_gdland_table.dat" ];
      override_diags = [ "RMW", "SST" ];
   }
];
```

The **domain_info** entry is an array of dictionaries. Each dictionary consists of five entries. The **domain** entry is a user-specified string that provides a name for the domain. Each **domain** name must also appear in a **-deck** command line option, and the reverse is also true.

The **n_range** entry is an integer specifying the number of equally spaced range intervals in the range-azimuth grid to be used for this data source.

The **n_azimuth** entry is an integer specifying the number of equally spaced azimuth intervals in the range-azimuth grid to be used for this data source. The azimuthal grid spacing is 360 / **n_azimuth** degrees. Azimuths are defined by MET as *degrees clockwise* from due east. However, the TC-Diag Python code expects

them as *radians counter-clockwise* from due east. The **tc_diag_driver/post_resample_driver.py** driver script performs the neccessary rotation and conversion operations.

The **delta_range_km** entry is a floating point value specifying the spacing of the range rings in kilometers.

The **diag_script** entry is an array of strings. Each string specifies the path to a Python script to be executed to compute diagnostics from the transformed cylindrical coordinates data for this domain. When multiple Python diagnostics scripts are run, the union of the diagnostics computed are written to the output.

The **override_diags** entry is an array of strings. Each string specifies the name of diagnostic value to be used for that domain. If set to an empty list, all diagnostics computed by the Python scripts in **diag_script** for that domain will be used. If non-empty, only the specific diagnostics listed will be used.

In the default configuration, seen above, the same Python script is run for both the *parent* and *nest* domains, each using a different configuration file. For the *parent* domain, all computed diagnostics are used since **override_diags** is empty. For the *nest* domain, only the specific diagnostics listed in **override_diags** are used to override the *parent* values. In general, diagnostics computed earlier in the list of **domain_info** entries can be overridden by diagnostics computed later in the list.

26.2.2.3 Configuring Data Censoring and Conversion Options

```
censor_thresh = [];
censor_val    = [];
convert(x)    = x;
```

These data censoring and conversion options are common to multiple MET tools and are described in Section 5. They do not actually appear in the default configuration file but can be specified separately in each **data.field** array entry, described below. If provided, those operations are performed after reading the gridded data but prior to converting to the cylindrical coordinate range-azimuth grid.

26.2.2.4 Configuring regridding options

```
regrid = {
   method     = BILIN;
   width      = 2;
   vld_thresh = 0.5;
   shape      = SQUARE;
}
```

The **regrid** dictionary is common to multiple MET tools and is described in Section 5. It specifies how the input data should be regridded to cylindrical coordinates prior to compute diagnostics. It can be specified separately in each **data.field** array entry, described below. The default setting uses bilinear interpolation for all fields.

26.2.2.5 Configuring Fields, Levels, and Domains

```
data = {

   // If empty, the field is processed for all domains
   domain = [];

   // Pressure levels to be used, unless overridden below
   level = [ "P1000", "P925", "P850", "P700", "P500",
             "P400",  "P300", "P250", "P200", "P150",
             "P100" ];

   field = [
      { name = "TMP";                  },
      { name = "UGRD";                 },
      { name = "VGRD";                 },
      { name = "RH";                   },
      { name = "HGT";                  },
      { name = "PRMSL"; level = "Z0";  },
      { name = "PWAT";  level = "L0";  },
      { name = "TMP";   level = "Z0";  },
      { name = "TMP";   level = "Z2";  },
      { name = "RH";    level = "Z2";  },
      { name = "UGRD";  level = "Z10"; },
      { name = "VGRD";  level = "Z10"; }
   ];
}
```

The **data** entry is a dictionary that contains the **field** entry to define what gridded data should be processed. The **field** entry is an array of dictionaries. Each **field** dictionary consists of at least three entries.

The **name** and **level** entries are common to multiple MET tools and are described in Section 5.

The **domain** entry is an array of strings. Each string specifies a domain name. If the **domain_info** domain name appears in this **domain** list, then this field will be read from that **domain_info** data source. If **domain** is set to an empty list, then this field will be read from all domain data sources.

26.2.2.6 Configuring Vortex Removal Option

```
vortex_removel = FALSE;
```

The **vortex_removal** flag entry is a boolean specifying whether or not vortex removal logic should be applied.

Note: As of MET version 12.0.0, vortex removal logic is not yet supported.

26.2.2.7 Configuring Data Input and Output Options

```
one_time_per_file_flag = TRUE;
```

The **one_time_per_file_flag** entry controls the logic for reading data from input files. This option describes how data is stored in the gridded input files specified with the **-data** command line option. Set this to true if each input file contains all of the data for a single initialization time and for a single valid time. If the input files contain data for multiple initialization or valid times, or if data for one valid time is spread across multiple files, set this to false.

If true, all input fields are read efficiently from each file in a single call. If false, each field is processed separately in a less efficient manner.

```
nc_cyl_grid_flag = TRUE; // resulting output file ends with "_cyl_grid_{domain}.nc"
nc_diag_flag     = TRUE; // resulting output file ends with "_diag.nc"
cira_diag_flag   = TRUE; // resulting output file ends with "_diag.dat"
```

These three flag entries are booleans specifying what output data types should be written. At least one of these flags must be set to true.

- The **nc_cyl_grid_flag** entry controls the writing of a NetCDF file containing the cylindrical coordinate range-azimuth data used to compute the diagnostics. These files are written with a *_cyl_grid_*{*domain*}*.nc* suffix, where {*domain*} is the domain name specified in the configuration file. One output file is written for each combination of model track and domain.
- The **nc_diag_file** entry controls the writing of the computed diagnostics to a NetCDF file. These files are written with a *_diag.nc* suffix. One output file is written for each model track processed.
- The **cira_diag_flag** entry controls the writing of the computed diagnostics to a formatted ASCII output file. These files are written with a *_diag.dat* suffix. One output file is written for each model track processed.

```
output_base_format = "s{storm_id}_{model}_doper_{init_time}";
```

The **output_base_format** entry is a string that defines the naming convention that should be used when writing the output files described above. The following keywords are supported and will be replaced with values from the corresponding track: {storm_id}, {basin}, {cyclone}, {storm_name}, {technique_number}, {technique}, {init_ymdh}, {init_ymd_hms}, {init_hour}.

26.2.3 tc_diag Output

The TC-Diag tool writes up to three output data types, as specified by flags in the configuration file. Each time TC-Diag is run it processes track data for a single initialization time. The actual number of output files varies depending on the number of model tracks provided.

CIRA Diagnostics Output

When the **cira_diag_flag** configuration entry is set to true, an ASCII CIRA diagnostics output file is written for each model track provided. These files are named using the **output_base_format**, described above, followed by the **_diag.dat** suffix.

These output files contain tabular ASCII data with diagnostic values either extracted directly from the input ATCF track file or computed from the gridded data, after converting it to a storm-centric cylindrical grid. One output file is created for each track from each model source. The output consists of the following sections:

- Two header lines list the model name, initialization time, storm basin, and integer storm number (of the season).
- The **STORM DATA** section contains single diagnostic values either extracted from the ATCF track file or computed from the cylindrical grid for each forecast lead time. This section begins with a line named **TIME** defining the forecast lead time of each track point in hours. The following lines contain the requested storm diagnostics. For example, **MAXWIND** contains the maximum wind speed reported in the ATCF track file and **SST** contains the average sea surface temperature computed in the range/azimuth grid.
- The **SOUNDING DATA** section contains diagnostics computed separately for each vertical level. The vertical levels are typically the surface (e.g. **SURF**) followed by pressure levels (e.g. **0850**). This section begins with two lines named **NLEV** and **TIME** defining the number of vertical levels and their values and the forecast lead times for which diagnostics were computed, respectively. The level name is appended to each diagnostic name. For example, the **T_0850** contains the average temperature value within the range/azimuth grid at the 850 mb pressure level.
- Each diagnostic output line contains:
  - Diagnostic name (with or without the level) e.g. **SHR_MAG** for magnitude of wind shear
  - Units string enclosed in parenthesis e.g. **(KT)** for knots
  - The diagnostic values computed for each lead time e.g. **13 10 14 . . .**

NetCDF Diagnostics Output

When the **nc_diag_flag** configuration entry is set to true, a NetCDF output file containing the computed diagnostics is written for each model track provided. These files contain the same data provided in the CIRA Diagnostics Output but formatted in NetCDF instead of ASCII. These files are named using the **output_base_format**, described above, followed by the **_diag.nc** suffix.

Table 26.1: Dimensions defined in NetCDF Diagnostics output

NetCDF Dimension	Description
time	Time dimension for the number of track point valid times
pressure	Vertical dimension for the number of pressure levels

Table 26.2: Variables defined in NetCDF Diagnostics output

NetCDF Variable	Dimension	Description	Data Type
storm_id	NA	Tropical Cyclone Storm ID (BBNNYYYY) consisting of 2-letter basin name, 2-digit storm number, and 4-digit year	String
model	NA	Track ATCF ID model name	String
init_time	NA	Track initialization time string in YYYYMMDD_HHMMSS format	Date-time String
init_time_ut	NA	Track initialization time string in unixtime (seconds since January 1, 1970) format	String
valid_time	time	Track point valid time string in YYYYMMDD_HHMMSS format	Date-time String
valid_time_ut	time	Track point valid time string in unixtime (seconds since January 1, 1970) format	String
lead_time	time	Track point forecast lead time string in HHMMSS format	Time String
lead_time_sec	time	Track point forecast lead time integer number of seconds	Integer
{DOMAIN}_domain	NA	Attributes define the range/azimuth grid for the {DOMAIN} domain: **n_range**, **n_azimuth**, **delta_range_km**	Integer
Diagnostic values	time or time and pressure	Computed diagnostic values for each track point and, optionally, pressure level. The **units** attribute defines the units of the diagnostic values.	Double

NetCDF Range-Azimuth Output

When the **nc_rng_azi_flag** configuration entry is set to true, a NetCDF output file containing the cylindrical range-azimuth data is written for each combination of model track provided and domain specified. For example, if three model tracks are provided and data for both *parent* and *nest* domains are provided, six of these NetCDF output files will be written.

The NetCDF range-azimuth output is named using the **output_base_format**, described above, followed by **_cyl_grid_{DOMAIN}.nc**, where **{DOMAIN}** is specified by the **domain** string in each **domain_info** array entry.

This NetCDF file contains a concatenation of the data from the temporary NetCDF files created for each track point. For each track point, TC-Diag creates a temporary NetCDF file and calls Python code to read the cylindrical grid data and compute diagnostics. By default, these temporary NetCDF files are deleted at the end of each run, but if the **nc_rng_azi_flag** is true, the data for each track point is concatenated into a single output file for each track.

Note: Setting the **MET_KEEP_TEMP_FILE** (Section 5.1.8) environment variable retains the temporary NetCDF cylindrical grid files for development, testing, and debugging purposes.

The NetCDF range-azimuth file contains the dimensions and variables shown in Table 26.3 and Table 26.4.

Table 26.3: Dimensions defined in NetCDF Range-Azimuth output

NetCDF Dimension	Description
track_line	Dimension for the raw ATCF track lines written to the **TrackLines** variable
time	Time dimension for the number of track point valid times
range	Dimension for the number of range rings in the range-azimuth grid
azimuth	Dimension for the number of azimuths in the range-azimuth grid
pressure	Vertical dimension for the number of pressure levels

Table 26.4: Variables defined in NetCDF Range-Azimuth output

NetCDF Variable	Dimension	Description	Data Type
storm_id	NA	Tropical Cyclone Storm ID (BBNNYYYY) consisting of 2-letter basin name, 2-digit storm number, and 4-digit year	String
model	NA	Track ATCF ID model name	String
TrackLines	track_lines	Raw input ATCF track lines	String
TrackLat	time	Track point location latitude	Double
TrackLon	time	Track point location longitude	Double
TrackMSLP	time	Track point minimum sea level pressure	Double
TrackVMax	time	Track point maximum wind speed	Double
init_time	NA	Track initialization time string in YYYYMMDD_HHMMSS format	Date-time String
init_time_ut	NA	Track initialization time string in unixtime (seconds since January 1, 1970) format	String
valid_time	time	Track point valid time string in YYYYMMDD_HHMMSS format	Date-time String
valid_time_ut	time	Track point valid time string in unixtime (seconds since January 1, 1970) format	String
lead_time	time	Track point forecast lead time string in HHMMSS format	Time String
lead_time_sec	time	Track point forecast lead time integer number of seconds	Integer
range	range	Range ring coordinate variable in kilometers	Double
azimuth	azimuth	Azimuth coordinate variable in degrees clockwise from north	Double
pressure	pressure	Vertical level pressure coordinate variable in millibars	Double
lat	time, range, azimuth	Latitude in degrees north for each range-azimuth grid point	Double
lon	time, range, azimuth	Longitude in degrees east for each range-azimuth grid point	Double
single level data (e.g. TMP_Z2, PRMSL_L0)	time, range, azimuth	Gridded range-azimuth data on a single level	Double
pressure level data (e.g. TMP, HGT)	time, pressure, range, azimuth	Gridded range-azimuth data on pressure levels	Double

Chapter 27

TC-Stat Tool

27.1 Introduction

The TC-Stat tool ties together results from the TC-Pairs tool by providing summary statistics and filtering jobs on TCST output files. The TC-Stat tool requires TCST output from the TC-Pairs tool. See Section 27.3.2 of this user's guide for information on the TCST output format of the TC-Pairs tool. The TC-Stat tool supports several analysis job types. The **filter** job stratifies the TCST data using various conditions and thresholds described in Section 27.3.1.3. The **summary** job produces summary statistics including frequency of superior performance, time-series independence calculations, and confidence intervals on the mean. The **rirw** job processes TCMPR lines, identifies adeck and bdeck rapid intensification or weakening events, populates a 2x2 contingency table, and derives contingency table statistics. The **probrirw job** processes PROBRIRW lines, populates an Nx2 probabilistic contingency table, and derives probabilistic statistics. The statistical aspects are described in Section 27.2, and practical use information for the TC-Stat tool is described in Section 27.3.

27.2 Statistical Aspects

27.2.1 Filter TCST Lines

The TC-Stat tool can be used to simply filter specific lines of the TCST file based on user-defined filtering criteria. All of the TCST lines that are retained from one or more files are written out to a single output file. The output file is also in TCST format.

Filtering options are outlined below in Section 27.3.1.3 (configuration file). If multiple filtering options are listed, the job will be performed on their intersection.

27.2.2 Summary Statistics for Columns

The TC-Stat tool can be used to produce summary information for a single column of data. After the user specifies the specific column of interest, and any other relevant search criteria, summary information is produced from values in that column of data. The summary statistics produced are listed in Table 27.1.

Confidence intervals are computed for the mean of the column of data. Confidence intervals are computed using the assumption of normality for the mean. For further information on computing confidence intervals, refer to Section 36 of the MET user's guide.

When operating on columns, a specific column name can be listed (e.g. TK_ERR), as well as the differences of two columns (e.g. AMAX_WIND-BMAX_WIND), and the absolute difference of the column(s) (e.g. abs(AMAX_WIND-BMAX_WIND)). Additionally, several shortcuts can be applied to choose multiple columns with a single entry. Shortcut options for the -column entry are as follows:

TRACK: track error (TK_ERR), along-track error (ALTK_ERR), and cross-track error (CRTK_ERR)

WIND: all wind radii errors (34-, 50-, and 64-kt) for each quadrant

TI: track error (TK_ERR) and absolute intensity error (abs(AMAX_WIND-BMAX_WIND))

AC: along- and cross-track errors (ALTK_ERR, CRTK_ERR)

XY: X- and Y-component track errors (X_ERR, Y_ERR)

The TC-Stat tool can also be used to generate frequency of superior performance and the time to independence calculations when using the TC-Stat summary job.

27.2.2.1 Frequency of Superior Performance

The frequency of superior performance (FSP) looks at multiple model forecasts (adecks), and ranks each model relative to other model performance for the column specified in the summary job. The summary job output lists the total number of cases included in the FSP, the number of cases where the model of interest is the best (e.g.: lowest track error), the number of ties between the models, and the FSP (percent). Ties are not included in the FSP percentage; therefore the percentage may not equal 100%.

27.2.2.2 Time-Series Independence

The time-series independence evaluates effective forecast separation time using the Siegel method, by comparing the number of runs above and below the mean error to an expected value. This calculation expects the columns in the summary job to be a time series. The output includes the forecast hour interval and the number of hours to independence.

27.2.3 Rapid Intensification/Weakening

The TC-Stat tool can be used to read TCMPR lines and compare the occurrence of rapid intensification (i.e. increase in intensity) or weakening (i.e. decrease in intensity) between the adeck and bdeck. The rapid intensification or weakening is defined by the change of maximum wind speed (i.e. **AMAX_WIND** and **BMAX_WIND** columns) over a specified amount of time. Accurately forecasting large changes in intensity is a challenging problem and this job helps quantify a model's ability to do so.

Users may specify several job command options to configure the behavior of this job. Using these configurable options, the TC-Stat tool analyzes paired tracks and for each track point (i.e. each TCMPR line) determines whether rapid intensification or weakening occurred. For each point in time, it uses the forecast and BEST track event occurrence to populate a 2x2 contingency table. The job may be configured to require that forecast and BEST track events occur at exactly the same time to be considered a hit. Alternatively, the job may be configured to define a hit as long as the forecast and BEST track events occurred within a configurable time window. Using this relaxed matching criteria false alarms may be considered hits and misses may be considered correct negatives as long as the adeck and bdeck events were close enough in time. Each rirw job applies a single intensity change threshold. Therefore, assessing a model's performance with rapid intensification and weakening requires that two separate jobs be run.

The RIRW job supports the **-out_stat** option to write the contingency table counts and statistics to a STAT output file.

27.2.4 Probability of Rapid Intensification

The TC-Stat tool can be used to accumulate multiple PROBRIRW lines and derive probabilistic statistics summarizing performance. The PROBRIRW line contains a probabilistic forecast for a specified intensity change along with the actual intensity change that occurred in the BEST track. Accurately forecasting the likelihood of large changes in intensity is a challenging problem and this job helps quantify a model's ability to do so.

Users may specify several job command options to configure the behavior of this job. The TC-Stat tools reads the input PROBI lines, applies the configurable options to extract a forecast probability value and BEST track event, and bins those probabilistic pairs into an Nx2 contingency table. This job writes up to four probabilistic output line types summarizing the performance.

27.3 Practical Information

The following sections describe the usage statement, required arguments, and optional arguments for tc_stat.

27.3.1 tc_stat Usage

The usage statement for tc_stat is shown below:

```
Usage: tc_stat
       -lookin source
       [-out file]
       [-log file]
       [-v level]
       [-config file] | [JOB COMMAND LINE]
```

TC-Stat has one required argument and accepts optional ones.

The usage statement for the TC-Stat tool includes the "job" term, which refers to the set of tasks to be performed after applying user-specified filtering options. The filtering options are used to pare down the TC-Pairs output to only those lines that are desired for the analysis. The job and its filters together comprise a "job command line". The "job command line" may be specified either on the command line to run a single analysis job or within the configuration file to run multiple analysis jobs at the same time. If jobs are specified in both the configuration file and the command line, only the jobs indicated in the configuration file will be run. The various jobs are described in Table 27.1 and the filtering options are described in Section 27.3.1.3.

27.3.1.1 Required Arguments for tc_stat

1. The **-lookin source** argument indicates the location of the input TCST files generated from tc_pairs. This argument can be used one or more times to specify the name of a TCST file or top-level directory containing TCST files to be processed. Multiple tcst files may be specified by using a wild card (*).

2. Either a configuration file must be specified with the **-config** option, or a **JOB COMMAND LINE** must be denoted. The **JOB COMMAND LINE** options are described in Section 27.3.1.3.

27.3.1.2 Optional Arguments for tc_stat

3. The **-out file** argument indicates the desired name of the TCST format output file.

4. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.

5. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

6. The **-config file** argument indicates the name of the configuration file to be used. The contents of the configuration file are discussed below.

An example of the tc_stat calling sequence is shown below:

```
tc_stat -lookin /home/tc_pairs/*al092010.tcst -config TCStatConfig
```

In this example, the TC-Stat tool uses any TCST file (output from tc_pairs) in the listed directory for the 9th Atlantic Basin storm in 2010. Filtering options and aggregated statistics are generated following configuration options specified in the **TCStatConfig** file. Further, using flags (e.g. **-basin, -column, -storm_name,** etc...) option within the job command lines may further refine these selections. See Section 27.3.1.3 for options available for the job command line and Section 6 for how to use them.

27.3.1.3 tc_stat Configuration File

The default configuration file for the **TC-Stat** tool named **TCStatConfig_default** can be found in the installed *share/met/config* directory. Like the other configuration files described in this document, it is recommended that users make a copy of these files prior to modifying their contents.

The contents of the tc_stat configuration file are described below.

```
storm_id     = [];
basin        = [];
cyclone      = [];
storm_name   = [];
init_beg     = "";
init_end     = "";
init_inc     = [];
init_exc     = [];
valid_beg    = "";
valid_end    = "";
valid_inc    = [];
valid_exc    = [];
init_hour    = [];
lead_req     = [];
init_mask    = [];
valid_mask   = [];
match_points = TRUE;
version      = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 6.

Note that the options specified in the first section of the configuration file, prior to the job list, will be applied to every job specified in the joblist. However, if an individual job specifies an option listed above, it will be applied to that job. For example, if model = [**"GFSI", "LGEM", "DSHP"**]; is set at the top, but the job in the joblist sets the -model option to "**LGEM**", that job will only run using the LGEM model data.

```
amodel = [];
bmodel = [];
```

The **amodel** and **bmodel** fields stratify by the amodel and bmodel columns based on a comma-separated list of model names used for all analysis performed. The names must be in double quotation marks (e.g.: "HWFI"). The **amodel** list specifies the model to be verified against the listed bmodel. The **bmodel** specifies the reference dataset, generally the BEST track analysis. Using the **-amodel** and **-bmodel** options within the job command lines may further refine these selections.

```
valid_hour = [];
lead       = [];
```

The **valid_hour**, and **lead** fields stratify by the valid time and lead time, respectively. This field specifies a comma-separated list of valid times and lead times in **HH[MMSS]** format. Using the **-valid_hour** and **-lead** options within the job command lines may further refine these selections.

```
line_type = [];
```

The **line_type** field stratifies by the line_type column.

```
track_watch_warn = [];
```

The **track_watch_warn** flag stratifies over the watch_warn column in the TCST files. If any of the watch/warning statuses are present in a forecast track, the entire track is verified. The value "ALL" matches HUWARN, HUWATCH, TSWARN, TSWATCH. Using the **-track_watch_warn** option within the job command lines may further refine these selections.

Other uses of the WATCH_WARN column include filtering when:

1. A forecast is issued when a watch/warning is in effect
2. A forecast is verifying when a watch/warning is in effect
3. A forecast is issued when a watch/warning is NOT in effect
4. A forecast is verified when a watch/warning is NOT in effect

The following filtering options can be achieved by the following:

Option 1. A forecast is issued when a watch/warning is in effect

```
init_str_name = ["WATCH_WARN"];
init_str_val = ["ALL"];
```

Option 2. A forecast is verifying when a watch/warning is in effect

```
column_str_name = ["WATCH_WARN"];
column_str_val = ["ALL"];
```

Option 3. A forecast is issued when a watch/warning is NOT in effect

```
init_str_name = ["WATCH_WARN"];
init_str_val = ["NA"];
```

Option 4. A forecast is verified when a watch/warning is NOT in effect

```
column_str_name = ["WATCH_WARN"];
column_str_val = ["NA"];
```

Further information on the **column_str** and **init_str** fields are described below. Listing a comma-separated list of watch/warning types in the **column_str_val** field will be stratified by a single or multiple types of warnings.

```
column_thresh_name = [];
column_thresh_val  = [];
```

The **column_thresh_name** and **column_thresh_val** fields stratify by applying thresholds to numeric data columns. Specify a comma-separated list of column names and thresholds to be applied. The length of **column_thresh_val** should match that of **column_thresh_name**. Using the **-column_thresh name thresh** option within the job command lines may further refine these selections.

```
column_str_name = [];
column_str_val  = [];
```

The **column_str_name** and **column_str_val** fields stratify by performing string matching on non-numeric data columns. Specify a comma-separated list of columns names and values to be **included** in the analysis. The length of the **column_str_val** should match that of the **column_str_name**. Using the **-column_str name value** option within the job command lines may further refine these selections.

```
column_str_exc_name = [];
column_str_exc_val  = [];
```

The **column_str_exc_name** and **column_str_exc_val** fields stratify by performing string matching on non-numeric data columns. Specify a comma-separated list of columns names and values to be **excluded** from the analysis. The length of the **column_str_exc_val** should match that of the **column_str_exc_name**. Using the **-column_str_exc name value** option within the job command lines may further refine these selections.

```
init_thresh_name = [];
init_thresh_val  = [];
```

The **init_thresh_name** and **init_thresh_val** fields stratify by applying thresholds to numeric data columns only when lead = 0. If lead = 0, but the value does not meet the threshold, discard the entire track. The length of the **init_thresh_val** should match that of the **init_thresh_name**. Using the **-init_thresh name thresh** option within the job command lines may further refine these selections.

```
init_str_name = [];
init_str_val  = [];
```

The **init_str_name** and **init_str_val** fields stratify by performing string matching on non-numeric data columns only when lead = 0. If lead = 0, but the string **does not** match, discard the entire track. The length of the **init_str_val** should match that of the **init_str_name**. Using the **-init_str name value** option within the job command lines may further refine these selections.

```
init_str_exc_name = [];
init_str_exc_val  = [];
```

The **init_str_exc_name** and **init_str_exc_val** fields stratify by performing string matching on non-numeric data columns only when lead = 0. If lead = 0, and the string **does** match, discard the entire track. The length of the **init_str_exc_val** should match that of the **init_str_exc_name**. Using the **-init_str_exc name value** option within the job command lines may further refine these selections.

```
diag_thresh_name = [];
diag_thresh_val  = [];
```

The **diag_thresh_name** and **diag_thresh_val** fields stratify individual track points by applying thresholds to numeric data columns from the TCDIAG lines. Specify a comma-separated list of diagnostics names and thresholds to be applied. The length of **diag_thresh_val** should match that of **diag_thresh_name**. If the storm diagnostic does not meet the threshold, discard both the TCMPR and TCDIAG lines for that track point. Using the **-diag_thresh name thresh** option within the job command lines may further refine these selections.

```
init_diag_thresh_name = [];
init_diag_thresh_val  = [];
```

The **init_diag_thresh_name** and **init_diag_thresh_val** fields stratify entire tracks by applying thresholds to numeric data columns from the TCDIAG lines, but only when lead = 0. If lead = 0, but the storm diagnostic does not meet the threshold, discard the entire track. The length of the **init_diag_thresh_val** should match that of the **init_diag_thresh_name**. Using the **-init_diag_thresh name thresh** option within the job command lines may further refine these selections.

```
water_only = FALSE;
```

The **water_only** flag stratifies by only using points where both the amodel and bmodel tracks are over water. When **water_only = TRUE;** once land is encountered the remainder of the forecast track is not used for the verification, even if the track moves back over water.

```
rirw = {
   track  = NONE;
   time   = "24";
   exact  = TRUE;
   thresh = >=30.0;
}
```

The **rirw** field specifies those track points for which rapid intensification (RI) or rapid weakening (RW) occurred, based on user defined RI/RW thresholds. The **track** entry specifies that RI/RW is not turned on **(NONE)**, is computed based on the bmodel only **(BDECK)**, is computed based on the amodel only **(ADECK)**, or computed when both the amodel and bmodel (the union of the two) indicate RI/RW (BOTH). If **track** is set to **ADECK, BDECK**, or **BOTH**, only tracks exhibiting rapid intensification will be retained. Rapid intensification is officially defined as when the change in the maximum wind speed over a 24-hour period is greater than or equal to 30 kts. This is the default setting, however flexibility in this definition is provided through the use of the **time, exact** and **thresh** options. The **time** field specifies the time window (HH[MMSS] format) for which the RI/RW occurred. The **exact** field specifies whether to only count RI/RW when the intensity change is over the exact time window (TRUE), which follows the official RI definition, or if the intensity threshold is met anytime during the time window (FALSE). Finally, the **thresh** field specifies the user defined intensity threshold (where ">=" indicates RI, and "<=" indicates RW).

Using the **-rirw_track, -rirw_time_adeck, -rirw_time_bdeck, -rirw_exact_adeck, -rirw_exact_bdeck, -rirw_thresh_adeck, -rirw_thresh_bdeck** options within the job command lines may further refine these selections. See Section 6 for how to use these options.

```
landfall     = FALSE;
landfall_beg = "-24";
landfall_end = "00";
```

The **landfall, landfall_beg**, and **landfall_end** fields specify whether only those track points occurring near landfall should be retained. The landfall retention window is defined as the hours offset from the time of landfall. Landfall is defined as the last bmodel track point before the distance to land switches from water to land. When **landfall_end** is set to zero, the track is retained from the **landfall_beg** to the time of landfall. Using the **-landfall_window** option with the job command lines may further refine these selections. The **-landfall_window** job command option takes one or two arguments in HH[MMSS] format. Use one argument to define a symmetric time window. For example, **-landfall_window 06** defines the time window +/- six hours around the landfall time. Use two arguments to define an asymmetric time window. For example, **-landfall_window 00 12** defines the time window from the landfall event to twelve hours after.

```
event_equal = FALSE;
```

The **event_equal** flag specifies whether only those track points common to all models in the dataset should be retained. The event equalization is performed only using cases common to all listed amodel entries. A case is defined by comparing the following columns in the TCST files: BMODEL, BASIN, CYCLONE, INIT, LEAD, VALID. This option may be modified using the **-event_equal** option within the job command lines.

```
event_equal_lead = [];
```

The **event_equal_lead** flag specifies lead times that must be present for a track to be included in the event equalization logic. The event equalization is performed only using cases common to all lead times listed, enabling the verification at each lead time to be performed on a consistent dataset. This option may be modified using the **-event_equal_lead** option within the job command lines.

```
out_init_mask = "";
```

The **out_init_mask** field applies polyline masking logic to the location of the amodel track at the initialization time. If the track point falls outside the mask, discard the entire track. This option may be modified using the **-out_init_mask** option within the job command lines.

```
out_valid_mask = "";
```

The **out_valid_mask** field applies polyline masking logic to the location of the amodel track at the valid time. If the track point falls outside the mask, discard the entire track. This option may be modified using the **-out_valid_mask** option within the job command lines.

```
jobs = [];
```

The user may specify one or more analysis jobs to be performed on the TCST lines that remain after applying the filtering parameters listed above. Each entry in the joblist contains the task and additional filtering options for a single analysis to be performed. The available job types include *filter*, *summary*, *rirw*, and *probrirw*. Please refer to Section 5 for details on how to call each job. The format for an analysis job is as follows:

```
-job job_name REQUIRED and OPTIONAL ARGUMENTS

e.g.: -job filter  -line_type TCMPR  -amodel HWFI   -dump_row ./tc_filter_job.tcst
      -job summary -line_type TCMPR  -column TK_ERR -dump_row ./tc_summary_job.tcst
      -job rirw    -line_type TCMPR  -rirw_time 24 -rirw_exact false -rirw_thresh ge20
      -job rirw    -line_type TCMPR  -rirw_time 24 -rirw_exact false -rirw_thresh ge20 -out_
↪stat ./tc_rirw.stat
      -job probrirw -line_type PROBRIRW -column_thresh RI_WINDOW ==24 \
                    -probrirw_thresh 30 -probrirw_prob_thresh ==0.25
```

27.3.2 tc_stat Output

The output generated from the TC-Stat tool contains statistics produced by the analysis. Additionally, it includes information about the analysis job that produced the output for each line. The output can be redirected to an output file using the **-out** option. The format of output from each tc_stat job command is listed below.

Job: Filter

This job command finds and filters TCST lines down to those meeting the criteria selected by the filter's options. The filtered TCST lines are written to a file specified by the **-dump_row** option. The TCST output from this job follows the TCST output description in Section 24 and Section 25.

> The "-set_hdr" job command option can be used to override any of the output header strings (e.g. "-set_hdr DESC EVENT_EQUAL" sets the output DESC column to "EVENT_EQUAL").

Job: Summary

This job produces summary statistics for the column name specified by the **-column** option. The output of the summary job consists of three rows:

1. "JOB_LIST", which shows the job definition parameters used for this job;

2. "COL_NAME", followed by the summary statistics that are applied;

3. "SUMMARY", which is followed by the total, mean (with confidence intervals), standard deviation, minimum value, percentiles (10th, 25th, 50th, 75th, 90th), maximum value, interquartile range, range, sum, time to independence, and frequency of superior performance.

The output columns are shown below in Table 27.1 The **-by** option can also be used one or more times to make this job more powerful. Rather than running the specified job once, it will be run once for each unique combination of the entries found in the column(s) specified with the **-by** option.

Table 27.1: Columnar output of "summary" job output from the TC-Stat tool.

Column number	Description
1	SUMMARY: (job type)
2	Column (dependent parameter)
3	Case (storm + valid time)
4	Total
5	Valid
6-8	Mean, including normal upper and lower confidence limits
9	Standard deviation
10	Minimum value
11-15	Percentiles (10th, 25th, 50th, 75th, 90th)
16	Maximum Value
17	Interquartile range (75th - 25th percentile)
18	Range (Maximum - Minimum)
19	Sum
20-21	Independence time
22-25	Frequency of superior performance

Job: RIRW

The RIRW job produces contingency table counts and statistics defined by identifying rapid intensification or weakening events in the adeck and bdeck track. Users may specify several job command options to configure the behavior of this job:

- The **-rirw_time HH[MMSS]** option (or **-rirw_time_adeck** and **-rirw_time_bdeck** to specify different settings) defines the time window of interest. The default is 24 hours.
- The **-rirw_exact bool** option (or **-rirw_exact_adeck** and **-rirw_exact_bdeck** to specify different settings) is a boolean defining whether the exact intensity change or maximum intensity change over that time window should be used. For rapid intensification, the maximum increase is computed. For rapid weakening, the maximum decrease is used. The default is true.
- The **-rirw_thresh threshold** option (or **-rirw_thresh_adeck** and **-rirw_thresh_bdeck** to specify different settings) defines the intensity change event threshold. The default is greater than or equal to 30 kts.
- The **-rirw_window** option may be passed one or two arguments in HH[MMSS] format to define how close adeck and bdeck events must be to be considered hits or correct negatives. One time string defines a symmetric time window while two time strings define an asymmetric time window. The default is 0, requiring an exact match in time.
- The **-out_line_type** option defines the output data that should be written. This job can write contingency table counts (CTC), contingency table statistics (CTS), and RIRW matched pairs (MPR). The default is CTC and CTS, but the MPR output provides a great amount of detail.

Users may also specify the **-out_alpha** option to define the alpha value for the confidence intervals in the CTS output line type. In addition, the **-by column_name** option is a convenient way of running the same job across multiple stratifications of data. For example, **-by AMODEL** runs the same job for each unique AMODEL name in the data.

Users may also specify the **-out_stat** option to write the contingency table counts and statistics (for the CTC and CTS output line types) to an output STAT file. Information about the RIRW timing information and filtering criteria are written to the STAT header columns while the contingency table counts and/or statistics are written to the CTC and/or CTS output columns.

When using the "-out_stat" option to create a .stat output file and stratifying results using one or more "-by" job command options, those columns may be referenced in the "-set_hdr" option.

```
-job rirw -line_type TCMPR -by CYCLONE -out_stat ctc.stat -set_hdr DESC CYCLONE
```

When using multiple "-by" options, use "CASE" to reference the full case information string.

```
-job rirw -line_type TCMPR -by CYCLONE,LEAD -out_stat ctc.stat -set_hdr DESC CASE
```

Job: PROBRIRW

The PROBRIRW job produces probabilistic contingency table counts and statistics defined by placing forecast probabilities and BEST track rapid intensification events into an Nx2 contingency table. Users may specify several job command options to configure the behavior of this job:

- The **-prob_thresh n** option is required and defines which probability threshold should be evaluated. It determines which **PROB_i** column from the PROBRIRW line type is selected for the job. For example,

use **-prob_thresh 30** to evaluate forecast probabilities of a 30 kt increase or use **-prob_thresh -30** to evaluate forecast probabilities of a 30 kt decrease in intensity. The default is a 30 kt increase.

- The **-prob_exact bool** option is a boolean defining whether the exact or maximum BEST track intensity change over the time window should be used. If true, the values in the **BDELTA** column are used. If false, the values in the **BDELTA_MAX** column are used. The default is true.
- The **-probrirw_bdelta_thresh** threshold option defines the BEST track intensity change event threshold. This should typically be set consistent with the probability threshold (**-prob_thresh**) chosen above. The default is greater than or equal to 30 kts.
- The **-probrirw_prob_thresh threshold_list** option defines the probability thresholds used to create the output Nx2 contingency table. The default is probability bins of width 0.1. These probabilities may be specified as a list (>0.00,>0.25,>0.50,>0.75,>1.00) or using shorthand notation (==0.25) for bins of equal width.
- The **-out_line_type** option defines the output data that should be written. This job can write PCT, PSTD, PJC, and PRC output line types. The default is PCT and PSTD. Please see Table 11.8 through Table 11.11 for more details.

Users may also specify the **-out_alpha** option to define the alpha value for the confidence intervals in the PSTD output line type. Multiple values in the **RI_WINDOW** column cannot be combined in a single PROBRIRW job since the BEST track intensity threshold should change for each. Using the **-by RI_WINDOW** option or **-column_thresh RI_WINDOW ==24** option provide convenient ways avoiding this problem.

Users should note that for the PROBRIRW line type, **PROBRI_PROB** is a derived column name. The -probrirw_thresh option defines the probabilities of interest (e.g. **-probrirw_thresh 30**) and the **PROBRI_PROB** column name refers to those probability values, regardless of their column number. For example, the job command options **-probrirw_thresh 30 -column_thresh PROBRI_PROB >0** select 30 kt probabilities and match probability values greater than 0.

Chapter 28

TC-Gen Tool

28.1 Introduction

The TC-Gen tool provides verification of deterministic and probabilistic tropical cyclone genesis forecasts in the ATCF file and shapefile formats. Producing reliable tropical cyclone genesis forecasts is an important metric for global numerical weather prediction models. This tool ingests deterministic model output post-processed by genesis tracking software (e.g. GFDL vortex tracker), ATCF edeck files containing probability of genesis forecasts, operational shapefile warning areas, and ATCF reference track dataset(s) (e.g. Best Track analysis and CARQ operational tracks). It writes categorical counts and statistics. The capability to modify the spatial and temporal tolerances when matching forecasts to reference genesis events, as well as scoring those matched pairs, gives users the ability to condition the criteria based on model performance and/or conduct sensitivity analyses. Statistical aspects are outlined in Section 28.2 and practical aspects of the TC-Gen tool are described in Section 28.3.

28.2 Statistical Aspects

The TC-Gen tool processes both deterministic and probabilistic forecasts.

For deterministic forecasts specified using the **-track** command line option, it identifies genesis events in both the forecasts and reference datasets, typically Best tracks. It applies user-specified configuration options to pair up the forecast and reference genesis events and categorize each pair as a hit, miss, or false alarm.

As with other extreme events (where the event occurs much less frequently than the non-event), the correct negative category is not computed since the non-events would dominate the contingency table. Therefore, only statistics that do not include correct negatives should be considered for this tool. The following CTS statistics are relevant: Base rate (BASER), Mean forecast (FMEAN), Frequency Bias (FBIAS), Probability of Detection (PODY), False Alarm Ratio (FAR), Critical Success Index (CSI), Gilbert Skill Score (GSS), Extreme Dependency Score (EDS), Symmetric Extreme Dependency Score (SEDS), Bias-Adjusted Gilbert Skill Score (BAGSS).

For probabilistic forecasts specified using the **-edeck** command line option, it identifies genesis events in the reference dataset. It applies user-specified configuration options to pair the forecast probabilities to the reference genesis events. These pairs are added to an Nx2 probabilistic contingency table. If the reference

genesis event occurs within in the predicted time window, the pair is counted in the observation-yes column. Otherwise, it is added to the observation-no column.

For warning area shapefiles specified using the **-shape** command line option, it processes metadata from the corresponding database files. The database file is assumed to exist at exactly the same path as the shapefile, but with a ".dbf" suffix instead of ".shp". Note that only shapefiles exactly following the NOAA National Hurricane Center's (NHC) "gtwo_areas_YYYYMMDDHHMM.shp" file naming and corresonding metadata conventions are supported. For each shapefile record, the database file defines corresponding probability values for one or more time periods. Percentages may be provided for the probability of genesis inside the shape within 2, 5, or 7 days from issuance time that is parsed from the file name. Note that 5 day probabilities were discontinued in 2023. The 2 and 7 day probabilities are provided in database file fields named "PROB2DAY" and "PROB7DAY", respectively. Care is taken to identify and either ignore or update duplicate shapes found in the input.

The shapes are then subset based on the filtering criteria in the configuration file. For each probability and shape, the reference genesis events are searched for a match within the defined time window. These pairs are added to an Nx2 probabilistic contingency table. The probabilistic contingeny tables and statistics are computed and reported separately for filter defined and lead hour encountered in the input.

Other considerations for interpreting the output of the TC-Gen tool involve the size of the contingency table output. The size of the contingency table will change depending on the number of matches. Additionally, the number of misses is based on the forecast duration and interval (specified in the configuration file). This change is due to the number of model opportunities to forecast the event, which is determined by the specified duration/interval.

Care should be taken when interpreting the statistics for filtered data. In some cases, variables (e.g. storm name) are only available in either the forecast or reference datasets, rather than both. When filtering on a field that is only present in one dataset, the contingency table counts will be impacted. Similarly, the initialization field only impacts the model forecast data. If the valid time (which will impact the reference dataset) isn't also specified, the forecasts will be filtered and matched such that the number of misses will erroneously increase. See Section 28.3 for more detail.

28.3 Practical Information

This section describes how to configure and run the TC-Gen tool. The following sections describe the usage statement, required arguments, and optional arguments for tc_gen.

28.3.1 tc_gen Usage

The usage statement for tc_gen is shown below:

```
Usage: tc_gen
       -genesis source
       -edeck source
       -shape source
       -track source
       -config file
```

(continues on next page)

(continued from previous page)

```
      [-out base]
      [-log file]
      [-v level]
```

TC-Gen has three required arguments and accepts optional ones.

28.3.1.1 Required Arguments for tc_gen

1. The **-genesis source** argument is the path to one or more ATCF or fort.66 (see documentation listed below) files generated by the Geophysical Fluid Dynamics Laboratory (GFDL) Vortex Tracker when run in tcgen mode or an ASCII file list or a top-level directory containing them. The required file format is described in the "Output formats" section of the GFDL Vortex Tracker users guide.

2. The **-edeck source** argument is the path to one or more ATCF edeck files, an ASCII file list containing them, or a top-level directory with files matching the regular expression ".dat". The probability of genesis are read from each edeck input file and verified against at the **-track** data.

3. The **-shape source** argument is the path to one or more NHC genesis warning area shapefiles, an ASCII file list containing them, or a top-level directory with files matching the regular expression "gtwo_areas.*.shp". The genesis warning areas and corresponding forecast probability values area verified against the **-track** data.

Note: At least one of the **-genesis**, **-edeck**, or **-shape** command line options are required.

4. The **-track source** argument is one or more ATCF reference track files or an ASCII file list or top-level directory containing them, with files ending in ".dat". This tool processes either Best track data from bdeck files, or operational track data (e.g. CARQ) from adeck files, or both. Providing both bdeck and adeck files will result in a richer dataset to match with the **-genesis** files. Both adeck and bdeck data should be provided using the **-track** option. The **-track** option must be used at least once.

5. The **-config** file argument indicates the name of the configuration file to be used. The contents of the configuration file are discussed below.

28.3.1.2 Optional Arguments for tc_gen

6. The **-out base** argument indicates the path of the output file base. This argument overrides the default output file base (./tc_gen)

7. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no log file.

8. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

28.3.1.3 Scoring Logic

The TC-Gen tool implements the following logic:

- Parse the Best and operational track data, and identify Best track genesis events. Note that Best tracks with a cyclone number greater than 50 are automatically discarded from the analysis. Large cyclone numbers are used for pre-season testing or to track invests prior to a storm actually forming. Running this tool at verbosity level 6 (-v 6) prints details about which tracks are discarded.
- For **-track** inputs:
  - Parse the forecast genesis data and identify forecast genesis events separately for each model present.
  - Loop over the filters defined in the configuration file and apply the following logic for each.
    - For each Best track genesis event meeting the filter critera, determine the initialization and lead times for which the model had an opportunity to forecast that genesis event. Store an unmatched genesis pair for each case.
    - For each forecast genesis event, search for a matching Best track. A configurable boolean option controls whether all Best track points are considered for a match or only the single Best track genesis point. A match occurs if the Best track point valid time is within a configurable window around the forecast genesis time and the Best track point location is within a configurable radius of the forecast genesis location. If a Best track match is found, store the storm ID.
    - If no Best track match is found, apply the same logic to search the operational track points with lead time of 0 hours. If an operational match is found, store the storm ID.
    - If a matching storm ID is found, match the forecast genesis event to the Best track genesis event for that storm ID.
    - If no matching storm ID is found, store an unmatched pair for the genesis forecast.
    - Loop through the genesis pairs and populate contingency tables using two methods, the development (dev) and operational (ops) methods. For each pair, if the forecast genesis event is unmatched, score it as a dev and ops FALSE ALARM. If the Best track genesis event is unmatched, score it as a dev and ops MISS. Score each matched genesis pair as follows:
    - If the forecast initialization time is at or after the Best track genesis event, DISCARD this case and exclude it from the statistics.
    - Compute the difference between the forecast and Best track genesis events in time and space. If they are both within the configurable tolerance, score it as a dev HIT. If not, score it as a dev FALSE ALARM.
    - Compute the difference between the Best track genesis time and model initialization time. If it is within the configurable tolerance, score it as an ops HIT. If not, score it as an ops FALSE ALARM.
    - Do not count any CORRECT NEGATIVES.

    - Report the contingency table hits, misses, and false alarms separately for each forecast model and configuration file filter. The development (dev) scoring method is indicated in the output as *GENESIS_DEV* while the operational (ops) scoring method is indicated as *GENESIS_OPS*.

- For **-edeck** inputs:

    - Parse the ATCF edeck files. Ignore any lines not containing "GN" and "genFcst", which indicate a genesis probability forecast. Also, ignore any lines which do not contain a predicted genesis location (latitude and longitude) or genesis time.

    - Loop over the filters defined in the configuration file and apply the following logic for each.

    - Subset the genesis probability forecasts based on the current filter criteria. Typically, genesis probability forecast are provided for multiple lead times. Create separate Nx2 probabilistic contingency tables for each unique combination of predicted lead time and model name.

    - For each genesis probability forecast, search for a matching Best track. A configurable boolean option controls whether all Best track points are considered for a match or only the single Best track genesis point. A match occurs if the Best track point valid time is within a configurable window around the forecast genesis time and the Best track point location is within a configurable radius of the forecast genesis location. If a Best track match is found, store the storm ID.

    - If no Best track match is found, apply the same logic to search the operational track points with lead time of 0 hours. If an operational match is found, store the storm ID.

    - If no matching storm ID is found, add the unmatched forecast to the observation-no column of the Nx2 probabilistic contingency table.

    - If a matching storm ID is found, check whether that storm's genesis occurred within the predicted time window: between the forecast initialization time and the predicted lead time. If so, add the matched forecast to the observation-yes column. If not, add it to observation-no column.

    - Report the Nx2 probabilistic contingency table counts and statistics for each forecast model, lead time, and configuration file filter. These counts and statistics are identified in the output files as *PROB_GENESIS*.

- For **-shape** inputs:

    - For each input shapefile, parse the timestamp from the "gtwo_areas_YYYYMMDDHHMM.shp" naming convention, and error out otherwise. Round the timestamp to the nearest synoptic time (e.g. 00, 06, 12, 18) and store that as the issuance time.

    - Open the shapefile and corresponding database file. Process each record.

    - For each record, extract the shape and metadata which defines the basin and 2, 5, and 7 day probabilities.

    - Check if this shape is a duplicate that has already been processed. If it is an exact duplicate, with the same basin, file timestamp, issue time, and min/max lat/lon values, ignore it. If the file timestamp is older than the existing shape, also ignore it. If the file timestamp is newer than the existing shape, replace the existing shape with the new one.

- Loop over the filters defined in the configuration file and apply the following logic for each.
- Subset the list of genesis shapes based on the current filter criteria.
- Search the Best track genesis events to see if any occurred inside the shape within 7 days of the issuance time. If multiple genesis events occurred, choose the one closest to the issuance time.
- If not found, score each probability as a miss.
- If found, further check the 2 and 5 day time windows to classify each probability as a hit or miss.
- Add each probability pair to an Nx2 probabilistic contingency table, tracking results separately for each lead time.
- Report the Nx2 probabilistic contingency table counts and statistics for each lead time. These counts and statistics are identified in the output files as *GENESIS_SHAPE*.

28.3.2 tc_gen Configuration File

The default configuration file for the **TC-Gen** tool named **TCGenConfig_default** can be found in the installed *share/met/config* directory. Like the other configuration files described in this document, it is recommended that users make a copy of these files prior to modifying their contents.

The tc_gen configuration file is divided into three main sections: criteria to define genesis events, options to subset and filter those events, and options to control the output. The contents of this configuration file are described below.

```
init_freq = 6;
```

The **init_freq** variable is an integer specifying the model initialization frequency in hours, starting at 00Z. The default value of 6 indicates that the model is initialized every day at 00Z, 06Z, 12Z, and 18Z. The same frequency is applied to all models processed. Models initialized at different frequencies should be processed with separate calls to tc_gen. The initialization frequency is used when defining the model opportunities to forecast the Best track genesis events.

```
valid_freq = 6;
```

The **valid_freq** variable is an integer specifying the valid time of the track points to be analyzed in hours, starting at 00Z. The default value of 6 indicates that only track points with valid times of 00Z, 06Z, 12Z, and 18Z will be checked for genesis events. Since Best and operational tracks are typically only available at those times, a match to a forecast genesis event is only possible for those hours.

```
fcst_hr_window = {
   beg = 24;
   end = 120;
}
```

The **fcst_hr_window** option is a dictionary defining the beginning (**beg**) and ending (**end**) model forecast hours to be searched for genesis events. Model genesis events occurring outside of this window are ignored. This forecast hour window is also used when defining the model opportunities to forecast the Best track genesis events.

```
min_duration = 12;
```

The **min_duration** variable is an integer specifying the minimum number of hours a track must persist for its initial point to be counted as a genesis event. Some models spin up many short-lived storms, and this setting enables them to be excluded from the analysis.

```
fcst_genesis = {
   vmax_thresh = NA;
   mslp_thresh = NA;
}
```

The **fcst_genesis** dictionary defines the conditions required for a model track's genesis point to be included in the analysis. Thresholds for the maximum wind speed (**vmax_thresh**) and minimum sea level pressure (**mslp_thresh**) may be defined. These conditions must be satisfied for at least one track point for the genesis event to be included in the analysis. The default thresholds (**NA**) always evaluate to true.

```
best_genesis = {
   technique   = "BEST";
   category    = [ "TD", "TS" ];
   vmax_thresh = NA;
   mslp_thresh = NA;
}
```

The **best_genesis** dictionary defines genesis criteria for the Best tracks. Like the **fcst_genesis** dictionary, the **vmax_thresh** and **mslp_thresh** thresholds define required genesis criteria. In addition, the **category** array defines the ATCF storm categories that should qualify as genesis events. The **technique** string defines the ATCF ID for the Best track.

```
oper_technique = "CARQ";
```

The **oper_technique** entry is a string which defines the ATCF ID for the operational track data that should be used. For each forecast genesis event, the Best tracks are searched for a track point valid at the time of forecast genesis and within the search radius. If no match is found, the 0-hour operational track points are searched for a match.

```
filter = [];
```

The **filter** entry is an array of dictionaries defining genesis filtering criteria to be applied. Each of the entries listed below (from **desc** to **best_unique_flag**) may be specified separately within each filter dictionary. If left empty, the default setting, a single filter is applied using the top-level filtering criteria. If multiple filtering dictionaries are defined, the **desc** entry must be specified for each to differentiate the output data. Output is written for each combination of filter dictionary and model ATCF ID encountered in the data.

```
desc = "ALL";
```

The **desc** configuration option is common to many MET tools and is described in Section 5.

```
model = [];
```

The **model** entry is an array defining the model ATCF ID's for which output should be computed. If left empty, the default setting, output will be computed for each model encountered in the data. Otherwise, output will be computed only for the ATCF ID's listed. Note that when reading ATCF track data, all instances of the string AVN are automatically replaced with GFS.

```
storm_id   = [];
storm_name = [];
```

The **storm_id** and **storm_name** entries are arrays indicating the ATCF storm ID's and storm names to be processed. If left empty, all tracks will be processed. Otherwise, only those tracks which meet these criteria will be included. Note that these strings only appear in the Best and operational tracks, not the forecast genesis data. Therefore, these filters only apply to the Best and operational tracks. Care should be given when interpreting the contingency table results for filtered data.

```
init_beg = "";
init_end = "";
init_inc = [];
init_exc = [];
```

The **init_beg**, **init_end**, **init_inc**, and **init_exc** entries define strings in YYYYMMDD[_HH[MMSS]] format which defines which forecast and operational tracks initializations to be processed. If left empty, all tracks will be used. Otherwise, only those tracks whose initialization time meets all the criteria will be processed. The initialization time must fall between **init_beg**, and **init_end**, must appear in **init_inc** inclusion list, and must not appear in the **init_exc** exclusion list. Note that these settings only apply to the forecast and operational tracks, not the Best tracks, for which the initialization time is undefined. Care should be given when interpreting the contingency table results for filtered data.

For genesis shapes, these options are used to filter the warning issuance time.

```
valid_beg = "";
valid_end = "";
```

The **valid_beg** and **valid_end** entries are similar to **init_beg** and **init_end**, described above. However, they are applied to all genesis data sources. Only those tracks falling completely inside this window are included in the analysis.

```
init_hour = [];
lead      = [];
```

The **init_hour** and **lead** entries are arrays of strings in HH[MMSS] format defining which forecast tracks should be included. If left empty, all tracks will be used. Otherwise, only those forecast tracks whose initialization hour and lead times appear in the list will be used. Note that these settings only apply to the forecast tracks, not the Best tracks, for which the initialization time is undefined. Care should be given when interpreting the contingency table results for filtered data.

For genesis shapes, the **init_hour** option is used to filter the warning issuance hour.

```
vx_mask = "";
```

The **vx_mask** entry is a string defining the path to a Lat/Lon polyline file or a gridded data file that MET can read to subset the results spatially. If specified, only those genesis events whose Lat/Lon location falls within the specified area will be included.

If specified for genesis shapes, the lat/lon of the central location of the shape will be checked. The central location is computed as the average of the min/max lat/lon values of the shape points.

```
basin_mask = [];
```

The **basin_mask** entry is an array of strings listing tropical cycline basin abbreviations (e.g. AL, EP, CP, WP, NI, SI, AU, and SP). The configuration entry **basin_file** defines the path to a NetCDF file which defines these regions. The default file (**basin_global_tenth_degree.nc**) is bundled with MET. If **basin_mask** is left empty, genesis events for all basins will be included. If non-empty, the union of specified basins will be used. If **vx_mask** is also specified, the analysis is done on the intersection of those masking areas.

The **vx_mask** and **basin_mask** names are concatenated and written to the **VX_MASK** output column.

If **vx_mask** is not specified for genesis shapes and **basin_mask** is, the basin name is extracted from the shapefile metadata and compared to the **basin_mask** list.

```
dland_thresh = NA;
```

The **dland_thresh** entry is a threshold defining whether the genesis event should be included based on its distance to land. The default threshold (**NA**) always evaluates to true.

```
genesis_match_point_to_track = TRUE;
```

The **genesis_match_point_to_track** entry is a boolean which controls the matching logic. When set to its default value of TRUE, for each forecast genesis event, all Best track points are searched for a match. This logic implements the method used by the NOAA National Hurricane Center. When set to FALSE, only the single Best track genesis point is considered for a match. When selecting FALSE, users are encouraged to adjust the **genesis_match_radius** and/or **gensesis_match_window** options, described below, to enable matches to be found.

```
genesis_match_radius = 500;
```

The **genesis_match_radius** entry defines a search radius, in km, relative to the forecast genesis location. When searching for a match, only Best or operational tracks with a track point within this radius will be considered. Increasing this search radius should lead to an increase in the number of matched genesis pairs.

```
genesis_match_window = {
   beg = 0;
   end = 0;
}
```

The **genesis_match_window** entry defines a time window, in hours, relative to the forecast genesis time. When searching for a match, only Best or operational tracks with a track point falling within this time window will be considered. The default time window of 0 requires a Best or operational track to exist at the forecast genesis time for a match to be found. Increasing this time window should lead to an increase in the number matched genesis pairs. For example, setting *end = 12;* would allow forecast genesis events to match Best tracks up to 12 hours prior to their existence.

```
dev_hit_radius = 500;
```

The **dev_hit_radius** entry defines the maximum distance, in km, that the forecast and Best track genesis events may be separated in order for them to be counted as a contingency table HIT for the development scoring method. Users should set this hit radius less than or equal to the genesis match radius. Reducing this radius may cause development method HITS to become FALSE ALARMS.

```
dev_hit_window = {
   beg = -24;
   end =  24;
}
```

The **dev_hit_window** entry defines a time window, in hours, relative to the forecast genesis time. The Best track genesis event must occur within this time window for the pair to be counted as a contingency table HIT for the development scoring method. Tightening this window may cause development method HITS to become FALSE ALARMS.

```
ops_hit_window = {
   beg =  0;
   end = 48;
}
```

The **ops_hit_window** entry defines a time window, in hours, relative to the Best track genesis time. The model initialization time for the forecast genesis event must occur within this time window for the pairs to be counted as a contingency table HIT for the operationl scoring method. Otherwise, the pair is counted as a FALSE ALARM.

```
discard_init_post_genesis_flag = TRUE;
```

The **discard_init_post_genesis_flag** entry is a boolean which indicates whether or not forecast genesis events from model intializations occurring at or after the matching Best track genesis time should be discarded. If true, those cases are not scored in the contingency table. If false, they are included in the counts.

```
dev_method_flag = TRUE;
ops_method_flag = TRUE;
```

The **dev_method_flag** and **ops_method_flag** entries are booleans which indicate whether the development and operational scoring methods should be applied and written to the output. At least one of these flags must be set to true.

```
nc_pairs_flag = {
   latlon       = TRUE;
   fcst_genesis = TRUE;
   fcst_tracks  = TRUE;
   fcst_fy_oy   = TRUE;
   fcst_fy_on   = TRUE;
   best_genesis = TRUE;
   best_tracks  = TRUE;
   best_fy_oy   = TRUE;
   best_fn_oy   = TRUE;
}
```

The **nc_pairs_flag** entry is a dictionary of booleans indicating which fields should be written to the NetCDF genesis pairs output file. Each type of output is enabled by setting it to TRUE and disabled by setting it to FALSE. The **latlon** option writes the latitude and longitude values of the output grid. The remaining options write a count of the number of points occuring within each grid cell. The **fcst_genesis** and **best_genesis** options write counts of the forecast and Best track genesis locations. The **fcst_track** and **best_track** options write counts of the full set of track point locations, which can be refined by the **valid_minus_genesis_diff_thresh** option, described below. The **fcst_fy_oy** and **fcst_fy_on** options write counts for the locations of forecast genesis event HITS and FALSE ALARMS. The **best_fy_oy** and **best_fn_oy** options write counts for the locations of Best track genesis event HITS and MISSES. Note that since matching forecast and Best track genesis events may occur in different grid cells, their counts are reported separately.

```
valid_minus_genesis_diff_thresh = NA;
```

The **valid_minus_genesis_diff_thresh** is a threshold which affects the counts in the NetCDF pairs output file. The fcst_tracks and best_tracks options, described above, turn on counts for the forecast and Best track points. This option defines which of those track points should be counted by thresholding the track point valid time minus genesis time difference. If set to NA, the default threshold which always evaluates to true, all track points will be counted. Setting <=0 would count the genesis point and all track points prior. Setting >0 would count all points after genesis. And setting >=-12||<=12 would could all points within 12 hours of the genesis time.

```
best_unique_flag = TRUE;
```

The **best_unique_flag** entry is a boolean which affects the counts in the NetCDF pairs output file. If true, the Best track HIT and MISS locations are counted for each genesis pair. If false, each Best track genesis event is counted only once. If it is a HIT in at least one genesis pair, it is counted as a HIT in the output. Otherwise, it is counted as a MISS.

```
basin_file = "MET_BASE/tc_data/basin_global_tenth_degree.nc";
```

The **basin_file** entry defines the path to the NetCDF basin data file that is included with MET. When a Best track storm moves from one basin to another, the Best track dataset can include two tracks for the same storm, one for each basin. However, both tracks have the same genesis point. When this occurs, this basin data file is read and used to determine the basin in which genesis actually occurred. The corresponding Best track is retained and the other is discarded.

```
nc_pairs_grid = "G001";
```

The **nc_pairs_grid** entry is a string which defines the grid to be used for the NetCDF genesis pairs output file. It can be specified as a named grid, the path to a gridded data file, or a grid specification string.

```
prob_genesis_thresh = ==0.25;
```

The **prob_genesis_thresh** entry defines the probability thresholds used to create the output Nx2 contingency table when verifying edeck probability of genesis forecasts and probabilistic shapefile warning areas. The default is probability bins of width 0.25. These probabilities may be specified as a list (>0.00,>0.25,>0.50,>0.75,>1.00) or using shorthand notation (==0.25) for bins of equal width.

```
ci_alpha = 0.05;
output_flag = {
   fho    = BOTH;
   ctc    = BOTH;
   cts    = BOTH;
   pct    = NONE;
   pstd   = NONE;
   pjc    = NONE;
   prc    = NONE;
   genmpr = NONE;
}
dland_file = "MET_BASE/tc_data/dland_global_tenth_degree.nc";
version    = "VN.N";
```

The configuration options listed above are common to many MET tools and are described in Section 5. TC-Gen writes output for 2x2 contingency tables to the **FHO**, **CTC**, and **CTS** line types when verifying deterministic genesis forecasts specified using the **-track** command line option. TC-Gen writes output for Nx2 probabilistic contingency tables to the **PCT**, **PSTD**, **PJC**, and **PRC** line types when verifying the probability of genesis forecasts specified using the **-edeck** command line option and probabilistic shapefiles using the **-shape** command line option. Note that the **genmpr** line type is specific to TC-Gen and describes individual genesis matched pairs.

28.3.3 tc_gen Output

TC-Gen produces output in STAT and, optionally, ASCII and NetCDF formats. The ASCII output duplicates the STAT output but has the data organized by line type. The output files are created based on the **-out** command line argument. The default output base name, **./tc_gen** writes output files in the current working directory named **tc_gen.stat** and, optionally, **tc_gen_pairs.nc** and **tc_gen_{TYPE}.txt** for each of the supported output line types. These output files can easily be redirected to another location using the **-out** command line option. The format of the STAT and ASCII output of the TC-Gen tool matches the output of other MET tools with the exception of the genesis matched pair line type. Please refer to the tables in Section 11.3.3 for a description of the common output line types. The genesis matched pair line type and NetCDF output file are described below.

Table 28.1: Header information for TC-Gen STAT output

Column Number	Header Column Name	Description	Data Type
1	VERSION	Version number	String
2	MODEL	Current ATCF Technique name	String
3	DESC	User-provided text string describing the "filter" options	String
4	FCST_LEAD	Forecast lead time in HHMMSS format	Time String
5	FCST_VALID_BEG	Minimum forecast valid time in YYYYMMDD_HHMMSS format	Datetime String
6	FCST_VALID_END	Maximum forecast valid time in YYYYMMDD_HHMMSS format	Datetime String
7	OBS_LEAD	Does not apply and is set to NA	Time String
8	OBS_VALID_BEG	Minimum Best track valid time in YYYYMMDD_HHMMSS format	Datetime String
9	OBS_VALID_END	Maximum Best track valid time in YYYYMMDD_HHMMSS format	Datetime String
10	FCST_VAR	Genesis methodology (GENESIS_DEV, GENESIS_OPS, PROB_GENESIS, or GENESIS_SHAPE)	String
11	FCST_UNITS	Does not apply and is set to NA	String
12	FCST_LEV	Does not apply and is set to NA	String
13	OBS_VAR	Genesis methodology (GENESIS_DEV, GENESIS_OPS, PROB_GENESIS, or GENESIS_SHAPE)	String
14	OBS_UNITS	Does not apply and is set to NA	String
15	OBS_LEV	Does not apply and is set to NA	String
16	OBTYPE	Verifying Best track technique name	String
17	VX_MASK	Verifying masking region	String
18	INTERP_MTHD	Does not apply and is set to NA	String
19	INTERP_PNTS	Does not apply and is set to NA	Integer
20	FCST_THRESH	Does not apply and is set to NA	Threshold String
21	OBS_THRESH	Does not apply and is set to NA	Threshold String
22	COV_THRESH	Does not apply and is set to NA	Threshold String
23	ALPHA	Error percent value used in confidence intervals	Double
24	LINE_TYPE	Various line type options, refer to Section 11.3.3 and the tables below.	String

Table 28.2: Format information for GENMPR (Genesis Matched Pairs) output line type

Column Number	GENMPR Column Name	Description	Data Type
5, 6	FCST_VALID_BEG, FCST_VALID_END	Forecast genesis time in YYYYMMDD_HHMMSS format	Date-time String
8, 9	OBS_VALID_BEG, OBS_VALID_END	Best track genesis time in YYYYMMDD_HHMMSS format	Date-time String
24	GENMPR	Genesis Matched Pairs line type	String
25	TOTAL	Total number of genesis pairs	Integer
26	INDEX	Index for the current matched pair	Integer
27	STORM_ID	BBCCYYYY designation of storm (basin, cyclone number, and year)	String
28	PROB_LEAD	Lead time in HHH format for the predicted probability of genesis (only for **-edeck** inputs)	Time String
29	PROB_VAL	Predicted probability of genesis (only for **-edeck** inputs)	Double
30	AGEN_INIT	Forecast initialization time	Date-time String
31	AGEN_FHR	Forecast hour of genesis event	Time String
32	AGEN_LAT	Latitude position of the forecast genesis event	Double
33	AGEN_LON	Longitude position of the forecast genesis event	Double
34	AGEN_DLAND	Forecast genesis event distance to land (nm)	Double
35	BGEN_LAT	Latitude position of the verifying Best track genesis event	Double
36	BGEN_LON	Longitude position of the verifying Best track genesis event	Double
37	BGEN_DLAND	Best track genesis event distance to land (nm)	Double
38	GEN_DIST	Distance between the forecast and Best track genesis events (km) (only for **-track** inputs)	Double
39	GEN_TDIFF	Forecast minus Best track genesis time in HHMMSS format (only for **-track** inputs)	Time String
40	INIT_TDIFF	Best track genesis minus forecast initialization time in HHMMSS format (only for **-track** inputs)	Time String
41	DEV_CAT	Category for the development methodology (FYOY, FYON, FNOY, or DISCARD) (only for **-track** inputs)	String
42	OPS_CAT	Category for the operational methodology (FYOY, FYON, FNOY, or DISCARD for **-track** inputs and FYOY or FYON for **-edeck** inputs)	String

Table 28.3: A selection of variables that can appear in the NetCDF matched pair output which can be controlled by the nc_pairs_flag configuration option.

NetCDF Variable	Dimension	Description	Data Type
DESC_MODEL_GENESIS	lat, lon	For each filter entry (DESC) and forecast ATCF ID (MODEL), count the number of forecast genesis events within each grid box.	Float
DESC_MODEL_TRACKS	lat, lon	For each filter entry (DESC) and forecast ATCF ID (MODEL), count the number of track points within each grid box.	Float
DESC_BEST_GENESIS	lat, lon	For each filter entry (DESC), count the number of Best track genesis events within each grid box.	Float
DESC_BEST_GENESIS	lat, lon	For each filter entry (DESC), count the number of Best track points within each grid box.	Float
DESC_MODEL_[DEV|OPS]_FY_OY	lat, lon	For each filter entry (DESC) and forecast ATCF ID (MODEL), count the number of forecast genesis events classified as hits by the development (DEV) or operational (OPS) methodology.	Float
DESC_MODEL_[DEV|OPS]_FY_ON	lat, lon	For each filter entry (DESC) and forecast ATCF ID (MODEL), count the number of forecast genesis events classified as false alarms by the development (DEV) or operational (OPS) methodology.	Float
DESC_MODEL_BEST_[DEV|OPS]_FY_OY	lat, lon	For each filter entry (DESC) and forecast ATCF ID (MODEL), count the number of Best track genesis events classified as hits by the development (DEV) or operational (OPS) methodology.	Float
DESC_MODEL_BEST_[DEV|OPS]_FN_OY	lat, lon	For each filter entry (DESC) and forecast ATCF ID (MODEL), count the number of Best track genesis events classified as misses by the development (DEV) or operational (OPS) methodology.	Float

Like all STAT output, the output of TC-Gen may be further processed using the Stat-Analysis tool, described in Section 17.

Chapter 29

TC-RMW Tool

29.1 Introduction

The TC-RMW tool regrids tropical cyclone model data onto a moving range-azimuth grid centered on points along the storm track provided in ATCF format, most likely the adeck generated from the file. The radial grid spacing can be defined in kilometers or as a factor of the radius of maximum winds (RMW). The azimuthal grid spacing is defined in degrees clockwise from due east. If wind vector fields are specified in the configuration file, the radial and tangential wind components will be computed. Any regridding method available in MET can be used to interpolate data on the model output grid to the specified range-azimuth grid. The regridding will be done separately on each vertical level. The model data files must coincide with track points in a user-provided ATCF formatted track file.

29.2 Practical Information

29.2.1 tc_rmw Usage

The following sections describe the usage statement, required arguments, and optional arguments for tc_rmw.

```
Usage: tc_rmw
       -data file_1 ... file_n | file_list
       -deck file
       -config file
       -out file
       [-log file]
       [-v level]
```

tc_rmw has four required arguments and can accept several optional arguments.

29.2.1.1 Required Arguments for tc_rmw

1. The **-data file_1 ... file_n | file_list** option specifies the gridded data files or an ASCII file containing a list of files to be used, as described in Section 4.1.1.
2. The **-deck source** argument is the ATCF format data source.
3. The **-config file** argument is the configuration file to be used. The contents of the configuration file are discussed below.
4. The **-out** argument is the NetCDF output file to be written.

29.2.1.2 Optional Arguments for tc_rmw

5. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no logfile.
6. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

29.2.2 tc_rmw Configuration File

The default configuration file for the TC-RMW tool named **TCRMWConfig_default** can be found in the installed *share/met/config/* directory. It is encouraged for users to copy these default files before modifying their contents. The contents of the configuration file are described in the subsections below.

```
model =       "GFS";
censor_thresh = [];
censor_val    = [];
data  = {
   field = [
        {
           name = "PRMSL";
           level = ["L0"];
        },
        {
           name = "TMP";
           level = ["P1000", "P500"];
        },
        {
           name = "UGRD";
           level = ["P1000", "P500"];
        },
        {
```

(continues on next page)

(continued from previous page)

```
            name = "VGRD";
            level = ["P1000", "P500"];
        }
    ];
}
regrid = { ... }
```

The configuration options listed above are common to many MET tools and are described in Section 5. The name and level entries in the data dictionary define the data to be processed. The regrid dictionary defines if and how regridding will be performed.

```
n_range = 100;
```

The **n_range** parameter is the number of equally spaced range intervals in the range-azimuth grid.

```
n_azimuth = 180;
```

The **n_azimuth** parameter is the number of equally spaced azimuth intervals in the range-azimuth grid. The azimuthal grid spacing is 360 / **n_azimuth** degrees.

```
delta_range_km = 10.0;
```

The **delta_range_km** parameter specifies the spacing of the range rings, in kilometers. The range values start with 0 km and extend out to **n_range - 1** times this delta spacing.

```
rmw_scale = NA;
```

If changed from its default value of **NA**, the **rmw_scale** parameter overrides the **delta_range_km** parameter. The radial grid spacing is defined using **rmw_scale** in units of the RMW, which varies along the storm track. For example, setting **rmw_scale** to 0.2 would define the delta range spacing as 20% of the radius of maximum winds around each point. Note that RMW is defined in nautical miles but is converted to kilometers for this computation.

```
compute_tangential_and_radial_winds = TRUE;
```

The **compute_tangential_and_radial_winds** parameter is a flag controlling whether a conversion from U/V to Tangential/Radial winds is done or not. If set to TRUE, additional parameters are used, otherwise they are not.

```
u_wind_field_name = "UGRD";
v_wind_field_name = "VGRD";
```

The **u_wind_field_name** and **v_wind_field_name** parameters identify which input data to use in converting to tangential/radial winds. The parameters are used only if **compute_tangential_and_radial_winds** is set to TRUE.

```
tangential_velocity_field_name = "VT";
tangential_velocity_long_field_name = "Tangential Velocity";
```

The **tangential_velocity_field_name** and **tangential_velocity_long_field_name** parameters define the field names to give the output tangential velocity grid in the netCDF output file. The parameters are used only if **compute_tangential_and_radial_winds** is set to TRUE.

```
radial_velocity_field_name = "VT";
radial_velocity_long_field_name = "Radial Velocity";
```

The **radial_velocity_field_name** and **radial_velocity_long_field_name** parameters define the field names to give the output radial velocity grid in the netCDF output file. The parameters are used only if **compute_radial_and_radial_winds** is set to TRUE.

29.2.3 tc_rmw Output File

The NetCDF output file contains the following dimensions:

1. *track_point* - the track points corresponding to the model output valid times
2. *pressure* - if any pressure levels are specified in the data variable list, they will be sorted and combined into a 3D NetCDF variable, which pressure as the vertical dimension and range and azimuth as the horizontal dimensions
3. *range* - the radial dimension of the range-azimuth grid
4. *azimuth* - the azimuthal dimension of the range-azimuth grid

For each data variable specified in the data variable list, a corresponding NetCDF variable will be created with the same name and units.

Chapter 30

RMW-Analysis Tool

30.1 Introduction

The RMW-Analysis tool analyzes a set of output files generated by the TC-RMW tool. For each grid cell it aggregates variable statistics across the set and across the track points of the TC-RMW output files. The statistics are mean, standard deviation, minimum and maximum. Note that TC-RMW should be set to use the same scale factor of the radius of maximum winds (RMW) as the unit of range for its range-azimuth grid. The specified data variables on the range-azimuth-vertical grid then share a common range scale of RMW before aggregation by RMW-Analysis.

30.2 Practical Information

30.2.1 rmw_analysis Usage

The following sections describe the usage statement, required arguments, and optional arguments for **rmw_analysis**.

```
Usage: rmw_analysis
       -data file_1 ... file_n | file_list
       -config file
       -out file
       [-log file]
       [-v level]
```

rmw_analysis has three required arguments and can accept several optional arguments.

30.2.1.1 Required Arguments for rmw_analysis

1. The **-data file_1 ... file_n | file_list** option specifies the NetCDF TC-RMW output files or ASCII file list of file names to be processed, as described in Section 4.1.1.
2. The **-config file** argument is the **RMWAnalysisConfig** to be used. The contents of the configuration file are discussed below.
3. The **-out** file argument is the NetCDF output file to be written.

30.2.1.2 Optional Arguments for rmw_analysis

4. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no logfile.
5. The **-v level** option indicates the desired level of verbosity. The contents of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity above 1 will increase the amount of logging.

30.2.2 rmw_analysis Configuration File

The default configuration file for the RMW-Analysis tool named **RMWAnalysisConfig_default** can be found in the installed *share/met/config/* directory. It is encouraged for users to copy these default files before modifying their contents. The contents of the configuration file are described in the subsections below.

```
data = {
   level = [ "" ];
   field = [
      { name = "PRMSL"; },
      { name = "TMP";   }
   ];
}

version = "VN.N";
```

The `data` dictionary specifies the name of the 2D and 3D gridded variables to be processed from the TC-RMW NetCDF output files. Its formatting is the same as the `fcst` and `obs` dictionaries, used in the MET statistics tools. However, while the `level` string must be specified, it is not actually used and can remain as its default setting of an empty string. TC-RMW reads data for all variables requested and summarizes it through time for all tracks that meet the filtering criteria described below.

The configuration options listed above are common to many MET tools and are described in Section 5.

```
model      = [];
storm_id   = [];
basin      = [];
cyclone    = [];
storm_name = [];

init_beg   = "";
init_end   = "";
init_inc   = [];
init_exc   = [];

valid_beg  = "";
valid_end  = "";
valid_inc  = [];
valid_exc  = [];

init_hour  = [];
valid_hour = [];
lead       = [];

init_mask  = "";
valid_mask = "";

category   = [];

column_thresh_name = [];
column_thresh_val  = [];

init_thresh_name = [];
init_thresh_val  = [];
```

The NetCDF output from TC-RMW contains ATCF-formatted storm track information in the `TrackLines` variable. The RMW-Analysis tool parses that track information and applies the filtering criteria listed above. Filtering is only applied for configuration entries that are non-empty lists or strings. The corresponding gridded data is only used for track points that meet all specified filtering criteria.

The `column_thresh_name` and `init_thresh_name` arrays specify the names of ATCF columns whose values should be checked. The former applies to each individual track point while the latter applies to the initial track point (i.e. lead time equals 0). If the filtering criteria is not satisfied for the initial time, the entire track is discarded. Only values from select ATCF columns (LAT, LON, VMAX, MSLP, POUTER, ROUTER, RMW, GUSTS, EYE, DIR, and SPEED) can be thresholded numerically with these options to filter the input data processed.

These configuration options are described in Section 6.

30.2.3 rmw_analysis Output File

The NetCDF output file will inherit the spatial grid from the first TC-RMW output file in the output file list. All TC-RMW files in this list must have the same grid dimension sizes, although their center (lat, lon) location can vary. A NetCDF output error will result if that is not the case. For each data variable specified in the config file, four corresponding NetCDF variables will be written. For example, for `TMP` input, output variables named `TMP_mean`, `TMP_stdev`, `TMP_min`, and `TMP_max` will be written containing the mean, standard deviation, minimum, and maximum temperature values for each RMW grid location. No track point dimension is retained in the RMW-Analysis output, but the `TrackLat_mean` and `TrackLon_mean` output variables specify the average location of the track points processed.

Chapter 31

Plotting and Graphics Support

31.1 Plotting Utilities

This section describes how to check your data files using plotting utilities. Point observations can be plotted using the Plot-Point-Obs utility. A single model level can be plotted using the plot_data_plane utility. For object based evaluations, the MODE objects can be plotted using plot_mode_field. Occasionally, a post-processing or timing error can lead to errors in MET. These tools can assist the user by showing the data to be verified to ensure that times and locations match up as expected.

31.1.1 plot_point_obs Usage

The usage statement for the Plot-Point-Obs utility is shown below:

```
Usage: plot_point_obs
       nc_file
       ps_file
       [-config config_file]
       [-point_obs file]
       [-title string]
       [-plot_grid name]
       [-gc code] or [-obs_var name]
       [-msg_typ name]
       [-dotsize val]
       [-log file]
       [-v level]
```

plot_point_obs has two required arguments and can take optional ones.

31.1.1.1 Required Arguments for plot_point_obs

1. The **nc_file** argument indicates the name of the point observation file to be plotted. This file is the output from one of the point pre-processing tools, such as pb2nc. Python embedding for point observations is also supported, as described in Section 38.4.2.
2. The **ps_file** argument indicates the name given to the output file containing the plot.

31.1.1.2 Optional Arguments for plot_point_obs

3. The **-config config_file** option specifies the configuration file to be used. The contents of the optional configuration file are discussed below.
4. The **-point_obs file** option is used to pass additional NetCDF point observation files to be plotted.
5. The **-plot_grid name** option defines the grid for plotting as a named grid, the path to a gridded data file, or an explicit grid specification string. This overrides the default global plotting grid. If configuring the tool to plot a base image, the path to the input gridded data file should be specified here.
6. The **-title string** option specifies the plot title string.
7. The **-gc code** and **-obs_var name** options specify observation types to be plotted. These overrides the corresponding configuration file entries.
8. The **-msg_typ name** option specifies the message type to be plotted. This overrides the corresponding configuration file entry.
9. The **-dotsize val** option sets the dot size. This overrides the corresponding configuration file entry.
10. The **-log file** option directs output and errors to the specified log file.
11. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

An example of the plot_point_obs calling sequence is shown below:

```
plot_point_obs sample_pb.nc sample_data.ps
```

In this example, the Plot-Point-Obs tool will process the input sample_pb.nc file and write a postscript file containing a plot to a file named sample_pb.ps.

An equivalent command using python embedding for point observations is shown below. Note that the entire python command is enclosed in single quotes to prevent embedded whitespace for causing parsing errors:

```
plot_point_obs 'PYTHON_NUMPY=MET_BASE/python/examples/read_met_point_obs.py sample_pb.nc'␣
↪sample_data.ps
```

Please see section Section 38.4.2 for more details about Python embedding in MET.

31.1.2 plot_point_obs Configuration File

The default configuration file for the Plot-Point-Obs tool named **PlotPointObsConfig_default** can be found in the installed *share/met/config* directory. The contents of the configuration file are described in the subsections below.

Note that environment variables may be used when editing configuration files, as described in the Section 5.1.1.

```
version        = "VN.N";
```

The configuration options listed above are common to multiple MET tools and are described in Section 5.

```
grid_data = {

   field = [];

   regrid = {
      to_grid    = NONE;
      method     = NEAREST;
      width      = 1;
      vld_thresh = 0.5;
      shape      = SQUARE;
   }

   grid_plot_info = {
      color_table   = "MET_BASE/colortables/met_default.ctable";
      plot_min      = 0.0;
      plot_max      = 0.0;
      colorbar_flag = TRUE;
   }
}
```

The **grid_data** dictionary defines a gridded field of data to be plotted as a base image prior to plotting point locations on top of it. The data to be plotted is specified by the **field** array. If **field** is empty, no base image will be plotted. If **field** has length one, the requested data will be read from the input file specified by the **-plot_grid** command line argument.

The **to_grid** entry in the **regrid** dictionary specifies if and how the requested gridded data should be regridded prior to plotting. Please see Section 5 for a description of the **regrid** dictionary options.

The **grid_plot_info** dictionary inside **grid_data** specifies the options for for plotting the gridded data. The options within **grid_plot_info** are described in Section 5.

```
point_data = [
  { fill_color = [ 255, 0, 0 ]; }
];
```

The **point_data** entry is an array of dictionaries. Each dictionary may include a list of filtering, data processing, and plotting options, described below. For each input point observation, the tool checks the **point_data** filtering options in the order specified. The point information is added to the first matching array entry. The default entry simply specifies that all points be plotted red.

```
msg_typ     = [];
sid_inc     = [];
sid_exc     = [];
obs_var     = [];
obs_quality = [];
```

The options listed above define filtering criteria for the input point observation strings. If empty, no filtering logic is applied. If a comma-separated list of strings is provided, only those observations meeting all of the criteria are included. The **msg_typ** entry specifies the message type. The **sid_inc** and **sid_exc** entries explicitly specify station id's to be included or excluded. The **obs_var** entry specifies the observation variable names, and **obs_quality** specifies quality control strings.

```
obs_gc      = [];
```

When using older point observation files which have GRIB codes, the **obs_gc** entry specifies a list of integer GRIB codes to be included.

```
valid_beg   = "";
valid_end   = "";
```

The **valid_beg** and **valid_end** options are time strings which specify a range of dates to be included. When left to their default empty strings no time filtering is applied.

```
lat_thresh  = NA;
lon_thresh  = NA;
elv_thresh  = NA;
hgt_thresh  = NA;
prs_thresh  = NA;
obs_thresh  = NA;
```

The options listed above define filtering thresholds for the input point observation values. The default NA thresholds always evaluate to true and therefore apply no filtering. The **lat_thresh** and **lon_thresh** thresholds filter the latitude and longitude of the point observations, respectively. The **elv_thresh** threshold filters by the station elevation. The **hgt_thresh** and **prs_thresh** thresholds filter by the observation height and pressure level. The **obs_thresh** threshold filters by the observation value.

```
convert(x)    = x;
censor_thresh = [];
censor_val    = [];
```

The **convert(x)** function, **censor_thresh** option, and **censor_val** option may be specified separately for each **point_data** array entry to transform the observation values prior to plotting. These options are further described in Section 5.

```
dotsize(x) = 1.0;
```

The **dotsize(x)** function defines the size of the circle to be plotted as a function of the observation value. The default setting shown above defines the dot size as a constant value.

```
line_color = [];
line_width = 1;
```

The **line_color** and **line_width** entries define the color and thickness of the outline for each circle plotted. When **line_color** is left as an empty array, no outline is drawn. Otherwise, **line_color** should be specified using 3 intergers between 0 and 255 to define the red, green, and blue components of the color.

```
fill_color = [];
fill_plot_info = { // Overrides fill_color
  flag          = FALSE;
  color_table   = "MET_BASE/colortables/met_default.ctable";
  plot_min      = 0.0;
  plot_max      = 0.0;
  colorbar_flag = TRUE;
}
```

The circles are filled in based on the setting of the **fill_color** and **fill_plot_info** entries. As described above for **line_color**, if **fill_color** is empty, the points are not filled in. Otherwise, **fill_color** must be specified using 3 integers between 0 and 255. If **fill_plot_info.flag** is set to true, then its settings override **fill_color**. The **fill_plot_info** dictionary defines a colortable which is used to determine the color to be used based on the observation value.

Users are encouraged to define as many **point_data** array entries as needed to filter and plot the input observations in the way they would like. Each point observation is plotted using the options specified in the first matching array entry. Note that the filtering, processing, and plotting options specified inside each **point_data** array entry take precedence over ones specified at the higher level of configuration file context.

For each observation, this tool stores the observation latitude, longitude, and value. However, unless the **dotsize(x)** function is not constant or the **fill_plot_info.flag** entry is set to true, the observation value is simply set to a flag value. For each **point_data** array entry, the tool stores and plots only the unique combination of observation latitude, longitude, and value. Therefore multiple obsevations at the same location will typically be plotted as a single circle.

31.1.3 plot_data_plane Usage

The usage statement for the plot_data_plane utility is shown below:

```
Usage: plot_data_plane
       input_filename
       output_filename
       field_string
       [-color_table color_table_name]
       [-plot_range min max]
       [-title title_string]
       [-log file]
       [-v level]
```

plot_data_plane has two required arguments and can take optional ones.

31.1.3.1 Required Arguments for plot_data_plane

1. The **input_filename** argument indicates the name of the gridded data file to be plotted.
2. The **output_filename** argument indicates the name given to the output PostScript file containing the plot.
3. The **field_string** argument contains information about the field and level to be plotted.

31.1.3.2 Optional Arguments for plot_data_plane

4. The **-color_table color_table_name** overrides the default color table (*MET_BASE/colortables/met_default.ctable*)
5. The **-plot_range min max** sets the minimum and maximum values to plot.
6. The **-title title_string** sets the title text for the plot.
7. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no logfile.
8. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default setting of 2. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

An example of the plot_data_plane calling sequence is shown below:

```
plot_data_plane test.grb test.ps 'name="TMP"; level="Z2";'
```

A second example of the plot_data_plane calling sequence is shown below:

```
plot_data_plane test.grb2 test.ps 'name="DSWRF"; level="L0";' -v 4
```

In the first example, the Plot-Data-Plane tool will process the input test.grb file and write a PostScript image to a file named test.ps showing temperature at 2 meters. The second example plots downward shortwave radiation flux at the surface. The second example is run at verbosity level 4 so that the user can inspect the output and make sure its plotting the intended record.

31.1.4 plot_mode_field Usage

The usage statement for the plot_mode_field utility is shown below:

```
Usage: plot_mode_field
       mode_nc_file_list
       -raw | -simple | -cluster
       -obs | -fcst
       -config file
       [-log file]
       [-v level]
```

plot_mode_field has four required arguments and can take optional ones.

31.1.4.1 Required Arguments for plot_mode_field

1. The **mode_nc_file_list** specifies the MODE output files to be used for plotting.
2. The **-raw | -simple | -cluster** argument indicates the types of fields to be plotted. Exactly one must be specified. For details about the types of objects, see the section in this document on MODE.
3. The **-obs | -fcst** option specifies whether to plot the observed or forecast field from the MODE output files. Exactly one must be specified.
4. The **-config file** specifies the configuration file to use for specification of plotting options.

31.1.4.2 Optional Arguments for plot_mode_field

5. The **-log file** option directs output and errors to the specified log file. All messages will be written to that file as well as standard out and error. Thus, users can save the messages without having to redirect the output on the command line. The default behavior is no logfile.
6. The **-v level** option indicates the desired level of verbosity. The value of "level" will override the default. Setting the verbosity to 0 will make the tool run with no log messages, while increasing the verbosity will increase the amount of logging.

An example of the plot_mode_field calling sequence is shown below:

```
plot_mode_field -simple -obs -config \
plotMODEconfig mode_120000L_20050807_120000V_000000A_obj.nc
```

In this example, the plot_mode_field tool will plot simple objects from an observed precipitation field using parameters from the configuration file plotMODEconfig and objects from the MODE output file

mode_120000L_20050807_120000V_000000A_obj.nc. An example plot showing twelve simple observed precipitation objects is shown below.

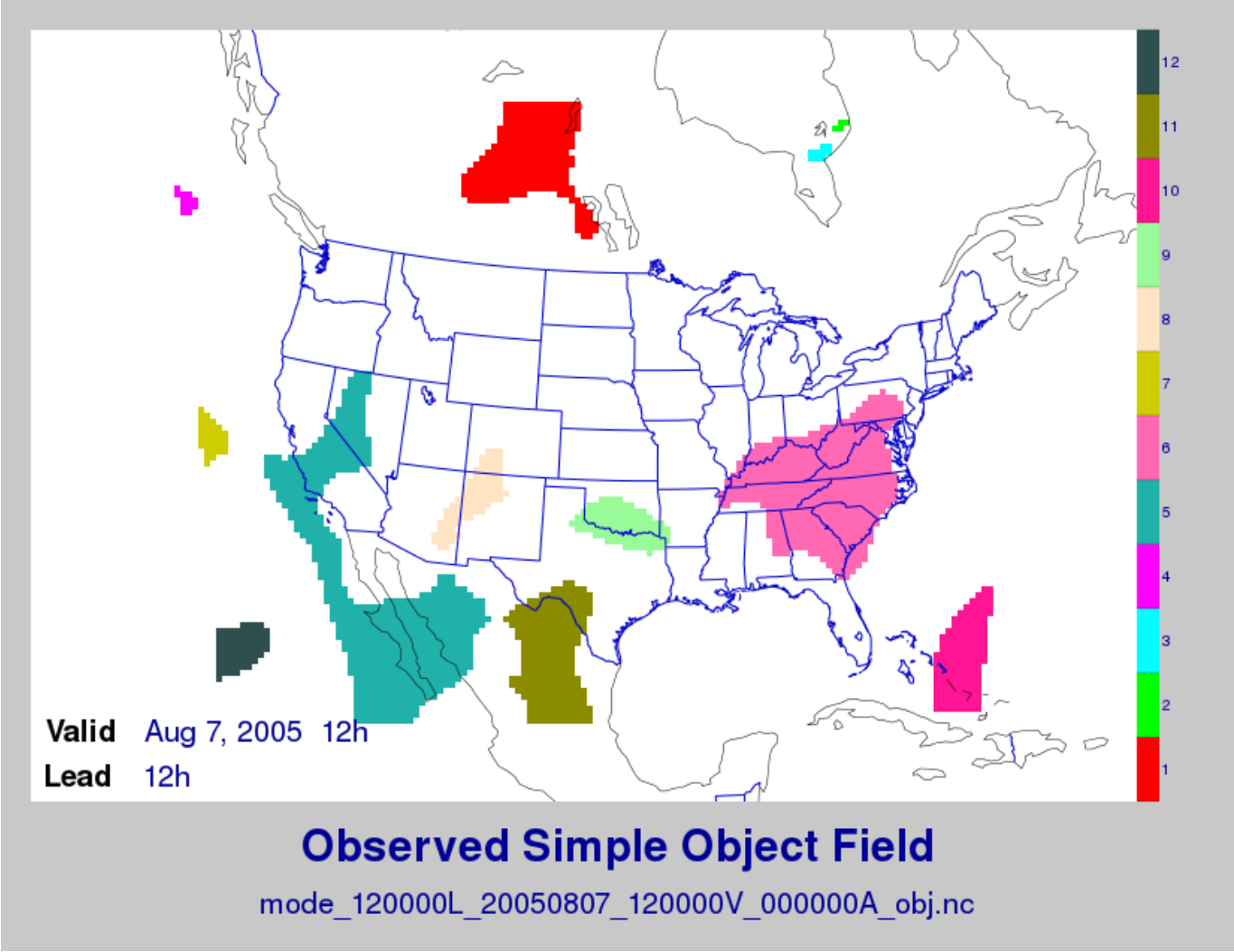

Figure 31.1: Simple observed precipitation objects

Once MET has been applied to forecast and observed fields (or observing locations), and the output has been sorted through the Analysis Tool, numerous graphical and summary analyses can be performed depending on a specific user's needs. Here we give some examples of graphics and summary scores that one might wish to compute with the given output of MET and MET-TC. Any computing language could be used for this stage; some scripts will be provided on the MET users web page as examples to assist users.

31.2 Examples of Plotting MET Output

31.2.1 Grid-Stat Tool Examples

The plots in Figure 31.2 show time series of frequency bias and Gilbert Skill Score, stratified according to time of day. This type of figure is particularly useful for diagnosing problems that are tied to the diurnal cycle. In this case, two of the models (green dash-dotted and black dotted lines) show an especially high Bias (near 3) during the afternoon (15-21 UTC; left panel), while the skill (GSS; right panel) appears to be best for the models represented by the solid black line and green dashed lines in the morning (09-15 UTC). Note that any judgment of skill based on GSS should be restricted to times when the Bias is close to one.

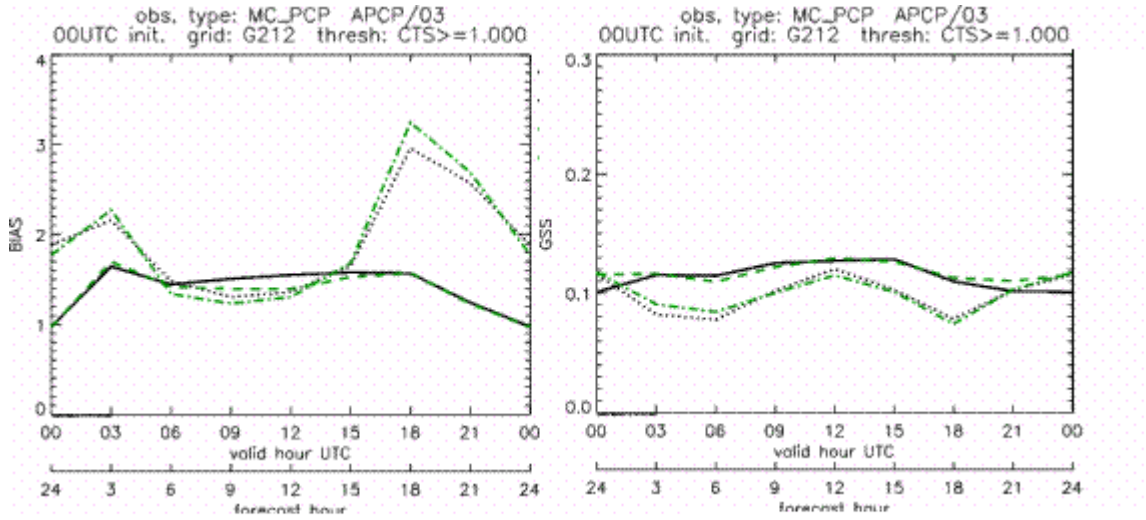

Figure 31.2: Time series of forecast area bias and Gilbert Skill Score for four model configurations (different lines) stratified by time-of-day.

31.2.2 MODE Tool Examples

When using the MODE tool, it is possible to think of matched objects as hits and unmatched objects as false alarms or misses depending on whether the unmatched object is from the forecast or observed field, respectively. Because the objects can have greatly differing sizes, it is useful to weight the statistics by the areas, which are given in the output as numbers of grid squares. When doing this, it is possible to have different matched observed object areas from matched forecast object areas so that the number of hits will be different depending on which is chosen to be a hit. When comparing multiple forecasts to the same observed field, it is perhaps wise to always use the observed field for the hits so that there is consistency for subsequent comparisons. Defining hits, misses and false alarms in this way allows one to compute many traditional verification scores without the problem of small-scale discrepancies; the matched objects are defined as being matched because they are "close" by the fuzzy logic criteria. Note that scores involving the number of correct negatives may be more difficult to interpret as it is not clear how to define a correct negative in this context. It is also important to evaluate the number and area attributes for these objects in order to provide a more complete picture of how the forecast is performing.

Figure 31.3 gives an example of two traditional verification scores (Bias and CSI) along with bar plots showing the total numbers of objects for the forecast and observed fields, as well as bar plots showing their

total areas. These data are from the same set of 13-km WRF model runs analyzed in Figure 31.3. The model runs were initialized at 0 UTC and cover the period 15 July to 15 August 2005. For the forecast evaluation, we compared 3-hour accumulated precipitation for lead times of 3-24 hours to Stage II radar-gauge precipitation. Note that for the 3-hr lead time, indicated as the 0300 UTC valid time in Figure 31.2, the Bias is significantly larger than the other lead times. This is evidenced by the fact that there are both a larger number of forecast objects, and a larger area of forecast objects for this lead time, and only for this lead time. Dashed lines show about 2 bootstrap standard deviations from the estimate.

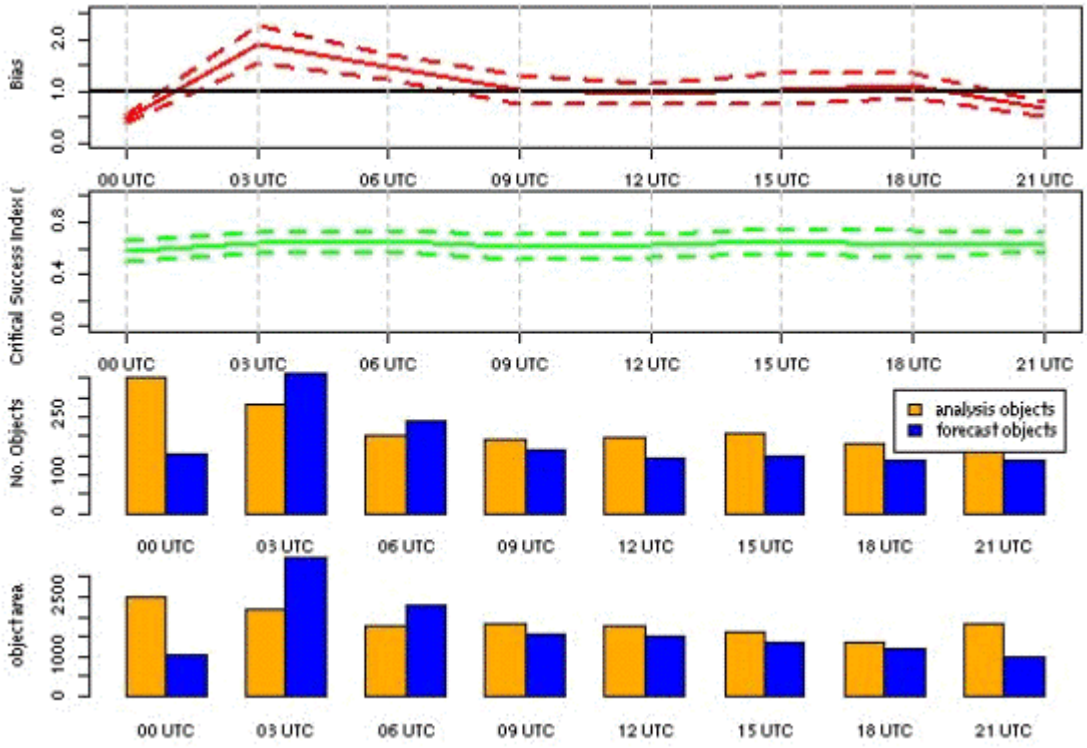

Figure 31.3: Traditional verification scores applied to output of the MODE tool, computed by defining matched observed objects to be hits, unmatched observed objects to be misses, and unmatched forecast objects to be false alarms; weighted by object area. Bar plots show numbers (penultimate row) and areas (bottom row) of observed and forecast objects, respectively.

In addition to the traditional scores, MODE output allows more information to be gleaned about forecast performance. It is even useful when computing the traditional scores to understand how much the forecasts are displaced in terms of both distance and direction. Figure 31.4, for example, shows circle histograms for matched objects. The petals show the percentage of times the forecast object centroids are at a given angle from the observed object centroids. In Figure 31.4 (top diagram) about 25% of the time the forecast object centroids are west of the observed object centroids, whereas in Figure 31.4 (bottom diagram) there is less bias in terms of the forecast objects' centroid locations compared to those of the observed objects, as evidenced by the petals' relatively similar lengths, and their relatively even dispersion around the circle. The colors on the petals represent the proportion of centroid distances within each colored bin along each direction. For example, Figure 31.4 (top row) shows that among the forecast object centroids that are located to the West of the observed object centroids, the greatest proportion of the separation distances (between the observed and forecast object centroids) is greater than 20 grid squares.

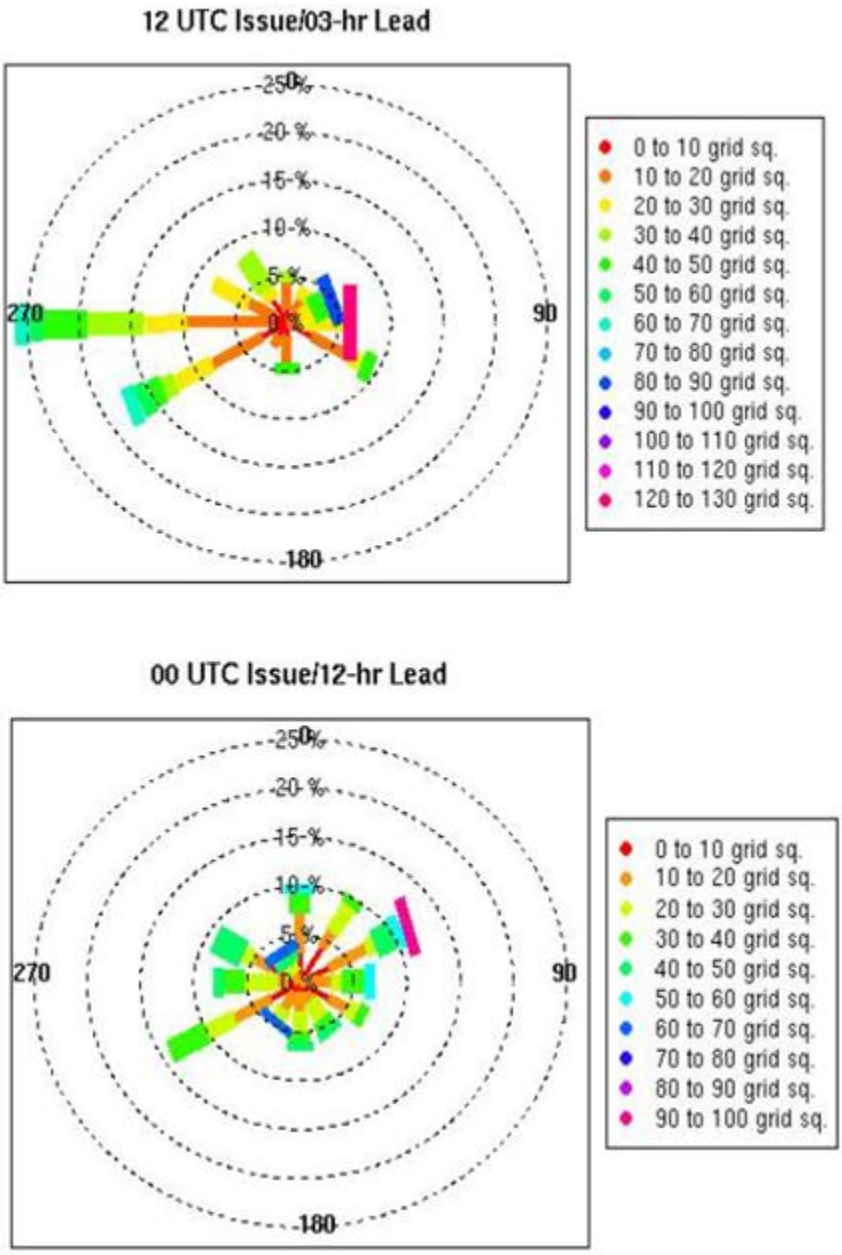

Figure 31.4: Circle histograms showing object centroid angles and distances (see text for explanation).

31.2.3 TC-Stat Tool Example

There is a basic R script located in the MET installation, *share/met/Rscripts/plot_tcmpr.R*. The usage statement with a short description of the options for *plot_tcmpr.R* can be obtained by typing: Rscript *plot_tcmpr.R* with no additional arguments. The only required argument is the **-lookin** source, which is the path to the TC-Pairs TCST output files. The R script reads directly from the TC-Pairs output, and calls TC-Stat directly for filter jobs specified in the *"-filter options"* argument.

In order to run this script, the MET_INSTALL_DIR environment variable must be set to the MET installation directory and the MET_BASE environment variable must be set to the *MET_INSTALL_DIR/share/met* directory. In addition, the Tc-Stat tool under *MET_INSTALL_DIR/bin* must be in your system path.

The supplied R script can generate a number of different plot types including boxplots, mean, median, rank, and relative performance. Pairwise differences can be plotted for the boxplots, mean, and median. Normal confidence intervals are applied to all figures unless the no_ci option is set to TRUE. Below are two example plots generated from the tools.

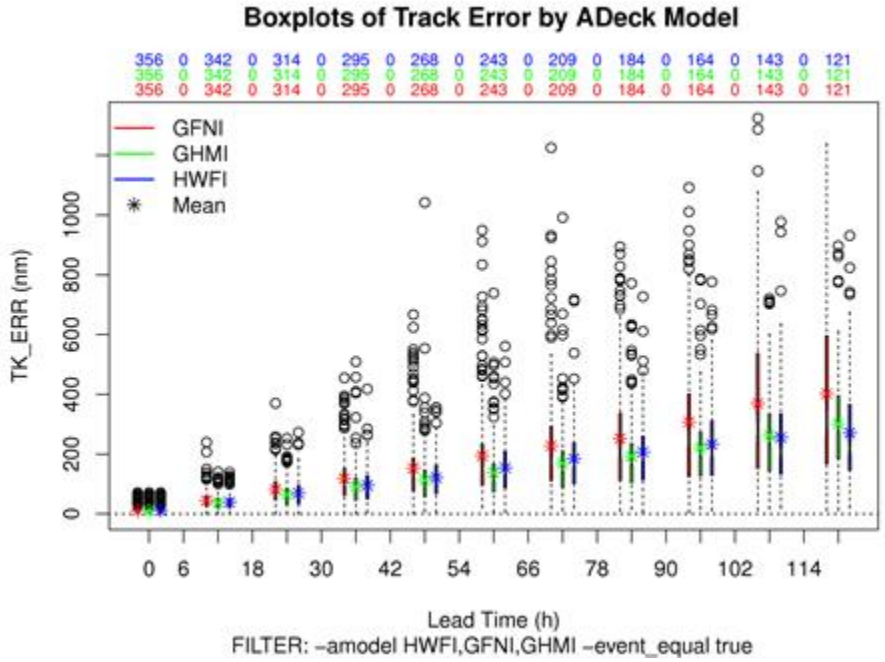

Figure 31.5: Example boxplot from plot_tcmpr.R. Track error distributions by lead time for three operational models GFNI, GHMI, HFWI.

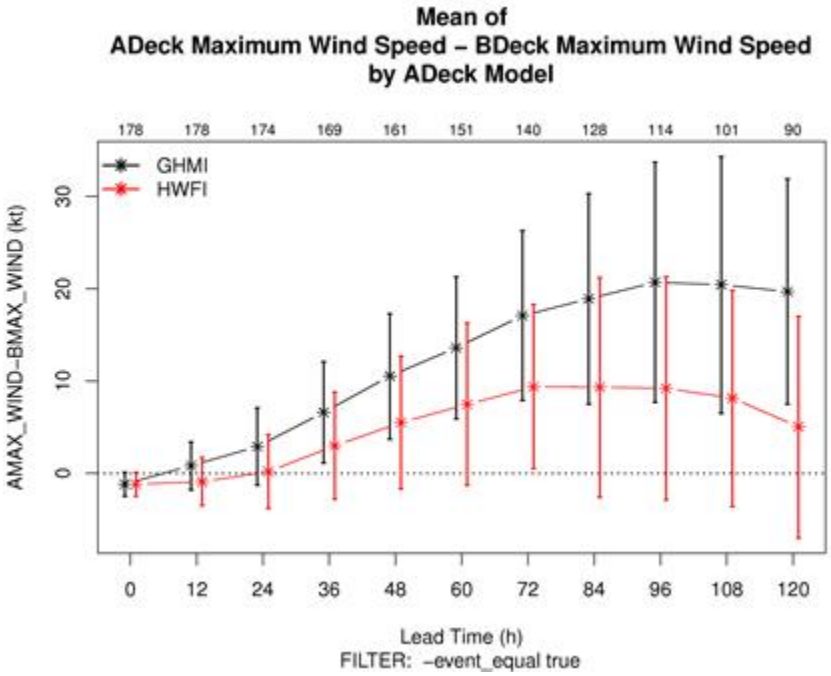

Figure 31.6: Example mean intensity error with confidence intervals at 95% from plot_tcmpr.R. Raw intensity error by lead time for a homogeneous comparison of two operational models GHMI, HWFI.

Chapter 32

References

Aberson, S.D., 1998: Five-day Tropical cyclone track forecasts in the North Atlantic Basin. *Weather and Forecasting*, 13, 1005-1015.

Adams, S. V., R. W. Ford, M. Hambley, J.M. Hobson, I. Kavčič, C. M. Maynard, T. Melvin, E. H. Müller, S. Mullerworth, A. R. Porter, M. Rezny, B. J. Shipway, and R. Wong, 2019: LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models. *Journal of Parallel and Distributed Computing*, **132**, 383-396, doi: https://doi.org/10.1016/j.jpdc.2019.02.007.

Ahijevych, D., E. Gilleland, B.G. Brown, and E.E. Ebert, 2009: Application of spatial verification methods to idealized and NWP-gridded precipitation forecasts. *Weather and Forecasting*, 24 (6), 1485 - 1497. doi: https://doi.org/10.1175/2009WAF2222298.1

Anderson JL., 1996: A method for producing and evaluating probabilistic forecasts from ensemble model integrations. *J. Clim.* 9: 1518-1530. doi: https://doi.org/10.1175/1520-0442(1996)009\T1\textless{}1518:AMFPAE\T1\textgreater{}2.0.CO;2

Barker, T. W., 1991: The relationship between spread and forecast error in extended-range forecasts. *Journal of Climate*, 4, 733-742.

Bradley, A.A., S.S. Schwartz, and T. Hashino, 2008: Sampling Uncertainty and Confidence Intervals for the Brier Score and Brier Skill Score. *Weather and Forecasting*, 23, 992-1006.

Brill, K. F., and F. Mesinger, 2009: Applying a general analytic method for assessing bias sensitivity to bias-adjusted threat and equitable threat scores. *Weather and Forecasting*, 24, 1748-1754.

Brown, B.G., R. Bullock, J. Halley Gotway, D. Ahijevych, C. Davis, E. Gilleland, and L. Holland, 2007: Application of the MODE object-based verification tool for the evaluation of model precipitation fields. *AMS 22nd Conference on Weather Analysis and Forecasting and 18th Conference on Numerical Weather Prediction*, 25-29 June, Park City, Utah, American Meteorological Society (Boston), Available at http://ams.confex.com/ams/pdfpapers/124856.pdf.

Bröcker J, Smith LA., 2007: Scoring probabilistic forecasts: The importance of being proper. *Weather Forecasting*, 22, 382-388. doi: https://doi.org/10.1175/WAF966.1

Buizza, R., 1997: Potential forecast skill of ensemble prediction and spread and skill distributions of the ECMWF ensemble prediction system. *Monthly Weather Review*, 125, 99-119.

Bullock, R., T. Fowler, and B. Brown, 2016: Method for Object-Based Diagnostic Evaluation. *NCAR Technical Note* NCAR/TN-532+STR, 66 pp.

Candille G, Côté C, Houtekamer PL, Pellerin G, 2007: Verification of an ensemble prediction system against observations. *Mon. Weather Rev.* 135: 2688-2699.
doi: https://doi.org/10.1175/MWR3414.1

Candille, G., and O. Talagrand, 2008: Impact of observational error on the validation of ensemble prediction systems. *Quarterly Journal of the Royal Meteorological Society* 134: 959-971.

Casati, B., G. Ross, and D. Stephenson, 2004: A new intensity-scale approach for the verification of spatial precipitation forecasts. *Meteorological Applications* 11, 141-154.

Davis, C.A., B.G. Brown, and R.G. Bullock, 2006a: Object-based verification of precipitation forecasts, Part I: Methodology and application to mesoscale rain areas. *Monthly Weather Review*, 134, 1772-1784.

Davis, C.A., B.G. Brown, and R.G. Bullock, 2006b: Object-based verification of precipitation forecasts, Part II: Application to convective rain systems. *Monthly Weather Review*, 134, 1785-1795.

Dawid, A.P., 1984: Statistical theory: The prequential approach. *Journal of*

the Royal Statistical Society A147, 278-292.

Ebert, E.E., 2008: Fuzzy verification of high-resolution gridded forecasts: a review and proposed framework. *Meteorological Applications*, 15, 51-64.

Ebert-Uphoff, I.„ 2024: An Investigation of Metrics to Evaluate the Sharpness in AI-Generated Meteorological Imagery. *Draft version - Jan 26, 2024*

Eckel, F. A., M.S. Allen, M. C. Sittel, 2012: Estimation of Ambiguity in Ensemble Forecasts. *Weather Forecasting*, 27, 50-69. doi: https://doi.org/10.1175/WAF-D-11-00015.1

Efron, B. 2007: Correlation and large-scale significance testing. *Journal of the American Statistical Association**, 102(477), 93-103.

Epstein, E. S., 1969: A scoring system for probability forecasts of ranked categories. *J. Appl. Meteor.*, 8, 985-987. doi: https://doi.org/10.1175/1520-0450(1969)008\T1\textless{}0985:ASSFPF\T1\textgreater{}2.0.CO;2

Ferro C. A. T., 2017: Measuring forecast performance in the presence of observation error. *Q. J. R. Meteorol. Soc.*, 143 (708), 2665-2676. doi: https://doi.org/10.1002/qj.3115

Gilleland, E., 2010: Confidence intervals for forecast verification. *NCAR Technical Note* NCAR/TN-479+STR, 71pp.

Gilleland, E., 2017: A new characterization in the spatial verification framework for false alarms, misses, and overall patterns. *Weather and Forecasting*, 32 (1), 187 - 198. doi: https://doi.org/10.1175/WAF-D-16-0134.1

Gilleland, E., 2020: Bootstrap methods for statistical inference. Part I: Comparative forecast verification for continuous variables. *Journal of Atmospheric and Oceanic Technology*, 37 (11), 2117 - 2134. doi: https://doi.org/10.1175/JTECH-D-20-0069.1

Gilleland, E., 2020: Bootstrap methods for statistical inference. Part II: Extreme-value analysis. *Journal of Atmospheric and Oceanic Technology*, 37 (11), 2135 - 2144. doi: https://doi.org/10.1175/JTECH-D-20-0070.1

Gilleland, E., 2021: Novel measures for summarizing high-resolution forecast performance. *Advances in Statistical Climatology, Meteorology and Oceanography*, 7 (1), 13 - 34. doi: https://doi.org/10.5194/ascmo-7-13-2021

Gneiting, T., A. Westveld, A. Raferty, and T. Goldman, 2004: *Calibrated Probabilistic Forecasting Using Ensemble Model Output Statistics and Minimum CRPS Estimation*. Technical Report no. 449, Department of Statistics, University of Washington. Available at

http://www.stat.washington.edu/www/research/reports/

Haiden, T., M.J. Rodwell, D.S. Richardson, A. Okagaki, T. Robinson, T. Hewson, 2012: Intercomparison of Global Model Precipitation Forecast Skill in 2010/11 Using the SEEPS Score. *Monthly Weather Review*, 140, 2720-2733. https://doi.org/10.1175/MWR-D-11-00301.1

Hamill, T. M., 2001: Interpretation of rank histograms for verifying ensemble forecasts. *Monthly Weather Review*, 129, 550-560.

Hersbach, H., 2000: Decomposition of the Continuous Ranked Probability Score for Ensemble Prediction Systems. *Weather and Forecasting*, 15, 559-570.

Jolliffe, I.T., and D.B. Stephenson, 2012: *Forecast verification. A practitioner's guide in atmospheric science.* Wiley and Sons Ltd, 240 pp.

Knaff, J.A., M. DeMaria, C.R. Sampson, and J.M. Gross, 2003: Statistical, Five-Day Tropical Cyclone Intensity Forecasts Derived from Climatology and Persistence. *Weather and Forecasting*, Vol. 18 Issue 2, p. 80-92.

Mason, S. J., 2004: On Using "Climatology" as a Reference Strategy in the Brier and Ranked Probability Skill Scores. *Monthly Weather Review*, 132, 1891-1895.

Mason, S. J., 2008: Understanding forecast verification statistics. *Meteor. Appl.*, 15, 31-40. doi: https://doi.org/10.1002/met.51

Mittermaier, M., 2014: A strategy for verifying near-convection-resolving model forecasts at observing sites. *Weather Forecasting*, 29, 185-204.

Mood, A. M., F. A. Graybill and D. C. Boes, 1974: *Introduction to the Theory of Statistics*, McGraw-Hill, 299-338.

Murphy, A.H., 1969: On the ranked probability score. *Journal of Applied Meteorology and Climatology*, 8 (6), 988 - 989, doi: https://doi.org/10.1175/1520-0450(1969)008\T1\textless{}0988:OTPS\T1\textgreater{}2.0.CO;2

Murphy, A.H., and R.L. Winkler, 1987: A general framework for forecast verification. *Monthly Weather Review*, 115, 1330-1338.

North, R.C., M.P. Mittermaier, S.F. Milton, 2022. *Using SEEPS with a* TRMM-derived Climatology to Assess Global NWP Precipitation Forecast Skill. *Monthly Weather Review*, 150, 135-155. https://doi.org/10.1175/MWR-D-20-0347.1

Ou, M. H., Charles, M., & Collins, D. C. 2016: Sensitivity of calibrated week-2

probabilistic forecast skill to reforecast sampling of the NCEP global ensemble forecast system. *Weather and Forecasting*, 31(4), 1093-1107.

Roberts, N.M., and H.W. Lean, 2008: Scale-selective verification of rainfall accumulations from high-resolution forecasts of convective events. *Monthly Weather Review*, 136, 78-97.

Rodwell, M.J., D.S. Richardson, T.D. Hewson and T. Haiden, 2010: A new equitable score suitable for verifying precipitation in numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, 136: 1344-1463. https://doi.org/10.1002/qj.656

Rodwell, M.J., T. Haiden, D.S. Richardson, 2011: Developments in Precipitation Verification. *ECMWF Newsletter* Number 128. https://www.ecmwf.int/node/14595

Röpnack A, Hense A, Gebhardt C, Majewski D., 2013: Bayesian model verification of NWP ensemble forecasts. *Mon. Weather Rev.* 141: 375–387. doi: https://doi.org/10.1175/MWR-D-11-00350.1

Saetra Ø., H. Hersbach, J-R Bidlot, D. Richardson, 2004: Effects of observation errors on the statistics for ensemble spread and reliability. *Monthly Weather Review*, 132: 1487-1501.

Santos C. and A. Ghelli, 2012: Observational probability method to assess ensemble precipitation forecasts. *Quarterly Journal of the Royal Meteorological Society* 138: 209-221.

Schwartz C. and Sobash R., 2017: Generating Probabilistic Forecasts from Convection-Allowing Ensembles Using Neighborhood Approaches: A Review and Recommendations. *Monthly Weather Review*, 145, 3397-3418.

Skamarock, W. C., J. B. Klemp, M. G. Duda, L. D. Fowler, S. Park, and T. Ringler, 2012: A Multiscale Nonhydrostatic Atmospheric Model Using Centroidal Voronoi Tesselations and C-Grid Staggering. *Mon. Wea. Rev.*, **140**, 3090-3105, doi: https://doi.org/10.1175/MWR-D-11-00215.1.

Stephenson, D.B., 2000: Use of the "Odds Ratio" for diagnosing forecast skill. *Weather and Forecasting*, 15, 221-232.

Stephenson, D.B., B. Casati, C.A.T. Ferro, and C.A. Wilson, 2008: The extreme dependency score: A non-vanishing measure for forecasts of rare events. *Meteorological Applications* 15, 41-50.

Tödter, J. and B. Ahrens, 2012: Generalization of the Ignorance Score: Continuous ranked version and its decomposition. *Monthly Weather Review*, 140 (6), 2005 - 2017.
doi: https://doi.org/10.1175/MWR-D-11-00266.1

Weniger, M., F. Kapp, and P. Friederichs, 2016: Spatial Verification Using Wavelet Transforms: A Review. *Quarterly Journal of the Royal Meteorological Society*, 143, 120-136.

Wilks, D.S. 2010: Sampling distributions of the Brier score and Brier skill score under serial dependence. *Quarterly Journal of the Royal Meteorological Society*, 136, 2109-2118. doi: https://doi.org/10.1002/qj.709

Wilks, D., 2011: *Statistical methods in the atmospheric sciences.* Elsevier, San Diego.

Chapter 33

Appendix A FAQs & How do I . . . ?

33.1 Frequently Asked Questions

33.1.1 File-IO

33.1.1.1 Q. How do I improve the speed of MET tools using Gen-Vx-Mask?

Answer

The main reason to run gen_vx_mask is to make the MET statistics tools (e.g. point_stat, grid_stat, or ensemble_stat) run faster. The verification masking regions in those tools can be specified as Lat/Lon polyline files or the NetCDF output of gen_vx_mask. However, determining which grid points are inside/outside a polyline region can be slow if the polyline contains many points or the grid is dense. Running gen_vx_mask once to create a binary mask is much more efficient than recomputing the mask when each MET statistics tool is run. If the polyline only contains a small number of points or the grid is sparse running gen_vx_mask first would only save a second or two.

33.1.1.2 Q. How do I use map_data?

Answer

The MET repository includes several map data files. Users can modify which map datasets are included in the plots created by modifying the configuration files for those tools. The default map datasets are defined by the map_data dictionary in the ConfigMapData file.

```
map_data = {

   line_color = [ 25, 25, 25 ]; // rgb triple values, 0-255
   line_width = 0.5;
   line_dash  = "";
```

(continues on next page)

(continued from previous page)

```
   source = [
      { file_name = "MET_BASE/map/country_data"; },
      { file_name = "MET_BASE/map/usa_state_data"; },
      { file_name = "MET_BASE/map/major_lakes_data"; }
   ];
}
```

Users can modify the ConfigMapData contents prior to running 'make install'. This will change the default map data for all of the MET tools which plots. Alternatively, users can copy/paste/modify the map_data dictionary into the configuration file for a MET tool. For example, you could add map_data to the end of the MODE configuration file to customize plots created by MODE.

Here is an example of running plot_data_plane and specifying the map_data in the configuration string on the command line:

```
plot_data_plane
sample.grib china_tmp_2m_admin.ps \
'name="TMP"; level="Z2"; \
map_data = { source = [ { file_name = \
"${MET_BASE}/map/admin_by_country/admin_China_data"; } \
]; }'
```

33.1.1.3 Q. How can I understand the number of matched pairs?

Answer

Statistics are computed on matched forecast/observation pairs data. For example, if the dimension of the grid is 37x37 up to 1369 matched pairs are possible. However, if the forecast or observation contains bad data at a point, that matched pair would not be included in the calculations. There are a number of reasons that observations could be rejected - mismatches in station id, variable names, valid times, bad values, data off the grid, etc. For example, if the forecast field contains missing data around the edge of the domain, then that is a reason there may be 992 matched pairs instead of 1369. Users can use the ncview tool to look at an example netCDF file or run their files through plot_data_plane to help identify any potential issues.

One common support question is "Why am I getting 0 matched pairs from Point-Stat?". As mentioned above, there are many reasons why point observations can be excluded from your analysis. If running point_stat with at least verbosity level 2 (-v 2, the default value), zero matched pairs will result in the following type of log messages to be printed:

```
DEBUG 2: Processing TMP/Z2 versus TMP/Z2, for observation type ADPSFC, over region␣
↪FULL, for interpolation method UW_MEAN(1), using 0 pairs.
DEBUG 2: Number of matched pairs   = 0
DEBUG 2: Observations processed    = 1166
DEBUG 2: Rejected: station id      = 0
```

(continues on next page)

(continued from previous page)

```
DEBUG 2: Rejected: obs var name    = 1166
DEBUG 2: Rejected: valid time      = 0
DEBUG 2: Rejected: bad obs value   = 0
DEBUG 2: Rejected: off the grid    = 0
DEBUG 2: Rejected: topography      = 0
DEBUG 2: Rejected: level mismatch  = 0
DEBUG 2: Rejected: quality marker  = 0
DEBUG 2: Rejected: message type    = 0
DEBUG 2: Rejected: masking region  = 0
DEBUG 2: Rejected: bad fcst value  = 0
DEBUG 2: Rejected: bad climo mean  = 0
DEBUG 2: Rejected: bad climo stdev = 0
DEBUG 2: Rejected: mpr filter      = 0
DEBUG 2: Rejected: duplicates      = 0
```

This list of the rejection reason counts above matches the order in which the filtering logic is applied in the code. In this example, none of the point observations match the variable name requested in the configuration file. So all of the 1166 observations are rejected for the same reason.

In addition, running point_stat with at least verbosity level 9 (-v 9) will result in a log message being printed to explain why each observation is skipped or retained for each verification task. This level of detail is intended only for debugging purposes.

33.1.1.4 Q. What types of NetCDF files can MET read?

Answer

There are three flavors of NetCDF that MET can read directly.

1. Gridded NetCDF output from one of the MET tools
2. Output from the WRF model that has been post-processed using the wrf_interp utility
3. NetCDF data following the climate-forecast (CF) convention

Lastly, users can write python scripts to pass data that's gridded to the MET tools in memory. If the data doesn't fall into one of those categories, then it's not a gridded dataset that MET can handle directly. Satellite data, in general, will not be gridded. Typically it contains a dense mesh of data at lat/lon points, but typically those lat/lon points are not evenly spaced onto a regular grid.

While MET's point2grid tool does support some satellite data inputs, it is limited. Using python embedding is another option for handling new datasets not supported natively by MET.

33.1.1.5 Q. How do I choose a time slice in a NetCDF file?

Answer

When processing NetCDF files, the level information needs to be specified to tell MET which 2D slice of data to use. The index is selected from a value when it starts with "@" for vertical level (pressure or height) and time. The actual time, @YYYYMMDD_HHMM, is allowed instead of selecting the time index.

Let's use plot_data_plane as an example:

```
plot_data_plane \
MERGE_20161201_20170228.nc \
obs.ps \
'name="APCP"; level="(5,*,*)";'

plot_data_plane \
gtg_obs_forecast.20130730.i00.f00.nc \
altitude_20000.ps \
'name = "edr"; level = "(@20130730_0000,@20000,*,*)";'
```

Assuming that the first array is the time, this will select the 6-th time slice of the APCP data and plot it since these indices are 0-based.

33.1.1.6 Q. How do I use the UNIX time conversion?

Answer

Regarding the timing information in the NetCDF variable attributes:

```
APCP_24:init_time_ut = 1306886400 ;
```

"ut" stands for UNIX time, which is the number of seconds since Jan 1, 1970. It is a convenient way of storing timing information since it is easy to add/subtract. The UNIX date command can be used to convert back/forth between unix time and time strings:

To convert unix time to ymd_hms date:

```
date -ud '1970-01-01 UTC '1306886400' seconds' +%Y%m%d_%H%M%S 20110601_000000
```

To convert ymd_hms to unix date:

```
date -ud ''2011-06-01' UTC '00:00:00'' +%s 1306886400
```

Regarding TRMM data, it may be easier to work with the binary data and use the trmm2nc.R script described on this page under observation datasets.

Follow the TRMM binary links to either the 3 or 24-hour accumulations, save the files, and run them through that script. That is faster and easier than trying to get an ASCII dump. That

Rscript can also subset the TRMM data if needed. Look for the section of it titled "Output domain specification" and define the lat/lon's that needs to be included in the output.

33.1.1.7 Q. Does MET use a fixed-width output format for its ASCII output files?

Answer

MET does not use the Fortran-like fixed width format in its ASCII output file. Instead, the column widths are adjusted for each run to insert at least one space between adjacent columns. The header columns of the MET output contain user-defined strings which may be of arbitrary length. For example, columns such as MODEL, OBTYPE, and DESC may be set by the user to any string value. Additionally, the amount of precision written is also configurable. The "output_precision" config file entry can be changed from its default value of 5 decimal places to up to 12 decimal places, which would also impact the column widths of the output.

Due to these issues, it is not possible to select a reasonable fixed width for each column ahead of time. The AsciiTable class in MET does a lot of work to line up the output columns, to make sure there is at least one space between them.

If a fixed-width format is needed, the easiest option would be writing a script to post-process the MET output into the fixed-width format that is needed or that the code expects.

33.1.1.8 Q. Do the ASCII output files created by MET use scientific notation?

Answer

By default, the ASCII output files created by MET make use of scientific notation when appropriate. The formatting of the numbers that the AsciiTable class writes is handled by a call to printf. The "%g" formatting option can result in scientific notation: http://www.cplusplus.com/reference/cstdio/printf/

It has been recommended that a configuration option be added to MET to disable the use of scientific notation. That enhancement is planned for a future release.

33.1.2 Gen-Vx-Mask

33.1.2.1 Q. I have a list of stations to use for verification. I also have a poly region defined. If I specify both of these should the result be a union of them?

Answer

These settings are defined in the "mask" section of the Point-Stat configuration file. You can define masking regions in one of 3 ways, as a "grid", a "poly" line file, or a "sid" list of station ID's.

If you specify one entry for "poly" and one entry for "sid", you should see output for those two different masks. Note that each of these settings is an array of values, as indicated by the

square brackets "[]" in the default config file. If you specify 5 grids, 3 poly's, and 2 SID lists, you'd get output for those 10 separate masking regions. Point-Stat does not compute unions or intersections of masking regions. Instead, they are each processed separately.

Is it true that you really want to use a polyline to define an area and then use a SID list to capture additional points outside of that polyline?

If so, your options are:

1. Define one single SID list which include all the points currently inside the polyline as well as the extra ones outside.
2. Continue verifying using one polyline and one SID list and write partial sums and contingency table counts.

Then aggregate the results together by running a Stat-Analysis job.

33.1.2.2 Q. How do I define a masking region with a GFS file?

Answer

Grab a sample GFS file:

```
wget
http://www.ftp.ncep.noaa.gov/data/nccf/com/gfs/prod/gfs/2016102512/gfs.t12z.pgrb2.
↪0p50.f000
```

Use the MET regrid_data_plane tool to put some data on a lat/lon grid over Europe:

```
regrid_data_plane gfs.t12z.pgrb2.0p50.f000 \
'latlon 100 100 25 0 0.5 0.5' gfs_euro.nc -field 'name="TMP"; level="Z2";'
```

Run the MET gen_vx_mask tool to apply your polyline to the European domain:

```
gen_vx_mask gfs_euro.nc POLAND.poly POLAND_mask.nc
```

Run the MET plot_data_plane tool to display the resulting mask field:

```
plot_data_plane POLAND_mask.nc POLAND_mask.ps 'name="POLAND"; level="(*,*)";'
```

In this example, the mask is in roughly the right spot, but there are obvious problems with the latitude and longitude values used to define that mask for Poland.

33.1.3 Grid-Stat

33.1.3.1 Q. How do I define a complex masking region?

Answer

A user can define intersections and unions of multiple fields to define masks. Prior to running Grid-Stat, the user can run the Gen-VX-Mask tool one or more times to define a more complex masking area by thresholding multiple fields.

For example, using a forecast GRIB file (fcst.grb) which contains 2 records, one for 2-m temperature and a second for 6-hr accumulated precip. The only grid points that are desired are grid points below freezing with non-zero precip. The user should run Gen-Vx-Mask twice - once to define the temperature mask and a second time to intersect that with the precip mask:

```
gen_vx_mask fcst.grb fcst.grb tmp_mask.nc \
-type data \
-mask_field 'name="TMP"; level="Z2"' -thresh le273
gen_vx_mask tmp_mask.nc fcst.grb tmp_and_precip_mask.nc \
-type data \
-input_field 'name="TMP_Z2"; level="(*,*)";' \
-mask_field 'name="APCP"; level="A6";' -thresh gt0 \
-intersection -name "FREEZING_PRECIP"
```

The first one is pretty straight-forward.

1. The input field (fcst.grb) defines the domain for the mask.
2. Since we're doing data masking and the data we want lives in fcst.grb, we pass it in again as the mask_file.
3. Lastly "-mask_field" specifies the data we want from the mask file and "-thresh" specifies the event threshold.

The second call is a bit tricky.

1. Do data masking (-type data)
2. Read the NetCDF variable named "TMP_Z2" from the input file (tmp_mask.nc)
3. Define the mask by reading 6-hour precip from the mask file (fcst.grb) and looking for values > 0 (-mask_field)
4. Apply intersection logic when combining the "input" value with the "mask" value (-intersection).
5. Name the output NetCDF variable as "FREEZING_PRECIP" (-name). This is totally optional, but convenient.

A user can write a script with multiple calls to Gen-Vx-Mask to apply complex masking logic and then pass the output mask file to Grid-Stat in its configuration file.

33.1.3.2 Q. How do I use neighborhood methods to compute fraction skill score?

Answer

A common application of fraction skill score (FSS) is comparing forecast and observed thunderstorms. When computing FSS, first threshold the fields to define events and non-events. Then look at successively larger and larger areas around each grid point to see how the forecast event frequency compares to the observed event frequency.

Applying this method to rainfall (and monsoons) is also reasonable. Keep in mind that Grid-Stat is the tool that computes FSS. Grid-Stat will need to be run once for each evaluation time. As an example, evaluating once per day, run Grid-Stat 122 times for the 122 days of a monsoon season. This will result in 122 FSS values. These can be viewed as a time series, or the Stat-Analysis tool could be used to aggregate them together into a single FSS value, like this:

```
stat_analysis -job aggregate -line_type NBRCNT \
-lookin out/grid_stat
```

Be sure to pick thresholds (e.g. for the thunderstorms and monsoons) that capture the "events" that are of interest in studying.

Also be aware that MET uses the "vld_thresh" setting in the configuration file to decide how to handle data along the edge of the domain. Let us say it is computing a fractional coverage field using a 5x5 neighborhood and it is at the edge of the domain. 15 points contain valid data and 10 points are outside the domain. Grid-Stat computes the valid data ratio as 15/25 = 0.6. Then it applies the valid data threshold. Suppose vld_thresh = 0.5. Since 0.6 > 0.5 MET will compute a fractional coverage value for that point using the 15 valid data points. Next suppose vld_thresh = 1.0. Since 0.6 is less than 1.0, MET will just skip that point by setting it to bad data.

Setting vld_thresh = 1.0 will ensure that FSS will only be computed at points where all NxN values contain valid data. Setting it to 0.5 only requires half of them.

33.1.3.3 Q. Is an example of verifying forecast probabilities?

Answer

There is an example of verifying probabilities in the test scripts included with the MET release. Take a look in:

```
MET/scripts/config/GridStatConfig_POP_12
```

The config file should look something like this:

```
fcst = {
        wind_thresh = [ NA ];
        field = [
         {
          name = "LCDC";
```

(continues on next page)

(continued from previous page)

```
          level = [ "L0" ];
          prob = TRUE;
          cat_thresh = [ >=0.0, >=0.1, >=0.2, >=0.3, >=0.4, >=0.5, >=0.6, >=0.7, >
↪=0.8, >=0.9];
         }
                ];
       };

obs = {
       wind_thresh = [ NA ];
       field = [
        {
         name = "WIND";
         level = [ "Z2" ];
         cat_thresh = [ >=34 ];
         }
               ];
       };
```

The PROB flag is set to TRUE to tell grid_stat to process this as probability data. The cat_thresh is set to partition the probability values between 0 and 1. Note that if the probability data contains values from 0 to 100, MET automatically divides by 100 to rescale to the 0 to 1 range.

33.1.3.4 Q. What is an example of using Grid-Stat with regridding and masking turned on?

Answer

Run Grid-Stat using the following commands and the attached config file

```
mkdir out
grid_stat \
gfs_4_20160220_0000_012.grb2 \
ST4.2016022012.06h \
GridStatConfig \
-outdir out
```

Note the following two sections of the Grid-Stat config file:

```
regrid = {
          to_grid = OBS;
          vld_thresh = 0.5;
          method = BUDGET;
          width = 2;
         }
```

This tells Grid-Stat to do verification on the "observation" grid. Grid-Stat reads the GFS and

Stage4 data and then automatically regrids the GFS data to the Stage4 domain using budget interpolation. Use FCST to verify the forecast domain. And use either a named grid or a grid specification string to regrid both the forecast and observation to a common grid. For example, to_grid = "G212"; will regrid both to NCEP Grid 212 before comparing them.

```
mask = { grid = [ "FULL" ];
poly = [ "MET_BASE/poly/CONUS.poly" ]; }
```

This will compute statistics over the FULL model domain as well as the CONUS masking area.

To demonstrate that Grid-Stat worked as expected, run the following commands to plot its NetCDF matched pairs output file:

```
plot_data_plane \
out/grid_stat_120000L_20160220_120000V_pairs.nc \
out/DIFF_APCP_06_A06_APCP_06_A06_CONUS.ps \
'name="DIFF_APCP_06_A06_APCP_06_A06_CONUS"; level="(*,*)";'
```

Examine the resulting plot of that difference field.

Lastly, there is another option for defining that masking region. Rather than passing the ascii CONUS.poly file to grid_stat, run the gen_vx_mask tool and pass the NetCDF output of that tool to grid_stat. The advantage to gen_vx_mask is that it will make grid_stat run a bit faster. It can be used to construct much more complex masking areas.

33.1.3.5 Q. How do I use one mask for the forecast field and a different mask for the observation field?

Answer

You can't define different masks for the forecast and observation fields in MET tools. MET only lets you define a single mask (a masking grid or polyline) and then you choose whether you want to apply it to the FCST, OBS, or BOTH of them.

Nonetheless, there is a way you can accomplish this logic using the gen_vx_mask tool. You run it once to pre-process the forecast field and a second time to pre-process the observation field. And then pass those output files to your desired MET tool.

Below is an example using sample data that is included with the MET release tarball. To illustrate, this command will read 3-hour precip and 2-meter temperature, and resets the precip at any grid point where the temperature is less than 290 K to a value of 0:

```
gen_vx_mask \
data/sample_fcst/2005080700/wrfprs_ruc13_12.tm00_G212 \
data/sample_fcst/2005080700/wrfprs_ruc13_12.tm00_G212 \
APCP_03_where_2m_TMPge290.nc \
-type data \
-input_field 'name="APCP"; level="A3";' \
```

(continues on next page)

(continued from previous page)

```
-mask_field 'name="TMP"; level="Z2";' \
-thresh 'lt290&&ne-9999' -v 4 -value 0
```

So this is a bit confusing. Here's what is happening:

- The first argument is the input file which defines the grid.
- The second argument is used to define the masking region and since I'm reading data from the same input file, I've listed that file twice.
- The third argument is the output file name.
- The type of masking is "data" masking where we read a 2D field of data and apply a threshold.
- By default, gen_vx_mask initializes each grid point to a value of 0. Specifying "-input_field" tells it to initialize each grid point to the value of that field (in my example 3-hour precip).
- The "-mask_field" option defines the data field that should be thresholded.
- The "-thresh" option defines the threshold to be applied.
- The "-value" option tells it what "mask" value to write to the output, and I've chosen 0.

The example threshold is less than 290 and not -9999 (which is MET's internal missing data value). So any grid point where the 2 meter temperature is less than 290 K and is not bad data will be replaced by a value of 0.

To more easily demonstrate this, I changed to using "-value 10" and ran the output through plot_data_plane:

```
plot_data_plane \
     APCP_03_where_2m_TMPge290.nc \
APCP_03_where_2m_TMPge290.ps \
'name="data_mask"; level="(*,*)";'
```

In the resulting plot, anywhere you see the pink value of 10, that's where gen_vx_mask has masked out the grid point.

33.1.4 Pcp-Combine

33.1.4.1 Q. How do I add and subtract with Pcp-Combine?

Answer

An example of running the MET pcp_combine tool to put NAM 3-hourly precipitation accumulations data into user-desired 3 hour intervals is provided below.

If the user wanted a 0-3 hour accumulation, this is already available in the 03 UTC file. Run this file through pcp_combine as a pass-through to put it into NetCDF format:

```
pcp_combine -add 03_file.grb 03 APCP_00_03.nc
```

If the user wanted the 3-6 hour accumulation, they would subtract 0-6 and 0-3 accumulations:

```
pcp_combine -subtract 06_file.grb 06 03_file.grb 03 APCP_03_06.nc
```

Similarly, if they wanted the 6-9 hour accumulation, they would subtract 0-9 and 0-6 accumulations:

```
pcp_combine -subtract 09_file.grb 09 06_file.grb 06 APCP_06_09.nc
```

And so on.

Run the 0-3 and 12-15 through pcp_combine even though they already have the 3-hour accumulation. That way, all of the NAM files will be in the same file format, and can use the same configuration file settings for the other MET tools (grid_stat, mode, etc.). If the NAM files are a mix of GRIB and NetCDF, the logic would need to be a bit more complicated.

33.1.4.2 Q. How do I combine 12-hour accumulated precipitation from two different initialization times?

Answer

The "-sum" command assumes the same initialization time. Use the "-add" option instead.

```
pcp_combine -add \
WRFPRS_1997-06-03_APCP_A12.nc 'name="APCP_12"; level="(*,*)";' \
WRFPRS_d01_1997-06-04_00_APCP_A12.grb 12 \
Sum.nc
```

For the first file, list the file name followed by a config string describing the field to use from the NetCDF file. For the second file, list the file name followed by the accumulation interval to use (12 for 12 hours). The output file, Sum.nc, will contain the combine 12-hour accumulated precipitation.

Here is a small excerpt from the pcp_combine usage statement:

Note: For "-add" and "-subtract", the accumulation intervals may be substituted with config file strings. For that first file, we replaced the accumulation interval with a config file string.

Here are 3 commands you could use to plot these data files:

```
plot_data_plane WRFPRS_1997-06-03_APCP_A12.nc \
WRFPRS_1997-06-03_APCP_A12.ps 'name="APCP_12"; level="(*,*)";'
```

```
plot_data_plane WRFPRS_d01_1997-06-04_00_APCP_A12.grb \
WRFPRS_d01_1997-06-04_00_APCP_A12.ps 'name="APCP" level="A12";'
```

```
plot_data_plane sum.nc sum.ps 'name="APCP_24"; level="(*,*)";'
```

33.1.4.3 Q. How do I correct a precipitation time range?

Answer

Typically, accumulated precipitation is stored in GRIB files using an accumulation interval with a "time range" indicator value of 4. Here is a description of the different time range indicator values and meanings: http://www.nco.ncep.noaa.gov/pmb/docs/on388/table5.html

For example, take a look at the APCP in the GRIB files included in the MET tar ball:

```
wgrib MET/data/sample_fcst/2005080700/wrfprs_ruc13_12.tm00_G212 | grep APCP
1:0:d=05080700:APCP:kpds5=61:kpds6=1:kpds7=0:TR=4:P1=0: \
P2=12:TimeU=1:sfc:0- 12hr acc:NAve=0
2:31408:d=05080700:APCP:kpds5=61:kpds6=1:kpds7=0:TR=4: \
P1=9:P2=12:TimeU=1:sfc:9- 12hr acc:NAve=0
```

The "TR=4" indicates that these records contain an accumulation between times P1 and P2. In the first record, the precip is accumulated between 0 and 12 hours. In the second record, the precip is accumulated between 9 and 12 hours.

However, the GRIB data uses a time range indicator of 5, not 4.

```
wgrib rmf_gra_2016040600.24 | grep APCP
291:28360360:d=16040600:APCP:kpds5=61:kpds6=1:kpds7=0: \
TR=5:P1=0:P2=24:TimeU=1:sfc:0-24hr diff:NAve=0
```

pcp_combine is looking in "rmf_gra_2016040600.24" for a 24 hour *accumulation*, but since the time range indicator is no 4, it doesn't find a match.

If possible switch the time range indicator to 4 on the GRIB files. If this is not possible, there is another workaround. Instead of telling pcp_combine to look for a particular accumulation interval, give it a more complete description of the chosen field to use from each file. Here is an example:

```
pcp_combine -add rmf_gra_2016040600.24 'name="APCP"; level="L0-24";' \
rmf_gra_2016040600_APCP_00_24.nc
```

The resulting file should have the accumulation listed at 24h rather than 0-24.

33.1.4.4 Q. How do I use Pcp-Combine as a pass-through to simply reformat from GRIB to NetCDF or to change output variable name?

Answer

The pcp_combine tool is typically used to modify the accumulation interval of precipitation amounts in model and/or analysis datasets. For example, when verifying model output in GRIB format containing runtime accumulations of precipitation, run the pcp_combine -subtract option every 6 hours to create 6-hourly precipitation amounts. In this example, it is not really necessary to run pcp_combine on the 6-hour GRIB forecast file since the model output already contains the 0 to 6 hour accumulation. However, the output of pcp_combine is typically passed to point_stat, grid_stat, or mode for verification. Having the 6-hour forecast in GRIB format and all other forecast hours in NetCDF format (output of pcp_combine) makes the logic for configuring the other MET tools messy. To make the configuration consistent for all forecast hours, one option is to choose to run pcp_combine as a pass-through to simply reformat from GRIB to NetCDF. Listed below is an example of passing a single record to the pcp_combine -add option to do the reformatting:

```
$MET_BUILD/bin/pcp_combine -add forecast_F06.grb \
'name="APCP"; level="A6";' \
forecast_APCP_06_F06.nc -name APCP_06
```

Reformatting from GRIB to NetCDF may be done for any other reason the user may have. For example, the -name option can be used to define the NetCDF output variable name. Presuming this file is then passed to another MET tool, the new variable name (CompositeReflectivity) will appear in the output of downstream tools:

```
$MET_BUILD/bin/pcp_combine -add forecast.grb \
'name="REFC"; level="L0"; GRIB1_ptv=129; lead_time="120000";' \
forecast.nc -name CompositeReflectivity
```

33.1.4.5 Q. How do I use "-pcprx" to run a project faster?

Answer

To run a project faster, the "-pcprx" option may be used to narrow the search down to whatever regular expression you provide. Here are a two examples:

```
# Only using Stage IV data (ST4)
pcp_combine -sum 00000000_000000 06 \
20161015_18 12 ST4.2016101518.APCP_12_SUM.nc -pcprx "ST4.*.06h"

# Specify that files starting with pgbq[number][number]be used:
pcp_combine \
-sum 20160221_18 06 20160222_18 24 \
gfs_APCP_24_20160221_18_F00_F24.nc \
```

(continues on next page)

(continued from previous page)

```
-pcpdir /scratch4/BMC/shout/ptmp/Andrew.Kren/pre2016c3_corr/temp \
-pcprx 'pgbq[0-9][0-9].gfs.2016022118' -v 3
```

33.1.4.6 Q. How do I enter the time format correctly?

Answer

Here is an **incorrect example** of running pcp_combine with sub-hourly accumulation intervals:

```
# incorrect example:
pcp_combine -subtract forecast.grb 0055 \
forecast2.grb 0005 forecast.nc -field APCP
```

The time signature is entered incorrectly. Let's assume that "0055" meant 0 hours and 55 minutes and "0005" meant 0 hours and 5 minutes.

Looking at the usage statement for pcp_combine (just type pcp_combine with no arguments): "accum1" indicates the accumulation interval to be used from in_file1 in HH[MMSS] format (required).

The time format listed "HH[MMSS]" means specifying hours or hours/minutes/seconds. The incorrect example is using hours/minutes.

Below is the **correct example**. Add the seconds to the end of the time strings, like this:

```
# correct example:
pcp_combine -subtract forecast.grb 005500 \
forecast2.grb 000500 forecast.nc -field APCP
```

33.1.4.7 Q. How do I use Pcp-Combine when my GRIB data doesn't have the appropriate accumulation interval time range indicator?

Answer

Run wgrib on the data files and the output is listed below:

```
279:503477484:d=15062313:APCP:kpds5=61:kpds6=1:kpds7=0:TR=␣
↪10:P1=3:P2=247:TimeU=0:sfc:1015min \
fcst:NAve=0 \
279:507900854:d=15062313:APCP:kpds5=61:kpds6=1:kpds7=0:TR=␣
↪10:P1=3:P2=197:TimeU=0:sfc:965min \
fcst:NAve=0
```

Notice the output which says "TR=10". TR means time range indicator and a value of 10 means that the level information contains an instantaneous forecast time, not an accumulation interval.

Here's a table describing the TR values: http://www.nco.ncep.noaa.gov/pmb/docs/on388/table5.html

The default logic for pcp_combine is to look for GRIB code 61 (i.e. APCP) defined with an accumulation interval (TR = 4). Since the data doesn't meet that criteria, the default logic of pcp_combine won't work. The arguments need to be more specific to tell pcp_combine exactly what to do.

Try the command:

```
pcp_combine -subtract \
forecast.grb 'name="APCP"; level="L0"; lead_time="165500";' \
forecast2.grb 'name="APCP"; level="L0"; lead_time="160500";' \
forecast.nc -name APCP_A005000
```

Some things to point out here:

1. Notice in the wgrib output that the forecast times are 1015 min and 965 min. In HHMMSS format, that's "165500" and "160500".
2. An accumulation interval can't be specified since the data isn't stored that way. Instead, use a config file string to describe the data to use.
3. The config file string specifies a "name" (APCP) and "level" string. APCP is defined at the surface, so a level value of 0 (L0) was specified.
4. Technically, the "lead_time" doesn't need to be specified at all, pcp_combine would find the single APCP record in each input GRIB file and use them. But just in case, the lead_time option was included to be extra certain to get exactly the data that is needed.
5. The default output variable name pcp_combine would write would be "APCP_L0". However, to indicate that its a 50-minute "accumulation interval" use a different output variable name (APCP_A005000). Any string name is possible. Maybe "Precip50Minutes" or "RAIN50". But whatever string is chosen will be used in the Grid-Stat, Point-Stat, or MODE config file to tell that tool what variable to process.

33.1.4.8 Q. How do I use "-sum", "-add", and "-subtract" to achieve the same accumulation interval?

Answer

Here is an example of using pcp_combine to put GFS into 24- hour intervals for comparison against 24-hourly StageIV precipitation with GFS data through the pcp_combine tool. Be aware that the 24-hour StageIV data is defined as an accumulation from 12Z on one day to 12Z on the next day: http://www.emc.ncep.noaa.gov/mmb/ylin/pcpanl/stage4/

Therefore, only the 24-hour StageIV data can be used to evaluate 12Z to 12Z accumulations from the model. Alternatively, the 6- hour StageIV accumulations could be used to evaluate any 24 hour accumulation from the model. For the latter, run the 6-hour StageIV files through pcp_combine to generate the desired 24-hour accumulation.

Here is an example. Run pcp_combine to compute 24-hour accumulations for GFS. In this example, process the 20150220 00Z initialization of GFS.

```
pcp_combine \
-sum 20150220_00 06 20150221_00 24 \
gfs_APCP_24_20150220_00_F00_F24.nc \
-pcprx "gfs_4_20150220_00.*grb2" \
-pcpdir /d1/model_data/20150220
```

pcp_combine is looking in the */d1/SBU/GFS/model_data/20150220* directory at files which match this regular expression "gfs_4_20150220_00.*grb2". That directory contains data for 00, 06, 12, and 18 hour initializations, but the "-pcprx" option narrows the search down to the 00 hour initialization which makes it run faster. It inspects all the matching files, looking for 6-hour APCP data to sum up to a 24-hour accumulation valid at 20150221_00. This results in a 24-hour accumulation between forecast hours 0 and 24.

The following command will compute the 24-hour accumulation between forecast hours 12 and 36:

```
pcp_combine \
-sum 20150220_00 06 20150221_12 24 \
gfs_APCP_24_20150220_00_F12_F36.nc \
-pcprx "gfs_4_20150220_00.*grb2" \
-pcpdir /d1/model_data/20150220
```

The "-sum" command is meant to make things easier by searching the directory. But instead of using "-sum", another option would be the "- add" command. Explicitly list the 4 files that need to be extracted from the 6-hour APCP and add them up to 24. In the directory structure, the previous "-sum" job could be rewritten with "-add" like this:

```
pcp_combine -add \
/d1/model_data/20150220/gfs_4_20150220_0000_018.grb2 06 \
/d1/model_data/20150220/gfs_4_20150220_0000_024.grb2 06 \
/d1/model_data/20150220/gfs_4_20150220_0000_030.grb2 06 \
/d1/model_data/20150220/gfs_4_20150220_0000_036.grb2 06 \
gfs_APCP_24_20150220_00_F12_F36_add_option.nc
```

This example explicitly tells pcp_combine which files to read and what accumulation interval (6 hours) to extract from them. The resulting output should be identical to the output of the "-sum" command.

33.1.4.9 Q. What is the difference between "-sum" vs. "-add"?

Answer

The -sum and -add options both do the same thing. It's just that '-sum' could find files more quickly with the use of the -pcprx flag. This could also be accomplished by using a calling script.

33.1.4.10 Q. How do I select a specific GRIB record?

Answer

In this example, record 735 needs to be selected.

```
pcp_combine -add 20160101_i12_f015_HRRR_wrfnat.grb2 \
'name="APCP"; level="R735";' \
-name "APCP_01" HRRR_wrfnat.20160101_i12_f015.nc
```

Instead of having the level as "L0", tell it to use "R735" to select grib record 735.

33.1.5 Plot-Data-Plane

33.1.5.1 Q. How do I inspect Gen-Vx-Mask output?

Answer

Check to see if the call to Gen-Vx-Mask actually did create good output with Plot-Data-Plane. The following commands assume that the MET executables are found in your path.

```
plot_data_plane \
out/gen_vx_mask/CONUS_poly.nc \
out/gen_vx_mask/CONUS_poly.ps \
'name="CONUS"; level="(*,*)";'
```

View that postscript output file, using something like "gv" for ghostview:

```
gv out/gen_vx_mask/CONUS_poly.ps
```

Please review a map of 0's and 1's over the USA to determine if the output file is what the user expects. It always a good idea to start with plot_data_plane when working with data to make sure MET is plotting the data correctly and in the expected location.

33.1.5.2 Q. How do I specify the GRIB version?

Answer

When MET reads Gridded data files, it must determine the type of file it's reading. The first thing it checks is the suffix of the file. The following are all interpreted as GRIB1: .grib, .grb, and .gb. While these mean GRIB2: .grib2, .grb2, and .gb2.

There are 2 choices to control how MET interprets a grib file. Renaming the files to use a particular suffix, or keep them named and explicitly tell MET to interpret them as GRIB1 or GRIB2 using the "file_type" configuration option.

The examples below use the plot_data_plane tool to plot the data. Set

```
"file_type = GRIB2;"
```

To keep the files named this as they are, add "file_type = GRIB2;" to all the MET configuration files (i.e. Grid-Stat, MODE, and so on) that you use:

```
plot_data_plane \
test_2.5_prog.grib \
test_2.5_prog.ps \
'name="TSTM"; level="A0"; file_type=GRIB2;' \
-plot_range 0 100
```

33.1.5.3 Q. How do I test the variable naming convention? (Record number example.)

Answer

Make sure MET can read GRIB2 data. Plot the data from that GRIB2 file by running:

```
plot_data_plane LTIA98_KWBR_201305180600.grb2 tmp_z2.ps 'name="TMP"; level="R2";
```

"R2" tells MET to plot record number 2. Record numbers 1 and 2 both contain temperature data and 2-meters. Here's some wgrib2 output:

```
1:0:d=2013051806:TMP:2 m above ground:anl:analysis/forecast error␣
↪2:3323062:d=2013051806:TMP:2 m above ground:anl:
```

The GRIB id info has been the same between records 1 and 2.

33.1.5.4 Q. How do I compute and verify wind speed?

Answer

Here's how to compute and verify wind speed using MET. Good news, MET already includes logic for deriving wind speed on the fly. The GRIB abbreviation for wind speed is WIND. To request WIND from a GRIB1 or GRIB2 file, MET first checks to see if it already exists in the current file. If so, it'll use it as is. If not, it'll search for the corresponding U and V records and derive wind speed to use on the fly.

In this example the RTMA file is named rtma.grb2 and the UPP file is named wrf.grb, please try running the following commands to plot wind speed:

```
plot_data_plane wrf.grb wrf_wind.ps \
'name"WIND"; level="Z10";' -v 3
plot_data_plane rtma.grb2 rtma_wind.ps \
'name"WIND"; level="Z10";' -v 3
```

In the first call, the log message should be similar to this:

```
DEBUG 3: MetGrib1DataFile::data_plane_array() ->
Attempt to derive winds from U and V components.
```

In the second one, this won't appear since wind speed already exists in the RTMA file.

33.1.6 Stat-Analysis

33.1.6.1 Q. How does '-aggregate_stat' work?

Answer

In Stat-Analysis, there is a "-vx_mask" job filtering option. That option reads the VX_MASK column from the input STAT lines and applies string matching with the values in that column. Presumably, all of the MPR lines will have the value of "FULL" in the VX_MASK column.

Stat-Analysis has the ability to read MPR lines and recompute statistics from them using the same library code that the other MET tools use. The job command options which begin with "-out" are used to specify settings to be applied to the output of that process. For example, the "-fcst_thresh" option filters strings from the input "FCST_THRESH" header column. The "-out_fcst_thresh" option defines the threshold to be applied to the output of Stat-Analysis. So reading MPR lines and applying a threshold to define contingency table statistics (CTS) would be done using the "-out_fcst_thresh" option.

Stat-Analysis does have the ability to filter MPR lat/lon locations using the "-mask_poly" option for a lat/lon polyline and the "-mask_grid" option to define a retention grid.

However, there is currently no "-mask_sid" option.

With MET-5.2 and later versions, one option is to apply column string matching using the "-column_str" option to define the list of station ID's you would like to aggregate. That job would look something like this:

```
stat_analysis -lookin path/to/mpr/directory \
-job aggregate_stat -line_type MPR -out_line_type CNT \
-column_str OBS_SID SID1,SID2,SID3,...,SIDN \
-set_hdr VX_MASK SID_GROUP_NAME \
-out_stat mpr_to_cnt.stat
```

Where SID1...SIDN is a comma-separated list of the station id's in the group. Notice that a value for the output VX_MASK column using the "-set_hdr" option has been specified. Otherwise, this would show a list of the unique values found in that column. Presumably, all the input VX_MASK columns say "FULL" so that's what the output would say. Use "-set_hdr" to explicitly set the output value.

33.1.6.2 Q. What is the best way to average the FSS scores within several days or even several months using 'Aggregate to Average Scores'?

Answer

Below is the best way to aggregate together the Neighborhood Continuous (NBRCNT) lines across multiple days, specifically the fractions skill score (FSS). The Stat-Analysis tool is designed to do this. This example is for aggregating scores for the accumulated precipitation (APCP) field.

Run the "aggregate" job type in stat_analysis to do this:

```
stat_analysis -lookin directory/file*_nbrcnt.txt \
-job aggregate -line_type NBRCNT -by FCST_VAR,FCST_LEAD,FCST_THRESH,INTERP_MTHD,
↪INTERP_PNTS -out_stat agg_nbrcnt.txt
```

This job reads all the files that are passed to it on the command line with the "-lookin" option. List explicit filenames to read them directly. Listing a top-level directory name will search that directory for files ending in ".stat".

In this case, the job running is to "aggregate" the "NBRCNT" line type.

In this case, the "-by" option is being used and lists several header columns. Stat-Analysis will run this job separately for each unique combination of those header column entries.

The output is printed to the screen, or use the "-out_stat" option to also write the aggregated output to a file named "agg_nbrcnt.txt".

33.1.6.3 Q. How do I use '-by' to capture unique entries?

Answer

Here is a stat-analysis job that could be used to run, read the MPR lines, define the probabilistic forecast thresholds, define the single observation threshold, and compute a PSTD output line. Using "-by FCST_VAR" tells it to run the job separately for each unique entry found in the FCST_VAR column.

```
stat_analysis \
-lookin point_stat_model2_120000L_20160501_120000V.stat \
-job aggregate_stat -line_type MPR -out_line_type PSTD \
-out_fcst_thresh ge0,ge0.1,ge0.2,ge0.3,ge0.4,ge0.5,ge0.6,ge0.7,ge0.8,ge0.9,ge1.0 \
-out_obs_thresh eq1.0 \
-by FCST_VAR \
-out_stat out_pstd.txt
```

The output statistics are written to "out_pstd.txt".

33.1.6.4 Q. How do I use '-filter' to refine my output?

Answer

Here is an example of running a Stat-Analysis filter job to discard any CNT lines (continuous statistics) where the forecast rate and observation rate are less than 0.05. This is an alternative way of tossing out those cases without having to modify the source code.

```
stat_analysis \
-lookin out/grid_stat/grid_stat_120000L_20050807_120000V.stat \
-job filter -dump_row filter_cts.txt -line_type CTS \
-column_min BASER 0.05 -column_min FMEAN 0.05
DEBUG 2: STAT Lines read = 436
DEBUG 2: STAT Lines retained = 36
DEBUG 2:
DEBUG 2: Processing Job 1: -job filter -line_type CTS -column_min BASER
0.05 -column_min
FMEAN 0.05 -dump_row filter_cts.txt
DEBUG 1: Creating
STAT output file "filter_cts.txt"
FILTER: -job filter -line_type
CTS -column_min
BASER 0.05 -column_min
FMEAN 0.05 -dump_row filter_cts.txt
DEBUG 2: Job 1 used 36 out of 36 STAT lines.
```

This job reads find 56 CTS lines, but only keeps 36 of them where both the BASER and FMEAN columns are at least 0.05.

33.1.6.5 Q. How do I use the "-by" flag to stratify results?

Answer

Adding "-by FCST_VAR" is a great way to associate a single value, of say RMSE, with each of the forecast variables (UGRD,VGRD and WIND).

Run the following job on the output from Grid-Stat generated when the "make test" command is run:

```
stat_analysis -lookin out/grid_stat \
-job aggregate_stat -line_type SL1L2 -out_line_type CNT \
-by FCST_VAR,FCST_LEV \
-out_stat cnt.txt
```

The resulting cnt.txt file includes separate output for 6 different FCST_VAR values at different levels.

33.1.6.6 Q. How do I speed up run times?

Answer

By default, Stat-Analysis has two options enabled which slow it down. Disabling these two options will create quicker run times:

1. The computation of rank correlation statistics, Spearman's Rank Correlation and Kendall's Tau. Disable them using "-rank_corr_flag FALSE".
2. The computation of bootstrap confidence intervals. Disable them using "-n_boot_rep 0".

Two more suggestions for faster run times.

1. Instead of using "-fcst_var u", use "-by fcst_var". This will compute statistics separately for each unique entry found in the FCST_VAR column.
2. Instead of using "-out" to write the output to a text file, use "-out_stat" which will write a full STAT output file, including all the header columns. This will create a long list of values in the OBTYPE column. To avoid the long, OBTYPE column value, manually set the output using "-set_hdr OBTYPE ALL_TYPES". Or set its value to whatever is needed.

```
stat_analysis \
-lookin diag_conv_anl.2015060100.stat \
-job aggregate_stat -line_type MPR -out_line_type CNT -by FCST_VAR \
-out_stat diag_conv_anl.2015060100_cnt.txt -set_hdr OBTYPE ALL_TYPES \
-n_boot_rep 0 -rank_corr_flag FALSE -v 4
```

Adding the "-by FCST_VAR" option to compute stats for all variables and runs quickly.

33.1.7 TC-Stat

33.1.7.1 Q. How do I use the "-by" flag to stratify results?

Answer

To perform tropical cyclone evaluations for multiple models use the "-by AMODEL" option with the tc_stat tool. Here is an example.

In this case the tc_stat job looked at the 48 hour lead time for the HWRF and H3HW models. Without the "-by AMODEL" option, the output would be all grouped together.

```
tc_stat \
-lookin d2014_vx_20141117_reset/al/tc_pairs/tc_pairs_H3WI_* \
-lookin d2014_vx_20141117_reset/al/tc_pairs/tc_pairs_HWFI_* \
-job summary -lead 480000 -column TRACK -amodel HWFI,H3WI \
-by AMODEL -out sample.out
```

This will result in all 48 hour HWFI and H3WI track forecasts to be aggregated (statistics and scores computed) for each model separately.

33.1.7.2 Q. How do I use rapid intensification verification?

Answer

To get the most output, run something like this:

```
tc_stat \
-lookin path/to/tc_pairs/output \
-job rirw -dump_row test \
-out_line_type CTC,CTS,MPR
```

By default, rapid intensification (RI) is defined as a 24-hour exact change exceeding 30kts. To define RI differently, modify that definition using the ADECK, BDECK, or both using -rirw_time, -rirw_exact, and -rirw_thresh options. Set -rirw_window to something larger than 0 to enable false alarms to be considered hits when they were "close enough" in time.

```
tc_stat \
-lookin path/to/tc_pairs/output \
-job rirw -dump_row test \
-rirw_time 36 -rirw_window 12 \
-out_line_type CTC,CTS,MPR
```

To evaluate Rapid Weakening (RW) by setting "-rirw_thresh <=-30". To stratify your results by lead time, you could add the "-by LEAD" option.

```
tc_stat \
-lookin path/to/tc_pairs/output \
-job rirw -dump_row test \
-rirw_time 36 -rirw_window 12 \
-rirw_thresh <=-30 -by LEAD \
-out_line_type CTC,CTS,MPR
```

33.1.8 Utilities

33.1.8.1 Q. What would be an example of scripting to call MET?

Answer

The following is an example of how to call MET from a bash script including passing in variables. This shell script is listed below to run Grid-Stat, call Plot-Data-Plane to plot the resulting difference field, and call convert to reformat from PostScript to PNG.

```
#!/bin/sh
for case in `echo "FCST OBS"`; do
export TO_GRID=${case}
grid_stat gfs.t00z.pgrb2.0p25.f000 \
```

(continues on next page)

(continued from previous page)

```
nam.t00z.conusnest.hiresf00.tm00.grib2 GridStatConfig
plot_data_plane \
*TO_GRID_${case}*_pairs.nc TO_GRID_${case}.ps 'name="DIFF_TMP_P500_TMP_P500_FULL";␣
↪\
level="(*,*)";'
convert -rotate 90 -background white -flatten TO_GRID_${case}.ps
TO_GRID_${case}.png
done
```

33.1.8.2 Q. How do I convert TRMM data files?

Answer

Here is an example of NetCDF that the MET software is not expecting. Here is an option for accessing that same TRMM data, following links from the MET website: http://dtcenter.org/community-code/model-evaluation-tools-met/input-data

```
# Pull binary 3-hourly TRMM data file
wget
ftp://disc2.nascom.nasa.gov/data/TRMM/Gridded/3B42_V7/201009/3B42.100921.00z.7.
precipitation.bin
# Pull Rscript from MET website
wget http://dtcenter.org/sites/default/files/community-code/met/r-scripts/
↪trmmbin2nc.R
# Edit that Rscript by setting
out_lat_ll = -50
out_lon_ll = 0
out_lat_ur = 50
out_lon_ur = 359.75
# Run the Rscript
Rscript trmmbin2nc.R 3B42.100921.00z.7.precipitation.bin \
3B42.100921.00z.7.precipitation.nc
# Plot the result
plot_data_plane 3B42.100921.00z.7.precipitation.nc \
3B42.100921.00z.7.precipitation.ps 'name="APCP_03"; level="(*,*)";'
```

It may be possible that the domain of the data is smaller. Here are some options:

1. In that Rscript, choose different boundaries (i.e. out_lat/lon_ll/ur) to specify the tile of data to be selected.

2. As of version 5.1, MET includes support for regridding the data it reads. Keep TRMM on it's native domain and use the MET tools to do the regridding. For example, the Regrid-Data-Plane" tool reads a NetCDF file, regrids the data, and writes a NetCDF file. Alternatively, the "regrid" section of the configuration files for the MET tools may be used to do the regridding on the fly. For example, run Grid-Stat to compare to the model output to TRMM and say

```
"regrid = { field = FCST;
...}"
```

That tells Grid-Stat to automatically regrid the TRMM observations to the model domain.

33.1.8.3 Q. How do I convert a PostScript to png?

Answer

Use the linux "convert" tool to convert a Plot-Data-Plane PostScript file to a png:

```
convert -rotate 90 -background white plot_dbz.ps plot_dbz.png
```

To convert a MODE PostScript to png

```
convert mode_out.ps mode_out.png
```

Will result in all 6-7 pages in the PostScript file be written out to a seperate .png with the following naming convention:

mode_out-0.png, mode_out-1.png, mode_out-2.png, etc.

33.1.8.4 Q. How does pairwise differences using plot_tcmpr.R work?

Answer

One necessary step in computing pairwise differences is "event equalizing" the data. This means extracting a subset of cases that are common to both models.

While the tc_stat tool does not compute pairwise differences, it can apply the "event_equalization" logic to extract the cases common to two models. This is done using the config file "event_equal = TRUE;" option or setting "-event_equal true" on the command line.

Most of the hurricane track analysis and plotting is done using the plot_tcmpr.R Rscript. It makes a call to the tc_stat tool to track data down to the desired subset, compute pairwise differences if needed, and then plot the result.

```
Rscript ${MET_BASE}/Rscripts/plot_tcmpr.R \
-lookin tc_pairs_output.tcst \
-filter '-amodel AHWI,GFSI' \
-series AMODEL AHWI,GFSI,AHWI-GFSI \
-plot MEAN,BOXPLOT
```

The resulting plots include three series - one for AHWI, one for GFSI, and one for their pairwise difference.

It's a bit cumbersome to understand all the options available, but this may be really useful. If nothing else, it could be adapted to dump out the pairwise differences that are needed.

33.1.9 Miscellaneous

33.1.9.1 Q. Regrid-Data-Plane - How do I define a LatLon grid?

Answer

Here is an example of the NetCDF variable attributes that MET uses to define a LatLon grid:

```
:Projection = "LatLon" ;
:lat_ll = "25.063000 degrees_north" ;
:lon_ll = "-124.938000 degrees_east" ;
:delta_lat = "0.125000 degrees" ;
:delta_lon = "0.125000 degrees" ;
:Nlat = "224 grid_points" ;
:Nlon = "464 grid_points" ;
```

This can be created by running the Regrid-Data-Plane" tool to regrid some GFS data to a LatLon grid:

```
regrid_data_plane \
gfs_2012040900_F012.grib G110 \
gfs_g110.nc -field 'name="TMP"; level="Z2";'
```

Use ncdump to look at the attributes. As an exercise, try defining these global attributes (and removing the other projection-related ones) and then try again.

33.1.9.2 Q. Pre-processing - How do I use wgrib2, pcp_combine regrid and reformat to format NetCDF files?

Answer

If you are extracting only one or two fields from a file, using MET's Regrid-Data-Plane can be used to generate a Lat-Lon projection. If regridding all fields, the wgrib2 utility may be more useful. Here's an example of using wgrib2 and pcp_combine to generate NetCDF files MET can read:

```
wgrib2 gfsrain06.grb -new_grid latlon 112:131:0.1 \
25:121:0.1 gfsrain06_regrid.grb2
```

And then run that GRIB2 file through pcp_combine using the "-add" option with only one file provided:

```
pcp_combine -add gfsrain06_regrid.grb2 'name="APCP"; \
level="A6";' gfsrain06_regrid.nc
```

Then the output NetCDF file does not have this problem:

```
ncdump -h 2a_wgrib2_regrid.nc | grep "_ll"
:lat_ll = "25.000000 degrees_north" ;
:lon_ll = "112.000000 degrees_east" ;
```

33.1.9.3 Q. TC-Pairs - How do I get rid of WARNING: TrackInfo Using Specify Model Suffix?

Answer

Below is a command example to run:

```
tc_pairs \
-adeck aep142014.h4hw.dat \
-bdeck bep142014.dat \
-config TCPairsConfig_v5.0 \
-out tc_pairs_v5.0_patch \
-log tc_pairs_v5.0_patch.log \
-v 3
```

Below is a warning message:

```
WARNING: TrackInfo::add(const ATCFLine &) ->
skipping ATCFLine since the valid time is not
increasing (20140801_000000 < 20140806_060000):
WARNING: AL, 03, 2014080100, 03, H4HW, 000,
120N, 547W, 38, 1009, XX, 34, NEQ, 0084, 0000,
0000, 0083, -99, -99, 59, 0, 0, , 0, , 0, 0,
```

As a sanity check, the MET-TC code makes sure that the valid time of the track data doesn't go backwards in time. This warning states that this is occurring. The very likely reason for this is that the data being used are probably passing tc_pairs duplicate track data.

Using grep, notice that the same track data shows up in "aal032014.h4hw.dat" and "aal032014_hfip_d2014_BERTHA.dat". Try this:

```
grep H4HW aal*.dat | grep 2014080100 | grep ", 000,"
aal032014.h4hw.dat:AL, 03, 2014080100, 03, H4HW, 000,
120N, 547W, 38, 1009, XX, 34, NEQ, 0084,
0000, 0000, 0083, -99, -99, 59, 0, 0, ,
0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS,
-9999, -9999, -9999, Y, 10, DT, -999
aal032014_hfip_d2014_BERTHA.dat:AL, 03, 2014080100,
03, H4HW, 000, 120N, 547W, 38, 1009, XX, 34, NEQ,
0084, 0000, 0000, 0083, -99, -99, 59, 0, 0, , 0, , 0,
0, , , , , 0, 0, 0, 0, THERMOPARAMS, -9999 ,-9999 ,
-9999 ,Y ,10 ,DT ,-999
```

Those 2 lines are nearly identical, except for the spelling of "THERMO PARAMS" with a space vs "THERMOPARAMS" with no space.

Passing tc_pairs duplicate track data results in this sort of warning. The DTC had the same sort of problem when setting up a real-time verification system. The same track data was making its way into multiple ATCF files.

If this really is duplicate track data, work on the logic for where/how to store the track data. However, if the H4HW data in the first file actually differs from that in the second file, there is another option. You can specify a model suffix to be used for each ADECK source, as in this example (suffix=_EXP):

```
tc_pairs \
-adeck aal032014.h4hw.dat suffix=_EXP \
-adeck aal032014_hfip_d2014_BERTHA.dat \
-bdeck bal032014.dat \
-config TCPairsConfig_match \
-out tc_pairs_v5.0_patch \
-log tc_pairs_v5.0_patch.log -v 3
```

Any model names found in "aal032014.h4hw.dat" will now have _EXP tacked onto the end. Note that if a list of model names in the TCPairsConfig file needs specifying, include the _EXP variants to get them to show up in the output or it won't show up.

That'll get rid of the warnings because they will be storing the track data from the first source using a slightly different model name. This feature was added for users who are testing multiple versions of a model on the same set of storms. They might be using the same ATCF ID in all their output. But this enables them to distinguish the output in tc_pairs.

33.1.9.4 Q. Why is the grid upside down?

Answer

The user provides a gridded data file to MET and it runs without error, but the data is packed upside down.

Try using the "file_type" entry. The "file_type" entry specifies the input file type (e.g. GRIB1, GRIB2, NETCDF_MET, NETCDF_WRF, NETCDF_PINT, NETCDF_NCCF) rather than letting the code determine it itself. For valid file_type values, see "File types" in the *data/config/ConfigConstants* file. This entry should be defined within the "fcst" or "obs" dictionaries. Sometimes, directly specifying the type of file will help MET figure out what to properly do with the data.

Another option is to use the Regrid-Data-Plane tool. The Regrid-Data-Plane tool may be run to read data from any gridded data file MET supports (i.e. GRIB1, GRIB2, and a variety of NetCDF formats), interpolate to a user-specified grid, and write the field(s) out in NetCDF format. See the Regrid-Data-Plane tool Section 8.2 in the MET User's Guide for more detailed information. While the Regrid-Data-Plane tool is useful as a stand-alone tool, the capability is also included to automatically regrid data in most of the MET tools that handle gridded data. This "regrid" entry is a dictionary containing information about how to handle input gridded data files. The "regird"

entry specifies regridding logic and has a "to_grid" entry that can be set to NONE, FCST, OBS, a named grid, the path to a gridded data file defining the grid, or an explicit grid specification string. See the *regrid* (page 53) entry in the Configuration File Overview in the MET User's Guide for a more detailed description of the configuration file entries that control automated regridding.

A single model level can be plotted using the plot_data_plane utility. This tool can assist the user by showing the data to be verified to ensure that times and locations matchup as expected.

33.1.9.5 Q. Why was the MET written largely in C++ instead of FORTRAN?

Answer

MET relies upon the object-oriented aspects of C++, particularly in using the MODE tool. Due to time and budget constraints, it also makes use of a pre-existing forecast verification library that was developed at NCAR.

33.1.9.6 Q. How does MET differ from the previously mentioned existing verification packages?

Answer

MET is an actively maintained, evolving software package that is being made freely available to the public through controlled version releases.

33.1.9.7 Q. Will the MET work on data in native model coordinates?

Answer

No - it will not. In the future, we may add options to allow additional model grid coordinate systems.

33.1.9.8 Q. How do I get help if my questions are not answered in the User's Guide?

Answer

First, look on our MET User's Guide website. If that doesn't answer your question, create a post in the METplus GitHub Discussions Forum.

33.1.9.9 Q. What graphical features does MET provide?

Answer

MET provides some *plotting and graphics support* (page 477). The plotting tools, including plot_point_obs, plot_data_plane, and plot_mode_field, can help users visualize the data.

MET is intended to be a set of command line tools for evaluating forecast quality. So, the development effort is focused on providing the latest, state of the art verification approaches, rather than on providing nice plotting features. However, the ASCII output statistics of MET may be plotted with a wide variety of plotting packages, including R, NCL, IDL, and GNUPlot. METViewer is also currently being developed and used by the DTC and NOAA It creates basic plots of MET output verification statistics. The types of plots include series plots with confidence intervals, box plots, x-y scatter plots and histograms.

R is a language and environment for statistical computing and graphics. It's a free package that runs on most operating systems and provides nice plotting features and a wide array of powerful statistical analysis tools. There are sample scripts on the MET website that you can use and modify to perform the type of analysis you need. If you create your own scripts, we encourage you to submit them to us through the METplus GitHub Discussions Forum so that we can post them for other users.

33.1.9.10 Q. How do I find the version of the tool I am using?

Answer

Type the name of the tool followed by **–version**. For example, type "pb2nc **–version**".

33.1.9.11 Q. What are MET's conventions for latitude, longitude, azimuth and bearing angles?

Answer

MET considers north latitude and east longitude positive. However, internally MET considers east longitude negative so users may encounter DEBUG statements with longitude of a different sign than they provided (e.g. for observation locations or grid metadata). Latitudes have range from -90° to $+90^\circ$. Longitudes have range from -180° to $+180^\circ$. Plane angles such as azimuths and bearing (example: horizontal wind direction) have range 0° to 360° and are measured clockwise from the north.

33.2 Troubleshooting

The first place to look for help with individual commands is this User's Guide or the usage statements that are provided with the tools. Usage statements for the individual MET tools are available by simply typing the name of the executable in MET's *bin/* directory. Example scripts available in the MET's *scripts/* directory show examples of how one might use these commands on example datasets. Here are suggestions on other things to check if you are having problems installing or running MET.

33.2.1 MET Won't Compile

Troubleshooting Help

- Have you specified the locations of NetCDF, GNU Scientific Library, and BUFRLIB, and optional additional libraries using corresponding MET_ environment variables prior to running configure?
- Have these libraries been compiled and installed using the same set of compilers used to build MET?

33.2.2 BUFRLIB Errors During MET Installation

Troubleshooting Help

```
error message: /usr/bin/ld: cannot find -lbufr
The linker can not find the BUFRLIB library archive file it needs.

export MET_BUFRLIB=/home/username/BUFRLIB_v11.3.0:$MET_BUFRLIB
```

It isn't making it's way into the configuration because BUFRLIB_v11.3.0 isn't showing up in the output of make. This may indicate the wrong shell type. The .bashrc file sets the environment for the Bourne shell, but the above error could indicate that the c- shell is being used instead.

Try the following 2 things:

1. Check to make sure this file exists:

   ```
   ls /home/username/BUFRLIB_v11.3.0/libbufr.a
   ```

2. Rerun the MET configure command using the following option on the command line:

   ```
   MET_BUFRLIB=/home/username/BUFRLIB_v11.3.0
   ```

After doing that, please try recompiling MET. If it fails, please submit the following log files: "make_install.log" as well as "config.log" with a new post in the METplus GitHub Discussions Forum.

33.2.3 Command Line Double Quotes

Troubleshooting Help

Single quotes, double quotes, and escape characters can be difficult for MET to parse. If there are problems, especially in Python code, try breaking the command up like the below example.

```
['regrid_data_plane',
'/h/data/global/WXQC/data/umm/1701150006',
'G003', '/h/data/global/WXQC/data/met/nc_mdl/umm/1701150006', '- field',
'\'name="HGT"; level="P500";\'', '-v', '6']
```

33.2.4 Environment Variable Settings

Troubleshooting Help

In the below incorrect example for many environment variables have both the main variable set and the INC and LIB variables set:

```
export MET_GSL=$MET_LIB_DIR/gsl
export MET_GSLINC=$MET_LIB_DIR/gsl/include/gsl
export MET_GSLLIB=$MET_LIB_DIR/gsl/lib
```

only MET_GSL *OR *MET_GSLINC *AND *MET_GSLLIB need to be set. So, for example, either set:

```
export MET_GSL=$MET_LIB_DIR/gsl
```

or set:

```
export MET_GSLINC=$MET_LIB_DIR/gsl/include/gsl export MET_GSLLIB=$MET_LIB_DIR/gsl/
↪lib
```

Additionally, MET does not use MET_HDF5INC and MET_HDF5LIB. It only uses MET_HDF5.

Our online tutorial can help figure out what should be set and what the value should be: https://met.readthedocs.io/en/latest/Users_Guide/installation.html

33.2.5 NetCDF Install Issues

Troubleshooting Help

This example shows a problem with NetCDF in the make_install.log file:

```
/usr/bin/ld: warning: libnetcdf.so.11,
needed by /home/zzheng25/metinstall/lib/libnetcdf_c++4.so,
may conflict with libnetcdf.so.7
```

Below are examples of too many MET_NETCDF options:

```
MET_NETCDF='/home/username/metinstall/'
MET_NETCDFINC='/home/username/local/include'
MET_NETCDFLIB='/home/username/local/lib'
```

Either MET_NETCDF **OR** MET_NETCDFINC **AND** MET_NETCDFLIB need to be set. If the NetCDF include files are in */home/username/local/include* and the NetCDF library files are in */home/username/local/lib*, unset the MET_NETCDF environment variable, then run "make clean", reconfigure, and then run "make install" and "make test" again.

33.2.6 Error While Loading Shared Libraries

Troubleshooting Help

- Add the lib dir to your LD_LIBRARY_PATH. For example, if you receive the following error: "./mode_analysis: error while loading shared libraries: libgsl.so.19: cannot open shared object file: No such file or directory", you should add the path to the gsl lib (for example, */home/user/MET/gsl-2.1/lib*) to your LD_LIBRARY_PATH.

33.2.7 General Troubleshooting

Troubleshooting Help

- For configuration files used, make certain to use empty square brackets (e.g. [ ]) to indicate no stratification is desired. Do NOT use empty double quotation marks inside square brackets (e.g. [""]).
- Have you designated all the required command line arguments?
- Try rerunning with a higher verbosity level. Increasing the verbosity level to 4 or 5 prints much more diagnostic information to the screen.

33.3 Where to Get Help

If none of the above suggestions have helped solve your problem, help is available through the METplus GitHub Discussions Forum.

33.4 How to Contribute Code

If you have code you would like to contribute, we will gladly consider your contribution. Please create a post in the METplus GitHub Discussions Forum.

Chapter 34

Appendix B Map Projections, Grids, and Polylines

34.1 Map Projections

The following map projections are currently supported in MET:

- Lambert Conformal Projection
- Lambert Azimuthal Equal Area Projection
- Polar Stereographic Projection (Northern)
- Polar Stereographic Projection (Southern)
- Mercator Projection
- Lat/Lon Projection
- Rotated Lat/Lon Projection
- Gaussian Projection
- Range/Azimuth Projection
- Semi Lat/Lon

34.2 Grid Specification Strings

Several configuration file and command line options support the definition of grids as a grid specification string. A description of the that string for each of the supported grid types is provided below.

34.2.1 Lambert Conformal Grid

To specify a Lambert Conformal Grid, the syntax is

```
lambert Nx Ny lat_ll lon_ll lon_orient D_km R_km standard_lat_1 [ standard_lat_2 ] N|S
```

Here, **Nx** and **Ny** are the number of points in, respectively, the **x** and **y** grid directions. These two numbers give the overall size of the grid. **lat_ll** and **lon_ll** are the latitude and longitude, in degrees, of the lower left point of the grid. North latitude and east longitude are considered positive. **lon_orient** is the orientation longitude of the grid. It's the meridian of longitude that's parallel to one of the vertical grid directions. **D_km** and **R_km** are the grid resolution and the radius of the Earth, both in kilometers. **standard_lat_1** and **standard_lat_2** are the standard parallels of the Lambert projection. If the two latitudes are the same, then only one needs to be given. **N|S** means to write either **N** or **S** depending on whether the Lambert projection is from the north pole or the south pole.

As an example of specifying a Lambert grid, suppose you have a northern hemisphere Lambert grid with 614 points in the x direction and 428 points in the y direction. The lower left corner of the grid is at latitude $12.190°$ north and longitude $133.459°$ west. The orientation longitude is $95°$ west. The grid spacing is $12.19058°$ km. The radius of the Earth is the default value used in many grib files: 6367.47 km. Both standard parallels are at $25°$ north. To specify this grid in the config file, you would write

```
To grid = "lambert 614 428 12.190 -133.459 -95.0 12.19058 6367.47 25.0 N";
```

34.2.2 Lambert Azimuthal Equal Area Grid

For a Lambert Azimuthal Equal Area grid, grid specification strings are not supported.

34.2.3 Polar Stereographic Grid

For a Polar Stereographic grid, the syntax is

```
stereo Nx Ny lat_ll lon_ll lon_orient D_km R_km lat_scale N|S
```

Here, **Nx, Ny, lat_ll, lon_ll, lon_orient, D_km** and **R_km** have the same meaning as in the Lambert case. **lat_scale** is the latitude where the grid scale **D_km** is true, while **N|S** means to write either **N** or **S** depending on whether the stereographic projection is from the north pole or the south pole.

34.2.4 Lat/Lon Grid

For Plate Carrée (i.e. Lat/Lon) grids, the syntax is

```
latlon Nx Ny lat_ll lon_ll delta_lat delta_lon
```

The parameters **Nx, Ny, lat_ll** and **lon_ll** are as before. **delta_lat** and **delta_lon** are the latitude and longitude increments of the grid-i.e., the change in latitude or longitude between one grid point and an adjacent grid point.

34.2.5 Rotated Lat/Lon Grid

For a Rotated Plate Carrée (i.e. Rotated Lat/Lon) grids, the syntax is

```
rotlatlon Nx Ny lat_ll lon_ll delta_lat delta_lon true_lat_sp true_lon_sp aux_rotation
```

The parameters **Nx, Ny, lat_ll, lon_ll, delta_lat,** and **delta_lon** are as before. **true_lat_sp** and **true_lon_sp** are the latitude and longitude for the south pole. **aux_rotation** is the auxilary rotation in degrees.

34.2.6 Mercator Grid

For a Mercator grid, the syntax is

```
mercator Nx Ny lat_ll lon_ll lat_ur lon_ur
```

The parameters **Nx, Ny, lat_ll** and **lon_ll** are again as before, while **lat_ur** and **lon_ur** are the latitude and longitude of the upper right corner of the grid.

34.2.7 Gaussian Grid

For a Gaussian grid, the syntax is

```
gaussian lon_zero Nx Ny
```

The parameters **Nx** and **Ny** are as before, while **lon_zero** defines the first longitude.

34.2.8 Range/Azimuth Grid

For a Range/Azimuth grid, the syntax is

```
rngazi range_n azimuth_n range_max_km lat_center lon_center
```

The parameters **lat_center** and **lon_center** define the latitude and longitude for the center of the grid. The **range_n** and **max_range_km** parameters define the number of and maximum value of the ranges, with spacing in kilometers defined as **max_range_km** / (**range_n** - 1). The **azimuth_n** parameter defines the number of equally-spaced azimuth values, with spacing of 360 / **azimuth_n** degrees clockwise from due east.

34.2.9 Semi Lat/Lon Grid

For a Semi Lat/Lon grid, no grid specification string is supported. This grid type is only supported via Python embedding or when reading NetCDF files generated by another MET tool. A Semi Lat/Lon grid defines the information about 2D field of data whose dimension are defined by arrays of latitude (**lats**), longitude (**lons**), level (**levels**), and time (**times**). Times are defined as unixtime, the number of seconds since January 1, 1970. Typically, the lats or lons array and the levels or times array has non-zero length. For example, a zonal mean field is defined using the lats and levels array. A meridional mean field is defined using the lons and levels array. A Hovmoeller field is defined using lats or lons versus times. An arbitrary cross-section is defined by specifying both the lats and lons array with exactly the same length versus levels or times.

Statistics can be computed from data on Semi Lat/Lon grids but only when all data resides on the same Semi Lat/Lon grid. Two Semi Lat/Lon grids are equal when their lats, lons, levels, and times arrays match. No functionality is provided to regrid Semi Lat/Lon data. The MET tools can plot Semi Lat/Lon data, however no map data is overlaid since these grids lack two spatial dimensions.

34.3 Named Grids

The majority of NCEP's predefined grids that reside on one of the projections listed above are implemented in MET. The user can specify one of these NCEP grids in the configuration files as "GNNN", where NNN is the 3-digit NCEP grid number. A small number of other grids are also predefined in the `find_grid_by_name.h` header file in MET's `vx_grid` library. New named grids can be added to that header file but MET must be recompiled for that change to take effect.

Please see NCEP's website for a description and plot of these predefined grids.

34.4 Polylines for NCEP Regions

Many of NCEP's predefined verification regions are implemented in MET as lat/lon polyline files. The user may specify one of these NCEP verification regions in the configuration files by pointing to the lat/lon polyline file in the installed *share/met/poly* directory. Users may also easily define their own lat/lon polyline files.

See NCEP's website for a description and plot of these predefined verification regions.

The NCEP verification regions that are implemented in MET as lat/lon polylines are listed below:

- APL.poly for the Appalachians
- ATC.poly for the Arctic Region
- CAM.poly for Central America
- CAR.poly for the Caribbean Sea
- ECA.poly for Eastern Canada
- GLF.poly for the Gulf of America

- GMC.poly for the Gulf of America Coast
- GRB.poly for the Great Basin
- HWI.poly for Hawaii
- LMV.poly for the Lower Mississippi Valley
- MDW.poly for the Midwest
- MEX.poly for Mexico
- NAK.poly for Northern Alaska
- NAO.poly for Northern Atlantic Ocean
- NEC.poly for the Northern East Coast
- NMT.poly for the Northern Mountain Region
- NPL.poly for the Northern Plains
- NSA.poly for Northern South America
- NWC.poly for Northern West Coast
- PRI.poly for Puerto Rico and Islands
- SAK.poly for Southern Alaska
- SAO.poly for the Southern Atlantic Ocean
- SEC.poly for the Southern East Coast
- SMT.poly for the Southern Mountain Region
- SPL.poly for the Southern Plains
- SWC.poly for the Southern West Coast
- SWD.poly for the Southwest Desert
- WCA.poly for Western Canada
- EAST.poly for the Eastern United States (consisting of APL, GMC, LMV, MDW, NEC, and SEC)
- WEST.poly for the Western United States (consisting of GRB, NMT, NPL, NWC, SMT, SPL, SWC, and SWD)
- CONUS.poly for the Continental United States (consisting of EAST and WEST)

Chapter 35

Appendix C Verification Measures

This appendix provides specific information about the many verification statistics and measures that are computed by MET. These measures are categorized into measures for categorical (dichotomous) variables; measures for continuous variables; measures for probabilistic forecasts and measures for neighborhood methods. While the continuous, categorical, and probabilistic statistics are computed by both the Point-Stat and Grid-Stat tools, the neighborhood verification measures are only provided by the Grid-Stat tool.

35.1 Which statistics are the same, but with different names?

Table 35.1: Statistics in MET and other names they have been published under.

Statistics in MET	Other names for the same statistic
Probability of Detection	Hit Rate
Probability of False Detection	False Alarm Rate (not Ratio)
Critical Success Index	Threat Score
Gilbert Skill Score	Equitable Threat Score
Hanssen and Kuipers Discriminant	True Skill Statistic, Pierce's Skill Score
Heidke Skill Score	Cohen's K
Odds Ratio Skill Score	Yule's Q
Mean Error	Magnitude Bias
Mean Error Squared (ME2)	MSE by Mean Difference
Bias Corrected MSE	MSE by Pattern Variation
MSESS	Murphy's MSESS
Pearson Correlation	Anomalous Pattern Correlation
Anomaly Correlation	Anomalous Correction
Rank Histogram	Talagrand Diagram
Reliability Diagram	Attributes Diagram
Ignorance Score	Logarithmic Scoring Rule

35.2 MET Verification Measures for Categorical (Dichotomous) Variables

The verification statistics for dichotomous variables are formulated using a contingency table such as the one shown in Table 35.2. In this table f represents the forecasts and o represents the observations; the two possible forecast and observation values are represented by the values 0 and 1. The values in Table 35.2 are counts of the number of occurrences of the four possible combinations of forecasts and observations.

Table 35.2: 2x2 contingency table in terms of counts. The $\mathbf{n_{ij}}$ values in the table represent the counts in each forecast-observation category, where **i** represents the forecast and **j** represents the observations. The "." symbols in the total cells represent sums across categories.

Forecast	Observation		Total
	o = 1 (e.g., "Yes")	o = 0 (e.g., "No")	
f = 1 (e.g., "Yes")	$\mathbf{n}_{11}$	$\mathbf{n}_{10}$	$\mathbf{n}_{1.} = \mathbf{n}_{11} + \mathbf{n}_{10}$
f = 0 (e.g., "No")	$\mathbf{n}_{01}$	$\mathbf{n}_{00}$	$\mathbf{n}_{0.} = \mathbf{n}_{01} + \mathbf{n}_{00}$
Total	$\mathbf{n}_{.1} = \mathbf{n}_{11} + \mathbf{n}_{01}$	$\mathbf{n}_{.0} = \mathbf{n}_{10} + \mathbf{n}_{00}$	$T = \mathbf{n}_{11} + \mathbf{n}_{10} + \mathbf{n}_{01} + \mathbf{n}_{00}$

The counts, $n_{11}, n_{10}, n_{01},$ and $n_{00},$ are sometimes called the "Hits", "False alarms", "Misses", and "Correct rejections", respectively.

By dividing the counts in the cells by the overall total, T, the joint proportions, $\mathbf{p}_{11}, \mathbf{p}_{10}, \mathbf{p}_{01},$ and $\mathbf{p}_{00}$ can be computed. Note that $\mathbf{p}_{11}+\mathbf{p}_{10}+\mathbf{p}_{01}+\mathbf{p}_{00} = 1$. Similarly, if the counts are divided by the row (column) totals, conditional proportions, based on the forecasts (observations) can be computed. All of these combinations and the basic counts can be produced by the Point-Stat tool.

The values in Table 35.2 can also be used to compute the F, O, and H relative frequencies that are produced by the NCEP Verification System, and the Point-Stat tool provides an option to produce the statistics in this form. In terms of the other statistics computed by the Point-Stat tool, F is equivalent to the Mean Forecast; H is equivalent to POD; and O is equivalent to the Base Rate. All of these statistics are defined in the subsections below. The Point-Stat tool also provides the total number of observations, **T**.

The categorical verification measures produced by the Point-Stat and Grid-Stat tools are described in the following subsections. They are presented in the order shown in Table 11.2 through Table 11.4.

35.2.1 TOTAL

The total number of forecast-observation pairs, **T**.

35.2.2 Base Rate

Called "O_RATE" in FHO output Table 11.2

Called "BASER" in CTS output Table 11.4

The base rate is defined as $\bar{o} = \frac{n_{11}+n_{01}}{T} = \frac{n_{\cdot 1}}{T}$. This value is also known as the sample climatology, and is the relative frequency of occurrence of the event (i.e., o = 1). The base rate is equivalent to the "O" value produced by the NCEP Verification System.

35.2.3 Mean Forecast

Called "F_RATE" in FHO output Table 11.2

Called "FMEAN" in CTS output Table 11.4

The mean forecast value is defined as $\bar{f} = \frac{n_{11}+n_{10}}{T} = \frac{n_{1\cdot}}{T}$.

This statistic is comparable to the base rate and is the relative frequency of occurrence of a forecast of the event (i.e., **f = 1**). The mean forecast is equivalent to the "F" value computed by the NCEP Verification System.

35.2.4 Accuracy

Called "ACC" in CTS output Table 11.4

Accuracy for a 2x2 contingency table is defined as

$$\text{ACC} = \frac{n_{11} + n_{00}}{T}.$$

That is, it is the proportion of forecasts that were either hits or correct rejections - the fraction that were correct. Accuracy ranges from 0 to 1; a perfect forecast would have an accuracy value of 1. Accuracy should be used with caution, especially for rare events, because it can be strongly influenced by large values of $\mathbf{n_{00}}$.

35.2.5 Frequency Bias

Called "FBIAS" in CTS output Table 11.4

Frequency Bias is the ratio of the total number of forecasts of an event to the total number of observations of the event. It is defined as

$$\text{Bias} = \frac{n_{11} + n_{10}}{n_{11} + n_{01}} = \frac{n_{1\cdot}}{n_1}.$$

A "good" value of Frequency Bias is close to 1; a value greater than 1 indicates the event was forecasted too frequently and a value less than 1 indicates the event was not forecasted frequently enough.

35.2.6 H_RATE

Called "H_RATE" in FHO output Table 11.2

H_RATE is defined as

$$\text{H_RATE} = \frac{n_{11}}{T}.$$

H_RATE is equivalent to the H value computed by the NCEP verification system. H_RATE ranges from 0 to 1; a perfect forecast would have H_RATE = 1.

35.2.7 Probability of Detection (POD)

Called "PODY" in CTS output Table 11.4

POD is defined as

$$\text{POD} = \frac{n_{11}}{n_{11} + n_{01}} = \frac{n_{11}}{n_1}.$$

It is the fraction of events that were correctly forecasted to occur. POD is also known as the hit rate. POD ranges from 0 to 1; a perfect forecast would have POD = 1.

35.2.8 Probability of False Detection (POFD)

Called "POFD" in CTS output Table 11.4

POFD is defined as

$$\text{POFD} = \frac{n_{10}}{n_{10} + n_{00}} = \frac{n_{10}}{n_{.0}}.$$

It is the proportion of non-events that were forecast to be events. POFD is also often called the False Alarm Rate. POFD ranges from 0 to 1; a perfect forecast would have POFD = 0.

35.2.9 Probability of Detection of the Non-Event (PODn)

Called "PODN" in CTS output Table 11.4

PODn is defined as

$$\text{PODN} = \frac{n_{00}}{n_{10} + n_{00}} = \frac{n_{00}}{n_{.0}}.$$

It is the proportion of non-events that were correctly forecasted to be non-events. Note that PODn = 1 - POFD. PODn ranges from 0 to 1. Like POD, a perfect forecast would have PODn = 1.

35.2.10 False Alarm Ratio (FAR)

Called "FAR" in CTS output Table 11.4

FAR is defined as

$$\text{FAR} = \frac{n_{10}}{n_{10} + n_{11}} = \frac{n_{10}}{n_{1.}}.$$

It is the proportion of forecasts of the event occurring for which the event did not occur. FAR ranges from 0 to 1; a perfect forecast would have FAR = 0.

35.2.11 Critical Success Index (CSI)

Called "CSI" in CTS output Table 11.4

CSI is defined as

$$\text{CSI} = \frac{n_{11}}{n_{11} + n_{10} + n_{01}}.$$

It is the ratio of the number of times the event was correctly forecasted to occur to the number of times it was either forecasted or occurred. CSI ignores the "correct rejections" category (i.e., $\mathbf{n_{00}}$). CSI is also known as the Threat Score (TS). CSI can also be written as a nonlinear combination of POD and FAR, and is strongly related to Frequency Bias and the Base Rate.

35.2.12 Gilbert Skill Score (GSS)

Called "GSS" in CTS output Table 11.4

GSS is based on the CSI, corrected for the number of hits that would be expected by chance. In particular,

$$\text{GSS} = \frac{n_{11} - C_1}{n_{11} + n_{10} + n_{01} - C_1},$$

where

$$C = \frac{(n_{11} + n_{10})(n_{11} + n_{01})}{T}.$$

GSS is also known as the Equitable Threat Score (ETS). GSS values range from -1/3 to 1. A no-skill forecast would have GSS = 0; a perfect forecast would have GSS = 1.

35.2.13 Hanssen-Kuipers Discriminant (HK)

Called "HK" in CTS output Table 11.4

HK is defined as

$$\text{HK} = \frac{n_{11}n_{00} - n_{10}n_{01}}{(n_{11} + n_{01})(n_{10} + n_{00})}.$$

More simply, HK = POD − POFD.

HK is also known as the True Skill Statistic (TSS) and less commonly (although perhaps more properly) as the Peirce Skill Score. HK measures the ability of the forecast to discriminate between (or correctly classify) events and non-events. HK values range between -1 and 1. A value of 0 indicates no skill; a perfect forecast would have HK = 1.

35.2.14 Heidke Skill Score (HSS)

Called "HSS" in CTS output Table 11.4 and "HSS" in MCTS output Table 11.7

HSS is a skill score based on Accuracy, where the Accuracy is compared to the number of correct forecasts that would be expected by chance. In particular,

$$\text{HSS} = \frac{n_{11} + n_{00} - C_2}{T - C_2},$$

where

$$C_2 = \frac{(n_{11} + n_{10})(n_{11} + n_{01}) + (n_{01} + n_{00})(n_{10} + n_{00})}{T}.$$

Note that the C_2 value is calculated based on the data fields supplied by the user. Therefore, for C2 to appropriately represent a random forecast, a sufficiently large sized dataset of forecasts and observations would be needed.

HSS can range from minus infinity to 1. A perfect forecast would have HSS = 1.

35.2.15 Heidke Skill Score - Expected Correct (HSS_EC)

Called "HSS_EC" in CTS output Table 11.4 and MCTS output Table 11.7

HSS_EC calculates the HSS as described above, but with a C2 value based on a set expected chance (EC) value. Instead of C2 being calculated by the user's dataset,

$$\text{HSS} = \text{T*EC},$$

where EC is allowed to be prescribed by the user ranging from 0 to 1. By default the EC is set to 1 divided by the number of contingency table categories, e.g. EC is set to 0.33333 for a 3 category (tercile) forecast and 0.5 for a two category (binary) forecast.

HSS_EC can range from minus infinity to 1. A perfect forecast would have HSS_EC = 1.

35.2.16 Odds Ratio (OR)

Called "ODDS" in CTS output Table 11.4

OR measures the ratio of the odds of a forecast of the event being correct to the odds of a forecast of the event being wrong. OR is defined as

$$\text{OR} = \frac{n_{11} \times n_{00}}{n_{10} \times n_{01}} = \frac{(\frac{\text{POD}}{1-\text{POD}})}{(\frac{\text{POFD}}{1-\text{POFD}})}.$$

OR can range from 0 to ∞. A perfect forecast would have a value of OR = infinity. OR is often expressed as the log Odds Ratio or as the Odds Ratio Skill Score (*Stephenson, 2000* (page 499)).

35.2.17 Logarithm of the Odds Ratio (LODDS)

Called "LODDS" in CTS output Table 11.4

LODDS transforms the odds ratio via the logarithm, which tends to normalize the statistic for rare events (*Stephenson, 2000* (page 499)). However, it can take values of $\pm\infty$ when any of the contingency table counts is 0. LODDS is defined as LODDS $= ln(OR)$.

35.2.18 Odds Ratio Skill Score (ORSS)

Called "ORSS" in CTS output Table 11.4

ORSS is a skill score based on the odds ratio. ORSS is defined as

$$\text{ORSS} = \frac{OR - 1}{OR + 1}.$$

ORSS is sometimes also referred to as Yule's Q. (*Stephenson, 2000* (page 499)).

35.2.19 Extreme Dependency Score (EDS)

Called "EDS" in CTS output Table 11.4

The extreme dependency score measures the association between forecast and observed rare events. EDS is defined as

$$\text{EDS} = \frac{2ln(\frac{n_{11}+n_{01}}{T})}{ln(\frac{n_{11}}{T})} - 1.$$

EDS can range from -1 to 1, with 0 representing no skill. A perfect forecast would have a value of EDS = 1. EDS is independent of bias, so should be presented along with the frequency bias statistic (*Stephenson et al., 2008* (page 499)).

35.2.20 Extreme Dependency Index (EDI)

Called "EDI" in CTS output Table 11.4

The extreme dependency index measures the association between forecast and observed rare events. EDI is defined as

$$\text{EDI} = \frac{\log F - \log H}{\log F + \log H},$$

where *H* and *F* are the Hit Rate and False Alarm Rate, respectively.

EDI can range from $-\infty$ to 1, with 0 representing no skill. A perfect forecast would have a value of EDI = 1 (*Ferro and Stephenson, 2011* (page 499)).

35.2.21 Symmetric Extreme Dependency Score (SEDS)

Called "SEDS" in CTS output Table 11.4

The symmetric extreme dependency score measures the association between forecast and observed rare events. SEDS is defined as

$$\text{SEDS} = \frac{2\ln[\frac{(n_{11}+n_{01})(n_{11}+n_{10})}{T^2}]}{\ln(\frac{n_{11}}{T})} - 1.$$

SEDS can range from $-\infty$ to 1, with 0 representing no skill. A perfect forecast would have a value of SEDS = 1 (*Ferro and Stephenson, 2011* (page 499)).

35.2.22 Symmetric Extremal Dependency Index (SEDI)

Called "SEDI" in CTS output Table 11.4

The symmetric extremal dependency index measures the association between forecast and observed rare events. SEDI is defined as

$$\text{SEDI} = \frac{\ln F - \ln H + \ln(1-H) - \ln(1-F)}{\ln F + \ln H + \ln(1-H) + \ln(1-F)},$$

where $H = \frac{n_{11}}{n_{11}+n_{01}}$ and $F = \frac{n_{10}}{n_{00}+n_{10}}$ are the Hit Rate and False Alarm Rate, respectively.

SEDI can range from $-\infty$ to 1, with 0 representing no skill. A perfect forecast would have a value of SEDI = 1. SEDI approaches 1 only as the forecast approaches perfection (*Ferro and Stephenson, 2011* (page 499)).

35.2.23 Bias-Adjusted Gilbert Skill Score (BAGSS)

Called "BAGSS" in CTS output Table 11.4

BAGSS is based on the GSS, but is corrected as much as possible for forecast bias (*Brill and Mesinger, 2009* (page 492)).

35.2.24 Economic Cost Loss Relative Value (ECLV)

Included in ECLV output Table 11.12

The Economic Cost Loss Relative Value (ECLV) applies a weighting to the contingency table counts to determine the relative value of a forecast based on user-specific information. The cost is incurred to protect against an undesirable outcome, whether that outcome occurs or not. No cost is incurred if no protection is undertaken. Then, if the event occurs, the user sustains a loss. If the event does not occur, there is neither a cost nor a loss. The maximum forecast value is achieved when the cost/loss ratio equals the climatological probability. When this occurs, the ECLV is equal to the Hanssen and Kuipers discriminant. The Economic Cost Loss Relative Value is defined differently depending on whether the cost / loss ratio is lower than the base rate or higher. The ECLV is a function of the cost / loss ratio (cl), the hit rate (h), the false alarm rate (f), the miss rate (m), and the base rate (b).

For cost / loss ratio below the base rate, the ECLV is defined as:

$$\text{ECLV} = \frac{(cl * (h + f - 1)) + m}{cl * (b - 1)}.$$

For cost / loss ratio above the base rate, the ECLV is defined as:

$$\text{ECLV} = \frac{(cl * (h + f)) + m - b}{b * (cl - 1)}.$$

35.2.25 Stable Equitable Error in Probability Space (SEEPS)

Included in SEEPS output Table 11.20 and SEEPS_MPR output Table 11.19

The SEEPS scoring matrix (equation 15 from *Rodwell et al, 2010* (page 498)) is:

$$\{S_{vf}^{S}\} = \frac{1}{2}\begin{Bmatrix} 0 & \frac{1}{1-p_1} & \frac{1}{p_3} + \frac{1}{1-p_1} \\ \frac{1}{p_1} & 0 & \frac{1}{p_3} \\ \frac{1}{p_1} + \frac{1}{1-p_3} & \frac{1}{1-p_3} & 0 \end{Bmatrix}$$

In addition, Rodwell et al (2011) note that SEEPS can be written as the mean of two 2-category scores that individually assess the dry/light and light/heavy thresholds (*Rodwell et al., 2011* (page 498)). Each of these scores is like 1 – HK, but written as:

$$\frac{n_{01}}{\text{Expected n}_{.1}} + \frac{n_{10}}{\text{Expected n}_{.0}}$$

where the word expected refers to the mean value deduced from the climatology, rather than the sample mean.

SEEPS scores are expected to lie between 0 and 1, with a perfect forecast having a value of 0. Individual values can be much larger than 1. Results can be presented as a skill score by using the value of 1 – SEEPS.

35.3 MET Verification Measures for Continuous Variables

For continuous variables, many verification measures are based on the forecast error (i.e., **f - o**). However, it also is of interest to investigate characteristics of the forecasts, and the observations, as well as their relationship. These concepts are consistent with the general framework for verification outlined by *Murphy and Winkler, 1987* (page 497). The statistics produced by MET for continuous forecasts represent this philosophy of verification, which focuses on a variety of aspects of performance rather than a single measure.

The verification measures currently evaluated by the Point-Stat tool are defined and described in the subsections below. In these definitions, **f** represents the forecasts, **o** represents the observation, and **n** is the number of forecast-observation pairs.

35.3.1 Mean Forecast

Called "FBAR" in CNT output Table 11.5

Called "FBAR" in SL1L2 output Table 11.13

The sample mean forecast, FBAR, is defined as $\bar{f} = \frac{1}{n}\sum_{i=1}^{n} f_i$.

35.3.2 Mean Observation

Called "OBAR" in CNT output Table 11.5

Called "OBAR" in SL1L2 output Table 11.13

The sample mean observation is defined as $\bar{o} = \frac{1}{n}\sum_{i=1}^{n} o_i$.

35.3.3 Forecast Standard Deviation

Called "FSTDEV" in CNT output Table 11.5

The sample variance of the forecasts is defined as

$$s_f^2 = \frac{1}{T-1}\sum_{i=1}^{T}(f_i - \bar{f})^2.$$

The forecast standard deviation is defined as $s_f = \sqrt{s_f^2}$.

35.3.4 Observation Standard Deviation

Called "OSTDEV" in CNT output Table 11.5

The sample variance of the observations is defined as

$$s_o^2 = \frac{1}{T-1}\sum_{i=1}^{T}(o_i - \bar{o})^2.$$

The observed standard deviation is defined as $s_o = \sqrt{s_o^2}$.

35.3.5 Pearson Correlation Coefficient

Called "PR_CORR" in CNT output Table 11.5

The Pearson correlation coefficient, **r**, measures the strength of linear association between the forecasts and observations. The Pearson correlation coefficient is defined as:

$$r = \frac{\sum_{i=1}^{T}(f_i - \bar{f})(o_i - \bar{o})}{\sqrt{\sum(f_i - \bar{f})^2}\sqrt{\sum(o_i - \bar{o})^2}}$$

r can range between -1 and 1; a value of 1 indicates perfect correlation and a value of -1 indicates perfect negative correlation. A value of 0 indicates that the forecasts and observations are not correlated.

35.3.6 Spearman Rank Correlation Coefficient (ρ_s)

Called "SP_CORR" in CNT Table 11.5

The Spearman rank correlation coefficient (ρ_s) is a robust measure of association that is based on the ranks of the forecast and observed values rather than the actual values. That is, the forecast and observed samples are ordered from smallest to largest and rank values (from 1 to **n**, where **n** is the total number of pairs) are assigned. The pairs of forecast-observed ranks are then used to compute a correlation coefficient, analogous to the Pearson correlation coefficient, **r**.

A simpler formulation of the Spearman-rank correlation is based on differences between the each of the pairs of ranks (denoted as d_i):

$$\rho_s = \frac{6}{n(n^2-1)} \sum_{i=1}^{n} d_i^2$$

Like **r**, the Spearman rank correlation coefficient ranges between -1 and 1; a value of 1 indicates perfect correlation and a value of -1 indicates perfect negative correlation. A value of 0 indicates that the forecasts and observations are not correlated.

35.3.7 Kendall's Tau Statistic (τ)

Called "KT_CORR" in CNT output Table 11.5

Kendall's Tau statistic (τ) is a robust measure of the level of association between the forecast and observation pairs. It is defined as

$$\tau = \frac{N_C - N_D}{n(n-1)/2}$$

where N_C is the number of "concordant" pairs and N_D is the number of "discordant" pairs. Concordant pairs are identified by comparing each pair with all other pairs in the sample; this can be done most easily by ordering all of the (f_i, o_i) pairs according to f_i, in which case the o_i values won't necessarily be in order. The number of concordant matches of a particular pair with other pairs is computed by counting the number of pairs (with larger values) for which the value of o_i for the current pair is exceeded (that is, pairs for which the values of **f** and **o** are both larger than the value for the current pair). Once this is done, N_C is computed by summing the counts for all pairs. The total number of possible pairs is N_C; thus, the number of discordant pairs is N_D.

Like **r** and ρ_s, Kendall's Tau (τ) ranges between -1 and 1; a value of 1 indicates perfect association (concordance) and a value of -1 indicates perfect negative association. A value of 0 indicates that the forecasts and observations are not associated.

35.3.8 Mean Error (ME)

Called "ME" in CNT output Table 11.5 Called "ME_OERR", "ME_GE_OBS", and "ME_LT_OBS" in ECNT output Table 14.2

The Mean Error, ME, is a measure of overall bias for continuous variables; in particular ME = Bias. It is defined as

$$\text{ME} = \frac{1}{n}\sum_{i=1}^{n}(f_i - o_i) = \bar{f} - \bar{o}.$$

A perfect forecast has ME = 0.

35.3.9 Mean Error Squared (ME2)

Called "ME2" in CNT output Table 11.5

The Mean Error Squared, ME2, is provided to give a complete breakdown of MSE in terms of squared Bias plus estimated variance of the error, as detailed below in the section on BCMSE. It is defined as $\text{ME2} = \text{ME}^2$.

A perfect forecast has ME2 = 0.

35.3.10 Multiplicative Bias

Called "MBIAS" in CNT output Table 11.5

Multiplicative bias is simply the ratio of the means of the forecasts and the observations: $\text{MBIAS} = \bar{f}/\bar{o}$

35.3.11 Mean-Squared Error (MSE)

Called "MSE" in CNT output Table 11.5

MSE measures the average squared error of the forecasts. Specifically, $\text{MSE} = \frac{1}{n}\sum(f_i - o_i)^2$.

35.3.12 Root-Mean-Squared Error (RMSE)

Called "RMSE" in CNT output Table 11.5 Called "RMSE" and "RMSE_OERR" in ECNT output Table 14.2

RMSE is simply the square root of the MSE, $\text{RMSE} = \sqrt{\text{MSE}}$.

35.3.13 Scatter Index (SI)

Called "SI" in CNT output Table 11.5

SI is the ratio of the root mean squared error to the average observation value, SI = RMSE/OBAR.

Smaller values of SI indicate better agreement between the model and observations (less scatter on scatter plot).

35.3.14 Standard Deviation of the Error

Called "ESTDEV" in CNT output Table 11.5

35.3.15 Bias-Corrected MSE

Called "BCMSE" in CNT output Table 11.5

MSE and RMSE are strongly impacted by large errors. They also are strongly impacted by large bias (ME) values. MSE and RMSE can range from 0 to infinity. A perfect forecast would have MSE = RMSE = 0.

MSE can be re-written as MSE = $(\bar{f} - \bar{o})^2 + s_f^2 + s_o^2 - 2s_f s_o r_{fo}$, where $\bar{f} - \bar{o}$ = ME and $s_f^2 + s_o^2 - 2s_f s_o r_{fo}$ is the estimated variance of the error, s_{fo}^2. Thus, MSE = $\text{ME}^2 + s_{f-o}^2$. To understand the behavior of MSE, it is important to examine both of the terms of MSE, rather than examining MSE alone. Moreover, MSE can be strongly influenced by ME, as shown by this decomposition.

The standard deviation of the error, s_{f-o}, is $s_{f-o} = \sqrt{s_{f-o}^2} = \sqrt{s_f^2 + s_o^2 - 2s_f s_o r_{fo}}$.

Note that the square of the standard deviation of the error (ESTDEV2) is sometimes called the "Bias-corrected MSE" (BCMSE) because it removes the effect of overall bias from the forecast-observation squared differences.

35.3.16 Mean Absolute Error (MAE)

Called "MAE" in CNT output Table 11.5 Called "MAE" and "MAE_OERR" in ECNT output Table 14.2

The Mean Absolute Error (MAE) is defined as MAE = $\frac{1}{n}\sum|f_i - o_i|$.

MAE is less influenced by large errors and also does not depend on the mean error. A perfect forecast would have MAE = 0.

35.3.17 InterQuartile Range of the Errors (IQR)

Called "IQR" in CNT output Table 11.5

The InterQuartile Range of the Errors (IQR) is the difference between the 75th and 25th percentiles of the errors. It is defined as IQR = $p_{75}(f_i - o_i) - p_{25}(f_i - o_i)$.

IQR is another estimate of spread, similar to standard error, but is less influenced by large errors and also does not depend on the mean error. A perfect forecast would have IQR = 0.

35.3.18 Median Absolute Deviation (MAD)

Called "MAD" in CNT output Table 11.5

The Median Absolute Deviation (MAD) is defined as MAD = median$|f_i - o_i|$.

MAD is an estimate of spread, similar to standard error, but is less influenced by large errors and also does not depend on the mean error. A perfect forecast would have MAD = 0.

35.3.19 Mean Squared Error Skill Score

Called "MSESS" in CNT output Table 11.5

The Mean Squared Error Skill Score is one minus the ratio of the forecast MSE to some reference MSE, usually climatology. It is sometimes referred to as Murphy's Mean Squared Error Skill Score.

$$\text{MSESS} = 1 - \frac{\text{MSE}_f}{\text{MSE}_r}$$

35.3.20 Root-Mean-Squared Forecast Anomaly

Called "RMSFA" in CNT output Table 11.5

RMSFA is the square root of the average squared forecast anomaly. Specifically, RMSFA $= \sqrt{\frac{1}{n}\sum(f_i - c_i)^2}$.

35.3.21 Root-Mean-Squared Observation Anomaly

Called "RMSOA" in CNT output Table 11.5

RMSOA is the square root of the average squared observation anomaly. Specifically, RMSOA $= \sqrt{\frac{1}{n}\sum(o_i - c_i)^2}$.

35.3.22 Percentiles of the Errors

Called "E10", "E25", "E50", "E75", "E90" in CNT output Table 11.5

Percentiles of the errors provide more information about the distribution of errors than can be obtained from the mean and standard deviations of the errors. Percentiles are computed by ordering the errors from smallest to largest and computing the rank location of each percentile in the ordering, and matching the rank to the actual value. Percentiles can also be used to create box plots of the errors. In MET, the 0.10th, 0.25th, 0.50th, 0.75th, and 0.90th quantile values of the errors are computed.

35.3.23 Anomaly Correlation Coefficient

Called "ANOM_CORR" and "ANOM_CORR_UNCNTR" for centered and uncentered versions in CNT output Table 11.5

The anomaly correlation coefficient is equivalent to the Pearson correlation coefficient, except that both the forecasts and observations are first adjusted by subtracting their corresponding climatology value. The anomaly is the difference between the individual forecast or observation and the typical situation, as measured by a forecast climatology (c_f) and observation climatology (c_o). It measures the strength of linear association between the forecast anomalies and observed anomalies. The anomaly correlation coefficient is defined as:

$$\text{Anomaly Correlation} = \frac{\sum(f_i - c_{f_i})(o_i - c_{oi})}{\sqrt{\sum(f_i - c_{f_i})^2}\sqrt{\sum(o_i - c_{oi})^2}}.$$

The centered anomaly correlation coefficient (ANOM_CORR) which includes the mean error is defined as:

$$\text{ANOM_CORR} = \frac{\overline{[(f - c_f) - \overline{(f - c_f)}][(o - c_o) - \overline{(o - c_o)}]}}{\sqrt{\overline{((f - c_f) - \overline{(f - c_f)})^2((o - c_o) - \overline{(o - c_o)})^2}}}$$

The uncentered anomaly correlation coefficient (ANOM_CORR_UNCNTR) which does not include the mean errors is defined as:

$$\text{Anomaly Correlation Raw} = \frac{\overline{(f - c_f)(o - c_o)}}{\sqrt{\overline{(f - c_f)^2(o - c_o)^2}}}$$

Anomaly correlation can range between -1 and 1; a value of 1 indicates perfect correlation and a value of -1 indicates perfect negative correlation. A value of 0 indicates that the forecast and observed anomalies are not correlated.

35.3.24 Partial Sums Lines (SL1L2, SAL1L2, VL1L2, VAL1L2)

Table 11.13, Table 11.14, Table 11.15, and Table 11.16

The SL1L2, SAL1L2, VL1L2, and VAL1L2 line types are used to store data summaries (e.g. partial sums) that can later be accumulated into verification statistics. These are divided according to scalar or vector summaries (S or V). The climate anomaly values (A) can be stored in place of the actuals, which is just a re-centering of the values around the climatological average. L1 and L2 refer to the L1 and L2 norms, the distance metrics commonly referred to as the "city block" and "Euclidean" distances. The city block is the absolute value of a distance while the Euclidean distance is the square root of the squared distance.

The partial sums can be accumulated over individual cases to produce statistics for a longer period without any loss of information because these sums are *sufficient* for resulting statistics such as RMSE, bias, correlation coefficient, and MAE (*Mood et al., 1974* (page 497)). Thus, the individual errors need not be stored, all of the information relevant to calculation of statistics are contained in the sums. As an example, the sum of all data points and the sum of all squared data points (or equivalently, the sample mean and sample variance) are *jointly sufficient* for estimates of the Gaussian distribution mean and variance.

Minimally sufficient statistics are those that condense the data most, with no loss of information. Statistics based on L1 and L2 norms allow for good compression of information. Statistics based on other norms, such

as order statistics, do not result in good compression of information. For this reason, statistics such as RMSE are often preferred to statistics such as the median absolute deviation. The partial sums are not sufficient for order statistics, such as the median or quartiles.

35.3.25 Scalar L1 and L2 Values

Called "FBAR", "OBAR", "FOBAR", "FFBAR", "OOBAR", and "MAE" in SL1L2 output Table 11.13

These statistics are simply the 1st and 2nd moments of the forecasts, observations and errors:

$$\text{FBAR} = \text{Mean}(f) = \frac{1}{n}\sum_{i=1}^{n} f_i$$

$$\text{OBAR} = \text{Mean}(o) = \frac{1}{n}\sum_{i=1}^{n} o_i$$

$$\text{FOBAR} = \text{Mean}(fo) = \frac{1}{n}\sum_{i=1}^{n} f_i o_i$$

$$\text{FFBAR} = \text{Mean}(f^2) = \frac{1}{n}\sum_{i=1}^{n} f_i^2$$

$$\text{OOBAR} = \text{Mean}(o^2) = \frac{1}{n}\sum_{i=1}^{n} o_i^2$$

$$\text{MAE} = \text{Mean}(|f-o|) = \frac{1}{n}\sum_{i=1}^{n} |f_i - o_i|$$

Some of the other statistics for continuous forecasts (e.g., RMSE) can be derived from these moments.

35.3.26 Scalar Anomaly L1 and L2 Values

Called "FABAR", "OABAR", "FOABAR", "FFABAR", "OOABAR", and "MAE" in SAL1L2 output Table 11.14

Computation of these statistics requires climatological values, where c_f is the forecast climatology value and c_o is the observation climatology value. These statistics are the 1st and 2nd moments of the scalar anomalies.

The moments are defined as:

$$\text{FABAR} = \text{Mean}(f - c_f) = \frac{1}{n}\sum_{i=1}^{n}(f_i - c_{f_i})$$

$$\text{OABAR} = \text{Mean}(o - c_o) = \frac{1}{n}\sum_{i=1}^{n}(o_i - c_{oi})$$

$$\text{FOABAR} = \text{Mean}[(f - c_f)(o - c_o)] = \frac{1}{n}\sum_{i=1}^{n}(f_i - c_{f_i})(o_i - c_{oi})$$

$$\text{FFABAR} = \text{Mean}[(f - c_f)^2] = \frac{1}{n}\sum_{i=1}^{n}(f_i - c_{f_i})^2$$

$$\text{OOABAR} = \text{Mean}[(o - c_o)^2] = \frac{1}{n}\sum_{i=1}^{n}(o_i - c_{oi})^2$$

$$\text{MAE} = \text{Mean}(|(f - c_f) - (o - c_o)|) = \frac{1}{n}\sum_{i=1}^{n}|(f_i - c_{f_i}) - (o_i - c_{oi})|$$

35.3.27 Vector L1 and L2 Values

Called "UFBAR", "VFBAR", "UOBAR", "VOBAR", "UVFOBAR", "UVFFBAR", and "UVOOBAR" in VL1L2 output Table 11.15

These statistics are the moments for wind vector values, where u is the E-W wind component and v is the N-S wind component (u_f is the forecast E-W wind component; u_o is the observed E-W wind component; v_f is the forecast N-S wind component; and v_o is the observed N-S wind component). The following measures are computed:

$$\text{UFBAR} = \text{Mean}(u_f) = \frac{1}{n}\sum_{i=1}^{n}u_{fi}$$

$$\text{VFBAR} = \text{Mean}(v_f) = \frac{1}{n}\sum_{i=1}^{n}v_{fi}$$

$$\text{UOBAR} = \text{Mean}(u_o) = \frac{1}{n}\sum_{i=1}^{n}u_{oi}$$

$$\text{VOBAR} = \text{Mean}(v_o) = \frac{1}{n}\sum_{i=1}^{n}v_{oi}$$

$$\text{UVFOBAR} = \text{Mean}(u_f u_o + v_f v_o) = \frac{1}{n}\sum_{i=1}^{n}(u_{fi}u_{oi} + v_{fi}v_{oi})$$

$$\text{UVFFBAR} = \text{Mean}(u_f^2 + v_f^2) = \frac{1}{n}\sum_{i=1}^{n}(u_{fi}^2 + v_{fi}^2)$$

$$\text{UVOOBAR} = \text{Mean}(u_o^2 + v_o^2) = \frac{1}{n}\sum_{i=1}^{n}(u_{oi}^2 + v_{oi}^2)$$

35.3.28 Vector Anomaly L1 and L2 Values

Called "UFABAR", "VFABAR", "UOABAR", "VOABAR", "UVFOABAR", "UVFFABAR", and "UVOOABAR" in VAL1L2 output Table 11.16

These statistics require climatological values for the wind vector components, where u_{cf} and v_{cf} are the forecast climatology vectors and u_{co} and v_{co} are the observation climatology vectors. The measures are defined below:

$$\text{UFABAR} = \text{Mean}(u_f - u_{cf}) = \frac{1}{n}\sum_{i=1}^{n}(u_{f_i} - u_{cf_i})$$

$$\text{VFBAR} = \text{Mean}(v_f - v_{cf}) = \frac{1}{n}\sum_{i=1}^{n}(v_{f_i} - v_{cf_i})$$

$$\text{UOABAR} = \text{Mean}(u_o - u_{co}) = \frac{1}{n}\sum_{i=1}^{n}(u_{oi} - u_{coi})$$

$$\text{VOABAR} = \text{Mean}(v_o - v_{co}) = \frac{1}{n}\sum_{i=1}^{n}(v_{oi} - v_{coi})$$

$$\begin{aligned}\text{UVFOABAR} &= \text{Mean}[(u_f - u_{cf})(u_o - u_{co}) + (v_f - v_{cf})(v_o - v_{co})] \\ &= \frac{1}{n}\sum_{i=1}^{n}(u_{f_i} - u_{cf_i}) + (u_{oi} - u_{coi}) + (v_{f_i} - v_{cf_i})(v_{oi} - v_{coi})\end{aligned}$$

$$\begin{aligned}\text{UVFFABAR} &= \text{Mean}[(u_f - u_{cf})^2 + (v_f - v_{cf})^2] \\ &= \frac{1}{n}\sum_{i=1}^{n}((u_{f_i} - u_{cf_i})^2 + (v_{f_i} - v_{cf_i})^2)\end{aligned}$$

$$\begin{aligned}\text{UVOOABAR} &= \text{Mean}[(u_o - u_{co})^2 + (v_o - v_{co})^2] \\ &= \frac{1}{n}\sum_{i=1}^{n}((u_{oi} - u_{coi})^2 + (v_{oi} - v_{coi})^2)\end{aligned}$$

35.3.29 Gradient Values

Called "TOTAL", "FGBAR", "OGBAR", "MGBAR", "EGBAR", "S1", "S1_OG", "FGOG_RATIO", "FGMAG", "OGMAG", "MAG_RMSE", and "LAPLACE_RMSE" in GRAD output Table 13.5

These statistics are only computed by the Grid-Stat tool and require vectors. Here ∇ is the gradient operator, which in this applications signifies the difference between adjacent grid points in both the grid-x and grid-y directions. TOTAL is the count of grid locations used in the calculations. The remaining measures are

defined below:

$$\text{FGBAR} = \text{Mean}|\nabla f| = \frac{1}{n}\sum_{i=1}^{n}|\nabla f_i|$$

$$\text{OGBAR} = \text{Mean}|\nabla o| = \frac{1}{n}\sum_{i=1}^{n}|\nabla o_i|$$

$$\text{MGBAR} = \text{Max(FGBAR, OGBAR)}$$

$$\text{EGBAR} = \text{Mean}|\nabla f - \nabla o| = \frac{1}{n}\sum_{i=1}^{n}|\nabla f_i - \nabla o_i|$$

$$\text{S1} = 100\frac{\sum_{i=1}^{n}(w_i(e_g))}{\sum_{i=1}^{n}(w_i(G_L))_i},$$

where the weights are applied at each grid location, with values assigned according to the weight option specified in the configuration file. The components of the $S1$ equation are as follows:

$$e_g = (|\frac{\delta}{\delta x}(f-o)| + |\frac{\delta}{\delta y}(f-o)|)$$

$$G_L = \max\left(|\frac{\delta f}{\delta x}|, |\frac{\delta o}{\delta x}|\right) + \max\left(|\frac{\delta f}{\delta y}|, |\frac{\delta o}{\delta y}|\right)$$

$$\text{S1_OG} = \frac{\text{EGBAR}}{\text{OGBAR}}$$

$$\text{FGOG_RATIO} = \frac{\text{FGBAR}}{\text{OGBAR}}$$

The following statistics are computed using the magnitude of the vectors formed by the forecast and observation gradients in the grid-x and grid-y directions.

$$\text{FGMAG} = \text{Mean}(||\nabla f||) = \frac{1}{n}\sum_{i=1}^{n}\sqrt{{\nabla f_{x_i}}^2 + {\nabla f_{y_i}}^2}$$

$$\text{OGMAG} = \text{Mean}(||\nabla o||) = \frac{1}{n}\sum_{i=1}^{n}\sqrt{{\nabla o_{x_i}}^2 + {\nabla o_{y_i}}^2}$$

$$\text{MAG_RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(||\nabla f_i|| - ||\nabla o_i||\right)^2}$$

Laplace RMSE is very similar to gradient RMSE, but instead of taking the magnitude of the gradient vector at each point, we compute the divergence of the gradient.

$$\text{LAPLACE_RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left((\nabla f_{xi} + \nabla f_{y_i}) - (\nabla o_{xi} + \nabla o_{y_i})\right)^2}$$

35.4 MET Verification Measures for Probabilistic Forecasts

The results of the probabilistic verification methods that are included in the Point-Stat, Grid-Stat, and Stat-Analysis tools are summarized using a variety of measures. MET treats probabilistic forecasts as categorical, divided into bins by user-defined thresholds between zero and one. For the categorical measures, if a forecast probability is specified in a formula, the midpoint value of the bin is used. These measures include the Brier Score (BS) with confidence bounds (*Bradley, 2008* (page 492)); the joint distribution, calibration-refinement, likelihood-base rate (*Wilks, 2011* (page 500)); and receiver operating characteristic information. Using these statistics, reliability and discrimination diagrams can be produced.

The verification statistics for probabilistic forecasts of dichotomous variables are formulated using a contingency table such as the one shown in Table 35.3. In this table f represents the forecasts and o represents the observations; the two possible forecast and observation values are represented by the values 0 and 1. The values in Table 35.3 are counts of the number of occurrences of all possible combinations of forecasts and observations.

Table 35.3: 2x2 contingency table in terms of counts. The $\mathbf{n_{ij}}$ values in the table represent the counts in each forecast-observation category, where **i** represents the forecast and **j** represents the observations. The "." symbols in the total cells represent sums across categories.

Forecast	Observation		Total
	o = 1 (e.g., "Yes")	**o = 0** (e.g., "No")	
p_1 = midpoint of (0 and threshold1)	n_{11}	n_{10}	$n_{1.} = n_{11} + n_{10}$
p_2 = midpoint of (threshold1 and threshold2)	n_{21}	n_{20}	$n_{2.} = n_{21} + n_{20}$
...	...	...	...
p_j = midpoint of (threshold **i** and 1)	**n**	n_{i0}	$n_j = n_{j1} + n_{j0}$
Total	$n_{.1} = \sum n_{i1}$	$n_{.0} = \sum n_{i0}$	$\mathbf{T} = \sum n_i$

35.4.1 Reliability

Called "RELIABILITY" in PSTD output Table 11.9

A component of the Brier score. Reliability measures the average difference between forecast probability and average observed frequency. Ideally, this measure should be zero as larger numbers indicate larger differences. For example, on occasions when rain is forecast with 50% probability, it should actually rain half the time.

$$\text{Reliability} = \frac{1}{T} \sum n_i (p_i - \bar{o}_i)^2$$

35.4.2 Resolution

Called "RESOLUTION" in PSTD output Table 11.9

A component of the Brier score that measures how well forecasts divide events into subsets with different outcomes. Larger values of resolution are best since it is desirable for event frequencies in the subsets to be different than the overall event frequency.

$$\text{Resolution} = \frac{1}{T} n_{i.}(\bar{o}_i - \bar{o})^2$$

35.4.3 Uncertainty

Called "UNCERTAINTY" in PSTD output Table 11.9

A component of the Brier score. For probabilistic forecasts, uncertainty is a function only of the frequency of the event. It does not depend on the forecasts, thus there is no ideal or better value. Note that uncertainty is equivalent to the variance of the event occurrence.

$$\text{Uncertainty} = \frac{n_{.1}}{T}(1 - \frac{n_{.1}}{T})$$

35.4.4 Brier Score

Called "BRIER" in PSTD output Table 11.9

The Brier score is the mean squared probability error. In MET, the Brier Score (BS) is calculated from the **nx2** contingency table via the following equation:

$$\text{BS} = \frac{1}{T} \sum_{i=1}^{K} [n_{i1}(1 - p_i)^2 + n_{i0} p_i^2]$$

The equation you will most often see in references uses the individual probability forecasts (ρ_i) and the corresponding observations (o_i), and is given as $\text{BS} = \frac{1}{T} \sum (p_i - o_i)^2$. This equation is equivalent when the midpoints of the binned probability values are used as the p_i .

BS can be partitioned into three terms: (1) reliability, (2) resolution, and (3) uncertainty (*Murphy, 1987* (page 497)).

$$\text{BS} = \frac{1}{T} \sum_i (p_i - o_i)^2 = \frac{1}{T} \sum n_{i.}(p_i - \bar{o}_i)^2 - \frac{1}{T} \sum n_{i.}(\bar{o}_i - \bar{o})^2 + \bar{o}(1 - \bar{o})$$

This score is sensitive to the base rate or climatological frequency of the event. Forecasts of rare events can have a good BS without having any actual skill. Since Brier score is a measure of error, smaller values are better.

35.4.5 Brier Skill Score (BSS)

Called "BSS" and "BSS_SMPL" in PSTD output Table 11.9

BSS is a skill score based on the Brier Scores of the forecast and a reference forecast, such as climatology. BSS is defined as

$$\text{BSS} = 1 - \frac{\text{BS}_{fcst}}{\text{BS}_{ref}}.$$

BSS is computed using the climatology specified in the configuration file while BSS_SMPL is computed using the sample climatology of the current set of observations.

35.4.6 OY_TP - Observed Yes Total Proportion

Called "OY_TP" in PJC output Table 11.10

This is the cell probability for row **i**, column **j=1** (observed event), a part of the joint distribution (*Wilks, 2011* (page 500)). Along with ON_TP, this set of measures provides information about the joint distribution of forecasts and events. There are no ideal or better values.

$$\text{OYTP}(i) = \frac{n_{i1}}{T} = \text{probability}(o_{i1})$$

35.4.7 ON_TP - Observed No Total Proportion

Called "ON_TP" in PJC output Table 11.10

This is the cell probability for row **i**, column **j=0** (observed non-event), a part of the joint distribution (*Wilks, 2011* (page 500)). Along with OY_TP, this set of measures provides information about the joint distribution of forecasts and events. There are no ideal or better values.

$$\text{ONTP}(i) = \frac{n_{i0}}{T} = \text{probability}(o_{i0})$$

35.4.8 Calibration

Called "CALIBRATION" in PJC output Table 11.10

Calibration is the conditional probability of an event given each probability forecast category (i.e. each row in the **nx2** contingency table). This set of measures is paired with refinement in the calibration-refinement factorization discussed in *Wilks, 2011* (page 500). A well-calibrated forecast will have calibration values that are near the forecast probability. For example, a 50% probability of precipitation should ideally have a calibration value of 0.5. If the calibration value is higher, then the probability has been underestimated, and vice versa.

$$\text{Calibration}(i) = \frac{n_{i1}}{n_{1.}} = \text{probability}(o_1|p_i)$$

35.4.9 Refinement

Called "REFINEMENT" in PJC output Table 11.10

The relative frequency associated with each forecast probability, sometimes called the marginal distribution or row probability. This measure ignores the event outcome, and simply provides information about the frequency of forecasts for each probability category. This set of measures is paired with the calibration measures in the calibration-refinement factorization discussed by *Wilks, 2011* (page 500).

$$\text{Refinement}(i) = \frac{n_{i.}}{T} = \text{probability}(p_i)$$

35.4.10 Likelihood

Called "LIKELIHOOD" in PJC output Table 11.10

Likelihood is the conditional probability for each forecast category (row) given an event and a component of the likelihood-base rate factorization; see *Wilks, 2011* (page 500) for details. This set of measures considers the distribution of forecasts for only the cases when events occur. Thus, as the forecast probability increases, so should the likelihood. For example, 10% probability of precipitation forecasts should have a much smaller likelihood value than 90% probability of precipitation forecasts.

$$\text{Likelihood}(i) = \frac{n_{i1}}{n_{.1}} = \text{probability}(p_i|o_1)$$

Likelihood values are also used to create "discrimination" plots that compare the distribution of forecast values for events to the distribution of forecast values for non-events. These plots show how well the forecasts categorize events and non-events. The distribution of forecast values for non-events can be derived from the POFD values computed by MET for the user-specified thresholds.

35.4.11 Base Rate

Called "BASER" in PJC output Table 11.10

This is the probability of an event for each forecast category p_i (row), i.e. the conditional base rate. This set of measures is paired with likelihood in the likelihood-base rate factorization, see *Wilks, 2011* (page 500) for further information. This measure is calculated for each row of the contingency table. Ideally, the event should become more frequent as the probability forecast increases.

$$\text{Base Rate}(i) = \frac{n_{i1}}{n_{i.}} = \text{probability}(o_{i1})$$

35.4.12 Reliability Diagram

The reliability diagram is a plot of the observed frequency of events versus the forecast probability of those events, with the range of forecast probabilities divided into categories.

The ideal forecast (i.e., one with perfect reliability) has conditional observed probabilities that are equivalent to the forecast probability, on average. On a reliability plot, this equivalence is represented by the one-to-one line (the solid line in the figure below). So, better forecasts are closer to the diagonal line and worse ones are

farther away. The distance of each point from the diagonal gives the conditional bias. Points that lie below the diagonal line indicate over-forecasting; in other words, the forecast probabilities are too large. The forecast probabilities are too low when the points lie above the line. The reliability diagram is conditioned on the forecasts so it is often used in combination with the ROC, which is conditioned on the observations, to provide a "complete" representation of the performance of probabilistic forecasts.

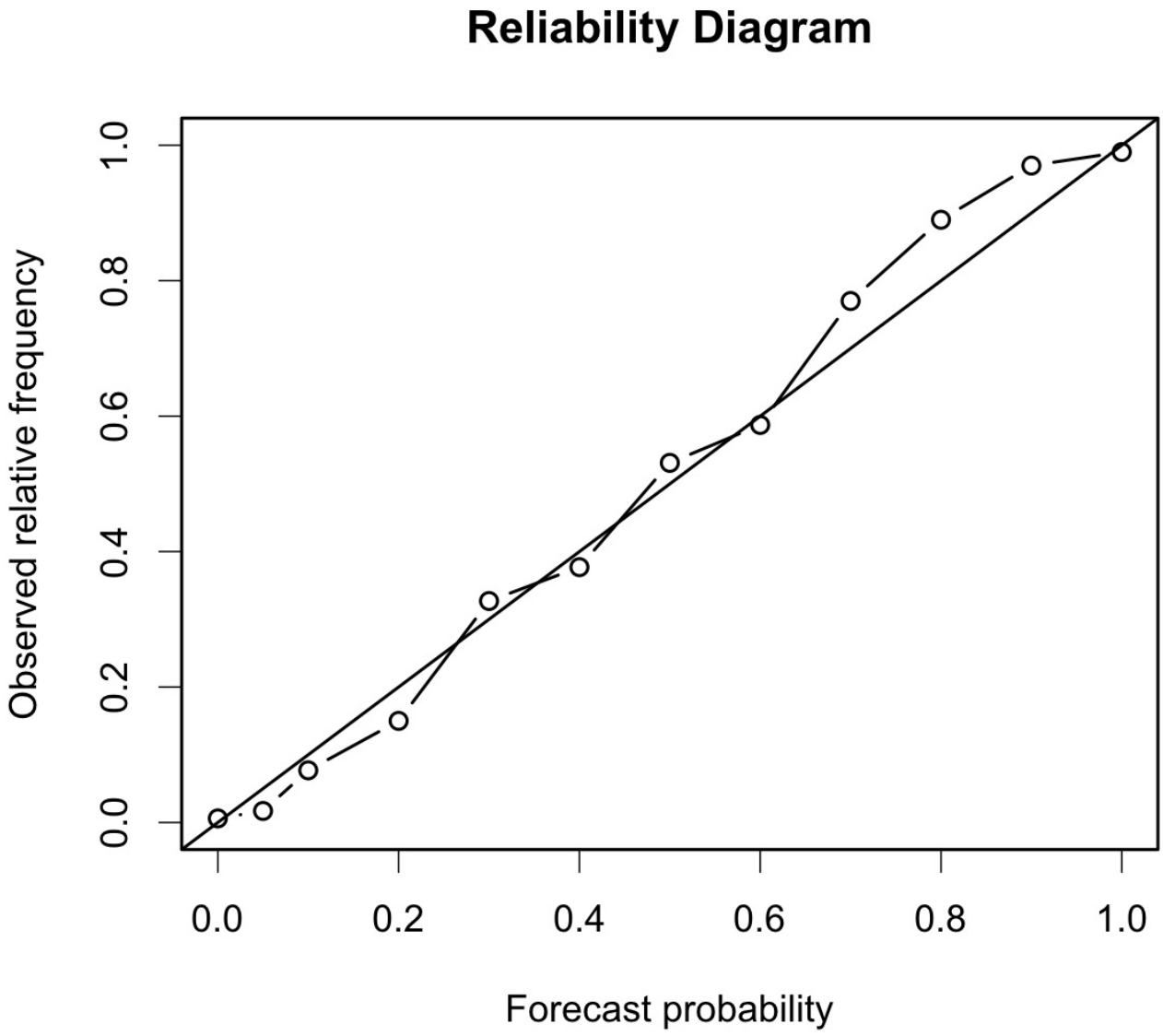

Figure 35.1: Example of Reliability Diagram

35.4.13 Receiver Operating Characteristic

MET produces hit rate (POD) and false alarm rate (POFD) values for each user-specified threshold. This information can be used to create a scatter plot of POFD vs. POD. When the points are connected, the plot is generally referred to as the receiver operating characteristic (ROC) curve (also called the "relative operating characteristic" curve). See the area under the ROC curve (AUC) entry for related information.

A ROC plot is shown for an example set of forecasts, with a solid line connecting the points for six user-specified thresholds (0.25, 0.35, 0.55, 0.65, 0.75, 0.85). The diagonal dashed line indicates no skill while the dash-dot line shows the ROC for a perfect forecast.

A ROC curve shows how well the forecast discriminates between two outcomes, so it is a measure of resolution. The ROC is invariant to linear transformations of the forecast, and is thus unaffected by bias. An unbiased (i.e., well-calibrated) forecast can have the same ROC as a biased forecast, though most would agree that an unbiased forecast is "better". Since the ROC is conditioned on the observations, it is often paired with the reliability diagram, which is conditioned on the forecasts.

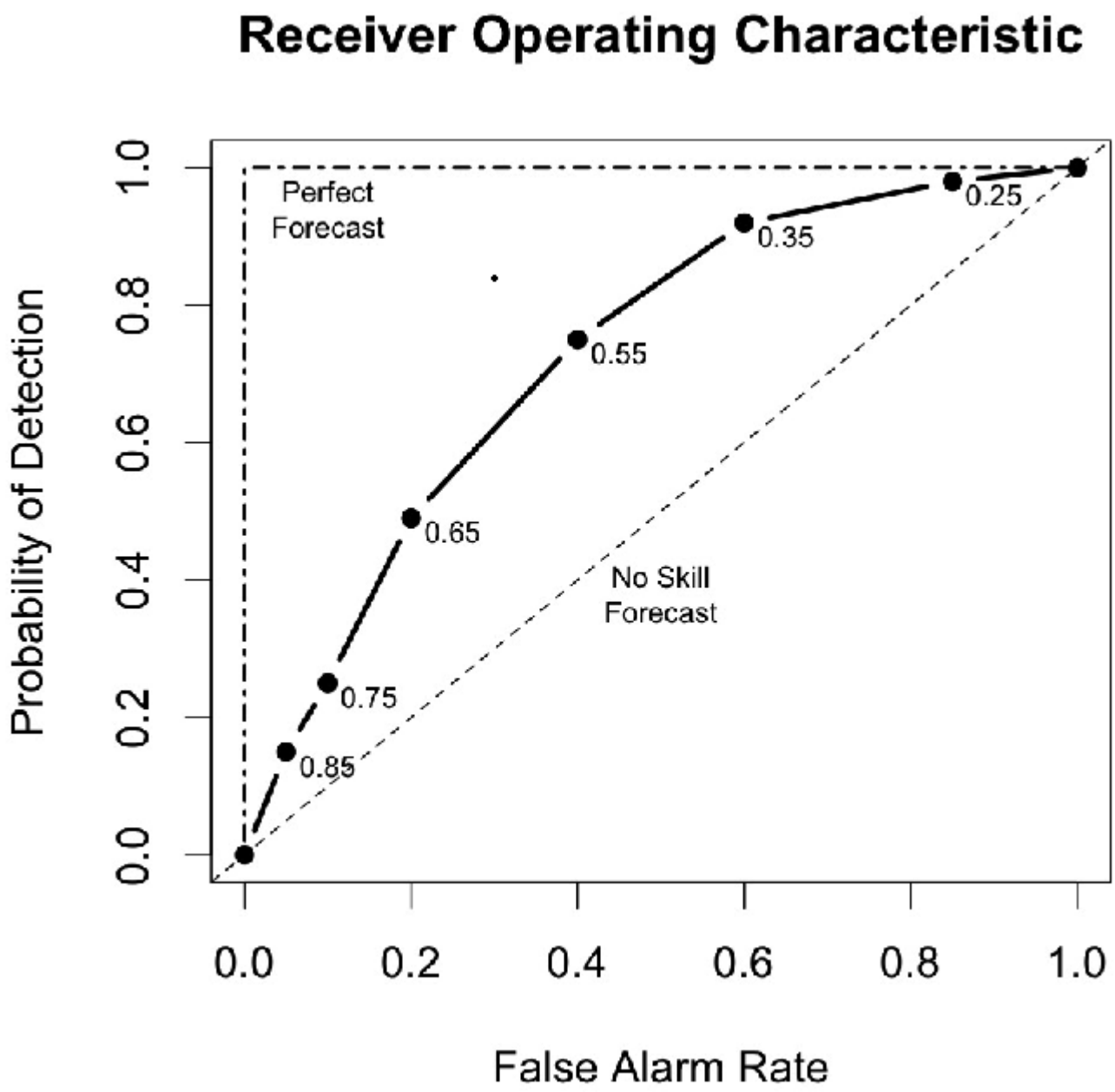

Figure 35.2: Example of ROC Curve

35.4.14 Area Under the ROC Curve (AUC)

Called "ROC_AUC" in PSTD output Table 11.9

The area under the receiver operating characteristic (ROC) curve is often used as a single summary measure. A larger AUC is better. A perfect forecast has AUC=1. Though the minimum value is 0, an AUC of 0.5 indicates no skill.

The area under the curve can be estimated in a variety of ways. In MET, the simplest trapezoid method is used to calculate the area. AUC is calculated from the series of hit rate (POD) and false alarm rate (POFD)

values (see the ROC entry below) for each user-specified threshold.

$$\text{AUC} = \frac{1}{2} \sum_{i=1}^{Nthresh} (\text{POD}_{i+1} + \text{POD}_i)(\text{POFD}_{i+1} - \text{POFD}_i)$$

35.5 MET Verification Measures for Ensemble Forecasts

35.5.1 RPS

Called "RPS" in RPS output Table 14.2

While the above probabilistic verification measures utilize dichotomous observations, the Ranked Probability Score (RPS, *Epstein, 1969* (page 494), *Murphy, 1969* (page 497)) is the only probabilistic verification measure for discrete multiple-category events available in MET. It is assumed that the categories are ordinal as nominal categorical variables can be collapsed into sequences of binary predictands, which can in turn be evaluated with the above measures for dichotomous variables (*Wilks, 2011* (page 500)). The RPS is the multi-category extension of the Brier score (*Tödter and Ahrens, 2012* (page 499)), and is a proper score (*Mason, 2008* (page 497)).

Let J be the number of categories, then both the forecast, f $= (f_1, \ldots, f_J)$, and observation, o $= (o_1, \ldots, o_J)$, are length-J vectors, where the components of f include the probabilities forecast for each category 1,...,J and o contains 1 in the category that is realized and zero everywhere else. The cumulative forecasts, F_m, and observations, O_m, are defined to be:

$F_m = \sum_{j=1}^{m}(f_j)$ and $O_m = \sum_{j=1}^{m}(o_j), m = 1, \ldots, J.$

To clarify, $F_1 = f_1$ is the first component of F_m, $F_2 = f_1 + f_2$, etc., and $F_J = 1$. Similarly, if $o_j = 1$ and $i < j$, then $O_i = 0$ and when $i >= j$, $O_i = 1$, and of course, $O_J = 1$. Finally, the RPS is defined to be:

$$\text{RPS} = \sum_{m=1}^{J}(F_m - O_m)^2 = \sum_{m=1}^{J} BS_m,$$

where BS_m is the Brier score for the m-th category (*Tödter and Ahrens, 2012* (page 499)). Subsequently, the RPS lends itself to a decomposition into reliability, resolution and uncertainty components, noting that each component is aggregated over the different categories; these are written to the columns named "RPS_REL", "RPS_RES" and "RPS_UNC" in RPS output Table 14.2.

35.5.2 CRPS

Called "CRPS", "CRPSCL", "CRPS_EMP", "CRPS_EMP_FAIR" and "CRPSCL_EMP" in ECNT output Table 14.2

The continuous ranked probability score (CRPS) is the integral, over all possible thresholds, of the Brier scores (*Gneiting et al., 2004* (page 495)). In MET, the CRPS is calculated two ways: using a normal distribution fit to the ensemble forecasts (CRPS and CRPSCL), and using the empirical ensemble distribution (CRPS_EMP and CRPSCL_EMP). The empirical ensemble CRPS can be adjusted (bias corrected) by subtracting 1/(2*m) times the mean absolute difference of the ensemble members, where m is the ensemble size. This is reported as a separate statistic called CRPS_EMP_FAIR. In some cases, use of other distributions would be better.

WARNING: The normal distribution is probably a good fit for temperature and pressure, and possibly a not horrible fit for winds. However, the normal approximation will not work on most precipitation forecasts and may fail for many other atmospheric variables.

Closed form expressions for the CRPS are difficult to define when using data rather than distribution functions. However, if a normal distribution can be assumed, then the following equation gives the CRPS for each individual observation (denoted by a lowercase crps) and the corresponding distribution of forecasts.

$$\text{crps}_i(N(\mu,\sigma^2),y) = \sigma(\frac{y-\mu}{\sigma}(2\Phi(\frac{y-\mu}{\sigma})-1)+2\phi(\frac{y-\mu}{\sigma})-\frac{1}{\sqrt{\pi}})$$

In this equation, the y represents the event threshold. The estimated mean and standard deviation of the ensemble forecasts (μ and σ) are used as the parameters of the normal distribution. The values of the normal distribution are represented by the probability density function (PDF) denoted by Φ and the cumulative distribution function (CDF), denoted in the above equation by ϕ.

The overall CRPS is calculated as the average of the individual measures. In equation form:

$$\text{CRPS} = \text{average(crps)} \ = \frac{1}{N}\sum_{i=1}^{N}\text{crps}_i$$

The score can be interpreted as a continuous version of the mean absolute error (MAE). Thus, the score is negatively oriented, so smaller is better. Further, similar to MAE, bias will inflate the CRPS. Thus, bias should also be calculated and considered when judging forecast quality using CRPS.

To calculate crps_emp_fair (bias adjusted, empirical ensemble CRPS) for each individual observation with m ensemble members:

$$\text{crps_emp_fair}_i = \text{crps_emp}_i - \frac{1}{2*m} * \frac{1}{m*(m-1)}\sum_{i\neq j}|f_i - f_j|$$

The overall CRPS_EMP_FAIR is calculated as the average of the individual measures. In equation form:

$$\text{CRPS_EMP_FAIR} = \text{average(crps_emp_fair)} \ = \frac{1}{N}\sum_{i=1}^{N}\text{crps_emp_fair}_i$$

35.5.3 Ensemble Mean Absolute Difference

Called "SPREAD_MD" in ECNT output Table 14.2

The ensemble mean absolute difference is an alternative measure of ensemble spread. It is computed for each individual observation (denoted by a lowercase spread_md) with m ensemble members:

$$\text{spread_md}_i = \frac{1}{m*(m-1)}\sum_{i\neq j}|f_i - f_j|$$

The overall SPREAD_MD is calculated as the average of the individual measures. In equation form:

$$\text{SPREAD_MD} = \text{average(spread_md)} \ = \frac{1}{N}\sum_{i=1}^{N}\text{spread_md}_i$$

A perfect forecast would have ensemble mean absolute difference = 0.

35.5.4 CRPS Skill Score

Called "CRPSS" and "CRPSS_EMP" in ECNT output Table 14.2

The continuous ranked probability skill score (CRPSS) is similar to the MSESS and the BSS, in that it compares its namesake score to that of a reference forecast to produce a positively oriented score between 0 and 1.

$$\text{CRPSS} = 1 - \frac{\text{CRPS}_{fcst}}{\text{CRPS}_{ref}}$$

For the normal distribution fit (CRPSS), the reference CRPS is computed using the climatological mean and standard deviation. For the empirical distribution (CRPSS_EMP), the reference CRPS is computed by sampling from the assumed normal climatological distribution defined by the mean and standard deviation.

35.5.5 Bias Ratio

Called "BIAS_RATIO" in ECNT output Table 14.2

The bias ratio (BIAS_RATIO) is computed when verifying an ensemble against gridded analyses or point observations. It is defined as the mean error (ME) of ensemble member values greater than or equal to the observation value to which they are matched divided by the absolute value of the mean error (ME) of ensemble member values less than the observation values.

$$\text{BIAS_RATIO} = \frac{\text{ME}_{f>=o}}{|\text{ME}_{f<o}|}$$

A perfect forecast has ME = 0. Since BIAS_RATIO is computed as the high bias (ME_GE_OBS) divide by the absolute value of the low bias (ME_LT_OBS), a perfect forecast has BIAS_RATIO = 0/0, which is undefined. In practice, the high and low bias values are unlikely to be 0.

The range for BIAS_RATIO is 0 to infinity. A score of 1 indicates that the high and low biases are equal. A score greater than 1 indicates that the high bias is larger than the magnitude of the low bias. A score less than 1 indicates the opposite behavior.

35.5.6 IGN

Called "IGN" in ECNT output Table 14.2

The ignorance score (IGN) is the negative logarithm of a predictive probability density function (*Gneiting et al., 2004* (page 495)). In MET, the IGN is calculated based on a normal approximation to the forecast distribution (i.e. a normal pdf is fit to the forecast values). This approximation may not be valid, especially for discontinuous forecasts like precipitation, and also for very skewed forecasts. For a single normal distribution **N** with parameters μ and σ, the ignorance score is

$$\text{ign}(N(\mu,\sigma),y) = \frac{1}{2}\ln(2\pi\sigma^2) + \frac{(y-\mu)^2}{2\sigma^2}.$$

Accumulation of the ignorance score for many forecasts is via the average of individual ignorance scores. This average ignorance score is the value output by the MET software. Like many error statistics, the IGN is negatively oriented, so smaller numbers indicate better forecasts.

35.5.7 PIT

Called "PIT" in ORANK output Table 14.7

The probability integral transform (PIT) is the analog of the rank histogram for a probability distribution forecast (*Dawid, 1984* (page 493)). Its interpretation is the same as that of the verification rank histogram: Calibrated probabilistic forecasts yield PIT histograms that are flat, or uniform. Under-dispersed (not enough spread in the ensemble) forecasts have U-shaped PIT histograms while over-dispersed forecasts have bell-shaped histograms. In MET, the PIT calculation uses a normal distribution fit to the ensemble forecasts. In many cases, use of other distributions would be better.

35.5.8 Observation Error Logarithmic Scoring Rules

Called "IGN_CONV_OERR" and "IGN_CORR_OERR" in ECNT output Table 14.2

One approach that is used to take observation error into account in a summary measure is to add error to the forecast by a convolution with the observation model (e.g., *Anderson, 1996* (page 491); *Hamill, 2001* (page 496); *Saetra et. al., 2004* (page 498); *Bröcker and Smith, 2007* (page 492); *Candille et al., 2007* (page 493); *Candille and Talagrand, 2008* (page 493); *Röpnack et al., 2013* (page 498)). Specifically, suppose $y = x + w$, where y is the observed value, x is the true value, and w is the error. Then, if f is the density forecast for x and ν is the observation model, then the implied density forecast for y is given by the convolution:

$$(f * \nu)(y) = \int \nu(y|x) f(x) dx$$

Ferro, 2017 (page 494) gives the error-convolved version of the ignorance scoring rule (referred to therein as the error-convolved logarithmic scoring rule), which is proper under the model where $w \sim N(0, c^2)$) when the forecast for x is $N(\mu, \sigma^2)$ with density function f, by

$$\text{IGN_CONV_OERR} = s(f, y) = \frac{1}{2}\log(2\pi(\sigma^2 + c^2)) + \frac{(y-\mu)^2}{2(\sigma^2 + c^2)}$$

Another approach to incorporation of observation uncertainty into a measure is the error-correction approach. The approach merely ensures that the scoring rule, s, is unbiased for a scoring rule s_0 if they have the same expected value. *Ferro, 2017* (page 494) gives the error-corrected ignorance scoring rule (which is also proposer when $w \sim N(0, c^2)$) as

$$\text{IGN_CORR_OERR} = s(f, y) = \frac{1}{2}\log(2\pi\sigma^2) + \frac{(y-\mu)^2 - c^2}{2\sigma^2}$$

The expected score for the error-convolved ignorance scoring rule typically differs from the expected score that would be achieved if there were no observation error. The error-corrected score, on the other hand, has the same expectation.

35.5.9 RANK

Called "RANK" in ORANK output Table 14.7

The rank of an observation, compared to all members of an ensemble forecast, is a measure of dispersion of the forecasts (*Hamill, 2001* (page 496)). When ensemble forecasts possess the same amount of variability as the corresponding observations, then the rank of the observation will follow a discrete uniform distribution. Thus, a rank histogram will be approximately flat.

The rank histogram does not provide information about the accuracy of ensemble forecasts. Further, examination of "rank" only makes sense for ensembles of a fixed size. Thus, if ensemble members are occasionally unavailable, the rank histogram should not be used. The PIT may be used instead.

35.5.10 SPREAD

Called "SPREAD" in ECNT output Table 14.2

Called "SPREAD" in ORANK output Table 14.7

The ensemble spread for a single observation is the standard deviation of the ensemble member forecast values at that location. When verifying against point observations, these values are written to the SPREAD column of the Observation Rank (ORANK) line type. The ensemble spread for a spatial masking region is computed as the square root of the mean of the ensemble variance for all observations falling within that mask. These values are written to the SPREAD column of the Ensemble Continuous Statistics (ECNT) line type.

Note that prior to met-9.0.1, the ensemble spread of a spatial masking region was computed as the average of the spread values within that region. This algorithm was corrected in met-9.0.1 to average the ensemble variance values prior to computing the square root.

35.6 MET Verification Measures for Neighborhood Methods

The results of the neighborhood verification approaches that are included in the Grid-Stat tool are summarized using a variety of measures. These measures include the Fractions Skill Score (FSS) and the Fractions Brier Score (FBS). MET also computes traditional contingency table statistics for each combination of threshold and neighborhood window size.

The traditional contingency table statistics computed by the Grid-Stat neighborhood tool, and included in the NBRCTS output, are listed below:

- Base Rate (called "BASER" in Table 13.3)
- Mean Forecast (called "FMEAN" in Table 13.3)
- Accuracy (called "ACC" in Table 13.3)
- Frequency Bias (called "FBIAS" in Table 13.3)
- Probability of Detection (called "PODY" in Table 13.3)
- Probability of Detection of the non-event (called "PODN" in Table 13.3)

- Probability of False Detection (called "POFD" in Table 13.3)
- False Alarm Ratio (called "FAR" in Table 13.3)
- Critical Success Index (called "CSI" in Table 13.3)
- Gilbert Skill Score (called "GSS" in Table 13.3)
- Hanssen-Kuipers Discriminant (called "HK" in Table 13.3)
- Heidke Skill Score (called "HSS" in Table 13.3)
- Odds Ratio (called "ODDS" in Table 13.3)

All of these measures are defined in Section 35.2.

In addition to these standard statistics, the neighborhood analysis provides additional continuous measures, the Fractions Brier Score and the Fractions Skill Score. For reference, the Asymptotic Fractions Skill Score and Uniform Fractions Skill Score are also calculated. These measures are defined here, but are explained in much greater detail in *Ebert, 2008* (page 494) and *Roberts and Lean, 2008* (page 498). *Roberts and Lean, 2008* (page 498) also present an application of the methodology.

35.6.1 Fractions Brier Score

Called "FBS" in NBRCNT output Table 13.4

The Fractions Brier Score (FBS) is defined as FBS $= \frac{1}{N}\sum_N[\langle P_f\rangle_s - \langle P_o\rangle_s]^2$, where N is the number of neighborhoods; $\langle P_f\rangle_s$ is the proportion of grid boxes within a forecast neighborhood where the prescribed threshold was exceeded (i.e., the proportion of grid boxes that have forecast events); and $\langle P_o\rangle_s$ is the proportion of grid boxes within an observed neighborhood where the prescribed threshold was exceeded (i.e., the proportion of grid boxes that have observed events).

35.6.2 Fractions Skill Score

Called "FSS" in NBRCNT output Table 13.4

The Fractions Skill Score (FSS) is defined as

$$\text{FSS} = 1 - \frac{\text{FBS}}{\frac{1}{N}[\sum_N \langle P_f\rangle_s^2 + \sum_N \langle P_o\rangle_s^2]},$$

where the denominator represents the worst possible forecast (i.e., with no overlap between forecast and observed events). FSS ranges between 0 and 1, with 0 representing no overlap and 1 representing complete overlap between forecast and observed events, respectively.

35.6.3 Asymptotic Fractions Skill Score

Called "AFSS" in NBRCNT output Table 13.4

The Asymptotic Fractions Skill Score (AFSS) is a special case of the Fractions Skill score where the entire domain is used as the single neighborhood. This provides the user with information about the overall frequency bias of forecasts versus observations. The formula is the same as for FSS above, but with N=1 and the neighborhood size equal to the domain.

35.6.4 Uniform Fractions Skill Score

Called "UFSS" in NBRCNT output Table 13.4

The Uniform Fractions Skill Score (UFSS) is a reference statistic for the Fractions Skill score based on a uniform distribution of the total observed events across the grid. UFSS represents the FSS that would be obtained at the grid scale from a forecast with a fraction/probability equal to the total observed event proportion at every point. The formula is $UFSS = (1+f_o)/2$ (i.e., halfway between perfect skill and random forecast skill) where f_o is the total observed event proportion (i.e. observation rate).

35.6.5 Forecast Rate

Called "F_rate" in NBRCNT output Table 13.4

The overall proportion of grid points with forecast events to total grid points in the domain. The forecast rate will match the observation rate in unbiased forecasts.

35.6.6 Observation Rate

Called "O_rate" in NBRCNT output Table 13.4

The overall proportion of grid points with observed events to total grid points in the domain. The forecast rate will match the observation rate in unbiased forecasts. This quantity is sometimes referred to as the base rate.

35.7 MET Verification Measures for Distance Map Methods

The distance map statistics include Baddeley's Δ Metric, a statistic which is a true mathematical metric. The definition of a mathematical metric is included below.

A mathematical metric, $m(A, B) \geq 0$, must have the following three properties:

1. Identity: $m(A, B) = 0$ if and only if $A = B$.
2. Symmetry: $m(A, B) = m(B, A)$
3. Triangle inequality: $m(A, C) \leq m(A, B) + m(B, C)$

The first establishes that a perfect score is zero and that the only way to obtain a perfect score is if the two sets are identical according to the metric. The second requirement ensures that the order by which the two sets are evaluated will not change the result. The third property ensures that if *C* is closer to *A* than *B* is to *A*, then $m(A, C) < m(A, B)$.

It has been argued in *Gilleland, 2017* (page 495) that the second property of symmetry is not necessarily an important quality to have for a summary measure for verification purposes because lack of symmetry allows for information about false alarms and misses.

The results of the distance map verification approaches that are included in the Grid-Stat tool are summarized using a variety of measures. These measures include Baddeley's Δ Metric, the Hausdorff Distance, the Mean-error Distance, Pratt's Figure of Merit, and Zhu's Measure. Their equations are listed below.

35.7.1 Baddeley's Δ Metric and Hausdorff Distance

Called "BADDELEY" and "HAUSDORFF" in the DMAP output Table 13.6

The Baddeley's Δ Metric is given by

$$\Delta_{p,w}(A, B) = [\frac{1}{N} \sum_{s \in D} |w(d(s, A)) - w(d(s, B))|]^{\frac{1}{P}}$$

where $d(s, \cdot)$ is the distance map for the respective event area, $w(\cdot)$ is an optional concave function (i.e., $w(t + u) \leq w(t) + w(u)$) that is strictly increasing at zero with $w(t) = 0$ if and only if $t = 0$, *N* is the size of the domain, and *p* is a user chosen parameter for the L_p norm. The default choice of $p = 2$ corresponds to a Euclidean average, $p = 1$ is a simple average of the difference in distance maps, and the limiting case of $p = \infty$ gives the maximum difference between the two distance maps and is called the Hausdorff distance, denoted as $H(A, B)$, and is the metric that motivated the development of Baddeley's Δ metric. A typical choice, and the only available with MET, for $w(\cdot)$ is $w(t) = \min\{t, c\}$, where *c* is a user-chosen constant with $c = \infty$ meaning that $w(\cdot)$ is not applied. This choice of $w(\cdot)$ provides a cutoff for distances beyond the pre-specified amount given by *c*.

In terms of distance maps, Baddeley's Δ is the L_p norm of the top left panel in Figure 13.4 provided $c = \infty$. If $0 < c < \infty$, then the distance maps in the bottom row of Figure 13.3 would be replaced by *c* wherever they would otherwise exceed *c* before calculating their absolute differences in the top left panel of Figure 13.4.

The range for BADDELEY and HAUSDORFF is 0 to infinity, with a score of 0 indicating a perfect forecast.

35.7.2 Mean-Error Distance

Called "MED_FO", "MED_OF", "MED_MIN", "MED_MAX", and "MED_MEAN" in the DMAP output Table 13.6

The mean-error distance (MED) is given by

$$\text{MED}\ (A, B) = \frac{1}{n_B} \sum_{s \in B} d(x, A)$$

where n_B is the number of non-zero grid points that fall in the event set *B*. That is, it is the average of the distance map for the event set *A* calculated only over those grid points that fall inside the event set *B*. It gives the average shortest-distance from every point in *B* to the nearest point in *A*.

Unlike Baddeley's Δ metric, the MED is not a mathematical metric because it fails the symmetry property. However, if a metric is desired, then any of the following modifications, which are metrics, can be employed instead, and all are available through MET.

$$min\text{MED}(A,B) = min(\text{MED}(A,B), \text{MED}(B,A))$$
$$max\text{MED}(A,B) = max(\text{MED}(A,B), \text{MED}(B,A))$$
$$mean\text{MED}(A,B) = \frac{1}{2}(\text{MED}(A,B) + \text{MED}(B,A))$$

From the distance map perspective, MED *(A,B)* is the average of the values in Figure 13.4 (top right), and MED *(B,A)* is the average of the values in Figure 13.4 (bottom left). Note that the average is only over the circular regions depicted in the figure.

The range for MED is 0 to infinity, with a score of 0 indicating a perfect forecast.

35.7.3 Pratt's Figure of Merit

Called "FOM_FO", "FOM_OF", "FOM_MIN", "FOM_MAX", and "FOM_MEAN" in the DMAP output Table 13.6

Pratt's Figure of Merit (FOM) is given by

$$\text{FOM}\ (A,B) = \frac{1}{max(n_A, n_B)} \sum_{s \in B} \frac{1}{1 + \alpha d(s,A)^2}$$

where $n_A$and n_B are the number of events within event areas *A* and *B*, respectively, $d(s,A)$ is the distance map related to the event area *A*, and α is a user-defined scaling constant. The default, and usual choice, is $\alpha = \frac{1}{9}$ when the distances of the distance map are normalized so that the smallest nonzero distance between grid point neighbors equals one. Clearly, FOM is not a metric because like MED, it is not symmetric. Like MED, MET computes the minimum, maximum, and average of FOM_FO and FOM_OF.

Note that $d(s,A)$ in the denominator is summed only over the grid squares falling within the event set *B*. That is, it represents the circular area in the top right panel of Figure 13.4.

The range for FOM is 0 to 1, with a score of 1 indicating a perfect forecast.

35.7.4 Zhu's Measure

Called "ZHU_FO", "ZHU_OF", "ZHU_MIN", "ZHU_MAX", and "ZHU_MEAN" in the DMAP output Table 13.6

Another measure incorporates the amount of actual overlap between the event sets across the fields in addition to the MED from above and was proposed by Zhu et al. (2011). Their main proposed measure was a comparative forecast performance measure of two competing forecasts against the same observation, which is not included here, but as defined is a true mathematical metric. They also proposed a similar measure of only the forecast against the observation, which is included in MET. It is simply

$$Z(A,B) = \lambda \sqrt{\frac{1}{N} \sum_{s \in D} (I_F(s) - I_O(s))^2} + (1 - \lambda) \cdot \text{MED}(A,B)$$

where MED *(A,B)* is as in the Mean-error distance, *N* is the total number of grid squares as in Baddeley's Δ metric, $I_F(s)((I_O(s))$ is the binary field derived from the forecast (observation), and λ is a user-chosen weight. The first term is just the RMSE of the binary forecast and observed fields, so it measures the average amount of overlap of event areas where zero would be a perfect score. It is not a metric because of the MED in the second term. A user might choose different weights depending on whether they want to emphasize the overlap or the MED terms more, but generally equal weight ($\lambda = \frac{1}{2}$) is sufficient. In Zhu et al (2011), they actually only consider $Z(F, O)$ and not $Z(O, F)$, but both are included in MET for the same reasons as argued with MED. Similar to MED, the average of these two directions (avg Z), as well as the min and max are also provided for convenience.

The range for ZHU is 0 to infinity, with a score of 0 indicating a perfect forecast.

35.7.5 G and G_β

Called "G" and "GBETA" in the DMAP output Table 13.6

See Section 13.2.9 for a description.

Let $y = y_1 y_2$ where $y_1 = n_A + n_B - 2n_{AB}$, and $y_2 = MED(A, B) \cdot n_B + MED(B, A) \cdot n_A$, with the mean-error distance (MED) as described above, and where n_A, n_B, and n_{AB} are the number of events within event areas *A*, *B*, and the intersection of *A* and *B*, respectively.

The G performance measure is given by

$$G(A, B) = y^{1/3}$$

and the G_β performance measure is given by

$$G_\beta(A, B) = max\{1 - \frac{y}{\beta}, 0\}$$

where $\beta > 0$ is a user-chosen parameter with a default value of $n^2/2.0$ with n equal to the number of points in the domain. The square-root of G will give units of grid points, where $y^{1/3}$ gives units of grid points squared.

The range for G_β is 0 to 1, with a score of 1 indicating a perfect forecast.

35.8 Calculating Percentiles

Several of the MET tools make use of percentiles in one way or another. Percentiles can be used as part of the internal computations of a tool, or can be written out as elements of some of the standard verification statistics. There are several widely-used conventions for calculating percentiles however, so in this section we describe how percentiles are calculated in MET.

The explanation makes use of the *floor* function. The floor of a real number *x*, denoted $\lfloor x \rfloor$, is defined to be the greatest integer $\leq x$. For example, $\lfloor 3.01 \rfloor = 3, \lfloor 3.99 \rfloor = 3, \lfloor -3.01 \rfloor = -4, \lfloor -3.99 \rfloor = -4$. These examples show that the floor function does *not* simply round its argument to the nearest integer. Note also that $\lfloor x \rfloor = x$ if and only if *x* is an integer.

Suppose now that we have a collection of *N* data points $x_i$for $i = 0, 1, 2, \ldots, N - 1$. (Note that we're using the C/C++ convention here, where array indices start at zero by default.) We will assume that the data

are sorted in increasing (strictly speaking, *nondecreasing*) order, so that $i \leq j$ implies $x_i \leq x_j$. Suppose also that we wish to calculate the *t* percentile of the data, where $0 \leq t < 1$. For example, $t = 0.25$ for the 25th percentile of the data. Define

$$I = \lfloor (N-1)t \rfloor$$
$$\Delta = (N-1)t - I$$

Then the value *p* of the percentile is

$$p = (1 - \Delta)x_I + \Delta x_{I+1}$$

Chapter 36

Appendix D Confidence Intervals

A single verification statistic is statistically meaningless without associated uncertainty information in accompaniment. There can be numerous sources of uncertainty associated with such a statistic including observational, physical uncertainties about the underlying processes governing the equation, sample uncertainty, etc. Although all of the sources of uncertainty can be important, the most heavily researched, and easiest to calculate, is that of sampling uncertainty. It is this source of uncertainty that can presently be obtained with MET, and the techniques for deriving these estimates are described here. Sampling uncertainty through MET is gleaned by way of confidence intervals (CIs) as these are generally most informative. A $(1-\alpha)\cdot 100\%$ confidence interval is interpreted, somewhat awkwardly, in the following way. If the test were repeated 100 times (so that we have 100 such intervals), then we expect the true value of the statistics to fall inside $(1-\alpha)\cdot 100$ of these intervals. For example, if $\alpha = 0.05$ then we expect the true value to fall within 95 of the intervals.

There are two main types of CIs available with MET: parametric and non-parametric. All of the parametric intervals used with MET rely on the underlying sample (or the errors, $F-O$) to be at least approximately independent and normally distributed. Future releases will allow for some types of dependency in the sample. The non-parametric techniques utilize what is known in the statistical literature as bootstrap resampling, which does not rely on any distributional assumptions for the sample; the assumption is that the sample is representative of the population. Bootstrap CIs can be inaccurate if the sample is not independent, but there are ways of accounting for dependence with the bootstrap, some of which will be added to MET in future releases. Details about which verification statistics have parametric CIs in MET are described next, and it should be noted that the bootstrap can be used for any statistic, though care should be taken in how it is carried out, and this is described subsequently.

The most commonly used confidence interval about an estimate for a statistic (or parameter), θ, is given by the normal approximation

$$\theta \pm z_{\alpha/2} \cdot V(\theta)$$

where $z_{\alpha/2}$ is the $\alpha -$ th quantile of the standard normal distribution, and $V(\theta)$ is the standard error of the statistic (or parameter), θ. For example, the most common example is for the mean of a sample, $X_1, \cdots, X_n$, of independent and identically distributed (iid) normal random variables with mean μ and variance σ. Here, the mean is estimated by $\frac{1}{n}\sum_{i=1}^{n} X_i = \bar{X}$, and the standard error is just the standard deviation of the random variables divided by the square root of the sample size. That is, $V(\theta) = V(\bar{X}) = \frac{\sigma}{\sqrt{n}}$, and this must be estimated by $\hat{V}(\bar{X})$, which is obtained here by replacing σ by its estimate, $\hat{\sigma}$, where $\hat{\sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$.

Mostly, the normal approximation is used as an asymptotic approximation. That is, the interval for θ may only be appropriate for large **n**. For small **n**, the mean has an interval based on the Student's **t** distribution with **n-1** degrees of freedom. Essentially, $z_{\alpha/2}$ of the question is replaced with the quantile of this **t** distribution. That is, the interval is given by

$$\mu \pm t_{\alpha/2,\nu-1} \cdot \frac{\sigma}{\sqrt{n}}$$

where again, σ is replaced by its estimate, $\hat{\sigma}$, as described above.

Table 36.1 summarizes the verification statistics in MET that have normal approximation CIs given by θ along with their corresponding standard error estimates, . It should be noted that for the first two rows of this table (i.e., Forecast/Observation Mean and Mean error) MET also calculates the interval around μ for small sample sizes.

Table 36.1: Verification statistics with normal approximation CIs given by the equation for θ provided in MET along with their associated standard error estimate.

$\hat{\theta}$	$V(\theta)$
Forecast / Observation Mean	$V(\bar{X}) = \frac{\sigma_x}{\sqrt{n}}$ where σ_x emphasizes that this is the estimated standard deviation of the underlying sample.
Mean error	$V(\bar{F}-\bar{O}) = \frac{\sigma_{F-O}}{\sqrt{n}}$, where σ_{F-O} emphasizes that this is the estimated standard deviation of the errors, $F-O$.
Brier Score (BS)	$V(\text{BS}) = \frac{1}{T}[\sum F^4 + \bar{O}(1 - 4\sum F^3_{F\|O=1} + 6\sum F^2_{F\|O=1} - 4\sum F_{F\|O=1}) - \text{BS}^2]$ where **F** is the **probability** forecast and **O** is the observation. See *Bradley et al, 2008* (page 492) for derivation and details.
Peirce Skill Score (PSS)	$V(\text{PSS}) = \sqrt{\frac{H(1-H)}{n_H} + \frac{F(1-F)}{n_F}}$, where **H** is the hit rate, **F** the false alarm rate, n_h the number of hits and misses, and n_F the number of false alarms and correct negatives.
Logarithm of the odds ratio (OR)	$V(\ln(\text{OR})) = \sqrt{\frac{1}{a} + \frac{1}{b} + \frac{1}{c} + \frac{1}{d}}$, where the values in the denominators are the usual contingency table counts.

Other statistics in MET having parametric CIs that rely on the underlying sample to be at least approximately iid normal, but have a different form derived from the normality assumption on the sample include the variance, standard deviation, and the linear correlation coefficient. These are addressed subsequently.

Generally, the normal interval around θ is appropriate for statistics of continuous variables, but a limit law for the binomial distribution allows for use of this interval with proportions. The most intuitive estimate for $V(\theta)$ in this case is given by $V(p) = \sqrt{\hat{p}(1-\hat{p})/n}$. However, this only applies when the sample size is large. A better approximation to the CI for proportions is given by Wilson's interval, which is

$$\frac{\hat{p} + z^2_{\alpha/2} + z_{\alpha/2}\sqrt{\hat{p}(1-\hat{p})/4n}}{1 + z^2_{\alpha/2}/n}$$

where $\hat{p}$ is the estimated proportion (e.g., hit rate, false alarm rate, PODy, PODn, etc.). Because this interval around $\hat{p}$ generally works better than the more intuitive normal approximation interval for both large and small sample sizes, this is the interval employed by MET.

The forecast/observation variance has CIs derived from the underlying sample being approximately iid normal with mean μ and variance σ. The lower and upper limits for the interval are given by

$$l(\sigma^2) = \frac{(n-1)s^2}{\chi^2_{\alpha/2,n-1}} \text{ and } u(\sigma^2) = \frac{(n-1)s^2}{\chi^2_{1-\alpha/2,n-1}}$$

respectively, where $\chi^2_{\alpha,\nu}$ is the $\alpha -$ th quantile of the chi-square distribution with **n-1** degrees of freedom. Taking the square roots of the limits of l yields the CI for the forecast/observation standard deviation.

Finally, the linear correlation coefficient has limits given by

$$(\frac{e^{2c_l}-1}{e^{2c_l}+1}, \frac{e^{2c_u}-1}{e^{2c_u}+1})$$

where $c_l = v - \frac{z_{\alpha/2}}{\sqrt{n-3}}$ and $c_u = v + \frac{z_{\alpha/2}}{\sqrt{n-3}}$.

All other verification scores with CIs in MET must be obtained through bootstrap resampling. However, it is also possible to obtain bootstrap CIs for any of the statistics given above, and indeed it has been proven that the bootstrap intervals have better accuracy for the mean than the normal approximation. The bootstrap algorithm is described below.

1. Assume the sample is representative of the population.
2. Resample with replacement from the sample (see text below).
3. Estimate the parameter(s) of interest for the current replicated sample.
4. Repeat steps 2 and 3 numerous times, say B times, so that you now have a sample of size B of the parameter(s).
5. Calculate CIs for the parameters directly from the sample (see text below for more details)

Typically, a simple random sample is taken for step 2, and that is how it is done in MET. As an example of what happens in this step, suppose our sample is X_1, X_2, X_3, X_4. Then, one possible replicate might be X_2, X_2, X_2, X_4. Usually one samples $m = n$ points in this step, but there are cases where one should use $m < n$. For example, when the underlying distribution is heavy-tailed, one should use a smaller size m than n (e.g., the closest integer value to the square root of the original sample size). See *Gilleland (2020, part II)* (page 495) for considerably more information about the issues with estimators that follow a heavy tailed distribution and the closely related issue of bootstrapping extreme-valued estimators, such as the maximum, in the atmospheric science domain.

There are numerous ways to construct CIs from the sample obtained in step 4. MET allows for two of these procedures: the percentile and the BCa. The percentile is the most commonly known method, and the simplest to understand. It is merely the $\alpha/2$ and $1 - \alpha/2$ percentiles from the sample of statistics. Unfortunately, however, it has been shown that this interval is too optimistic in practice (i.e., it doesn't have accurate coverage). One solution is to use the BCa method, which is very accurate, but it is also computationally intensive. This method adjusts for bias and non-constant variance, and yields the percentile interval in the event that the sample is unbiased with constant variance.

If there is dependency in the sample, then it is prudent to account for this dependency in some way. *Gilleland (2010)* (page 495) describes the bootstrap procedure, along with the above-mentioned parametric methods, in more detail specifically for the verification application. If there is dependency in the sample, then it is prudent to account for this dependency in some way (see *Gilleland (2020, part I)* (page 495) part I for an in-depth discussion of bootstrapping in the competing forecast verification domain). One method that is particularly appropriate for serially dependent data is the circular block resampling procedure for step 2.

Chapter 37

Appendix E WWMCA Tools

There are two WWMCA tools available. The WWMCA-Plot tool makes a PostScript plot of one or more WWMCA cloud percent files and the WWMCA-Regrid tool regrids WWMCA cloud percent files and reformats them into netCDF files that the other MET tools can read.

The WWMCA tools get valid time and hemisphere (north or south) information from the file names, so it's important for both of the WWMCA tools that these file names not be changed.

The usage statement for wwmca_plot is

```
wwmca_plot [ -outdir path ] wwmca_cloud_pct_file_list
```

Here, **wwmca_cloud_pct_file_list** represents one or more WWMCA cloud percent files given on the command line. As with any command given to a UNIX shell, the user can use meta-characters as a shorthand way to specify many filenames.

The optional **-outdir** argument specifies a directory where the output PostScript plots will be placed. If not specified, then the plots will be put in the current (working) directory. Figure 37.1 shows an example of the wwmca_plot output.

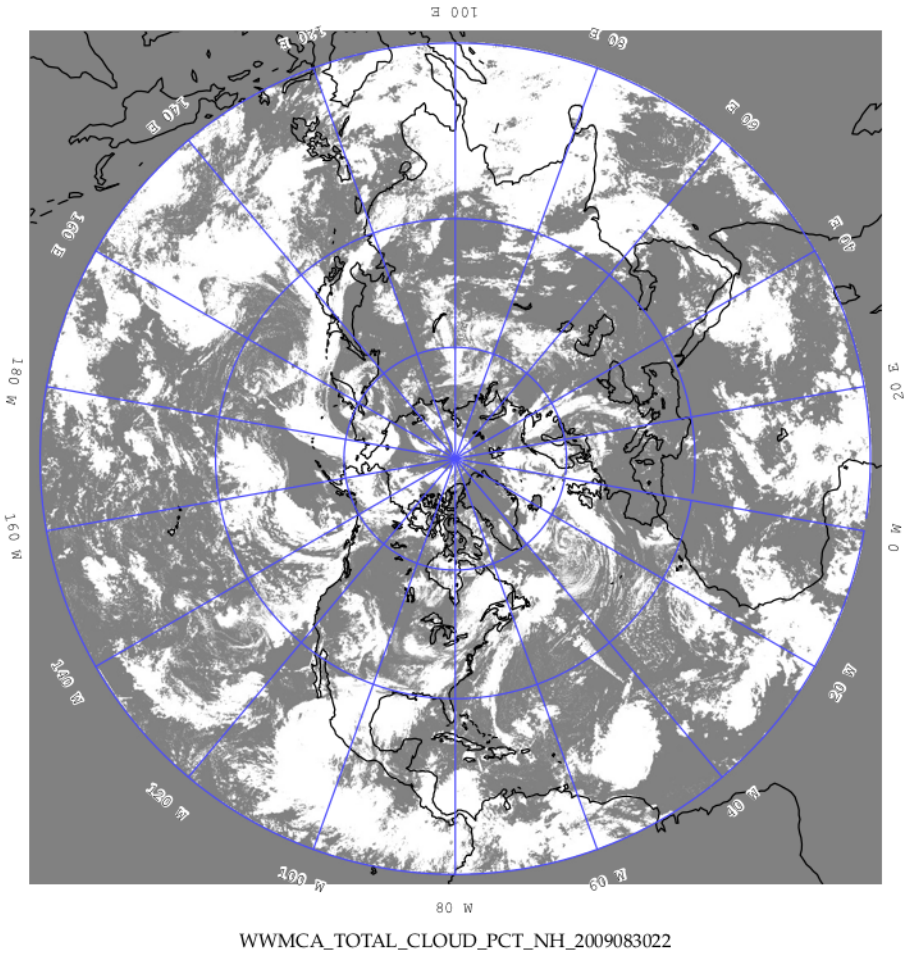

Figure 37.1: Example output of WWMCA-Plot tool.

The usage statement for wwmca_regrid is

```
wwmca_regrid -out filename config filename [ -nh filename ] [ -sh filename ]
```

Here, the **-out** switch tells wwmca_regrid what to name the output netCDF file. The **-config** switch gives the name of the config file that wwmca_regrid should use-like many of the MET tools, wwmca-regrid uses a configuration file to specify user-changeable parameters. The format of this file will be explained below.

The **-nh** and **-sh** options give names of WWMCA cloud percent files that wwmca_regrid should use as input. Northern hemisphere files are specified with **-nh**, and southern hemisphere files with **-sh**. At least one of these must be given, but in many cases both need not be given.

In any regridding problem, there are two grids involved: the "From" grid, which is the grid the input data are on, and the "To" grid, which is the grid the data are to be moved onto. For wwmca_regrid, the "To" grid is specified in the config file. If this grid is entirely confined to one hemisphere, then only the WWMCA data file for that hemisphere needs to be given. It's only when the "To" grid straddles the equator that data files for both hemispheres need to be given (though the interpolation width parameter in the config file can change this-see below). Once the "To" grid is specified in the config file, the WWMCA-Regrid tool will know which input data files it needs, and will complain if it's not given the right ones.

Now let's talk about the details of the config file. The config file has the same C-like syntax that all the other MET config files use. The first (and most complicated) thing to specify is the "To" grid. This is given by the **to_grid** parameter. If you are using one of the standard NCEP grids, for example grid #218, you can simply write

```
To grid = "G218";
```

and that will work. Failing that, you must give the parameters that specify the grid and it's projection. Please refer the description of the grid specification strings in *Appendix B Map Projections, Grids, and Polylines* (page 537).

Thankfully, the rest of the parameters in the config file are easier to specify.

The next two config file parameters have to do with specifying the interpolation scheme used. The **interp_method** parameter specifies which interpolation method is to be used. Four methods are supported: average, maximum, minimum and nearest neighbor. As an example, to specify the "average" method, one would write

```
interp_method = "average";
```

The other interpolation parameter is **interp_width**. This specifies the width of the interpolation box used in the above interpolation method. An example value could be

```
interp_width = 5;
```

The value must be odd and $\geqq 1$. If a value of 1 is specified, then nearest neighbor interpolation will be used regardless of the value assigned to **interp_method**.

The fact that an interpolation box is used has one subtle implication-the "To" grid is effectively fattened by half the width of the interpolation box. This means that even for a "To" grid that is entirely contained in one hemisphere, if it comes close to the equator, this virtual fattening may be enough to push it over the equator, and the user will then have to provide input WWMCA files for both hemispheres, even though the "To" grid doesn't cross the equator. The WWMCA-Regrid tool should detect this situation and complain to the user if not given the correct input files.

The next variable, **good_percent**, tells what fraction of the values in the interpolation square needs to be "good" in order for the interpolation scheme to return a "good" result. Example:

```
good percent = 0;
```

The rest of the config file parameters have to do with how the output netCDF file represents the data. These should be self-explanatory, so I'll just give an example:

```
variable_name = "Cloud Pct";
long_name     = "cloud cover percent";
grib_code     = 100;
units         = "percent";
level         = "SFC";
```

Chapter 38

Appendix F Python Embedding

38.1 Introduction

MET includes the ability to embed Python to a limited degree. Users may use their own Python scripts and any associated Python packages they wish in order to prepare 2D gridded data fields, point observations, and matched pairs as input to the MET tools. We fully expect that this degree of embedding will increase in the future. In addition, plans are in place to extend Python with MET in upcoming releases, allowing users to invoke MET tools directly from their Python script. While MET version 8.0 was built on Python 2.x, MET versions 9.0 and beyond are built on Python 3.6+.

38.2 Compiling MET for Python Embedding

In order to use Python embedding, a local Python installation must be available when compiling the MET software with the following requirements:

1. Python version 3.10.4+
2. C-language Python header files and libraries
3. **NumPy** Python package
4. **Pandas** Python package
5. **Xarray** Python package
6. **YAML** Python package
7. **SciPy** Python package
8. **netCDF4** Python package

Users should be aware that in some cases, the C-language Python header files and libraries may be deleted at the end of the Python installation process, and they may need to confirm their availability prior to compiling MET. Once the user has confirmed the above requirements are satisfied, they can compile the MET software for Python embedding by passing the **--enable-python** option to the **configure** script on the command line. This will link the MET C++ code directly to the Python libraries.

The **NumPy**, **Xarray**, and **Pandas** Python packages are required by the Python scripts included with the MET software that facilitate the passing of data in memory. The **SciPy** and **YAML** Python packages are required by the tropical cyclone diagnostics Python scripts called by the TC-Diag tool. The **netCDF4** package is used for reading and writing temporary files for Python embedding, but only when the **MET_PYTHON_TMP_FORMAT** environment variable is set to *netcdf* at runtime.

In addition to using **--enable-python** with **configure** as mentioned above, the following environment variables must also be set prior to executing **configure**: **MET_PYTHON_BIN_EXE**, **MET_PYTHON_CC**, and **MET_PYTHON_LD**. These may either be set as environment variables or as command line options to **configure**. These environment variables are used when building MET to enable the compiler to find the requisite Python executable, header files, and libraries in the user's local filesystem. Fortunately, Python provides a way to set these variables properly. This frees the user from the necessity of having any expert knowledge of the compiling and linking process. Along with the **Python** executable in the users local Python installation, there should be another executable called **python3-config**, whose output can be used to set these environment variables as follows:

- Set **MET_PYTHON_BIN_EXE** to the full path of the desired Python executable.
- On the command line, run "**python3-config --cflags**". Set the value of **MET_PYTHON_CC** to the output of that command.
- Again on the command line, run "**python3-config --ldflags --embed**". Set the value of **MET_PYTHON_LD** to the output of that command.

Make sure that these are set as environment variables or that you have included them on the command line prior to running **configure**

If a user attempts to invoke Python embedding with a version of MET that was not compiled with Python, MET will return an ERROR:

Listing 38.1: MET Errors Without Python Enabled

```
ERROR  : Met2dDataFileFactory::new_met_2d_data_file() -> Support for Python has not been␣
↪compiled!
ERROR  : To run Python scripts, recompile with the --enable-python option.

- or -

ERROR  : process_point_obs() -> Support for Python has not been compiled!
ERROR  : To run Python scripts, recompile with the --enable-python option.
```

38.3 Controlling Which Python MET Uses When Running

When MET is compiled with Python embedding support, MET uses the Python executable in that Python installation by default when Python embedding is used. However, for users of highly configurable Python environments, the Python instance set at compilation time may not be sufficient. Users may want to use an alternate Python installation if they need additional packages not available in the Python installation used when compiling MET. In MET versions 9.0+, users have the ability to use a different Python executable when running MET than the version used when compiling MET by setting the environment variable

MET_PYTHON_EXE. Whenever **MET_PYTHON_EXE** is set, MET writes a temporary file, as described in `Contributor's Guide Section %s`.

If a user's Python script requires packages that are not available in the Python installation used when compiling the MET software, they will encounter a runtime error when using MET. In this instance, the user will need to change the Python MET is using to a different installation with the required packages for their script. It is the responsibility of the user to manage this Python installation, and one popular approach is to use a custom Anaconda (Conda) Python environment. Once the Python installation meeting the user's requirements is available, the user can force MET to use it by setting the **MET_PYTHON_EXE** environment variable to the full path of the Python executable in that installation. For example:

Listing 38.2: Setting MET_PYTHON_EXE

```
export MET_PYTHON_EXE=/usr/local/python3/bin/python3
```

Setting this environment variable triggers slightly different processing logic in MET than when MET uses the Python installation that was used when compiling MET. When using the Python installation that was used when compiling MET, Python is called directly and data are passed in memory from Python to the MET tools. When the user sets **MET_PYTHON_EXE**, MET does the following:

1. Wrap the user's Python script and arguments with a wrapper script (write_tmp_mpr.py, write_tmp_point.py, or write_tmp_dataplane.py) and specify the name of a temporary file to be written.
2. Use a system call to the **MET_PYTHON_EXE** Python instance to execute these commands and write the resulting data objects to a temporary ASCII or NetCDF file.
3. Use the Python instance that MET was compiled with to run a wrapper script (read_tmp_ascii.py or read_tmp_dataplane.py) to read data from that temporary file.

With this approach, users are able to execute Python scripts using their own custom Python installations.

38.4 Data Structures Supported by Python Embedding

Python embedding with MET tools offers support for three different types of data structures:

1. Two-dimensional (2D) gridded dataplanes
2. Point observation data conforming to the *MET 11-column format* (page 161)
3. Statistics data, including matched-pair data conforming to the *MET MPR Line Type* (page 239)

Details for each of these data structures are provided below.

Note: All sample commands and directories listed below are relative to the top level of the MET source code directory.

38.4.1 Python Embedding for 2D Gridded Dataplanes

Currently, MET supports two different types of Python objects for two-dimensional gridded dataplanes: NumPy N-dimensional arrays (ndarrays) and Xarray DataArrays. The keyword **PYTHON_NUMPY** is used on the command line when using ndarrays, and **PYTHON_XARRAY** when using Xarray DataArrays. Example commands are included at the end of this section.

38.4.1.1 Python Script Requirements for 2D Gridded Dataplanes

1. The data must be stored in a variable with the name **met_data**
2. The **met_data** variable must be of type **Xarray DataArray** or **NumPy N-D Array**
3. The data inside the **met_data** variable must be **double precision floating point** type
4. A Python dictionary named **attrs** must be defined in the user's script and contain the *required attributes* (page 592) and any *optional attributes* (page 592)

38.4.1.2 Attributes for 2D Gridded Dataplanes

Table 38.1: 2D Dataplane Attributes

key	description	data type/format	required/optional
valid	valid time	string (YYYYMMDD_HHMMSS)	required
init	initialization time	string (YYYYMMDD_HHMMSS)	required
lead	forecast lead	string (HHMMSS)	required
accum	accumulation interval	string (HHMMSS)	required
name	variable name	string	required
long_name	variable long name	string	required
level	variable level	string	required
units	variable units	string	required
grid	*grid information* (page 593)	string or dict	required
fill_value	*missing data value* (page 592)	int or float	optional

Note: Often times Xarray DataArray objects come with their own set of attributes available as a property. To avoid conflict with the required attributes for MET, it is advised to strip these attributes and rely on the **attrs** dictionary defined in your script.

Python embedding for 2D gridded dataplanes provides support for a user-defined missing data (or fill value). By default, the MET tools will respect (and ignore) the following special values in a user's **met_data** variable:

1. NaN
2. Inf

3. -9999
4. -9999.

If a user has a 2D dataplane with another value that should be considered a fill value by MET, then the user must use the **fill_value** attribute in the **attrs** dictionary. An example would be if a user had a 2D dataplane with missing data indicated with -99. A user can use the **fill_value** attribute in their **attrs** dictionary which will tell MET to ignore those values:

Listing 38.3: User Fill Value for 2D Dataplane

```
'fill_value': -99
```

Alternatively, the user can choose to replace their special values with one of the four supported values instead of setting the **fill_value** attribute. Note that only a single user-defined fill value is supported at this time.

The grid entry in the **attrs** dictionary must contain the grid size and projection information in the same format that is used in the netCDF files written out by the MET tools. The value of this item in the dictionary can either be a string, or another dictionary. Examples of the **grid** entry defined as a string are:

- Using a named grid supported by MET:

Listing 38.4: Named Grid

```
'grid': 'G212'
```

- As a grid specification string, as described in *Appendix B Map Projections, Grids, and Polylines* (page 537):

Listing 38.5: Grid Specification String

```
'grid': 'lambert 185 129 12.19 -133.459 -95 40.635 6371.2 25 25 N'
```

- As the path to an existing gridded data file:

Listing 38.6: Grid From File

```
'grid': '/path/to/sample_data.grib'
```

When specified as a dictionary, the contents of the **grid** entry vary based upon the grid **type**. The required elements for supported grid types are:

- **Lambert Conformal** grid dictionary entries:
  - type ("Lambert Conformal")
  - name (string)
  - hemisphere (string: "N" or "S")
  - scale_lat_1, scale_lat_2 (double)
  - lat_pin, lon_pin, x_pin, y_pin (double)
  - lon_orient (double)

  - d_km, r_km (double)
  - nx, ny (int)
- **Polar Stereographic** grid dictionary entries:
  - type ("Polar Stereographic")
  - name (string)
  - hemisphere (string: "N" or "S")
  - scale_lat (double)
  - lat_pin, lon_pin, x_pin, y_pin (double)
  - lon_orient (double)
  - d_km, r_km (double)
  - nx, ny (int)
- **Mercator** grid dictionary entries:
  - type ("Mercator")
  - name (string)
  - lat_ll (double)
  - lon_ll (double)
  - lat_ur (double)
  - lon_ur (double)
  - nx, ny (int)
- **LatLon** grid dictionary entries:
  - type ("LatLon")
  - name (string)
  - lat_ll, lon_ll (double)
  - delta_lat, delta_lon (double)
  - Nlat, Nlon (int)
- **Rotated LatLon** grid dictionary entries:
  - type ("Rotated LatLon")
  - name (string)
  - rot_lat_ll, rot_lon_ll (double)
  - delta_rot_lat, delta_rot_lon (double)
  - Nlat, Nlon (int)
  - true_lat_south_pole, true_lon_south_pole (double)

    - aux_rotation (double)
- **Gaussian** grid dictionary entries:
    - type ("Gaussian")
    - name (string)
    - lon_zero (double)
    - nx, ny (int)
- **Range/Azimuth** grid dictionary entries:
    - type ("Range Azimuth")
    - range_n, azimuth_n (int)
    - range_max_km (double)
    - lat_center, lon_center (double)
- **SemiLatLon** grid dictionary entries:
    - type ("SemiLatLon")
    - name (string)
    - lats (list of doubles)
    - lons (list of doubles)
    - levels (list of doubles)
    - times (list of doubles)

Additional information about supported grids can be found in *Appendix B Map Projections, Grids, and Polylines* (page 537).

Finally, an example **attrs** dictionary is shown below:

Listing 38.7: Sample Attrs Dictionary

```
attrs = {

   'valid':     '20050807_120000',
   'init':      '20050807_000000',
   'lead':      '120000',
   'accum':     '120000',

   'name':      'Foo',
   'long_name': 'FooBar',
   'level':     'Surface',
   'units':     'None',

   # Define 'grid' as a string or a dictionary
```

(continues on next page)

(continued from previous page)

```
   'grid': {
      'type': 'Lambert Conformal',
      'hemisphere': 'N',
      'name': 'FooGrid',
      'scale_lat_1': 25.0,
      'scale_lat_2': 25.0,
      'lat_pin': 12.19,
      'lon_pin': -135.459,
      'x_pin': 0.0,
      'y_pin': 0.0,
      'lon_orient': -95.0,
      'd_km': 40.635,
      'r_km': 6371.2,
      'nx': 185,
      'ny': 129,
    }
}
```

38.4.1.3 Running Python Embedding for 2D Gridded Dataplanes

On the command line for any of the MET tools which will be obtaining its data from a Python script rather than directly from a data file, the user should specify either **PYTHON_NUMPY** or **PYTHON_XARRAY** wherever a (forecast or observation) data file would normally be given. Then in the **name** entry of the config file dictionaries for the forecast or observation data (typically used to specify the field name from the input data file), the user should list the **full path** to the Python script to be run followed by any command line arguments for that script. Note that for tools like MODE that take two data files, it is entirely possible to use the **PYTHON_NUMPY** for one file and the **PYTHON_XARRAY** for the other.

Listed below is an example of running the Plot-Data-Plane tool to call a Python script for data that is included with the MET release tarball. Assuming the MET executables are in your path, this example may be run from the top-level MET source code directory:

Listing 38.8: plot_data_plane Python Embedding

```
plot_data_plane PYTHON_NUMPY fcst.ps \
'name="scripts/python/examples/read_ascii_numpy.py data/python/fcst.txt FCST";' \
-title "Python enabled plot_data_plane"
```

The first argument for the Plot-Data-Plane tool is the gridded data file to be read. When calling Python script that has a two-dimensional gridded dataplane stored in a NumPy N-D array object, set this to the constant string **PYTHON_NUMPY**. The second argument is the name of the output PostScript file to be written. The third argument is a string describing the data to be plotted. When calling a Python script, set **name** to the full path of the Python script to be run along with any command line arguments for that script. Lastly, the **-title** option is used to add a title to the plot. Note that any print statements included in the Python script will be printed to the screen. The above example results in the following log messages:

```
DEBUG 1: Opening data file: PYTHON_NUMPY
Input File: 'data/python/fcst.txt'
Data Name : 'FCST'
Data Shape: (129, 185)
Data Type:  dtype('float64')
Attributes: {'name': 'FCST',  'long_name': 'FCST_word',
             'level': 'Surface', 'units': 'None',
             'init': '20050807_000000', 'valid': '20050807_120000',
             'lead': '120000',  'accum': '120000'
             'grid': { ... } }
DEBUG 1: Creating postscript file: fcst.ps
```

38.4.1.4 Special Case for Gen-Ens-Prod, Ensemble-Stat, Series-Analysis, and MTD

The Gen-Ens-Prod, Ensemble-Stat, Series-Analysis, and MTD tools all have the ability to read multiple input files. Because of this feature, a different approach to Python embedding is required. A typical use of these tools is to provide a list of files on the command line. For example:

Listing 38.9: Gen-Ens-Prod Command Line

```
gen_ens_prod ens1.nc ens2.nc ens3.nc ens4.nc -out ens_prod.nc -config GenEnsProd_config
```

In this case, a user is passing 4 ensemble members to Gen-Ens-Prod to be evaluated, and each member is in a separate file. If a user wishes to use Python embedding to process the ensemble input files, then the same exact command is used; however special modifications inside the GenEnsProd_config file are needed. In the config file dictionary, the user must set the **file_type** entry to either **PYTHON_NUMPY** or **PYTHON_XARRAY** to activate the Python embedding for these tools. Then, in the **name** entry of the config file dictionaries for the forecast or observation data, the user must list the **full path** to the Python script to be run. However, in the Python command, replace the name of the input gridded data file to the Python script with the constant string **MET_PYTHON_INPUT_ARG**. When looping over all of the input files, the MET tools will replace that constant **MET_PYTHON_INPUT_ARG** with the path to the input file currently being processed and optionally, any command line arguments for the Python script. Here is what this looks like in the GenEnsProd_config file for the above example:

Listing 38.10: Gen-Ens-Prod MET_PYTHON_INPUT_ARG Config

```
file_type = PYTHON_NUMPY;
field = [ { name = "gen_ens_prod_pyembed.py MET_PYTHON_INPUT_ARG"; } ];
```

In the event the user requires command line arguments to their Python script, they must be included alongside the file names separated by a delimiter. For example, the above Gen-Ens-Prod command with command line arguments for Python would look like:

Listing 38.11: Gen-Ens-Prod Command Line with Python Args

```
gen_ens_prod ens1.nc,arg1,arg2 ens2.nc,arg1,arg2 ens3.nc,arg1,arg2 ens4.nc,arg1,arg2 \
-out ens_prod.nc -config GenEnsProd_config
```

In this case, the user's Python script will receive "ens1.nc,arg1,arg2" as a single command line argument for each execution of the Python script (i.e. 1 time per file). The user must parse this argument inside their Python script to obtain **arg1** and **arg2** as separate arguments. The list of input files and optionally, any command line arguments can be written to a single file (called **python_input_list** in the example below) that is substituted for the file names and command line arguments. ASCII file list elements are white-space separated (space-separated in the example below), as described in Section 4.1.1. For example:

Listing 38.12: Gen-Ens-Prod File List

```
echo "file_list ens1.nc,arg1,arg2 ens2.nc,arg1,arg2 ens3.nc,arg1,arg2 ens4.nc,arg1,arg2" >␣
↪python_input_list
gen_ens_prod python_input_list -out ens_prod.nc -config GenEnsProd_config
```

Finally, the above tools do not require data files to be present on a local disk. If the user wishes, their Python script can obtain data from other sources based upon only the command line arguments to their Python script. For example:

Listing 38.13: Gen-Ens-Prod Python Args Only

```
gen_ens_prod 20230101,0 20230102,0 20230103,0 -out ens_prod.nc -confg GenEnsProd_config
```

In the above command, each of the arguments "20230101,0", "20230102,0", and "20230103,0" are provided to the user's Python script in separate calls. Then, inside the Python script these arguments are used to construct a filename or query to a data server or other mechanism to return the desired data and format it the way MET expects inside the Python script, prior to calling Gen-Ens-Prod.

38.4.1.5 Examples of Python Embedding for 2D Gridded Dataplanes

Grid-Stat with Python embedding for forecast and observations

Listing 38.14: GridStat Command with Dual Python Embedding

```
grid_stat 'PYTHON_NUMPY' 'PYTHON_NUMPY' GridStat_config -outdir /path/to/output
```

Listing 38.15: GridStat Config with Dual Python Embedding

```
fcst = {
   field = [
      {
        name = "/path/to/fcst/python/script.py python_arg1 python_arg2";
```

(continues on next page)

(continued from previous page)

```
    }
  ];
}

obs = {
  field = [
    {
      name = "/path/to/obs/python/script.py python_arg1 python_arg2";
    }
  ];
}
```

38.4.2 Python Embedding for Point Observations

MET also supports point observation data supplied in the *MET 11-column format* (page 161).

38.4.2.1 Python Script Requirements for Point Observations

1. The data must be stored in a variable with the name **point_data**
2. The **point_data** variable must be a Python list representation of a NumPy N-D Array created from a Pandas DataFrame
3. The **point_data** variable must have data in each of the 11 columns required for the MET tools even if it is NA

To provide the data that MET expects for point observations, the user is encouraged when designing their Python script to consider how to map their observations into the MET 11-column format. Then, the user can populate their observations into a Pandas DataFrame with the following column names and dtypes:

Table 38.2: Point Observation DataFrame Columns and Dtypes

column name	data type (dtype)	description
typ	string	Message Type
sid	string	Station ID
vld	string	Valid Time (YYYYMMDD_HHMMSS)
lat	numeric	Latitude (Degrees North)
lon	numeric	Longitude (Degrees East)
elv	numeric	Elevation (MSL)
var	string	Variable name (or GRIB code)
lvl	numeric	Level
hgt	numeric	Height (MSL or AGL)
qc	string	QC string
obs	numeric	Observation Value

To create the variable for MET, use the **.values** property of the Pandas DataFrame and the **.tolist()** method of the NumPy N-D Array. For example:

Listing 38.16: Convert Pandas DataFrame to MET variable

```
# Pandas DataFrame
my_dataframe = pd.DataFrame()

# Convert to MET variable
point_data = my_dataframe.values.tolist()
```

38.4.2.2 Running Python Embedding for Point Observations

The Point2Grid, Plot-Point-Obs, Ensemble-Stat, and Point-Stat tools support Python embedding for point observations. Python embedding for these tools can be invoked directly on the command line by replacing the input MET NetCDF point observation file name with the **full path** to the Python script and any arguments. The Python command must begin with the prefix **PYTHON_NUMPY=**. The full command should be enclosed in quotes to prevent embedded whitespace from causing parsing errors. An example of this is shown below for Plot-Point-Obs:

Listing 38.17: plot_point_obs with Python Embedding

```
plot_point_obs \
"PYTHON_NUMPY=scripts/python/examples/read_ascii_point.py data/sample_obs/ascii/sample_ascii_
↪obs.txt" \
output_image.ps
```

The ASCII2NC tool also supports Python embedding, however invoking it varies slightly from other MET tools. For ASCII2NC, Python embedding is used by providing the "-format python" option on the command line. With this option, point observations may be passed as input. An example of this is shown below:

Listing 38.18: ascii2nc with Python Embedding

```
ascii2nc -format python \
"scripts/python/examples/read_ascii_point.py data/sample_obs/ascii/sample_ascii_obs.txt" \
sample_ascii_obs_python.nc
```

Both of the above examples use the **read_ascii_point.py** example script which is included with the MET code. It reads ASCII data in MET's 11-column point observation format and stores it in a Pandas DataFrame to be read by the MET tools using Python embedding for point data. The **read_ascii_point.py** example script can be found in:

- MET installation directory in *scripts/python/examples*.
- MET GitHub repository in *scripts/python/examples*.

38.4.2.3 Examples of Python Embedding for Point Observations

Point-Stat with Python embedding for forecast and observations

Listing 38.19: PointStat Command with Dual Python Embedding

```
point_stat 'PYTHON_NUMPY' 'PYTHON_NUMPY=/path/to/obs/python/script.py python_arg1 python_arg2
↪' PointStat_config -outdir /path/to/output
```

Listing 38.20: PointStat Config with Dual Python Embedding

```
fcst = {
   field = [
      {
        name = "/path/to/fcst/python/script.py python_arg1 python_arg2";
      }
   ];
 }
```

38.4.3 Python Embedding for STAT Data

The MET Stat-Analysis tool also supports Python embedding. By using the command line option **-lookin python**, Stat-Analysis can read statistics data generated by the MET statistics tools. This is particularly useful for reading matched pair (MPR) data formatted in the MET MPR line-type format via Python.

The MET Pair-Stat tool also supports Python embedding of matched pair (MPR) data using the **-format python** command line option. This functionality expects data formatted in the MET MPR format within Python prior to passing to Pair-Stat.

Note: While Stat-Analysis can read all STAT line types through Python embedding, Pair-Stat only reads the MPR line type. Note that the MET statistics tools write all output line types to a STAT file, but can also be

configured to write each line type to separate text (TXT) files. The example below reads data from an MPR text file generated by Point-Stat where each line has the same number of columns. It will not work for STAT files, in general, where the number of columns vary by line type.

38.4.3.1 Python Script Requirements for MPR Data

1. The data must be stored in a variable with the name **mpr_data**, although Stat-Analysis actually supports any STAT line type
2. The **mpr_data** variable must be a Python list representation of a NumPy N-D Array created from a Pandas DataFrame
3. The **met_data** variable must have data in **exactly** 36 columns for MPR data, corresponding to the summation of the *common STAT output* (page 227) and the *MPR line type output* (page 239).

If a user does not have an existing MPR line type file created by the MET tools, they will need to map their data into the 36 columns expected by Stat-Analysis for the MPR line type data. If a user already has MPR line type files, the most direct way for a user to read MPR line type data is to model their Python script after the sample **read_ascii_mpr.py** script. Sample code is included here for convenience:

Listing 38.21: Reading MPR line types with Pandas

```
# Open the MPR line type file
mpr_dataframe = pd.read_csv(input_mpr_file,\
                            header=None,\
                            delim_whitespace=True,\
                            keep_default_na=False,\
                            skiprows=1,\
                            usecols=range(1,36),\
                            dtype=str)

# Convert to the variable MET expects
mpr_data = mpr_dataframe.values.tolist()
```

Note: If reading non-MPR STAT line types as input, the example above should be modified based on the number of columns for the input line type.

38.4.3.2 Running Python Embedding for MPR Data

Stat-Analysis can be run using the **-lookin python** command line option:

Listing 38.22: Stat-Analysis with Python Embedding of MPR Data

```
stat_analysis \
-lookin python scripts/python/examples/read_ascii_mpr.py point_stat_mpr.txt \
-job aggregate_stat -line_type MPR -out_line_type CNT \
-by FCST_VAR,FCST_LEV
```

In this example, rather than passing the MPR output lines from Point-Stat directly into Stat-Analysis (which is the typical approach), the **read_ascii_mpr.py** Python embedding script reads that file and passes the data to Stat-Analysis. The aggregate_stat job is defined on the command line and CNT statistics are derived from the MPR input data. Separate CNT statistics are computed for each unique combination of FCST_VAR and FCST_LEV present in the input.

The **read_ascii_mpr.py** sample script can be found in:

- MET installation directory in *scripts/python/examples*.
- MET GitHub repository in *MET/scripts/python/examples*.

38.5 MET Python Package

MET comes with a Python package that provides core functionality for the Python embedding capability. In rare cases, advanced users may find the classes and functions included with this Python package useful.

To utilize the MET Python package **standalone** when NOT using it with Python embedding, users must add the following to their **PYTHONPATH** environment variable:

Listing 38.23: MET Python Module PYTHONPATH

```
export PYTHONPATH={MET_INSTALL_DIR}/share/met/python
```

where {MET_INSTALL_DIR} is the top level directory where MET is installed, for example **/usr/local/met**.

Chapter 39

Appendix G Vectors and Vector Statistics

In this appendix, we discuss some basic properties of vectors, concentrating on the two-dimensional case. To keep the discussion simple, we will assume we are using a Cartesian coordinate system.

Traditionally, vectors have been defined as quantities having both magnitude and direction, exemplified by a directed line segment. The magnitude of the vector is shown by the length of the segment, and the direction of the vector is usually shown by drawing an arrowhead on one end of the segment. Computers (and, in the author's experience, people) are usually more comfortable working with numbers, and so instead of the usual graphical definition of a vector, we will take the definition used in analytic geometry: A (two-dimensional) vector is an ordered pair of numbers. We will use boldface type to denote vectors, and so we can write

$$\mathbf{v} = (a, b)$$

to show that the vector $\mathbf{v}$ consists of the ordered pair of numbers *a* and *b*. The number *a* is called the first (or *x*) component of $\mathbf{v}$, and *b* is called the second (or *y*) component. Vector addition is performed componentwise: $(a, b)+(c, d) = (a+c, b+d)$, and similarly for subtraction. If α is a scalar, then we define multiplication by the scalar α as $\alpha(a, b) = (\alpha a, \alpha b)$, and similarly for division by a (nonzero) scalar.

The *norm* (or length, or magnitude) of a vector $\mathbf{v} = (a, b)$, is

$$|\mathbf{v}| = \sqrt{a^2 + b^2}$$

Note that $|\mathbf{v}| = 0$ if and only if $a = b = 0$, in which case we say that $\mathbf{v}$ is the *zero vector*. If α is a scalar, then

$$|\alpha\mathbf{v}| = |\alpha||\mathbf{v}|$$

The most important relation between vectors and their norms is given by the *triangle inequality*:

$$|\mathbf{v} + \mathbf{w}| \leq |\mathbf{v}| + |\mathbf{w}|$$

In some cases, only the direction of a vector is of interest, and in such cases we can replace a nonzero vector $\mathbf{v}$ by the unique vector $N(\mathbf{v})$ that has the same direction as $\mathbf{v}$, but has norm 1:

$$N(\mathbf{v}) = \frac{\mathbf{v}}{|\mathbf{v}|}$$

The vector $N(\mathbf{v})$ will be called the *unit vector* corresponding to $\mathbf{v}$, or more simply the *direction* of $\mathbf{v}$. Note that the zero vector has no direction.

Since vectors are characterized by magnitude (norm) and direction, this gives two ways to compare vectors: we can compare either their magnitudes or their directions. If $\mathbf{v}$ and $\mathbf{w}$ are vectors, then we can compare their norms by either taking the norm of the difference $|\mathbf{v} - \mathbf{w}|$ or the difference of the norms $|\mathbf{v}| - |\mathbf{w}|$. It's not always made clear in verification studies which of these is meant, and in general these two quantities will be different. However, by making use of the triangle inequality it can be shown that there is a relation between them. To derive this, let $\mathbf{z} = \mathbf{v} - \mathbf{w}$, from which we get $\mathbf{v} = \mathbf{w} + \mathbf{z}$. Now taking norms and using the triangle inequality,

$$|\mathbf{v}| = |\mathbf{w} + \mathbf{z}| \leq |\mathbf{w}| + |\mathbf{z}| = |\mathbf{w}| + |\mathbf{v} - \mathbf{w}|$$

which gives

$$|\mathbf{v}| - |\mathbf{w}| \leq |\mathbf{v} - \mathbf{w}|$$

Reversing the roles of $\mathbf{v}$ and $\mathbf{w}$ now gives the result:

$$||\mathbf{v}| - |\mathbf{w}|| \leq |\mathbf{v} - \mathbf{w}|$$

In the same manner, we can compare the directions of two different nonzero vectors $\mathbf{v}$ and $\mathbf{w}$ by either the direction of the difference $N(\mathbf{v} - \mathbf{w})$, or by the difference in the directions $N(\mathbf{v}) - N(\mathbf{w})$. Unlike the case for magnitudes, however, there is in general no relationship at all between these two measures of direction difference.

Now let us specialize this discussion of vectors to verification of wind vector data. We will denote the forecast wind vector by **F**, and the observed wind vector by **O**. These are two-dimensional horizontal vectors with *u* and *v* components as follows:

$$\mathbf{F} = (u_f, v_f) \text{ and } \mathbf{O} = (u_o, v_o)$$

We will assume that we have *N* observations of forecast and observed wind:

$$\mathbf{F}_i = (u_{fi}, v_{fi}) \text{ and } \mathbf{O}_i = (u_{oi}, v_{oi})$$

for $1 \leq i \leq N$. We also have the forecast and observed wind *speeds*:

$$s_f = |\mathbf{F}| = \sqrt{u_f^2 + v_f^2} \text{ and } s_o = |\mathbf{O}| = \sqrt{u_o^2 + v_o^2}$$

and, at each data point,

$$s_{fi} = |\mathbf{F}_i| = \sqrt{u_{fi}^2 + v_{fi}^2} \text{ and } s_{oi} = |\mathbf{O}_i| = \sqrt{u_{oi}^2 + v_{oi}^2}$$

It will be convenient to denote the average forecast and observed wind vectors by $\mathbf{F}_a$ and $\mathbf{O}_a$:

$$\mathbf{F}_a = \frac{1}{N}\sum_i \mathbf{F}_i \text{ and } \mathbf{O}_a = \frac{1}{N}\sum_i \mathbf{O}_i$$

Now let us look at the definitions of the vector statistics produced by MET:

FBAR and OBAR are the average values of the forecast and observed wind speed.

$$\text{FBAR} = \frac{1}{N}\sum_i s_{fi}$$

$$\text{OBAR} = \frac{1}{N}\sum_i s_{oi}$$

FS_RMS and OS_RMS are the root-mean-square values of the forecast and observed wind speeds.

$$\text{FS_RMS} = [\frac{1}{N}\sum_i s_{fi}^2]^{1/2}$$

$$\text{OS_RMS} = [\frac{1}{N}\sum_i s_{oi}^2]^{1/2}$$

MSVE and RMSVE are, respectively, the mean squared, and root mean squared, lengths of the vector difference between the forecast and observed wind vectors.

$$\text{MSVE} = \frac{1}{N}\sum_i |\mathbf{F}_i - \mathbf{O}_i|^2$$

$$\text{RMSVE} = \sqrt{MSVE}$$

FSTDEV and OSTDEV are the standard deviations of the forecast and observed wind speeds.

$$\text{FSTDEV} = \frac{1}{N}\sum_i (s_{fi} - \text{FBAR})^2 = \frac{1}{N}\sum_i s_{fi}^2 - \text{FBAR}^2$$

$$\text{OSTDEV} = \frac{1}{N}\sum_i (s_{oi} - \text{OBAR})^2 = \frac{1}{N}\sum_i s_{oi}^2 - \text{OBAR}^2$$

FDIR and ODIR are the direction (angle) of $\mathbf{F}_a$ and $\mathbf{O}_a$ with respect to the grid directions.

$$\text{FDIR} = \text{ direction angle of } \mathbf{F}_a$$

$$\text{ODIR} = \text{ direction angle of } \mathbf{O}_a$$

FBAR_SPEED and OBAR_SPEED are the lengths of the average forecast and observed wind vectors. Note that this is *not* the same as the average forecast and observed wind speeds (*ie.,* the length of an average vector $\neq$ the average length of the vector).

$$\text{FBAR_SPEED} = |\mathbf{F}_a|$$

$$\text{OBAR_SPEED} = |\mathbf{O}_a|$$

VDIFF_SPEED is the length (*ie. speed*) of the vector difference between the average forecast and average observed wind vectors.

$$\text{VDIFF_SPEED} = |\mathbf{F}_a - \mathbf{O}_a|$$

Note that this is *not* the same as the difference in lengths (speeds) of the average forecast and observed wind vectors. That quantity is called SPEED_ERR (see below). There is a relationship between these two statistics however: using some of the results obtained in the introduction to this appendix, we can say that $||\mathbf{F}_a| - |\mathbf{O}_a|| \leq |\mathbf{F}_a - \mathbf{O}_a|$ or, equivalently, that |SPEED_ERR | $\leq$ VDIFF_SPEED.

VDIFF_DIR is the direction of the vector difference of the average forecast and average observed wind vectors. Note that this is *not* the same as the difference in direction of the average forecast and average observed wind vectors. This latter quantity would be FDIR $-$ ODIR.

$$\text{VDIFF_DIR} = \text{direction of } (\mathbf{F}_a - \mathbf{O}_a)$$

SPEED_ERR is the difference in the lengths (speeds) of the average forecast and average observed wind vectors. (See the discussion of VDIFF_SPEED above.)

$$\text{SPEED_ERR} = |\mathbf{F}_a| - |\mathbf{O}_a| = \text{FBAR_SPEED} - \text{OBAR_SPEED}$$

SPEED_ABSERR is the absolute value of SPEED_ERR. Note that we have SPEED_ABSERR $\leq$ VDIFF_SPEED (see the discussion of VDIFF_SPEED above).

$$\text{SPEED_ABSERR} = |\text{SPEED_ERR }|$$

DIR_ERR is the signed angle between the directions of the average forecast and average observed wind vectors. Positive if the forecast vector is counterclockwise from the observed vector.

$$\text{DIR_ERR} = \text{direction between } N(\mathbf{F}_a) \text{ and } N(\mathbf{O}_a)$$

DIR_ABSERR is the absolute value of DIR_ERR. In other words, it's an unsigned angle rather than a signed angle.

$$\text{DIR_ABSERR} = |\text{DIR_ERR }|$$

The following statistics are computed by comparing the forecast and observed wind directions for each individual matched pair.

For each point, the directed angle difference in degrees is computed between the forecast and observed wind vectors and rescaled to the range from -180, exclusive, to 180, inclusive.

$$N(\mathbf{F}_i) - N(\mathbf{O}_i) \in (-180, 180]$$

Note however that the direction of the zero vector is undefined. Points for which the forecast or observed wind direction is undefined are excluded from the analysis and result in a warning message being printed. The "wind_thresh" and "wind_logic" configuration options, described in Section 5, can be used to filter the wind vectors down to a subset that meet the specified wind speed threshold.

DIR_ME is the average of the signed difference between the forecast and observed wind directions.

$$\text{DIR_ME} = \frac{1}{N}\sum_i (N(\mathbf{F}_i) - N(\mathbf{O}_i))$$

DIR_MAE is the average of the absolute value of the difference between the forecast and observed wind directions.

$$\text{DIR_MAE} = \frac{1}{N}\sum_i |N(\mathbf{F}_i) - N(\mathbf{O}_i)|$$

DIR_MSE is the average of the squared difference between the forecast and observed wind directions.

$$\text{DIR_MSE} = \frac{1}{N}\sum_i (N(\mathbf{F}_i) - N(\mathbf{O}_i))^2$$

DIR_RMSE is the square root of the average squared difference between the forecast and observed wind directions.

$$\text{DIR_RMSE} = \sqrt{DIR_MSE}$$

Chapter 40

Appendix H Unstructured Grids

40.1 Introduction

METv12.0.0+ includes limited support for using datasets on unstructured grids (UGRIDs) in the PointStat and GridStat tools. This support includes verifying UGRID forecasts against point observations (PointStat), and verifying UGRID forecasts regridded to a structured grid against gridded observations on a structured grid (GridStat). The implementation of UGRID support in METv12.0.0+ provides a mechanism for a user to describe the topology of their unstructured grid. Thus far, development to support datasets on unstructured grids has been driven by the LFRic ("elfrick") NWP model (*Adams et al. 2019* (page 491)), and the Model for Prediction Across Scales (MPAS) NWP model (*Skamarock et al., 2012* (page 499)). However, the support for unstructured grids was implemented in such a way that additional unstructured grids can be utilized, with the appropriate mapping of the user's topology to the elements that MET requires.

40.2 Unstructured Grid Files

Support for UGRID files in NetCDF format is provided. MET will attempt to auto-detect UGRID files. If the file is CF-compliant and the "Conventions" global attribute is "UGRID" or "MPAS", or if the global attribute "mesh_spec" is present, MET recognizes the file as a UGRID file. If the user is not using a supported UGRID, or a UGRID file that does not contain the required global attributes for autodetection, the user should set the **file_type** configuration entry to **NETCDF_UGRID**. In some cases, the UGRID topology is provided in an ancillary file separate from forecast variables. This is controlled via the **ugrid_coordinates_file** configuration entry. This file is checked first, and then the UGRID data file. If the same elements are found in both files, the information from the data file takes precedence, and thus it is recommended to leave **ugrid_coordinates_file** unset if all required elements are found in the data file. If both a data and coordinates file are provided, the shape of the latitude and longitude dimensions are cross-referenced as a limited enforcement measure that the coordinates file and data file belong to the same UGRID.

40.3 Required Unstructured Grid Metadata

To correctly parse the UGRID topology, MET needs the following information that must be contained within the UGRID coordinates file or the UGRID data file. It is often the case that the variable names for these elements differ between UGRIDs. If the user is not using one of the supported UGRIDs, they will need to define the **ugrid_metadata_map** in a custom configuration file that they provide on the command line using the **-ugrid_config** command line argument, which tells MET the variable names that correspond to the following elements:

Table 40.1: Required UGRID Metadata

Metadata Map Key	Description	Required
dim_face	Dimension name for UGRID cells	Required
lat_face	Coordinate variable name for the latitude of each UGRID cell	Required
lon_face	Coordinate variable name for the longitude of each UGRID cell	Required
dim_time	Dimension name for the time coordinate	Required
time	Coordinate variable for valid time	Required
init_time	Coordinate variable for initialization time (used to compute the lead time)	Optional
dim_vert	Dimension name for the vertical coordinate, if present	Optional
vert_face	Coordinate variable name for the vertical coordinate	Optional

40.4 Unstructured Grid Configuration Entries

In the PointStat and GridStat config files, a user can control aspects of the UGRID capability. Since the UGRID support in MET is optional, these items are not included in the default MET config files. If the user requires modification for any of the following, they must add them to their MET config file.

40.4.1 ugrid_dataset

If the UGRID dataset is supported by MET, then this item is set to the string identifying the UGRID. Currently supported ugrid_dataset options include "lfric", and "mpas". If this item is not set, the user must provide a separate UGRID configuration file on the command line using the **-ugrid_config** command line argument.

40.4.2 ugrid_max_distance_km

For PointStat, this is the maximum allowable distance in kilometers for a point observation to be matched to the nearest UGRID cell center. If left unspecified, each point observation will be matched to the closest UGRID cell, regardless of how far apart they are. If no UGRID cells are within the specified distance, the observation will be not used for verification. For GridStat, this is the distance from each forecast grid point to search for observation grid cells to include when interpolating to the forecast grid point locations. Currently, only the nearest point observation or gridded observation cell is used. See *Unstructured Grid Limitations* (page 613) for additional details about interpolation when using GridStat.

40.4.3 ugrid_coordinates_file

The absolute path to the ancillary NetCDF file that describes the UGRID topology. This file is checked for the required UGRID metadata first, and the data file is checked second and takes precedence over this file if the same elements are found in both files.

40.4.4 ugrid_metadata_map

The mapping dictionary which allows a user to specify the variable names of the required UGRID topology elements that are required by MET. For "lfric" and "mpas", **ugrid_metadata_map** comes pre-configured with MET and a user can simply set **ugrid_dataset**. Otherwise, the user must create a separate UGRID configuration file containing the **ugrid_metadata_map** containing the elements from Table 40.1 and provide it on the command line using the **-ugrid_config** command line argument.

40.5 Unstructured Grid Limitations

We anticipate expanding the UGRID capabilities in MET in the near future. Until then, users are encouraged to be mindful of the following limitations:

1. In GridStat, there is no support for verifying directly on a UGRID. The UGRID (either forecast, obs, or both) must be regridded to a structured verification grid prior to verification being performed. In most cases, gridded observations are on a structured grid and so the users are encouraged to regrid their UGRID forecast to the structured observation grid. If the user wishes to verify a UGRID observation against a UGRID forecast (for example, a UGRID model analysis), the user must define a custom verification grid tailored to their UGRID characteristics, being mindful that the regridding can be slow for large verification grids. The user is referred to *Appendix B Map Projections, Grids, and Polylines* (page 537) for guidance on defining a structured verification grid. For Point-Stat, evaluation occurs by interpolating UGRID data directly to observation point locations and does not require regridding.
2. Data at cell edges are currently not supported, only those variables which have data at the cell centers are supported. Users should note in particular that wind components that are typically derived using data at cell edges are currently unsupported.
3. No aggregation methods of point observations within the **ugrid_max_distance_km** are supported except NEAREST, and no aggregation methods of gridded observations within the **ugrid_max_distance_km** are supported except NEAREST.