ARL-TN-0876 • MAR 2018

US Army Research Laboratory

User-Defined Meteorological (MET) Profiles from Climatological and Extreme Condition Data

by JL Cogan

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0876 ● MAR 2018

US Army Research Laboratory

User-Defined Meteorological (MET) Profiles from Climatological and Extreme Condition Data

by JL Cogan
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
March 2018	Technical Note	8 January 2018–1 February 2018

4. TITLE AND SUBTITLE
User-Defined Meteorological (MET) Profiles from Climatological and Extreme Condition Data

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
JL Cogan

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
Computational and Information Sciences Directorate (ATTN: RDRL-CIE)
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0876

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Longer-term strategic and tactical planning need to include the potential effects of the atmosphere on operations as well as test and evaluation. Estimates of changes in the atmosphere, often forecasted via numerical weather models, have application in the shorter term of from an hour or less to perhaps 2 weeks. For longer outlooks beyond the skill of forecasting methods, planners often resort to climate data to provide the required meteorological information. This report investigates the application of climate data for use in test and evaluation, though the same methods could be applied to planning for other purposes. More specifically, the method of this report first extracts data including climatic extremes from files supplied by the Air Force 14th Weather Squadron. It then generates vertical profiles of meteorological variables using height levels and layers that may be defined by the user.

15. SUBJECT TERMS
climate data, climate data processing, extreme atmospheric conditions, vertical profiles, climate data applications

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			JL Cogan
Unclassified	Unclassified	Unclassified	UU	42	19b. TELEPHONE NUMBER (Include area code) (301) 394-2304

Contents

List of Tables

Acknowledgments

I acknowledge Jeffrey Zautner of the Air Force 14th Weather Squadron for rapidly and accurately providing climate files with the requested types of data.

INTENTIONALLY LEFT BLANK.

1. Introduction

Longer-term strategic and tactical planning need to include the potential effects of the atmosphere on operations as well as test and evaluation (T&E). Estimates of changes in the atmosphere, often forecasted via numerical weather models, have application in the shorter term of from an hour or less to perhaps 2 weeks. For longer outlooks beyond the skill of forecasting methods, planners often resort to climate data to provide the required meteorological information. T&E and operational planning may include knowledge of potential extreme weather conditions that a system may have to overcome to perform its function, as well as statistics such as mean and standard deviation.

The Air Force 14th Weather Squadron (14 WS) has climate data for sites around the world for periods running from several years to many decades. Personnel of the 14 WS can produce tables of climate data having means and standard deviations of many measured (e.g., temperature) and derived (e.g., density) meteorological variables at user-requested or standard height or pressure levels from their data archive. For the work leading to this report, the data lines were at regular intervals every 300 m from the surface to 30 km. Heights were listed in meters above ground level (AGL). Users may request additional climate data aside from the "standard" set of statistics. In this report, extreme conditions for wind and density were provided for those data lines. An example of an extreme condition for a given location is the 95th percentile wind speed, which represents the wind speed where 95% of the archived wind speeds for the specified location are lower and 5% are higher.

The methods of this report were used to extract the necessary data from the tables supplied by the 14 WS, produce similar tables with only the data needed for the application, and generate user-specified height level and layer vertical profiles of the base statistics and user-selected extreme values. A Python program (aka Python script) was used to revise the input climate tables for processing by an external C program called by the parent Python script. The C program that produces the resultant height level and layer vertical profiles is a modified version of the program described by Cogan (2015, 2017).

2. Method

The method to process the climate files into user-specified vertical profiles consists of 2 main parts. The first employs a Python program or script (Python 3.5) to extract the relevant statistical and extreme value data from the large tabular files supplied by the 14 WS. The output file from the processed climate file becomes the input for

the external C program called from the Python script. The C program converts the columns of the processed climate data into vertical profiles of the statistical values (e.g., mean, standard deviation) of measured and derived variables, and extreme values of wind speed and/or density, for user-defined levels and layers. The user selects the extreme values to be processed as well as the vertical height structure of the output tables. These and other user-defined parameters are read from parameter files. The Python and C parts may be considered as one "combined program" in the remainder of this report.

2.1 Preparation of the Parameter Files

Information for running the combined program is entered into a parameter file currently named input_pars, which is located in the same directory as the Python script. The first line holds the extreme value identifiers or types for wind and/or density. An example for an extreme wind identifier is w95 for the 95th percentile wind speed, representing the wind speed where 95% of wind speeds are less and 5% are more. Similarly as for wind speed, 90th percentile density represents a value where 90% of the densities are smaller and 10% are greater. However, for the climate data files received for the work of this report, the density data used to generate the extreme values were taken from subsets defined by the extreme wind values. An example of an extreme density type for a given level and site is d90_90 for the 90th percentile density computed using the subset of all data for that level and site having wind speeds that are greater than the 90th percentile wind speed. Another example could be d95_80 for the 95th percentile density within the subset of data for the 80th percentile wind speed. In the parameter file, the wind and density identifiers can be listed in any order, but those selected should be in the file provided by the 14 WS. If an extreme value type listed in the parameter file is not in the input climate file from the 14 WS (e.g., w995 instead of w95), then the program will continue with the other listed types. It will also print out a message to remind the user to check the validity of the identifiers listed in the parameter file.

The second line has the directory of the external C program. The location does not have to be the same as the Python script and, for this report, it was in a separate directory. The third and fourth lines are the locations (directories) of the input and output files for the C program. The input directory is the same for the Python and C programs. The output directory may be the same or different; for this report, they were the same. Note that the C program uses a parameter file (i.e., input_parameters) to define directories and so on. The Python script provides that file with the necessary information for operation of the C program. Table 1 presents a sample of the parameter file used by the combined program.

Table 1 Sample parameter file (input_pars) for the combined program. YBBN is the World Meteorological Organization identifier for Brisbane, Australia

w90 w95 d90_90 d95_95
/data/user/C_progs/Climate/
/data/user/pyfiles/Clim_data/YBBN/
/data/user/pyfiles/Clim_data/YBBN/

Although the Python script generates the input_parameters file for the C program, there are others within the C program's directory that are necessary for the program to operate. Those other parameter files contain the height structures needed for the output vertical profiles. One has the height structure of a computer meteorological message (METCM) and produces values for the surface (aka zone 0) and layer or zone values from the layer adjacent to the surface to as high as the uppermost one with a maximum upper boundary of 30 km. The filename is metcm_lvls. The second has a user-defined height structure, where the user defines the height levels (layer boundaries), and is named usrmsg_lvls. Separate output files are produced for the listed height levels and the layers defined by those levels. For the layered output, the surface value is the same as for level 0. If one of the "lvls" files is missing or has a different name (e.g., metcm_lvls1), the C program will continue to run, but will skip the generation of the respective vertical profile. If both are missing or have different names, the program will not produce an output file and print an error message (i.e., "No met messages computed. Need height structure input files.").

2.2 The Combined Program Procedure

The combined program (climstat_all.py) includes code provided by Reen (2017) that produces a log file (clim.log) that contains additional statements as needed that should assist in debugging and reduces the number of "housekeeping" statements printed to the screen. After checking to make sure that the input_pars file has the correct extreme value identifiers and directory paths, the procedure to run the program is straightforward. To run the combined program, enter the following:

python3 climstat_all.py PATH_AND_INPUT_FILENAME,

where the use of all capital letters indicates a generic name, in this case the path and filename of the input file.

For example,

python3 climstat_all.py /data/user/Clim_data/KTUS/Climate_KTUS_mo6,

where Climate_KTUS_mo6 is the input filename. Here KTUS refers to the Tucson International Airport and mo6 indicates month 6 or June. The output file from the Python script has "_out" added to the end of the filename (e.g., Climate_KTUS_mo6_out).

The processed climate file with "_out" added to the input filename becomes the input file for the external C program. That program (convertclim) is called from within the Python script. The C program is nearly the same as that described in Cogan (2015, 2017), but was modified to read in and process a variable number of extreme value columns as well as the columns for the means and standard deviations. Certain variables were either included or excluded from the output relative to those produced by the previous versions of the C program. For example, density, which appears in the input climate data file, was included, but wind direction was excluded, since it was not in the input climate file. If it is run as a standalone program, a user would enter, for example, the following:

```
./convertclim  Climate_KTUS_mo6_out.
```

The output file from the combined program, or the standalone C program, will have "_METCM" added to the input filename for the output with the height structure of a METCM and "_USRLVL" and "_USRMSG" for the user-defined level and layer output, respectively (e.g., Climate_KTUS_mo6_out_USRMSG). Appendix A has a listing of the Python script except for the large definition that includes logging code and Appendix B contains the code for modified parts of the C program.

3. Input and Output Samples

The typical input file from the 14 WS for the work of this report is a large file in comma-separated value (csv) format, which is readable using MS Excel. One file was produced for each site and month. However, to make it readily readable for many computer programs, each csv file was converted to a standard text file (no commas). In this process, the name was changed to match the site name and month. For example, the climate data for Fairbanks, AK (PAFA) for June was converted to a text file and renamed to Climate_PAFA_mo6. To cover the largest likely number of extreme values that could be applied to potential applications, 36 extreme value columns were included by the 14 WS along with 8 columns of means and standard deviations plus the heights (0 through 30 km), the number of wind speed samples for each line, and the location identifier and month (location and month the same for all lines). Table 2 has a section of one of the files from the 14 WS.

Table 2 A section of the climate data file from the 14 WS: 18 of 48 columns and 22 (including the column headers) of 102 rows. The identifiers for the extreme values are explained in the text (e.g., w80 = 80% wind). Mean and std refer to mean and standard deviation, respectively, of temperature (t, in °C), wind speed (w, in ms^{-1}), density (d, in gm^{-3}), and pressure (p, in hPA [mb]). Site refers to the site identifier, obsw to the number of wind speed observations or samples, and mo and hgt to month and height, respectively.

Site	mo	hgt	meanw	meant	meand	meanp	stdw	stdt	stdd	stdp	obsw	w80	w90	w95	w99	d80_1	d80_5
PAFA	JAN	0	1.5012	-20.33	1373.5	994.73	1.9422	10.824	68.466	14.653	2777	2.1	3	4.1	9.3	1239.7	1280.2
PAFA	JAN	300	5.2091	-14.57	1289.7	955.66	3.5622	10.336	57.776	13.788	2767	7.9	10.1	12.1	16.3	1172.4	1198
PAFA	JAN	600	6.7947	-13.21	1232.8	918.58	4.0231	9.7338	50.071	13.103	2767	10.1	12.3	14.3	18	1137.9	1159
PAFA	JAN	900	7.7302	-12.13	1180.2	883.14	4.4385	9.4717	44.922	12.65	2767	11.4	14	16.2	19.3	1092.9	1116
PAFA	JAN	1200	8.2238	-11.67	1132.6	849.14	4.8697	9.044	40.421	12.56	2767	12.1	15.1	17.4	22.1	1054.2	1073.7
PAFA	JAN	1500	8.5772	-12.06	1090.2	816.31	4.9361	8.3294	35.278	12.289	2763	12.3	15.5	17.9	22.6	1024.2	1039
PAFA	JAN	1800	8.9143	-13.12	1052.1	784.74	5.144	7.7342	30.826	12.085	2762	12.85	16.2	18.7	23.3	995.1	1007.8
PAFA	JAN	2100	9.2704	-14.47	1016.4	754.27	5.3161	7.2882	27.159	11.904	2762	13.3	16.5	19.7	24.3	963.6	977
PAFA	JAN	2400	9.6718	-16.02	982.62	724.89	5.5764	7.0221	24.337	11.759	2762	14	17.2	20.3	25.9	934.7	946.5
PAFA	JAN	2700	10.088	-17.69	950.34	696.56	5.8752	6.8542	22.034	11.69	2761	14.8	18	20.8	27.8	907.4	917.2
PAFA	JAN	3000	10.555	-19.44	918.97	668.99	5.9937	6.7304	20.042	11.717	2760	15.3	18.8	21.7	28.2	880.5	888.8
PAFA	JAN	3300	10.954	-21.25	888.58	642.28	6.2093	6.6539	18.386	11.698	2759	15.9	19.6	22.5	28.8	853.1	861
PAFA	JAN	3600	11.394	-23.11	859.24	616.49	6.4786	6.6284	17.028	11.673	2758	16.5	20.6	23.4	30.3	825.2	833.5
PAFA	JAN	3900	11.922	-25.01	830.85	591.6	6.7248	6.6025	15.758	11.653	2757	17.1	21.5	24.4	32	796.3	806.6
PAFA	JAN	4200	12.508	-26.92	803.27	567.59	7.0985	6.5567	14.633	11.609	2756	18.1	22.6	25.7	34.2	769.9	779.7
PAFA	JAN	4500	13.146	-28.89	776.61	544.36	7.5154	6.5563	13.648	11.572	2756	19.2	23.7	27.4	36.4	746.5	754
PAFA	JAN	4800	13.779	-30.87	750.71	521.97	8.0209	6.5337	12.75	11.502	2755	20.2	24.8	29.2	38.2	721.8	728.9
PAFA	JAN	5100	14.527	-32.82	725.61	500.45	8.4821	6.4866	11.954	11.404	2752	21.4	26.5	30.3	40	696.3	705.1
PAFA	JAN	5400	15.25	-34.8	700.94	479.48	8.8886	6.4488	11.246	11.446	2750	22.6	28.2	31.9	40.2	672.4	681.5
PAFA	JAN	5700	15.903	-36.78	676.92	459.22	9.3559	6.3539	10.65	11.358	2746	23.6	29.5	33.7	42.3	649.4	657.8
PAFA	JAN	6000	16.545	-38.73	653.48	439.68	9.8203	6.2262	10.076	11.279	2746	24.4	30.7	35.1	44.8	629	634.9

The Python script reduced the number of columns by keeping only those columns of extreme wind or density that were identified by the user in the parameter file and removing certain columns of redundant information. For example, the site was identified in the filename and did not need to be repeated as a separate column. Table 3 presents a section of the reduced file Climate_PAFA_mo1_out.

Table 3 A section of the reduced file produced by the Python program. The header includes the name of the input file, which provides the site and month (e.g., Fairbanks, Alaska, for January). The column headers have the same meaning as in Table 2. In this example, the extreme values chosen were w95, w99, d95_95, and d99_99. Shown are 23 (including the input filename and column headers) of 103 rows.

/data/user/Clim_data/PAFA/Climate_PAFA_mo1													
hgt	meanp	meanw	meant	meand	stdp	stdt	stdd	stdw	obsw	w95	w99	d95_95	d99_99
0	994.73	1.5	-20.33	1373.5	14.65	10.82	68.47	1.9422	2777	4.1	9.3	1473.8	1447.2
300	955.66	5.21	-14.57	1289.7	13.79	10.34	57.78	3.5622	2767	12.1	16.3	1370.9	1331.7
600	918.58	6.79	-13.21	1232.8	13.1	9.73	50.07	4.0231	2767	14.3	18	1305.5	1266.7
900	883.14	7.73	-12.13	1180.2	12.65	9.47	44.92	4.4385	2767	16.2	19.3	1240.8	1211.6
1200	849.14	8.22	-11.67	1132.6	12.56	9.04	40.42	4.8697	2767	17.4	22.1	1191.6	1164.4
1500	816.31	8.58	-12.06	1090.2	12.29	8.33	35.28	4.9361	2763	17.9	22.6	1144.4	1128
1800	784.74	8.91	-13.12	1052.1	12.09	7.73	30.83	5.144	2762	18.7	23.3	1097.2	1092.5
2100	754.27	9.27	-14.47	1016.4	11.9	7.29	27.16	5.3161	2762	19.7	24.3	1055.8	1041.6
2400	724.89	9.67	-16.02	982.62	11.76	7.02	24.34	5.5764	2762	20.3	25.9	1021.6	1004.3
2700	696.56	10.09	-17.69	950.34	11.69	6.85	22.03	5.8752	2761	20.8	27.8	984.8	968.4
3000	668.99	10.55	-19.44	918.97	11.72	6.73	20.04	5.9937	2760	21.7	28.2	946.4	928.6
3300	642.28	10.95	-21.25	888.58	11.7	6.65	18.39	6.2093	2759	22.5	28.8	914.2	897.6
3600	616.49	11.39	-23.11	859.24	11.67	6.63	17.03	6.4786	2758	23.4	30.3	883.3	868.4
3900	591.6	11.92	-25.01	830.85	11.65	6.6	15.76	6.7248	2757	24.4	32	854.5	838.8
4200	567.59	12.51	-26.92	803.27	11.61	6.56	14.63	7.0985	2756	25.7	34.2	824.2	810.6
4500	544.36	13.15	-28.89	776.61	11.57	6.56	13.65	7.5154	2756	27.4	36.4	797.3	783.4
4800	521.97	13.78	-30.87	750.71	11.5	6.53	12.75	8.0209	2755	29.2	38.2	766.8	757.3
5100	500.45	14.53	-32.82	725.61	11.4	6.49	11.95	8.4821	2752	30.3	40	739.2	732.7
5400	479.48	15.25	-34.8	700.94	11.45	6.45	11.25	8.8886	2750	31.9	40.2	712.8	707.9
5700	459.22	15.9	-36.78	676.92	11.36	6.35	10.65	9.3559	2746	33.7	42.3	688.7	680.7
6000	439.68	16.55	-38.73	653.48	11.28	6.23	10.08	9.8203	2746	35.1	44.8	665.8	656.1

The Python script fills the parameter file input_parameters used by the C program with the input and output directories defined in the file input_pars (used by the combined program), the number of extreme value types (e.g., 4 for the example in Table 3), and the number of data rows (e.g., 101 for the example in Table 3). The input and output directories can be the same or different. Table 4 shows a sample input_parameters file that applies to the example used in Table 3.

Table 4 Sample of the parameter file (input_parameters) used for the example of Table 3. Here the input and output directories are the same, and 4 and 101 are the number of extreme value columns and number of input data rows, respectively.

```
/data/user/Clim_data/PAFA/
/data/user/Clim_data/PAFA/
4 101
```

The Python script then calls the external C program, which in turn produces vertical profiles for the METCM and the user-defined height structures. Table 5 presents the output for the METCM structure for the example in Table 3 (Climate_PAFA_mo1_out_METCM).

Table 5 METCM structured output for the example in Table 3. The midpoint of each layer or zone is listed except for the surface (line or zone 0), which has the surface values. The meteorological data values for each line are weighted means vs. the values at the midpoints.

METCM output for Climate_PAFA_mo1_out									
Line	Height (m)	Wind_Speed (kn)	Temperature (K*10)	Pressure (mb)	Density (g/m3)	w90 (kn)	w95 (kn)	d90_90 (g/m3)	d95_95 (g/m3)
0	0	2.9	2528	994.7	1373.50	3.0	4.1	1452.40	1473.80
1	100	5.3	2548	981.6	1345.57	5.4	6.8	1412.60	1439.50
2	350	10.4	2586	949.4	1281.71	10.2	12.1	1327.72	1362.08
3	750	14.0	2605	900.7	1206.81	13.1	15.2	1250.82	1273.69
4	1250	16.1	2613	843.6	1126.23	15.1	17.4	1165.10	1184.00
5	1750	17.2	2602	789.9	1058.77	16.0	18.6	1089.38	1105.84
6	2250	18.4	2579	739.4	999.62	16.9	20.0	1023.99	1038.85
7	2750	19.8	2552	691.9	945.23	18.1	21.0	965.39	978.19
8	3250	21.2	2522	646.7	893.78	19.5	22.4	909.55	919.74
9	3750	22.7	2491	603.9	845.10	21.1	23.9	858.87	868.92
10	4250	24.5	2459	563.6	798.95	22.8	26.0	813.11	820.17
11	4750	26.6	2426	525.5	755.13	24.7	28.8	768.40	772.27
12	5500	30.0	2377	472.6	693.20	28.5	32.5	705.34	705.58
13	6500	34.3	2312	408.6	616.12	32.7	37.8	627.83	628.04
14	7500	37.7	2252	351.7	544.52	35.7	42.6	557.12	557.48
15	8500	39.7	2207	301.7	476.85	38.0	45.1	491.83	495.59
16	9500	38.7	2187	258.3	412.25	36.6	43.2	432.26	432.34
17	10500	36.5	2190	221.1	352.39	32.6	38.4	375.59	376.60
18	11500	35.0	2202	189.2	300.00	30.3	34.6	316.75	324.54
19	12500	35.2	2211	162.1	255.91	29.5	33.7	268.99	270.70
20	13500	36.3	2214	138.9	218.88	29.8	33.9	229.32	230.08
21	14500	37.7	2214	119.0	187.60	31.1	35.2	195.79	196.99
22	15500	38.8	2212	102.0	160.87	32.6	36.7	167.57	168.84
23	16500	40.2	2210	87.4	137.96	34.6	38.4	143.79	144.40
24	17500	41.6	2208	74.9	118.33	36.9	41.0	122.68	122.64
25	18500	43.2	2206	64.2	101.47	39.0	43.6	104.89	104.64
26	19500	44.7	2206	55.0	87.02	40.9	46.0	89.95	89.27
27	21000	47.5	2204	43.7	69.29	44.8	50.3	71.43	70.65
28	23000	50.9	2206	32.1	50.91	49.3	54.7	52.38	51.63
29	25000	53.5	2213	23.6	37.41	53.0	59.4	38.26	37.66
30	27000	56.1	2221	17.5	27.54	57.0	65.6	27.98	27.51
31	29000	58.6	2230	12.9	20.27	58.9	68.6	20.57	20.11

Tables 6 and 7 present sections of the user-defined height level and layer structures. For the example in Table 3, the filenames are Climate_PAFA_mo1_out_USRLVL and Climate_PAFA_mo1_out_USRMSG, respectively. In Table 7, the midpoint of each layer is listed except for line 0, which has the surface values.

Table 6 Section of the output for the user-defined height level output. Levels from the surface through 3000 m AGL are shown.

USER DEFINED LEVEL OUTPUT for Climate_PAFA_mo1_out									
Level	Height	Wind Speed	Temperature	Pressure	Density	w90	w95	d90_90	d95_95
	(m)	(kn)	(K)	(hPa)	(g/m3)	(kn)	(kn)	(g/m3)	(g/m3)
0	0	2.9	252.8	994.7	1373.50	5.8	8.0	1452.40	1473.80
1	50	4.1	253.8	988.0	1359.53	8.1	10.6	1432.50	1456.65
2	100	5.3	254.8	981.4	1345.57	10.4	13.2	1412.60	1439.50
3	200	7.7	256.7	968.4	1317.63	15.0	18.3	1372.80	1405.20
4	300	10.1	258.6	955.7	1289.70	19.6	23.5	1333.00	1370.90
5	400	11.2	259.0	943.1	1270.73	21.1	24.9	1315.13	1349.10
6	500	12.2	259.5	930.8	1251.77	22.5	26.4	1297.27	1327.30
7	600	13.2	260.0	918.6	1232.80	23.9	27.8	1279.40	1305.50
8	700	13.8	260.3	906.6	1215.27	25.0	29.0	1260.27	1283.93
9	800	14.4	260.7	894.8	1197.73	26.1	30.3	1241.13	1262.37
10	900	15.0	261.0	883.1	1180.20	27.2	31.5	1222.00	1240.80
11	1000	15.3	261.2	871.7	1164.33	27.9	32.3	1205.37	1224.40
12	1100	15.7	261.3	860.3	1148.47	28.6	33.0	1188.73	1208.00
13	1200	16.0	261.5	849.1	1132.60	29.4	33.8	1172.10	1191.60
14	1300	16.2	261.4	838.1	1118.47	29.6	34.1	1156.93	1175.87
15	1400	16.4	261.2	827.2	1104.33	29.9	34.5	1141.77	1160.13
16	1500	16.7	261.1	816.3	1090.20	30.1	34.8	1126.60	1144.40
17	1600	16.9	260.7	805.7	1077.50	30.6	35.3	1111.43	1128.67
18	1700	17.1	260.4	795.2	1064.80	31.0	35.8	1096.27	1112.93
19	1800	17.3	260.0	784.7	1052.10	31.5	36.3	1081.10	1097.20
20	1900	17.6	259.6	774.5	1040.20	31.7	37.0	1067.67	1083.40
21	2000	17.8	259.1	764.3	1028.30	31.9	37.6	1054.23	1069.60
22	2200	18.3	258.2	744.4	1005.14	32.5	38.7	1029.50	1044.40
23	2400	18.8	257.1	724.9	982.62	33.4	39.5	1006.90	1021.60
24	2600	19.3	256.0	705.8	961.10	34.5	40.1	982.77	997.07
25	2800	19.9	254.9	687.3	939.88	35.5	41.0	959.43	972.00
26	3000	20.5	253.7	669.0	918.97	36.5	42.2	936.90	946.40

Table 7 Section of the output for the user-defined layer output. The midpoint heights of the layers are listed except for line 0, which contains the surface values. The meteorological data values for each layer are weighted mean values vs. the values at the midpoints. The surface and layer values up through the 2800–3000 m AGL layer (midpoint at 2900 m) are shown.

USER DEFINED LAYER OUTPUT for Climate_PAFA_mo1_out									
Layer	Height	Wind Speed	Temperature	Pressure	Density	w90	w95	d90_90	d95_95
	(m)	(kn)	(K)	(hPa)	(g/m3)	(kn)	(kn)	(g/m3)	(g/m3)
0	0	2.9	252.8	994.7	1373.5	5.80	8.0	1452.4	1473.80
1	25	3.5	253.3	991.4	1366.52	7.00	9.3	1442.5	1465.23
2	75	4.7	254.3	984.7	1352.55	9.30	11.9	1422.6	1448.07
3	150	6.5	255.7	974.9	1331.6	12.70	15.7	1392.7	1422.35
4	250	8.9	257.6	962.0	1303.67	17.30	20.9	1352.9	1388.05
5	350	10.6	258.8	949.4	1280.22	20.30	24.2	1324.1	1360.00
6	450	11.7	259.3	936.9	1261.25	21.80	25.7	1306.2	1338.20
7	550	12.7	259.7	924.7	1242.28	23.20	27.1	1288.3	1316.40
8	650	13.5	260.1	912.6	1224.03	24.50	28.4	1269.8	1294.72
9	750	14.1	260.5	900.7	1206.5	25.60	29.6	1250.7	1273.15
10	850	14.7	260.9	888.9	1188.97	26.70	30.9	1231.6	1251.58
11	950	15.2	261.1	877.4	1172.27	27.60	31.9	1213.7	1232.60
12	1050	15.5	261.3	866.0	1156.4	28.30	32.7	1197.1	1216.20
13	1150	15.8	261.4	854.7	1140.53	29.00	33.4	1180.4	1199.80
14	1250	16.1	261.4	843.6	1125.53	29.50	34.0	1164.5	1183.73
15	1350	16.3	261.3	832.6	1111.4	29.70	34.3	1149.4	1168.00
16	1450	16.6	261.2	821.8	1097.27	30.00	34.6	1134.2	1152.27
17	1550	16.8	260.9	811.0	1083.85	30.40	35.1	1119.0	1136.53
18	1650	17.0	260.6	800.4	1071.15	30.80	35.6	1103.9	1120.80
19	1750	17.2	260.2	790.0	1058.45	31.30	36.1	1088.7	1105.07
20	1850	17.4	259.8	779.6	1046.15	31.60	36.7	1074.4	1090.30
21	1950	17.7	259.4	769.4	1034.25	31.80	37.3	1061.0	1076.50
22	2100	18.0	258.7	754.3	1016.56	32.10	38.2	1041.3	1056.40
23	2300	18.5	257.7	734.5	993.88	33.00	39.1	1018.2	1033.00
24	2500	19.1	256.6	715.3	971.86	34.00	39.8	994.8	1009.33
25	2700	19.6	255.5	696.6	950.42	35.00	40.5	970.9	984.67
26	2900	20.2	254.3	678.1	929.43	36.00	41.6	948.2	959.20

4. Summary and Conclusion

This report presents a method to process climate data files from the 14 WS for different sites and months. The input climate data files have columns of means and standard deviations of measured and some derived meteorological variables along with numerous columns with extreme values of wind speed and density, which are reduced to a more manageable form using a Python script. An external C program called by the Python script converts the processed climate files to tables that have the height layer (zone) structure of a METCM plus the surface values and/or height levels and layers as defined by the user plus the surface. The user can select one or more of the 36 extreme value types for processing and define the height structure

of the output vertical profiles. The combined program in its present form only processes climate files of the type and format shown in the report. However, with fairly minor modifications, the combined program should be able to process other versions of the climate data files.

5. References

Cogan J. A generalized method for vertical profiles of mean layer values of meteorological variables. Adelphi (MD): Army Research Laboratory (US); 2015 Mar. Report No.: ARL-TR-7434.

Cogan J. Evaluation of model-generated vertical profiles of meteorological variables: method and initial results. Meteorol Appl. 2017;24:219–229.

Headquarters, Department of the Army tactics, techniques, and procedures for field artillery meteorology. Washington (DC): Headquarters, Department of the Army; 2007. Field Manual No.: FM 3-09.15/MCWP 3-16.5.

Reen B. Army Research Laboratory (US), Adelphi, MD. Personal communication, 2017.

INTENTIONALLY LEFT BLANK.

Appendix A. Python Script

This Appendix contains a listing of most of the Python 3.5 script (aka program) that reads in a climate data file from the Air Force 14th Weather Squadron (14 WS), extracts only the user-requested columns of extreme wind speeds and/or densities, and produces vertical height level or layer profiles through the use of an external C program. The large, multiple-application file definition adopted from Reen[1] is not shown, although a series of vertical dots indicates its location within the script. The output profiles may have the height and layer (zone) structure of a computer meteorological message (METCM), height level and layer structures as defined by the user, or all 3. The output based on the METCM has data values for the surface (zone 0) and layers (zones) up through zone 31 (29–30 km). The user-defined profiles are in 2 output files, one for height levels and the second for height layers plus the surface.

```
#!/bin/env python3

import re
import sys
from collections import defaultdict
import string
import statistics
import logging
import subprocess
import os
import shlex
import ntpath
.
.
.
. Location of multi-function definition
.
.

#Read in extreme wind and density values from a parameter file.

with open('input_pars', "r") as fp:
    input_ext = fp.readline() #Read the first line.
    cprog_dir = fp.readline() #Directory where 'C' program is located.
    i_dir = fp.readline()     #Input directory for 'C' program.
    o_dir = fp.readline()     #Output directory for 'C' program.

c_dir = cprog_dir.replace('\n', '')
ext_list = input_ext.split()
ext_length = len(ext_list)

# Read in climate data file.

with open(sys.argv[1], "r") as f:
```

[1]Reen B. Army Research Laboratory (US), Adelphi, MD. Personal communication, 2017.

```
    input_line = f.readline()
    input_data = f.readlines()  # All remaining lines are read in.

print('Reading from file', sys.argv[1])

infile_name = ntpath.basename(sys.argv[1])  # Define output file and put it in 'C' program's input
directory.
i_directory = re.sub('\n','', i_dir)
output_file = i_directory + infile_name + '_out'
#
# Obtain header information from first line of input file.
#
header_list = input_line.split()
in_length = len(header_list)

extreme_list = [] #Set up empty list for extreme values (e.g., w95 or d95_95) on 1st line of data
file.
index_list = []   #Set up empty list for index of where extreme value appears in 1st line of input
file.

print('\nNumber of extremes listed in input_pars = ', ext_length)
for n in range(ext_length):              # Check for valid extreme value types.
  for m in range(in_length):             # Invalid types are ignored.
    if ext_list[n] == header_list[m]:
      extreme_list.append(header_list[m])
      index_list.append(m)
index_list_length = len(index_list)
if ext_length != index_list_length:       # If invalid types in input then note the number of valid
ones.
  ext_length = index_list_length          # and print "error" message.
  print('Number of valid extremes = ', index_list_length)
  print('Number of valid and entered extremes are not equal.  Check input_pars\n ')

data_val = defaultdict(dict)  # Define dictionary variable and a set.
data_values = set()
#
# Process data
#
for currentline in input_data:
  data_list = currentline.split()
  data_len = len(data_list)
  site_name = data_list[0]
  month = data_list[1]
  altitude = data_list[2]

  for m in range(in_length):
    if data_list[m] == '-99999':
      data_list[m] = '-999'
    if m == 3:
      data_val['meanw'][altitude] = data_list[m]
    elif m == 4:
      data_val['meant'][altitude] = data_list[m]
```

```
    elif m == 5:
      data_val['meand'][altitude] = data_list[m]
    elif m == 6:
      data_val['meanp'][altitude] = data_list[m]
    elif m == 7:
      data_val['stdw'][altitude] = data_list[m]
    elif m == 8:
      data_val['stdt'][altitude] = data_list[m]
    elif m == 9:
      data_val['stdd'][altitude] = data_list[m]
    elif m == 10:
      data_val['stdp'][altitude] = data_list[m]
    elif m == 11:
      data_val['obsw'][altitude] = data_list[m]
    for i in range(0, ext_length):
      data_val[extreme_list[i]][altitude] = data_list[index_list[i]]

# Produce a data_values list for altitude.
  data_values.add(float(altitude))

# Then sort data_values (list of altitudes).
sorted_datavalues = sorted(data_values)
#
# Output of revised data file for input to profile convert program.
#
with open(output_file, "w") as fo:
   print("Writing to file: ", output_file, "\n")
   header2_string = ''
   header1_string = '\n{0:30s}\n   {1:7s} {2:7s} {3:7s}  {4:7s}  {5:7s}   {6:7s}  {7:7s} {8:7s}  {9:7s}
{10:7s} '.format(sys.argv[1], header_list[2], header_list[6], header_list[3], header_list[4],
header_list[5], header_list[10], header_list[8], header_list[9], header_list[7], header_list[11])
   for n in range(0, ext_length):
     header2_string = header2_string + ' {0:7s} '.format(extreme_list[n])
   header_string = header1_string + header2_string + '\n'
   fo.write(header_string)

   numlines = 0
   for alt in sorted_datavalues:
      data2_string = ''
      data1_string = '{0:7.0f}  {1:7.2f} {2:7.2f}  {3:7.2f}  {4:8.4f} {5:7.2f} {6:7.2f} {7:7.2f}  {8:8.4f}
{9:6d} '.format(float(alt), float(data_val['meanp'][str(int(alt))]),
float(data_val['meanw'][str(int(alt))]), float(data_val['meant'][str(int(alt))]),
float(data_val['meand'][str(int(alt))]), float(data_val['stdp'][str(int(alt))]),
float(data_val['stdt'][str(int(alt))]), float(data_val['stdd'][str(int(alt))]),
float(data_val['stdw'][str(int(alt))]), int(data_val['obsw'][str(int(alt))]))
      for n in range(0, ext_length):
        data2_string = data2_string + ' {0:7.2f}
'.format(float(data_val[extreme_list[n]][str(int(alt))]))
      data_string = data1_string + data2_string + '\n'
      fo.write(data_string)
      numlines = numlines + 1
```

```
# Name of logging file
climate_log_name = "/data/jcogan/pyfiles/clim.log"
# Open and configure logging file
logging.basicConfig(format='%(asctime)s
%(levelname)s:%(message)s',filename=climate_log_name,
                filemode='w',level=logging.DEBUG)

#Get output file name.
out_list = re.split('/', output_file)
out_list_element = len(out_list)-1
output_file = out_list[out_list_element]

#Define arguments for running external program.
command_to_execute = ['./convertclim', output_file]
name_of_command = 'Diagnostic_out'
string_indicating_success = '***** COMPLETED ALL SELECTED OUTPUT TYPES. *****'
in_param = c_dir + 'input_parameters'   #Location and name of parameter file for 'C' program.

try:
  # Load the input_parameters file for the 'C' program.
  with open(in_param, "w+") as po:
    in_string = i_dir + '\n'
    po.write(in_string)
    out_string = o_dir + '\n'
    po.write(out_string)
    extr_lines_string = str(ext_length) + ' ' + str(numlines) + '\n'
    po.write(extr_lines_string)

  # Run the external 'C' program.
  run_external_program(command_to_execute, name_of_command, c_dir,
                  string_indicating_success)
  print("Selected output types generated from climate data.\n\n")
except:
  print("\nCannot run external program.\n")
  print("Check external program path, c-dir, or other argument.\n\n")
```

INTENTIONALLY LEFT BLANK.

Appendix B. Code for Modified Parts of the C Program

This Appendix presents parts of the code from the C program that were modified for the purpose of processing the climatological data files after modification by the parent Python script. The routines (C functions) for output of user defined profiles are not listed here since they are closely similar to the one for output of profiles that have the computer meteorological message (METCM) height structure (Appendix B-2). The other routines are the same as those described in Cogan.[1,2] The code without the modifications for processing the climate files from the 14WS is available at the US Army Research Laboratory GitHub site, https://github.com/USArmyResearchLab. Although not shown, the sections of code in the main routine (convertdata_clim.c) that related to the generation of messages other than the METCM and the user-defined profiles were commented out. They were not needed for the work for this report, but could be reinstated by removing the symbols denoting comments (/* and */). Also, an additional print statement was added at the end of that routine for use by the logging procedure of the Python script to indicate successful completion of the C program.

B-1 Data Input

The routine or function to read the data was modified to read the several data types including a varying number of extreme wind and/or density conditions. The listing shown here does not include several commented-out print statements that were available for debugging and checking intermediate values.

```
/* FILE NAME: readclim.c.
Reads climate data from the AF 14th WS in tabular format. This program produces input for the
programs that produce level and layer profiles in various tabular formats. This function reads in
one file in the climate data format generated by the climstat_all.py Python3 script.  It can read in
any number of met data lines. However, for processing into an output sounding at least two lines
are needed, one of which normally is the surface. The upper limit is the size of the arrays as
defined in the appropriate structure definition.
*/

#include "convert_clim2.h"

int readclim(struct sound *clim, char *inputfilename)
 {

  FILE *fin;
  int n, len;
```

[1]Cogan J. A generalized method for vertical profiles of mean layer values of meteorological variables. Adelphi (MD): Army Research Laboratory (US); 2015 Mar. Report No.: ARL-TR-7434.

[2]Cogan J. Evaluation of model-generated vertical profiles of meteorological variables: method and initial results. Meteorol Appl. 2017;24:219–229.

```
  char label[81], ext_type[2];
  char site_id[6], site[35], sitename[141];
  float hgt_dif, extreme_value;
  int i, k;


/* Open input data file.  ******/
   /*printf("Input file name: %s \n", inputfilename);*/
   if(!(fin = fopen(inputfilename,"r")))
    {
      fprintf(stderr,"\nUnable to open clim input data file.\n\n");
      exit(1);
    }

 /* Set indices for read statements.*/

   n = clim->ind + 10;

 /* Set header values for ceiling and visibility equal to ERROR (-999) to avoid confusion with
   possible real values (vs. 0.0 for each), since the climate data do not include these variables.
****/

    clim->site.ceil = ERROR;
    clim->site.vis = ERROR;

 /* Read in header information. ******************************/

      fscanf(fin, "%s\n", sitename);

      i=0;
      do
       {
         i++;
         fscanf(fin, "%s", label);
       }
      while(strcmp(label,"obsw") != 0);

     for(i=0;i<clim->ind;i++)
         fscanf(fin,"%s", clim->site.extrvar[i]);

      strcpy(clim->site.time_id,"GMT");  /* set the time_id for format reasons */

 /*** End of optional header reading. Common format of data follows. *********/
 /*** Read data lines. ********/

   i = 0;
   while(i<clim->numlines)
    {
      fscanf(fin,"%f%f%f%f%f%f%f%f%f%d", &clim->level[i].hgt, &clim->level[i].prs, &clim-
>level[i].spd, &clim->level[i].tmp, &clim->level[i].den, &clim->level[i].sdp, &clim->level[i].sdt,
&clim->level[i].sdd, &clim->level[i].sdw, &clim->level[i].lvlnum);
```

```
    for(k=0; k<clim->ind; k++)
     {
      fscanf(fin,"%f", &extreme_value);
      sscanf(clim->site.extrvar[k], "%1s", ext_type);
      if(strcmp(ext_type, "w") == 0)
        clim->extrval[i][k].ext_wind = extreme_value;
      else
        clim->extrval[i][k].ext_dens = extreme_value;
     }
    i++;
   }

  clim->nht = i;

 /* No need to convert MSL heights to AGL. Already in AGL.*************************/

 /* Convert temperature from C to K.  *************************/

  for (i=0;i<clim->nht;i++)
     clim->level[i].tmp += 273.16;

 /* Latitude, longitude, and elevation are not provided in input climate files. */

  clim->site.lat = -999;
  clim->site.lon = -999;
  clim->site.elev = -999;

 printf("\n");

/*************** END OF DATA INPUT ******************************/

 fclose(fin);

 return;
}
```

B-2 METCM Type Output

The routine or function to produce output with the height structure of a METCM was modified to output the several data types including a varying number of extreme wind and/or density conditions. The listing shown here does not include several commented-out print statements that were available for debugging and checking intermediate values. Since no wind direction values are in the climate files from the Air Force 14th Weather Squadron (14 WS) no General Trajectory (GTRAJ) model compatible output was generated. Very similar write functions not shown here produce output for a user-defined height structure.

```
/**   This function (writemetcm) prints output with the height structure of the METCM to
appropriate files.
      Input climate data was taken from AF 14th WS files. This version processes any number of
extreme values for wind and/or density as determined via the parameter file
"input_parameters".  This version (in file writemetcm_clim.c) does not produce GTRAJ
       compatible files since the climate files do not have wind direction.
**/

#include "convert_clim2.h"
#define MAXCMLVL 32

int writemetcm(struct sound *metcm, char *outpath)
{
  FILE *fcmout;
  int i, k, size;
  float zone, height;
  char outfile[161], infilename[71];
  char ext_type[2];

 /* Get output file name. ********************/

   strcpy(outfile, outpath)
   sscanf(metcm->site.filename, "%29s", infilename);
   strcat(outfile, infilename);
   strcat(outfile, "_METCM");

 /* Open output and GTRAJ input files.  ***********************************/

    if(!(fcmout = fopen(outfile,"w")))
     {
      fprintf(stderr,"Unable to open metcm data file.\n");
      exit(1);
     }

  /* Print output to files. ***********************************/

    size = metcm->nht+1; /*** loop indices ***/

  /*   METCM output **********************/

   fprintf(fcmout,"METCM output for %s \n\n", metcm->site.filename);

   /* No date or time or location or elevation information in climate data files, but code was
saved in case such becomes available. At this time no need to print missing data indicators for
those      parameters. */

   fprintf(fcmout,"Elevation: %7.1f\n\n",
                metcm->site.elev);*/

   /* No wind direction in climate files used for this evaluation.  Removed wind direction
output.*/
```

```
   fprintf(fcmout,"        (m)  (tens_of_mils)    (kn)      (K*10)      (mb)      (g/m3)      (kn)      (kn)
(g/m3)     (g/m3)\n\n");*/
   fprintf(fcmout," Line   Height   Wind_Speed  Temperature  Pressure   Density    ");
   for(k=0;k<metcm->ind;k++)
        fprintf(fcmout, "   %s ", metcm->site.extrvar[k]);
   fprintf(fcmout, "\n");

   fprintf(fcmout,"        (m)      (kn)      (K*10)      (mb)      (g/m3)     ");
   for(k=0;k<metcm->ind;k++)
    {
      sscanf(metcm->site.extrvar[k], "%1s", ext_type);
      if(strcmp(ext_type, "w") == 0)
       fprintf(fcmout, "  (kn)  ");
      else
       fprintf(fcmout, " (g/m3)  ");
    }
   fprintf(fcmout, "\n\n");

   for(i=0;i<size;i++)
    {
      if(metcm->level[i].spd != ERROR)
         metcm->level[i].spd *= NM;  /* Input wind speeds in m/s; convert to kn. */

      for(k=0;k<metcm->ind;k++)
       {
        sscanf(metcm->site.extrvar[k], "%1s", ext_type);
        if(strcmp(ext_type, "w") == 0)
         {
           if(metcm->extrval[i][k].ext_wind != ERROR)
             metcm->extrval[i][k].ext_wind *= NM;
         }
       }

      if(metcm->level[i].tmp != ERROR)
        metcm->level[i].tmp *= 10;

      if(i == 0)
       height = metcm->level[i].hgt;
      else
       height = (metcm->level[i].hgt + metcm->level[i-1].hgt)*0.5;

      /* No wind direction in climate files used for this evaluation.  Therefore no wind direction
output.*/

      fprintf(fcmout,"%3d   %8.0f   %7.1f    %8.0f     %7.1f    %8.2f   ",
            i, height, metcm->level[i].spd, metcm->level[i].tmp, metcm->level[i].prs,
            metcm->level[i].den);

            for(k=0;k<metcm->ind;k++)
       {

         sscanf(metcm->site.extrvar[k], "%1s", ext_type);
```

```
          if(strcmp(ext_type, "w") == 0)
            fprintf(fcmout, " %7.1f", metcm->extrval[i][k].ext_wind);
          else
            fprintf(fcmout, " %8.2f ", metcm->extrval[i][k].ext_dens);
        }
       fprintf(fcmout, "\n");
   }

 fprintf(fcmout, "\n");

 printf("\n MET message printed to METCM output.\n");

/********** End of output statements. -**********************/

  fclose(fcmout);

  return;
}
```

B-3 Generation of Level and Layer Values

The program for generation of level and layer values was modified to process the climatological profiles including the ones for extreme wind and density. The values for mean pressure were processed in the same way as in Cogan[2] for height-based profiles. Much of this routine (msgvaluesclim.c) was not changed, but since the changes affect several sections of the code, the entire file is presented.

```
/** FILE NAME: msgvaluesclim.c
  This program computes level and layer values using input climate data files from   the 14 WS as
modified by the climstat_all.py Python3 script.  The   pressure values are computed, as
previously, with hypsometric method from observation level immediately below, if within the
layer, or from bottom level of that layer. No wind   direction values are in the climate files.  Wind
direction is given the missing data   indicator (-999) vs. removal from this routine so as to have a
place holder in case later climate files include wind direction.  **/

#include "convert_clim.h"

int msgvalues(struct sound *snd, struct sound *msg, struct sound *mlevel)
{
                        /* msg = layer values, mlevel = level values */
  int i, j, k;
  int size, msize, tsize, wsize;  /*tsize is for temperature, wsize for wind (here the same) */
  float zmax, dir;
  float t, t0, z, z0, z_ratio;  /* No virtual temperature.  Use sensible temperature. */
  char ext_type[2];

  struct temporary *sound;  /* use in level and layer functions */
```

```
  struct temporary *leveltemp;
  struct temporary *layertemp;

  sound = (struct temporary *)malloc(sizeof(struct temporary));
  leveltemp = (struct temporary *)malloc(sizeof(struct temporary));
  layertemp = (struct temporary *)malloc(sizeof(struct temporary));

/* Initialize structure and temporary variables *********************************/

    for(i=0;i<MAXSIZE;i++)
     {
      sound->h[i] = ERROR;
      sound->t[i] = ERROR;
      sound->tv[i] = ERROR;
      sound->p[i] = ERROR;
      sound->den[i] = ERROR;
      sound->spd[i] = ERROR;
      sound->dir[i] = ERROR;
      sound->exwind[i] = ERROR;
      sound->exdens[i] = ERROR;
      leveltemp->h[i] = ERROR;
      leveltemp->t[i] = ERROR;
      leveltemp->tv[i] = ERROR;
      leveltemp->p[i] = ERROR;
      leveltemp->den[i] = ERROR;
      leveltemp->spd[i] = ERROR;
      leveltemp->dir[i] = ERROR;
      leveltemp->exwind[i] = ERROR;
      leveltemp->exdens[i] = ERROR;
      layertemp->h[i] = ERROR;
      layertemp->t[i] = ERROR;
      layertemp->tv[i] = ERROR;
      layertemp->p[i] = ERROR;
      layertemp->den[i] = ERROR;
      layertemp->spd[i] = ERROR;
      layertemp->dir[i] = ERROR;
      layertemp->exwind[i] = ERROR;
      layertemp->exdens[i] = ERROR;
      for(k=0;k<snd->ind;k++)
       {
         sound->extrtemp[i][k].ext_wind = ERROR;
         sound->extrtemp[i][k].ext_dens = ERROR;
         leveltemp->extrtemp[i][k].ext_wind = ERROR;
         leveltemp->extrtemp[i][k].ext_dens = ERROR;
         layertemp->extrtemp[i][k].ext_wind = ERROR;
         layertemp->extrtemp[i][k].ext_dens = ERROR;
       }
     }

/* Parameters for level and layer values. ****************/

  msg->nht = mlevel->nht - 1;  /* number of layer values one less than level values */
```

```
  msg->ind = snd->ind;
  mlevel->ind = snd->ind;
  size = snd->nht;
  msize = mlevel->nht;
  zmax = snd->level[size-1].hgt + 0.001;

/* Cannot compute components from snd wind speed and direction (no direction values).
Cannot compute virtual temperature from the climate input data (no humidity values).
**********/

   /******* Compute level values. ************************************/

  j=-1;           /*set up temporary variables for use in level and layer functions as needed */
  for (i=0;i<size;i++)
   {
    if(snd->level[i].hgt != ERROR && snd->level[i].tmp != ERROR &&
          snd->level[i].prs != ERROR)
     {
      j++;
      sound->h[j] = snd->level[i].hgt;
      sound->t[j] = snd->level[i].tmp;
      sound->tv[j] = sound->t[j];         /*NO HUMIDITY value in climate data set.  Set tv to t. */
      sound->p[j] = snd->level[i].prs;
      sound->den[j] = snd->level[i].den;
      sound->spd[j] = snd->level[i].spd;
      for(k=0;k<snd->ind;k++)
       {
        sscanf(snd->site.extrvar[k], "%1s", ext_type);
        if(strcmp(ext_type, "w") == 0)
         sound->extrtemp[j][k].ext_wind = snd->extrval[i][k].ext_wind;
        else
         sound->extrtemp[j][k].ext_dens = snd->extrval[i][k].ext_dens;
       }
     }
   }

  tsize = j; /*printf("tsize = %4d\n\n", tsize);*/
 wsize = tsize;

/* Load in height values. ****/

  for (i=0;i<msize;i++)
   {
    leveltemp->h[i] = mlevel->level[i].hgt;
    layertemp->h[i] = leveltemp->h[i];
   }

/* Convert 2D arrays to 1D for entry into level and layer functions. Then compute level and layer
values.  Convert back to 2D arrays.*/

  for(k=0;k<snd->ind;k++)
   {
```

```
    sscanf(snd->site.extrvar[k], "%1s", ext_type);
    if(strcmp(ext_type, "w") == 0)
     {
      for(i=0;i<size;i++)     /* Wind speed */
          sound->exwind[i] = sound->extrtemp[i][k].ext_wind;  /* Form 1D arrays from 2D arrays.
*/

      /* Compute level and layer values for the kth type of extreme wind speeds. */
       level(zmax, msize, leveltemp->h, sound->exwind, leveltemp->exwind, sound->h);
       layer(zmax, wsize, msize, layertemp->h, sound->exwind, leveltemp->exwind, layertemp-
>exwind, sound->h);

      /* Enter level and layer values "back" into 2D arrays.*/
       for(i=0;i<size;i++)
         {
          leveltemp->extrtemp[i][k].ext_wind = leveltemp->exwind[i];
          layertemp->extrtemp[i][k].ext_wind = layertemp->exwind[i];
         }
     }
    else
     {
      for(i=0;i<size;i++)     /* Density */
        sound->exdens[i] = sound->extrtemp[i][k].ext_dens;  /* Form 1D arrays from 2D arrays.
*/

     /* Compute level and layer values for the kth type of extreme densities. */
       level(zmax, msize, leveltemp->h, sound->exdens, leveltemp->exdens, sound->h);
       layer(zmax, tsize, msize, layertemp->h, sound->exdens, leveltemp->exdens, layertemp-
>exdens, sound->h);

     /* Enter level and layer values "back" into 2D arrays.*/
       for(i=0;i<size;i++)
         {
          leveltemp->extrtemp[i][k].ext_dens = leveltemp->exdens[i];
          layertemp->extrtemp[i][k].ext_dens = layertemp->exdens[i];
         }
     }
   }

 /* Compute level values for other non-pressure values. *************/

  level(zmax, msize, leveltemp->h, sound->t, leveltemp->t, sound->h);
  level(zmax, msize, leveltemp->h, sound->den, leveltemp->den, sound->h);
  level(zmax, msize, leveltemp->h, sound->spd, leveltemp->spd, sound->h);

 /* Compute layer values for other non-pressure values. *************************/

  layer(zmax, tsize, msize, layertemp->h, sound->t, leveltemp->t, layertemp->t, sound->h);
  layer(zmax, tsize, msize, layertemp->h, sound->den, leveltemp->den, layertemp->den, sound-
>h);
  layer(zmax, wsize, msize, layertemp->h, sound->spd, leveltemp->spd, layertemp->spd, sound-
>h);
```

```
  /* No input wind direction information. Set level and layer values to ERROR (-999).
*****************/

   for(i=0;i<msize;i++)
       mlevel->level[i].dir = ERROR;

    msg->level[0].spd = snd->level[0].spd;
    msg->level[0].dir = ERROR;

    for(i=1;i<msize;i++)
        msg->level[i].spd = layertemp->spd[i-1];

 /* Load height values into msg structures (level values already via mlevel).  **********/

  for (i=0;i<msize;i++)
      msg->level[i].hgt = mlevel->level[i].hgt;

 /* Load values into mlevel and msg structures.  ****************************/

  for (i=0;i<msize;i++)                    /***** level values *****/
    {
      mlevel->level[i].tmp = leveltemp->t[i];
      mlevel->level[i].vtmp = mlevel->level[i].tmp; /*NO HUMIDITY values in climate data set.  Set
vtmp = tmp. */
      mlevel->level[i].den = leveltemp->den[i];
      mlevel->level[i].spd = leveltemp->spd[i];
      mlevel->level[i].dir = ERROR;
      for(k=0;k<snd->ind;k++)
       {
         mlevel->extrval[i][k].ext_wind = leveltemp->extrtemp[i][k].ext_wind;
         mlevel->extrval[i][k].ext_dens = leveltemp->extrtemp[i][k].ext_dens;
       }
    }

  msg->level[0].tmp = snd->level[0].tmp;       /***** layer values (= surface + layers) *****/
  msg->level[0].vtmp = msg->level[0].tmp;       /*NO HUMIDITY values in climate data set.  Set
vtmp = tmp. */
  msg->level[0].den = snd->level[0].den;
  msg->level[0].spd = snd->level[0].spd;
  for(k=0;k<snd->ind;k++)
    {
      msg->extrval[0][k].ext_wind = snd->extrval[0][k].ext_wind;
      msg->extrval[0][k].ext_dens = snd->extrval[0][k].ext_dens;
    }

  for (i=1;i<msize;i++)
    {
      msg->level[i].tmp = layertemp->t[i-1];
      msg->level[i].vtmp = msg->level[i].tmp;  /*NO HUMIDITY values in climate data set.  Set
vtmp = tmp.  */
      msg->level[i].den = layertemp->den[i-1];
```

```
      msg->level[i].spd = layertemp->spd[i-1];
      for(k=0;k<snd->ind;k++)
       {
         msg->extrval[i][k].ext_wind = layertemp->extrtemp[i-1][k].ext_wind;
         msg->extrval[i][k].ext_dens = layertemp->extrtemp[i-1][k].ext_dens;
       }
    }

  /* Pressure values. ********************************************************/
  /* Compute level pressure values using better hypsometric method. This part was modified for
case of few observation levels and large vertical gap(s) between observation levels
******************/

  mlevel->level[0].prs = snd->level[0].prs;

  j=0;
  for(i=1; i<msize; i++)       /*Values for temporary sound structure set before level calculations
above.*/
    {
     while(sound->h[j] < mlevel->level[i].hgt)
        j++;

     if(sound->h[j] == mlevel->level[i].hgt)
      mlevel->level[i].prs = sound->p[j];
     else
      {

        if(sound->h[j-1] < mlevel->level[i-1].hgt && j > 0)
           mlevel->level[i].prs = presscomp(mlevel->level[i].vtmp, mlevel->level[i-1].vtmp,
                          mlevel->level[i-1].prs, mlevel->level[i].hgt, mlevel->level[i-1].hgt);
        else
           mlevel->level[i].prs = presscomp(mlevel->level[i].vtmp, sound->tv[j-1],
                          sound->p[j-1], mlevel->level[i].hgt, sound->h[j-1]);
      }
    }

  /* Compute layer pressure values.
************************************************/

  msg->level[0].prs = mlevel->level[0].prs;     /* msg values (= surface + layers)*/

  j=1;
  for(i=1;i<msize;i++)
   {
       while(sound->h[j] < (mlevel->level[i].hgt + mlevel->level[i-1].hgt)*0.5)
       j++;

     if(sound->h[j] == (mlevel->level[i].hgt + mlevel->level[i-1].hgt)*0.5)
      msg->level[i].prs = sound->p[j];
     else
      {
       /* Only have sensible temperature available.  No humidity data in climate input. */
```

```
        if(sound->h[j-1] < (mlevel->level[i-1].hgt + mlevel->level[i-2].hgt)*0.5)
         {
           t = msg->level[i].tmp;
           t0 = mlevel->level[i-1].tmp;
           z = (mlevel->level[i].hgt + mlevel->level[i-1].hgt)*0.5;
           z0 = mlevel->level[i-1].hgt;
           msg->level[i].prs = presscomp(t, t0, mlevel->level[i-1].prs, z, z0);
         }
        else
         {
           t = msg->level[i].tmp;
           t0 = sound->t[j];
           z = (mlevel->level[i].hgt + mlevel->level[i-1].hgt)*0.5;
           z0 = sound->h[j-1];
           msg->level[i].prs = presscomp(t, t0, sound->p[j-1], z, z0);
         }
       }
    }

  /*printf("Ending at computation of layer pressure values.\n\n");exit(0);*/

  /* Load in site information (date, time, lat, lon, etc.). ***********/

 msg->site = snd->site;
 mlevel->site = snd->site; /* For use with level output as needed.*/

/* Free temporary arrays. *********/

 free(sound);
 free(leveltemp);
 free(layertemp);

 return;

}  /* End of level and layer computation section. *****************/
```

List of Symbols, Abbreviations, and Acronyms

14 WS	Air Force 14th Weather Squadron
AGL	above ground level
csv	comma-separated value
d	density
GTRAJ	General Trajectory
hgt	height
KTUS	Tucson International Airport
METCM	computer meteorological message
mo	month
obsw	wind speed observations or samples
p	pressure
PAFA	Fairbanks, Alaska
std	standard deviation
t	temperature
T&E	test and evaluation
w	wind speed
YBBN	Brisbane, Australia

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIR ARL IMAL HRA RECORDS MGMT RDRL DCL TECH LIB
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	ARL RDRL WMP E J COGAN

INTENTIONALLY LEFT BLANK.