EARTH OBSERVING LABORATORY

Memorandum:

26 February 2018

TO:	WindInSOCRATES file
FROM:	Al Cooper
SUBJECT:	Workflow for WindInSOCRATES.Rnw

This document contains a reasonably detailed description of the development of algorithms for processing data from the pitot-static sensor and the gust pod on the GV during SOCRATES. That development does not need much elaboration in this document, but some functions were developed in the process of constructing this document and some supporting explanation will help anyone wanting to duplicate or extend this work to understand how they operate. This workflow therefore focuses mostly on those functions and a few related aspects like the sources of data used in the report.

Sources of Data

At this time, only the data files produced in the field during the SOCRATES project are available. These are labeled as "Preliminary" and should not be used for final analyses, so it will be important to revisit this analysis when final production files are available. In the meantime, to facilitate reproducibility, subset data files as used by this program are saved in the EOL directory ~cooperw/RStudio/Reprocessing, with the names AKRDdata.Rdata, AKRDforSOCRATES.Rdata, DataM.Rdata, and AKRD-fit-coef.Rdata. These files are too large to include in the GitHub repository, and EOL data-access rules prevent that until the data are released for public access so only internal-to-EOL personnel with responsibility for data processing for the project should use these files until they are released in "Production" format.. The program has the code to reconstruct these files from the temporary EOL data repository (/scr/raf_data/SOCRATES, etc.), and later could do so from the released data files. The data files may change as EOL personnel review them and correct errors in or make changes in processing (e.g., perhaps by reducing the presence of spikes in the high-rate data that affect variance spectra); that is one reason for caution when using these data files.

Special Functions

Some functions were isolated to the 'chunks' directory because they were expected to be of further use, for example by processing routines that implement the algorithms recommended in the "WindInSOCRATES" report. Some will be incorporated into the "Ranadu" package of R routines in the future, and others are re-used in the "QAtools" shiny app that provides temporary data processing during and shortly after the field project for quality assessment.

VSpec(.data, .Variable)

Plots of spectral variance (often called "variance spectra" here) are used extensively in this report, so a separate function was used to facilitate construction of those plots. In its simplest form, "VSpec" constructs a frequency-weighted variance spectrum from a single variable ".Variable" contained in a supplied data.frame (".data") and returns a "ggplot2" plot specification suitable for printing. The returned specification can also be saved so that subsequent calls can add additional spectra to the plot. This usage would follow these forms:

```
VSpec(Data, 'TASX')    ## a single spectrum
## for three spectra on one plot:
g <- VSpec (Data, 'QCF')
g <- VSpec (Data, 'QCTC', ADD=g)
VSpec (Data, 'QC_GP', ADD=g)
```

A set of default specifications will be used in that case. A description of the available parameters and options is contained in the header information for the function. It is planned that this function will be included in "Ranadu", at which time the normal help command for the function ("?Ranadu::VSpec) will provide information on the options and defaults. In the meantime, the following will serve as a partial description of the function:

- ".data" must be a data.frame containing the variable ".Variable" as one among possibly many columns. The data.frame must also contain the "Time" variable and should have a "Rate" attribute; this is the case for all data.frames constructed by Ranadu::getNetCDF.
- ".Variable" should be specified as a character variable containing the name of the variable (e.g., .Variable='TASX')
- The default variance spectrum is constructed using the "spectrum()" function of R. This function accepts a smoothing argument named "span", with a default value in VSpec() of 49. This can be changed by setting the calling argument via spans=NewValue. In effect, centered-average values of "spans" length are averaged to produce a smooth spectrum, so larger numbers produce more smoothing.
- The routine will also produce a "maximum entropy method" (MEM) spectrum if the "type" argument is set to anything other than "spectrum" (the default). In this case, the Ranadu functions memCoef() and memEstimate() are used to construct the spectrum. Two additional arguments are relevant in this case: "poles" (the number of poles to use in the MEM estimate), with default value 50, and "resolution", which specifies the fraction of the logarithm of the frequency range) between successive evaluations of the MEM estimate of spectral variance, with default value 0.0001 leading to evaluation at 10,000 discrete frequencies. A small number of poles gives a smoother spectrum; a small value for "resolution" is needed to identify possible sharp spikes in the spectrum.

- Additional smoothing can be provided for either type of spectrum by specifying a value for the “smoothBins” argument. The default is 0, and this value or any other value less than or equal to 9 leads to no smoothing. For other values, the spectrum is binned into “smoothBins” bins evenly spanning the displayed logarithmic frequency range and spectral estimates are averaged within each bin before the plot is constructed. Often a value of 50 or 100 is useful.

- There is a faint orange background of lines in the generated plot that represent a -5/3 logarithmic slope or, for the frequency-weighted spectrum, a -2/3 slope. For measurements representing components of the wind (like WIC or VYC) the reference lines are constructed so that there is a factor-of-10 difference in eddy dissipation rate between the lines and the line representing $10^{-4}\mathrm{m}^2\mathrm{s}^{-3}$ is shown as a heavier dotted line. This uses a default value of a parameter “ae” of 0.2. This should be changed to ae=0.15 for a longitudinal spectrum like TASX or UXC to shift the reference lines to account for the expected 4:3 ratio between the expected magnitudes of the longitudinal and lateral spectra in an inertial subrange.

- An additional argument “VLabel” can be used to change to label used in the legend for the specified variable. The default is the name of the supplied “.Variable”. This can be useful to supply a more descriptive name than the short name used in the netCDF file; for example “VSpec (Data, ’UXC”, VLabel=’longitudinal wind component [m/s]’).

- The final argument is “ADD” with default value NA. If this argument is supplied, it should provide a variable containing a previously defined ggplot2 specification of a plot, in which case the new spectrum will be added to the existing plot. This can be used to plot up to four variance spectra on a single plot.

- The default ggplot2 theme is used unless you specify another. I usually add “theme_WAC()” to the final plot specification before plotting to use the theme I prefer.

Future modifications may add the capability to plot the spectrum generated by “bspec::bspec()”, which divided the spectra into segments and averages the spectra from the segments in order to reduce the variance in the estimate, following the method developed (I think) by Welch. It would be desirable to add options to change the axis limits and y-axis label as well, so this may be part of the eventual Ranadu function. For this report, however, the version residing in the directory “chunks” and saved in the GitHub repository with this archive file was used. For additional information on variance spectra and the R routines used, see this tutorial: https://ncar-eol.shinyapps.io/VarSpec/.

MergeMatt()

This function was used to combine the processed line-of-sight beam speeds in Matt Hayman’s processing into the standard netCDF file produced by normal “nimbus” processing, optionally processed also by the Python script “LAMS_ARISTO.py” discussed below. The reason for needing a special step (vs. normal data.frame merging) was that the “Time” variable in Matt’s file was not the usual variable used in nimbus-produced netCDF files because it lacked data information, so some additional processing was needed to revise the time variable. This function was only used once and the resulting merged data.frame was saved for future use, so this is suppressed in the "WindInSOCRATES.Rnw" file by use of the "eval=FALSE" parameter in the "merge" program chunk.

processWind()

This function was used to calculate wind from the LAMS-provided beam speeds. It used the variables "V_LOS_Beam{1,2,3,4}" produced by Matt Hayman, although it would optionally use the beam speeds produced by the Python routine using Savitzgy-Golay polynomials. As used, this function produced the wind vector in cartesian coordinates from beams 1–3 and did not use beam 4, which was generally weak and questionable in this project and was not included in Matt Hayman's processing. Once a relative-wind vector was determined from the LAMS beam speeds, that vector was combined with the CMIGITS measurements of attitude angles and Earth-relative aircraft motion to calculate the wind and return a data.frame with variables representing the airspeed and wind components.

AddWind(DF)

This function implements the processing developed in this report by adding new wind variables based on the pitot-static sensor and, separately, on the gust pod. There are several additional arguments that can control which variables are added, but all are TRUE by default and so add all these results to the supplied data.frame. All of these rely on the Ranadu function "WindProcessor()". The arguments to AddWind(), in addition to the data.frame, are:

addAKY: Add the angle of attack based on the complementary-filter algorithm described in Section 1.2 of the report, with the name **AKY.** This option also adds a variable **WIY,** the vertical wind calculated using the new angle-of-attack.

addGP: Add the wind measurements based on the gust pod, using the empirical representation developed in this report. The new variables are **AK_GP, SS_GP, TASG, WDG, WSG, WIG, UXG, VYG,** representing respectiveley the angles of attack and sideslip, the airspeed, the wind direction and speed, the vertical wind, and the longitudinal and lateral components of the horizontal wind.

addTC: Add wind measurements and a new airspeed measurement based on measurements from the pitot-static sensor. New variables are **TASTC, WDTC, WSTC, WITC, UXTC, VYTC,** with meanings analogous to the basic measurements with X or C suffixes in place of TC.

addROC: Add a rate-of-climb variable **ROC** representing the vertical speed of the aircraft as determined by integration of the hydrostatic equation.

In addition, it is possible to supply a “Rate” argument to handle the case where the data.frame does not have this attribute. If this attribute is included in the data.frame, the “Rate” argument is overridden by the value of the attribute.

This function is also used by the QAtools shiny app to support the addition of these variables to a netCDF file. The function itself does not do this; it only adds these variables to the R data.frame. The routine also makes use of SplitDV() so that function must be available when it is called. It also uses several Ranadu functions so that package should be available when it is called.

removeSpikes(v)

To support the removal of spikes, this routine calculates the rolling mean and standard deviation (width 99) of a variable “v”, identifies values of the variable that differ from the rolling mean by more standard deviations than a limit specified by sdLimit (default 4), and replaces those values by interpolation in the returned variable. This may also be added to Ranadu, but the archived version in “chunks” is the specific one used in thie report.

The Python routine LAMS_ARISTO.py

This routine is replaced now by Matt Hayman’s processing, especially now that his variables include a measure of uncertainty, so this routine will likely not be used in the future and it is not essential to the production of the present report. Nevertheless, the following provides some documentation in case it is ever needed again, because for this work it had to be retrieved from a previous version and modified, involving considerable work that might be reduced by the following guide to how to use this routine:

1. On tikal, copy the routine ~cooperw/ARISTO-2017/LAMS_ARISTO.py to another directory where you have write permission.
2. On lines 254 and 255, change the names of the data file and the directory containing the file to be processed. Do not include the trailing “.nc” in the file name. Save the edited file. CAUTION: This routine will create a new file in that same directory as the existing file with “LAMS” appended to the file name, and if that file is present it will be overwritten. This won’t overwrite Matt’s files because it adds “LAMS” instead of “_LAMS”, but if his files ending in “_LAMS” are used the new file will end in “_LAMSLAMS”. In that case, the new file will contain the line-of-sight speeds obtained by both algorithms, in Matt’s case with the names Beam1_LAMS – Beam4_LAMS and in the case of the Python program with names BEAM1speed – BEAM4speed. Note, however, that these were not used in the present report except, when present, for comparison tests. Instead, Matt’s variables “V_LOS_Beamx” (with x={1,2,3}) were be used.

3. Run the program using the command "python LAMS_ARISTO.py". There will be regular messages during processing, ending (if successful) with the message "Reached end of routine ...". This will typically take a few minutes, maybe as much as 5 min. If this fails immediately, there may be a problem with the available python packages that need updating or changing. In particular, the "netCDF4" python package must be present.

4. On successful completion, there should then be a new file with a name ending in "LAMS.nc" in the prescribed working directory. It should be slightly larger than the original file because a few variables are added containing the line-of-sight LAMS speeds and some derived quantities. These are then the netCDF files that were read by the next routine (the processor associated with this document) to produce the final document,.

– End of Memo –