16 September 2019

TO: Ranadu documentation
FROM: Al Cooper
SUBJECT: Ranadu manual and tutorial
VERSION: Ranadu_2.5-19-08-27

1 Introduction

This manual describes an R package named "Ranadu" (R-based analysis for aircraft data users), which is intended to facilitate use of the data archives produced by the data systems on the NSF/NCAR/EOL/RAF research aircraft. (NSF=National Science Foundation; NCAR=National Center for Atmospheric Research; EOL=Earth Observing Laboratory; RAF=Research Aviation Facility) All the recent data files are in netCDF format, so that is the format emphasized here. Those files contain measurements made in field campaigns that use the NSF research aircraft operated by NCAR, presently consisting of a C-130 and a Gulfstream V. A list of recent projects is available at this EOL web page, and data requests can be made via links on that page.

The standard data archives consist of netCDF files that contain a set of measurements each second during a flight. Special archives may be available at higher rate, usually 25 Hz. The Ranadu functions produce R data.frames containing subsets of those measurements and then facilitate plotting and analyses using those data.frames. The table in the Appendix contains an alphabetical list of the functions, brief descriptions, and references where additional information can be found.

There are several groups of functions that serve different purposes. Some examples are listed below. For a complete list, see the Appendix.

1. *Functions for data access:*
   (a) `getNetCDF()` for reading netCDF files to R data.frames
   (b) `DataFileInfo()` for a summary of the content of a netCDF archive file
   (c) `standardVariables()` to define common variables for (a)
   (d) `getProjectData()` for assembling a data frame covering an entire project
   (e) `setVariableList()` to provide a GUI interface to variable selection
   (f) `TimeRoutines()` including `setRange()`, `getIndex()`, `getStartEnd()`, and `selectTime()`
2. *Functions that implement calculation algorithms for derived measurements:* [partial list]
   (a) `TrueAirspeed()`
   (b) `WindProcessor()` to calculate wind from basic measurements
   (c) `AirTemperature()` to calculate ambient temperature
   (d) `DPfromE()` to get the dew point temperature from the water vapor pressure

(e) `PCorFunction()` to correct pressure measurements for the static defect

3. *Convenience functions for plots:*

   (a) `plotTrack()` for a simple flight track.

   (b) `plotWAC()` and `lineWAC()` to generate plots; also ggplotWAC().

   (c) `plotSD()` for size-distribution plots.

   (d) `contourPlot()` for colored-area plots (substituting for scattergrams).

   (e) `OpenInProgram()` to transfer variables in a data.frame to another analysis program like ncplot

4. *Utilities for handling and examining data:*

   (a) `makeNetCDF()` to make a new netCDF file from a data.frame

   (b) `Rsubset()` to create subset data.frames while preserving global and variable attributes

   (c) `Rmutate()` to add or modify variables in a data.frame.

   (d) `selectTime ()` to select a subset from a data.frame within specified times. See the related `setRange ()`.

   (e) `TellAbout()` to list characteristics of variables

   (f) `ValueOf()` to look up values from the data.frame

   (g) `getAttributes()` to see the original netCDF attributes or any added attributes

   (h) `WACf()` for a set of statistical functions suitable for combined use to summarize characteristics of variables.

   (i) `potentialTemperatures ()` for a similar set of potential temperatures to use in summaries.

5. *Miscellaneous functions for analysis:*

   (a) `DemingFit()` to minimize the perpendicular distance of points from a line

   (b) `VSpec()` to generate plots of spectral variance.

   (c) `skewT()` to construct a skew-T thermodynamic diagram and superimposing measurements

   (d) `LCL()` to calculate the lifted condensation level

   (e) `DPfromE ()` to calculate the dewpoint from the water vapor pressure.

   (f) `CAPE()` to calculate the convective available potential energy as well as adiabatic profiles for ascent

   (g) `AdiabaticTandLWC ()` for adiabatic-ascent calculations.

(h) `KingProbe ()` for calculations related to the CSIRO LW probe.

(i) `LagrangeInterpolation()`

(j) `binStats()` to aid in constructing error-bar or bar-and-whisker plots

(k) `CohPhase ()` to show the coherence and phase for cross-spectra. This routine also produces a cospectrum suitable for flux calculations.

(l) `flux ()` to display the cospectrum and the exceedance flux vs frequency.

6. *Functions to improve measurements:*

(a) `CorrectPitch(), CorrectRoll()` to improve measurements of pitch and roll

(b) `CorrectHeading()` to improve measurements of heading

When data.frames are created, global attributes in the netCDF file become attributes of the generated data.frame and variable attributes in the netCDF file are associated with column variables in the data.frame. Other information in the netCDF file is also stored with the data.frame, so the data.frame becomes a full representation of the netCDF file and complete netCDF files can be re-generated from the data.frames. However, usually data.frames are generated with only subsets of the data needed for particular analysis routines. They therefore can be small and suitable for archiving with analysis projects to document the data as used for the project.

Because these functions are assembled into an R package, they each have associated usage information. This manual is intended to provide additional information regarding the basis for the algorithms used, the limitations of the functions, and more general documentation. Some of the functions (will) have their own pages for this information.

This manual includes an introductory set of sections that discuss R and how to use it for data analysis. This is intended to be tutorial information, focused on the applications to aircraft data, and some of the information is also specific to uses on NCAR/EOL computers, so Sections 2–4 can be skipped by those who already are comfortable with R. Section 5 is the central discussion of Ranadu. Section 6 is mostly advocacy for using these routines and other capabilities of R to make analyses reproducible by others, and Sect. 7 presents a brief discussion of how these routines can be used for data-quality review during field projects.

2 Introduction to R

The very limited intent here is to provide a brief context in R for using the Ranadu package. There are many books, tutorials, and web resources that provide complete information on R, including the manuals available via R itself (e.g., via the 'help' tab in RStudio). Therefore, this and the following few chapters are provided only to get a reader started with R and to illustrate some of the analysis tasks that are possible.

2.1 Getting access: Some recommended set-up work

This section will guide you through obtaining R (if necessary), RStudio (which I recommend but which isn't essential), and the Ranadu package itself. I'm going to use barolo.eol.ucar.edu, a linux machine, for these illustrations. R is available on barolo, but RStudio no longer is, so it will likely work better to install R and RStudio on your own machines or laptop.

The central repository for R is CRAN (http://cran.r-project.org/), the Comprehensive R Archive Network. That page has links for downloading versions for Windows, Mac, and linux. For RStudio, you can download from http://www.rstudio.com/products/RStudio/. To run R (without RStudio) on barolo or any linux machine where it is installed, just type "R" into a terminal window[1] and you get the classic command-line interface that has been in use since at least 1998.[2]

My suggestion for getting started is this:

The best way to install Ranadu is described in Subsection 2.3 below. You may first want to do the following:

1. Make a new directory "RStudio" in your workspace on the machine you will use, named '~/RStudio'. You can do this using a terminal or using the RStudio interface; for the latter,
   (a) click the 'Files' tab in the lower right pane
   (b) use the 'New Folder' item just below that tab to create the new folder named 'RStudio' under your home directory.
2. Subsection 2.3 below describes how to download your own copy of the libraries that will be needed so that you can maintain and update your own copy.
3. Type "rstudio".
4. Go to Tools -> Global Options -> R General
   (a) enter '~/RStudio' (without quotes) in the line that says "Default Working Directory"
5. Go to Sweave under the same window
   (a) in the top where it says "Weave Rnw files using:", select 'knitr'. You may get a message saying knitr is not available; in that case, knitr will be added to the available packages when you follow the steps in subsection 2.3.

[1] To exit, type "quit()"; reply 'n' to 'save workspace' for now.

[2] It is also possible to run RStudio and R on Amazon Web Services, for example in a linux AMI machine used as a web server for RStudio. To do this, follow instructions here: https://blogs.aws.amazon.com/bigdata/post/Tx3IJSB6BMHWZE5/Running-R-on-AWS . You will also have to install the netcdf library (netcdf and netcdf-devel), and for that you will first need hdf5 and hdf5-devel. Download the AWS version of Ranadu, which omits one routine dependent on tcltk and omits that package dependence, if the version of R you use doesn't support tcltk.

6. Go to "Git/SVN" under the same window

   (a) check "Enable version control". This is very useful and will be the best way to stay current with the Ranadu package, which is installed and maintained there.

7. For other options, the defaults are probably reasonable to start, so click "OK" to save the options.

8. Optionally, make a new project called RSessions for the purpose of a tutorial having this name:

   (a) Go to the 'File' menu and select 'New Project'
   (b) Select 'Version Control'
   (c) Select 'Git"
   (d) Use this repository name: https://github.com/WilliamCooper/RSessions.git .
   (e) Give the project the name 'RSessions' and create it as a subdirectory of '~/RStudio'
   (f) At this point, RStudio will switch to windows that apply to this new project and copy from the repository into your new project.

9. If you are preparing your personal machine, you may eventually want LaTeX, and you may need to install git, although it seems to be standard on linux and Mac machines. If you are installing on another machine that doesn't have them try these links: For LaTeX, http://miktex.org/ and for git http://git-scm.com.

2.2 A quick tour of the RStudio window panes:

If you haven't already done this, start RStudio on your machine so you can see the windows, panes, and functions described here. You have seen the 'Files' window before if you followed the steps in the preceding section; click on that to see the files in RStudio/RSessions. Then, if you have followed step 8 above, click on 'RSessionsDataFrame.Rnw' to have something to look at. It will appear in the top left window. Don't worry about the details of what is there just yet; we'll get back to that. First, notice that there are four panes visible. At start-up, you will usually see these panes:

1. Top left: The source editor, where a working file (or many working files in tabs) are displayed.

2. Bottom left: The console, where you can enter R commands just as you would in the command-line mode that you would use without RStudio. Try it now: Click where the '>' prompt is displayed, type 2+2 [enter]. I use this frequently as a very capable calculator, because the math functions and R functions are all available here. Try 'pi', 'rnorm(5, 3.5, 0.5)', plot(0:360, sin(0:360 * pi / 180), type='l'), and 'Ranadu::WetEquivalentPotentialTemperature (P=800., AT=25., E=Ranadu::MurphyKoop (DP=20., P=800)).

3. Top right: This pane has several functions.

   (a) The 'History' tab displays the history of what you have entered in the console, along with options to transfer lines either to the source editor or to the console for re-execution, possibly with editing (although that function is available more conveniently from the console using the up and down arrows to move through the history).

   (b) Another tab will show the environment, where defined variables for your workspace are shown. [Try X = 5 in the console and see what appears here.] There is also a useful tab for importing a data set from either a web location or a text file.

   (c) This pane also has a 'git' tab if you have set up the project as recommended above, where you can monitor the status of what has changed since you last saved to your local repository and where you can commit back to your local repository. You can also "pull" from the github repository where appropriate (e.g., to get Ranadu updates).

4. Bottom right: Again, there are several functions.

   (a) Plot: If you tried the examples in item 2 above, you will have noticed this pane switch to 'Plots' and display the latest plot. RStudio also maintains a history of the plots that have been generated, and you can skip back through that history, which seems endless in my experience so far. The 'export' function in the 'Plot' tab allows you to save the plot externally to either png or pdf format.

   (b) Files: This displays the files in the working directory, which for now should be the project directory. There are a few functions for dealing with those files.

   (c) Help: Requests for help are displayed in this window. If you click on the 'Home' button, you are taken to a page that shows links to some of the basic manuals and also a definitive definition of the R language, along with other more specialized manuals. If you want to know how a particular function works, you can enter your search phrase into the console as, for example, ?atan2 to see information on the arctangent function, or you can enter this into the entry line on the help window, which has the advantage that it displays auto-completion options as you type. [Try 'plot' to see this in action.] There are many cross-references among the help pages that are quite useful. If you click on the left drop-down menu you will see a history of the help topics you have consulted.

   (d) Viewer: I'm going to postpone discussion of this until later; you won't need the viewer for a while.

   (e) Packages: This displays the packages that are available for your use, either in the system-wide repository or in your personal repository. To test if you have access to Ranadu, look for 'Ranadu' in the list of packages. If it is there, let your mouse linger over it and its location will be displayed. You could try this for other packages also; check, for example, ggplot2, a very useful and standard plotting package not yet part of the EOL repository. The check boxes indicate which packages will be available in your

session without special loading; if unchecked, you can either check the box or include a statement similar to 'require(Ranadu)' in your session (or when you use these functions, use the form Ranadu::routine_you_want). The 'install' and 'update' functions at the top of the 'Packages' pane provide easy access to the 'CRAN' archive of packages or to packages you may have from other sources that you want to load for use. There is a very extensive set of packages available via CRAN. If you need something new, check there to see if someone has already provided a package for the task you need.

2.3 Getting the 'Ranadu' package

You may first need to install some packages on which Ranadu depends. Ranadu uses these packages:

```
R (version >= 3.5), ncdf4, maps, ggplot2, grid, graphics, utils,
scales, ggthemes, tcltk, nleqslv, zoo, fields, signal, reshape2,
bspec, dplyr, plyr, tibble, stats.
```

`Package magrittr` is also recommended and used in some examples that follow. You will also want `knitr` if you want to combine text and R code in files that generate documents, as described in a later section. If any of these are missing, they can be installed by clicking 'Install' at the top of the 'Packages' pane of RStudio, selecting 'repository (CRAN)' in the first drop-down menu, and entering a list of packages that you are missing in the second line labeled 'Packages'. The default location for local libraries on the last line should be kept, and the check-box labelled 'Install dependencies' should be checked. Click 'Install" and those packages should be downloaded from the CRAN repository, compiled where necessary, and added to your list of packages.

Then to get the package 'Ranadu', follow these steps:

Preferred Method:

1. In the R or RStudio console, type `devtools::install_github(‘‘NCAR/Ranadu’’)`. If devtools is not already present, you may need to install it first.
2. Change to the "Ranadu" project.
3. Go to the "Build" window and click "Build & Reload".

An advantage of this method is that, if you create Shiny apps that use Ranadu, the installer will be able to upload the needed parts of Ranadu with the Shiny app.

Alternate Method:

1. Within RStudio, select 'File' -> 'New Project' -> 'Version Control' -> 'Git' (clone a repository)
2. For the package URL, enter 'https://github.com/WilliamCooper/Ranadu.git' (without the quote marks). Use the package name 'Ranadu' and make the directory for it a subdirectory under 'RStudio'.
3. Click 'Create Project' and RStudio will download the files and change the working project to Ranadu.
4. Go to the 'Build" window in the top-right pane and click 'Build & Reload'. You may get an error listing missing dependencies if any required packages are unavailable. In that case, return to the above description of how to install packages and install the missing dependencies.
5. The package 'Ranadu' should appear in your list of packages.

3 Background: R objects, with focus on the data.frame

3.1 Some basic objects

vector

The fundamental objects we will deal with the most is the vector. This is an indexed set of quantities, which can be numeric (double, integer), logical, character, or some other less common types. Vectors are 'atomic' if they consist all of the same type; 'generic' vectors can have mixed types, but I won't deal much with them. Vectors can have only one element, which would be called a scalar in other languages, but scalars in R are just vectors with one element. Vectors have a length property [try length(x)] but not a dimension property: Try the following:

```
x <- 2
print (x[1])
length(x)
dim(x)
```

The '<-' operator above is the R assignment operator. It is mostly equivalent to the '=' operator from other languages, and '=' can usually be used in place of this operator, but it does have the clarity of indicating that the variable on the left is assigned the value of the quantity on the right. The operator '->' can also be used, but usually will result in rather confusing code.

list

A list resembles an atomic vector but can have mixed types. Lists also have additional ways of selecting elements, e.g., by character name. Lists are generic vectors, but when I talk about vectors I'll almost always be talking about atomic vectors.

array

A special vector-like object is the array or matrix. These are like vectors but with an assigned dimension so that, in the two-dimensional or matrix case, the vector is a vector of columns in a matrix. The dimension can be single-valued, but more often there are at least two dimensions. R convention is column-major format (like FORTRAN but opposite to C); i.e., the row index represents adjacent values in the underlying vector (items that lie vertically in the conventional display of a matrix) so that for example z[1,1] and z[2,1] are adjacent in the underlying vector and aligned vertically in a display of the matrix. The first index in a two-dimensional array can be thought of as specifying the row and the second the column; all rows in a particular column appear together. You can construct an array from a vector by assigning it a dimension via 'dim(x) <- c(dim1, dim2)' as in the following example,[3] which also illustrates the column-major structure and the effect of transposing the array using t():

```
a <- 1:12            # generate a vector containing the sequence 1 to 12
print (a)

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12

dim (a) <- c(3,4)  # c() constructs a vector from its elements
print (a)

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

print (t(a))

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

[3] Want your original vector back? Use x <- as.vector(x)

(Note that '#' indicates the the remainder of the line is a comment.) This works well for arrays constructed from our netCDF files because the time index can be thought of as the row index and each column represents a vector of measurements of a particular variable.

data.frame

A data.frame is a named list of vectors and resembles a matrix or a spreadsheet. For example, it may consists of columns each representing a variable and rows that are the time sequence of observations of that variable. Applied to RAF data files, it may have a structure like this:

Time	ATX	PSXC	WDC	WSC	...
15:00:00	-25.1	410.8	275.4	25.4	...
15:00:01	-25.1	410.9	275.2	25.6	...
15:00:02	-25.2	411.1	275.4	25.5	...
15:00:03	-25.1	411.1	275.1	25.7	...
15:00:04	-25.0	411.2	275.3	25.3	...
...	...	...	...	...	...

All columns must be of the same length, but they may contain different types of variables (Time, character, numeric, logical). Like a spreadsheet, the columns can be assigned names like the header in this table. Rows can also be assigned names, but in the absence of special assignment they will default to the character names '1', '2', '3', '4', ... and can be addressed by an index that is the first in a two-index specification; i.e., D[500, 10] represents the 10th variable as measured at the 500-th time in the dataframe.

Let's get an example. Included with Ranadu is a sample data.frame named `RAFdata` that contains a short sample of measurements from a research flight.

```
require(Ranadu, quietly = TRUE, warn.conflicts = FALSE)
names (RAFdata)
```

```
##  [1] "Time"   "ATTACK" "SSLIP"  "GGVEW"  "GGVNS"  "GGVSPD" "VEW"
##  [8] "VNS"    "ATX"    "DPXC"   "THDG"   "ROLL"   "PITCH"  "PSXC"
## [15] "QCXC"   "EWX"    "RTH1"   "RTH2"   "ADIFR"  "BDIFR"  "TASX"
## [22] "GGALT"  "LATC"   "LONC"   "PLWC"   "CONCD"  "WDC"    "WSC"
## [29] "WIC"
```

```
Ranadu::getStartEnd (RAFdata)
```

```
## [1] 201000 201500
```

`names()` is a standard R function that lists the names associated with an object, so as used in this illustration it lists the names of the variables in the data.frame. `getStartEnd()` is a Ranadu function that returns the start and end times in the data.frame in the form HHMMSS. The leading `'Ranadu::'` is not necessary here because the package is loaded, but these notes will tend to include that superfluous header to highlight which parts of the code refer to the Ranadu package. The lines above starting with '##' are the output from the R code in the listing. The first few rows and first few columns in the data.frame `'RAFdata'` look like this:

```
##                      Time   ATTACK      SSLIP    GGVEW    GGVNS
## 9721 2013-10-01 20:10:00 1.999224 -0.1875578 53.44360 226.5322
## 9722 2013-10-01 20:10:01 1.943987 -0.1734225 53.57179 226.9628
## 9723 2013-10-01 20:10:02 1.981951 -0.1649992 53.69322 227.4117
## 9724 2013-10-01 20:10:03 1.993126 -0.1854287 53.81462 227.8630
## 9725 2013-10-01 20:10:04 1.996357 -0.1891418 53.94920 228.2795
## 9726 2013-10-01 20:10:05 1.987496 -0.1899927 54.07005 228.7055
## 9727 2013-10-01 20:10:06 1.996506 -0.1594946 54.18315 229.1384
## 9728 2013-10-01 20:10:07 1.989084 -0.1842116 54.26129 229.5390
## 9729 2013-10-01 20:10:08 1.994024 -0.1646231 54.33908 229.9423
## 9730 2013-10-01 20:10:09 1.989765 -0.1765334 54.41174 230.3561
```

The `'Time'` variable is a special-format variable in POSIX format, discussed later when the Ranadu access functions are described. The first item in each row is the row name, seldom used but here a remnant from the index in the original data.frame from which this subset was extracted. Having the time variable in this format is particularly convenient when examining time series of measurements, as will be described later.

3.2 Working with data.frames

Addressing elements of a data.frame

You can address particular elements in many ways, using syntax like the following:

```
RAFdata$ATX[5]
## [1] -36.71588
RAFdata[5, 2]           # note the [row,column] syntax
## [1] 1.996357
RAFdata[5, ]
##                      Time   ATTACK      SSLIP   GGVEW    GGVNS   GGVSPD
## 9725 2013-10-01 20:10:04 1.996357 -0.1891418 53.9492 228.2795 -0.58602
##           VEW      VNS       ATX      DPXC     THDG      ROLL    PITCH
## 9725 54.11258 228.2388 -36.71588 -50.19225 2.978105 0.3509116 1.760113
##          PSXC     QCXC        EWX      RTH1     RTH2     ADIFR      BDIFR
```

```
## 9725 301.8117 126.1786 0.06180029 -12.34667 -12.4678 -13.82708 -0.7944382
##          TASX    GGALT     LATC     LONC     PLWC CONCD      WDC      WSC
## 9725 223.3128 9424.435 44.99973 -101.271 8.453611     0 262.9346 42.98902
##            WIC
## 9725 0.5486414
RAFdata[5, "ATX"]       # using " or ' to surround text is usually equivalent
## [1] -36.71588
RAFdata$ATX[1:10]
##  [1] -36.77266 -36.77094 -36.75904 -36.73860 -36.71588 -36.70886 -36.69385
##  [8] -36.68116 -36.71080 -36.72738
RAFdata$ATX[Ranadu::getIndex(RAFdata, 201105)]
## [1] -36.58235
RAFdata$ATX[RAFdata$Time == as.POSIXct("2013-10-01 20:11:05", tz='UTC')]
## [1] -36.58235
```

('getIndex ()' is part of Ranadu and provides a convenient way to determine the data.frame index that corresponds to a specified time.)

Creating subsets of a data.frame

New data.frames that contain subsets of original data.frames can be created using logical vectors. For example:

```
RAFdata[RAFdata$TASX < 213, "TASX"]

## [1] 212.9421 212.9741 212.9477

RAFdata[Ranadu::setRange(RAFdata, 201210, 201215), c(1, 6:10)]

##                    Time    GGVSPD      VEW      VNS       ATX      DPXC
## 9851 2013-10-01 20:12:10 -10.78104 57.81869 243.1800 -33.63138 -54.88298
## 9852 2013-10-01 20:12:11 -10.80652 57.80622 243.0562 -33.50258 -54.90860
## 9853 2013-10-01 20:12:12 -10.88169 57.78197 242.9372 -33.42321 -54.71121
## 9854 2013-10-01 20:12:13 -10.89062 57.74649 242.8225 -33.37087 -55.01767
## 9855 2013-10-01 20:12:14 -10.89485 57.71845 242.7063 -33.28124 -54.75344
## 9856 2013-10-01 20:12:15 -10.81118 57.69278 242.5950 -33.20294 -54.59950
```

Another useful subset is that omitting all missing-variable rows from the data.frame:

```
RAFdataNoNA <- na.omit(RAFdata)
```

However, be careful using this and other subsetting commands because if the time sequence has gaps some functions like 'setRange()' won't work, although plots will just skip the missing values. When it is necessary to omit missing values (e.g., for filtering), it is best to do other subsetting before using the na.omit() function.

Adding or changing variables in a data.frame

You can operate on variables in the data.frame, changing values, and you can add new variables to the data.frame as follows:

```
# wind component from the east:
RAFdata$UEW <- RAFdata$WSC * sin (RAFdata$WDC * pi / 180)
summary (RAFdata$UEW)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -44.64  -43.67  -42.66  -41.73  -38.79  -38.01
```

A few comments on this example may be useful. The function `summary()` provides a quick indication of the range of a variable and also includes an indication of the number of missing elements (having the R value `NA`). The assignment equivalently could have been made to `RAFdata['UEW']`. The standard math functions in R include the `sin()` function used here, but it takes an argument in radians so the wind direction is multiplied by `pi/180` to convert from the wind direction `WDC` in degrees. "pi" is always available, and R makes no distinction between the numerical values ``180.' and ``180" so the decimal point that would be used in many other languages is superfluous here.

Simple plots

Let's plot something, using '`plotWAC()`' from Ranadu for convenience, although you can try the standard '`plot ()`' function from R base routines to see what changes I have made:

```
Ranadu::plotWAC(RAFdata$Time, RAFdata$GGVSPD, ylab='rate of climb [m/s]')
```

It is also useful to define special data.frames for constructing plots, especially when using the more advanced plotting capabilities provided by '`ggplot2`'. To see a simple scatterplot, you can use the following:

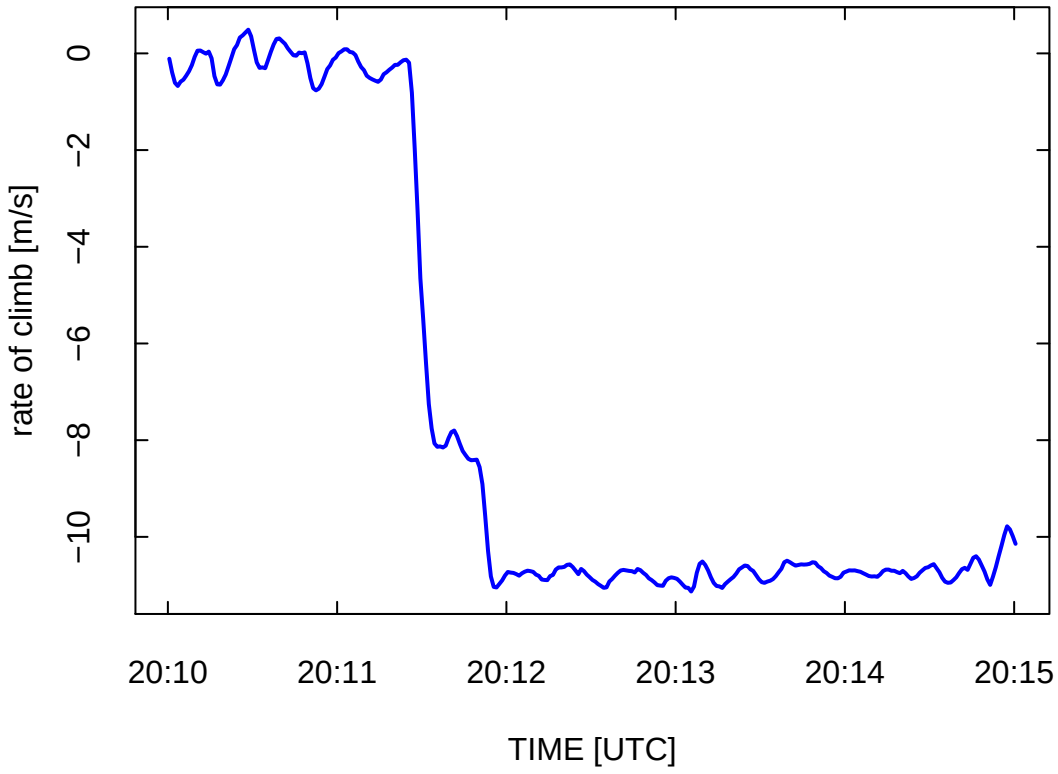


Figure 1: Geometric altitude as a function of time.

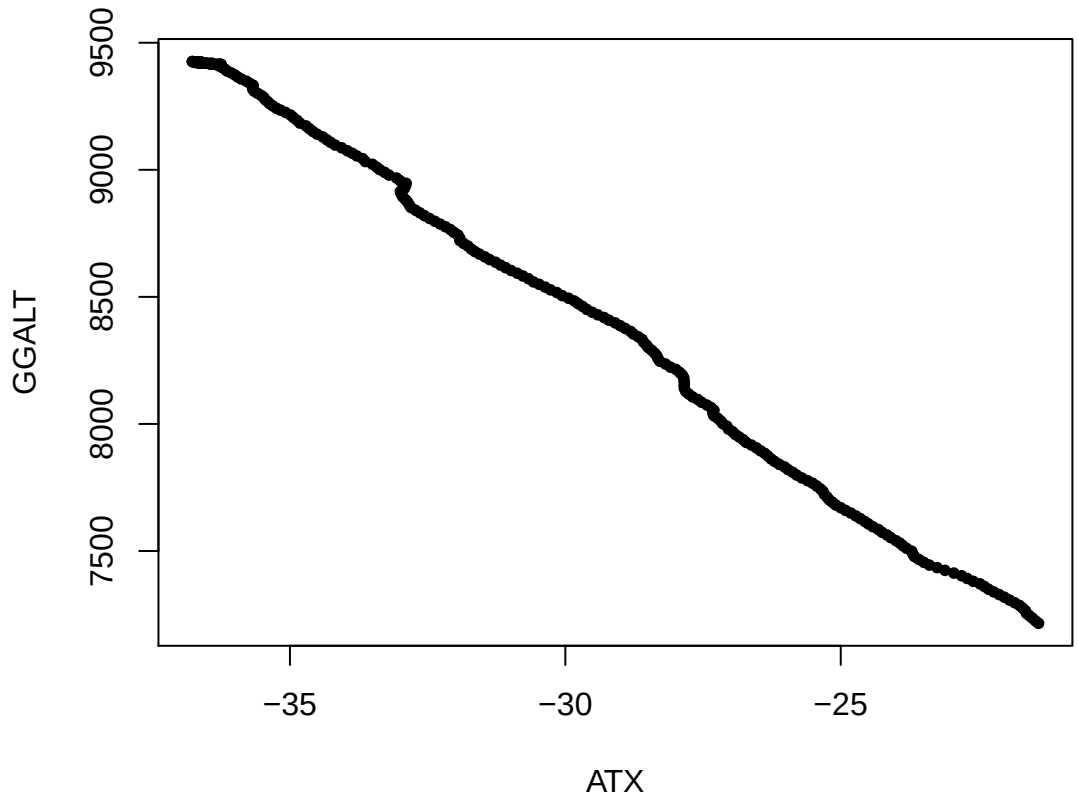


Figure 2: Scatterplot of temperature vs. geometric altitude.

```
D <- RAFdata[, c("ATX", "GGALT")]
plot(D, pch=20)          # pch=20 plots small solid dots
```

```
## equivalent to:
# plot (RAFdata[, c("ATX", "GGALT")], pch=20)
## or to
# plot (subset (RAFdata, , c(ATX, GGALT)), pch=20)
```

You may find the last form attractive because it avoids the need to type the quote marks in the second form, and those quote marks can seem tiresome in extended strings. Exercise: See what happens if you instead include three variables in the preceding plot, e.g., by adding `DPXC`.

Exporting to Excel

If you have the xlsx package available, you can create an Excel spreadsheet with the data:

```
# require(xlsx)
xlsx::write.xlsx (RAFdata, file="RAFdata.xlsx")
# system("libreoffice RAFdata.xlsx")  ## to see the resulting spreadsheet
```

This can be very convenient for those who are comfortable working with the features of Excel spreadsheets, and this is also a convenient way to examine data.frames.

Exporting to ncplot

Many users of these aircraft data archives will be familiar with ncplot, a program provided by NCAR/EOL. Use the OpenInProgram() to start ncplot with the variables in an R data.frame, as follows:

```
# OpenInProgram (RAFdata)  ## not executed here
## try ?OpenInProgram to see the options for this function
```

New variables added to 'Data', perhaps representing alternate calculations of standard variables, can then be plotted along with the original variables. [4]

4 Background: Some basic operations in R

4.1 Using the RStudio console as a calculator

The standard R interface or the console pane in RStudio (the lower left pane) can be used as a convenient calculator, and it is possible to do some simple programming interactively. As a simple example, let's find the roll angle for a 3-min turn (i.e., a turn through 360° in 3 minutes). If the airspeed is v, the radius of the turn is r, the acceleration of gravity is g, the time taken for a turn through 2π radians is T, and the roll angle is ϕ, then

$$\frac{v^2}{r} = g \tan\phi$$

[4] This also works with 'Xanadu', a legacy program not included with this package and no longer supported that nevertheless provides some additional capabilities including the possibility to transfer the data to still other analysis programs including IDL, NCL, and python and to take a variety of approaches to spectral analysis.

is the required horizontal acceleration and the distance flown around the turn is

$$2\pi r = vT$$

so

$$\phi = \arctan\left(\frac{2\pi v}{gT}\right)$$

Here is an evaluation in R:

```
v <- 220          ## do the calculation for a flight speed of 220 m/s
g <- 9.8          ## acceleration of gravity in m/s^2
T <- 3 * 60
phi <- atan (2 * pi * v / (g * T)) * (180 / pi)  ## convert to degrees
print (sprintf ("required roll angle is %.1f degrees", phi))

## [1] "required roll angle is 38.1 degrees"
```

The console is often very useful for exploring aspects of data. For example, after running R scripts, the R objects remain available in the working environment and can be examined, plotted, or otherwise used in exploratory analysis.

4.2 Some math conventions

Here the focus is on what might seem different to those experienced in other programming languages.

Operator precedence

- :: $ [ ] PEU:MDAS
  :: references a package and has highest precedence
  $ references a column of a data.frame (for example)
  [] are used for subsetting operations
  P denotes parentheses
  E denotes exponentiation (see below)
  U denotes the unary operators '-' and '+' when used to indicate signs
  : is an operator that generates sequences
  %% operators include the modulus operator and some others like %/%
  M,D denote multiplication and division and have equal precedence so are evaluated in order
  A,S denote addition and subtraction and are equal in precedence.

- Here are some special cases to note:
  In "1:5 * 2", : has precedence so you get c(2,4,6,8,10) not c(1,2,3,4,5,6,7,8,9,10)
  The precedence of logical operators is ! (& &&) (| ||) xor . Note that & has precedence over |, so for example the result of TRUE | FALSE & FALSE is TRUE.

Operators to note:

- The exponentiation operator is '^', although '**' is also accepted.
- & and | are vectorized while && and || return a single logical result.
- %% is the modulus operator; i.e., `8 %% 3` is 2. %/% gives integer division; e.g., `16 %/% 3` gives 5.
- %in% tests if an element is present in a vector; e.g., `'TASX' %in% names(RAFdata)`
- The function c() combines its arguments, usually to form a vector; e.g., A <- c(1, 3, 6)
- The test for equality is '==', not '='; the latter will result in assignment and is usually an acceptable substitute for '<-' but '<-' is preferred except to specify defaults for function arguments.
- R does not have operators like '+=", "++", etc, so you have to write out statements like 'i <- i + 1'
- The preferred values to use for the status of logical operations are TRUE and FALSE. R does not recognize True or False. It does accept T and F, but these are dangerous to use and should be avoided because you may inadvertently use T or F for variable names.

4.3 Vector operations

Almost everything can be accomplished using vectorized operators, and loops are seldom needed. Vectorized functions are often significantly faster than equivalent loops and usually result in less cluttered and clearer code so it is worthwhile to take advantage of vector capabilities. Here are a few examples, which you should try:

```
AT <- 2 * (RAFdata$ATX - 1)      ## for each element, subtracts 1,
                                 ## then multiplies by 2
length (AT)      ## shows that AT is a vector with 301 elements

## [1] 301
```

```
ROCsq <- RAFdata$GGVSPD^2
MACH <- Ranadu::MachNumber (RAFdata$PSXC, RAFdata$QCXC)
str (MACH)      ## str() shows the structure of a variable

##  num [1:301] 0.719 0.72 0.722 0.723 0.724 ...

# This is a replacement for a for loop calling SmoothInterp for
# each column in RAFdata:
SData <- as.data.frame(lapply(RAFdata, SmoothInterp))
```

Here are some comments about vectorized operations:

1. Almost all Ranadu functions are vectorized, with a few exceptions (esp. where fits are required that depend on R functions like 'nleqslv' that are not vectorized).
2. If vectors are supplied that have different lengths, the shorter one is recycled; this makes it possible, for example, to use calls like `MachNumber (500, RAFdata$QCXC)`.
3. Vectorized logical operators can be particularly useful for selecting subsets of data:
   (a) To create a new data.frame that excludes periods where the airspeed is less than some limit MinTAS:
   D <- Data[Data$TASX < MinTAS,] Note the comma; this keeps all columns but only includes rows where TASX exceeds the limit. 'Data$TASX < MinTAS' becomes a logical vector, and when used as an index selects only those values of Data where the logical vector is true.[5]
   (b) More complex selection tests can be made by combining logical arrays; for example
   Valid <- (Data$TASX > MinTAS) & (abs (Data$ROLL) < 4) & (Data$GGVSPD > 2)
   Data[!Valid, c("WDC", "WSC")] <- NA
   (c) Consider also the 'subset' command, for which the second argument is a logical test and the third specifies the variables to include in the subset (see ?subset for more information):[6]

```
D <- subset (RAFdata, RAFdata$GGVSPD > 0, c(Time, ATX, GGVSPD))
str (D)
```

[5] A better approach is provided by dplyr::filter(), which has the advantages that attributes of variables are preserved and it is convenient to use in magrittr-style pipes.

[6] In this case also, the routines dplyr::select() and dplyr::filter() usually are better choices, again because all attributes are preserved. The attributes of variables include measurand names and units that are useful in default labels.

```
## 'data.frame': 29 obs. of  3 variables:
##  $ Time  : POSIXct, format: "2013-10-01 20:10:10" "2013-10-01 20:10:11" ...
##  $ ATX   : num  -36.7 -36.7 -36.6 -36.7 -36.6 ...
##  $ GGVSPD: num  0.0557 0.0586 0.0251 0.0359 0.0864 ...

library(magrittr)  # This package is needed for the %>% pipes:
D <- RAFdata %>% dplyr::filter(GGVSPD > 0) %>%
     dplyr::select(Time, ATX, GGVSPD)
# str(D) will now show the long retained list of attributes
tibble::as_tibble(D) # Convert to a tibble for concise display

## # A tibble: 29 x 3
##    Time                  ATX GGVSPD
##    <dttm>              <dbl>  <dbl>
##  1 2013-10-01 20:10:10 -36.7 0.0557
##  2 2013-10-01 20:10:11 -36.7 0.0586
##  3 2013-10-01 20:10:12 -36.6 0.0251
##  4 2013-10-01 20:10:14 -36.7 0.0359
##  5 2013-10-01 20:10:23 -36.6 0.0864
##  6 2013-10-01 20:10:24 -36.6 0.172
##  7 2013-10-01 20:10:25 -36.6 0.327
##  8 2013-10-01 20:10:26 -36.6 0.371
##  9 2013-10-01 20:10:27 -36.6 0.431
## 10 2013-10-01 20:10:28 -36.6 0.490
## # ... with 19 more rows
```

4.4 R packages

A strength of R is the extensive set of packages available to help with analysis tasks. The standard repository is called "CRAN", although there are many packages available on GitHub also. To see the packages available on CRAN, check this URL. Most of what has been discussed to this point is in the 'base' package, including the math functions. Some additional frequently used functions, like '`plot()`', are in other standard packages (in this case, 'graphics') and are usually available without special actions. However, others require special action like loading via the '`require()`' or '`library()`' functions, or access via statements like '`packageName::function()`' (e.g., `Ranadu::getNetCDF()`). A few of the packages used by Ranadu routines, and some others that have proven especially useful, are discussed in this subsection to provide some context for what capabilities they provide.

These are some of the packages referenced by Ranadu:

1. ncdf: basic netCDF functions. This package is used for access to the netCDF data files archived by NCAR/EOL/RAF.

2. ggplot2 and ggthemes: These are alternatives to the standard graphics package and provide particularly aesthetic and flexible plots.
3. stats and signal: used for interpolation and filtering of signals
4. nleqslv: solve non-linear equations (used e.g. by DPfromE() and LCL())
5. maps: provides state or national boundaries for maps
6. fields: used by the skewT-generating program to average measurements in bins
7. zoo: used to replace missing values by interpolation (na.approx())
8. grid and reshape2: used to customize the ggplot theme

Two other packages that are particularly useful for the purposes of this manual are knitr and devtools. The former provides the ability to intermix text-generating code with R code, and so provides an important route to making analyses reproducible, as discussed later. 'devtools' is useful for checking and building packages and may be useful if you want to add components to a package like Ranadu.

5 Using the Ranadu package

5.1 Getting help

All functions will provide information regarding their usage in response to a `?FunctionName` query in the console or the RStudio help window. (Example: `'?getNetCDF'`) In addition, in most cases that help window provides an example that uses the `'RAFdata'` data.frame and shows how results should look. These help displays should be considered the definitive information because they are more likely to remain up-to-date than is this manual.

5.2 The data.frame containing measurements

Using getNetCDF

Ranadu is based on data.frames that contain subsets of data representing measurements in the NCAR/EOL/RAF netCDF data archives. The starting point therefore is to construct those data.frames. These routines assist in that task:

- `Data <- getNetCDF( )`: loads a data.frame named Data with requested variables
- `getProjectData( )`: Loads a data.frame with multiple flights from a project.

- `standardVariables( )`: defines a common set of variables for input to getNetCDF(). This list is the default if no variables are specified in the call to getNetCDF.
- `DataDirectory( )`: This is system dependent and is intended to aid in portability by providing the usual location of the netCDF data files. It tries a set of likely names, but may need redefinition for some file systems. It has no calling arguments. Example:

```
Project <- 'SOCRATES'; Flight <- 15
Data <- getNetCDF(file.path(DataDirectory(), Project,
'/', Project, sprintf('rf%02d.nc',
  Flight), fsep = ''))
```

- `getIndex ( )`: finds the index for a specified time
- `setRange ( )`: sets a range of indices covering a specified time interval
- `TellAbout (V)`: lists some characteristics of variable V
- `Z <- getAttributes (V)`: lists the original netCDF attributes associated with V (either a data.frame or a variable in a data.frame)
- `FI <- DataFileInfo ( )`: produces a list containing properties of a specified netCDF file or the corresponding Rdata file.

The starting point will usually be getNetCDF(). Here is an example of how this might be used:

```
require (Ranadu, quietly=TRUE)
## try ?getNetCDF
Data <- getNetCDF (fnm <- sprintf ("%s/extdata/RAFdata.nc",
                           path.package ("Ranadu")), "ALL")
dim (Data)             ## the length in seconds and the number of variables

## [1] 301  28

names (Data)           ## see the variables in Data

##  [1] "Time"   "ATTACK" "SSLIP"  "GGVEW"  "GGVNS"  "GGVSPD" "VEW"
##  [8] "VNS"    "ATX"    "DPXC"   "THDG"   "ROLL"   "PITCH"  "PSXC"
## [15] "QCXC"   "EWX"    "RTH1"   "RTH2"   "ADIFR"  "BDIFR"  "TASX"
## [22] "GGALT"  "LATC"   "LONC"   "PLWC"   "CONCD"  "WDC"    "WSC"

getStartEnd (Data)     ## the start and end times

## [1] 201000 201500
```

```
## try TellAbout (Data)
Z <- getAttributes (Data$ATX)

## [1] "attributes for variable"
## [1] "units: deg_C"
## [1] "_FillValue: -32767"
## [1] "long_name: Ambient Temperature, Reference"
## [1] "standard_name: air_temperature"
## [1] "Category: Atmos. State"
## [1] "DataQuality: Preliminary"
## [1] "Dependencies: 1 ATHR1"
## [1] "measurand: air_temperature"
## [1] "label: temperature (ATX) [deg C]"

FI <- DataFileInfo (fnm)
## try str (FI)
## try str (Data)
```

If you have Ranadu installed, these and other segments of R commands should work as they do in these examples. Where there are comments that start with ## try, you should type the command that follows, but the output is usually too large to include here.

These comment elaborate on the code in the preceding code block:

1. Using `Ranadu::DataDirectory()`: In the case shown above the path to the example netCDF data file provided with Ranadu is special and path.package() is used to find the file. Usually you will not load the data from this path. The function DataDirectory() is provided to help make routines portable among systems where the data files might be loaded in different locations. That function tries a set of possibilities and sets the returned directory to the first found. The possibilities are: $DATA_DIR, "/scr/raf_data/", "/home/data/", "/home/Data/", "/Data/", and "~/Data/". You may want to set your environment variable $DATA_DIR appropriately if one of these doesn't match your system. A typical use of DataDirectory() is shown in the code block that follows.

2. Using `Ranadu::standardVariables()`: In the code block shown above, "ALL" is used to load all the variables in the data file. Usually this will result in data.frames that are very large and will slow processing, and if this is done for a high-rate file you may exceed available memory, so it is better to list only the variables that you want. "standardVariables()" provides a basic set, and an argument to standardVariables like c("VAR1", "VAR2") can be used to add more variables. Alternately, you can just provide a list of variables that you want, as illustrated in the code block that follows.

3. Ranadu::getAttributes(): The data.frames produced by getNetCDF() have assigned attributes that match those in the original netCDF file, and this can be retrieved via getAttributes().[7] The result of that call is a printed listing of attributes and creation of a list (here in Z) that contains the attributes, so it is usually preferable to assign the returned value to something to avoid the double listing that would be produced otherwise.

4. Ranadu::TellAbout() and str() provide some details about the characteristics of variables.

5. Ranadu::DataFileInfo(): Returns a list with properties of the netCDF file including the Project, Flight number, the flight date and times, the data rate, the range of latitudes and longitudes included in the flight, and a list of the variable names in the file.

A more typical starting block of code would look like the following, listed here but not executed:

```
require (Ranadu)
Project <- "DEEPWAVE"
Flight <- "rf15"
fname <- sprintf ("%s%s/%s%s.nc", DataDirectory(), Project, Project, Flight)
## or you can provide the name via
## fname <- "/scr/raf_data/DEEPWAVE/DEEPWAVErf15.nc"

## try this option for selecting fname:
## WD <- getwd ()  ## save working directory to return to it
## setwd(DataDirectory ())  ## set usual data directory
## fname <- file.choose ()  ## use the R gui file chooser
## setwd (WD)}     ##  return to original directory

VarList <- standardVariables (c("ADIFR", "BDIFR", "CONCU_"))
## try 'standardVariables()' to see the standard list
Data <- getNetCDF (fname, VarList)
## try ?getNetCDF to see additional optional arguments to getNetCDF()
```

Some additional comments on getNetCDF:

1. The variable name "CONCD_" is not a standard name, but this structure is provided to avoid the need to know the full names. Variables from probes like the UHSAS or CDP have suffixes that indicate where the probe was located on the aircraft (e.g., _RWO for right wing outboard pylon). If a variable name is provided that ends in '_', a matching variable with a location index will be provided. The full name (e.g., "CONCU_RWO") can be provided if there is possibility of ambiguity, but usually this avoids the need to lookup where a probe was installed for a particular experiment.

[7] Subsetting will often lose some attributes so special steps are needed to preserve them if that is desired.

2. The "Time" variable needs special discussion. In NCAR/EOL/RAF data files, time (except for some very old projects) is provided as an integer representing seconds after a reference time, usually 0:0:0 UTC on the day the flight started. The "units" attribute is listed as, for example, "seconds since 2014-06-06 00:00:00 +0000". For files at rates higher than 1 Hz, the time remains integer seconds but variables are provided with additional dimensions that represent the measurements within that second. The Ranadu function getNetCDF() always provides "Time" as the first variable, but this is converted into POSIXct format (see ?POSIXct in R) which is the number of seconds since the start of 1970 (UTC). Fractional values are used in cases of data rates higher than 1 Hz, giving a "flat" file. This format has the advantage that plot routines interpret times appropriately and produce appropriate labels in terms of time. To see the date and time in conventional format, use "as.POSIXlt ()' with a POSIXct-format time. This also returns a named list with names like "hour", "min", "sec" that you can use to create different-format representations of the time or date.

3. The optional getNetCDF argument "F" can be used to preserve the flight number, by for example providing F=15 for flight 15. This results in the addition of a new variable called "RF" to the data.frame. This is provided for cases where multiple data.frames are combined in order to facilitate analysis tasks using data that span more than one flight. If data.frames have exactly the same variables, then can be combined using the 'rbind()' function:

   ```
   DataM <- rbind (Data1, Data2)
   ```

   If flight numbers have been added, the individual flights can be retrieved by subsetting; e.g.,

   ```
   D1 <- DataM[RF == 1, ]
   ```

   This may be useful, for example, when finding radome sensitivity coefficients for an entire project, to check if individual flights provide outlier measurements. This is the method used by `getProjectData( )` to construct a single data.frame covering the project but still permit selection of individual flights.

4. In the netCDF files, missing values are usually indicated by the value -32767. getNetCDF converts such values to NA, the R value for missing values. is.na() will test for such values.

Examining the measurements

Ranadu provides some functions intended for examination of the measurements. Some, ''`getStartEnd()`'', ''`TellAbout()`'' and ''`getAttributes()`'', were discussed above. Others include:

1. `Ranadu::getIndex(Data, HHMMSS):` For data.frame Data, `getIndex()` returns the index corresponding to the time provided by HHMMSS, which should be a time like 201213 (without : separators). Data[i,] will then contain all the measurements at the time corresponding to HHMMSS. See the examples below.

2. `Ranadu::setRange(Data, Start, End):` This function returns a sequence of inclusive indices covering the period from "Start" to "End" (both provided in HHMMSS format). Data[setRange(Data, Start, End),] will then be a subset data.frame containing all the measurements in Data for the specified period. Perhaps a more convenient method is provided by `selectTime( )`, which is particularly suited to inclusion in pipes.

3. `Ranadu::ValueOf(Data, V, HHMMSS):` This will return the value of variable V in data.frame Data at time HHMMSS.

4. `Ranadu::ValueOfAll(Data, HHMMSS):` This is like "`ValueOf()`" but returns the values of all the measurements at the specified time.

5. `Ranadu::WACf(V)` : This provides a set of statistical functions like mean and standard deviation. The function is designed to use with `lapply()` to provide a set of results, as illustrated below.

```
RAFdata[getIndex (RAFdata, 201213), "ATX"]

## [1] -33.37087

RAFdata[getIndex (RAFdata, 201213), ]

##                     Time   ATTACK      SSLIP    GGVEW   GGVNS    GGVSPD
## 9854 2013-10-01 20:12:13 1.698893 -0.1679749 57.76046 242.881 -10.89062
##           VEW      VNS       ATX      DPXC     THDG       ROLL     PITCH
## 9854 57.74649 242.8225 -33.37087 -55.01767 3.345159 -0.2978565 -1.002424
##          PSXC     QCXC        EWX     RTH1     RTH2     ADIFR      BDIFR
## 9854 320.1704 149.8021 0.03480845 -6.05707 -6.13966 -18.06383 -0.7949235
##          TASX    GGALT     LATC      LONC    PLWC CONCD      WDC      WSC
## 9854 236.3266 9000.719 45.28072 -101.1768 8.39129     0 260.7146 44.72694
##            WIC
## 9854 0.9369057

RAFdata[setRange (RAFdata, 201200, 201210), "TASX"]

##  [1] 239.9797 239.5875 239.3123 239.0751 238.8636 238.4475 238.0735
##  [8] 237.7671 237.5088 237.1420 237.0540

ValueOf (RAFdata, GGALT, 201200)

## [1] 9140.305
```

```
Z <- ValueOfAll (RAFdata, 201200)

##                     Time    ATTACK      SSLIP    GGVEW    GGVNS    GGVSPD
## 9841 2013-10-01 20:12:00 1.666395 -0.1723428 58.18249 244.6779 -10.72564
##           VEW      VNS       ATX     DPXC     THDG       ROLL      PITCH
## 9841 58.17815 244.6206 -34.50745 -54.7943 3.396143 -0.1813508 -0.9280989
##          PSXC     QCXC        EWX      RTH1      RTH2     ADIFR      BDIFR
## 9841 313.9382 152.9558 0.03578181 -6.334847 -6.429331 -18.57253 -0.8421018
##          TASX    GGALT     LATC      LONC     PLWC CONCD    WDC      WSC
## 9841 239.9797 9140.305 45.25227 -101.1864 8.307867     0 263.08 44.49925
##            WIC
## 9841 0.7697609

str(lapply(WACf, function(f) f(RAFdata$ATX))) # All results from WACf

## List of 6
##  $ mean  : num -31.3
##  $ sd    : num 4.98
##  $ sdMean: num 0.287
##  $ median: num -32.6
##  $ length: int 301
##  $ kount : int 301
```

5.3 Constructing plots

Ranadu standard plot routines

Plots can always be constructed using the base graphics routines in R rather than Ranadu routines. Try the following example, which compares a standard plot to those generated by `Ranadu::plotWAC()` and `Ranadu::ggplotWAC()`. The second simply calls `plot()` with some changes in plot parameters: inward tics, tics duplicated on opposing axes, different time labels, etc. The third uses the "grammar of graphics" package `ggplot2` with a special theme. You can tailor any of these to meet your needs. The follow-up function Ranadu::lineWAC() will add lines to a previously constructed plot, but it does not modify the axis limits or add a legend so there are advantages to constructing plots using single calls to `plotWAC()` or `ggplotWAC()`.

```
plot(RAFdata[, c("Time", "GGALT")], type='l')
```

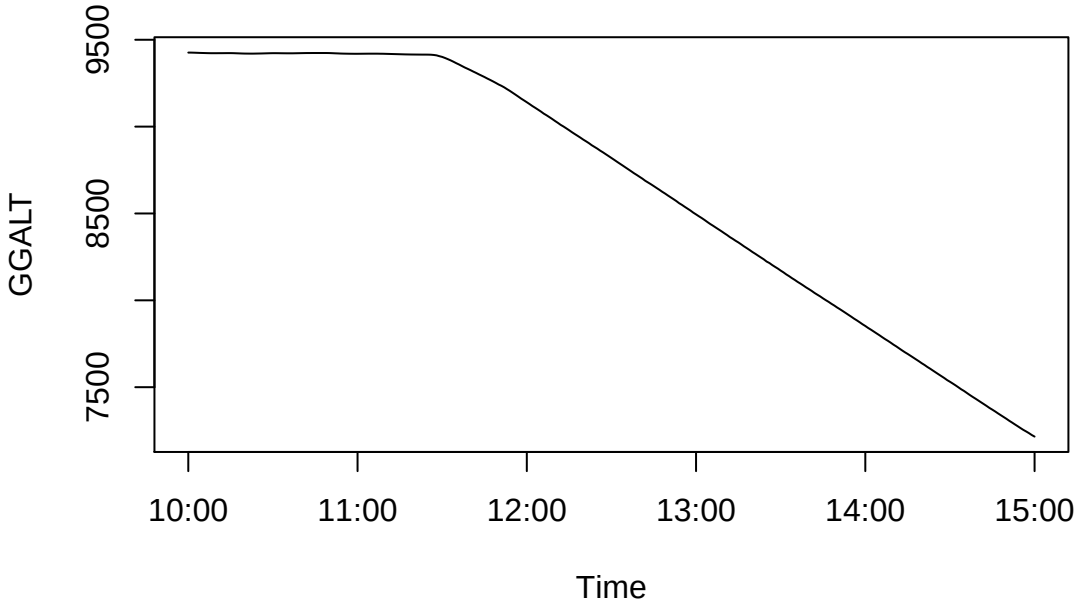


Figure 3: Plotted history of flight altitude as generated by the standard R plot function.

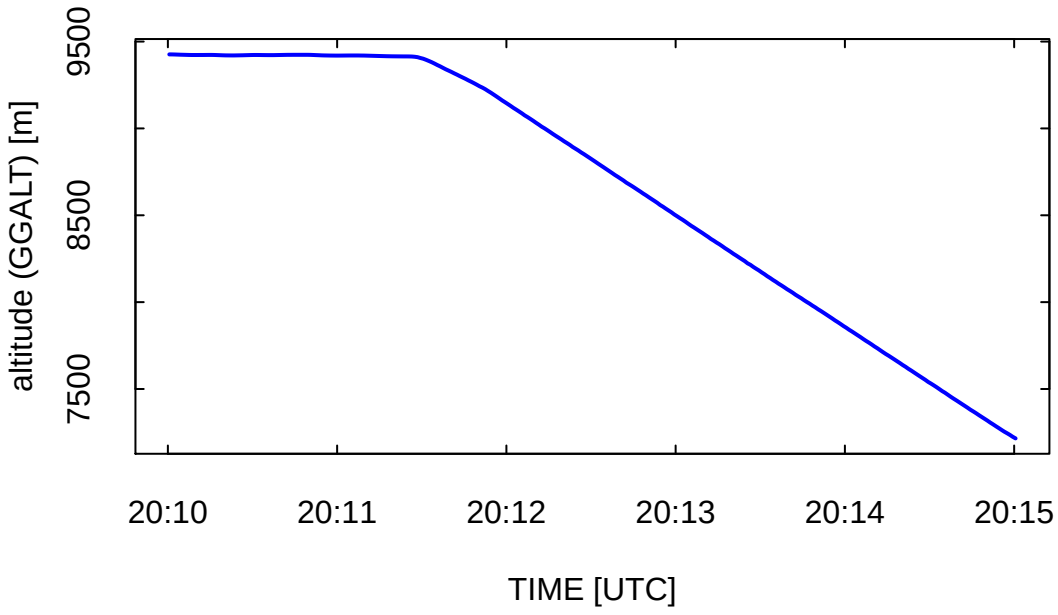


Figure 4: History plot of flight altitude generated by plotWAC()

```
plotWAC(RAFdata[, c("Time", "GGALT")])
```

```
## try plotWAC (subset (RAFdata,, c(Time, GGALT))) ##note absence of " marks
```

```
ggplotWAC (RAFdata[, c("Time", "GGALT")])
```

By default, the last two forms use the "label" attribute, constructed for each variable from the "standard_name" and "units" attributes, for the ordinate-axis label. This can be replaced by explicit assignment of the "ylab" parameter in the function call.

Other manipulations for plots

1. The `Ranadu::setRange()` function can be used to plot a limited range in time.
2. Legends are added automatically by `Ranadu::plotWAC()` and `Ranadu::ggplotWAC()`. See `?legend` for information on how to add or modify or suppress legends.

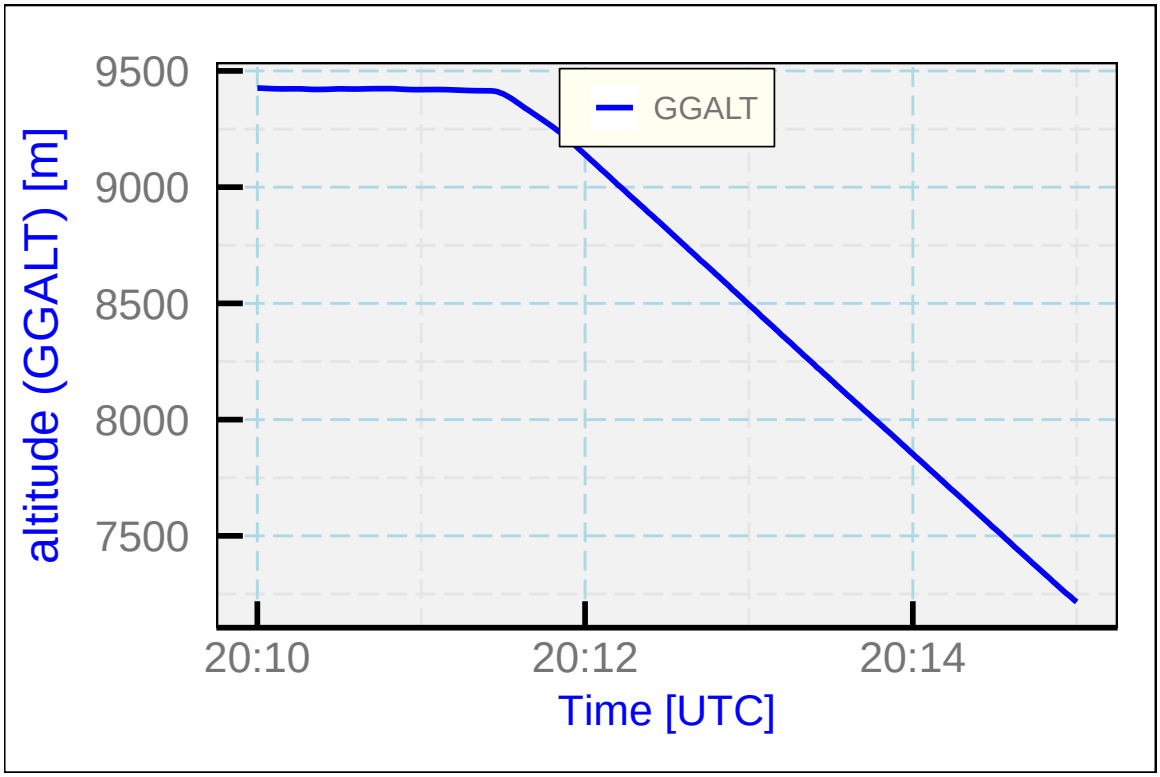


Figure 5: History plot generated by ggplotWAC().

3. The time convention for NCAR/RAF netCDF files is that the time corresponds to the start of the interval over which the measurements are averaged. Therefore Ranadu plot functions plot variables at times displaced from the Time variable by half the time between samples, or 0.5 s for 1-Hz data files. The function `lineWAC()` is provided to be able to add lines to plots with this same offset; if the standard R routine lines() is used, the added lines will be offset 0.5 the sampling interval from the first plot.

4. Many additional parameters can be used as arguments to these plotting routines, for example "lwd=2" or "lty=2" to set the line width or type. You can explore the possibilities by starting with `?par`. For any function, `args(function)` will show the available parameters and their default values.

5. Missing values are handled in special ways in plots, as illustrated in the next code block:

   (a) If a measurement is missing in the data.frame, plotted lines will show a gap.

   (b) If the index itself is missing, plotted lines will connect across the gap. A subset constructed as is R2 below omits the entire row of values when R1$ATX is NA, so the plotted blue line just connects the remaining points and shows no indication of the gap.

   (c) The Ranadu function blankNA ( ) provides a convenient way to set values missing where particular conditions are met, so that gaps in plots will be introduced rather than connecting lines across times where values are missing.

```
R1 <- RAFdata
R1$ATX[setRange (R1, 201100,201120)] <- NA
R2 <- RAFdata[!is.na (R1$ATX), ]
plotWAC (R2[, c("Time", "ATX")])
lineWAC (R1$Time, R1$ATX+0.5, col='red')  ## note 0.5 offset
```

Flight tracks

The routine `Ranadu::plotTrack( )` is provided to plot flight tracks on a background that shows geographic boundaries. In its simplest form it is called with a single argument of a data.frame that contains at least latitude and longitude in variables named LATC and LONC. This routine also has an option to plot the track in a reference frame that drifts with the wind. See ?plotTrack for more information on options.

Size distributions

Ranadu::plotSD ( ) will plot aerosol or hydrometeor size distributions in various formats. The plotted size distribution is the average over all times included in the provided data.frame. There is additional discussion of this function in Section 5.6.

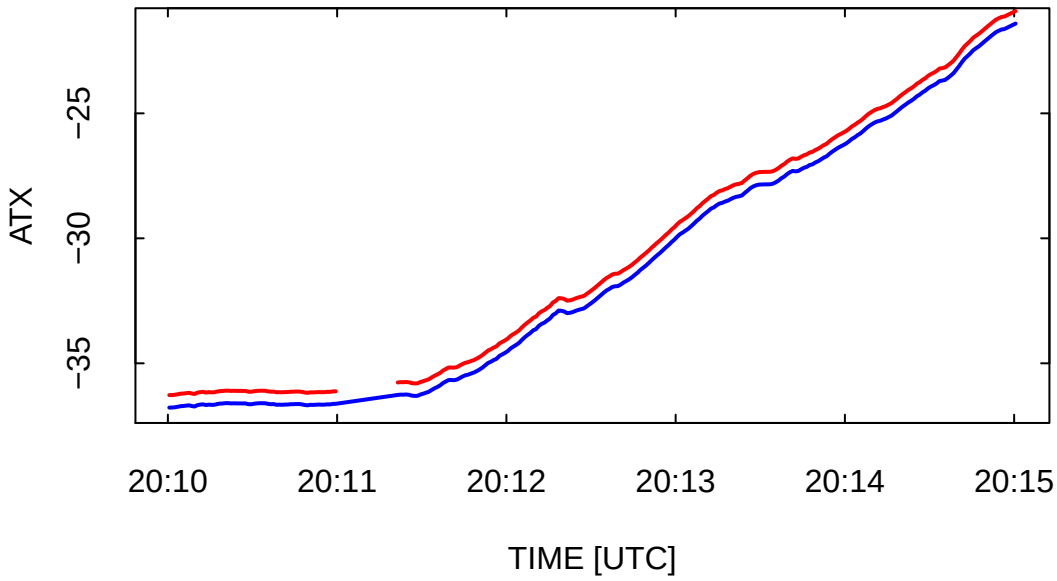


Figure 6: Example of how missing values are handled.

Variance spectra

Several routines for displaying spectral variance are discussed below. See `VSpec( )`, `CohPhase( )` and `Flux( )`.

5.4 Implementing processing algorithms

Many of the algorithms used for calculation of the variables in the netCDF data archives are available as Ranadu functions. These may be useful during data analysis for checking how values might change with different algorithms or to apply the algorithms to new measurements. The processing algorithms are documented in a Technical Note available at this URL. The Appendix lists the Ranadu functions and, under references, notes those that represent descriptions in that technical note.

A few will benefit from additional description, but these are only a select set of the algorithm routines:

1. Two functions, `CorrectPitch()` and `CorrectHeading()`, provide corrections to the measurements of pitch, roll, and heading. They are based on detection of the errors in INS measurements that result in discrepancies between those measurements and measurements from GPS. These are the basis for a manuscript that is accessible from this GitHub repository. For example, an error in pitch results in the acceleration of gravity being rotated into a false component of horizontal acceleration, and this causes a false derivative in the INS-provided ground speed components. This can be detected by comparing to the GPS-provided measurements, so a correction is possible. Similarly, an error in heading results in the accelerations measured in the aircraft reference frame being transformed erroneously into Earth-relative accelerations, so again comparison to GPS can provide an estimate of this error. These two Ranadu functions provide estimates of those errors and can be used to improve the measurements. These are not part of standard NCAR/EOL/RAF data processing.

2. The function `MurphyKoop()` provides the water vapor pressure in equilibrium at a specified temperature. `MurphyKoopIce()` provides the equilibrium water vapor pressure relative to a plane ice surface. References are provided in the technical note.

3. Equivalent potential temperature can be calculated via three different formulas:

   (a) `RossbyEquivalentPotentialTemperature()`: The original Rossby formulation

   (b) `BoltonEquivalentPotentialTemperature()`: The Bolton revision, used in many earlier data archives and still retained as the variable THETAE

   (c) `EquivalentPotentialTemperature()`: The Davies-Jones formula, leading to the variable THETAP (for pseudo-adiabatic equivalent potential temperature) now in standard archives. This is now the favored calculation. It uses the Murphy-Koop values for equilibrium water vapor pressure and otherwise represents the favored equation for analysis, as discussed in the technical note.

A special function `potentialTemperatures( )` will return any of these and also the wet-equivalent and virtual potential temperatures. It is designed for use with `lapply( )` to generate all of these in a form that is displayed concisely. See ?potentialTemperatures and the associated examples.

4. `WindProcessor()` replicates the code for calculation of wind that is represented in RAF Bulletin 23 (except for the omission of a small term representing the motion of the INS relative to the aircraft during turns). It can be used to calculate wind from new sets of measurements, as for example is needed for new sensors of the relative wind or for corrected attitude angles resulting from the first item above.

5. `XformLA()` transforms a vector as measured in the aircraft coordinate system to one in an Earth-relative coordinate system, or vice versa.

6. `KingProbe()` calculates the liquid water content measured by a CSIRO (King) sensor with drift removed by reference to a cloud-droplet spectrometer like a Cloud Droplet Probe. This duplicates the algorithm used for data processing. It is useful because it may be preferable to change the characteristics of the updating and the relationship between Nusselt number and Reynolds number used in this processing.

7. `PCorFunction()` provides the correction for static defect developed in this reference. That has led to significant improvement in the measurements of pressure, airspeed, and temperature, as described in that reference.

8. `StandardConstant(), Gravity()` and `SpecificHeats()` return the values of various constants used in processing, as listed in the section on defined constants in the technical note. For example, some of the constants available are the molecular weights of water and of air, the universal gas constant, the angular rotation rate of the Earth, and Avogadro's constant, among others.

5.5 Some Ranadu utilities

Some of the Ranadu functions that are provided to implement tasks that may be useful are described here:

1. `binStats()` is intended to bin measurements and then compute averages and standard deviations in bins, as might be needed for constructing error-bar plots or soundings vs. height or pressure. (For most uses, equivalent functionality is provided in more standard ways via the 'fields' package for R.)

2. `SmoothInterp()` interpolates to fill regions with missing values and then filters the result to obtain a smoothed variable.

3. ShiftInTime() moves variables forward or backward in time to support shifting variables that are sampled at different times.

4. `makeNetCDF()` makes a netCDF file from a data.frame, provided it has the required characteristics (dimensions, attributes, etc.) If a data.frame is constructed by getNetCDF(), it is usually possible to construct the corresponding netCDF file that will have the originally selected subset of variables from the original file.

5. `LagrangeInterpolate()` implements standard Lagrange interpolation.

6. `DemingFit()` provides a fit that minimizes the perpendicular distance from points to a line, as is appropriate when comparing measurements when neither is known to be a standard.

5.6 Miscellaneous analysis tools

Skew-T thermodynamic diagram

`Ranadu::SkewTsounding()` provides the capability of plotting measurements on a skew-T diagram. The diagram itself has been constructed using modern reference values for the water vapor pressure, adiabatic ascent, and air and water properties and represents a project in itself (documented in this repository. The usual usage is to average measurements of temperature and dewpoint in pressure intervals and then superimpose those measurements on the previously generated background. It is also possible to add multiple soundings to a single plot. See ?SkewTSounding for more information. An example using the short flight segment "`RAFdata`" is shown below:

```
SkewTSounding (RAFdata[, c('PSXC', 'ATX', 'DPXC')])
```

LCL ( ):

The LCL function determines the lifted condensation level given measurements of pressure, temperature, and mixing ratio. The function uses iteration to find the point where the lifted parcel will have a temperature corresponding to the dewpoint. An approximate equation for the LCL is that published by Bolton:

$$T_{Bolton} = \frac{2840}{3.5\ln\theta - \ln(1000e/p) - 4.805} + 55$$

The Bolton equation usually provides answers differing from this routine by only a few 0.01°C, but this routine uses moist-air properties, the Murphy-Koop vapor pressure relationship, and convergence to near-machine-precision.

CAPE ( ):

This function performs a set of calculations on a supplied sounding and returns information useful to evaluate the potential for severe weather and other aspects of instability. In addition to

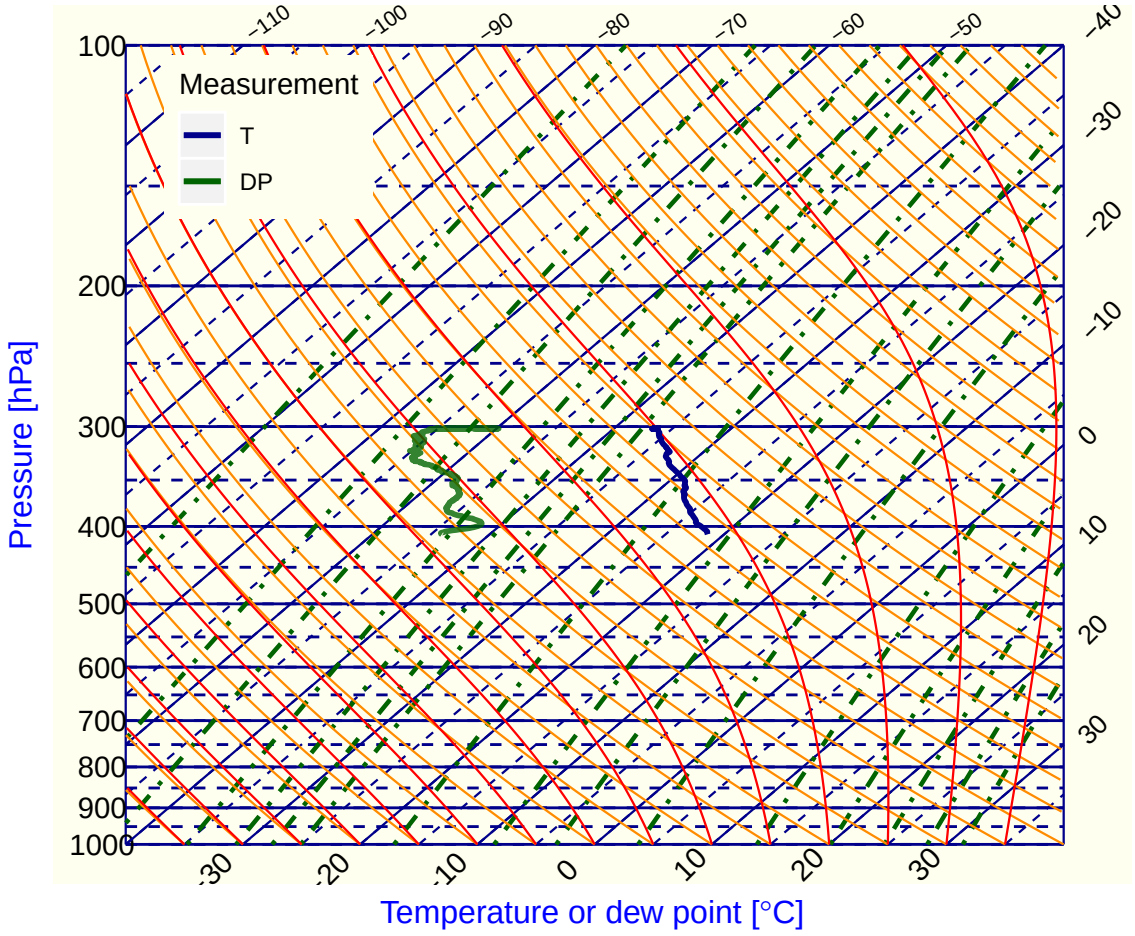


Figure 7: Example of a Skew-T diagram generated by SkewTsounding().

estimating the convective available potential energy, it returns the convective inhibition, level of free convection, LCL determined by averaging a lowest-level layer, and profiles of temperature for pseudo-adiabatic ascent or reversible adiabatic ascent from the LCL. These are in data.frame variables suitable for plotting on the skew-T diagram discussed above. The returned data.frame also includes a column giving the liquid water content for adiabatic ascent and the virtual temperatures for the environment and for the adiabatic parcels, as are needed for calculating buoyancy. If there is no region of positive buoyancy, the function returns only the sounding variables (pressure, temperature, and dew point) and does not add the data.frame attributes discussed below.

Here are some details:

1. The routine begins by some adjustments to the input data, to remove measurements where any are missing at a particular level and to enforce a ceiling on the dewpoint at the corresponding temperature. The latter is often needed because, on descent, the dewpoint instruments on the aircraft sometimes overshoot and produce erroneous measurements that otherwise would contaminate the sounding.

2. The lowest 'player' increment of the sounding (by default, 50 hPa) is then averaged to determine mean values of the lowest-level potential temperature and mixing ratio. These are used, via the 'LCL ( )' function, to find the pressure and temperature at the LCL.

3. Next the measurements are binned according to pressure to give a set of average profiles. If the supplied sounding contains more than one sounding, all are averaged together by this procedure.

4. A new data.frame is then constructed from these binned values of pressure, temperature, and dewpoint, and the corresponding profile of virtual temperature is added to the data.frame as variable TV.

5. Values of the pseudo-adiabatic equivalent potential temperature (using the Davies-Jones equation, and returned as THP') and of the wet-equivalent potential temperature THQ are then calculated at the LCL.

6. Next a set of profiles are added to the data.frame:

   (a) TP, representing the temperature for pseudoadiabatic ascent from the LCL or moist-air descent from the LCL. This profile is calculated at each pressure level above the LCL by finding the temperature that, at the given pressure in the sounding and for vapor pressure corresponding to equilibrium at that temperature, would give the same value of pseudoadiabatic equivalent potential temperature as the value at the LCL, THP. Below the LCL, the same procedure is followed to find the temperature that would give the mean value of potential temperature as foundd in step 2 above. Contrary to standard practice, this is calculated using moist-air, not dry-air, properties like specific heats and the gas constant.

(b) TQ, the temperature analogous to TP but for reversible ascent above the LCL. Below the LCL, the values are the same as those for TP.

(c) LWC, the liquid water content (in units of g/m^3) for ascent from the LCL.

(d) TPV and TQV, the virtual temperatures corresponding to TP and TQ. These are needed for the calculation of buoyancy. In the case of TQV, the virtual temperature is adjusted for the liquid water content contained in the adiabatic parcel.

7. An integration is then performed from the first positively buoyant point above the LCL (the level of free convection or LFC) to the last positively buoyant point (or the top of the sounding) to find the CAPE. The starting and ending points are both interpolated between bounding levels to refine the integration limits, and the integration is performed by constructing, at each level, variables $R_d(T_{pv} - T_v)$ and $\ln p$ where R_d is the dry-air gas constant[8] (287.0653), T_{pv} the virtual temperature for pseudoadiabatic ascent, T_v the virtual temperature of the environment., and p the pressure. A 7-term Lagrangian intorpolation function is used to produce a smooth representation of the sounding, and the R routine 'integrate' is then used to find the integral $\int R_d(T_{pv} - T_v)\, d\ln p$ between the specified limits. It was found that a 7th-order polynomial produced better convergence than lower-orders. The integration uses the method of quadrature documented in R. Piessens, E. deDoncker–Kapenga, C. Uberhuber, D. Kahaner (1983) Quadpack: a Subroutine Package for Automatic Integration; Springer Verlag. It was tested by Euler-method integration, as documented in the code.

8. A similar integration is also performed using TQV, the virtual temperature for reversible adiabatic ascent after adjustment for the retained liquid water content.

9. An analogous integration is performed using TPV from the LCL to the LFC, to determine the convective inhibition.

10. A set of attributes are then associated with the new data.frame to support passing these values as returned arguments from the function. The addributes are: LCLp (pressure at the LCL), LCLt (temperature at the LCL), THP, THQ, CAPE, CAPEW (the CAPE for reversible ascent), CIN (the convective inhibition), MaxLWC (the peak liquid water content for the reversible ascent), and pMaxLWC (the pressure at which this maximum occurs). Note that this is often not at the top of the sounding because at some point dilution from expansion reduces the liquid water content (expressed in mass per volume) more than continuing condensation increases it. See Fig. 8.

11. The new data.frame is then returned from the function. It has the original measurements of pressure, temperature, and dewpoint but now in a set of bins rather than the original data.frame. The returned data.frame can be used as input to SkewT( ) and the columns added to the data.frame can be plotted on that diagram to show the ascenting trajectories, as in the following example:

[8] not the moist-air gas constant, as a consequence of the definition of virtual temperature as the temperature corresponding to the density of dry air having the same density as the moist air

```
Data <- getNetCDF ('/Data/CONTRAST/CONTRASTrf01.nc', c('PSXC', 'ATX', 'DPXC'),
                   Start=250400, End=260100)
suppressWarnings(NSND <- CAPE (Data, nbins=50, player=50))

S <- SkewTSounding (Data, AverageInterval=2)
S2 <- SkewTSounding (data.frame (Pressure=NSND$Pressure,
Temperature=NSND$TP, DewPoint=-120), ADD=TRUE)
S3 <- SkewTSounding (data.frame (Pressure=NSND$Pressure,
Temperature=NSND$TQ, DewPoint=-120), ADD=TRUE)
S <- S + geom_path (data=S2, aes (x=AT, y=P), colour='red', lwd=1.0)
S <- S + geom_path (data=S3, aes (x=AT, y=P), colour='green', lwd=1.0)
plcl <- attr (NSND, 'LCLp')
tlcl <- attr (NSND, 'LCLt')
maxLWC <- attr (NSND, 'MaxLWC')
pmaxLWC <- attr (NSND, 'pMaxLWC')
SP <- SkewTSounding (data.frame(Pressure=plcl, Temperature=tlcl,
                     DewPoint=-120), ADD=TRUE)
S <- S + geom_point (data=SP, aes(x=AT, y=P), pch=19, colour='darkorange', size=4)
labelText <- paste(sprintf('orange dot: LCL %.1f hPa %.2f degC', plcl, tlcl),
                   'red line: pseudo-adiabatic ascent',
                   'bright green line: wet-adiabatic ascent',
                   sprintf ('max LWC: %.2f g/m3 at %.1f hPa', maxLWC, pmaxLWC),
                   sprintf ('cape=%.0f J/kg (adiabatic cape=%.0f)',
                            attr(NSND, 'CAPE'), attr (NSND, 'CAPEW')),
                   sprintf ('conv. inh. %.0f J/kg, LFC=%.0f hPa',
                            attr(NSND, 'CIN'), attr(NSND, 'LFC')), sep='\n')
S <- S + geom_label (aes(x=0, y=2.85, label=labelText), size=3,
                     fill='ivory', hjust='left')
print(S)
```

VSpec():

The function VSpec() generates plots of spectral variance. The help info generated by ?Ranadu::VSpec is fairly extensive, and the function has many arguments, but using the defaults is usually a good starting point. A Shiny-app tutorial on plots of spectral variance is available here, although that tutorial is focussed on the underlying functions used by VSpec(). It explains the various methods of calculating the variance spectrum, shows ways of reducing the uncertainty in the estimates, and discusses the effect of random error or white noise on the spectrum.

The following plot (Fig. 9 illustrates a typical plot of spectral variance that can be generated by this function.

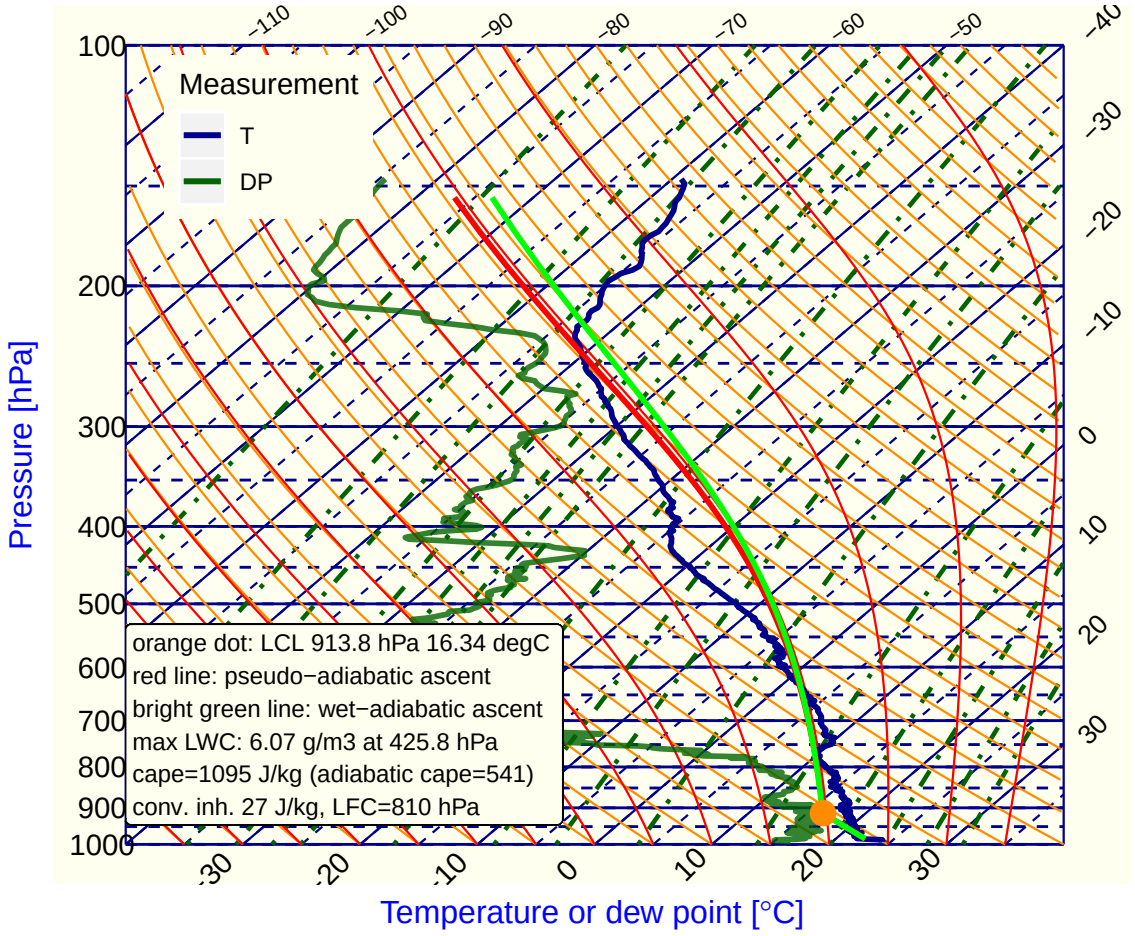


Figure 8: Illustration of the calculations of the LCL and CAPE in the SkewTSounding routine.

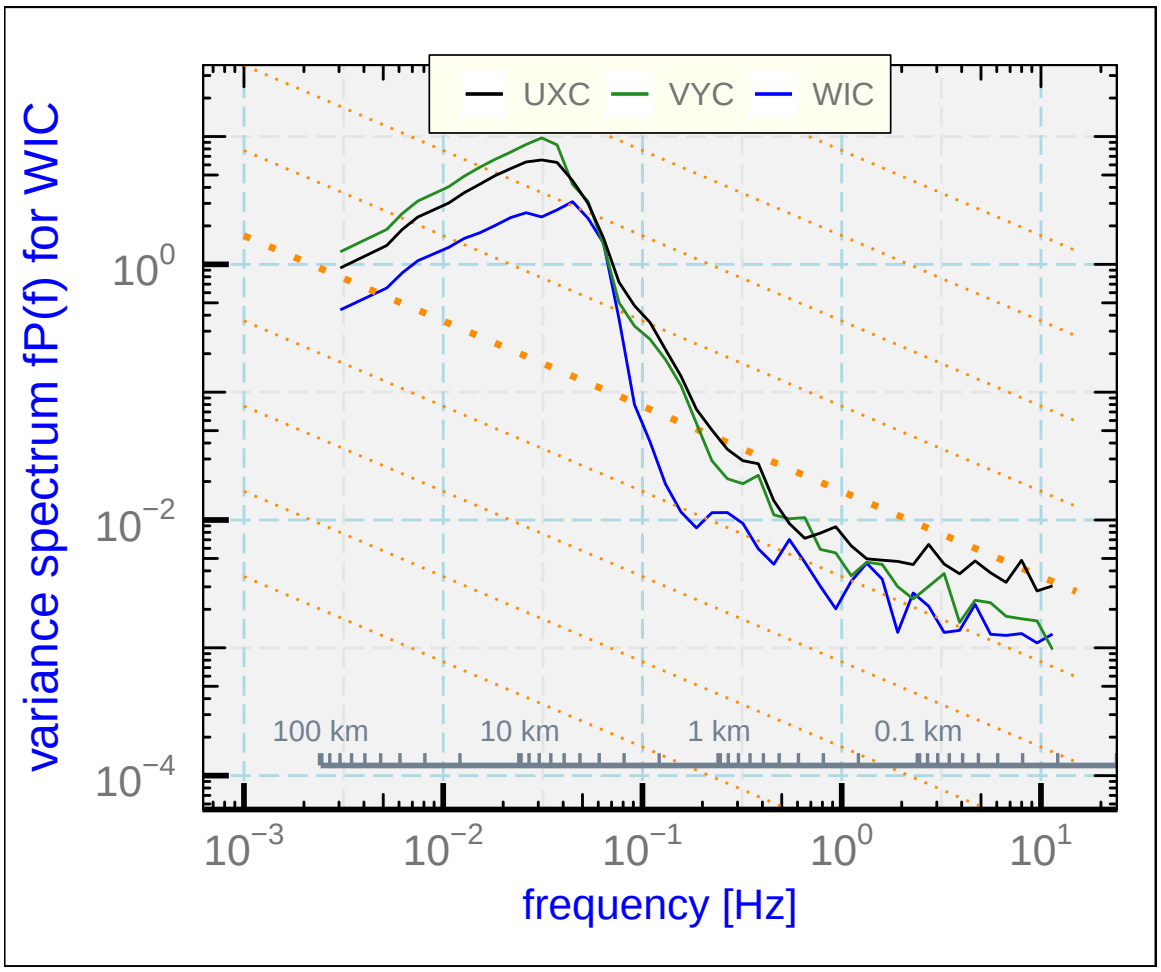


Figure 9: Variance spectra from DEEPWAVE flight 16, 8:30:00 – 8:40:00 UTC. WIC is the vertical wind, UXC the longitudinal wind along the axis of the aircraft, and VYC the lateral wind perpendicular to the other two components.

```
g <- VSpec(Data, 'WIC', smoothBins=50, ylim=c(0.0001,20))
g <- VSpec(Data, 'VYC', smoothBins=50, ADD=g)
VSpec(Data, 'UXC', smoothBins=50, ADD=g) + theme_WAC()
```

The VSpec( ) function is particularly flexible regarding how arguments are provided. These calls are all equivalent: `VSpec(Data, 'WIC')`, `VSpec(Data, WIC)`, `VSpec(Data$WIC)` and `Data %>% dplyr::select(Time, TASX, WIC) %>% VSpec( )`. The last option using pipes is particularly recommended and can include additional steps to select time intervals or calculate new variables.

CohPhase( ) and Flux ( ):

These functions provide additional information for spectral analyses. The first generates plots of the coherence and phase between two variables, and it optionally also returns the cospectrum. Flux( ) displays a plot of that cospectrum and a cumulative contribution to the flux starting from the highest frequencies, so the contribution from all frequencies above a particular frequency can be read from the plot.

plotSD():

Plots of size distributions of aerosol particles or hydrometeors can be generated using this function. An example is shown in Fig. 10. A memo that discusses these size distributions and some of the options available is available here. That memo has some information on VSpec() as well, as does the help information generated by ?Ranadu::plotSD .

```
D <- getNetCDF('/Data/CSET/CSETrf06.nc',
               c('CCDP_', 'C1DC_', 'CUHSAS_'), 173000, 173500)
T <- plotSD(D[,1:4], logAxis='xy', CDF=TRUE,
            FirstCell=c(1,3,1), legend.position='topright')
```

contourPlot():

Scatterplots with large numbers of points often are hard to interpret because the density of points is hard to perceive. The R routine “filled.contour()” provides an alternate way of displaying the density of points in such a plot, and the Ranadu function “contourPlot()” provides an interface to that function that facilitates generation of a two-dimensional plot of event density. See ?Ranadu::contourPlot for more information.

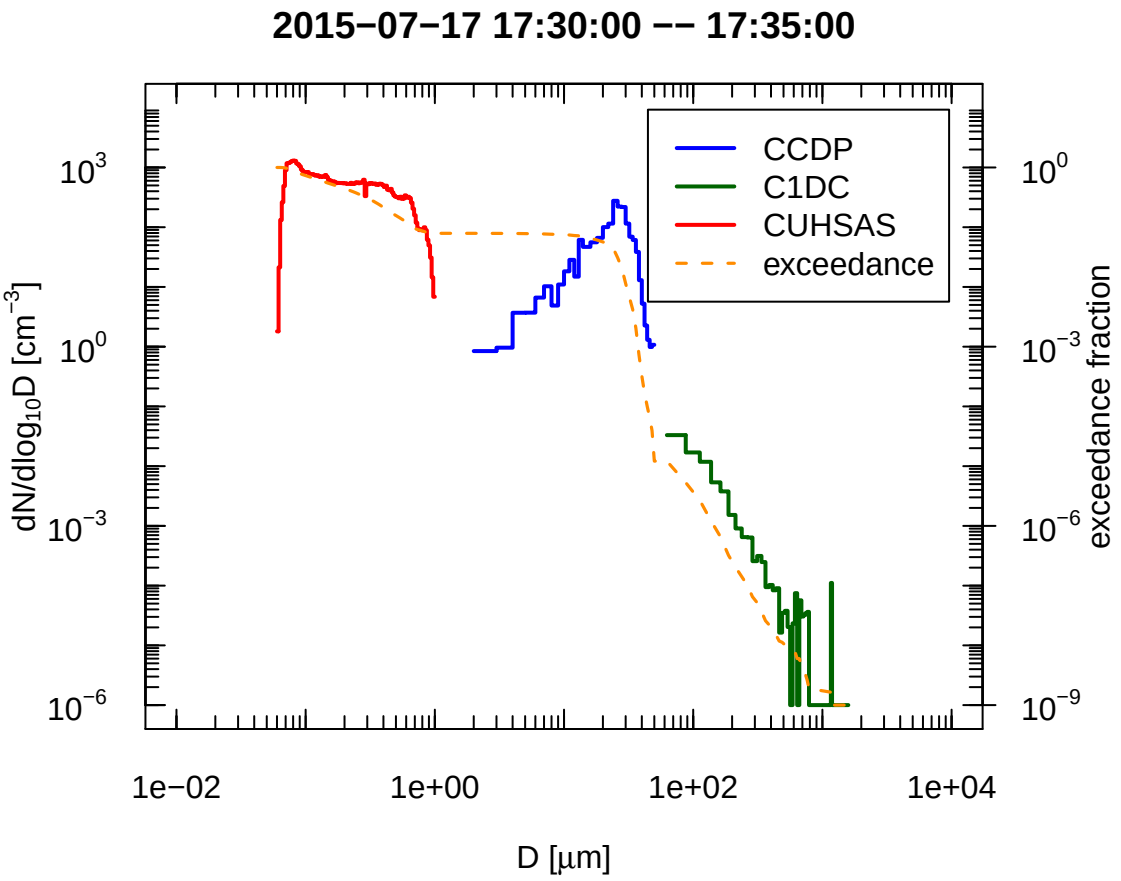


Figure 10: Size distribution measured during the CSET project, research flight 6, 17:30:00 – 17:35:00 UTC.

6 Using R for reproducible data analysis

6.1 Key steps toward reproducibility

There is a growing and beneficial trend toward documentation of research and analysis in ways that permit others to check, duplicate, and extend the work. In R, the package 'knitr' has made a substantial contribution toward that goal by making it possible to blend the text of a document with the R code that generates the results, as discussed in the next subsection. A minimum set of actions toward reproducibility might include these steps:

1. Use of data from public repositories where the data files are assigned DOI numbers and where there is a commitment to long-term preservation and public access.

2. Use of programs, similarly archived and associated with a DOI number where possible. When these programs generate the final document while containing the R code and accessing the original data, they will be reproducible by others.

3. A workflow and/or provenance document describing the steps taken in the analysis project. This goes beyond what is in the code, and it should help those trying to understand the code by providing context and intent for various steps. For example, how were the data collected and by whom, how were they processed before being used in this analysis, what approaches were tried and rejected or abandoned, how does control flow through the program and through any subprograms, how was the final document reviewed and archived, and who are the appropriate contacts for the data, analysis, and interpretation? A good workflow document can be of great aid to someone who wants to repeat the analysis, because snippets of code can be difficult to understand and the reasons for the steps in a program are often obscure.

As an example of one attempt, see the work-in-progress that is contained in this GitHub repository: https://github.com/WilliamCooper/AMTD-AAC.

These principles can also be applied to smaller analysis projects that might need to be repeated. For an example of the latter, the following will describe one use of R and Ranadu to determine appropriate empirical coefficients relating the pressure difference measured between radome ports to the flow angle perpendicular to those ports. This is a common task needed for the measurement of vertical or horizontal wind, and the empirical coefficients may change, for example when the radome is replaced or even re-installed after removal or when sensors are placed in locations that might affect the airflow around the aircraft. The document is this memo giving the recommended empirical coefficients, and it was generated by this program file which combines the R code and LaTeX commands that produced the memo. At the end of the memo there is an appended section called "Reproducibility" that contains these entries:

Reproducibility:	
PROJECT:	WI-HIPPO1
ARCHIVE PACKAGE:	WI-HIPPO1.zip
CONTAINS:	attachment list below
PROGRAM:	AKRDforHIPPO1.Rnw
THIS DOCUMENT:	AKRDforHIPPO1.pdf
WORKFLOW::	WorkflowFindAKRDcal.pdf
ORIGINAL DATA:	/scr/raf_data/HIPPO/HIPPO-1rf01.nc etc
DATA ARCHIVE:	NCAR HPSS (not github)
GIT:	https://github.com/WilliamCooper/Reprocessing.git

Attachments:	AKRDforHIPPO1.Rnw
	AKRDforCSET.pdf
	WorkflowFindAKRDcal.pdf
	SessionInfo

The content of some of these items is as follows:

- Archive package: The content of this package is given by the attachment list, and it is located in the repository listed as the last item. The actual address is this URL.
- Program: The listed program produces the document and contains the R code. It uses 'knitr' as discussed in the next subsection.
- Workflow: In this case the workflow description is a general description of the task, with some discussion of the application to the HIPPO-1 project.
- Original data and Data archive: In this case the original data are in temporary files on EOL computer systems. The reprocessing task to which this document contributes was underway at the time this was written, and those files may change or change location. Therefore a separate archive was constructed that consists of the data.frames used in this analysis. Those are saved on the NCAR HPSS. (They tend to be larger than seems appropriate to place on github.) At this time, these datasets have not been assigned DOI numbers, but the underlying data archive eventually will meet this requirement.
- Attachments: A "SessionInfo" section is included. This is the result of the R command 'sessionInfo()' and includes information like the R version, platform, operating system, packages used, etc.

6.2 Using knitr

As for R itself, this manual will only give very sketchy information regarding the use of `knitr`, but this may help readers of the `'Rnw'` files. This manual itself is constructed from the file `'RanaduManual.Rnw'` in which the R examples and the descriptive code are merged. The file

is basically a LaTeX file into which segments of R code called 'chunks' have been inserted. There is also an alternate format, 'Rmd', in which the text is html (markdown) instead of LaTeX. In that format, the chunks have a different but similar format. In the Rnw format, chunks are set apart from other LaTeX prose by headers and trailers that have this format:

```
<<chunk-name, chunk options>>=
[R code]
@
```

Results contained in R variables can be referenced in the LaTeX part of the document using the '\Sexpr{}' command, and plots can be referenced by a label corresponding to the chunk-name and will be placed in the document at an appropriate location. In the case of this manual, the first chunk has this content:

```
<<initialization,echo=FALSE,include=FALSE>>=
library(knitr)
opts_chunk$set(echo=FALSE, include=FALSE, fig.lp="fig:")
opts_chunk$set(fig.width=6, fig.height=5, fig.align="center",
               digits=4)
thisFileName <- "RanaduManual"
require(Ranadu, quietly = TRUE, warn.conflicts=FALSE)
@
```

This loads the `knitr` library, sets some options, and loads the `Ranadu` package. Here is a small example of how R calculations and text can be merged:

```
<<knitr-example-1, echo=FALSE, include=TRUE>>=
meanTemperature <- mean (RAFdata$ATX)
sdTemperature <- sd (RAFdata$ATX)
@
```

The mean and standard deviation of temperature is -31.33$\pm$4.98, placed here using the command \Sexpr{meanTemperature} where the first number appears and \Sexpr{sdTemperature} for the second.

For more information on using knitr, see the resources at the end of this manual.

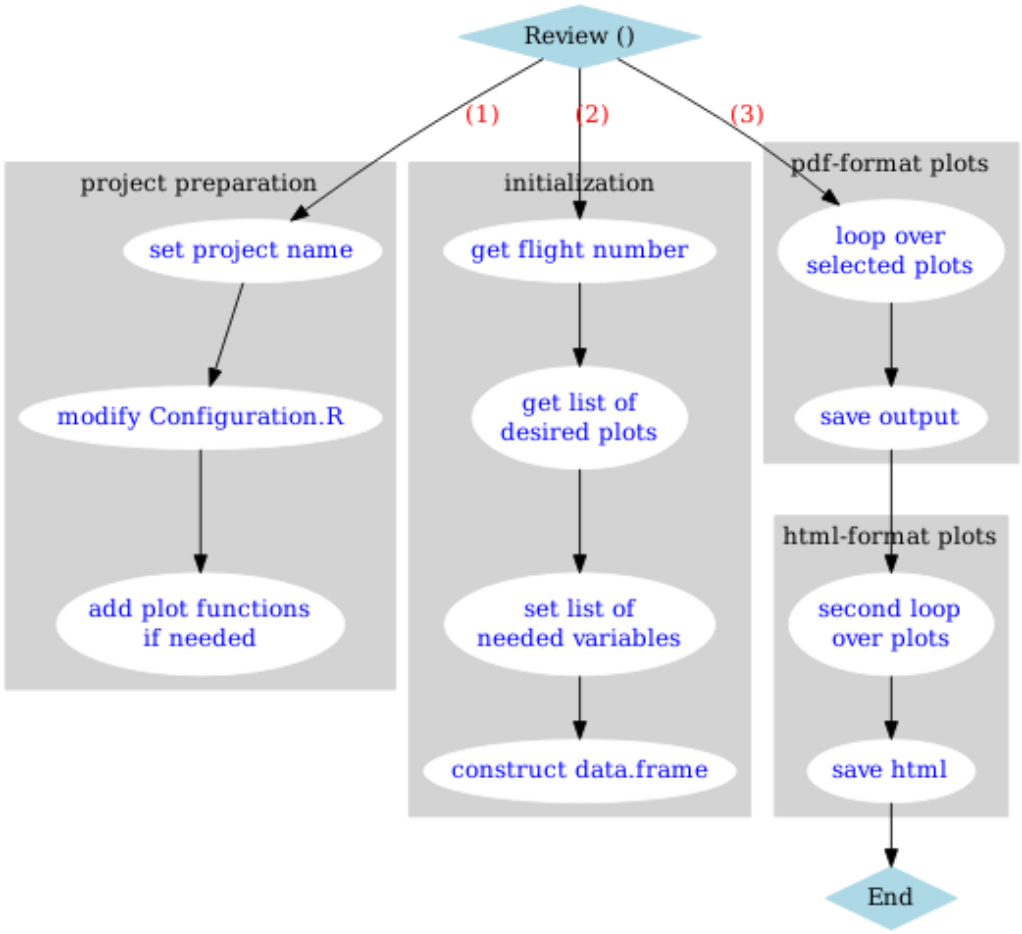
7 An example: Data quality review

7.1 Standard 'Review.R' approach

During field projects, monitoring data quality is essential so that problems can be found and corrected to minimize data loss. One approach to this has been to set up an R script that constructs

plots of key measurements that can be reviewed quickly after flights. These plots are constructed from the preliminary netCDF files produced in the field, and in recent projects they have been stored in the field catalog for projects where they are available to all personnel in the project as well as for documentation after the project.

A flowchart for the Review() program, constructed using the 'DiagrammeR' package for R, is shown below:



1. Project preparation: This only needs to be done once at the start of a project:

   (a) Data files by convention are in a directory with the project name and have names beginning with the project name, so "Project" should be set appropriately.

   (b) There is a 'Configuration.R' file that is read by Review(), and it has sections for each project. A new section should be added that provides the variables used by the various plots. A template from recent projects can be a good starting point. If runs terminate with messages that variables are not present in the data files, the project configuration

should be modified to omit those variables. There is a list of what is plotted by each plot function in the program, so that can help determine what variables are appropriate for the different entries in 'Configuration.R'.

(c) The plot functions used by Review() are located in the directory 'PlotFunctions" and have names like RPlot15.R. There is a model in that directory, named RPlotModel.R, that can be used as a starting point for adding new plots, and the existing plots also can be used as starting points for new plots. Usually no new plots will be needed, and it may be best to get started with the available plots and add others as needed.

(d) The default list of plots for the project also should be defined. It may be useful to skip some plots, e.g., to skip the display of CDP size distributions in cases where no CDP is used in the project.

2. Initialization: This may involve interactive entry of information or entry via the 'Rscript' method of running Review(). See the comments near the start of the program for information on optional methods for running and providing the needed information.

   (a) Each run processes an individual flight, so the flight number must be provided.

   (b) An individual-run set of plots can be provided, or the default set can be accepted.

   (c) The program will read the appropriate section of 'Configuration.R' for the project and construct a list of the variables needed.

   (d) The data file is then read and the needed variables are returned in a data.frame.

3. pdf-format and html-format plots: The program produces plots in two formats, a PDF format that can be magnified without loss of resolution and an html format that is much smaller and faster to read and transfer. For each, it loops through the plot routines, loading them from 'PlotFunctions' if they are needed then running them with appropriate variables. The result will be two files named `PROJECTrfxxPlots.pdf` and `PROJECTrfxxPlots.html` where `PROJECT is replaced by, e.g., DEEPWAVE, the name of the project.`

7.2 'Shiny' approach

A 'shiny' app that provides more interactive control is also available. It provides the same plots and can construct a set of PDF-format plots similar to those from 'Review.R',

but also supports more costomization and interactive displays.

See the repository at https://github.com/WilliamCooper/DataReview.git.

8 Additional resources

These are some resources that have proven useful in constructing and using the Ranadu package. Almost all of the analyses reported here were performed using R (R Core Team [2013]),

with RStudio (RStudio [2009]) and knitr (Xie [2013, 2014]). An example of a good general introductory book on R is Lander [2013]. The "ggplot2" package (Wickham [2009]) was used for some figures. The effort to make these results reproducible benefited greatly from the work represented in these analysis tools, especially those provided by Y. Xie. The book by C. Gandrud (Gandrud [2014]) and material and presentations related to the "Geoscience Paper of the Future" (http://www.ontosoft.org/gpf/) also had a strong influence on the approach to this work. For those wanting more advanced information about R, consider Wickham [2014].

References

C. Gandrud. *Reproducible Research with R and RStudio*. CRC Press, Boca Raton, FL, USA, 2014. ISBN 13:978-1-4665-7284-3.

Jared P. Lander. *R for Everyone: Advanced Analytics and Graphics*. Addison-Wesley Professional, 1st edition, 2013. ISBN 0321888030, 9780321888037.

R Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `http://www.R-project.org`.

RStudio. *RStudio: Integrated development environment for R (Version 0.98.879)*, 2009. URL `http://www.rstudio.org`.

H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009. ISBN 978-0-387-98140-6. URL `http://had.co.nz/ggplot2/book`.

H. Wickham. *Advanced R*. Chapman & Hall/CRC The R Series. Taylor & Francis, 2014. ISBN 9781466586963. URL `https://books.google.com/books?id=PFHFNAEACAAJ`.

Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2013. URL `http://yihui.name/knitr/`. ISBN 978-1482203530.

Y. Xie. *knitr: A general-purpose package for dynamic report generation in R*, 2014. URL `http://yihui.name/knitr/`. R package version 1.6.

– End of Memo –

Appendix

name	purpose	reference[9]
addLabels	add attributes to variables with appropriate labels generated from the "standard_name" and "units" attributes.	called by getNetCDF()
AdiabaticT-andLWC	For adiabatic ascent, calculates the temperature and liquid water content at specified levels	
AirTemperature	Implements the algorithm for calculating ambient air temperature from measurements, with inclusion of the correction for dynamic heating	[a] Section 4.4
AOAfromRadome	Calculates angle of attack from the differential pressure measurements from the radome, applying the standard empirical coefficients.	GV only at present; named GV_...
binStats	A convenience function for binning measurements of one variable in bins of another, then calculating means and standard deviations for the bins. Useful for constructing error-bar plots.	
blankNA	Uses a logical test to set values of variables NA	
BoltonEquivalent-PotentialTemperature	Calculates equivalent potential temperature using the Bolton equation. See also potentialTemperatures.	[a] Sect. 4.6
calcAttack	A function that returns the angle of attack that would be present in the absence of vertical wind. Useful for determining sensitivity coefficients for the angle of attack	[b] Sect.
CAPE	Find convective available potential energy and adiabatic trajectories, given a sounding	
CohPhase	Plot the coherence and phase between two variables.	
ComplementaryFilter	A duplication of the filter used in original processing that combined low-pass measurements from GPS and high-pass measurements from an inertial system to provide a representation of wind that has the long-term stability of GPS combined with the fast response of an inertial system	[a] Sect. 3.4; [b] Sect.
contourPlot	A replacement for a scattergram that shows the density of points, possibly in logarithmic intervals, as colored regions.	
CorrectHeading	Estimate the error in measured heading based on comparison of accelerations in the body frame and Earth reference frame, the latter measured by GPS.	[c] Sect.

name	**purpose**	**reference**[10]
CorrectPitch, CorrectRoll	Estimates the error in measured pitch or roll based on the Schuler oscillation as represented by the difference between ground speeds measured by GPS and by the inertial system providing the measurement of pitch and roll.	cf. this presentation,; [b] Sect.; [c] Sect.
DataDirectory	A convenience function providing the directory in which data reside on different systems, so that code can be written that is portable among systems. (May need modification for new systems.)	
DataFileInfo	Describes the contents (data/time, variables, project, etc) of a netCDF data file	
DemingFit	Implements a fit that minimizes the distance between points and a fitted line, treating pairs of measurements equally in the fit (in contrast to standard regression, treating one as a function of the other)	special ref.
detrend	remove the mean and trend from a variable	
df2tibble	convert from data.frame to tibble and back	
DPfromE	Given the water vapor pressure, calculates the dew point. Uses the Murphy-Koop representation of equilibrium water vapor pressure as a function of temperature.	[a] Sect. 4.5
Equivalent-PotentialTemperature	An implementation of the Davies-Jones (2009) formula for pseudo-adiabatic equivalent potential temperature, the THETAP variable in RAF netCDF files. See also potentialTemperatures.	[a] Sect.4.6
flux	Plot the flux cospectrum, calculate the flux and plot the flux exceedance function.	
GeoPotHeight	geopotential height.	[a] Sect. 3.2 p. 19
getarg	model for flexible specification of variables; not used explicitly in Ranadu, but incorporated into VSpec()	
getAttributes	List the attributes of a data.frame or a column in a data.frame. These duplicate the attributes present in the original netCDF file.	
getIndex	Determine the index (row number) in the data.frame that corresponds to a specific time	
getNetCDF	The starting point for all these functions. Provides a data.frame consisting of specified variables and times from the original netCDF file.	
getProjectData	Assemble a multiple-flight data.frame for a project.	

name	purpose	reference[10]
getRAFData	A specialized routine for transfer of data using NCAR/EOL storage and ftp; not useful except for those with an NCAR/EOL computer account.	[special use]
getStartEnd	Provides the start and end times for a data.frame	
ggplotWAC	An incomplete early try at using ggplot for plots in ways similar to those provided for plotWAC below. CAVEAT: work in progress.	[special use]
GV_AOA-fromRadome	Calculates the angle of attack on the GV aircraft given the pressure measurements on the radome. Same as AOAfromRadome	[a] Sect. 4.7.1
GV_Yaw-FromRadome	Calculates the sideslip angle on the GV aircraft given the pressure measurements on the radome	[a] Sect. 4.7.1
Gravity	A representation of the acceleration of gravity as a function of latitude and altitude	[a] Section 1.3
KingProbe	Calculates the liquid water content sensed by a CSIRO (King) sensor, with correction for the power required in clear air	[a] Sect. 5.1
Lagrange-Interpolate	Standard implementation of Lagrange interpolation	
LCL	Given pressure, temperature, and mixing ratio, finds the lifted condensation level	
lineWAC	Following plotWAC(), lineWAC() can be used to add additional lines to the plot.	
MachNumber	From pressure, dynamic pressure, and humidity, calculate the Mach number representing the flight speed	[a] Sect. 4.7.1
makeNetCDF	From a data.frame generated by getNetCDF, construct a netCDF file containing the measurements in that data.frame	
memCoef	Calculates the Burg-algorithm coefficients for MEM spectral estimation	
memEstimate	Calculates the estimated value of the MEM-based spectral amplitude at the specified frequency or frequencies.	
MixingRatio	Calculation of water vapor mixing ratio as implemented in the original processing	[a] Sect. 4.5
MurphyKoop	A function that provides the equilibrium water vapor pressure as a function of temperature, following the formulation of Murphy and Koop (2005), with optional "enhancement factor" arising from the ambient pressure	[a] Sect. 4.5

name	purpose	reference[10]
MurphyKoopIce	Similar to MurphyKoop but for equilibrium over ice	[a] Sect. 4.5
ncsubset	Simple subsetting routine for netCDF files. Uses ncks, and ncks could be used directly to do the same thing.	[special use]
OpenInProgram	Start 'ncplot' with data from an R data.frame	
PCorFunction	The pressure-correction functions applied to NCAR aircraft to account for the static defect in measured pressure and dynamic pressure	[a] Sect. 4.3
plotSD	Plot size distributions from hydrometeor instruments like the CDP or aerosol-particle sensors like the UHSAS.	
plotTrack	Plot a flight track on a simple representation of geographic boundaries, with an option to plot the track in a coordinate frame drifting with the wind. (Better looking flight tracks can be constructed using the .kml files provided by EOL/RAF.)	
plotWAC	A convenience function that calls base-R plot() with some tailoring of the parameters of the call. See also lineWAC.	
PotentialTemperature	Standard calculation of potential temperature, with an option to apply a correction for moist air properties. See also potentialTemperatures	[a] Sect. 4.6
PressureAltitude	Standard formula for pressure altitude, International Standard Atmosphere	[a] Sect. 3.3
RAFdata	A simple example of a data.frame that can be used in examples.	
RecoveryFactor	The formulas used to correct temperature sensors for the effect of dynamic heating.	[a] Sect. 4.4
removeSpikes	An algorithm for identifying and removing spikes in time series.	
Rmutate	Add variables to a data.frame based on simple formulas provided as arguments to the function. Designed for use in pipes.	
RossbyEquivalent-PotentialTemperature	Implementation of the original Rossby formula for equivalent potential temperature. See also BoltonEquivalentPotentialTemperature and EquivalentPotentialTemperature for other representations, and potentialTemperatures	[a] Sect. 4.6
Rsubset	Create subsets of data.frames while preserving attributes of the data.frame and its columns. (dplyr::select( ) is a perhaps better alternative.)	

name	purpose	reference[10]
selectTime	Select a specified time interval from a data.frame. Designed for use in pipes.	
setRange	Generate a sequence of indices that spans a specified time range, for constructing plots that cover specified periods	
setVariableList	Displays a GUI with variables from a netCDF file, displaying those selected for inclusion in a data.frame and allowing interactive addition and subtraction of variables	[special use]
ShiftInTime	Shifts a variable forward or backward in time to improve timing matches among measurements	
SkewTSounding	Plot measurements of temperature and humidity on a skew-T background. The background was generated by code outside this package, and plots are superimposed on that background. It was generated with modern representation of equilibrium water vapor pressure (Murphy and Koop, 2005) and using full entropy-preserving trajectories with specific heats and gas constant dependent on humidity, and was checked by comparison to the formula of Davies-Jones (2009), so the plot may differ in minor ways from standard plots. A separate github repository, WilliamCooper/SkewT.git, contains the code that generated the background.	
SmoothInterp	Interpolate to fill missing values and smooth via a Savitzky-Golay filter	
SpecificHeats	A representation of the specific heats and gas constant for moist air	[a] Sect. 4.4, pp. 37–38
sqs	Quasi-steady supersaturation and relaxation time	
StandardConstant	Returns a value or set of values as used in processing data from research aircraft	[a] Sect. 1.3
standardVariables	A standard set of variables that can be included in a call to getNetCDF, with optional addition of more variables	
TellAbout	Simple functions to describe R objects and attributes of measurements.	
theme_WAC	A ggplot2 theme used by WAC; others may prefer the standard theme for most uses.	[special use]
transferAttributes	Preserve attributes upon subsetting data.frames	
TrueAirspeed	The standard equation for true airspeed in terms of Mach number and temperature	[a] Sect. 4.7

name	purpose	reference[10]
ValueOf	Quick look-up of values in the data.frame	
ValueOfAll	Like ValueOf but for all measurements in the data.frame at a specified time	
VirtualTemperature	Standard calculation of virtual temperature	[a] Sect. 4.6
Virtual-PotentialTemperature	Standard calculation of virtual potential temperature, with optional adjustment of the specific heats and gas constant for moist air. See also potentialTemperatures	[a] Sect. 4.6
VSpec	Plot variance spectra for measurements.	
WACf	List of statistical functions suited to generating summaries.	
WetEquivalent-PotentialTemperature	The wet-equivalent potential temperature as defined by Paluch (1978); see also Emanual (1994). See also potentialTemperatures	[a] Sect. 4.6
WindProcessor	An implementation of the calculation of wind from basic measurements, incorporating transformations among reference frames and combination of the relative wind and the ground track to obtain wind with reference to the Earth coordinate system	RAF Bulletin 23
XformLA	Transform a vector from the aircraft reference frame to the local Earth-relative (ENU) frame, or v.v.	

Reproducibility:

PROJECT:	RanaduManual
ARCHIVE PACKAGE:	RanaduManual.zip
CONTAINS:	attachment list below
PROGRAM:	RanaduManual.Rnw
ORIGINAL DATA:	/scr/raf_data/
WORKFLOW:	WorkflowRanaduManual.pdf
GIT:	https://github.com/WilliamCooper/Ranadu.git

Attachments: RanaduManual.Rnw
RanaduManual.pdf
SessionInfo