Session 8: Data Review and Other RAF Applications

Also, Some Miscellaneous Advice and Tips

Al Cooper

[1]Research Aviation Facility, Earth Observing Laboratory
National Center for Atmospheric Research

December 17, 2017

MAKING DATA REVIEW MORE COMPREHENSIVE/EASIER

A starting point rather than an end point!

The Goal:

Facilitate review of data to identify problems during experiments.

Two Components

1. Capabilities to generate a fixed set of plots that can be reviewed quickly for initial inspection of key measurements.
   (a) Implemented in scripts that produce standard sets of plots.
   (b) See "Data Review"
2. A shiny app ("QAtools") that supports interactive generation of plots.
   (a) A Shiny Server on the RAF ground station supports this app.
   (b) Requires project-specific set-up.
   (c) GitHub is used for the reference repository.

CURRENT STATUS (Dec 2017)

Component #1: Script Review.R

- Available for data review in the field and set up for all recent projects.
- Part of the batch processing of new data files that is run automatically
- generates both pdf and html files with plots
- plots saved to the field catalog
- Memo describing use and interpretation:
  see this link
- Program and memo-generating program reside on tikal
- Also archived on github here (as Review.zip, part of larger archive)

CURRENT STATUS (Dec 2017)

Component #2: QAtools

- Development is complete and documented. (See User Guide.)
- The code is installed on tikal.eol.ucar.edu (user cooperw), on the RAF ground station, and in GitHub (https://github.com/WilliamCooper/QAtools).
- A Shiny Server has been installed on the RAF ground station and supports the interactive application. The current URL is 128.117.84.138:3838/QAtools.
- The Shiny Server also supports the Ranadu Shiny App, at this URL: 128.117.84.138:3838/Ranadu.
- The latest installed configuration was for the WECAN-TEST program. A new configuration will be needed for SOCRATES, for which tests begin in January 2018.

Only this sketchy description is provided here because there are extensive manual or user guides for these programs.

Some Comments on Ranadu-produced data.frames

1. Want to load every variable? Use "ALL".

R code and result

```
Data <- getNetCDF(sprintf("%sMPEX/MPEXrf05.nc",
    DataDirectory()), "ALL")
length(names(Data))  ## all 352 variables loaded
```

```
## [1] 352
```

Some Comments on Ranadu-produced data.frames

1. Want to load every variable? Use "ALL".
2. data.frames produced by getNetCDF() preserve attributes.

R code and result

```
Z <- getAttributes(Data$ATX)


## [1] "attributes for variable"
## [1] "_FillValue: -32767"
## [1] "units: deg_C"
## [1] "long_name: Ambient Temperature, Reference"
## [1] "standard_name: air_temperature"
## [1] "actual_range: c(-60.5517883300781, 10.5936450958252)"
## [1] "Category: Atmos. State"
## [1] "DataQuality: Preliminary"
## [1] "Dependencies: 1 ATF1"


## getAttributes(Data) lists all global
## attributes. Beware: attributes are lost by
## subsetting:
Data1 <- Data[setRange(Data, 1e+05, 110000), ]
Z <- getAttributes(Data1$ATX)


## [1] "attributes for variable"


## see help for getNetCDF for information on
## how to avoid this loss.
```

Some Comments on Ranadu-produced data.frames

1. Want to load every variable? Use "ALL".
2. data.frames produced by getNetCDF() preserve attributes.
3. [caution: use sparingly] The command "attach()" – consider "with()" instead.

R code and result

```
summary(ATX)  ## this fails, ATX not defined

## Error in summary(ATX): object 'ATX' not found

attach(Data)
summary(ATX)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -60.55  -55.67  -52.75  -47.61  -48.58   10.59

detach(Data)
## better because scope limited to inside parentheses:
with(Data, summary(ATX))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -60.55  -55.67  -52.75  -47.61  -48.58   10.59
```

Some Comments on Ranadu-produced data.frames

1. Want to load every variable? Use "ALL".
2. data.frames produced by getNetCDF() preserve attributes.
3. [caution: use sparingly] The command "attach()" – consider "with()" instead.
4. R uses NA for missing variables. netCDF -32767. missing-values are converted to NA.

R code and result

```
## the variable EW_VXL has 7 missing-value
## measurements
nrow(Data[is.na(Data$EW_VXL), ])


## [1] 7
```

Some Comments on Ranadu-produced data.frames

1. Want to load every variable? Use "ALL".
2. data.frames produced by getNetCDF() preserve attributes.
3. [caution: use sparingly] The command "attach()" – consider "with()" instead.
4. R uses NA for missing variables. netCDF -32767. missing-values are converted to NA.
5. FI <- DataFileInfo (file.address) gives info on a netCDF file.

R code and result

```
fileName <- sprintf("%sMPEX/MPEXrf05.nc",
    DataDirectory())
FI <- DataFileInfo(fileName)
## best to assign to a variable;
## otherwise, a large number of lines will
## be printed. You can then look at the
## individual components.
names(FI)
```

```
##  [1] "Number"    "Project"   "Platform"  "DataFile"  "Sta
##  [6] "End"       "Rate"      "LatMin"    "LatMax"    "Lon
## [11] "LonMax"    "Variables"
```

```
FI$Platform
```

```
## [1] "N677F"
```

```
sort(FI$Variables[51:60])
```

```
##  [1] "ALT_MTP"   "AQRATIO"   "AT_A"      "AT_A2"     "AT_
##  [6] "ATF1"      "ATH2"      "ATTACK"    "ATX"       "BAL
```

BASIC OPERATIONS ON NETCDF FILES

Some useful things to do:

1. Add /modify a variable and/or attributes
2. Create subset files for economy and for reproducibility archives

Examples:

1. Apply Schuler pitch correction "Ranadu::CorrectPitch()"
2. Smooth / interpolate / filter, esp. with centered filters
3. Check calculations of, e.g., wind ("WindProcessor()") or "PCORS" ("PCorFunction()")
4. Add quality flags for variables
5. Add height-above-terrain variable
6. Add LAMS variables like wind to netCDF files
7. Add results from a Kalman-filter processor.

BASIC STRUCTURE

Steps:

1. load libraries
2. open file
3. manipulate
4. close

code:

The library used is ncdf4. A template describing the process of adding variables to a netCDF file is available at this link. This is page 50 from the Kalman Filter Tech Note, available at this URL

ANOTHER EXAMPLE

Add "HOT" = height of terrain, returned by "HeightOfTerrain()"

```
# copy file to avoid changing original:
Z <- file.copy (fname, fnew) ## use a new name
# load data needed to calculate the new variables:
D <- getNetCDF (fnew, c("LATC", "LONC", "GGALT"))
HOT < HeightOfTerrain (D$LATC, D$LONC) # function
NF <- nc_open (fnew, write=TRUE)
varHOT <- ncvar_def (‘‘HOT’’, ‘‘m’’,
   netCDFfile$dim[‘‘Time’’], -32767.,
   ‘‘Elevation of the Earth’s surface below the GV’’)
newf <- ncvar_add (NF, varHOT)   # add variable
# ncatt_put (newf, ‘‘HOT’’, ... )  # add attribute
ncvar_put (newf, ‘‘HOT’’, HOT)     # save values
nc_close (newf)  # closes and saves file
```

CREATE A SUBSET NETCDF FILE

Easy solution for small subsets: ncsubset()

Ranadu::ncsubset will save a subset time range and subset of variables. Requires the program "ncks".

More general solution: makeNetCDF()

- Ranadu::getNetCDF() creates a data.frame with a subset of measurements from a netCDF file, with attributes.
- If further subsetting is desired:
  - edit as desired, e.g., using commands like these:
    DataS <- Data[setRange(Data, Start, End)
    DataS <- Data[, c("Time", "Var1", "Var2", ...)]
    Data$New <- NewVariable ## add a variable
  - Transfer/create attributes for variables (see ?getNetCDF):

```
A <- attributes (Data$VAR)
A$dim <- NULL ## remove this attribute
atttributes (DataNew$VAR) <- A
```

- use makeNetCDF() to write the new netCDF file

Routines that duplicate processing:
(Useful for checking new algorithms or recoded processing)

Functions that duplicate normal processing

1. WindProcessor(): calculates wind variables
2. Various functions for standard variables, including
   (a) AirTemperature()
   (b) EquivalentPotentialTemperature()
   (c) BoltonEquivalentPotentialTemperature() [the old calculation]
   (d) MurphyKoop(): water vapor pressure
   (e) MachNumber()
   (f) etc, – many more
3. GeoPotHeight(): geopotential height
4. KingProbe(): liquid water content calculation
5. etc. – see ?Ranadu index for list

SOME THINGS I HAVE FOUND USEFUL

- **rbind:** concatenate data.frames by rows. Example: Construct a composite data.frame with all the measurements from a project to use when determining sensitivity coefficients, etc.
- **smoothing and interpolation:** Example: see use in Review.R
- **restrictions on data:**
  example: DataV <- Data[Data$TAS > 130,]
- **saving data:** can speed repeated execution:
  save(Data, file="./Data.Rdata"); load ("./Data.Rdata")
- **str(object):** very useful for seeing the 'structure' of an object
- **object.size (object):** how big is the object in memory?
  (dim, length, nrow: complementary information but not size)
- **adding to a list, where c() can mix list entries:**
  vlist[[length(vlist)+1]] <- v
- **remove or add a variable from a data.frame:**
  Data$Var <- NULL; Data["Var"] <- Var
- **common error: misuse of ":" operator;** best to use parenthesis, as in i <- (1:10)*5 or i <- 1:(10*5)

USEFUL PRACTICES

1. Save the data.frame so running can be reproducible.
2. Put segments of code into 'chunks' and load them into scripts, to make it easy to re-use them. 'source()' and 'read_chunk' are useful for this.
3. Use and re-use functions for common tasks like smoothing or summarizing fit results. RStudio makes this particularly easy.
4. I find it's usually easier to use base plots first, then change to ggplot for more attractive plots.
5. Use round(x,digits) and format (x, nsmall) to format output; use options and opts_chunk$set to set global options.
6. If you are going to save a data.frame, make sure the attributes get transfered to it if necessary. These are lost on sub-setting. See HELP with getNetCDF for a way to do this.

Hate typing <- ? In RStudio, use [Alt]-

NEXT TAB: Shiny Apps