EARTH OBSERVING LABORATORY

Memorandum:

16 October 2014

TO:	RSessions file
FROM:	Al Cooper
SUBJECT:	R objects, focusing on the data.frame

R objects

vector

The fundamental object for most of what I will discuss 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 may 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])

## [1] 2

length(x)

## [1] 1

dim(x)

## NULL
```

Vectors are important in R because the language is vectorized, so loops are usually unnecessary. Try this:

```
x <- 1:10
2*x

##  [1]  2  4  6  8 10 12 14 16 18 20
```

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, which is like a vector 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, 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. You can construct an array from a vector by assigning it a dimension via 'dim(x) <- c(dim1, dim2)' as in the following example,[1] 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
```

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

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. Here is a segment of R code that loads a few selected variables from a netCDF file to a data.frame:

```
require(Ranadu, quietly = TRUE, warn.conflicts = FALSE) # my package of routines

## Loading required package:  RJSONIO
##
## Attaching package:  'signal'
##
## The following objects are masked from 'package:stats':
##
##     filter, poly

Directory <- DataDirectory ()     # for portability; sets the local data directory
Flight <- "rf08"                  # select a flight
Project = "CONTRAST"              # select a project
fname = sprintf("%s%s/%s%s.nc", Directory,Project,Project,Flight)
# set variables needed, here a standard list plus GGVSPDB
```

```
Data <- getNetCDF (fname, standardVariables(c("GGVSPDB")), 60000, 60010)
saveDataFile <- 'RSessionsDataFrame.Rdata.gz'
save (Data, file = saveDataFile, compress='gzip')
N <- names(Data)
```

Here are a few explanations of aspects of this code:

1. The 'require (Ranadu)' statement is a reference to a package of routines I have developed to provide 'convenience' functions that make this easier.[2]

2. The 'DataDirectory ()' function, part of Ranadu, simply sets the directory appropriately for the system in use. On tikal, it returns "/scr/raf_data/", on my home computer "/home/Data", etc. I use this because I got tired of changing references to the data directory whenever I transferred programs between systems.

3. I perhaps slipped up by typing the '=' sign when assigning to the variable 'Project'. You can do this anytime; I have not found any compelling reason to use the '<-' assignment operator, although I have followed R advice and tried to adopt that convention. I like the arguments for the logic of this assignment operator, but I wish I hadn't started using it because when I go to another language I'm prone to type '<-' which will be an error.[3]

4. 'fname' is the data file name; here, on tikal, the statement generates '/scr/raf_data/CONTRAST/CONTRASTrf08.nc'

5. 'standardVariables()' is another Ranadu function that returns a character vector with some of the variables I usually want present, like ATX, PSXC, WDC, etc. See the help functions for the standard list. In addition, any arguments, like 'GGVSPDB' here, are added to the list of variables to use; if there are more than one, use 'c("Var1, Var2, ...") as the argument to standardVariables(). You can also just provide a list of variables you want directly to 'getNetCDF ()'.

6. 'getNetCDF ()' is another convenience routine that returns a data.frame containing 'Time' and any variables requested as the second argument. Here that returned value is assigned to the data.frame 'Data'. You can also restrict the retrieved time period to the times between and including the 3rd and 4th arguments.

7. The next two lines save the data in a format easy to reload without the overhead of fetching the full netCDF file, which is strangely slow on tikal. This is part of the structure I'm trying to implement to make analyses reproducible by archiving the data used, esp. when the datasets are small as is this one.

[2]I hope we can rename this 'RAF' if there is sufficient interest and participation, but for now that seemed presumptious.

[3]I think you have to use '=' for assignment to arguments in functions, as I have in the 'require ()' statement.

8. 'names ()' is a standard R function that lists the names associated with an object. Here it lists the names of the variables included in the data.frame. I assigned the names to 'N' because I want to include them in the text, which is done as in the next paragraph. (See the .Rnw source for an illustration of how this is done; '\Sexpr{N}' is used there to include the names in the text.)

The resulting names are Time, ATX, DPXC, EWX, GGALT, LATC, LONC, MACHX, MR, PALT, PSXC, QCXC, TASX, WDC, WSC, WIC, GGVSPDB. The data.frame 'Data' looks like this:

```
##                   Time    ATX     DPXC   EWX GGALT  LATC  LONC  MACHX
## 1  2014-02-01 06:00:00  9.664 -1.74589 5.380  3271 14.14 154.3 0.5517
## 2  2014-02-01 06:00:01  9.600 -0.58043 5.860  3258 14.14 154.3 0.5523
## 3  2014-02-01 06:00:02  9.608 -0.10240 6.067  3245 14.14 154.3 0.5524
## 4  2014-02-01 06:00:03  9.649  0.04999 6.135  3231 14.14 154.3 0.5525
## 5  2014-02-01 06:00:04  9.745  0.18771 6.197  3218 14.14 154.3 0.5521
## 6  2014-02-01 06:00:05  9.832  0.30350 6.249  3205 14.14 154.3 0.5522
## 7  2014-02-01 06:00:06  9.898  0.65490 6.410  3191 14.14 154.3 0.5525
## 8  2014-02-01 06:00:07  9.972  0.56748 6.369  3178 14.15 154.3 0.5528
## 9  2014-02-01 06:00:08 10.021  0.50978 6.342  3164 14.15 154.3 0.5536
## 10 2014-02-01 06:00:09 10.103  0.45882 6.319  3151 14.15 154.3 0.5537
## 11 2014-02-01 06:00:10 10.179  0.53403 6.354  3137 14.15 154.3 0.5538
##       MR PALT  PSXC  QCXC  TASX   WDC   WSC     WIC GGVSPDB
## 1  4.866 3091 693.0 159.2 185.9 59.76 10.64 -0.2153  -13.53
## 2  5.295 3078 694.2 159.9 186.1 57.36 10.54 -0.2423  -13.41
## 3  5.475 3064 695.4 160.2 186.2 56.32 10.52 -0.3595  -13.37
## 4  5.528 3052 696.4 160.5 186.2 55.32 10.58 -0.3402  -13.30
## 5  5.575 3039 697.6 160.5 186.1 54.89 10.55 -0.3907  -13.34
## 6  5.613 3027 698.7 160.8 186.2 53.42 10.54 -0.4517  -13.40
## 7  5.750 3015 699.8 161.3 186.3 51.61 10.53 -0.4760  -13.46
## 8  5.704 3001 701.0 161.7 186.4 49.83 10.57 -0.5229  -13.49
## 9  5.671 2989 702.0 162.5 186.7 47.27 10.72 -0.5907  -13.52
## 10 5.640 2976 703.2 162.9 186.8 45.96 10.79 -0.6423  -13.52
## 11 5.662 2963 704.4 163.2 186.8 44.47 10.80 -0.6980  -13.58
```

Working with data.frames

Addressing elements of a data.frame

You can address particular elements using syntax like the following:

```
Data$ATX[5]

## [1] 9.745

Data[5, 2]           # note the [row,column] syntax

## [1] 9.745

Data[5, ]

##                  Time   ATX   DPXC   EWX GGALT  LATC  LONC  MACHX    MR
## 5 2014-02-01 06:00:04 9.745 0.1877 6.197  3218 14.14 154.3 0.5521 5.575
##   PALT  PSXC  QCXC  TASX   WDC   WSC     WIC GGVSPDB
## 5 3039 697.6 160.5 186.1 54.89 10.55 -0.3907  -13.34

Data[5, "ATX"]

## [1] 9.745

Data$ATX

##  [1]  9.664  9.600  9.608  9.649  9.745  9.832  9.898  9.972 10.021 10.103
## [11] 10.179

Data$ATX[getIndex(Data$Time, 60004)]

## [1] 9.745

Data$ATX[Data$Time == as.POSIXct("2014-02-01 6:00:04", tz='UTC')]

## [1] 9.745
```

('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:

```
Data[Data$TASX > 186.2, ]

##                   Time    ATX    DPXC   EWX GGALT  LATC  LONC  MACHX    MR
## 4  2014-02-01 06:00:03  9.649 0.04999 6.135  3231 14.14 154.3 0.5525 5.528
## 7  2014-02-01 06:00:06  9.898 0.65490 6.410  3191 14.14 154.3 0.5525 5.750
## 8  2014-02-01 06:00:07  9.972 0.56748 6.369  3178 14.15 154.3 0.5528 5.704
## 9  2014-02-01 06:00:08 10.021 0.50978 6.342  3164 14.15 154.3 0.5536 5.671
## 10 2014-02-01 06:00:09 10.103 0.45882 6.319  3151 14.15 154.3 0.5537 5.640
## 11 2014-02-01 06:00:10 10.179 0.53403 6.354  3137 14.15 154.3 0.5538 5.662
##    PALT  PSXC  QCXC  TASX   WDC   WSC     WIC GGVSPDB
## 4  3052 696.4 160.5 186.2 55.32 10.58 -0.3402  -13.30
## 7  3015 699.8 161.3 186.3 51.61 10.53 -0.4760  -13.46
## 8  3001 701.0 161.7 186.4 49.83 10.57 -0.5229  -13.49
## 9  2989 702.0 162.5 186.7 47.27 10.72 -0.5907  -13.52
## 10 2976 703.2 162.9 186.8 45.96 10.79 -0.6423  -13.52
## 11 2963 704.4 163.2 186.8 44.47 10.80 -0.6980  -13.58

Data[setRange(Data$Time, 60005, 60008), ]

##                  Time    ATX   DPXC   EWX GGALT  LATC  LONC  MACHX    MR
## 6 2014-02-01 06:00:05  9.832 0.3035 6.249  3205 14.14 154.3 0.5522 5.613
## 7 2014-02-01 06:00:06  9.898 0.6549 6.410  3191 14.14 154.3 0.5525 5.750
## 8 2014-02-01 06:00:07  9.972 0.5675 6.369  3178 14.15 154.3 0.5528 5.704
## 9 2014-02-01 06:00:08 10.021 0.5098 6.342  3164 14.15 154.3 0.5536 5.671
##   PALT  PSXC  QCXC  TASX   WDC   WSC     WIC GGVSPDB
## 6 3027 698.7 160.8 186.2 53.42 10.54 -0.4517  -13.40
## 7 3015 699.8 161.3 186.3 51.61 10.53 -0.4760  -13.46
## 8 3001 701.0 161.7 186.4 49.83 10.57 -0.5229  -13.49
## 9 2989 702.0 162.5 186.7 47.27 10.72 -0.5907  -13.52
```

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

```
na.omit(Data)

##                   Time    ATX     DPXC   EWX GGALT  LATC  LONC  MACHX
## 1  2014-02-01 06:00:00  9.664 -1.74589 5.380  3271 14.14 154.3 0.5517
## 2  2014-02-01 06:00:01  9.600 -0.58043 5.860  3258 14.14 154.3 0.5523
## 3  2014-02-01 06:00:02  9.608 -0.10240 6.067  3245 14.14 154.3 0.5524
## 4  2014-02-01 06:00:03  9.649  0.04999 6.135  3231 14.14 154.3 0.5525
## 5  2014-02-01 06:00:04  9.745  0.18771 6.197  3218 14.14 154.3 0.5521
## 6  2014-02-01 06:00:05  9.832  0.30350 6.249  3205 14.14 154.3 0.5522
## 7  2014-02-01 06:00:06  9.898  0.65490 6.410  3191 14.14 154.3 0.5525
## 8  2014-02-01 06:00:07  9.972  0.56748 6.369  3178 14.15 154.3 0.5528
## 9  2014-02-01 06:00:08 10.021  0.50978 6.342  3164 14.15 154.3 0.5536
## 10 2014-02-01 06:00:09 10.103  0.45882 6.319  3151 14.15 154.3 0.5537
## 11 2014-02-01 06:00:10 10.179  0.53403 6.354  3137 14.15 154.3 0.5538
##       MR PALT  PSXC  QCXC  TASX   WDC   WSC     WIC GGVSPDB
## 1  4.866 3091 693.0 159.2 185.9 59.76 10.64 -0.2153  -13.53
## 2  5.295 3078 694.2 159.9 186.1 57.36 10.54 -0.2423  -13.41
## 3  5.475 3064 695.4 160.2 186.2 56.32 10.52 -0.3595  -13.37
## 4  5.528 3052 696.4 160.5 186.2 55.32 10.58 -0.3402  -13.30
## 5  5.575 3039 697.6 160.5 186.1 54.89 10.55 -0.3907  -13.34
## 6  5.613 3027 698.7 160.8 186.2 53.42 10.54 -0.4517  -13.40
## 7  5.750 3015 699.8 161.3 186.3 51.61 10.53 -0.4760  -13.46
## 8  5.704 3001 701.0 161.7 186.4 49.83 10.57 -0.5229  -13.49
## 9  5.671 2989 702.0 162.5 186.7 47.27 10.72 -0.5907  -13.52
## 10 5.640 2976 703.2 162.9 186.8 45.96 10.79 -0.6423  -13.52
## 11 5.662 2963 704.4 163.2 186.8 44.47 10.80 -0.6980  -13.58
```

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. Compare the results from plotWAC (Data$Time, Data$ATX) to D <- na.omit(Data) ; plotWAC (D$Time, D$ATX). When I need to omit missing values (e.g., for filtering), I try 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:

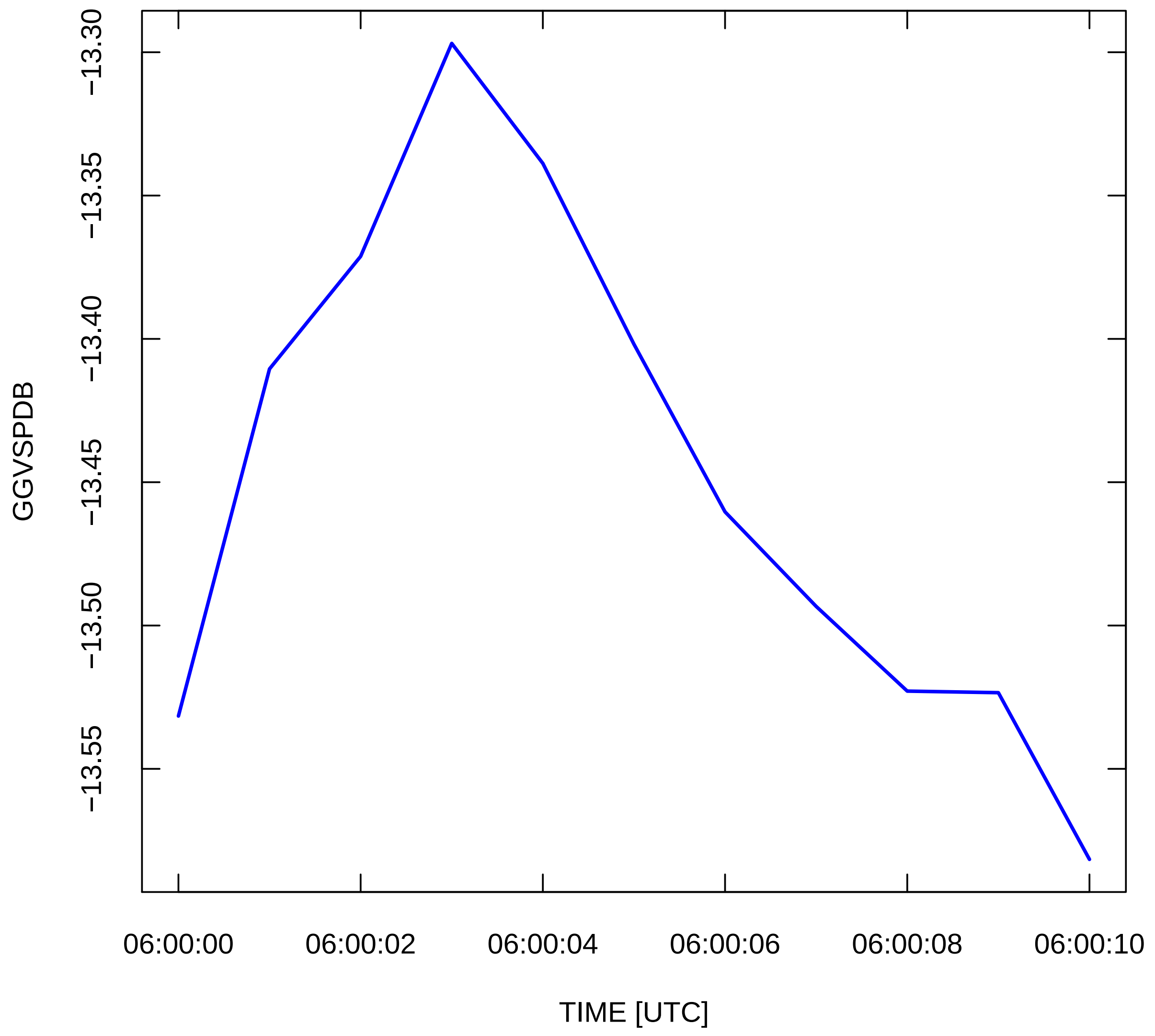
```
# wind component from the east:
Data["UEW"] <- Data$WSC * sin (Data$WDC * pi / 180)
Data$UEW


##  [1] 9.196 8.873 8.751 8.704 8.634 8.464 8.256 8.074 7.877 7.757 7.567
```

Simple plots

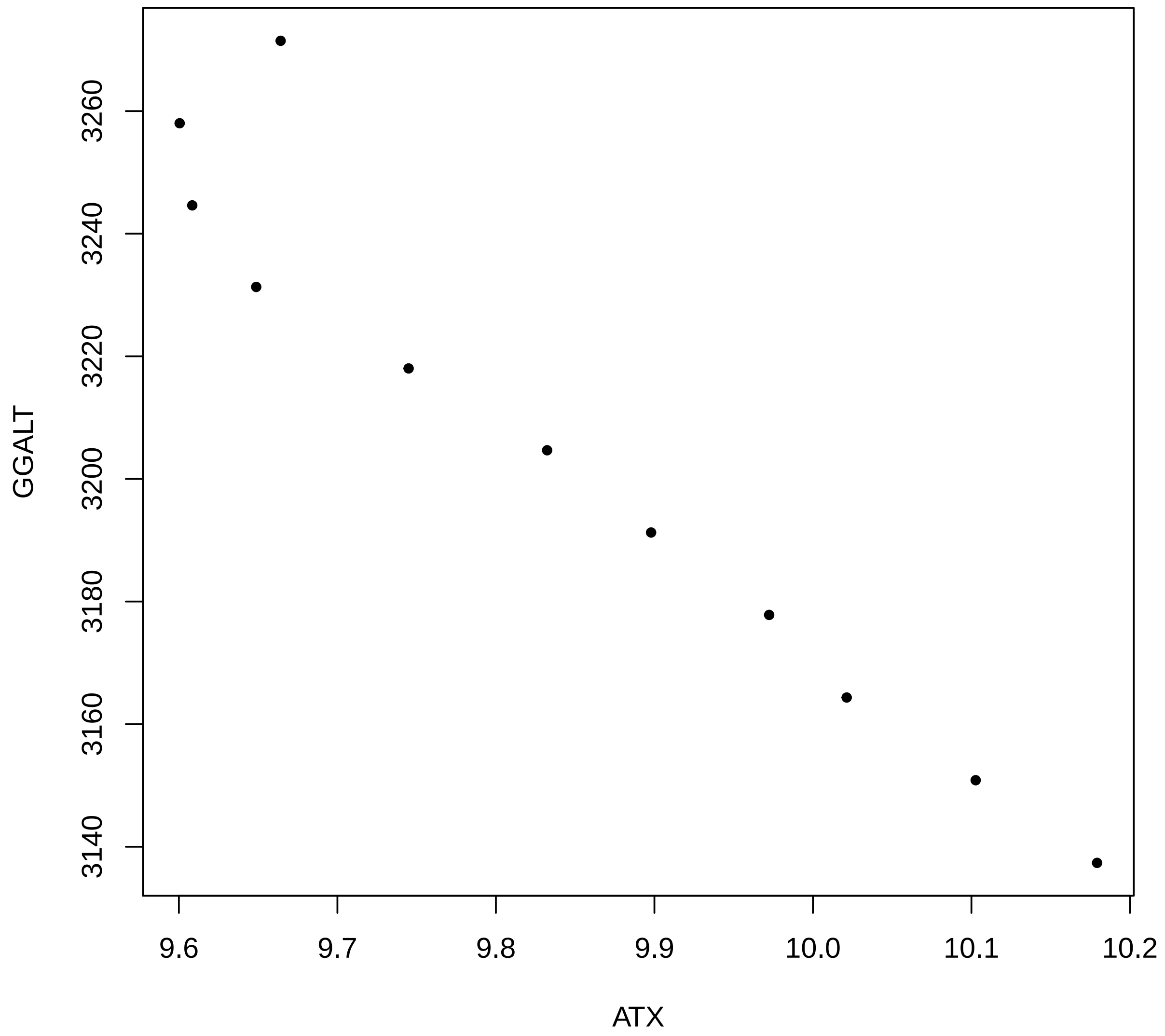
Let’s plot something, using ’plotWAC ()’ from Ranadu for convenience, although you can try the standard ’plot ()’ function from R base routines to see what changes I have made:

```
Z <- plotWAC(Data$Time, Data$GGVSPDB, ylab='GGVSPDB')
```



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:

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



Exercises: Try adding 'lines(D)' after the preceding plot command. Also, see what happens if you instead include three variables in the preceding plot.

Exporting to Excel

and now create an Excel-format spreadsheet with the data:

```
#require(xlsx)
xlsx::write.xlsx (Data, file="Data.xlsx")  # xlsx:: uses package xlsx
#system("libreoffice Data.xlsx")        # include this if your system has libreoffice
```

– End of Memo –

Reproducibility:

PROJECT:	RSessions
ARCHIVE PACKAGE:	RSessionsDataFrame.zip
CONTAINS:	attachment list below
PROGRAM:	/h/eol/cooperw/RStudio/RSessions/RSessionsDataFrame.Rnw
ORIGINAL DATA:	/scr/raf_data/CONTRAST/CONTRASTrf08.nc
GIT:	https:github.com/WilliamCooper/RSessions.git

Attachments: ProgramFile
RSessionsDataFrame.pdf
SessionInfo
RSessionsDataFrame.Rdata.gz

```
system ("zip RSessionsDataFrame.zip RSessionsDataFrame.Rnw RSessionsDataFrame.pdf \\
        SessionInfo RSessionsDataFrame.Rdata.gz")
```