R for Engineers Handout

Jeff Newmiller

2016-05-09

Background

This handout is intended to provide an overview of the prepared materials on using R for engineering calculations and links to useful information on programming in R. Some exposure to basic syntax is assumed, but the intent of these materials is to illustrate how reproducible engineering research can be performed using R and R-Studio. (All of this can be done without RStudio, but it is likely to take rather more effort to get some of it working.)

Concepts

- Three phases: *Input*, *Analysis*, *Output*
- *Input* and *Output* use functions with side effects
- *Analysis* should use functions with no side effects. Such functions often return a (single) list containing results and (possibly) copies of inputs. You may start out experimenting with little steps, but try to wrap up sequences of steps into functions before there are very many steps instead of using copy-paste. Keep track of what you *need* (inputs) and what you are *producing* (outputs).
- Three graphics systems that don't work together: *Base graphics* is not "stretchy", and it has functions that "paint" graphics on the screen. *Lattice* is stretchy and uses "panel functions" to build plots. *Ggplot* is also stretchy but it depends heavily on "long" data frames and can easily add "layers" with consistent interpretation.
- RStudio debugging works best if functions are in their own file.
- Top-level R source file invokes *Input* and *Analysis* statements
- *Knitr* package is used for stitching R-generated graphics and text together. It is an R code text pre-processor that feeds text output with references to image files to `pandoc` or `latex` to make it pretty. *Knitr* files are a good place to put *Output* phase R code.

Reference

R Help

```
help.start() # for general help
```

The manuals are available through `help.start()`, including two particularly useful ones: *An Introduction to R* and *R Data Import/Export*.

```
help("keyword") # or ?keyword for loaded packages
help.search("keyword") # or ??keyword for all installed packages
```

There is also a contributed package `sos` that searches all packages on the *Comprehensive R Archive Network*.

```
install.packages(sos)
library(sos)
???keyword
```

Google will often find questions and answers on stackexchange[1] or in the archives[2] of the `R-help` mailing list.

[1] `http://stackexchange.com`

[2] `https://stat.ethz.ch/pipermail/r-help/`

R Syntax Reference

```
# integer vector (sequence)
v1 <- 1:4
# floating point vector (sequence)
v2 <- seq( 3.3, 3.6, by = 0.1 )
# concatenating multiple vectors
v3 <- c( 4:5, 4.5, 5.1 )
# character vector, integer index into builtin vector
v4 <- LETTERS[ v1 ]
# factor vector
v5 <- factor( v4, levels = c( "B", "A", "C", "D" ) )
# lists can have different kinds of elements
lst1 <- list( AA = "hello", BB = v3, CC = v1 )
str( lst1 )
```

```
## List of 3
##  $ AA: chr "hello"
##  $ BB: num [1:4] 4 5 4.5 5.1
##  $ CC: int [1:4] 1 2 3 4
```

```
# regular indexing of lists gets a shorter list
str( lst1[ 1:2 ] )
```

```
## List of 2
##  $ AA: chr "hello"
##  $ BB: num [1:4] 4 5 4.5 5.1
```

```
# element indexing of lists only gets one element, without the name!
str( lst1[[ 1 ]] )
```

```
##  chr "hello"
```

```
# element indexing of lists can also use list names
str( lst1[[ "AA" ]] )
```

```
##  chr "hello"

# the $ operator is a partial-matching version of [[]]
str( lst1$A )

##  chr "hello"

# data frame is a list but all elements must be vectors of same length
# it is usually best to disable automatic conversion of strings to factors
DF1 <- data.frame( x = v1, y = v2, z = v3, Lbl = v4, stringsAsFactors = FALSE )
str( DF1 )

## 'data.frame':    4 obs. of  4 variables:
##  $ x  : int  1 2 3 4
##  $ y  : num  3.3 3.4 3.5 3.6
##  $ z  : num  4 5 4.5 5.1
##  $ Lbl: chr  "A" "B" "C" "D"

# extract one column (vector) from data frame two ways
str( DF1[[ 1 ]] )

##  int [1:4] 1 2 3 4

str( DF1[ , 1 ] )

##  int [1:4] 1 2 3 4

# data frame with one column
str( DF1[ , 1, drop = FALSE ] )

## 'data.frame':    4 obs. of  1 variable:
##  $ x: int  1 2 3 4

# extract multiple columns of data frame by integer indexing
str( DF1[ , 1:2 ] )

## 'data.frame':    4 obs. of  2 variables:
##  $ x: int  1 2 3 4
##  $ y: num  3.3 3.4 3.5 3.6

# extract multiple columns of data frame by logical indexing
str( DF1[ , c( TRUE, TRUE, FALSE, FALSE ) ] )

## 'data.frame':    4 obs. of  2 variables:
##  $ x: int  1 2 3 4
##  $ y: num  3.3 3.4 3.5 3.6

# extract multiple columns of data frame by name indexing
str( DF1[ , c( "x", "y" ) ] )
```

```
## 'data.frame':    4 obs. of  2 variables:
##  $ x: int  1 2 3 4
##  $ y: num  3.3 3.4 3.5 3.6
```

```
# extract one or more rows of data frame by integer indexing
str( DF1[ 1:2, ] )
```

```
## 'data.frame':    2 obs. of  4 variables:
##  $ x  : int  1 2
##  $ y  : num  3.3 3.4
##  $ z  : num  4 5
##  $ Lbl: chr  "A" "B"
```

```
# extract multiple columns of data frame by logical indexing
str( DF1[ DF1$x < 3, ] )
```

```
## 'data.frame':    2 obs. of  4 variables:
##  $ x  : int  1 2
##  $ y  : num  3.3 3.4
##  $ z  : num  4 5
##  $ Lbl: chr  "A" "B"
```

Watch out for forgetting the comma, as that treats the data frame like a list:

```
str( DF1[ DF1$x < 3 ] )
```

```
## 'data.frame':    4 obs. of  2 variables:
##  $ x: int  1 2 3 4
##  $ y: num  3.3 3.4 3.5 3.6
```

```
# all elements of a matrix must be the same type, unlike data frames
mat1 <- matrix( v1, ncol = 2 ); mat1
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
# elementwise multiplication is the default for matrices
mat1 * mat1
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

```
# matrix multiplication
mat1 %*% mat1
```

```
##      [,1] [,2]
## [1,]    7   15
## [2,]   10   22
```

```
# matrices can have row and column names, though it is not very common
colnames( mat1 ) <- c( "A", "B" ); rownames( mat1 ) <- c( "C", "D" ); mat1
```

```
##   A B
## C 1 3
## D 2 4
```

References