STAT40730 Data Programming with R (online)

Isabella Gollini

Lecture 12 - Advanced R

Advanced R

- To finish off the module, we will go through a number of useful topics for which you now have the necessary ability to go off and explore by yourself.
  - Interfacing R with other languages
  - Make your own R package

Interfacing R with other languages

SC 1

- Background
- Compiling C code
- Writing wrapper functions
- Using `inline`
- Speed-ups

Using other languages with R

- We know that R is generally slow when we have to resort to loops.
- If we can see no way round avoiding a big loop it might be beneficial to write your code in another language.
- R can be linked with other languages such as C, C++, Fortran, Java, Python.
- These languages can be much faster than R so we can use them to create R objects (matrices, lists, etc) that we can then analyse and plot.
- We will cover how to get C code into R as this is, in general, the fastest way to program in R.
- As with other advanced methods we cover, extra installation of software is required for Windows: see http://cran.rstudio.com/bin/windows/Rtools/. Xcode is required on Mac.
- For combining R with Python see the Python program `RPy` or the `rPython` and `reticulate` R packages.

Calling in C code directly

`findsums.c` contains a C function called `findsum`:

```
void findsum(double *data, double *out, int *N)
   {
     int i;
     for(i = 0;i < *N; i++) {
       *out = *out + data[i];
     }
}
```

Then, in a terminal/command prompt window, type:

```
R CMD SHLIB findsums.c
```

or in the R console:

```
system("R CMD SHLIB findsums.c")
```

This last command compiles your C code and checks it doesn't do anything too stupid (but doesn't say whether it will work or not).

To load it into R you need to write a wrapper function - an R function that calls the C code.

Wrapper function

```
dyn.load("findsums.so") # in Windows use dyn.load("findsums.dll")
find.sum <- function(Z) {
  lenZ <- length(Z)
  out <- 0
  ans <- .C("findsum", as.double(Z),
    as.double(out), as.integer(lenZ))
  return(ans[[2]])
}
X <- rnorm(10000)
sum(X)
```

```
## [1] -80.50778
```

```
find.sum(X)
```

```
## [1] -80.50778
```

```
dyn.unload("findsums.so") # in Windows use dyn.unload("findsums.dll")
```

- On Windows this file will be called `findsums.dll`
- On linux/OS X this file will be called `findsums.so`
- The `.C` function will call in the C code compiled in the previous slide
- Note that the `.C` line specifies the type of all the variables (if this goes wrong R will crash).
- The `dyn.load`/`dyn.unload` functions call in the external library `findsums.so` or `findsums.dll`

Using `inline`.

An alternative way of loading in C code is to use the `inline` package

No need to externally compile, just keep your code as a character string and the cfunction function does the rest.

```
library(inline)
findsuminline <- '
    int i;
    for(i = 0;i < *N; i++) {
      *out = *out + data[i];
     }
'
find.sum2 <- cfunction(signature(out = 'numeric', data = 'numeric', N = 'numeric'),
  findsuminline, language = 'C', convention = '.C')
```

```
## ld: warning: text-based stub file /System/Library/Frameworks//CoreFoundation.framework/CoreFoundatio
```

```
find.sum2(0, X, length(X))[[1]]
```

```
## [1] -80.50778
```

- The `cfunction` part declares the variables and language you're using
- The output is a list of all the arguments provided to it - the first here being what we want - out.

Speed

```r
SumVersion <- function(X) return(sum(X))

LoopVersion <- function(X) {
  Y <- 0
  for(i in 1:length(X)) Y <- Y + X[i]
  return(Y)
}

InlineVersion <- function(X) return(find.sum2(0, X, length(X))$out)

library(rbenchmark)
benchmark(SumVersion(X), LoopVersion(X), InlineVersion(X))[1:5]
```

```
##                test replications elapsed relative user.self
## 3 InlineVersion(X)          100   0.015     3.75     0.010
## 2   LoopVersion(X)          100   0.123    30.75     0.120
## 1    SumVersion(X)          100   0.004     1.00     0.004
```

- Here `SumVersion` is an R function using `sum`, `LoopVersion` is a function using a `for` loop, and `InlineVersion` is the function created via the inline package.
- `InlineVersion` nearly as fast as `SumVersion` - a huge speed up over the loops.

Write your own R package

SC 1

- Useful references
- `devtools`

Write your own R package

- Official guide: https://cran.r-project.org/doc/manuals/r-release/R-exts.html
- Guide on RStudio website: https://support.rstudio.com/hc/en-us/articles/200486488-Developing-Packages-with-RStudio
- Before start you have to check that you have:
  - GNU software development tools including a C/C++ compiler; and
  - LaTeX for building R manuals and vignettes.
    - more details are at: https://support.rstudio.com/hc/en-us/articles/200486498-Package-Development-Prerequisi

Using `devtools`

- We use the `devtools` package which contains **Tools to Make Developing R Packages Easier** https://devtools.r-lib.org/
- Great book, available online for Free: *R Packages* by Hadley Wickham http://r-pkgs.had.co.nz/
- Cheatsheet: https://www.rstudio.com/wp-content/uploads/2015/03/devtools-cheatsheet.pdf

```r
install.packages("devtools")
install.packages("roxygen2")
library(devtools)
```

You can check if your system is ready by using:

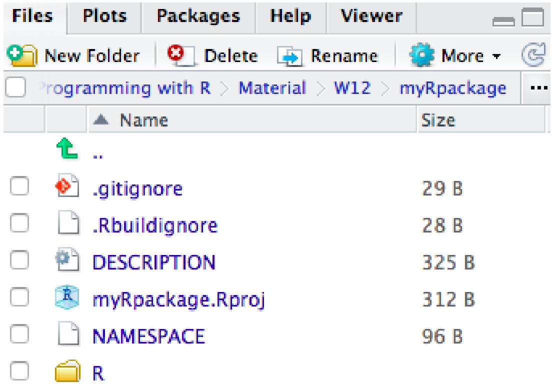
```r
has_devel()
```

TRUE means the system is ready!

Create the package myRpackage.

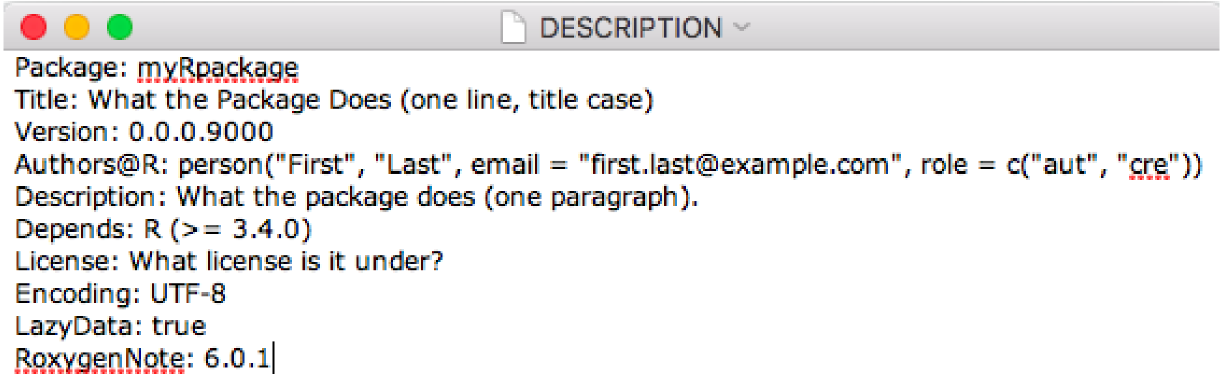
```r
create("myRpackage")
```

- R created the folder myRpackage in your working directory.



Name	Size
..	
.gitignore	29 B
.Rbuildignore	28 B
DESCRIPTION	325 B
myRpackage.Rproj	312 B
NAMESPACE	96 B
R	

- You have the file DESCRIPTION fill it in



```
Package: myRpackage
Title: What the Package Does (one line, title case)
Version: 0.0.0.9000
Authors@R: person("First", "Last", email = "first.last@example.com", role = c("aut", "cre"))
Description: What the package does (one paragraph).
Depends: R (>= 3.4.0)
License: What license is it under?
Encoding: UTF-8
LazyData: true
RoxygenNote: 6.0.1
```

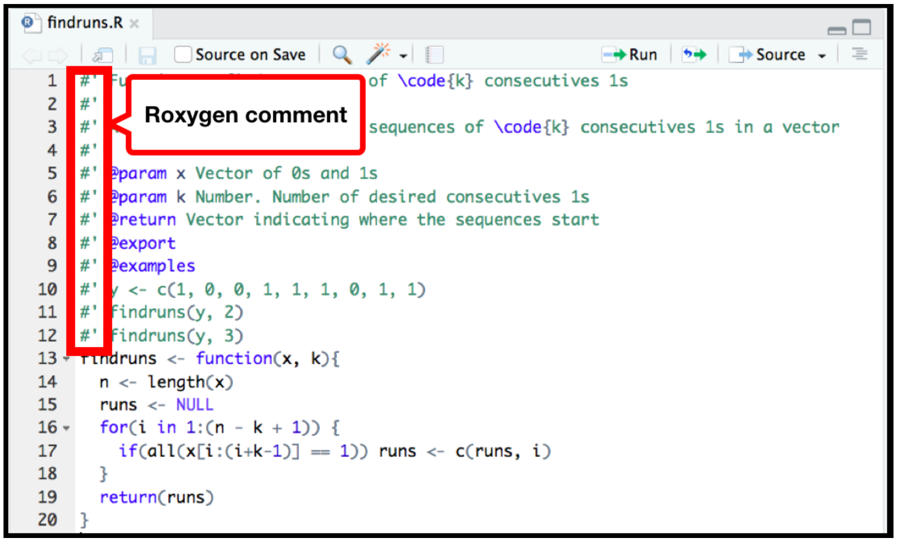
- For more details and other options see: http://r-pkgs.had.co.nz/description.html

Add a function

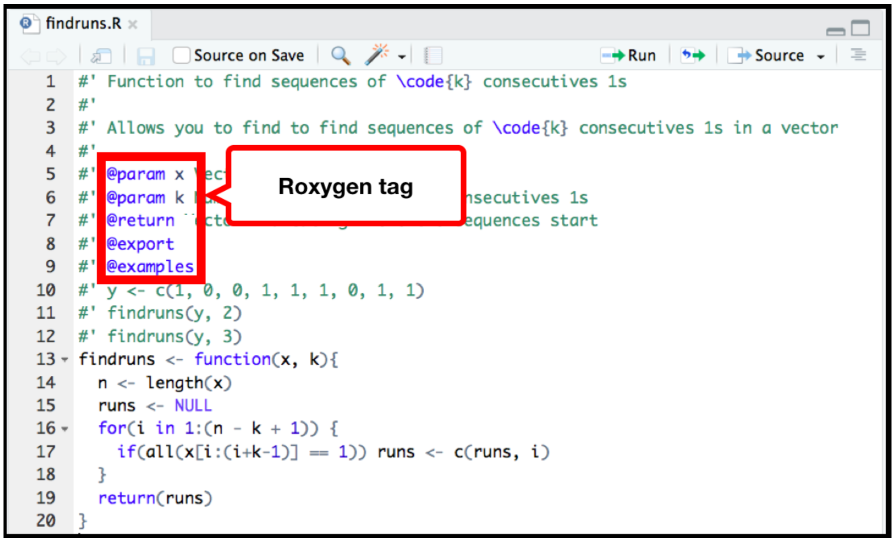
Create a file containing your function, for example findruns.R save it in the folder R. Add the documentation in the findruns.R file by using roxygen2

```r
#' Function to find sequences of \code{k} consecutives 1s
#'
#' Allows you to find to find sequences of \code{k} consecutives 1s in a vector
#'
#' @param x Vector of 0s and 1s
#' @param k Number. Number of desired consecutives 1s
#' @return Vector indicating where the sequences start
#' @export
#' @examples
#' y <- c(1, 0, 0, 1, 1, 1, 0, 1, 1)
#' findruns(y, 2)
#' findruns(y, 3)
findruns <- function(x, k){
  n <- length(x)
  runs <- NULL
  for(i in 1:(n - k + 1)) {
    if(all(x[i:(i+k-1)] == 1)) runs <- c(runs, i)
  }
  return(runs)
}
```

Documentation with roxygen2



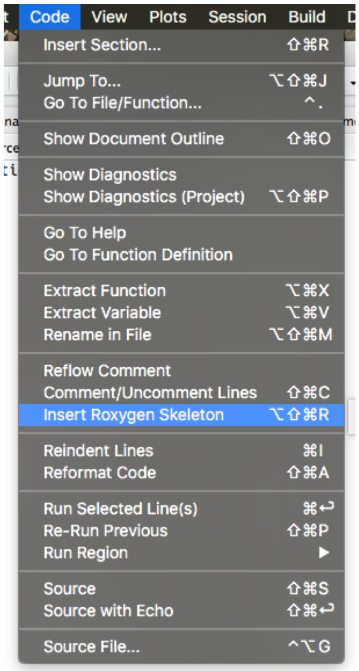
```r
#' Fu                of \code{k} consecutives 1s
#'
#'                   sequences of \code{k} consecutives 1s in a vector
#'
#' @param x Vector of 0s and 1s
#' @param k Number. Number of desired consecutives 1s
#' @return Vector indicating where the sequences start
#' @export
#' @examples
#' y <- c(1, 0, 0, 1, 1, 1, 0, 1, 1)
#' findruns(y, 2)
#' findruns(y, 3)
findruns <- function(x, k){
  n <- length(x)
  runs <- NULL
  for(i in 1:(n - k + 1)) {
    if(all(x[i:(i+k-1)] == 1)) runs <- c(runs, i)
  }
  return(runs)
}
```



```r
#' Function to find sequences of \code{k} consecutives 1s
#'
#' Allows you to find to find sequences of \code{k} consecutives 1s in a vector
#'
#' @param x Vec
#' @param k                  nsecutives 1s
#' @return                   equences start
#' @export
#' @examples
#' y <- c(1, 0, 0, 1, 1, 1, 0, 1, 1)
#' findruns(y, 2)
#' findruns(y, 3)
findruns <- function(x, k){
  n <- length(x)
  runs <- NULL
  for(i in 1:(n - k + 1)) {
    if(all(x[i:(i+k-1)] == 1)) runs <- c(runs, i)
  }
  return(runs)
}
```

5 tags you'll use for most functions

Tag	Purpose
`@param` arg	Describe inputs
`@examples`	Show how the function works
`@seealso`	Pointers to related functions
`@return`	Describe outputs (value)
`@export`	Is this a user-visible function?



Read online about how to document other objects

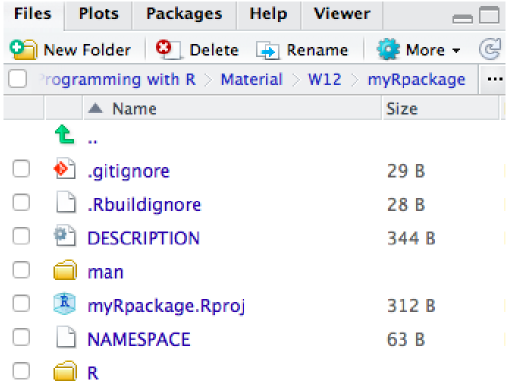
- Data http://r-pkgs.had.co.nz/data.html#documenting-data
- Classes & methods http://r-pkgs.had.co.nz/man.html#man-classes
- Packages http://r-pkgs.had.co.nz/man.html#man-packages

Create documentation files

Create the documentation files with (Cmd/Ctrl + Shift + D)

```
devtools::document()
```

- R created the folder `man` which contains the documentation files - do not modify them!



rogramming with R > Material > W12 > myRpackage

Name	Size
..	
.gitignore	29 B
.Rbuildignore	28 B
DESCRIPTION	344 B
man	
myRpackage.Rproj	312 B
NAMESPACE	63 B
R	

Document workflow

- Modify R comment
- Load the package with `devtools::load_all()` (Cmd/Ctrl + Shift + L)
- Update Rd files `devtools::document()` (Cmd/Ctrl + Shift + D)
- Install package & restart R Install and Restart (Cmd/Ctrl + Shift + B)

Automated checking

- Runs automated checks for common problems in R packages.
- Useful for local packages, even with some false positives.
- If you want to submit to CRAN, you must pass R CMD check cleanly.

Cmd/Ctrl + Shift + E

```r
devtools::check()
```

Now you can share your package

- If you haven't changed the version of the package you have created the source of your package contains the package name and the version you specified in the description file `myRpackage_0.0.0.9000.tar.gz`

```r
install.packages("myRpackage", repos = NULL, type = "source")
library(myRpackage)
?findruns
```

Summary

If you use `devtools` and `roxygen2`:

- Modify the following files:
  - The `DESCRIPTION` file
  - The files in the folder `R`
- Do not modify:
  - `NAMESPACE`

  - The files in the `man` folder

Other tips

- Have your package on GitHub, you can have it public or in a private repository. Unlimited private repository are free for students https://education.github.com/discount_requests/new
- For more details on how to work with Github and R can be found at http://r-pkgs.had.co.nz/git.html
- To have an impact submit the package to CRAN, be aware of CRAN polices https://cran.r-project.org/web/packages/policies.html. See also http://r-pkgs.had.co.nz/release.html.

Lessons from this week

- Write slow code in C .
- Write your own R package!
- Continue to learn how to use R with other modules (e.g. STAT40830)
- Join your Local R User Group!
  - R User Group: https://jumpingrivers.github.io/meetingsR/r-user-groups.html
  - R-Ladies https://rladies.org/

Module content:

1. Introduction to R.
2. Vectors, matrices and arrays.
3. Lists and data frames.
4. Factors and tables.
5. Graphics.
6. R programming structures.
7. Simple statistical programming.
8. Object-oriented programming.
9. Input/output and string manipulation.
10. Debugging code.
11. Performance enhancement and parallel R.
12. Advanced R.

Very best of luck with the exam!