The R Environment for Data Analysis

• R is open source; freely distributed

• Get at http://www.r-project.org/

• We will learn R gradually, in the best way:
  – Get a brief general introduction
  – Use it

• R is extended by packages –
  “install” (download) then “load”
  A package is also called a “library”.
A Little Bit on R

R is based on S developed at Bell Labs in the 1970s by John Chambers. R (and S, and S-Plus, another software package based on S) is an interactive, interpretive, function language.

R is a graphically-oriented data analysis system and object-oriented programming language.

Available on Unix, Linux, MS Windows, and Mac systems. Documentation exists in several volumes, and in an on-line help system.

Get it from http://www.r-project.org/
(Do a Google search for "R").
R Fundamentals

Most statements are of the form

\texttt{variable <- function(...)}

or

\texttt{function(...)}

R is case sensitive.
R Fundamentals

R objects:

variables,
vectors,
lists,
matrices,
arrays,
formulas,
factors (for statistical applications),
data frames (for statistical applications),
fits (for statistical applications)

R has an extensive set of functions, i.e. verbs. The specific meaning depends on the class of the object to which it is applied.
Objects are built by constructor functions:

> A <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8), nrow=2)

This object is a matrix and can participate in matrix multiplication and so on. Typing A results in

A =

1 3 5 7
2 4 6 8

Notice that matrices in R are “column major”.

It’s always a good idea to write R scripts so that they are easily readable. The statement above would look better written this way:

> A <- matrix(c(1, 3, 5, 7, 2, 4, 6, 8), nrow=2, byrow=TRUE)
R Syntax, Functions

All actions are “functions”. A function name is followed by a set of parentheses to enclose arguments.

`help()` or `help(plot)`, e.g. or `?plot`

`q()`

The syntax is somewhat similar to C++, except that there are no special delimiters for statements; statements continue as necessary to complete the function.

Statements may be grouped by `{ and }.

Blanks are significant only when it makes sense for them to be.

The language is not strongly typed.

Numbers are generally single precision floating point numbers.
Examples

> x <- c(2, 5, 3)
> x
> z1 <- 3 + 4*i
> z2 <- complex(real=3, imaginary=4)
> z <- scan(file="example.dat")
> y <- scan()
7: 23. 23. 28. 28. 27. 26.
13: 24. 22. 27. 26. 28. 28.
25:
> a <- 0:10
> b <- 0:10/10
> c <- seq(from=0, t=1, by=.1)
> plot(y,type="1")
General Design

R deals with *functions*. This dictates the syntax – no statement delimiters (but does use `{` and `}`).

Comments begin with `#`.

No fixed naming conventions; the wise user, however, adopts mnemonic conventions. Use periods to represent components.

Various ways of extracting components. `$` component operator: `object$member`

See objects with `objects()`.
R Matrices

A <- matrix(c(1, 3, 5, 2, 4, 6), nrow=3)
x <- c(3, 4)
z <- A[2, ]
w <- A[2, 1:2]
v <- c(1, 2, 1)
b <- A%*%x;
cc <- v%*%A
D <- cbind(A, v)
E <- solve(D)
f <- solve(D, v)
H <- D%*%A
L <- D*D
M <- D^2
objects()
rm(...)
Other Operators and Functions

<, <=, >, >=, ==, !=
&|, !
sin, cos, ...
abs, Arg, sqrt, Re, Im, Conj,
round, trunc, floor, ceiling, sign,
%%,
exp, log, log10
t, crossprod, solve,
sink("filename")
sink()

Plus lots of functions for statistical analyses.
Packages in R

There are many “packages” that extend the functionality of R.

Each package may contain several functions in a given area of application or of analysis.

These must first be “installed” using files that you usually get from some online repository. Once you choose a mirror site from which to obtain the files, you choose the specific package from a drop-down list.

Once a package is installed, it must be “loaded” in any R session in which you need a function from the package.

A package can be loaded by using the drop-down menu under the “Packages” button, or else it can be loaded by means of the library function in R.
Using R; Writing R Scripts

A list of typed commands is called a “program”, a “script”, or a “chunk”.

The advantage of typed commands is that they can be saved and reused.

These typed commands are stored in an ASCII file.

They can either be typed into the active R console or sourced into the active R environment by means of the `source` function in R.
Programming

Conditionals:

```r
if(x >= 3) {
  y <- 2
  z <- 4
} else if (x <= 1) { ## note how the command must be continued
  y <- 1
} else
  y <- 5
```
Conditionals

Conditionals resolve to a numeric value of 0 for false and 1 for true.

They can be operated on.

\[
x \leftarrow 3 \\
y \leftarrow 5 \times (x > 2) + 4
\]

Yields a value of 9 for \( y \).
Suppose the vector $x$ is given.

```r
n1 <- length(x)-1
y <- rep(1,n1)
for (i in 1:n1) {
  if(x[i+1]<x[i]) y[i]<- -1
}
```

or

```r
y <- rep(1,n1)
i <- 1
while (i <= n1) {
  if(x[i+1]<x[i]) y[i]<- -1
  i <- i+1
}
```
Vectorization

Loops are inefficient in R.
Try to implement as statements involving vectors.

\[ y <- \text{ifelse}(x[-1]<x[1:n1],-1,1) \]

Now, let A be a matrix. Instead of

\[
y <- \text{numeric}(\text{dim}(A)[2])
\text{for} \ (i \ \text{in} \ 1: \text{dim}(A)[2]) \ \{ \\
y[i] <- \text{sum}(A[,i]) \\
\}
\]

Use apply:

\[ y <- \text{apply}(A,2,\text{sum}) \]
R Functions for Graphics

plot
plot.factor
pairs
brush
hist
stem
barplot
persp
faces
stars
matplot

And others. The actual appearance of the graph depends on the class of the object.

Arguments for functions may be required or optional. Most required ones are positional, many optional ones are keyword.
Annotating R Graphics

Titles and other text in R graphics can be formatted using `expression()`.

```R
plot(x[,1],x[,2],xlim=c(min(x[,1],x[,2]),max(x[,1],x[,2])),
ylim=c(min(x[,1],x[,2]),max(x[,1],x[,2])),
xlab=expression(italic(x)[1]),ylab=expression(italic(x)[2]),
pch=3)

plot(1:10, 1:10)
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y)",
     cex = .8)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
text(4, 6.4, "expression(bar(x) == sum(frac(x[i], n), i==1, n))",
     cex = .8)
text(8, 5, expression(paste(frac(1, sigma*sqrt(2*pi)), " ",
    plain(e)^{frac(-(x-mu)^2, 2*sigma^2)})))
     cex = 1.2)
```
Controlling the Environment

The environment includes such things as the active objects (in a "library"), the working directory, and graphics layouts.

Graphics layouts are controlled by the R function `par`. ("Parameters", but specifically, **graphics** parameters.)

```r
library(MASS)
oldpar <- par(no.readonly = T)
par(mfcol = c(2, 3)) ### gives two rows and three columns
... 
par(oldpar)
```
Working Directory

It’s convenient to set up a separate directory for a project, such as a class you’re taking.

> setwd("c:/CSI779")
Error in setwd("c:/CSI779") : cannot change working directory

I had not created the directory.

Create it.
(Within your operating system: MS Windows, MacOS, Linux, etc.)
Working Directory

For a directory that exists,

> setwd("c:/CSI779")
> getwd()
[1] "c:/CSI779"

Note the “forward” slash, even in MS Windows instead of the usual backward slash.
Working Directory

Save datasets and programs for the class in the working directory.

Good idea to use extension ".dat" for text datasets and ".R" for R programs.

A simple way to bring a chunk of R code (an R function, for example) into an R program is to use `source("filename")`.

Probably the simplest way to input data into an R program is to use `scan("filename")`.

`scan` just assumes the data are all one long vector, and in the data file the data on a single line are separated somehow, and that new lines are also separators.

`scan` will read to the end of each line in the file.
Using R

Much of the functionality of R can be accessed from a GUI.

The GUI is platform-dependent.

It is generally better to used typed commands than to use the buttons and menus of a GUI.

A list of typed commands is called a “program”, a “script”, or a “chunk”. The advantage of typed commands is that they can be saved and reused.

They can also be read and inspected by someone else.
Using R; Writing R Scripts

There is an emacs-based editor that comes with the standard R distribution package.

The standard GUI provides the ability to execute a highlighted portion of R code in the R editor.

Alternatively, any text editor can be used to write R code. Execution, however, generally requires that text be moused into the R console window.

The advantage of a separate text editor is that they can be more intelligent and even more R-aware than the standard R editor. (Keywords can be colored, etc.)

RStudio is one of the best editors. (For many years, I have used Crimson Editor on MS Windows and emacs with ESS (“emacs speaks statistics”) on other platforms.)
Using R; Saving Graphics Files

There are various commands to save a graphics file. They are, however, platform-dependent.

Graphics files can be save in various formats, .jpeg, .eps, etc.

For example, in Microsoft Windows,

```r
graphdir<-'c:/MyGraphs/'
...
savePlot(filename=paste(graphdir,'plot02',sep='"'),type='eps')
```

saves the active graphics window ("device") in a file called plot02.eps in the directory c:/MyGraphs/.
R Functions for Standard Distributions

Functions

• d – density

• p – cumulative probability

• q – quantile

• r – random number generation
R Functions for Standard Distributions

Distributions

- beta
- f
- gamma
- norm
- t
- unif
- etc.

Examples: 
\texttt{rnorm(25, 100, 8)} generates 25 \text{N}(100,64) numbers
\texttt{qf(.95, 5, 10)} the .95 quantile of an F with 5 and 10 degrees of freedom.
The Object Orientation of R

Most of the functions of R are overloaded and are data-driven.

A new class can be defined by defining a constructor function that gives an identifier to the class:

```r
factor <- function(x, levels = sort(unique(x)),
                  labels = as.character(levels)) {
    y <- match(x, levels)
    names(y) <- names(x)
    levels(y) <- labels
    class(y) <- "factor"
    y
}
```

The function `factor` will construct an object of class “factor”.
The Object Orientation of R

The function `mode` can be used to coerce an object to a particular type:

```
mode(x) <- "factor"
```

The function `inherits` can be used to determine if an object is of a particular class. Functions specific to the class can be written.
R Stuff for This Course: Date Data

POSIX is a portable operating system interface.

One of the most useful aspects of POSIX is the date format.

There are two POSIX date/time classes.

The POSIXct class stores date/time values as the number of seconds since January 1, 1970.

The POSIXlt class stores date/time values as a list with elements for second, minute, hour, day, month, and year.

The POSIXct class is usually the best for storing dates in R.
The default input format for POSIX dates consists of the year, followed by the month and day, separated by slashes or dashes.

For date/time values, the date is followed by white space and a time in the form hour:minutes:seconds or hour:minutes.

2014/1/23

2014-01-23 11:25

2014/1/23 11:25:05
Dates in R

R provides several functions for handling date and date/time data.

The builtin \texttt{as.Date} function handles dates with no times.

The library \texttt{chron} handles dates and times.

The \texttt{POSIXct} and \texttt{POSIXlt} classes allow for dates and times with control for time zones.

It is generally best to use the simpler methods.

Thus, for date only data, \texttt{as.Date} will usually be the best choice.
Dates in R

Dates are stored internally as the number of days or seconds from some reference date.

Thus, dates in R will generally have a numeric mode, and the class function can be used to find the way they are actually being stored.

(The POSIXlt class stores date/time values as a list of components.)

To get the current date, the Sys.Date function will return a Date object which can be converted to a different class if necessary.
Dates in R

The `as.Date` function allows a variety of input formats through the `format=` argument.

The default format is a four digit year, followed by a month, then a day, separated by either dashes or slashes, but `as.Date` will accept other formats.

%{d} Day of the month (decimal number)
%{m} Month (decimal number)
%{b} Month (abbreviated)
%{B} Month (full name)
%{y} Year (2 digit)
%{Y} Year (4 digit)

> as.Date('January 23, 2014',format='%B %d, %Y')
[1] "2014-01-23"
More R Stuff for This Course

- Time series objects
  - **ts** simple time series object used by many R functions; rudimentary handling of date data.
  - **xts** more complicated time series object used by xts, TTR, quantmod packages

- Scripts
  John Nolan’s finance tools `get.stock.price`, etc.

- Packages
  fBasics, fGarch, fPortfolio, quantmod
  others mentioned in text