

Math 131/135/194, Fall 2004

Applied Calculus

Profs. Kaplan & Flath

Macalester College

Lab 1: An Introduction to R

Goal of this lab

To begin to see how to use R.

What is R?

R is a computer package for doing statistics; we will use it during this first semester, but it will really show its power for us next semester. R includes all of the commands needed for basic statistics and also extensive commands for doing advanced statistics. But it also has good capabilities for the topics we will be studying in Applied Calculus: functions of one and several variables, derivatives, differential equations, and linear algebra. Today we will focus on learning the ways to express commands in R.

There are versions of the R package available for Windows and Macintosh and UNIX systems such as LINUX. R is free software. The R language is compatible with an advanced, professional commercial statistics package called S/S-Plus.

R has an interface where you type in commands, and the computer returns responses or creates graphics. This is less “friendly” than point-and-click statistics packages, but the typed-command interface makes it possible for you to write up a concise lab report, and, more important, allows you precise control over the calculations being performed. In addition, the professors (and interested students) can easily install new R commands that allow the labs to be customized to the needs and abilities of the students.

You can use a word-processor (like Word) to cut and paste the R commands and responses into a document that you can edit to include your own commentary and descriptions. You can similarly cut and paste graphics. R produces professional scientific graphics, unlike the chartjunk spewed out by programs like Excel.

We most often want R to do work on data sets. Indeed, a continuous function (like an exponential or a linear function) will be viewed as a ‘connected’ version of a discrete set of points. Exactly how the points get connected together will be explored today. You should think of the discrete set of points – the vector – as a ‘sampling’ of the function’s values. There are a variety of ways to get data into R. A very simple case is described here. Suppose we want to compute some statistic – say the mean – of the data 1.1, 5.4, 3.2, 6.7, 8, and 0.8. We use the command `c` to create a vector. If, as in this

case, you have a very small data set, you can easily do this on the command line:

```
>> s = c(1.1, 5.4, 3.2, 6.7, 8, 0.8)
```

To find the mean, we would then type

```
>> mean(s)
```

It is often useful to read into R data from a spreadsheet. This will be the main way we handle data next semester.

Starting R under Windows

1. Use the START/PROGRAMS/Rstats menu item. A large window called RGUI will appear with a smaller window inside called the “R Console.” You will type your commands in this window.
2. Now you can type your commands to R in the “R Console.” Standard editing commands can be used, including the up and down arrows to re-edit previous commands.

Getting going in R

R is oriented around commands. You type a command in response to a “command prompt.” R evaluates your command and types back a response, or creates some graphics. For example, to calculate the mean of a variable named `a`, you would type

```
>> mean(a)
```

and the computer would respond with something like

```
[1] 4.0
```

To see what is in the variable `a`, just type the name of the variable:

```
>> a
```

```
[1] 1 2 3 10
```

To quit the R session, you give a command

```
>> q() — but don't do this until you're ready to quit!
```

A very common command is to type the name of a variable. R responds by typing back the contents of the variable.

Because R is not point-and-click software, it is fairly easy to make mistakes. This may be frustrating at first, but with a little practice you will become quite comfortable. These are

some suggestions that will help to make your initiation into R smooth:

- Look carefully at the characters being used. Although it is easy to mistake a square bracket (`[`) for a parenthesis (`(`), these have completely different meanings to R.
- When you make a typing mistake, R will often respond with an error message. For example

```
> mean[a]
Error: "subset" type error
```

It should have been `mean(a)`, using parentheses not square brackets.

Or, consider

```
> maen(a)      NB: Spelling error
Error: couldn't find function "maen"
```

You should look carefully at the error message, and remember that most of the time the errors you make are simple things like misspelling a word (maen instead of mean) or using the wrong symbol (`[a]` instead of `(a)`). The error messages usually only provide a hint about what you did wrong, but it is a valuable hint.

- Be very precise in your spelling, and remember that R treats upper-case letters as distinct from lower-case letters. Thus, the variables `a` and `A` are different and unrelated.
- Remember that in R most commands involve the parenthesis, even when there is nothing between the parentheses. For example `q()` is the command to quit R.
- If you type something without parentheses, that is telling R to type back the value of the thing you typed, as opposed to evaluating that thing. For example, type `mean` and you will see the computer program that calculates the mean. (It's pretty complicated looking, because it also handles more advanced calculations such as the trimmed mean.) Alternatively, if you type `mean()` you get an error message that says that you are trying to evaluate the mean of something, but you haven't said what that something is.

Variables and Functions

For our purposes, you can consider that the world of R is divided into two categories:

Variables Variables contain data or the results of calculations. You will be creating and using variables very often.

Functions Functions act on variables to carry out a computation. Please note that this is a more general usage of the word function in the context of the elementary mathematical functions – rational functions, logs and exps, trigonometric functions and their inverses. In general, a 'function' is something that acts on something else ('input→output' model); an elementary math function simply acts on a single number to produce another single number. Below, you see functions that act on vectors of numbers (e.g., `mean`). R provides a very large number of functions that carry out computations. For example, a few basic R functions are

`mean(a)` calculates the mean of the data in `a`.

`max(a)` and `min(a)` compute the maximum and minimum values in `a`.

`plot(x)` makes a plot of `x` where the vertical axis shows the values of the data points in `x` and the horizontal axis shows which data point is first in order, which is second, and so on.

`plot(x,y)` makes a scatter plot of the data in variables `x` and `y`. You can control the appearance of the plot in various ways as we'll describe below.

`c()` collects up its arguments into a vector.

You will use some of these today. It is also possible to write your own functions to carry out specialized computations that are not already included in R. We will only write very simple functions in this course; we'll show you how when the time arrives.

Arithmetic

R is a very convenient calculator for doing arithmetic. The commands for arithmetic look very much like conventional math notation:

```
>> 5 + 4
[1] 9
>> (5 + 4)/3
[1] 3
>> (5 + 4)*3
[1] 27
For exponentiation, use the ^ character. For instance,
2^3 is
>> 2^3
      8
```

Other important mathematical functions are `exp`, `sqrt`, `log`, and the trigonometric functions `sin` and `cos`.

If you have a numerical stored in a variable, you can do arithmetic on all the elements of that variable. For instance

```
>> a = c(1,2,3,4)
>> a^2
[1] 1 4 9 16
```

More on Variables

Variable Names

You can create new variables whenever you need them, and name them according to certain simple rules –

- They must begin with a letter
- They cannot have spaces or underscores or other punctuation characters. The only punctuation character that is allowed in a name is a period.
- Variables should not have the same name as a function that you want to use. This means that you should AVOID names like `mean` `median` and so on. After a bit of experience, you will know what functions you use, and so you will naturally avoid giving variables the same name. However, be aware that there is an important function named `c` and there are important quantities named `T`, `F`, and `NA`. **Never make a variable with the following names:** `c` `T` `F` or `NA`.
- Examples of legal names: `satscores` `sat.scores` `SATscores`
- Examples of illegal names: `2sat` `sat_scores`

It is good practice to use names that help you remember what the contents mean!

Contents of Variables

You can look at the contents of a variable by typing its name. For example, in my workspace there is a variable called `a`; I can see its contents by typing its name.

```
>> a
[1] 1 2 3 10
```

In this case, the variable `a` contains 4 numbers: 1,2,3, and 10. The contents of a variable are generally called the variable's *value*.

Variables can contain different sorts of things.

Numbers Almost of the variables you will be dealing with will contain numbers. A single variable might contain very many numbers.

We'll mostly be using numbers in this course, but you should be aware of other possibilities:

Logical values These appear as `T` and `F` (short for “true” and “false”).

Text These are words that appear in quotation marks.

You can always print out the value of a variable simply by typing its name. This will not affect any subsequent calculations. (Sometimes, when a variable would take a lot of space to print, it will make sense to use the `summary` function to see what is in the variable.)

Giving variables a value

Variables are assigned a value by using the *assignment operator*. This looks like `=` and is used like this.

```
>> m = mean(a)
```

There are many ways that you will use the assignment operator. Try out the following. (The assignment will not print a value; just type the name of the variable by itself to see its value.)

```
pi = 3.141593
a = c(1,2,3,10)
b = c("one", "two", "three", "ten")
```

Finding out what variables you have

Use the `ls()` command to get a listing of the variables in your workspace.

Vectors and Subscripting

A variable that contains more than one value is called a *vector*.

Picking elements from a vector

There are two main ways that you will pick numbers from a vector. They both involve the use of the SQUARE BRACKETS [].

By index. The first number in a vector has index 1, the second has index 2, and so on. To pick one element from a vector, you can just specify its index. For example:

```
>> a[4]
[1] 10
```

You can also specify more than one index. For example:

```
>> a[ c(4,2,3) ]
[1] 10 2 3
```

(Remember that the function `c` creates a vector. Try giving the simple command `c(4,2,3)`.)

By a logical vector containing T and F. For instance, you can create a vector that tells whether each point in `a` is greater than 6.5

```
>> a > 6.5
[1] FALSE FALSE FALSE TRUE
```

You can then use this vector to select elements out of `a`

```
>> a[ a > 6.5 ]
[1] 10
```

More on Functions

Functions are things that perform some calculation on data. There are two things you have to know in order to use a function:

1. What are the *arguments* to the function.

2. What is the value *returned* by the function.

Function Arguments

The arguments to a function tell what data the function is to work on. For example, `mean(a)` tells the function `mean` to work on the data in variable `a`.

Some functions have more than one argument. For example, the function `quantile` takes two arguments,

```
>> quantile(a, 0.75)
```

finds the 75th percentile of variable `a`.

When a function takes more than one argument, the order of the arguments is very important. If you give a function arguments in the wrong order, then the result will be meaningless. (If you are lucky, the function will give an error message, because the arguments don't make sense. If you are unlucky, the wrong arguments will make sense to the function, and it will return a result that will be different from the one you intended to get. You will have no way to know that an error has occurred.)

Some functions have *default arguments*, so that you don't have to specify one or more of the arguments. For instance, here is a function that will be used a lot in Introduction to Statistical Modeling (Math 155):

```
>> quantile(a)
[1] 1.0 1.5 2.5 6.5 10.0
```

The default second argument to `quantile` is `c(0, .25, .5, .75, 1)` which is arranged so that `quantile(a)` returns the *5-number summary* of the data in `a`. That is, `quantile(a)` is exactly the same as `quantile(a, c(0, .25, .5, .75, 1))`.

Some functions have *named arguments*. An example is the function `hist` which has a named argument to control whether the histogram shows the frequency or the relative frequency. Named arguments typically have default values

```
> hist(a, freq=F)      shows the relative frequency
```

The default value is `freq=T` which instructs `hist` to draw the absolute count instead of the relative frequency.

Returned Values

A function does a calculation based on some data, but it does not change that data. Instead, the function *returns a value* which you can display or use in subsequent calculations. If you want to use the result of the calculation in some subsequent calculation, you must do one of three things:

- Assign the returned value to a variable. You can then use that variable in a subsequent calculation. For example, suppose we want to know what fraction of data points in variable `a` are larger than the mean of `a`

```
>> m = mean(a)
>> sum( a > m )
[1] 1
```

We see that only one of the numbers in `a` is larger than

the mean of `a`. (Remember: type the variable's name as a command to see the variable's contents if you want to confirm that R has counted properly. The function `sum` adds up the elements in a vector. If the vector is Boolean (that is, Ts and Fs) then T counts as a 1 and F counts as 0.)

Here are two other ways to obtain the same result:

- Copy over the returned value by hand. For example:


```
>> mean(a)
[1] 4

>> sum( a > 4 )
[1] 1
```
- Use the function as one of the arguments in the subsequent calculation.


```
>> sum( a > mean(a) )
[1] 1
```

In addition to returning a value, functions may also write a little report (for example, try typing `t.test(a, mu=0)`, you will see some output – we are ‘testing’ to see whether or not the mean of the 4 numbers in the vector `a` is 0; we will see what this means later in the year) or produce some graphics. It's important to remember that these reports or graphics do not change the value of any of the variables.

The only way to change the value of a variable is by assignment (that is, using `=`). It's important to remember that computer notation differs from mathematical notation. For example, in mathematics, $a = b$ is the same thing as $b = a$. But the computer statements `a = b` and `b = a` are utterly different. The statement `a = b` assigns a new value to the variable `a`, but leaves the value of `b` unchanged.

Plotting Mathematical Functions

You can make graphs easily using the `plot`, `lines`, and `points` functions. Usually you give these functions two arguments, the sequence of x -coordinates of the points you want to plot, and the corresponding sequence of y coordinates.

For plotting a function, $y = f(x)$, you can follow these steps:

1. Decide what range of x you want to show on the graph. Let's call this `xmin` and `xmax`. For example, we'll set


```
>> xmin = -4

>> xmax = 10
```
2. Decide what spacing you want to have between the tabulated values. Generally it's convenient to specify this in terms of the number of points you want to plot out. 1000 points is often appropriate.

3. Generate a sequence of x points. This can be done with the `seq` function:

```
>> x = seq(xmin, xmax, length=1000)
```

4. Evaluate the function at each of the points in x . For many built-in functions such as multiplication, addition, trigonometry, and so on, this is extremely simple. For instance, here is the cosine function

```
>> y = cos(x)
```

Or, take a periodic function as another example

```
>> y2 = 0.75*sin(3.14159*x) + 1
```

5. Use the `plot` command.

```
>> plot(x,y)
```

If you are unhappy with the axes labels that R has chosen, you can specify your own:

```
>> plot(x,y, xlab = 'x', ylab='f(x)')
```

Each time you use `plot`, the old plot is erased and a new one is started. If you wish to put a second graph on top of the first one, you can use either `lines` or `points`.

```
>> lines(x,y2)
```

You can overlap as many graphs as you like this way. You can also set other aspects of the plot, such as the color of lines, the limits of axes, and whether or not a logarithmic scale is used. This is done using additional, named arguments to the `plot` command. Some of these arguments are: `col`, `xlim`, `ylim`, `log`. An example:

```
>> plot(x,y,ylim=c(-2,10), col='red')
```

Defining your own functions

Sometimes it is convenient to define your own functions, perhaps just to save typing. This is not too difficult once you know the right syntax. To illustrate, suppose we want to define a function $f(x) = x^2 + 1$. The function's name is f , and the value of the function is $x^2 + 1$ where x is the input. In R, we could define this function as follows:

```
> f = function(x){ x^2 + 1 }
```

Once it has been defined, the function can be used in the ordinary way, e.g.,

```
>> f(3)
10
```

If a function has parameters, you can define the function in terms of the parameters and then set the values of the parameter later. For example, consider the function $g(x) = x^b + a$, which has the parameters a and b . We can define this function in the regular way,

```
> g = function(x){x^b + a}
```

If we try to evaluate the function, we get an error message since the parameters a and b have not been defined:

```
> g(3)
Error in g(3) : Object "b" not found
```

All we need to do is define the values of a and b , for instance

```
> b = 2
> a = 1
```

Now things will work fine:

```
> g(3)
[1] 10
```

If we change the values of the parameters, the behavior of the function will change appropriately:

```
> b = 4
> g(3)
[1] 82
```

Cutting and Pasting

To write up your lab reports, you may want to use a word processing program. As part of your write-up, you will generally want to include some R commands verbatim. You can cut and paste from the R command window in the same way you would from just about any program. Try not to overdo it: it's best to cut and paste just what you need to tell your story. As a rule of thumb, only include what you yourself would be interested in reading.

Cutting and pasting graphics is done somewhat differently. First, activate the graphics window so that it is "on top" of the display. Then, use the EDIT/COPY TO THE CLIPBOARD menu item. If you are on a Windows computer, select the AS A METAFILE item to get the best results. Once you have done this, you can paste into the word processor in the usual way. You can resize the figure inside the word processor. Note that graphs don't need to be very big to be effective in most cases.

To hand in

By the beginning of next class, you are to hand in a brief 'lab' report. In it, you are to answer all questions posed below, giving full reasoning for your answers. You must also include graphics and other R computations. You should use a Word document, and copy-and-paste into it from your R session. We will come around and help with this during the hour. You are to hand in one report per group, with all names on the top. Each member of the group will receive the same grade for the report. Before you start, recall that the `help` command can come in very useful. Type `help(cos)` right now and see what happens.

1. Produce a nice graph of $-.85 \sin x$, for $x > 0$. Tell us as much as you can about the domain and range of this function.
2. Produce a nice graph of $x \cdot \exp(1 - x/3)$, for $x > 0$. Tell us as much as you can about the domain and range of this function.
3. Plot the three points $(1, 3), (2, 5), (3, 6)$ in R. These points are not collinear. What is the equation of the line passing through the two points $(1, 3)$ and $(2, 5)$? (Find the equation by hand.) Produce an R plot containing the line and the 3 points.
4. Consider the two functions $f(x) = x^2$ and $g(x) = 3x - 1$. Define R functions **f** and **g** appropriately and use them to draw graphs of these several different functions for the domain $0 \leq x \leq 2$:
 - (a) $f(x) + g(x)$
 - (b) $f(g(x))$
 - (c) $g(f(x))$
 - (d) $g(g(x))$
5. Here is a definition you can give for the Heaviside function which was the subject of Exercise 34 in Section 1.8 of Applied Calculus.


```
> H = function(x){ c(0,1,1)[sign(x)+2] }
```

Don't worry about why the function is written this way, but do type in the above definition exactly as printed above. Then, all on one graph, for the domain $-3 \leq x \leq 3$, plot the several functions

 - (a) $2H(x)$
 - (b) $H(x) + 1$ using `col='red'`
 - (c) $H(x + 1)$ using `col='green'`
 - (d) $H(-x)$ using `col='blue'`
 - (e) $-H(x)$ using `col='orange'`

Put all of the plots on a single plot. This means that you will have to set the `xlim` and `ylim` values appropriately in the first `plot` command so that all the functions can show up.