

Using R for mathematics

28th September 2006

R is a programming language and an “environment” for numerical computation of many types. It is available for free¹ at <http://www.r-project.org>. This document is meant to be a very basic introduction to using R as a replacement for a graphing calculator and/or spreadsheet. R has many more uses, some of which are described in documentation available (free) on the R web site.

1 Using R as a calculator

When you start R, you’ll see a window with a prompt like

```
>
```

at which you can type an arithmetic expression and have it evaluated, like

```
> 2+3
[1] 5
>
```

where the [1] can be ignored for now (we’ll see what it’s for later), and the answer 5 follows. R is meant to do lots of calculations at once, and to support that, there is the `c()` function, for “concatenate” (join together). You use it like

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

If you wanted to compute the values of $7x + 2$ for $x = 1, 2, 3, 4$ you could do so like this:

```
> 7*c(1,2,3,4) + 2
[1] 9 16 23 30
>
```

¹Licensed under the “Gnu Public License” or GPL, see Free Software Foundation <http://www.fsf.org>

(Note that like in most of the computer world, “*” means multiply in R.) In this way, R can do most spreadsheet-type tasks.

It’s convenient to store data values, intermediate calculations, etc., and in R you do that as follows.

```
> x <- c(1,2,3,4)
>
```

Now you can refer to x anytime. For example,

```
> 7*x + 2
[1] 9 16 23 30
>
```

You can see what things are stored with ls()

```
> ls()
[1] “x”
>
```

If you don’t need x anymore, you can delete it with rm():

```
> rm(x)
>
```

It’s gone, but we can check anyway:

```
> ls()
character(0)
>
```

The “character(0)” means “nothing”: the ls() function returns a list of names, and names are a type of character strings, so here ls() returns a list of 0 character strings.

2 Graphing

R is very good at making graphs. Suppose we wanted to plot the following points: $A = (0, 1)$, $B = (0.1, 0.7)$, $C = (1, 2)$, $D = (1.2, 3)$, $E = (2, 4)$. We could do

```
> plot(x=0,y=1)
```

which sets up axes and plots the point (0,1), then

```
> points(x=0.1,y=0.7)
> points(x=1,y=2)
```

and so on, to place the other points on the same plot. This can be done in one line, using `c()`, as follows:

```
> plot(x=c(0,0.1,1,1.2,2),y=c(1,0.7,2,2.3,4))
```

This has the additional advantage of choosing the plot area appropriately to your points.

If we'd like to add some lines to our plot, we can do so with the function `abline()`:

```
> abline(a=1,b=0.6)
```

will add the line with y -intercept 1 (the argument named `a`) and slope 0.6 (the argument named `b`). The names `a` and `b` instead of `b` and `m` are due to the fact that *R* was developed by statisticians, who write a linear equation as $y = a + bx$.

You can add several lines this way, and to make them easier to differentiate you may want them to have different colors. You can do this with the `col=` option, like

```
> abline(a=1,b=0.6,col="red")
```

for example.

3 Functions

R has lots of functions, but you can also define your own. For example, we may wish to calculate the y -coordinates of points on a line for several x -values and not have to type the same equation several times. We could define a function to do this and name it something, say f , like

```
> f <- function(x) 0.6*x + 1
>
```

Now, whenever we need the y -coordinate for certain x -values, we can just use $f()$ like

```
> f(c(0,1,2,3))
[1] 1 1.6 2.2 2.8
>
```

A lot of the functions already in *R* are actually written in *R* this way. You can see their definition just by typing their name (without parenthesis). Most of the basic functions are built-in for efficiency (they're the building blocks for the rest). One example of a useful function written in *R* is "outer", the so-called "outer product". It produces a multiplication table (where instead of multiplication you can have any function of two variables). Here is a division table:

```
> outer(X=c(1,2,3,4),Y=c(1,2,3,4),"/")
      [,1] [,2]      [,3] [,4]
[1,]    1  0.5 0.3333333 0.25
[2,]    2  1.0 0.6666667 0.50
[3,]    3  1.5 1.0000000 0.75
[4,]    4  2.0 1.3333333 1.00
```

4 Vectors

The `c()` function makes vectors.

5 Help()