

# Teach Yourself R

Chuck Anderson

Department of Computer Science  
Colorado State University

January 23, 2008

## Contents

<b>1 Starting and Stopping R</b>	<b>1</b>
<b>2 Introduction to R</b>	<b>3</b>
<b>3 Vectors and Matrices</b>	<b>4</b>
<b>4 Lists</b>	<b>10</b>
<b>5 Graphics</b>	<b>12</b>
<b>6 Using R Source Files</b>	<b>14</b>
<b>7 Reading in Data</b>	<b>15</b>
<b>8 Writing a Report in LaTeX with Figures from R</b>	<b>16</b>

---

## 1 Starting and Stopping R

---

How do I find information about R?

- Use a web browser to visit <http://www.r-project.org>.
- Use keywords “R” and “S-Plus” or “SPlus” with other topic-specific words in search engines like <http://www.google.com>.
- Search the R Mailing Lists Archive at <http://tolstoy.newcastle.edu.au/R/>.

How do I start the R environment on the CS department Unix systems?

The Unix prompt will be represented here by a dollar sign. The name of the R executable is just capital “R”, so the Unix command to start R is

\$ R

---

This is usually caused by not including in your PATH environmental variable the directory path to the location of the R executable. On the CSU Department of Computer Science network, you can find the path to R at <http://www.cs.colostate.edu> and following the Systems and Software links. On any Unix system you can find the path to R by

```
$ locate R
$ which R
$ find / -name "R" -print
```

R is supposedly installed on my Unix system, but when I run

```
$ R
```

I see

```
R: Command not found.
```

To add the R path to your PATH environmental variable, add a line like the following ones to the indicated file. If you use a Bourne style shell (sh, bash, etc.), add

```
#for R
export RDIR=/usr/local/R
export PATH=${RDIR}/bin:${PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${RDIR}/lib
export MANPATH=${MANPATH}:${RDIR}/man
```

to your .profile file in your home directory. If you use a C-style shell (csh, tcsh, etc.), add

```
#for R
setenv RDIR /usr/local/R
setenv PATH ${RDIR}/bin:${PATH}
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${RDIR}/lib
setenv MANPATH "${MANPATH}:${RDIR}/man"
```

to your .cshrc file.

---

What if R is not installed on my system?

See <http://www.r-project.org> for instructions on downloading and installing R on Linux, Solaris, MacOS, and MS Windows. Click on the CRAN (Comprehensive R Archive Network).

---

How do I exit from the R environment?

The R prompt is >. To exit R, evaluate the function `quit()`:

```
> quit()
```

which may be abbreviated to `q()`:

```
> q()
```

An often used alternative is to type control-d at the R prompt:

```
> <control-d>
```

You are then asked

```
Save workspace image? [y/n/c]:
```

to which you may reply `y` to save all variables that have been assigned values during this R session into a file named `.RData` in the current working directory. If you later start R while in this same directory, your variables and their values are automatically made available by loading the `.RData` file.

## 2 Introduction to R

---

What commands are available in R?	None.
-----------------------------------	-------

---

What?!!	<p>What you type at the R prompt is an expression. R attempts to evaluate the expression and type the result. This is why you exit R by</p> <pre>&gt; q()</pre> <p>q() is an expression that is evaluated by calling the function q() with no arguments.</p>
---------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

So what do I see when I type the following?	<p>The expression 42 is evaluated. The result is 42, so you will see</p> <pre>&gt; 42 [1] 42</pre>
> 42	<p>You will see the meaning of [1] as soon as we consider vector and matrix values.</p>

---

What is the value of $(100 * 2 - 12^2)/7 * 5 + 2$ ?	<pre>&gt; (100 * 2 - 12 ^ 2) / 7 * 5 + 2 [1] 42</pre>
-----------------------------------------------------	-------------------------------------------------------

---

What is the value of $\sin(\frac{\pi}{2})$ ?	<pre>&gt; sin(pi/2) [1] 1</pre> <p>Notice that pi is a predefined constant.</p>
----------------------------------------------	---------------------------------------------------------------------------------

---

How do I find out what other trigonometric functions are available?	<p>“How do I find out...” is the most important question for you to answer, and there are many answers. You can check the on-line documentation for particular functions,</p> <pre>&gt; ?sin</pre> <p>This shows a documentation page somewhat like standard Unix man pages. To end the viewing of the documentation and return to the R environment, just press “q”. You may also ask R to search for functions involving sin:</p> <pre>&gt; help.search("sin")</pre> <p>start a web browser on the on-line documentation,</p> <pre>&gt; help.start()</pre> <p>or search the web. For example, use <a href="http://www.google.com">http://www.google.com</a> to search using the words “R splus trigonometric functions”.</p>
---------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

How do I apply trigonometric functions to arguments in degrees?	<p>You can use the methods suggested above to try to find out. At some point, hopefully soon, you will realize that you are not finding any functions that do this for you. So you must do it yourself...duh.</p> <pre>&gt; sin(90 * pi/180) [1] 1</pre>
-----------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### 3 Vectors and Matrices

---

Can I work with vectors and matrices in R?	Of course! No data analysis tool is worth the bytes it burns if it doesn't. To create a matrix, use the <code>matrix()</code> function. Try typing <code>?matrix</code> .
How do I construct the matrix $\begin{pmatrix} 10 & 10 \\ 10 & 10 \\ 10 & 10 \end{pmatrix}$	<pre>&gt; matrix(10,3,2)       [,1] [,2] [1,]  10  10 [2,]  10  10 [3,]  10  10</pre> <p>Result is a matrix with 3 rows and 2 columns. The first argument provides the contents of the matrix. Values are repeated until matrix is full. Use the concatenate function <code>c()</code> to provide a list of numbers.</p>
What is the value of <code>matrix(c(1,2,3,4,5,6),3,2)</code> ?	<pre>&gt; matrix(c(1,2,3,4,5,6),3,2)       [,1] [,2] [1,]   1   4 [2,]   2   5 [3,]   3   6</pre> <p>The matrix is filled column by column.</p>
What is the value of <code>matrix(c(1,2,3),3,2)</code> ?	<pre>&gt; matrix(c(1,2,3),3,2)       [,1] [,2] [1,]   1   1 [2,]   2   2 [3,]   3   3</pre>
How do I learn about other <code>matrix()</code> arguments?	<p>Keep reading! But, as an R programmer showing tremendous potential, you decide to learn for yourself. Try <code>?matrix</code>. For a short summary of the arguments try <code>args(matrix)</code>.</p> <pre>&gt; args(matrix) function (data = NA, nrow = 1, ncol = 1, byrow = FALSE,           dimnames = NULL) NULL</pre> <p>The returned value is <code>NULL</code>. Every expression must return some value.</p>
How do I fill a matrix by row by row?	<pre>&gt; matrix(c(1,2,3,4,5,6),3,2,byrow=TRUE)       [,1] [,2] [1,]   1   2 [2,]   3   4 [3,]   5   6</pre> <p>The matrix is filled row by row. As you see, arguments can be assigned values by position or by name. You also see that boolean constants in R are <code>TRUE</code> and <code>FALSE</code>.</p>

---

---

Are there easy ways of specifying sequences, rather than typing `c(1,2,3,4,5,6)`?

Yep.

```
> c(1,2,3,4,5,6)
[1] 1 2 3 4 5 6
> 1:6
[1] 1 2 3 4 5 6
> seq(1,6,by=1)
[1] 1 2 3 4 5 6
> seq(1,6,by=2)
[1] 1 3 5
> seq(1,by=2,length=6)
[1] 1 3 5 7 9 11
```

Want to know more? Try `?seq`.

---

Okay, I get expressions. Now, how do I assign the result of an expression to a variable?

The assignment operator in R is `<-`. For example,

```
> a <- 4
> a
[1] 4
> a <- 4
> b <- 6
> c <- b * b + 2 * a - 2
> c
[1] 42
```

And watch this!

```
> 42 -> humm
> humm
[1] 42
```

Assignments can be made at the end of an expression, too. This is not used very often as it leads to unreadable code. It is probably allowed to let you decide at the last minute while typing a line to assign the result to a variable. This is not very useful with command line editing. For example, type control-a to move the cursor back to beginning of line and type in a variable name and `<-` then press return.

---

What is the value of the last expression here?

```
a <- matrix(2,3,3)
b <- matrix(1:9,3,3)
a * b
```

```
> a <- matrix(2,3,3)
> b <- matrix(1:9,3,3)
> a * b
      [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
```

The `*` operator does a component-wise multiplication. Use `%%` to do matrix multiplication.

---

What is `a %% b`?

```
> a %% b
      [,1] [,2] [,3]
[1,]   12   30   48
[2,]   12   30   48
[3,]   12   30   48
```

---

A matrix is transposed by the `t()` function. What is `a %*% t(b)`?

```
> a %*% t(b)
      [,1] [,2] [,3]
[1,]  24  30  36
[2,]  24  30  36
[3,]  24  30  36
```

---

Elements and sub-matrices are easily extracted. Given the previous assignment of `b`, what are the values of the following expressions?

```
> b[1,1]
[1] 1
> b[2,3]
[1] 8
> b[1,]
[1] 1 4 7
> b[,2]
[1] 4 5 6
> b[1:2,]
      [,1] [,2] [,3]
[1,]  1  4  7
[2,]  2  5  8
> b[,2:3]
      [,1] [,2]
[1,]  4  7
[2,]  5  8
[3,]  6  9
> b[,c(1,2)]
      [,1] [,2]
[1,]  1  4
[2,]  2  5
[3,]  3  6
> b[c(1,3),c(1,2)]
      [,1] [,2]
[1,]  1  4
[2,]  3  6
```

`b[1,1]`  
`b[2,3]`  
`b[1,]`  
`b[,2]`  
`b[1:2,]`  
`b[,2:3]`  
`b[,c(1,2)]`  
`b[c(1,3),c(1,2)]`

---

```
> b
      [,1] [,2] [,3]
[1,]  1  4  7
[2,]  2  5  8
[3,]  3  6  9
> b[-1,]
      [,1] [,2] [,3]
[1,]  2  5  8
[2,]  3  6  9
> b[c(-2,-3),]
      [,1] [,2] [,3]
[1,]  2  5  8
[2,]  3  6  9
> b[c(-2,-3),]
[1] 1 4 7
```

Negative integers as indices removes the specified portions of a matrix. What are

```
> b[-2,]
      [,1] [,2]
[1,]  1  7
[2,]  2  8
[3,]  3  9
> b[1:2,-3]
      [,1] [,2]
[1,]  1  4
[2,]  2  5
```

`b[-1,]`  
`b[c(-2,-3),]`  
`b[-2,]`  
`b[1:2,-3]`

---

```
> a %*% b[,1]
      [,1]
[1,]  12
[2,]  12
[3,]  12
```

What is `a %*% b[,1]`?

---

```
> a[1,] %*% b
      [,1] [,2] [,3]
[1,]  12  30  48
```

This and the previous answer are a little tricky. In the previous question, `b[,1]` was treated as a column vector. In this answer, `a[1,]` was treated as a row vector. However, if you evaluate each one, you see

```
> b[,1]
[1] 1 2 3
> a[1,]
[1] 2 2 2
```

They look like they are the same shape. The trickiness here is that R will change the shape of the arguments to the matrix multiplication operator to try to make the operation succeed.

---

Sometimes I don't want this.  
What is the value of `b[,1] %**% b`?  
b?

```
> b[,1] %**% b
      [,1] [,2] [,3]
[1,]   14   32   50
```

`b[,1]` was treated as a row vector here, even though it looks like `b[,1]` is a column vector.

---

This can be accomplished in two ways. You can tell R not to drop the single-valued dimension by using the `drop` argument.

```
> b[,1]
[1] 1 2 3
> b[,1,drop=FALSE]
      [,1]
[1,]    1
[2,]    2
[3,]    3
```

Now

But I really want `b[,1]` to be a column vector.

```
> b[,1,drop=FALSE] %**% b[,1]
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    8   14
[3,]    3   12   21
```

The second way of doing this is to replace the usual `%**%` operator by `%o%`, the outer product operator.

```
> b[,1] %o% b[,1]
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    8   14
[3,]    3   12   21
```

---

How do I calculate the inverse of a matrix?

Go ask <http://www.google.com>. Use keywords "R SPlus inverse matrix". You will soon discover that the function `solve()` is used for matrix inversion.

---

Using `runif()` to produce random values uniformly distributed between 0 and 1, I should get matrix `b` back from the last expression in

```
a <- matrix(runif(9),3,3)
b <- matrix(runif(9),3,3)
c <- a %**% b
bnew <- solve(a) %**% c
```

```
> a <- matrix(runif(9),3,3)
> b <- matrix(runif(9),3,3)
> c <- a %**% b
> bnew <- solve(a) %**% c
      [,1] [,2] [,3]
[1,] 0.8475927 0.1675809 0.27199269
[2,] 0.1293552 0.5486751 0.57194672
[3,] 0.4011202 0.5970193 0.07890513
> b
      [,1] [,2] [,3]
[1,] 0.8475927 0.1675809 0.27199269
[2,] 0.1293552 0.5486751 0.57194672
[3,] 0.4011202 0.5970193 0.07890513
```

---

How can I verify that the above answer is the same as `b`?

Many operators can be applied to matrices as well as scalars. Try the equality operator `==`.

---

What is `b == bnew`?

```
> b == bnew
      [,1] [,2] [,3]
[1,] FALSE FALSE FALSE
[2,] FALSE FALSE  TRUE
[3,] FALSE FALSE FALSE
```

To see if all elements are TRUE use the function `all`.

```
> all(b == bnew)
[1] FALSE
```

---

`b` and `bnew` are not equal. I'm sure they are equal within some tolerance. How can I check this?

Use `all.equal()`.

```
> all.equal(b,bnew)
[1] TRUE
```

Using `?all.equal` you will see that the optional argument `tolerance` can be set and that by default it is the precision of the machine. On my machine, I see

```
> .Machine$double.eps
[1] 2.220446e-16
```

---

I want to do this tolerance test myself.

Good for you. That's the spirit. You can use the `abs()` function applied to the difference matrix.

```
> all(abs(b-bnew) < 0.001)
[1] TRUE
```

---

How do I do the inner product between vectors `[1,3]` and `[2,4]`?

One way is

```
> matrix(c(1,3),1,2) %*% matrix(c(2,4),2,1)
      [,1]
[1,]   14
```

Or, you can rely on R doing the right thing and simply type

```
> c(1,3) %*% c(2,4)
Error in c(1, 3) %*% c(2,4) : non-conformable arguments
```

Whoops. Was I wrong? Look again. I mis-typed the `[2,4]` vector.

```
> c(1,3) %*% c(2,4)
      [,1]
[1,]   14
```

Which way is better? I prefer to carefully define all arguments as matrices of the correct shape. This allows R to catch any errors I have in my math logic.

---

Do this either as

```
> t(x) %*% y
      [,1] [,2]
[1,]    2    4
[2,]    6   12
```

or as

```
> drop(x %o% y)
      [,1] [,2]
[1,]    2    4
[2,]    6   12
```

Let

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

be two row vectors. How do I perform an outer product to produce matrix

$$\begin{bmatrix} 2 & 4 \\ 6 & 12 \end{bmatrix}?$$

The outer product results in a four-dimensional structure, because you are taking an outer product of two two-dimensional matrices. Drop the single-valued first and third dimensions using the function `drop()`. This can also be accomplished by calling the function `outer()`

```
> drop(outer(x,y))
      [,1] [,2]
[1,]    2    4
[2,]    6   12
```

The function `outer()` takes other arguments, including a function to apply to each pair of elements.

---

How would I use vector operations to calculate the root, mean, square error (RMSE) between vectors of actual and predicted values? Say the actual values are 1, 5, 10, and 20 and their predicted values are 1.1, 4.8, 10.5, and 18.6.

```
> actual <- c(1, 5, 10, 20)
> predicted <- c(1.1, 4.8, 10.5, 18.6)
> sqrt(mean( (actual-predicted)^2 ))
[1] 0.7516648
```

```
> calcRMSE <- function (a, b) {
+   sqrt( mean( (a - b)^2 ) )
+ }
> calcRMSE(actual,predicted)
[1] 0.7516648
```

I want to do this often. How do I define a new function to do this?

The first expression is continued on multiple lines, shown by the prompt changing to `+`. Here the variable `calcRMSE` is assigned the value that results from an expression of the form `function (a,b) { ... }`. This expression evaluates to a function object that is assigned to the variable `calcRMSE` making this variable the name of the new function.

Say I define the vectors  $ax = 0.1, 0.2, \dots, 0.5$  and  $ay = -1, -0.8, \dots, 1$  as the values along the  $x$  and  $y$  axis of a graph for which you want to calculate the values of the function  $f(x, y) = \frac{(x-y)}{(x+y)}$ . How do I build the matrix of values of  $f$ ?

```
> ax <- (1:5) * 0.1
> ay <- seq(-1,1,by=0.2)
> options(digits=3,width=40)
> f <- outer(ax, ay, function(x,y) { (x-y)/(x+y) })
> f
      [,1] [,2] [,3]      [,4]
[1,] -1.22 -1.29 -1.4 -1.67e+00
[2,] -1.50 -1.67 -2.0 -3.00e+00
[3,] -1.86 -2.20 -3.0 -7.00e+00
[4,] -2.33 -3.00 -5.0 7.21e+15
[5,] -3.00 -4.33 -11.0 9.00e+00
      [,5] [,6]      [,7]      [,8]
[1,] -3.00e+00 1 -3.33e-01 -6.00e-01
[2,] 7.21e+15 1 -4.16e-16 -3.33e-01
[3,] 5.00e+00 1 2.00e-01 -1.43e-01
[4,] 3.00e+00 1 3.33e-01 -1.39e-16
[5,] 2.33e+00 1 4.29e-01 1.11e-01
      [,9] [,10] [,11]
[1,] -0.714 -0.778 -0.818
[2,] -0.500 -0.600 -0.667
[3,] -0.333 -0.455 -0.538
[4,] -0.200 -0.333 -0.429
[5,] -0.091 -0.231 -0.333
```

This shows that anonymous functions can be defined and passed as arguments. The function `options` may be used to control how values are printed. See `?options`.

## 4 Lists

In C, a data structure containing items of different types is built as a `struct`. In C++ and Java, it is a `class`. In R, the functions `list` and `c` are used to build and concatenate lists. How do I make a list of the names of the first three months of the year and their number of days and assign it to `months`?

```
> months <- list("January",31,"February",28,"March",31)
```

How do I access "January", the pair "January" and 31, or all three month names?

```
> months[1]
> months[1:2]
> months[c(1,3,5)]
```

Now what if I want to change the number of days in February to 29?

A list access like `months[1]` actually returns a list of one item. To return the actual element, use `months[[1]]`. Therefore, to change one element, use

```
months[[4]] <- 29
```

---

How do I add the "April" and 30 to my list?

Either

```
> c(months,"April", 30)
or
> c(months,list("April", 30))
```

---

A common use of lists is for returning multiple values from a function. Suppose I have a vector of targets which are (1,0,0,1,0) and predictions of (0.6,0.2,0.5,0.8,0.2). How do I calculate the root mean square error value and assemble all three things in a list called `results`?

```
> targets <- c(1,0,0,1,0)
> predictions <- c(0.6,0.2,0.5,0.8,0.2)
> rmse <- sqrt(mean((targets-predictions)^2))
> rmse
[1] 0.3255764
> results <- list(targets,predictions,rmse)
```

---

I don't want to remember that RMSE is at index 3. Some or all elements of a list can be given names using `name = value` syntax. How do I do this for the variable `results`? The R function `names` results a vector of strings showing the names of the elements.

```
> results <- list("correct"=targets,
+ "predictions"=predictions,"RMSE"=rmse)
> results[[3]]
[1] 0.3255764
> results[["RMSE"]]
[1] 0.3255764
> names(results)
[1] "correct"      "predictions" "RMSE"
```

---

A list can be converted to a vector with `unlist`. I'll try this with `results`, but what will the type of the elements be? And if I do this for `months`, what will those elements be?

```
> unlist(results)
  correct1    correct2    correct3
    1.000     0.000     0.000
  correct4    correct5 predictions1
    1.000     0.000     0.600
predictions2 predictions3 predictions4
    0.200     0.500     0.800
predictions5      RMSE
    0.200     0.326
> unlist(months)
[1] "January" "31"      "February"
[4] "28"      "March"   "31"
```

The type of the elements of `unlist(results)` is numeric, while the elements of `unlist(months)` are of type character. The `mode` function will tell you this.

```
> mode(unlist(results))
[1] "numeric"
> mode(unlist(months))
[1] "character"
```

---

For practice, I want to define a function `sortedMonths` that takes a list of months and days as an argument, sorts the months by the number of days, and returns a list containing a vector of sorted month names and a vector of sorted days.

```
> months <- list("Jan",31,"Feb",28,"Mar",31,"Apr",30,
+ "May",31,"Jun",30,"Jul",31,"Aug",31,"Sep",30,"Oct",31,
+ "Nov",30,"Dec",31)
> sortedMonths <- function (months) {
+   monthNames <- unlist(months[seq(1,24,by=2)])
+   days <- unlist(months[seq(2,24,by=2)])
+   indexOrder <- order(days)
+   list(names=monthNames[indexOrder],days=days[indexOrder])
+ }
> m <- sortedMonths(months)
> names(m)
[1] "names" "days"
> m[["names"]]
[1] "Feb" "Apr" "Jun" "Sep" "Nov" "Jan"
[7] "Mar" "May" "Jul" "Aug" "Oct" "Dec"
> m[["days"]]
[1] 28 30 30 30 30 31 31 31 31 31 31 31
```

## 5 Graphics

---

Can data be graphed in R?

Definitely! R has a fairly complete graphics capability that can be used to produce gorgeous visualizations of data. Do

```
> demo(graphics)
```

in R to see sample displays of many of R's graphics functions.

---

Let me guess...R has a function named `plot`.

Of course. Take a look at its documentation:

```
> ?plot
```

A fun thing to do is to scroll down through a function's documentation to the examples near the end and run an example by using your mouse to select part of the example text and paste it into your R prompt. Then use `|control-p|`, `|control-n|`, `|control-f|`, `|control-b|` and `|backspace|` to repeat a previous command with modifications.

---

I want to start simply. How do I generate data from a parabola and graph it?

```
> x <- seq(-10,10,by=1)
> y <- x^2
> plot(x,y)
```

---

Try

```
> plot(x,y,type="lines")
```

or to see lines and points, do

```
> plot(x,y,type="both")
```

I only see points, but I want the curve.

Arguments and values can be abbreviated as long as they are not ambiguous with other arguments and values:

```
> plot(x,y,t="b")
```

but you would never do this in an R source file (see next section) because it is unreadable.

---

I'm sure I can specify an x and y axis label and a title. How do I do that? Before looking at the answer, I'll check out `?plot` in R.

Yep, you found it. Here is an example:

```
> plot(x,y,xlab="Cleverly Generated Sequence",
      ylab="Cleverly Squared", main="Top Title")
```

---

`rnorm` can be used to produce a number of values drawn from a Normal distribution. How can I use this to produce data from a noisy quadratic function? Let's use a Normal distribution with a variance of 10.

```
> ynoisy <- y + rnorm(length(y),sd=10)
```

---

Now I want to plot both the quadratic values and the noisy quadratic values on same plot, but a call to `plot` always erases the old plot and creates a new one. I've just been told that `lines` and `points` will add lines and points to an existing plot, so I will use those. First I'll check out `?lines` and `?points`.

```
> plot(x,y,type="b")
> points(x,ynoisyy,col="red",pch=4)
> lines(x,ynoisyy,col="green")
```

---

I can imagine cases where I have constructed a matrix where each column contains samples of a particular variable. For example, column 1 might be temperatures and column 2 might be pressures. How might I plot the columns on the same plot without calling `plot` then `lines` or `points`? What did you say? Try `matplot`?

```
> temps <- c(60, 65, 72, 66, 78, 74, 84, 83, 88, 91)
> pressures <- c(30.1, 30.5, 29.3, 29.5, 32.3, 31.2,
+ 30.8, 28.9, 28.5, 29.1)
> data <- cbind(temps,pressures)
> data
```

```
      temps pressures
[1,]    60      30.1
[2,]    65      30.5
[3,]    72      29.3
[4,]    66      29.5
[5,]    78      32.3
[6,]    74      31.2
[7,]    84      30.8
[8,]    83      28.9
[9,]    88      28.5
[10,]   91      29.1
```

```
# Notice the automatically named columns!
```

```
> matplot(data,type="b",pch=c("T","P"),
          xlab="Day",ylab="Measurements")
```

## 6 Using R Source Files

---

So far I have been typing directly into the R command line. I'd really like to save a sequence of commands in an R source file to be run anytime I choose. Given that the file is a "source" file, maybe I can figure out the name of the function to use to run the file.

Did you look up `?source` ?

---

Call the source file `plotTempPress.R`. The `system` function can be used to run any Unix command:

```
> system("more plotTempPress.R")
temps <- c(60, 65, 72, 66, 78, 74, 84, 83, 88, 91)
pressures <- c(30.1, 30.5, 29.3, 29.5, 32.3, 31.2, 30.8,
+ 28.9, 28.5, 29.1)
data <- cbind(temps,pressures)
matplot(data,type="b",pch=c("T","P"),
         xlab="Day",ylab="Measurements")
```

What will the source file look like that produces the temperature and pressure plot? How do I run it?

Run this with

```
> source("plotTempPress.R")
```

---

```
> system("more plotTempPress.R")
plotTempPress <- function () {
  temps <- c(60, 65, 72, 66, 78, 74, 84, 83, 88, 91)
  pressures <- c(30.1, 30.5, 29.3, 29.5, 32.3, 31.2,
                30.8, 28.9, 28.5, 29.1)
  data <- cbind(temps,pressures)
  matplot(data,type="b",pch=c("T","P"),
          xlab="Day",ylab="Measurements")
}
```

Usually an R source file first defines a set of functions to be used. The above commands can be encapsulated into a function named `plotTempPress` in a file. How?

Now,

```
> source("plotTempPress.R")
```

does not produce a plot, it just defines the function that can now be called

```
> plotTempPress()
```

## 7 Reading in Data

---

Say the temperature and pressure data already exists in a file, named `temppress.data`, that contains

```
60    30.1
65    30.5
72    29.3
66    29.5
78    32.3
74    31.2
84    30.8
83    28.9
88    28.5
91    29.1
```

```
> data <- read.table("temppress.data")
> data
  V1  V2
1 60 30.1
2 65 30.5
3 72 29.3
4 66 29.5
5 78 32.3
6 74 31.2
7 84 30.8
8 83 28.9
9 88 28.5
10 91 29.1
```

The function `read.table` can be used to read this data into R. How?

---

I like keeping those column labels with the data. To do so, I hope R can read in a data file that looks like

```
Temperature Pressure
60    30.1
65    30.5
72    29.3
66    29.5
78    32.3
74    31.2
84    30.8
83    28.9
88    28.5
91    29.1
```

```
> data <- read.table("temppress.data", head=TRUE)
> data
  Temperature Pressure
1          60      30.1
2          65      30.5
3          72      29.3
4          66      29.5
5          78      32.3
6          74      31.2
7          84      30.8
8          83      28.9
9          88      28.5
10         91      29.1
```

`read.table` can also be used for this file. The documentation shows how.

---

This looks like a matrix, but it is actually a data.frame, an R data structure that can contain columns of different types. Because it is a data.frame, it cannot be treated as a matrix:

```
> t(data) %*% data
Error in t(data) %*% data :
  requires numeric
  matrix/vector arguments
```

```
> data <- as.matrix(read.table("temppress.data",head=TRUE))
> t(data) %*% data
```

```
          Temperature Pressure
Temperature 58895.0 22815.30
Pressure    22815.3  9024.64
```

R includes functions for type conversion. How would I use `as.matrix` to convert the data.frame to a matrix?

## 8 Writing a Report in LaTeX with Figures from R

---

Type this minimal LaTeX code into a file named `simple.tex`

```
\documentclass{article}
\begin{document}
Hello.
\end{document}
```

and run

```
latex simple
```

then

```
xdvi simple.dvi
```

or convert to postscript and PDF by

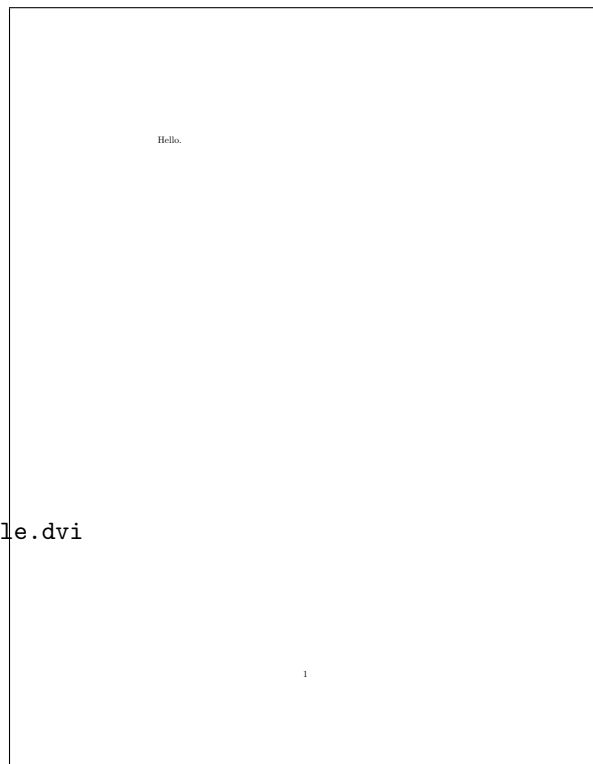
```
dvips -t letter -Ppdf -o simple.ps simple.dvi
ps2pdf simple.ps
```

and view results by

```
gv simple.ps
```

or

```
acroread simple.pdf
```



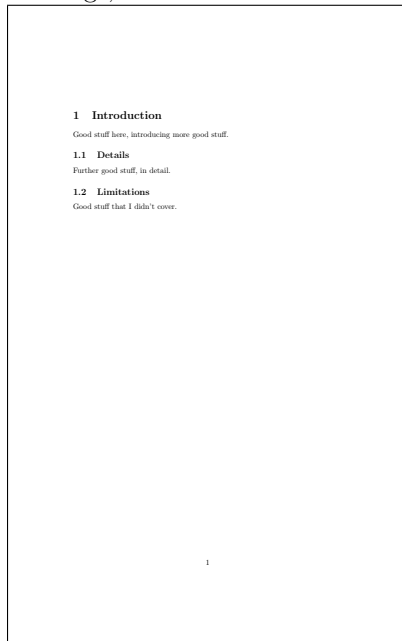
---

I've heard LaTeX is excellent at formatting math. I will type these lines right after the Hello. line

```
I am so smart, I do math like
\begin{displaymath}
RMSE = \sqrt{\frac{1}{n}
\sum_{i=1}^n
\left( t_i - y_i \right)^2}
\end{displaymath}
```

and then produce the formatted document again as I did before. What will I see?

If `\section{Section Name}` and `\subsection{SubSection Name}` create section and subsection headings, how would I make



Hello. I am so smart, I do math like

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (t_i - y_i)^2}$$

```
\documentclass{article}
\usepackage{geometry}
\geometry{letterpaper,textheight=9in,lmargin=1in,
rmargin=1in}

\begin{document}

\section{Introduction}
Good stuff here, introducing more good stuff.
\subsection{Details}
Further good stuff, in detail.
\subsection{Limitations}
Good stuff that I didn't cover.

\end{document}
```

---

To include graphic figures, we must tell LaTeX to use the `graphicx` package at the top of the file. While we are at it, let's also use the `geometry` package to tell LaTeX to use more of the paper than it will by default. Insert these lines just after the `\documentclass{article}` line.

You should see no change in the output.

```
\usepackage{graphicx}
\usepackage{geometry}
\geometry{letterpaper,textheight=9in,
  lmargin=1in,rmargin=1in}
```

and format the document again.  
What do you see?

---

Put these lines into the file named `Makefile`

I'm getting tired of typing all of those commands to format the document. How would I write a makefile that would process any of the unix commands

```
make simple.pdf
make simple.ps
make simple.dvi
```

```
simple.pdf: simple.ps
    ps2pdf simple.ps

simple.ps: simple.dvi
    dvips -t letter -Ppdf -o simple.ps simple.dvi

simple.dvi: simple.tex
    latex simple
    latex simple
```

Be careful. The indented lines all start with a single character, a tab. `make` requires this. `latex` is run twice to correctly produce figure references, the table of contents, and other things that require two passes.

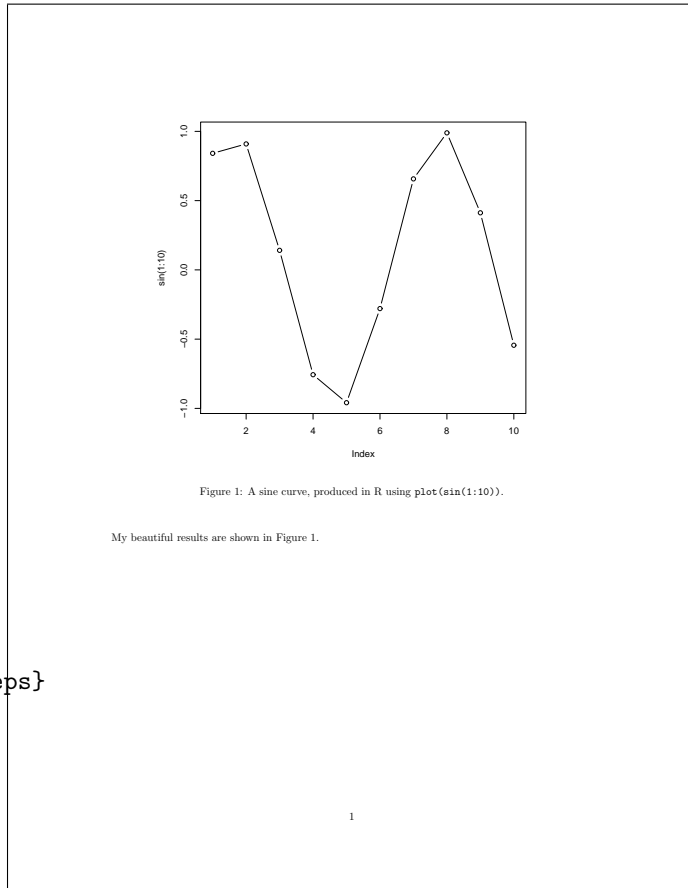
Now I want to try pasting a figure into my document. I've been told (just now) that this is what I must do to make a floating figure that includes a postscript file produced by R. I'll insert a blank line and the following lines after the `\end{displaymath}` line. A blank line starts a new paragraph.

```
\documentclass{article}
\usepackage{graphicx}
\usepackage{geometry}
\geometry{letterpaper,textheight=9in,
         lmargin=1in,rmargin=1in}
\begin{document}
```

My beautiful results are shown in Figure~\ref{fig:goodone}.

```
\begin{figure}
\begin{center}
\includegraphics[width=5in]{goodone.eps}
\caption{A sine curve, produced in R
         using \texttt{plot(sin(1:10))}.}
\label{fig:goodone}
\end{center}
\end{figure}
```

```
\end{document}
```



Watch this! I can use the same file `Makefile` to produce my figure files, if I have already written my R code in an R source file named `simplecode.R`. In `simplecode.R` I will define a function `drawSaveGoodOne()` that draws the `goodone` figure and saves it in a file named `goodone.eps`, since this is the name I use in `simple.tex`. R can be applied non-interactively to an R source file as explained in the man page for R that is shown by the unix command

`man R`

What will the new `Makefile` look like:

```
simple.pdf: simple.ps
         ps2pdf simple.ps

simple.ps: simple.dvi goodone.eps
         dvips -t letter -Ppdf -o simple.ps simple.dvi

simple.dvi: simple.tex goodone.eps
         latex simple
         latex simple

goodone.eps: simplecode.R
         R --quiet --no-save < simplecode.R > simplecode.output
```