

Introduction To The Statistical Language R

M. Hlynka
Dept. of Mathematics and Statistics
University of Windsor

September 28, 2008.

TABLE OF CONTENTS:

1. Introduction. (Obtaining a free copy of R).	Page 2.
2. History	Page 3.
3. Using R as a calculator.	Page 4.
4. Vectors.	Page 5.
5. Matrices.	Page 7.
6. Entering data from a file. Reading EXCEL files	Page 11.
7. Elementary statistics.	Page 13.
8. Plots.	Page 14.
9. Using R instead of statistical tables.	Page 17.
10. Simulation.	Page 19.
11. Regression.	Page 20.
12. Programming in R.	Page 26.
13. Tricks and Techniques	Page 29.
14. References	Page 31.

1. INTRODUCTION

Because of the current and spreading use of R, practicing statisticians should be fluent in R. It is also useful to probabilists, mathematicians, and lots of others. R is free.

The major web resource for R is <http://www.r-project.org/>

The download site is <http://www.cran.r-project.org> with mirrors in many countries.

For those using the Windows operating system, do the following to download.

Step 1. For Canadians, click on a Canadian mirror site, e.g. <http://probability.ca/cran/>

Step 2. Click on “Windows (95 and later)”

Step 3. Click on “base”

Step 4: Click on “R-2.4.1-win32.exe”

Step 5. Click on “Run” (this part is very slow)

Step 6. Accept all the defaults offered.

After all this, you should find an icon for R on your desktop.

After you download your own copy of R, you may want to download this manual and work through it. Or you may choose one of the many much better manuals available.

This manual gives only a small sample of the capabilities of R. To learn more, look at the manuals in the bibliography (most free and online) and use the (not so helpful) help facility in R.

2. HISTORY (FROM S TO R)

The statistical language S was developed at Bell Labs in the late 1970's mainly by Becker and Chambers and later (1984) by Wilks.

For a little history, see

<http://stat.bell-labs.com/S/history.html>

and

<http://cm.bell-labs.com/stat/doc/94.11.ps>

The name S refers to “statistics.”

A commercial version of S is marketed as S-PLUS by the company Insightful.

<http://www.insightful.com/>

S-PLUS is very popular among statisticians.

A public domain version (i.e. free) of S was developed under the name R. The name R was chosen because R is the letter in the alphabet next to S, and because it was developed by two people whose first names started with the letter R (Robert Gentleman and Ross Ihaka). OK. Good choice.

R and S-PLUS look almost the same but not quite. There are some command differences and the interface can be very different. Users have said that they behave differently internally and each has advantages. One BIG advantage of R is that it is free. So it can be used in elementary statistics courses without students complaining about the price. Also people in poor (but well educated) countries will choose it. Hence it is an international favorite. It takes some time to get used to R. It is worth the investment in time.

Before beginning section 3, please check the tip at section 13.1.

3. USING R AS A CALCULATOR (PLUS SOME HELP COMMANDS).

Enter R by clicking on the R icon that should be on your desktop. (My icon says R 2.4.1)

You should be presented with a prompt `>`.

Try the following and see what happens.

```
>2+3 [enter]          #(See – you have a calculator.)  
[1] 5
```

Try out $3*2$, 3^2 , $\log(3)$, $\exp(3)$ as well. ($\log(3)$ uses base e.)

```
> help(factorial)      #(This asks for information about factorials.)  
(You should get a screen of information. Close the screen after you find what you want.)
```

```
> ?factorial          #(This is shorter way of asking for information.)  
> example(factorial)  #(This will give some examples –perhaps useful, perhaps  
not. We do not discuss these here. )
```

```
>factorial(5)         #(We know  $5!=120$ .)  
[1] 120
```

```
> choose(8,3)         #( We can get binomial coefficients.)  
[1] 56
```

```
> pi                  #(pi is a built-in constant in R)  
[1] 3.141593
```

```
> sin(pi)            #(R has functions like sin, cos, etc. Here  $\sin(\pi)$  is zero, but  
[1] 1.224606e-16     doesn't look like it. Too bad.)
```

```
> round(sin(pi),4)   #(Round  $\sin(\pi)$  to four decimal places. Now it is 0.)  
[1] 0
```

Complex numbers are a bit strange. For example, one types “ $2+1i$ ” or “ $3-2i$ ” rather than the expected “ $2+i$.” or “ $3-2*i$.”

```
>sqrt(2-3i)          #(That is “q” with left and right brackets. Use this to quit.)  
[1] 1.674149-0.895977i
```

Writing several expressions on one line: Use semicolons. It saves space.

```
>a=2;b=7;c=5
```

We have assigned values to a,b,c.

```
>q()                  #(That is “q” with left and right brackets. Use this to quit.)
```

4. VECTORS

```
> x<-c(2,2,3,4)      #(This enters a list of numbers into a vector called x. The “c”
                      means concatenate.)
> x                  #(This asks for a list of the components of x.)
[1] 2 2 3 4

> x=c(2,2,3,4)      #(The equal sign gives an easier way to enter the vector x.)

> cumsum(x)         #(Find cumulative sums of vector x. Compare with sum(x).)
[1] 2 4 7 11
> prod(x)           #(Find product of all entries in vector x.)
[1] 48
> x%*%x             #(Find the dot product of x with itself, a 1x1 matrix.)
      [,1]
[1,] 33
> x*x              #(Componentwise multiplication of entries in x.)
[1] 4 4 9 16
> y                #(But there is no vector y. What happens?)
Error: object "y" not found.

> ls()             #(This asks for a list of all named items.)
[1] "x"

> rm(x)            # (“rm” means “remove.” This deletes the vector x.)
> x
Error: object "x" not found (See. x really is deleted.)

> x=c(2,2,3,4)     #(Start over.)

> x[1]             #(This asks for the first component of vector x.)
[1] 2

> x=2:7            $(This is a quick way of entering consecutive integers in x.)
> x
[1] 2 3 4 5 6 7
> x=seq(from=1,to=100,by=2)    #or x=seq(1,100,2)
                               (This enters numbers 1,3,5,...,99 in x. “seq” means “sequence.”)
```

R also has commands like `mean(x)`, `max(x)`, `min(x)`, or `rev(x)` to reverse the order of the vector, or `rep(x, 3)` which repeats the elements of `x` three times. For the vector `x=2:7` we can pick out components. For example `x[c(1,2)]` gives the first two components – namely 2,3. Also `x[-1]` will omit the first component of `x`.

Here are a few more examples and commands.

```

> x=c(2,4,7,5,4)
> x
[1] 2 4 7 5 4
> sort(x)
[1] 2 4 4 5 7
> sort(x,decreasing=TRUE)
[1] 7 5 4 4 2
> sort(x,d=T)           #Try to use abbreviations. They just might work.
[1] 7 5 4 4 2
> y=rev(x)             #This reverses the elements of x.
> y
[1] 4 5 7 4 2

```

Here are a couple of built in vectors in R.

```

> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

> LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"

> letters[1:10]       #(This asks for the first 10 letters.)
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

> q()                #(quit this session.)

```

5. MATRICES

```
> y=matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16),4,4)
#(This gives a 4 by 4 matrix with entries 1:16)
#(y=matrix(1:16,4,4) is faster.)
```

```
> y
  [,1] [,2] [,3] [,4]
[1,]  1  5  9 13
[2,]  2  6 10 14
[3,]  3  7 11 15
[4,]  4  8 12 16
```

We see that R always puts the entries in columns. That is unfortunate, but get used to it. We could use `> matrix(1:16,4,4,byrow=TRUE)` or `matrix(1:16,4,4,b=T)` to put entries in rows. Next, suppose we really want the transpose of `y`. That's easy.

```
> w=t(y)
> w
  [,1] [,2] [,3] [,4]
[1,]  1  2  3  4
[2,]  5  6  7  8
[3,]  9 10 11 12
[4,] 13 14 15 16
```

Now concatenate our matrix `y` with itself. Just for fun.

```
> z=matrix(c(y,y),4,8)
> z
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]  1  5  9 13  1  5  9 13
[2,]  2  6 10 14  2  6 10 14
[3,]  3  7 11 15  3  7 11 15
[4,]  4  8 12 16  4  8 12 16
>
```

We next find the fifth column of `z`. Then we find the second row.

```
> z[,5]
[1] 1 2 3 4
```

```
> z[2,]
[1] 2 6 10 14 2 6 10 14
```

To solve for `x` in the linear system $Ax=b$ (where `A` is a matrix and `b` is a vector or a matrix and `x` is a vector or a matrix), use `solve(A,b)`

If b is not specified, then it is taken to be an identity matrix so $\text{solve}(A)$ gives the inverse of A . We'll give an example a bit later.

Next we introduce matrix multiplication and identity matrices. Matrix y is on previous page.

```
y%*%y                                     #(This gives matrix multiplication of y with y.)
      [,1] [,2] [,3] [,4]
[1,]  90 202 314 426
[2,] 100 228 356 484
[3,] 110 254 398 542
[4,] 120 280 440 600
```

Note that $y*y$ gives component multiplication as shown below, not usually what we want.

```
> y*y
      [,1] [,2] [,3] [,4]
[1,]   1  25  81 169
[2,]   4  36 100 196
[3,]   9  49 121 225
[4,]  16  64 144 256
```

Suppose we want a 4 by 4 identity matrix that we call id . R uses a wrapping method so that if the matrix does not have enough entries, it starts using the entries over again. So we can trick R into giving us a 4 by 4 identity by only using 5 numbers instead of 16.

```
> id=matrix(c(1,0,0,0,0),4,4)
Warning message:
data length [5] is not a sub-multiple or multiple of the number of rows [4] in matrix
      (Don't worry about the warning)
```

```
> id
      [,1] [,2] [,3] [,4]
[1,]   1   0   0   0
[2,]   0   1   0   0
[3,]   0   0   1   0
[4,]   0   0   0   1
```

An easier way to get this identity matrix would be $id=\text{diag}(4)$. Next find the inverse of matrix $y+id$. R uses $\text{solve}()$ for this, as mentioned above. This is not a great name choice.

```
B=solve(y+id)
> B
      [,1]      [,2]      [,3]      [,4]
[1,] -0.2222222 -0.6  0.02222222  0.64444444
[2,] -0.6666667  0.6 -0.13333333  0.13333333
[3,] -0.1111111 -0.2  0.71111111 -0.37777778
[4,]  0.4444444  0.0 -0.44444444  0.11111111
```

Let's move to a new example.

Here is a 3 by 3 identity matrix where we label the rows and columns.

```
> id=matrix(c(1,0,0,0),3,3,dimnames=list(c("r1","r2","r3"),c("c1","c2","c3")))
Warning message: data length [4] is not a sub-multiple or multiple of the number of
rows [3] in matrix. (We ignore the warning.)
```

```
> id
  c1 c2 c3
r1 1 0 0
r2 0 1 0
r3 0 0 1
```

Other matrix commands to try are `det(A)`, `eigen(A)` for determinants, eigenvalues, eigenvectors.

Next we move to Markov chains. As an example, suppose we have a transition matrix P for a Markov chain. We want the limiting vector.

```
> P=matrix(c(.5,.3,.5,.7),2,2)
> P
```

```
  [,1] [,2]
[1,] 0.5 0.5
[2,] 0.3 0.7
```

```
> P2=P%*%P
> P2
```

```
  [,1] [,2]
[1,] 0.40 0.60
[2,] 0.36 0.64
```

```
> P4=P2%*%P2
> P4
```

```
  [,1] [,2]
[1,] 0.3760 0.6240
[2,] 0.3744 0.6256
```

```
> P8=P4%*%P4
> P8
```

```
  [,1] [,2]
[1,] 0.3750016 0.6249984
[2,] 0.3749990 0.6250010
```

```
> P16=P8%*%P8
> P16
```

```
  [,1] [,2]
[1,] 0.375 0.625
[2,] 0.375 0.625
```

That is far enough. The two rows are identical.

There is a theorem in applied probability that if P is a probability transition matrix, then every row of P^n (for large n) will be close to the limiting matrix and each row will give the limiting probability vector (.375, .625) of being in the two states.

Also from applied probability, we know that another way to get the limiting vector v (which is 1×2) for our 2 by 2 transition matrix P is to solve $v=vP$ together with the condition $v[1]+v[2]=1$.

Now $v=vP$ implies $vI=vP$ so $v(I-P)=0$ so $(I-P)^T v^T=0$. Here 0 and v^T are 2 by 1 vectors. Replace the first row of $(I-P)^T$ by $[1,1]$ and replace the first entry of the 0 vector by 1 in order to insert the condition $v[1]+v[2]=1$. We now have a system of type $Ax=b$. To solve for x , type `solve(A,b)`

```
> P=matrix(c(.5,.3,.5,.7),2,2)
> P
      [,1] [,2]
[1,] 0.5 0.5
[2,] 0.3 0.7
> id=matrix(c(1,0,0,1),2,2)
> A=t(id-P)
> A
      [,1] [,2]
[1,] 0.5 -0.3
[2,] -0.5 0.3
> A[1,1]=1
> A[1,2]=1
> b=c(1,0)
> solve(A,b)
[1] 0.375 0.625
```

There we go! We get the same answer as before.

6. ENTERING DATA FROM A FILE/ FROM EXCEL

I created a file in a directory on my f drive called Exams in a directory called course1. It was created with MSWord and saved as a text file (txt).

The file looks like

	Exam1	Exam2	Exam3
john	23	46	35
mary	42	31	36
sam	58	22	55
oksana	81	88	79
tom	11	19	18
peter	55	64	69
larisa	81	78	52

To read the file into R, I need to enter R and give the command

```
> x=read.table("f:course1/Exams.txt", header=TRUE)
```

I can also use

```
> x=read.table("f:/course1/Exams.txt", header=TRUE)
```

or

```
> x=read.table("f:\\course1\\Exams.txt", header=TRUE)
```

Apparently we need double slashes in one direction and single slashes in the other.

The “header=TRUE” part means that there is a header (Exam1 Exam2 Exam3) in the file so the first line should be treated as a header.

Then x will contain exactly the table of grades, including the names and Exam labels.

R treats the contents of the file like an object so R is called an object oriented language.

```
> x[,1] #(This asks for the first column.)  
[1] 23 42 58 81 11 55 81
```

```
> x[1,]  
      Exam1 Exam2 Exam3  
john    23    46    35
```

This asked for row 1, which is 23 46 35 (with a label).

If the original file is an EXCEL file, the simplest method for handling it is to save it as a txt file and read it as in the above example.

Rather than the read.table command, we can use the scan command if we are reading a vector rather than a table.

I created a txt file temp.txt on my f drive in the subdirectory “554” of the directory “courses”. The file has only one line of data, namely
13 26 49 52 67 88 89 56 64 68 22 47

I want to read this file in R. This is done with the command
`x=scan("f:\\courses\\554\\temp.txt")`

I can also write to files. I write a vector to a file using
`write(x,file="f:\\course1\\temp1.txt")`

This will create a file temp1.txt and write the vector to it. Be careful that you do not destroy an existing file. The command “write” overwrites rather than concatenates.

7. ELEMENTARY STATISTICS.

```
> x=seq(from=1,to=100,by=2)      #(This gives the vector 1,3,5,...,99.)
> mean(x)                        #(This finds the mean of the vector x.)
[1] 50
> sd(x)                          #(This gives sample standard deviation.)
[1] 29.15476
> var(x)                         #(This gives sample variance.)
[1] 850
```

Here is another example.

```
> grades=c(65,70, 82, 76,65,90, 71,73,22, 66,59, 75,73, 79, 2, 46, 62,68,71)
> summary(grades)                #(This gives max, min, quartiles, median, mean.)
Min. 1st Qu.  Median    Mean 3rd Qu.  Max.
 2.00  63.50   70.00   63.95  74.00  90.00
```

Let us do a goodness of fit test for the random number generator in R

```
> x=sample(0:9, 1000, replace=T)  #sample 1000 values with replacement from 0:9
> table(x)                       #creates a table of frequencies
x
 0  1  2  3  4  5  6  7  8  9
99 105 95 79 110 102 110 103 104 93
```

```
> a=c(99, 105, 95, 79, 110, 102, 110, 103, 104, 93)
> b=rep(.1,10)                  #Repeats .1 ten times for expected probabilities.
```

We want to test H_0 : good fit vs H_1 : bad fit.

```
> chisq.test(a,p=b)             #Performs a goodness of fit test.
```

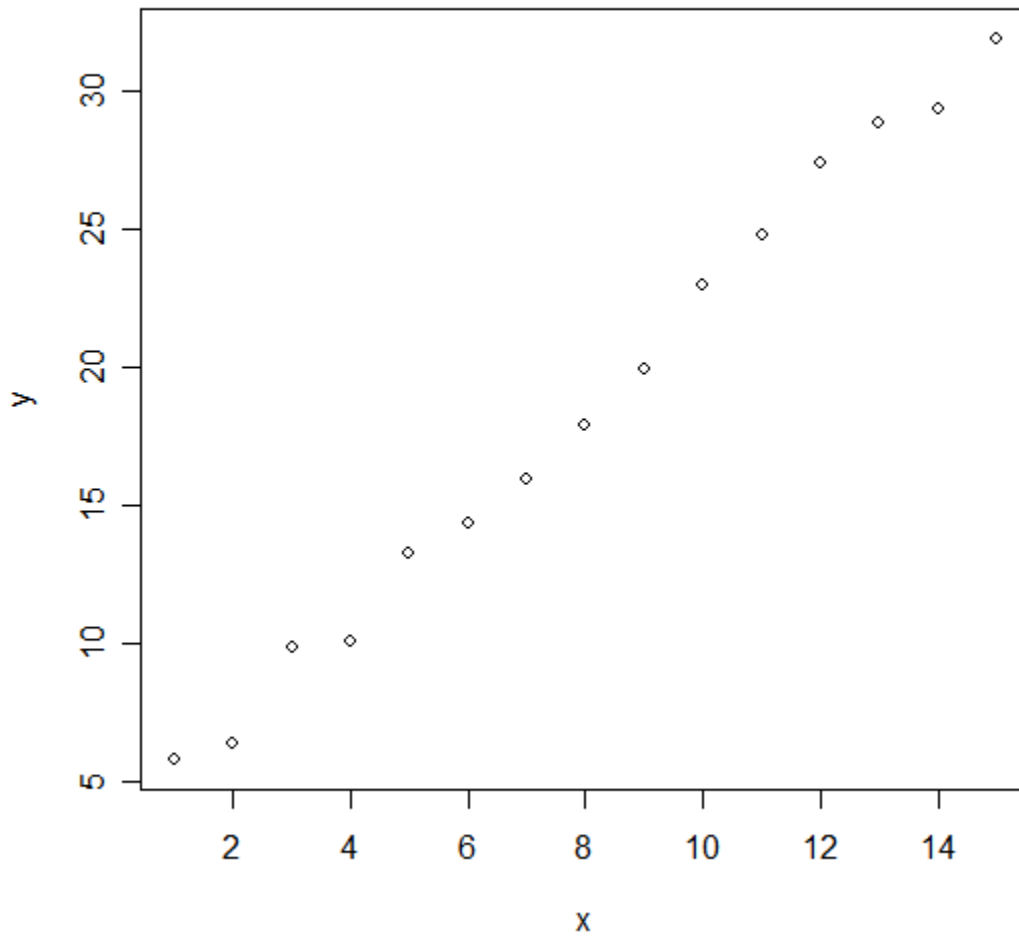
Chi-squared test for given probabilities

```
data: a
X-squared = 7.7, df = 9, p-value = 0.5646
```

The p-value is not less than .05 so there is insufficient evidence to conclude a bad fit.

8. PLOTS

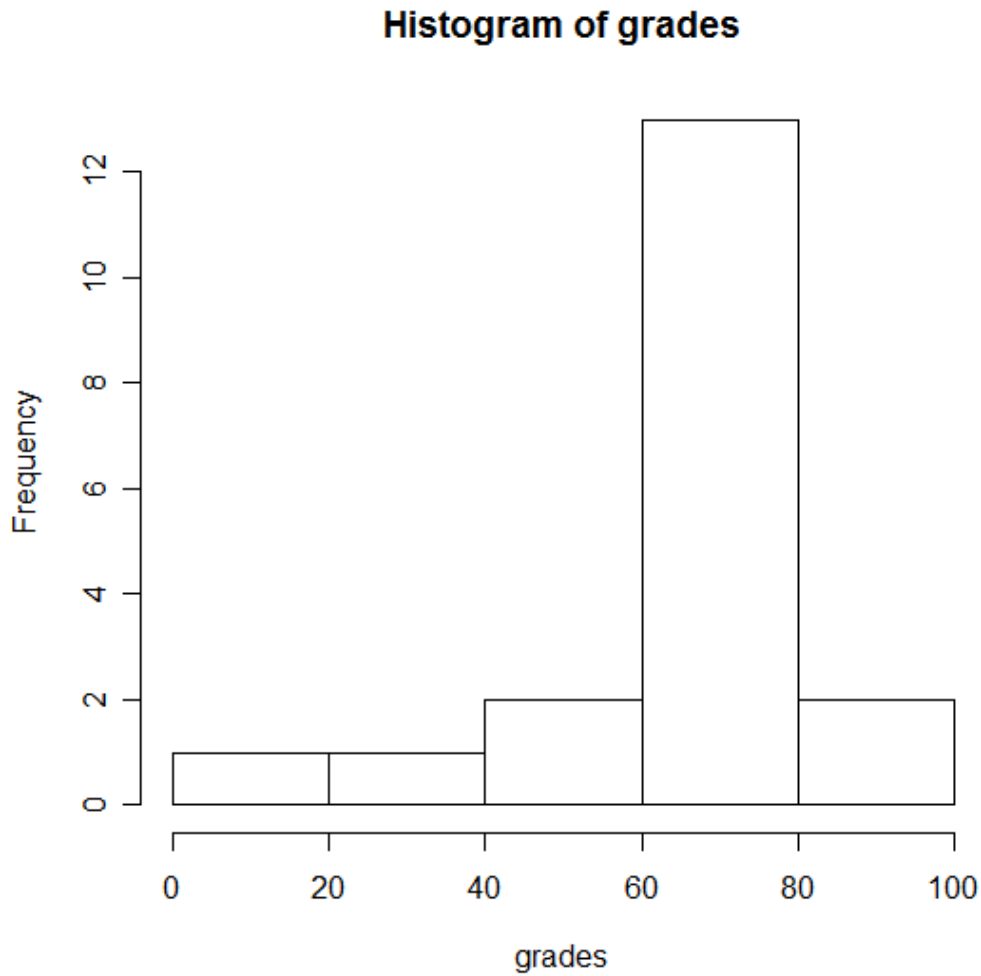
```
> x=1:15 # (This gives values 1,2,...,15.)  
> w=rnorm(x) # (This gives as many random Normal(0,1) values as were in x.)  
> y=3+2*x+w # (This plots y vs x, not x vs y. Nice looking plot appears in its own  
> plot(x,y) window. )
```



Let's try another example.

```
> grades=c(65,70, 82, 76,65,90, 71,73,22, 66,59, 75,73, 79, 2, 46, 62,68,71)
```

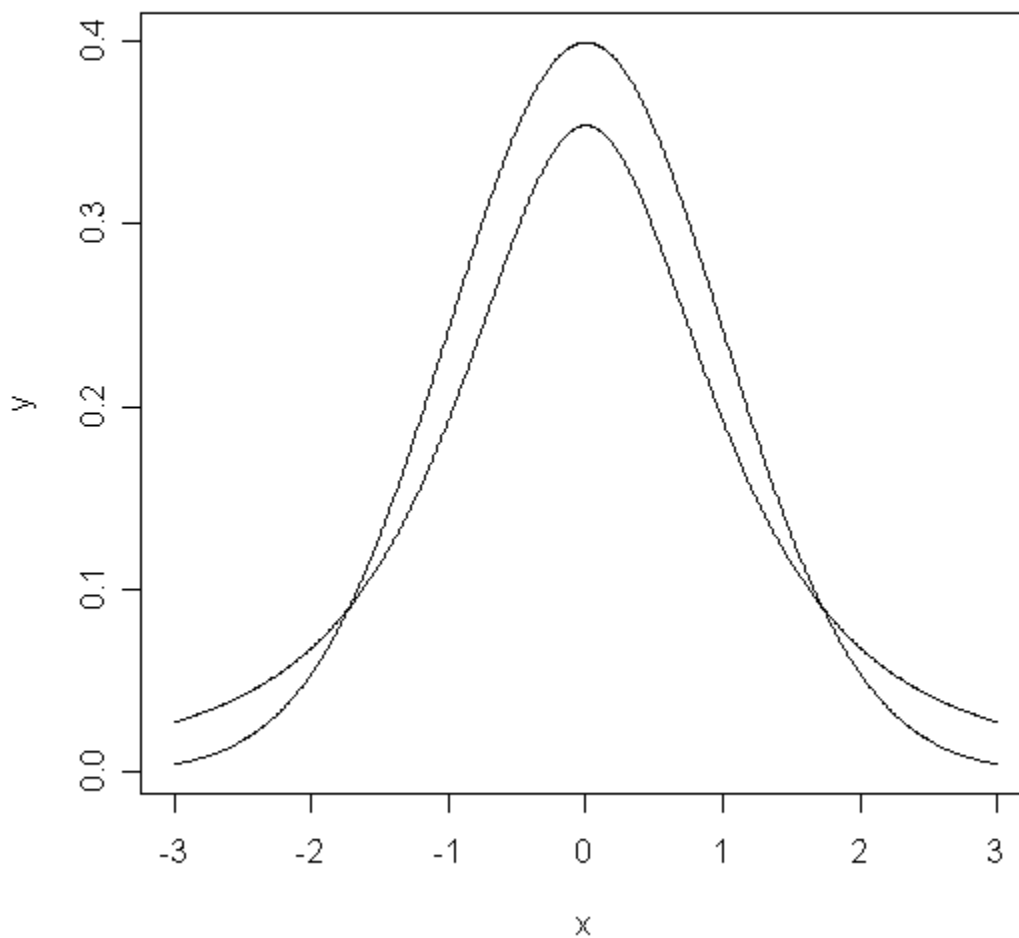
```
> hist(grades)       #(This gives a histogram of grades. This is a quick way to see how  
the class did. The scaling is automatic here, although we could  
specify bins.)
```



We can overlay plots quite easily. We illustrate by plotting the standard normal pdf and the t with 2 df on the same graph.

```
> x=seq(-3,3,.01)           #This gives values from -3 to 3 in steps of size .01
> y=dnorm(x)                #This gives values of the density of N(0,1).
> plot(x,y, type="l")       #Output is the graph of the pdf of N(0,1). "l" is a line plot.
> y=dt(x,2)                 #This gives values of the density of t with 2 d.f.
> lines(x,y)                #This gives overlay of the two plots.
```

The result is



9. USING R INSTEAD OF STATISTICAL TABLES.

```
> pbinom(0:15,15,.5)      #(This gives cdf F(0),...F(15) of a binomial (n=15, p=.5).)
[1] 3.051758e-05 4.882812e-04 3.692627e-03 1.757813e-02 5.923462e-02
[6] 1.508789e-01 3.036194e-01 5.000000e-01 6.963806e-01 8.491211e-01
[11] 9.407654e-01 9.824219e-01 9.963074e-01 9.995117e-01 9.999695e-01
[16] 1.000000e+00
```

This list does not look nice. Let's fix it up.

```
> x=pbinom(0:15,15,.5)
> y=round(x,4)          #(This rounds numbers to 4 decimal places.)
> matrix(y,16,1)
      [,1]
[1,] 0.0000
[2,] 0.0005
[3,] 0.0037
[4,] 0.0176
[5,] 0.0592
[6,] 0.1509
[7,] 0.3036
[8,] 0.5000
[9,] 0.6964
[10,] 0.8491
[11,] 0.9408
[12,] 0.9824
[13,] 0.9963
[14,] 0.9995
[15,] 1.0000
[16,] 1.0000
```

Let's try to make normal tables. The `pnorm` command gives cdf values of a normal(0,1). The `seq(from=0,to=3.99, by=.01)` part gives numbers 0.00, 0.01, ...,3.99. The resulting matrix is rounded to 4 decimal places and printed by rows with 10 entries per row. The result should look just like the normal tables in most texts, but without proper labels on the borders.

```
x=round(matrix(pnorm(seq(from=0,to=3.99,by=.01)),40,10,byrow=T),4)
> x
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
[1,]	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
[2,]	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
[3,]	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
[4,]	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
[5,]	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
[6,]	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
[7,]	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
[8,]	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
[9,]	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
[10,]	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
[11,]	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
[12,]	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
[13,]	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
[14,]	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
[15,]	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
[16,]	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
[17,]	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
[18,]	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
[19,]	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
[20,]	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
[21,]	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
[22,]	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
[23,]	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
[24,]	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
[25,]	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
[26,]	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
[27,]	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
[28,]	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
[29,]	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
[30,]	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
[31,]	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990
[32,]	0.9990	0.9991	0.9991	0.9991	0.9992	0.9992	0.9992	0.9992	0.9993	0.9993
[33,]	0.9993	0.9993	0.9994	0.9994	0.9994	0.9994	0.9994	0.9995	0.9995	0.9995
[34,]	0.9995	0.9995	0.9995	0.9996	0.9996	0.9996	0.9996	0.9996	0.9996	0.9997
[35,]	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9998
[36,]	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998
[37,]	0.9998	0.9998	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
[38,]	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
[39,]	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
[40,]	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000>

Check the standard normal table in an elementary statistics book for comparison.

10. SIMULATION

```
> rnorm(5)          #(This gives 5 simulated (random) normal random values with
                    mean 0 and variance 1.)
[1] 0.60273856 -0.05440138 0.17139366 -0.29927510 0.21562929

> rnorm(5,mean=2,sd=3)  #(This gives 5 random normals with mu=2, sigma=3.)
[1] 3.5270600 5.2531258 0.6363426 -1.9333987 5.5748965

> runif(5)           #(This gives 5 uniform (0,1) values.)
[1] 0.5028261 0.1145588 0.3254734 0.6854846 0.5357382

> rpois(5)           #(This was a guess for Poisson values. It did not work.)
Error in rpois(5) : argument "lambda" is missing, with no default

> ?rpois             #(This gave info about the rpois command. Try it. After reading
                    the information, we can enter the correct parameters.)

> rpois(5,3)         #(This gives 5 random Poisson values with parameter/mean 3.)
[1] 2 2 1 1 2

> rbino(8,3,.5)      #(Another bad guess, this time for binomial.)
Error: could not find function "rbino"

> ?binomial          #(This gives us something that we don't want. However, there is a
                    Table of contents on the left of the help window and we can
                    search for what we need. There it is! rbinom )

> ?rbinom            #(This should give us some help. Try it. Ah! Now we see how to
                    enter parameters.)

> rbinom(8,3,.5)     #(This gives 8 random values from binomial with n=3 and p=.5.)
[1] 2 3 2 0 2 3 1 1
```

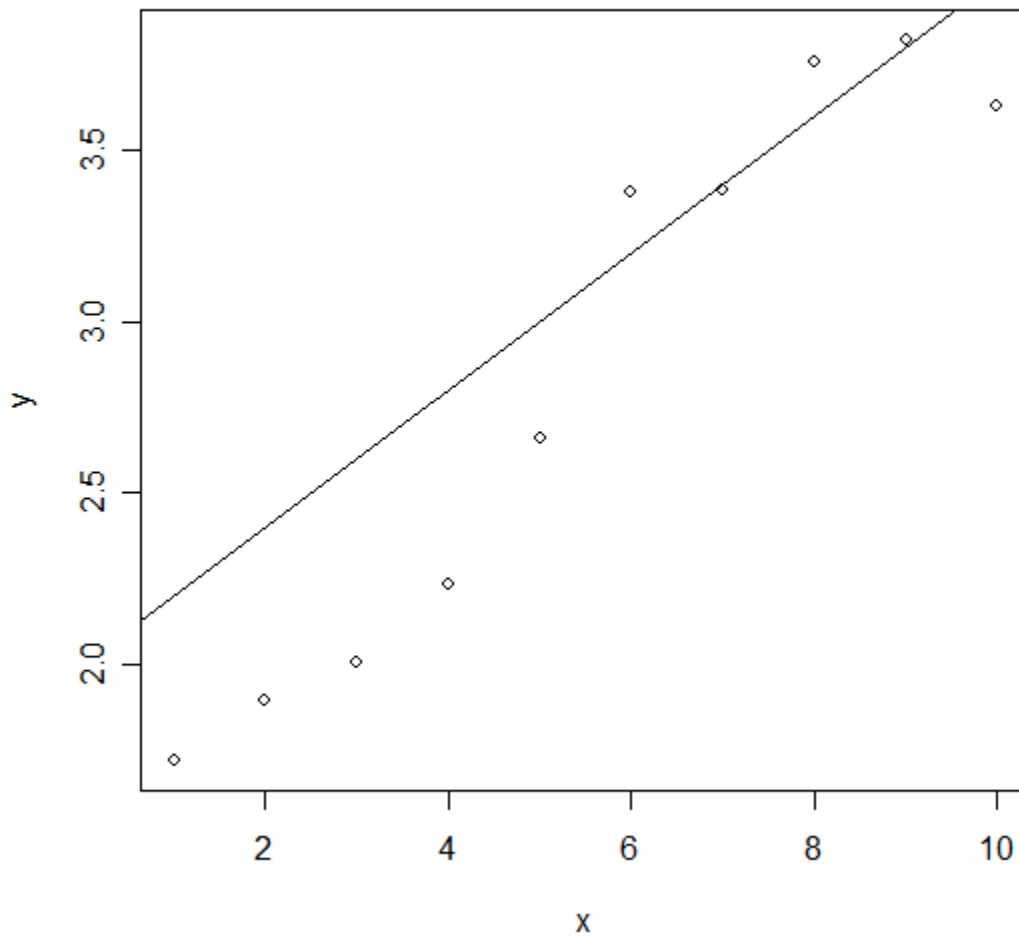
11. REGRESSION.

Before we start regression, let us just plot some values and add a straight line to the plot.

```
> x=1:10
> y=sqrt(x)+runif(x)
> y
[1] 1.718571 1.894796 2.006406 2.235122 2.658321 3.376392 3.383604 3.758569
[9] 3.821805 3.627096

> plot(x,y)           #(This plots 10 points. Do not close the graphics window.)

> abline(2,2)        #(This superimposes the line  $y=2+.2x$  onto the plot. Now close the
                    graphics window.)
```



Regression is performed with the `lm` command. (Here `lm` means “linear model.”)

```
> lm(y~x)
```

```
Call:
```

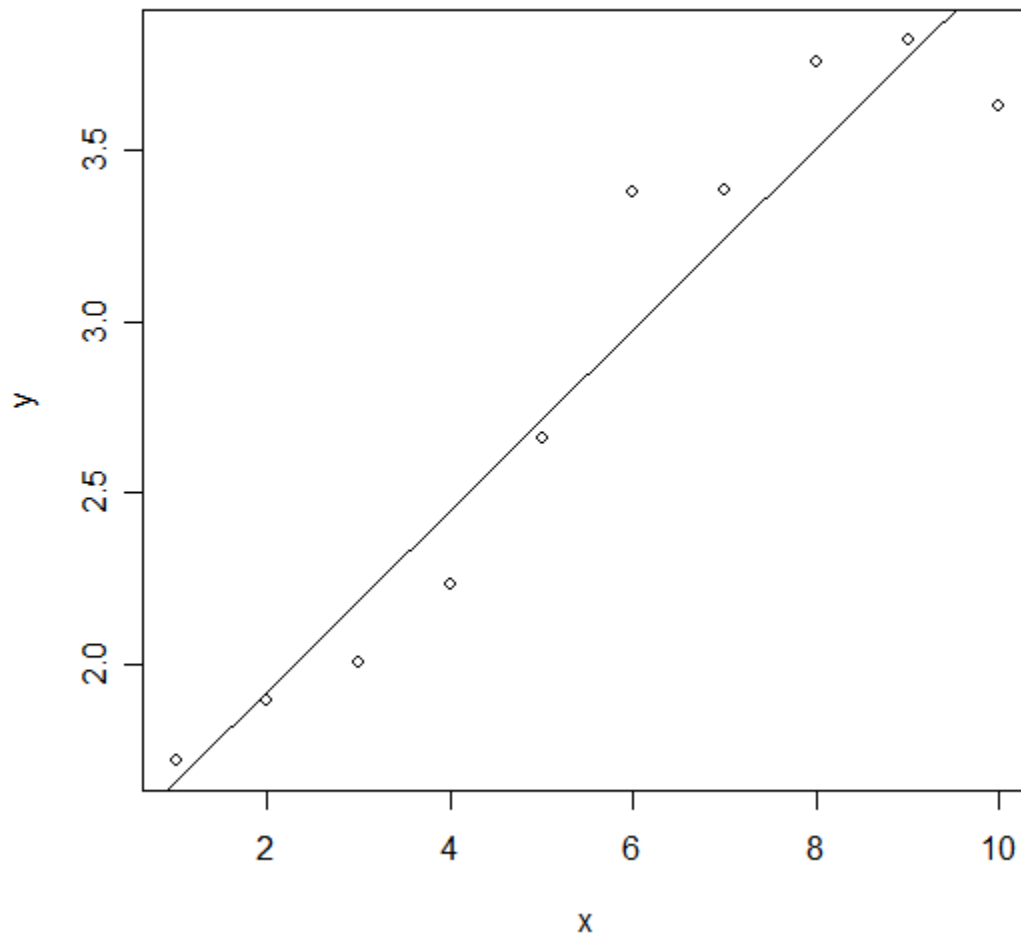
```
lm(formula = y ~ x)
```

```
Coefficients:
```

```
(Intercept)      x  
  1.3951      0.2642
```

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

```
> abline(1.3951,.2642)
```



Multiple regression is similar. As an example, we read an external file of grades consisting of 41 students with ID, Test1 grade, Test 2 grade and Final Exam grade. We want to perform regression of Final on Test1 and Test2. We will not make the complete data set available to the reader, so all that the reader can do in this case is to read the example.

```
> read.table("f:courses/231/Fall06/allgrades.txt", header=TRUE)
  id  t1 t2 final
1 102985 12 13  36
2 102393 27 37  43
3 102861 10 12  13
4 102306 30 30  48
5 102717 23 37  59
6 102607 36 38  53
...
40 101976 29 28  44
41 101731 17 13  17
> a=read.table("f:courses/231/Fall06/allgrades.txt", header=TRUE)
> x1=a[,2]
> x2=a[,3]
> y=a[,4]
> b=lm(y~ x1+x2)
> b
```

##(We skip the printout of many of the cases.)

##(This command does the regression.)

Call:
lm(formula = y ~ x1 + x2)

Coefficients:
(Intercept) x1 x2
-1.9417 0.8324 0.8223

```
> summary(b)
```

Call:
lm(formula = y ~ x1 + x2)

Residuals:
Min 1Q Median 3Q Max
-26.3292 -5.8990 -0.1541 7.9941 20.9592

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-1.9417	4.1595	-0.467	0.64329
x1	0.8324	0.2618	3.179	0.00294 **
x2	0.8223	0.2285	3.599	0.00091 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.06 on 38 degrees of freedom
Multiple R-Squared: 0.7073, Adjusted R-squared: 0.6919
F-statistic: 45.91 on 2 and 38 DF, p-value: 7.28e-11

```
> res=matrix(residuals(b), 41,1)
```

```
> res
```

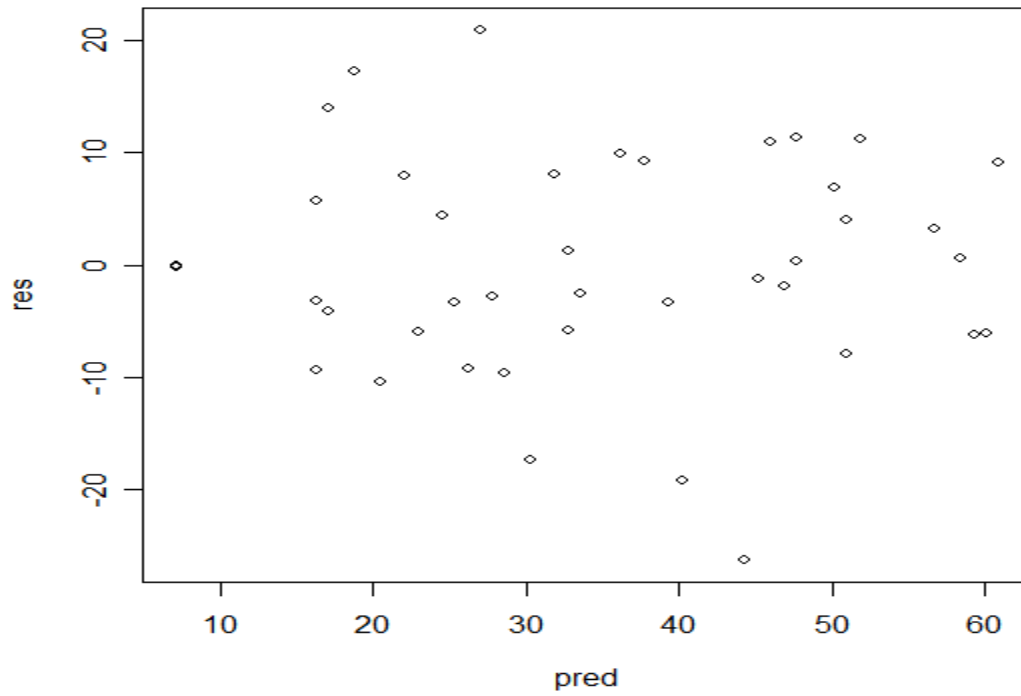
```
      [,1]  
[1,] 17.2630439  
[2,] -7.9580921  
[3,] -3.2498495  
[4,]  0.3007395  
[5,] 11.3715348  
[6,] -6.2720459  
...  
[41,] -5.8989898
```

```
> pred=matrix(predict(b),41,1)
```

```
> pred
```

```
      [,1]  
[1,] 18.736956  
[2,] 50.958092  
[3,] 16.249850  
[4,] 47.699260  
[5,] 47.628465  
[6,] 59.272046  
...  
[41,] 47.628465
```

```
>plot(pred,res)
```



This plots the residuals vs the predicted values. Everything looks nice and random about the line $res=0$, as it should.

We can use regression for things other than its intended purpose. For example, we know that the sum of the first n integers is $n(n+1)/2$. Suppose we have forgotten the formula for the sum of the squares of the first n integers. We do remember that the answer is a cubic in n . Let's find the formula using least squares regression.

```
> x=c(1,2,3,4,5,6,7,8,9)
> x2=x*x
> x3=x2*x
> y=cumsum(x2)
> y
[1] 1 5 14 30 55 91 140 204 285
> lm(y~x+x2+x3)
```

Call:
lm(formula = y ~ x + x2 + x3)

Coefficients:
(Intercept) x x2 x3
-0.08933 0.25849 0.47785 0.33469

So our formula apparently is
 $-0.08933 + .25849n + .47785n^2 + .33469n^3$.

But we now look up the formula in a math text and see it is $n(n+1)(2n+1)/6$. So our regression answer cannot be exact as it is showing a constant term that does not exist according to the true answer. This indicates a problem with accuracy. In a regression course, we learn the advantage of centering data to improve accuracy. Let's try.

```
> w=x-5
> w2=w*w
> w3=w*w*w
> lm(y~ w+w2+w3)
```

Call:
lm(formula = y ~ w + w2 + w3)

Coefficients:
(Intercept) w w2 w3
55.0000 30.1667 5.5000 0.3333

Now our formula seems to be
 $55 + (181/6)(n-5) + (11/2)(n-5)^2 + (1/3)(n-5)^3$
If we simplify this, it turns out to be exactly $n(n+1)(2n+1)/6$, which is correct. This example gives a good argument for centering data before using regression in statistical packages.

12. PROGRAMMING IN R.

EXAMPLE 12.1 (sum of the first n integers): R uses functions instead of subroutines. We choose not to use the sum command because we want to illustrate programming in R. R uses a semi colon as a separator between commands. We begin with the sum 'a' equal to zero and then keep adding i , for i from one to n . Then we print out the final sum.

```
> summ=function(n){a=0; for(i in 1:n) a=a+i; a}
> summ(10)
[1] 55
```

EXAMPLE 12.2 (trace of a matrix): Here is a program to find the matrix trace. Here A is an $n \times n$ matrix. If M is a square matrix, then $\text{diag}(M)$ is the vector of diagonal elements. If v is a vector of length n , then $\text{diag}(v)$ is a matrix with zeros off the diagonal and the elements of v on the diagonal. If n is an integer, then $\text{diag}(n)$ is the $n \times n$ identity matrix.

```
>trace=function(A){sum(diag(A))}
For example,
> M=matrix(c(3,5,3,7),2,2)
> trace(M)
10
```

EXAMPLE 12.3 (find the first n Fibonacci numbers): Define $F(1)=1$, $F(2)=1$, $F(n)=F(n-1)+F(n-2)$ for other values. The Fibonacci numbers are thus 1,1,2,3,5,8,13,21,... . We start with a vector $v=(1,1)$ and then keep adding new components to the vector v from the last two components of v , for n steps.

```
> fib= function(n){v=c(1,1); for(i in 3:n) {k=length(v);v=c(v,v[k]+v[k-1])}; v}
> fib(5)
[1] 1 1 2 3 5
```

EXAMPLE: Newton Raphson approximation applied to a theory of interest problem. We want to find n such that $90((1.0075^n)-1)/.0075=100n$.

The Newton Raphson method solves $f(x)=0$ by the iteration formula $x_{n+1}=x_n-f(x_n)/f'(x_n)$, where f' is the derivative of f . In our case $f(x)=(90(1.0075)^x)/.0075-100x$.

Our program is

```
>f=function(x){90*(1.0075 ^ x-1)/.0075-100*x}
>fp=function(x){90*1.0075^x*log(1.0075)/.0075-100}
>loop=function(n){x=20; for (i in 1:n){y=x-f(x)/fp(x);x=y};x}
>loop(1)\
35.98162\
>loop(5)\
28.68967\
> loop(100)\
28.68967\
```

EXAMPLE 12.4 (Generating a random Markov transition matrix)

Each row must sum to 1 and all entries must be nonnegative.

```
> PP=function(n)
+ {PP=matrix(runif(n^2),n,n); {for (i in (1:n))PP[i,]=PP[i,]/sum(PP[i,]);PP}
> PP(5)
      [,1] [,2] [,3] [,4] [,5]
[1,] 0.06101608 0.1472489 0.11200752 0.29199798 0.3877295
[2,] 0.24591234 0.0670017 0.25058688 0.17338313 0.2631160
[3,] 0.29430671 0.1207184 0.11930289 0.20366058 0.2620115
[4,] 0.22977504 0.3743099 0.07717427 0.02647456 0.2922662
[5,] 0.03572300 0.2337237 0.23654608 0.28479296 0.2092142
```

EXAMPLE 12.5 (find the nth power of a square matrix A):

```
> Power= function(A,n){B=id=diag(dim(A)[1]); for (i in 1:n) B=B%%A; B}
```

We apply this to a probability transition matrix.

```
> P=matrix(c(.5,.3,.5,.7),2,2)
> P
```

```
      [,1] [,2]
[1,] 0.5 0.5
[2,] 0.3 0.7
> P1=Power(P,1)
> P1
```

```
      [,1] [,2]
[1,] 0.5 0.5
[2,] 0.3 0.7
> P12=Power(P,12)
> P12
      [,1] [,2]
[1,] 0.375 0.625
[2,] 0.375 0.625
```

EXAMPLE 12.6 (choose grades above 3 from a vector of grades): This example uses the “if” command. We have a vector of grades. We only want to keep grades above 3.

```
> x=c(1,3,6,3,6,4,8,2,5,4,7,3,8,1,6)
> y=c(); n=length(x)
> for(i in 1:n)
+ if (x[i]>3) {y=c(y,x[i])}
> y
[1] 6 6 4 8 5 4 7 8 6
A shorter version is y=x[x>3].
```

R has similarities to the old computer language APL. APL was noted for its abilities to write a fairly complex program (function) in only one or two lines.

EXAMPLE 12.7 (simple linear regression – matrix approach) Below we write a program to find the coefficients of a simple linear regression of y on x for data in vectors y and x. A quick way to get a vector of 1's with the same length as the vector x is to use x^0 . We use the formula $b=(X^T X)^{-1} X^T Y$.

```
>regcoeff=function(x,y){;X=matrix(c(x^0,x),ncol=2);solve(t(X)%*%X)%*%t(X)%*%y}
> x=1:3; y=c(3,5,8)
> regcoeff(x,y)
      [,1]
[1,] 0.3333333
[2,] 2.5000000
```

See, the program part of this is on one line. The data are on the second line.

13. TRICKS

1. Suppose we make a mistake in R and get an error message. To avoid retyping the whole line, use the “up arrow” key on your keyboard to repeat the line and make the correction. Then hit Enter. We can use the “up arrow” key several times in succession to get earlier lines. Make any changes and hit Enter. We can use both the up arrow and down arrow keys.

2. To enter almost patterned matrices, enter the patterned part and then modify it. Suppose we want to enter a rate matrix M of the following type.

```
-1 1 0 0 0 0
2 -3 1 0 0 0
0 2 -3 1 0 0
0 0 2 -3 1 0
0 0 0 2 -3 1
0 0 0 0 2 -2
```

Remember the matrix wrap feature.

```
> v=c(-3,2,0,0,0,1)
```

```
> M=matrix(v,6,6)
```

Check to see what M looks like. Then we fix up the components that are incorrect.

```
>M[1,1]=-1; M[6,6]=-2
```

Try it out.

```
> M
```

```
  [,1] [,2] [,3] [,4] [,5] [,6]
[1,] -1  1  0  0  0  0
[2,]  2 -3  1  0  0  0
[3,]  0  2 -3  1  0  0
[4,]  0  0  2 -3  1  0
[5,]  0  0  0  2 -3  1
[6,]  0  0  0  0  2 -2
```

3. Suppose our workspace is cluttered. We want to get rid of everything.

```
> ls()
```

```
[1] "w" "w2" "w3" "x" "x2" "x3" "y" # See. Lots of clutter.
```

```
> rm(list=ls())      #(This will do the job.)
```

```
> ls()
```

```
character(0)        #(See. All done.)
```

A better method is to go to the Misc button on your toolbar. Click Misc. Then click “Remove all objects”, then Yes.

4. R allows abbreviations sometimes. Since they may not be listed anywhere, try to guess what might work. For example, enter a matrix using “byrow=TRUE.” Then try
> A=matrix(c(2,3,7,9),2,2,b=T) #(This gives a 2x2 matrix with rows (2,3), (7,9).)

5. Are you getting tired of entering matrix data by columns? Cure the problem for your whole session by defining a ‘Matrix’ function, different from the ‘matrix’ function.

```
> Matrix=function(v,m,n){matrix(v,m,n,byrow=TRUE)}  
> Matrix(1:4,2,2)  
  [,1] [,2]  
[1,]  1  2  
[2,]  3  4
```

Now that’s better.

6. Edit a function. Suppose we already have a function, say Power. To edit it, type
> edit(Power)

This puts us in edit mode, but the mode starts with the word ‘function’ rather than “Power=”. Make any changes you want. Then insert the “Power=” at the beginning while still in edit mode. Use the exit icon in the upper right corner. You will be asked if you want to save your changes. Say Yes. If you try this without inserting “Power =”, the corrections may not be saved. I don’t know why.

7. Edit a function. I like to write my functions outside of R, especially if they are long. I usually write them in MS Word and then copy the whole function into R and try it out. If it works, I am happy. If I get an error, I go back to Word to fix it and try again.

REFERENCES.

1. An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics. Version 2.4.0 (October 10, 2006). By W. N. Venables, D. M. Smith and the R Development Core Team. Taken from <http://cran.r-project.org/manuals.html>
2. simpleR – Using R for Introductory Statistics. By John Verzani. <http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf>
Version 0.4. August 22, 2002 Very nice.
3. A Quick, Painless Tutorial on the R Statistical Package. By Norm Matloff. 2005. <http://heather.cs.ucdavis.edu/~matloff/r.html>
Definitely useful.
4. A simple tutorial of the R language for statistical computing and graphics, by Eric Wajnberg. http://www.popecol.uni-bremen.de/download/BEPAR/r_tutorial_e.wajnberg.pdf