

R WORKSHOP

UC Davis
Alan R. Lemmon

OVERVIEW

Session I (October 27, 2008)		Time (min)	Page
1.00	RESOURCES FOR LEARNING R	5	2
1.01	CHANGING THE WORKING DIRECTORY	5	2
1.02	USING R AS A CALCULATOR	5	3
1.03	EVALUATING LOGICAL EXPRESSIONS	5	3
1.04	DEFINING OBJECTS AND MANIPULATING THEIR VALUES	10	4
1.05	WRITING FUNCTIONS...A PREVIEW	10	4
1.06	A FEW PRE-DEFINED FUNCTIONS	10	5
1.07	R IS VERY GOOD AT HANDLING VECTORS	10	5
1.08	CHANGING THE VALUES IN A VECTOR	5	6
1.09	USING R AS A CALCULATOR WITH VECTORS	10	6
1.10	MULTIDIMENSIONAL ARRAYS	10	7
1.11	LOADING COMMANDS/FUNCTIONS INTO R	5	8
1.12	SAVING OUTPUT TO A FILE	5	9
1.13	SAVING/LOADING MATRICES	5	9
1.14	SAVING/LOADING OBJECTS	5	10
1.15	LOADING DATA INTO R	10	10
 Session II (October 29, 2008)			
2.01	LOADING A TABLE INTO R	20	11
2.02	LOADING A MATRIX INTO R	15	12
2.03	STATISTICAL DISTRIBUTIONS AT YOUR FINGERTIPS	15	13
2.04	SUMMARIZING DATA	15	13
2.05	VISUALIZING DATA	15	14
2.06	PLOTTING X vs. Y	10	15
2.07	ADDING ADDITIONAL POINTS/LINES TO THE CURRENT GRAPH	10	15
2.08	POINT SHAPES AND SIZES	10	15
2.09	LINE WIDTHS AND TYPES	10	16
2.10	COLORS	20	16
 Session III (October 31, 2008)			
3.01	IF AND ELSE	10	18
3.02	FOR LOOPS	10	19
3.03	MORE FUN WITH FUNCTIONS	15	20
3.04	SORTING VECTORS AND DATA FRAMES	20	21
3.05	RANDOMIZATION TESTS (WORKED OUT)	20	22
3.06	RANDOMIZATION TESTS (YOU TRY)	30	23
3.07	PACKAGES AND DEMOS	0	23
3.08	ANSWERS TO RANDOMIZATION TEST PROBLEMS	10	24

1.00 RESOURCES FOR LEARNING R (5)

Online Resources: <http://cran.r-project.org/other-docs.html>

Examples:

Statistics Using R with Biological Examples: http://cran.r-project.org/doc/contrib/Seefeld_StatsRBio.pdf

Practical Regression and Anova using R: <http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf>

R Reference Card: <http://cran.r-project.org/doc/contrib/Short-refcard.pdf>

Matrix Algebra: <http://personality-project.org/r/sem.appendix.1.pdf>

Books (66+): <http://www.r-project.org/doc/bib/R-books.html>

Examples:

The R Book Michael Crawley **2007**, Wiley

R Graphics Paul Murrell, Chapman & Hall/CRC **2005**

Computational Genome Analysis An Introduction Deonier, Richard C., et al. Springer, **2007**

Bioinformatics and Computational Biology Solutions Using R... Gentleman, R et al. Springer, **2005**

Analysis of Phylogenetics and Evolution with R Paradis, Emmanuel Springer, **2006**

Bayesian Computation with R Jim Albert. Springer, **2007**

The Statistics of Gene Mapping David Siegmund and Benjamin Yakir. Springer, **2007**

Bioinformatics with R. Robert Gentleman. Chapman & Hall/CRC, **2008**

1.01 CHANGING THE WORKING DIRECTORY (5)

The working directory is the folder on your computer from which R will load files and to which R will save files.

To change the working directory for the current session only:

Go to File > Change Dir (different on a mac).

To change the working directory for all future sessions:

Right click on Rgui.exe shortcut, then change "Start in" path.

1.02 USING R AS A CALCULATOR (5)

The following **operators** are used to perform mathematical operations on two or more values or objects:

Example:	Answer:	Function:
11 + 5	16	Add
11 - 5	6	Subtract
11 * 5	55	Multiply
11 / 5	2.2	Divide
11 ^ 5	161051	Power
11 %% 5	1	Remainder
11 %/% 5	2	Whole value
11*(5+2)%%2	11	<i>Don't forget to use parentheses</i>
(11*(5+2))%%2	1	

NOTE: If R is waiting for you to finish a command, a "+" will appear at the beginning of the line instead of a ">".

1.03 EVALUATING LOGICAL EXPRESSIONS (5)

Example:	Answer:	Function:
3 > 4	FALSE	Greater than
3 == 4	FALSE	Equal to (note the double equals sign)
3 < 4	TRUE	Less than
3 >= 4	FALSE	Greater than or equal to
3 <= 4	TRUE	Less than or equal to
3 > 4 & 5 > 2	FALSE	AND condition
3 > 4 && 5 > 2	FALSE	AND condition (note the double '&' symbol)
3 > 4 5 > 2	TRUE	OR condition
3 > 4 5 > 2	TRUE	OR condition (note the double ' ' symbol)
!(3 > 4)	TRUE	NOT operator
3 != 4	TRUE	NOT operator

NOTE: Pressing the up key retrieves previous commands

1.04 DEFINING OBJECTS AND MANIPULATING THEIR VALUES (10)

`x = 5` *Create an object named "x" and assign to it the value 5*
`x` *Report the value of the variable named "x"*
`y = x + 1` *Terms on the right side can be values or objects*

An object stores one or more values...each value has a mode (type):

Object	Value	Mode
<code>mydec =</code>	<code>3.14</code>	Numeric
<code>myint =</code>	<code>897</code>	Numeric
<code>mytxt =</code>	<code>"word"</code>	Character
<code>mylgl =</code>	<code>TRUE</code>	Logical
<code>mylgl2=</code>	<code>T</code>	Logical

`mode(mytxt)` *Check the mode of an object*

Rules for naming objects:

- Object names must start with a letter or "."
- Object names are case sensitive
- Object names may contain only letters, numbers, underscores, or "."

1.05 WRITING FUNCTIONS...A PREVIEW (10)

A function is an object that does something for you.

`add = function(a,b){ a + b }` *Create a function named "add" and assign to it a specific task*
`add(a=x,b=y)` *Use the function by giving it the required arguments.*
`add(a=5,b=11)`
`add(5,11)`
`add3=function(a,b=3){ a + b }` *Arguments can have default values.*
`add3(5)`
`add3(5,4)` *You can override the default value by providing a new value.*
`add` *You can look at the function definition.*

1.06 A FEW PRE-DEFINED FUNCTIONS (10)

<code>dir()</code>	<i>Report a list of the files in the working directory</i>
<code>directory=getwd()</code>	<i>Report the path of the working directory</i>
<code>setwd("C:/")</code>	<i>Set the working directory</i>
<code>setwd(directory)</code>	<i>Return the working directory to the path stored in the object named 'directory'</i>
<code>objects()</code>	<i>Report the list of variables currently in memory (same as <code>ls()</code>)</i>
<code>rm(x)</code>	<i>Remove the object named "x" from memory</i>
<code>help()</code>	<i>Pull up the help documentation</i>
<code>help(dir)</code>	<i>Pull up the help documentation for the topic "dir"</i>
<code>?dir</code>	<i>Another way to pull up the documentation for a particular topic.</i>
<code>help.search("colors")</code>	<i>Search the help documentation for the word "colors"</i>
<code>help.search(colors)</code>	<i>This doesn't work because there is no object named "colors" in memory</i>

1.07 R IS VERY GOOD AT HANDLING VECTORS (10)

<code>x</code>	<i>x is a vector with one element, 11 is the first element of the vector x</i>
<code>[1] 11</code>	<i>Pay no attention to the [1] (for now)</i>

There are many ways to create a vector...

<code>x = c(1,2)</code>	<i>Concatenate the values 1 and 2 and assign the resultant vector to the object named "x"</i>
<code>mixed = c(1,2.5,"word")</code>	<i>Note that all elements in a vector are converted to the same mode.</i>
<code>x = rep(0,10)</code>	<i>Make a vector by the repeating quantity 0 ten times</i>
<code>x = 1:10</code>	<i>Make a vector out of the integers between 1 and 10</i>
<code>x = 10:1</code>	<i>Make a vector out of the integers between 1 and 10 (in reverse order)</i>
<code>x = rep(x,10)</code>	<i>Make a vector by repeating the vector x ten times</i>
<code>x = rep(x,each=5)</code>	<i>Make a vector by repeating each element of x ten times</i>
<code>x = append(x,999)</code>	<i>Append the value 999 to the end of the vector named 'x'. Don't use <code>append()</code> many times...</i>
<code>seq(from=1,to=20,by=2.6)</code>	<i>Generate a sequence with a specified interval between numbers</i>

1.08 CHANGING THE VALUES IN A VECTOR (5)

<code>x[4] = -2</code>	<i>Reference a single element in the vector</i>
<code>x[30] = 54</code>	<i>If you reference an element that does not yet exist, the vector will be lengthened</i>
<code>x[11:20]=-1</code>	<i>Reference a range within the vector</i>
<code>x[c(1,4,7)]=-7</code>	<i>Reference specific elements of the vector</i>
<code>length(x)</code>	<i>Check the length of the vector</i>
<code>length(x)=5</code>	<i>Truncate the vector</i>
<code>length(x)=10</code>	<i>Extend the vector (note that NA, which represents a missing value, is the default value).</i>
<code>x=1:10</code>	
<code>x[length(x):1]=x</code>	<i>Reverse the order of the vector</i>
<code>x=x[length(x):1]</code>	<i>Another way to reverse the order of the vector</i>
<code>x = rev(x)</code>	<i>An easier way to do the same thing</i>

1.09 USING R AS A CALCULATOR WITH VECTORS (10)

Now you will begin to see the power of R...

Operators can be applied to vectors.

<code>x = 1:10</code>	
<code>x + 5</code>	<i>Add 5 to every element of x</i>
<code>add3(x)</code>	<i>Pass the x vector into our function</i>
<code>x<5</code>	<i>Evaluate a logical expression for every element of the vector</i>
<code>x[x%%2==1]</code>	<i>Pull out all of the odd values in x</i>
<code>x[x<5]=0</code>	<i>Give a new value to all elements of the vector with values less than 5</i>
<code>x+c(1:10)</code>	<i>Add the corresponding elements of two vectors</i>
<code>x+c(1:10)*c(1:10)</code>	<i>Perform mathematical operations on any number of vectors</i>
<code>x*c(1:5)</code>	<i>Perform mathematical operations on any number of vectors</i>
<code>x*c(1:4)</code>	<i>The vector lengths must be multiples of each other</i>
<code>x[-3]</code>	<i>Remove an element from the vector (reduces vector length)</i>
<code>x[-5:-3]</code>	<i>Remove a section from the vector (here, the 3rd, 4th and 5th elements are removed)</i>

1.10 MULTIDIMENSIONAL ARRAYS (10)

Ready to add another dimension of complexity?...

<code>x = matrix(1:25)</code>	<i>The default is to create a matrix with one column</i>
<code>x = matrix(1:25,nrow=5)</code>	<i>You can specify the number of rows and columns</i>
<code>x = matrix(1:25,nrow=5,byrow=TRUE)</code>	<i>You can fill across instead of down</i>
<code>x = matrix(nrow=5,ncol=5)</code>	<i>You can start with an empty matrix</i>

<code>dim(x)</code>	<i>Check the dimension of the matrix</i>
<code>nrow(x)</code>	<i>Check the number of rows</i>
<code>ncol(x)</code>	<i>Check the number of columns</i>

Ways to change values in a matrix...

<code>x = matrix(1:25,nrow=5)</code>	
<code>x[x>=13]=-1</code>	<i>Change all elements with a value that meets a certain condition</i>
<code>x[2,3]=999</code>	<i>Change the value in the second row, third column</i>
<code>x[2:4,c(1,5)]=0</code>	<i>Change the values that exist both in rows 2-4 and in columns 1 or 5</i>
<code>x[,3]</code>	<i>Reference a particular column from the matrix</i>
<code>x[2,]</code>	<i>Reference a particular row from the matrix</i>
<code>x=t(x)</code>	<i>Transpose the matrix</i>
<code>x=fix(x)</code>	<i>Manually alter values in the matrix</i>

Or choose Edit > data editor...

<code>x = array(1:25,dim=c(5,5))</code>	<i>You can also create a matrix (2-D array) with the array() function</i>
<code>x = array(1:125,dim=c(5,5,5))</code>	<i>You can create an array with any number of dimensions</i>
<code>x[1,1,3]</code>	<i>Access the value in the first row, first column, and third layer</i>
<code>x[, ,1]</code>	<i>Access the entire first layer</i>

BREAK (15)

1.11 LOADING COMMANDS/FUNCTIONS INTO R (5)

You don't have to type everything out at the command line!

Choose File > New Script...

Open a new script window

Write the next four lines in the script window:

```
z=c(1,2,3,4,5)
exponent=function(x,a){
  x^a
}
```

Choose File > Save...

Save the script as "ExampleScript1.r"

```
> z
```

Why doesn't this work?

```
Error: object "z" not found
```

```
source("ExampleScript1.r")
```

The source() command executes each line of a script as a command

The script can be created in any text editor

```
ls()
```

Now the objects "z" and "exponent" appear

```
z
```

```
exponent(2,3)
```

You can use the function now that it is loaded into memory as an object

```
exponent(z,2)
```

Try right clicking on the top line of the script...

1.12 SAVING OUTPUT TO A FILE (5)

```
sink("ExampleSink1.txt")
```

Initiate a sink file and redirect all console output to it...

```
exponent(z,1)
```

```
exponent(z,2)
```

```
exponent(z,3)
```

```
sink()
```

Stop redirecting the output...

```
sink("ExampleSink1.txt",split=T)
```

Write the output to a sink file and to the console

Choose File > Display file(s)...

View the output that was saved in a text file...

1.13 SAVING/LOADING MATRICES (5)

```
x=matrix(1:80,ncol=10)
```

```
write(x,"ExampleMatrix1.txt")
```

Choose File > Display file(s)...

Write a matrix to a file

Oops! We forgot to specify the number of columns

```
write(x,"ExampleMatrix1.txt", ncol=10)
```

Choose File > Display file(s)...

Write a matrix to a file, specifying the number of columns

Something is still not right...

```
write(t(x),"ExampleMatrix1.txt",ncol=10)
```

Choose File > Display file(s)...

Write a matrix to a file after transposing it

Now that is better...

```
y=scan("ExampleMatrix1.txt")
```

```
y=matrix(y,ncol=10)
```

Use the scan function to read all values from a file into R

You can then convert the vector of values into a matrix

1.14 SAVING/LOADING OBJECTS (5)

```
x=matrix(1:80,ncol=10)
save(x,file="xSaved.r")
```

Write the object named “x” to a file

Misc > Remove all objects...
`ls()`

Clear the memory of all objects
Verify that all objects have been removed

Choose File > Display file(s)...
`load("xSaved.r")`

Try to view the object in the file...
Read the object named “x” from a file

1.15 LOADING DATA INTO R (10)

Choose File > New Script...

Open a new script window

Type the following table into the script window (use tabs to delimit):

	C1	C2	C3
R1	1	2	3
R2	4	5	6
R3	7	8	9

Choose File > Save...

Save the table as ExampleTable1.txt

```
x=read.table("ExampleTable1.txt")
```

Read the table and store it in the variable “x”

NOTE: *read.csv()* and *read.delim()* can be used to load comma and tab delimited tables

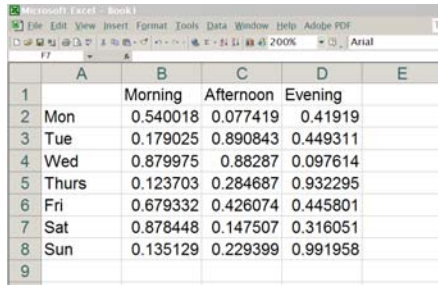
Referencing a particular column from a table...

```
x$C3
x[,3]
x[,3]=c("a","b","c")
mode(x[,2])
mode(x[,3])
```

Reference the column with the name “C3”
Another way to reference the same column
Each column can have its own mode

2.01 LOADING A TABLE INTO R (20)

Suppose that you have entered some data into a table in Excel that **includes row and column labels**:



A screenshot of a Microsoft Excel spreadsheet. The table has 5 columns labeled A through E and 9 rows. Row 1 contains column labels: 'Morning' in B1, 'Afternoon' in C1, and 'Evening' in D1. Rows 2 through 8 contain data for days of the week: 'Mon', 'Tue', 'Wed', 'Thurs', 'Fri', 'Sat', and 'Sun' in column A, with corresponding numerical values in columns B, C, and D. Row 9 is empty.

	A	B	C	D	E
1		Morning	Afternoon	Evening	
2	Mon	0.540018	0.077419	0.41919	
3	Tue	0.179025	0.890843	0.449311	
4	Wed	0.879975	0.88287	0.097614	
5	Thurs	0.123703	0.284687	0.932295	
6	Fri	0.679332	0.426074	0.445801	
7	Sat	0.878448	0.147507	0.316051	
8	Sun	0.135129	0.229399	0.991958	
9					

The simplest way to get the table into R is the following:

1) In Excel, choose File > Save as > Text (tab delimited)(.txt)

2) Use the following command in R to load the table:

```
x=read.delim("Data1.txt",header=T,row.names=1)
```

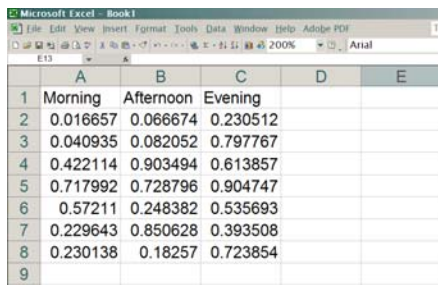
Note: "Data1.txt" is the name of the file saved by Excel

Try the following alternatives to see how things can go wrong:

```
x=read.delim("Data1.txt")
```

R thinks that there is no row labels

Suppose that you have entered some data into a table in Excel that includes **only column labels**:



A screenshot of a Microsoft Excel spreadsheet. The table has 5 columns labeled A through E and 9 rows. Row 1 contains column labels: 'Morning' in A1, 'Afternoon' in B1, and 'Evening' in C1. Rows 2 through 8 contain numerical data in columns A, B, and C. Row 9 is empty.

	A	B	C	D	E
1	Morning	Afternoon	Evening		
2	0.016657	0.066674	0.230512		
3	0.040935	0.082052	0.797767		
4	0.422114	0.903494	0.613857		
5	0.717992	0.728796	0.904747		
6	0.57211	0.248382	0.535693		
7	0.229643	0.850628	0.393508		
8	0.230138	0.18257	0.723854		
9					

The simplest way to get the table into R is the following:

1) In Excel, choose File > Save as > Text (tab delimited)(.txt)

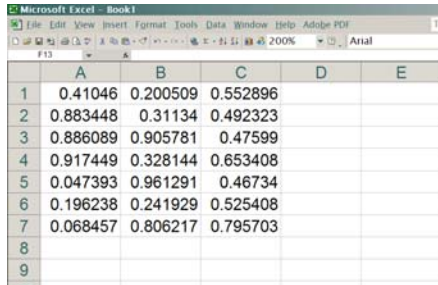
2) Use the following command in R to load the table:

```
x=read.delim("Data2.txt",header=T)
```

Note: "Data2.txt" is the name of the file saved by Excel

2.02 LOADING A MATRIX INTO R (15)

Suppose that you have entered some data into a table in Excel that includes **no labels**:



	A	B	C	D	E
1	0.41046	0.200509	0.552896		
2	0.883448	0.31134	0.492323		
3	0.886089	0.905781	0.47599		
4	0.917449	0.328144	0.653408		
5	0.047393	0.961291	0.46734		
6	0.196238	0.241929	0.525408		
7	0.068457	0.806217	0.795703		
8					
9					

The simplest way to get the matrix into R is the following:

- 1) In Excel, choose File > Save as > Text (tab delimited)(.txt)
- 2) Use the following commands in R to load the matrix:

```
x=scan("Data3.txt")  
x=matrix(x,ncol=3,byrow=T)
```

Note: "Data3.txt" is the name of the file saved by Excel

Note: You must set byrow=T because the scan reads one row at a time

See what happens if you forget to set the byrow argument:

```
x=scan("Data3.txt")  
x=matrix(x,ncol=3)
```

The difference between a table and a matrix:

A matrix is an object containing only numeric values

A table is a more complex object that contains:

One vector for each column of data (each vector can be of any mode)

One vector for the row labels

Other attributes

NOTE: For more detailed information on data input/output in R, go to Help > Manuals (in PDF) > R Data Import/Export

2.03 STATISTICAL DISTRIBUTIONS AT YOUR FINGERTIPS (15)

There are other ways to get data...make them up!

```
x=rnorm(n=100)
x=runif(n=100)
x=rgamma(n=100, shape=5)
x=rbeta(n=100, shape1=1, shape2=2)
x=rpois(n=100, lambda=5)
x=rchisq(n=100, df=3)
x=rbinom(n=100, size=10, prob=0.5)
x=dnorm(0)
x=pnorm(0)
x=qnorm(c(0.025, 0.975))
```

*Draw random numbers from a **normal** distribution*

*Draw random numbers from a **uniform** distribution*

*Draw random numbers from a **gamma** distribution (also used for **exponential**)*

*Draw random numbers from a **beta** distribution*

*Draw random numbers from a **Poisson** distribution*

*Draw random numbers from a **chi-square** distribution*

*Draw random numbers from a **binomial** distribution*

*Compute the **density** of the normal distribution at zero*

*Compute the **area under the normal curve** to the left of zero*

*Compute **quantiles** of the normal distribution*

2.04 SUMMARIZING DATA (15)

```
x=rnorm(n=10000)
mean(x)
sd(x)
var(x)
quantile(x)
range(x)
sum(x)
prod(x)
```

*Take the **mean** of the values in a vector or array*

*Take the **standard deviation** of the values in a vector or array*

*Take the **variance** deviation of the values in a vector or array*

*Find the **quantiles** for the values in a vector or array*

*Find the **range** of the values in a vector or array*

*Compute the **sum** of the values in a vector or array*

*Compute the **product** of the values in a vector or array*

Some other useful mathematical functions/constants (see help documentation for related functions):

```
pi
exp(1)
log(10)
sqrt(10)
abs(10)
factorial(5)
cos(5)
```

*Obtain the constant **pi***

*Obtain the constant **e** or evaluate e^x*

*Obtain the **natural log** of 10*

*Obtain the **square root** of 10*

*Obtain the constant **pi***

*Evaluate $5! = 5*4*3*2*1$*

*Evaluate the **cosine** of 5*

2.05 VISUALIZING DATA (15)

Now we finally get to some eye candy!

```
hist(x)
```

*View a **histogram** of the values in a vector*

```
hist(rnorm(1000),plot=F)  
hist(rnorm(1000))$breaks
```

You can access the raw data used to draw the histogram

```
hist(rnorm(n=10000))  
hist(rnorm(n=10000),nclass=100)  
hist(rnorm(n=10000),nclass=100,prob=T)
```

You can specify the number of bins

```
x11()
```

Open a new graphics window (use `quartz()` on Mac)

```
plot(x)
```

Plot the values

```
plot(rnorm(n=10000))  
x11()  
plot(rnorm(n=10000),xlim=c(1,100))  
x11()  
plot(rnorm(n=10000),ylim=c(-10,10))
```

Modify the scale along the x axis

Modify the scale along the y axis

Plots can be highly customized...

```
par(bg="black",fg="white",col.lab="white",col.axis="white")  
plot(rnorm(1000),col=sample(rainbow(1000),1000),pch=sample(1:20,1000,replace=T),cex=sample(1:20,1000,replace=T)/10)  
par(bg="white",fg="black",col.lab="black",col.axis="black")
```

2.06 PLOTTING X vs. Y (10)

```
x=rnorm(n=1000)
plot(x,sqrt(x))
plot(x,sqrt(x),type="l")
plot(-5:5,dnorm(-5:5),type="l")
plot((-500:500)/100,dnorm((-500:500)/100),type="l")
```

Plot Y against X
Plot values using lines instead of points (try “b”, and “h”)
This is useful for looking at a density function

NOTE: *x and y must be of the same length*

2.07 ADDING ADDITIONAL POINTS/LINES TO THE CURRENT GRAPH (10)

```
plot(rnorm(1000,mean=-5),ylim=c(-10,10))
points(rnorm(1000,mean=5))
lines(rnorm(1000,mean=0))

x=rnorm(10000)
hist(x,nclass=100,prob=T)
lines(density(x))
```

Add additional points to the current graph
Add an additional line to the current graph

You can add lines to a histogram as well

2.08 POINT SHAPES AND SIZES (10)

Points can take many shapes and sizes (don't type all of this stuff in)...

```
plot(NULL,NULL,xlim=c(0,3),ylim=c(0,30))
points(rep(1,25),25:1,pch=1:25,bg="gray")
text(rep(1,25),25:1,pos=4)
points(rep(2,25),25:1,cex=(1:25)/10)
text(rep(2,25),25:1,pos=4,labels=c(1:25)/10)
text(1,27,labels="pch")
text(2,27,labels="cex")
plot(x=1:100, y=rnorm(100,(1:100)/10),cex=1, pch=rep(1:25,each=4), bg="red")
```

pch defines the point type
Add text to the plot
cex defines the point size

NOTE: *to “permanently” set graphical parameters, use the par() function. To temporarily set them, pass them as arguments in plot.*

2.09 LINE WIDTHS AND TYPES (10)

Lines can be of many types and widths (don't type all of this stuff in)...

```
plot(NULL,NULL,xlim=c(0,15),ylim=c(0,30))
lines(c(1,1),c(1,25),lty=1)
lines(c(2,2),c(1,25),lty=2)
lines(c(3,3),c(1,25),lty=3)
lines(c(4,4),c(1,25),lty=4)
lines(c(5,5),c(1,25),lty=5)
lines(c(6,6),c(1,25),lty=6)
lines(c(8,8),c(1,25),lwd=1)
lines(c(9,9),c(1,25),lwd=2)
lines(c(10,10),c(1,25),lwd=3)
lines(c(11,11),c(1,25),lwd=4)
lines(c(12,12),c(1,25),lwd=5)
lines(c(13,13),c(1,25),lwd=6)
text(c(3,10),c(29,29),labels=c("lty","lwd"))
text(1:13,rep(27,13),labels=c(1,2,3,4,5,6,"",1,2,3,4,5,6))
```

***lty** is used to set the line type*

***lwd** is used to set the line width*

2.10 COLORS (20)

Fun with colors...

```
colors()
plot(rnorm(10000),col="maroon")
plot(rnorm(10000),col="#12ABEF")
plot(rnorm(10000),col=rgb(0.8,0.7,0.1))
plot(rnorm(10000),col=hsv(0.6,0.9,0.9))
plot(rnorm(10000),col=hcl(h=250,c=20,l=65,alpha=0.5))
plot(rnorm(10000),col=gray(0.7))
plot(rnorm(10000),col=gray(runif(10000)))
plot(rnorm(10000),col=rainbow(10000))
plot(rnorm(10000),col=rainbow(1000))
plot(rnorm(10000),col=heat.colors(10000))
plot(rnorm(10000),col=terrain.colors(10000))
```

There are lots of pre-defined colors

*The **col** argument is used to specify the color(s)*

Plot using a hexadecimal rgb value

*Plot using **rgb** values between zero and one*

*Colors defined by **hue, saturation, and value***

*Colors defined by **hue, chroma, and luminance***

*Colors defined as value along **gray scale***

*You can plot using a **vector of colors***

***Rainbow** creates a spectrum of colors*

*If $\text{length}(\text{col}) < \text{length}(x)$, **colors are recycled***

*Other color spectra exist, e.g., **heat***

*Other color spectra exist, e.g., **terrain***

Histograms can also be colored...

```
hist(rnorm(10000), col = hcl(240))  
hist(rnorm(100000), col=rainbow(15))
```

By combining different colors, line widths, line types and point types, you can make graphs look very nice (don't type all of this)...

```
plot(NULL,xlim=c(0,50),ylim=c(0,16))  
y1=rnorm(50,3)  
lines(y1,col=rgb(1,0.5,0.5),lwd=3)  
points(y1,col=rgb(1,0,0),pch=15,cex=1)  
y2=rnorm(50,6)  
lines(y2,col=rgb(0.5,0.5,1),lwd=3)  
points(y2,col=rgb(0,0,1),pch=21,bg=rgb(0.5,0.5,1),cex=1.5,lwd=2)  
y3=rnorm(50,9)  
lines(y3,col=rgb(0,0,0),lwd=7)  
lines(y3,col=rgb(0.4,1,0.4),lwd=3)  
points(y3,col=rgb(0,0,0),pch=23,bg=rgb(0,1,0),cex=1.5,lwd=3)  
y4=rnorm(50,12)  
lines(y4,col=rgb(0,0,0),lwd=7)  
lines(y4,col=rgb(1,1,0),lwd=3,lty=2)  
points(y4,col=rgb(0,0,0),pch=23,bg=rgb(1,1,0),cex=1.5,lwd=3)
```

To save your work of art:

Choose File > Save As

or

```
pdf("plot.pdf")
```

...<plotting commands go here>...

```
dev.off()
```

3.01 IF AND ELSE (10)

```
x = 3
```

Open up a new script and type the following:

```
if(x < 3){  
    cat("x is less than 3\n")  
}
```

This statement will only be executed if $x < 3$

After saving the script as "ifscript.r", type:

```
source("ifscript.r")
```

Now modify the script to read:

```
if(x < 3){  
    cat("x is less than 3\n")  
}else{  
    cat("x is not less than 3\n")  
}
```

This statement will only be executed if $x < 3$

This statement will only be executed if $x \geq 3$

After saving the script, type:

```
source("ifscript.r")
```

Finally, modify the script to read:

```
if(x < 3){  
    cat("x is less than 3\n")  
}else if(x == 3){  
    cat("x is equal to 3\n")  
}else{  
    cat("x is greater than 3\n")  
}
```

This statement will only be executed if $x < 3$

This statement will only be executed if $x = 3$

This statement will only be executed if $x > 3$

After saving the script, type:

```
source("ifscript.r")
```

Modify x and reload the script

3.02 For LOOPS (10)

Open up a new script and type the following:

```
for(i in 1:10){  
  cat(i, "\t")  
}  
cat("\n")
```

This statement will be repeated 10 times

This statement prints a carriage return

Execute the script after saving it as “forscript.r”...

Now modify the script to read:

```
x=seq(-5,5,0.1)  
plot(x,dnorm(x),xlim=c(-5,5),ylim=c(0,0.4),type="l",lwd=2)  
for(i in 1:20){  
  lines(x,dnorm(x,sd=1+i/5),lwd=2)  
}
```

Execute the new script

If statements can go inside for loops...

```
x=seq(-5,5,0.1)  
plot(x,dnorm(x),xlim=c(-5,5),ylim=c(0,0.4),type="l",lwd=2)  
for(i in 1:20){  
  if(i%%2==0){  
    lines(x,dnorm(x,sd=1+i/5),lwd=2,col="black")  
  }else{  
    lines(x,dnorm(x,sd=1+i/5),lwd=2,col="orange")  
  }  
}
```

when I is even

when I is odd

3.03 MORE FUN WITH FUNCTIONS (15)

Keep in mind that functions have their own (temporary) working environment...

```
x = 3
setxto5 = function(){ x = 5 }
setxto5()
x
[1] 3
```

This function sets the object x equal to 5

The value of x did not change!

```
x = 3
x <- 3
3 -> x
x <<- 3
3 ->> x
```

Create a local object named x and assign to it the value 3

Create a local object named x and assign to it the value 3

Create a local object named x and assign to it the value 3

*Create a **global** object named x and assign to it the value 3*

*Create a **global** object named x and assign to it the value 3*

```
x <<- 3
setxto5 = function(){ x <<- 5 }
setxto5()
x
[1] 5
```

Use the <<- operator to create a global object

This function sets the object x equal to 5

Now we see the value of x was changed

```
setxto5 = function(){ x <- 5 }
setxto5 = function(){ x <<- 5 }
```

Note: This will not work

Note: This will work

3.04 SORTING VECTORS AND DATA FRAMES (20)

Sorting vectors is different from sorting tables ...

```
x=rnorm(100)
x=sort(x)
```

Sorting a vector is easy

*Last time we loaded a table from a delimited text file.
Here is another way to create a table...*

1) Define each column as a vector, specifying which are factors...

```
pop=factor(rep(1:5,20))
sex=factor(c(rep("male",50),rep("female",50)))
reprod=sample(c(T,F),100,replace=T)
bodysize=c(rnorm(50,10),rnorm(50,10.5))
weight=bodysize*10+rnorm(100,10)
sex
reprod
```

Note the differences between these two objects

2) Group the columns into a single wrapper object using the list function...

```
frogs=list(pop=pop,sex=sex,reprod=reprod,bodysize=bodysize,weight=weight)
```

3) Convert the object into a data frame (table)...

```
frogs=data.frame(frogs)
rm(pop,sex,reprod,bodysize,weight)
```

The data are in the table, so we can remove the individual vectors

```
attach(frogs)
```

The attach function allows us to use the column names directly

```
pop
```

Recall the populations

```
frogs=frogs[order(sex,pop,reprod),]
```

Sort the data frame by sex, then by pop, then by reprod

```
pop
frogs$pop
```

Note: the order function returns a list of ranks given the priorities given

*Recall the populations, note that the order **did not** change*

*Recall the populations this way, note that the order **did** change*

```
attach(frogs)
pop
```

It is probably a good idea to use the attach function after sorting

Now the pop data are correct with respect to the sorted table

3.05 RANDOMIZATION TESTS (WORKED OUT) (20)

Let's begin by performing a simple two-sample t-test...

$H_0: \mu_F = \mu_M$

(μ_F and μ_M are the mean weights in females and males respectively)

$H_A: \mu_F \neq \mu_M$

`t.test(weight[sex=="female"], weight[sex=="male"])` *t.test performs the parametric version of this test*

Now let's test the same hypotheses using a randomization test...

In general, the steps are as follows:

- 1) compute the test statistic on the original data*
- 2) randomize the data in the appropriate way*
- 3) recompute the test statistic*
- 4) repeat 2-3 many times to generate the null distribution*

For the example at hand, the test statistic is the difference between the mean weights of males and females:

`is.male = sex=="male"`

This vector will come in handy later

`test.stat = mean(weight[!is.male]) - mean(weight[is.male])`

`null.dist = rep(NA, 100000)`

Setup the vector to hold the points from the null distribution

`for(i in 1:100000){`

`is.male=sample(is.male)`

Randomize the order of the vector

`null.dist[i] = mean(weight[!is.male]) - mean(weight[is.male])`

`}`

`cat("p.value=", mean(abs(null.dist) > abs(test.stat)), "\n")`

Compute the p-value

`hist(null.dist, nclass=100)`

3.06 RANDOMIZATION TESTS (YOU TRY) (30)

Now try to set up these randomization tests on your own...

- 1) Test the null hypothesis of no significant variation in mean bodysize among populations.*
- 2) Test the null hypothesis of no significant variation in mean bodysize among populations in males.*

The “answers” are on the next page...don’t cheat unless you get stuck!

3.07 PACKAGES AND DEMOS (0)

Some useful things to know:

There are >1600 packages online that are not included by default in your R installation. You can view them at:

<http://cran.r-project.org/web/packages/>

To install a package, open R and type:

```
install.packages("packagename")
```

, where packagename is the name of the package you desire (make sure to use the quotes).

Once the package is installed on your computer, you only need to load the package using:

```
library(packagename)
```

each time you load R.

Also note that some functions have demos, for example, try:

```
demo(persp)
demo(graphics)
demo(Hershey)
demo(plotmath)
```

```
demo()
```

Use this to get a list of all available demos in the installed packages.

“Answer” to 1)

```
pops=pop
calcTestStat = function(){      var( c(  mean(bodysize[pops==1]),
                                          mean(bodysize[pops==2]),
                                          mean(bodysize[pops==3]),
                                          mean(bodysize[pops==4]),
                                          mean(bodysize[pops==5]) ) )    }

test.stat = calcTestStat()
null.dist = rep(NA,100000)
for(i in 1:100000){
  pops=sample(pops)
  null.dist[i] = calcTestStat()
}
cat("test.stat= ",test.stat, "\n")
cat("p.value= ",mean(null.dist>test.stat), "\n")
hist(null.dist,nclass=100)
```

“Answer” to 2)

We want to focus on the males

```
pops=pop[sex=="male"]
bodysizem=bodysize[sex=="male"]

calcTestStat = function(){      var( c(  mean(bodysizem[pops==1]),
                                          mean(bodysizem[pops==2]),
                                          mean(bodysizem[pops==3]),
                                          mean(bodysizem[pops==4]),
                                          mean(bodysizem[pops==5]) ) )    }

test.stat = calcTestStat()
null.dist = rep(NA,100000)
for(i in 1:100000){
  pops=sample(pops)
  null.dist[i] = calcTestStat()
}
cat("test.stat= ",test.stat, "\n")
cat("p.value= ",mean(null.dist>test.stat) , "\n")
hist(null.dist,nclass=100)
```