

Table of contents

Lecture 1	2
Using R	2
TeXmacs	4
	4
The TeXmacs interface	4
Help in TeXmacs	4
Opening files	4
The status line	5
Starting R inside TeXmacs	5
Important hints for the R interface in TeXmacs	6
Managing R input and Text	6
The basics of R	7
More usefull functions in R	7
Vectors	7
Subsetting vectors, and operation on vectors	9
What is statistics?	12
So?	14
Some simple random functions in R	14
sample	14
rnorm	15

Lecture 1

Using R

Let us start R from the command line. First make a directory, and then run R inside it:

```
ssh bio41 -X -lcourse
```

or

```
ssh bioXX -X -lusername
```

Then, in the shell do the following:

Shell session inside TeXmacs

```
shell] cd class
```

```
shell] ls -la
```

```
total 24
drwxr-sr-x  2 dirk  dirk    4096 Jan 25 16:15 .
drwxr-sr-x 79 dirk  dirk   20480 Jan 25 16:15 ..
```

```
shell] R
```

Now we start R (here I am cheating, and starting R inside texmacs)

```
library(TeXmacs,lib.loc="/usr/share/texmacs/TeXmacs/plugins/r/r/")
```

```
R : Copyright 2003, The R Foundation for Statistical Computing
Version 1.8.1 (2003-11-21), ISBN 3-900051-00-3
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
```

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.
```

```
>
```

```
> a=5
> q()
Save workspace image? [y/n/c]: y
```

To quit R use the function `q()`.

After we quit, look at the directory again:

```
shell] ls -la
total 32
drwxr-sr-x  2 dirk  dirk    4096 Jan 25 16:17 .
drwxr-sr-x 79 dirk  dirk   20480 Jan 25 16:15 ..
-rw-r--r--  1 dirk  dirk     56 Jan 25 16:17 .RData
-rw-----  1 dirk  dirk     8 Jan 25 16:17 .Rhistory
```

```
shell]
```

When you quit R asks whether to save everything. If you say yes, then all the data is saved, and loaded next time you start R from the same directory. The data is stored in the file `.RData`

```
library(TeXmacs,lib.loc="/usr/share/texmacs/TeXmacs/plugins/r/r/")
```

```
R : Copyright 2003, The R Foundation for Statistical Computing
Version 1.8.1 (2003-11-21), ISBN 3-900051-00-3
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
```

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.
```

```
[Previously saved workspace restored]
```

```
>
> a
[1] 5
>
```

As you see, after restarting, R loaded the data, and thus the variable `'a'` was restored.

TeXmacs

TeXmacs provides a nice interface for working with R.

Do the same as before, but now start texmacs instead of R

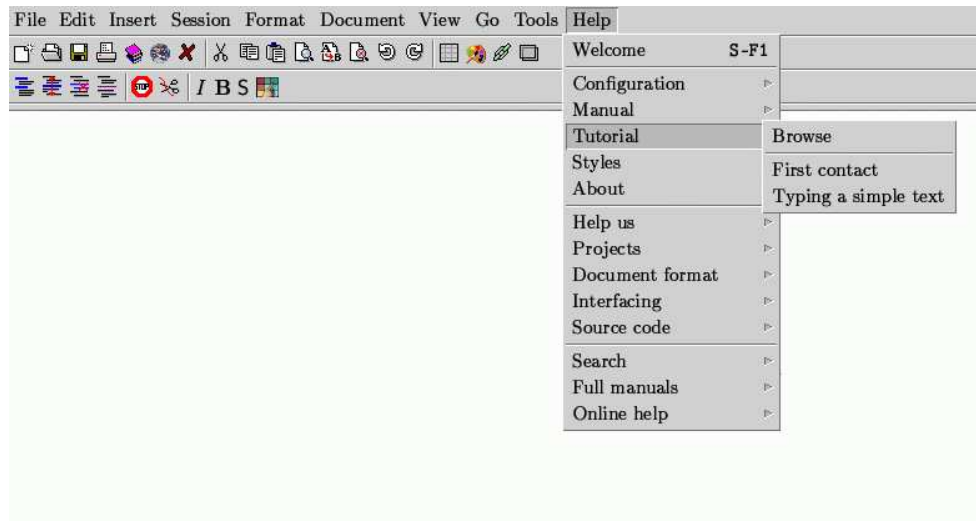
```
shell] cd ~/class
shell] texmacs
```

The TeXmacs interface

TeXmacs is very similar to Emacs, so if you know Emacs keystrokes, they usually work in TeXmacs.

Help in TeXmacs

To get help in TeXmacs, click on the Help menu



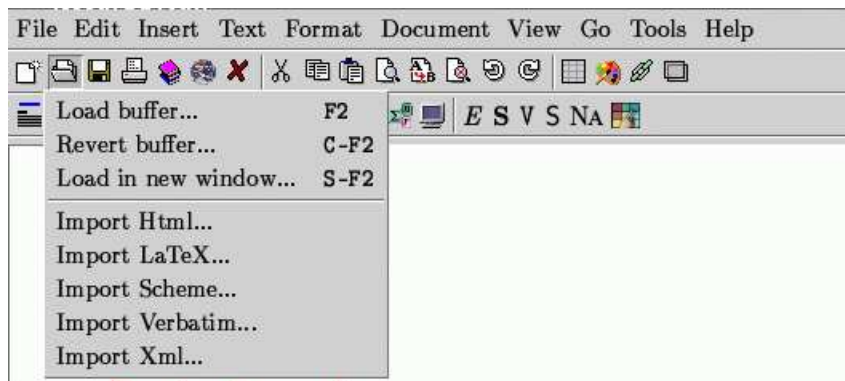
The tutorials are very basic. Try opening one of them.

To close the help that you open, click on File->Close buffer,

Or select another buffer from the 'Go' menu.

Opening files

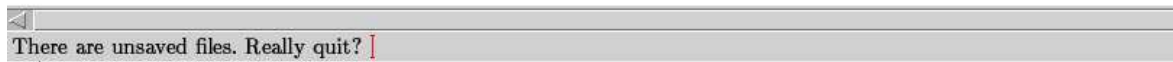
To open a file, select 'File->Load', or click on the file open icon.



Notice that some entries list their keyboard equivalents.

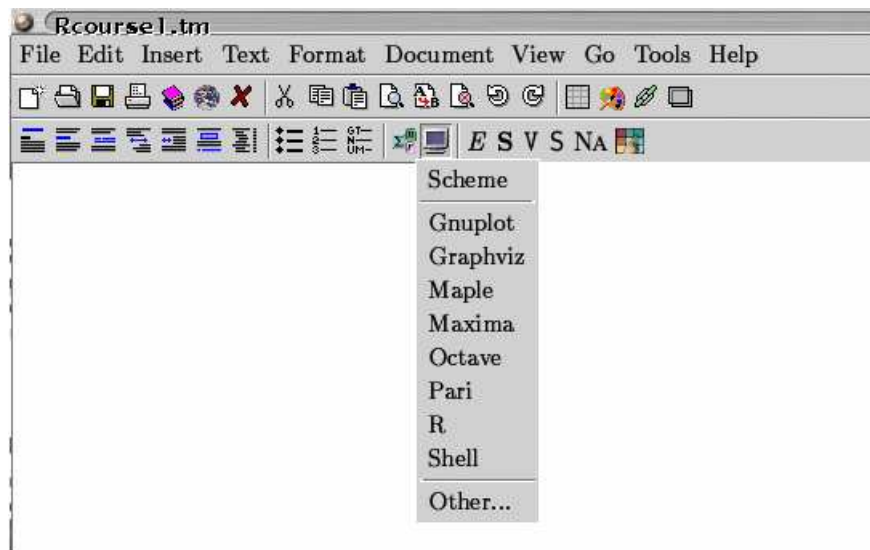
The status line

At the bottom of the TeXmacs window is the status line. Sometimes, TeXmacs will ask you a question there - remember to look!



Starting R inside TeXmacs

To start R, you need to click on the little monitor icon



Which should give you the following:

```
library(TeXmacs,lib.loc="/usr/share/texmacs/TeXmacs/plugins/r/r/")
```

R : Copyright 2003, The R Foundation for Statistical Computing
Version 1.8.1 (2003-11-21), ISBN 3-900051-00-3

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

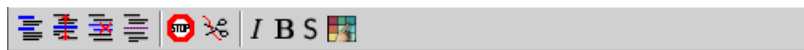
[Previously saved workspace restored]

>

>


Important hints for the R interface in TeXmacs

There are two important icons: the 'STOP' and the 'scissors':



You see them only when you are inside the R session.

The  will stop the current R session, maybe in a long calculation.


The  will exit the current R session, and a new one will be started when another R command is entered.

If, for some reason the output from the R session becomes uncoordinated with your input, just enter @@@ on an empty input line:

> @@@

> a=6

>

(You might then need to hit the  to regain the input)

Managing R input and Text

To enter text (i.e. non-R input), just click beyond the blue lines that deliniate the R session. To continue with the R session, just select R from the menu on the monitor icon.

> a

[1] 6

>

The basics of R

When you enter an expression without parentheses at the end, R prints the expression

```
> a
[1] 6
> q
function (save = "default", status = 0, runLast = TRUE)
.Internal(quit(save, status, runLast))
<environment: namespace:base>
>
```

as you see, when we entered 'q', and not 'q()', R printed the function q(). To actually call that function, we have to call it with 'q()'.

More usefull functions in R

help.start() - brings up a help system:

```
> help.start()
Making links in per-session dir ...

If netscape is already running, it is *not* restarted, and you must
switch to its window.
Otherwise, be patient ...

>
```

ls() lists all objects in memory:

```
> ls()
[1] "a"
> b=5
> ls()
[1] "a" "b"
>
```

rm() removes an object:

```
> rm(b)
> ls()
[1] "a"
>
```

Vectors

Data in R is in general stored as vectors, or lists. First we'll learn about vectors.

When we printed the value of a before, we also saw '[1]':

```
> a
[1] 5
```

>

That is because a is actually a vector that contains one element, 5. There are many methods to create longer vectors:

> 3:9

```
[1] 3 4 5 6 7 8 9
```

>

the colon creates a vector in ascending order. The '[1]' in front of the first element tells us that 3 is element number 1. This is useful for longer vectors:

> -50:50

```
[1] -50 -49 -48 -47 -46 -45 -44 -43 -42 -41 -40 -39 -38 -37 -36 -35 -34 -33
[19] -32 -31 -30 -29 -28 -27 -26 -25 -24 -23 -22 -21 -20 -19 -18 -17 -16 -15
[37] -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3
[55] 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
[73] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
[91] 40 41 42 43 44 45 46 47 48 49 50
```

>

Here we see that -32 is the 19th element of the vector.

We can store a vector in a variable:

> b = 5:100

> b

```
[1] 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[20] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
[39] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
[58] 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
[77] 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
[96] 100
```

> length(b)

```
[1] 96
```

> ls()

```
[1] "a" "b"
```

>

length() tells us how many elements are in a vector. We can also see that the result of ls() is also a vector, a vector of strings.

The function c() allows us to list elements in the vector one after the other:

> c(1,4,10,2)

```
[1] 1 4 10 2
```

>

With c() you can also join several vectors to one longer one

> c(1,4:9,2)

```
[1] 1 4 5 6 7 8 9 2
```

> c(1:3,1:3,1:3)

```

[1] 1 2 3 1 2 3 1 2 3
> a=1:4
> c(a,a,a)
[1] 1 2 3 4 1 2 3 4 1 2 3 4
> c(a,ls())
[1] "1" "2" "3" "4" "a" "b"
>

```

In the last example you see that when you put a vector of numbers together with a vector of strings, the result is a vector of strings.

With rep() you can repeat something several times, to create a long vector:

```

> rep(3,4)
[1] 3 3 3 3
> rep("R",5)
[1] "R" "R" "R" "R" "R"
> rep(1:4,10)
[1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2
[39] 3 4
>

```

Subsetting vectors, and operation on vectors

It is easy to get elements of a vector:

```

> b
[1] 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[20] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
[39] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
[58] 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
[77] 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
[96] 100
> b[2]
[1] 6
> b[2]=13
> b[2]
[1] 13
> b
[1] 5 13 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[20] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
[39] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
[58] 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
[77] 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
[96] 100
>

```

But it is also easy to address subvectors of vectors:

```

> b[1:4]

```

```

[1] 5 13 7 8
> b[3:7]
[1] 7 8 9 10 11
> b[c(1,3,5)]
[1] 5 7 9

```

>

You just list inside the square brackets [] the vector of the elements you are addressing. You can also set them:

```

> b[2:6]
[1] 13 7 8 9 10
> b[2:6]=rep(1,5)
> b[2:6]
[1] 1 1 1 1 1
> b
[1] 5 1 1 1 1 1 11 12 13 14 15 16 17 18 19 20 21 22 23
[20] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
[39] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
[58] 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
[77] 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
[96] 100

```

>

Mathematical operations will usually work on a whole vector:

```

> a=1:4; print(a)
[1] 1 2 3 4
> a+2
[1] 3 4 5 6
> a * 3
[1] 3 6 9 12
> a ^ 7
[1] 1 128 2187 16384
> sin(a)
[1] 0.8414710 0.9092974 0.1411200 -0.7568025

```

>

Other than strings and numbers, there are also vectors of boolean values (i.e. TRUE and FALSE)

```

> a = 1:4 ; print(a)
[1] 1 2 3 4
> a<2
[1] TRUE FALSE FALSE FALSE
> a>2

```

```

[1] FALSE FALSE TRUE TRUE
> a==2
[1] FALSE TRUE FALSE FALSE
> c(T,T,F)
[1] TRUE TRUE FALSE

```

>

You see that one can use T and F as an abbreviation for TRUE and FALSE. Also notice that to test for equality, we used two equal signs: '=='.

Boolean vectors give an alternative way to get at elements of a vector: instead of listing the elements that we want to address, we can give a boolean vector saying if we want or don't want certain elements:

```

> a
[1] 1 2 3 4
> a[c(F,T,T,F)]
[1] 2 3

```

>

This gives us very convenient access to elements:

```

> b
[1] 5 1 1 1 1 1 11 12 13 14 15 16 17 18 19 20 21 22 23
[20] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
[39] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
[58] 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
[77] 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
[96] 100

```

```

> b[b<12]
[1] 5 1 1 1 1 1 11

```

```

> b[b>90] = b[b>90] + 4

```

```

> b
[1] 5 1 1 1 1 1 11 12 13 14 15 16 17 18 19 20 21 22 23
[20] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
[39] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
[58] 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
[77] 81 82 83 84 85 86 87 88 89 90 95 96 97 98 99 100 101 102 103
[96] 104

```

When you join a boolean vector to a numeric vector, TRUE becomes 1, and FALSE becomes 0:

```

> c(T,F,3)
[1] 1 0 3
> a<10
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE
> (a<10)+2
[1] 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

```

>

The paratheses above are necessary, and one often needs them in R. Compare:

> a<10

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE
```

> a<10+2

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE
```

> 1:10

```
[1] 1 2 3 4 5 6 7 8 9 10
```

> 1:10+2

```
[1] 3 4 5 6 7 8 9 10 11 12
```

> 1:(10+2)

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12
```

> length(a)

```
[1] 30
```

> 1:length(a)-1

```
[1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
[26] 25 26 27 28 29
```

> 1:(length(a)-1)

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29
```

I often make the above mistake.

If you want to address all elements but some, you use negative indexes:

> a

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30
```

> a[-1]

```
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
[26] 27 28 29 30
```

> a[-(1:10)]

```
[1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

>

What is statistics?

“Statistics is the science of learning from experience” (Brad Efron)

Statistics can do various things, here we will deal with the following:

- Given an experiment we would like to estimate a parameter (in the world).

- Given an experiment, we would like to test whether a certain hypothesis is can be said to be unlikely.

These are essentially the same thing. If we knew the world exactly, we would be able to say for sure if a hypothesis is true or false, and we would know the parameter exactly.

Two factors prevent us from gaining certainty:

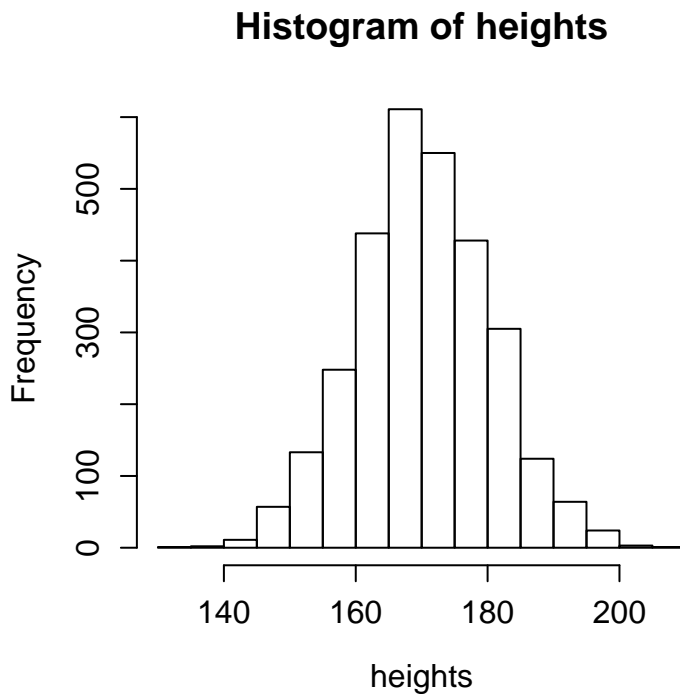
1. Sampling
2. Measurement error

Let us look at an example:

Let us say that in a school there are 3000 students, and that we wish to know the average height.

First, let us chose the heights:

```
> heights=rnorm(3000,mean=170,sd=10)
> hist(heights);v()
```



```
>
```

These are the heights of all the students in school. We can easily calculate their mean:

```
> mean(heights)
[1] 170.2498
```

```
>
```

Now let us assume that we want to estimate the mean height. we take 10 students, measure their height with an error of around 1cm, and then round to the nearest cm:

```
> measured.heights=sample(heights,10)+rnorm(10,mean=0,sd=1)
```

```
> measured.heights=round(measured.heights)
> measured.heights
[1] 176 172 149 186 179 168 146 187 166 160
>
```

Now we can calculate the mean height in the sample

```
> mean(measured.heights)
[1] 168.9
>
```

So, we know that this is the mean of the sample. But what does it say about the real mean? What are the ranges that we believe now the real mean to be?

If we knew the sampling process, and all the various errors involved in measuring (and in this case we do know it), then we could estimate how far we are from the real mean.

But how would we know all those parameters? We would have to estimate them, too!

Bootstrapping (boot-straping) allows us some magic without all the estimation. It says that a good estimate for distribution that we measured are our measurements. so instead of trying to estimate heights and their error and sampling, we just sample from the measurements we have:

```
> b=sample(measured.heights,10,replace=T)
> mean(b)
[1] 170.7
> b=sample(measured.heights,10,replace=T)
> mean(b)
[1] 172.2
> b=sample(measured.heights,10,replace=T)
> mean(b)
[1] 169.3
>
```

Now we have some idea in what range the real mean could be.

So?

This was done to show two things:

1. R is really convenient for addressing such matters. We can easily simulate situations and then assess our statistical methods.
2. Bootstrapping is useful, simple, and (as we'll see later), convenient to do with R.

Some simple random functions in R

We saw the following 2 ways of generating random data in R:

`sample`

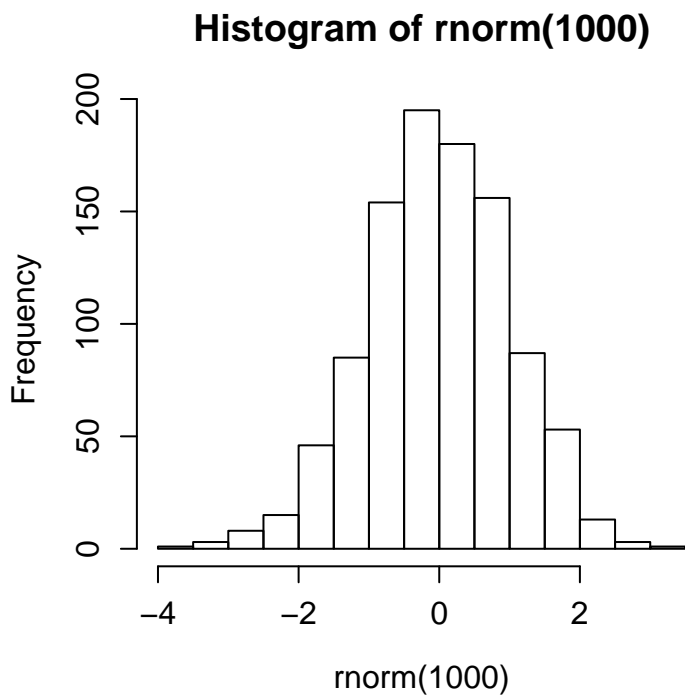
Sample randomly:

Roll a die

```

> sample(1:6,1)
[1] 3
>
Roll 50 die:
> sample(1:6,50,replace=T)
[1] 3 5 6 1 6 5 1 3 2 2 6 5 5 4 1 1 4 6 6 3 4 2 5 5 4 3 3 2 5 5 5 2 2 2 6 5 2 4
[39] 5 1 6 5 6 1 6 3 5 6 4 3
>
or
> sample(6,50,replace=T)
[1] 3 1 3 5 3 2 4 4 3 4 1 1 3 3 5 5 4 5 2 5 1 4 3 1 4 2 2 2 1 1 3 6 5 6 6 1 4 5
[39] 2 6 1 5 2 1 3 1 2 2 4 1
>
rnorm
Sample from a normal distribution
> rnorm(10)
[1] -1.9329962  1.0924528  0.5751292 -0.3561676  0.2812530 -1.3551893
[7] -1.4555352  1.7605307 -0.6611010  0.6395928
> rnorm(10,mean=3,sd=0.1)
[1] 3.032633 3.123962 2.989857 3.142417 2.874144 2.855629 3.048683 2.898982
[9] 2.923673 2.987789
> hist(rnorm(1000));v()

```



>