Simulation with R

These notes are not a complete introduction to R. They are designed to give you enough information that in conjuction with Rs built in help you can complete the exercises on sampling distribtuions, maximum likelihood and bootstrapping.

Finding what you need

The hardest part about learning a new program is trying to find functions you know exist but you don't know what they are called. The easiest way to find what you need in R is with,

```
help.search('keyword')
```

R also has a HTML version of the help pages that you can access using `help.start()`. If you have never used R before this is a good place to start - there is a link to an introductory manual.

If you need more information on any of the functions talked about in these notes, try

```
?topic
```

or

```
help('topic')
```

This will bring up the relevent help page and hopefully answer any questions.

Simulating random variables

R contains by default many distributions. For example, `?Normal` tells us about four functions relating to the normal distribution,

- `pnorm` - the distribution function
- `rnorm` - a function to generate normal random variates
- `qnorm` - the quantile function
- `dnorm` - the density function

All of these functions can take arguments to specify the mean and variance of the distribution. In general, the first letter of the function determines what information about the distribution we want and then the distribution nickname follows directly. For example, if we want the density function for the Poisson distribution we use `dpois`. Some useful distributions are:

For more info	Nickname	Distribution
`?Normal`	`norm`	the normal distribution
`?Possion`	`pois`	the poisson distribution
`?Binomial`	`binom`	the binomial distribution
`?Geometric`	`geom`	the geometric distribution
`?TDist`	`t`	the Students t distibution
`?FDist`	`f`	the F distribution
`?Chisquare`	`chisq`	the Chi-Squared Distribution

Remember to check the functions define the distribution the way you expect.

Often we want to draw random numbers but want our results to replicable. The `set.seed` command lets us set the starting point for the random number generation and if we use the same seed we will always get the same random numbers (note: sometimes this is not what you want).

Example

```
> rnorm(10)

 [1] -0.30125091 -0.23552783 -0.61879922  0.23110361  1.31541443  0.41800526
 [7]  0.39434739  1.53440933 -0.37052444  0.03481197

> rnorm(10)

 [1] -1.25067676 -1.60956084  0.29984995 -0.58450931 -1.03388333  1.40477092
 [7] -0.04511211  1.21148672  0.15405686 -0.42036729

> set.seed(18749)
> rnorm(10)

 [1] -0.54555908  1.22938955  2.58249311  1.57057115 -0.03186376  0.32983807
 [7] -0.44899542  0.56599635 -0.23902963  0.79350037

> set.seed(18749)
> rnorm(10)

 [1] -0.54555908  1.22938955  2.58249311  1.57057115 -0.03186376  0.32983807
 [7] -0.44899542  0.56599635 -0.23902963  0.79350037
```

It is often useful to be able to randomly draw from a set of numbers we have already (for example when bootstrapping). This is done using the function `sample`. We supply `sample` with a vector of numbers to draw from. Without any other arguments `sample` will randomly permute the vector. If in addition we supply an argument `size`, `sample` will randomly draw `size` numbers from our vector without replacement. There is also an argument `replace` that specifies whether drawing should be with replacement (by defualt without replacement). The argument `prob` can be used to specify a probability distribution other than uniform with which to draw the numbers.

Example

```
> sample(1:10)

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

> sample(1:10, size = 5)

[1]  5  8  6  1 10

> sample(1:10, size = 15, replace = TRUE)

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

Basic functions

Being a statistical package `R` has plenty of built in functions for preforming basic statistical operations. Some are given below and you can probably guess many others.

Example

```
> x <- rnorm(20, mean = 10, sd = 20)
> mean(x)

[1] 8.144194

> sd(x)

[1] 17.36403

> sqrt(var(x))

[1] 17.36403

> median(x)

[1] 8.448636

> quantile(x, probs = c(0.05, 0.95))

       5%       95%
-18.35355  31.73562

> summary(x)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-18.630  -6.728   8.449   8.144  25.220  36.880
```

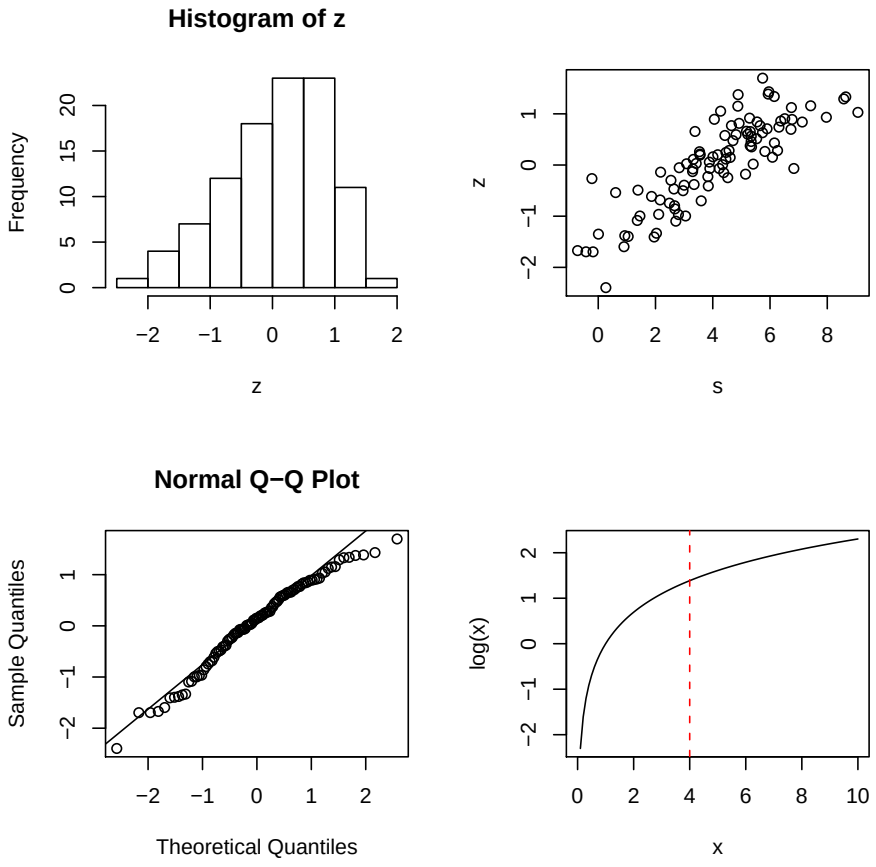
There are also some basic mathemathical functions that will be useful.

```
> y <- 1:20
> length(y)

[1] 20

> max(y)

[1] 20

> min(y)

[1] 1

> log(y)

 [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
 [8] 2.0794415 2.1972246 2.3025851 2.3978953 2.4849066 2.5649494 2.6390573
[15] 2.7080502 2.7725887 2.8332133 2.8903718 2.9444390 2.9957323

> sum(y)

[1] 210

> prod(y)

[1] 2.432902e+18
```

Plots

There are lots of ways to create plots in `R`. Functions like `hist`, `boxplot` and `qqnorm` produce standard statistical plots given some data. `plot` can be adjusted to plot a lot of different types of plot (and there are often defaults for different `R` objects, for example try `plot(log)`). There are also a number of functions that will add things to a plot: `points`, `lines`, `text`. The most useful one for the exercises is `abline` which will add a straight line to a plot.

```
> z <- rnorm(100)
> s <- 2 * z + 4 + rnorm(100)
> par(mfrow = c(2, 2))
> hist(z)
> plot(s, z)
> qqnorm(z)
> qqline(z)
> curve(log(x), from = 0, to = 10)
> abline(v = 4, col = "red", lty = "dashed")
```



Optimizing functions

Most of the functions in R for optimization are minimizers. You generally have to rewrite your problem so that it is one of minimization. For example, to maximise a likelihood you minimize the negative likelihood. Or, to find a root of a function you could minimize its square. There are many functions that will do this: `nlm`, `optim`, `optimize`, `nlminb` and `constrOptim`. My favourite is `nlminb` because it handles both one parameter and multiparameter problems and it is easy to place constraints on the parameters. At the very least you need to give the function a place to start and a function to minimize.

Example - Maximum Likelihood

Imagine we have 20 observations from an exponential distribution with unknown parameter λ (we'll simulate this data). We want to find the maximum likelihood estimate for λ We know the density for an exponential distribution is

$$f(x|\lambda) = \lambda e^{-\lambda x} \quad x \geq 0.$$

We can write the log likelihood as,

$$\begin{aligned} l(\lambda) &= \sum_{i=1}^{20} (\log \lambda - \lambda x_i) \\ &= n \log \lambda - \lambda \sum_{i=1}^{20} x_i \end{aligned}$$

So in R we want to minimize the negative of this function,

```
> x <- rexp(20, rate = 4)
> n <- length(x)
> nllhood = function(lambda) {
+     -1 * (n * log(lambda) - lambda * sum(x))
+ }
> fit <- nlminb(6, nllhood)
> fit

$par
[1] 3.491674

$objective
[1] -5.007625

$convergence
[1] 0

$message
[1] "relative convergence (4)"

$iterations
[1] 6

$evaluations
```

```
function gradient
       8        9
```

```
> fit$par
```

```
[1] 3.491674
```

apply

`apply` is an incredibly useful function when you are making the same calculation repeatedly over the columns or rows of an object. It takes three arguments. The first the object you wish to apply the function to, the second either 1 or 2 depending on whether you are working across the rows or down the columns, and the third the function you wish to apply.

Example

```
> x <- matrix(rep(1:5, 5), ncol = 5)
> x
```

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

```
> apply(x, 1, mean)
```

```
[1] 1 2 3 4 5
```

```
> apply(x, 2, mean)
```

```
[1] 3 3 3 3 3
```

```
> apply(x, 2, function(y) c(mean(y), sd(y)))
```

```
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 3.000000 3.000000 3.000000 3.000000 3.000000
[2,] 1.581139 1.581139 1.581139 1.581139 1.581139
```

Also look at `lapply`, `sapply`, `tapply` and `mapply`.

Reading in data

```
> p.data <- read.table("http://www.stat.berkeley.edu/~wickham/poisson.txt")
> head(p.data)
```

```
  V1
1  4
2  2
3  4
4  4
5 11
6  8
```