S-Plus/R Programming

- Statistical programming environment using the S language.
- Available from Mathsoft Inc.
- Free clone R is available (see link on class' page).
- For further references see the book by Venables and Ripley.
- These notes are just a brief introduction.
- I will be assuming a Unix based system but most commands work equally on Windows.
- They will also work with R.

Data Storage and Types

- **Numeric** Any real numbers, Inf or -Inf.
- **Logical** T (true) or F (false).
- **Character** A string of alpha-numeric characters enclosed by double quotes.
- **Factor** Level of a categorical variable.

  The special value NA denotes missing value of any data type.
- **Vectors** One-dimensional collections of data all of the same type.

- **Matrices** Two-dimensional collections of data all of the same type.
- **Dataframes** Two dimensional matrices for which different columns of data can be of different types. Character vectors are transformed into factors when they become a column or a row of a dataframe.
- **Arrays** Multi-dimensional arrays of data of the same type.
- **lists** A collection of an arbitrary number of elements each of which is one of the data storage types listed here.

Assignments and Subscripts

- The assignment operator is

  `<-`

  and is used for any data type or storage type.

- When creating a new data object we need to specify its type.

```
> vec1 <- c(2,4,6,8,10)
> vec1
[1] 2 4 6 8 10
> mat1 <- matrix(c(1,2,3,4,5,6),nrow=3,ncol=2)
> mat1
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> list1 <- list(vec1,mat1)
list
[[1]]
[1]  2  4  6  8 10
[[2]]
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

- For lists we can also name the components.
- This is also possible for vectors but is not as useful.

```
> list2 <- list(item1=vec1,item2=mat1,
+ item3=list(x=c(1,2,3),y=c(T,NA,F)))
> list2
$item1
[1]  2  4  6  8 10

$item2
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

$item3
$item3$x
[1] 1 2 3

$item3$y
[1]  TRUE    NA FALSE
```

- We use subscripts to get various parts of an object.
- For vectors, the subscripts can be positive or negative integers or can be a logical vector of the same length as the original
- NA cannot be used in a subscript.

```
> vec1[3]
[1] 6
> vec1[c(1,4)]
[1] 2 8
> vec1[-3]
[1]  2  4  8 10
> vec1[c(T,T,F,F,T)]
[1]  2  4 10
```

- Components of a matrix are extracted using two subscripts $[row, col]$.
- Blocks can be found using vectors for one or both of the subscripts.
- Omitting a subscript defaults to all possible subscript values.

```
> mat1[1,2]
[1] 4
> mat1[c(1,3),2]
[1] 4 6
> mat1[2,]
[1] 2 5
> mat1[2]
[1] 2
```

- For lists we use [[subscript]].

```
> list2[[1]]
[1]  2  4  6  8 10
> list2$item1
[1]  2  4  6  8 10
> list2[[3]]$x
[1] 1 2 3
```

- Any component which can be accessed using a subscript can be assigned a new value.

```
> vec1[3]<- -5
> vec1
[1]  2  4 -5  8 10
> mat1[c(1,2),]<-matrix(c(10,9,8,7),nrow=2)
> mat1
     [,1] [,2]
[1,]   10    8
[2,]    9    7
[3,]    3    6
> list2$item2 <- c("a","string","vector")
> list2
$item1
[1]  2  4  6  8 10

$item2
[1] "a"      "string" "vector"

$item3$x
[1] 1 2 3

$item3$y
[1]  TRUE    NA FALSE
```

Operators

Operator work on numeric data. These return numeric results.

```
^ exponentiation
*,/ multiplication, division
+,- addition, substraction
```

Comparison operators (return a logical value).

```
== equal
<  less than
> greater than
>= greater or equal
<= less or equal
!= not equal
```

The logical operators & (AND) and | (OR) work on logical data.

All operators will work on single data points or vectors.

Generating vectors

- Systematic data can be generated using the functions rep and seq.

seq:

```
seq(from, to, by=1)
```

```
> seq(1,5)
[1] 1 2 3 4 5
> seq(1,5,by=0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> 1:5
[1] 1 2 3 4 5
```

rep:

```
rep(x,times)
```

```
> rep(1:5,2)
 [1] 1 2 3 4 5 1 2 3 4 5
> rep(1:5,c(2,0,1,3,2))
[1] 1 1 3 4 4 4 5 5
```

- sample is a function to take samples from a vector.

```
> sample(1:10,7)
[1] 10  3  2  6  5  9  1
> sample(1:10,7,rep=T)
[1] 1 2 2 3 1 5 4
```

Random Generation

- There are also functions to generate data from various distributions.

`runif, rnorm, rbinom, rpois, rgeom, rhyper, rexp, rgamma, rbeta, rt,rchisq, rweibull and rcauchy.`

```
> n <-100
> x<- runif(n,-1,1)
> y <- rnorm(n,0,2)
```

- The "d" and "q" versions of this functions evaluate the density and the distribution functions respectively.

Help in Splus

- S-plus has an extensive online help facility.
- You can get help on any function using help(function) or ?function.
- It is often useful to use help.start() to create a special help window.
- This allows us to search for functions by strings in the name or by category.
- Help files give details of arguments and the returned value as well as examples.

Looping

- for (index in sequence). Example:

  `for(i in 1:5){  }`

  Loops over the expressions in { } once for each value in sequence.

- while (*condition*) { } The body is evaluated as long as the logical value *condition* evaluates to T.

- repeat { } Repeat forever. There must be a break statement at some point so that the loop will eventually terminate.

- It is best to use *for* or *while* for most loops.

- Loops take a lot of computing time. R is faster than S-plus.

- Other functions like apply or lapply help with loops.

Example:

Suppose that 'data' is a list of 10 dataframes each of which has columns named 'x' and 'y'. One thing we may want to do is to fit a linear regression model to each of these data and store the coefficients in a 10×2 matrix.

We can do this using the following code.

```
>data.coefs <- matrix(NA,nrow=10,ncol=2)
>for (i in 1:length(data)){
+   model <- lm(y~x,data=data[[i]])
+   data.coefs[i,]<- coefficients(model)
+ }
```

Example: (Newton-Raphson algorithm) Suppose that dll and d2ll are functions representing the first and second derivatives of the log-likelihood in a single parametric family. The following code could be used to implement the algorithm.

```
> theta.hat <- 1  # Some inital value
> theta.hat.old <- 0
> while (abs(theta.hat-theta.hat.old)>1e-6) {
+   theta.hat.old <- theta.hat
+   theta.hat <- theta.hat.old -
+                   dll(theta.hat.old,data)/
+                    d2ll(theta.hat.old,data)
+}
```

Writing Functions

- We will often want to write our own functions.
- For example the functions dll and d2ll in the Newton-Raphson example would need to be programmed by us.
- We write a function by assigning a 'name' to a function definition.
- A function definition is of the form: function(argument list) { }
- The argument list is a series of variable names to which the arguments of the function will be assigned.
- Arguments can be given default values using the form name=default.
- The function return can be used to terminate the function prior to the last statement.

```
 xsquare <- function(x){
+   y <- x^2
+   return(y) }
> xsquare(6)
[1] 36
```

Functions to compute first and second derivatives of the log likelihood in the Gamma example.

```
dlgamma <- function(k,x){
#
  n <- length(x)
  n*(log(k)-digamma(k)-log(mean(x)) +
  mean(log(x)))
 }
#
#
d2lgamma <- function(k,x) {
  n <- length(x)
  n/k - n*trigamma(k)
  }
```

Finally let us write a function to do the Newton-Raphson iteration.

```
newton <- function(data,dll,d2ll,init,
     eps=1e-6,maxiter=20){
  theta <- init
  out <- matrix(NA, nrow=maxiter+1,ncol=2)
  dlt <- dll(init,data)
  out[1,] <- c(init,dlt)
  i <- 1
  continue <- T
  while (continue) {
      i <- i+1
      theta.old <- theta
      theta <- theta - dlt/d2ll(theta,data)
      dlt <- dll(theta,data)
      out[i,] <- c(theta,dlt)
      continue <- (abs(theta-theta.old) > eps) &&
                   (i <= maxiter)
   }
   if (i > maxiter) {
     warning(``Maximum number of iterations reached'')
     return(out)
       }
    out <- out[!is.na(out[,1]),]
    out
 }
```

Example Consider the following data recording the intervals between air-conditioning failures on a Boeing 720 aircraft.

```
 487   18   100    7    98    5
  85   91    43  230     3  130
```

The gamma model should be a reasonable model for this data so let us estimate the parameters.

```
> air9 <- c(487,18,100,7,98,5,85,91,43,230,3,130)
> mu.hat <- mean(air9)
> k.iters <-newton(air9,dlgamma,d2lgamma,1)
> k.hat <- rev(k.iters[,1])[1]
> k.iters
          [,1]          [,2]
[1,] 1.0000000 -3.325184e+00
[2,] 0.5703458  2.846622e+00
[3,] 0.6781423  4.900856e-01
[4,] 0.7052513  2.056553e-02
[5,] 0.7064908  3.947537e-05
[6,] 0.7064932  1.460219e-10
[7,] 0.7064932  5.329071e-15
> c(mu.hat,k.hat)
[1] 108.0833333   0.7064932
```