Basic R Programming

if/then

This command performs an *action* if the *condition* is met. One can specify an alternative *action2* if an alternative *condition2* is met, and a further alternative *action3* if not any condition was met.

```
if (condition){
      action
} else if (condition2){
      action2
} else {
      action3
}
```

Both the parts commencing with `else` and `else if` are optional. For instance,

```
if (log(10)<pi){
      pi
} else {
      log(10)
}
```

gives the value of `pi`.

for

A `for` loop repeats an *action* for all elements of a *set*. Formally,

```
for (i in set ){
      action
}
```

For instance,

```
for (i in 1:10){
      cat('This is loop', i, '\n')
}
```

will produce 10 rows of text which report the number of the loop (The string '\ n' is borrowed from the C language and means to start a new line).

while

A `while` loop works similar as `for`, but instead of working though a *set*, it checks in every iteration whether a *condition* is met:

`while (`*condition*`){`
action
}

apply

This function allows to carry out some operation onto all rows or columns of a matrix. For instance, if `W` is a $n \times p$ matrix, then

$$\texttt{apply}(\texttt{W}, 1, \texttt{sum})$$

would give a $n \times 1$ vector which contains the sums over each row, and

$$\texttt{apply}(\texttt{W}, 2, \texttt{mean})$$

would give the column means. Useful variants are `tapply` (carries out operations on the elements of `W` grouped by a factor, the name of which is given as second argument), and `lapply` (for operations on each element of a list `W`; here the second argument is not needed).

Functions

Functions allow to prepare some code which can be used later with different function arguments. For instance,

```
testlog <- function(x){
    if (x>0){
       log(x)
    } else {
       cat("log not defined for non-positive argument.")
    }
}
```

will give the logarithm of `x` if `x` is positive, and an error message otherwise.
Functions can also have more than one argument, which are then separated by commas. Default values can be given behind a `=` symbol, for instance

```
max1<- function(a,b=1){
   result<- max(a,b)
   return(result)
}
```

```
max1(0.5)
[1] 1
max1(0.5,0)
[1] 0.5
```