Introduction to S-plus & R

Einar Heegaard
Botanical Institute
University of Bergen
Allegaten 41
5007 Bergen
Norway
einar.heegaard@bot.uib.no

What is S-plus & R?

S-plus & R are software for organisation, description, statistical analysis and presentation of information. S-plus is a commercial software and R is its free counterpart. R has a tremendous flexibility and a rapid development by the international statistical community. S-plus in particular is a combination of high flexibility with a user friendliness, and it incorporates over 4.000 different functions that can be readily applied to our observations. Both software has extended interaction with various programs, and can be used on several platforms.

Both modern and classical statistical analysis is available, and the multitude of manuals and literature provide an extensive documentation for the statistical use of both S and R for users at different statistical level.

The following text is written as a soft starter to S and R, and focus on essential part. There is a slight difference between the program as S has a higher user friendliness, but the basic language used by both software are similar. The commands and examples used below can be used in both S and R if otherwise is not explicitly stated. This introduction was written at time of S-plus version 6.0 and R 1.7.1.

R can be freely downloaded from the web-page, search for **Cran**.

S-plus organization:

1) Menus and toolbars
2) Report window
3) object browser / object explorer
4) command window
5) command history
6) history log
7) script window
8) graphical window

1) Menus and toolbars

These are found at the top of the screen when Splus is opened, and they supply an extended user friendliness by including standard analyses. Almost all functions available in Splus can be handled from the menus.
Example:

Import of data:

File → import data → from file ...

Choose file type and find the file with the browser. Name the object where Splus should organise the imported file.

2) Report window

Output window for analysis done through the menus.

3) Object explorer (Opened by button on toolbar) NB!

The left window is a list of object-types and a searchpath. The right window is a list of the contents of the particular folders in the left window. The organisation of our files is usually done through the object.explorer. Here is all available data listed and all outputs from our analyses. The data we can manipulate, and we can change the searchpath of the session. The searchpath is the way Splus searches for functions and objects to be used in the command we express. When Splus gets a request from us to print a matrix it starts looking at the top of the searchpath and then successively goes through all the folders in the searchpath until it has found the right object.

The first line is the workspace and the following folders are system folders. Workspace is where all data and results will be stored.

4) Commando window (Important)

It is in this window that the main work is done, here we can do all the analyses and exploit the Splus flexibility most. Here we can modify data, create plots, and more advanced statistical analyses can be undertaken here.

This window is opened by the command-button on the toolbar. The window always starts with the sign:

>

Now we can write different commands:
Example:

>fit.lm <- lm(y~x)

This is a least square regression of y on x, and the results of this analysis is assigned to the object fit.lm.

>x<-1:100

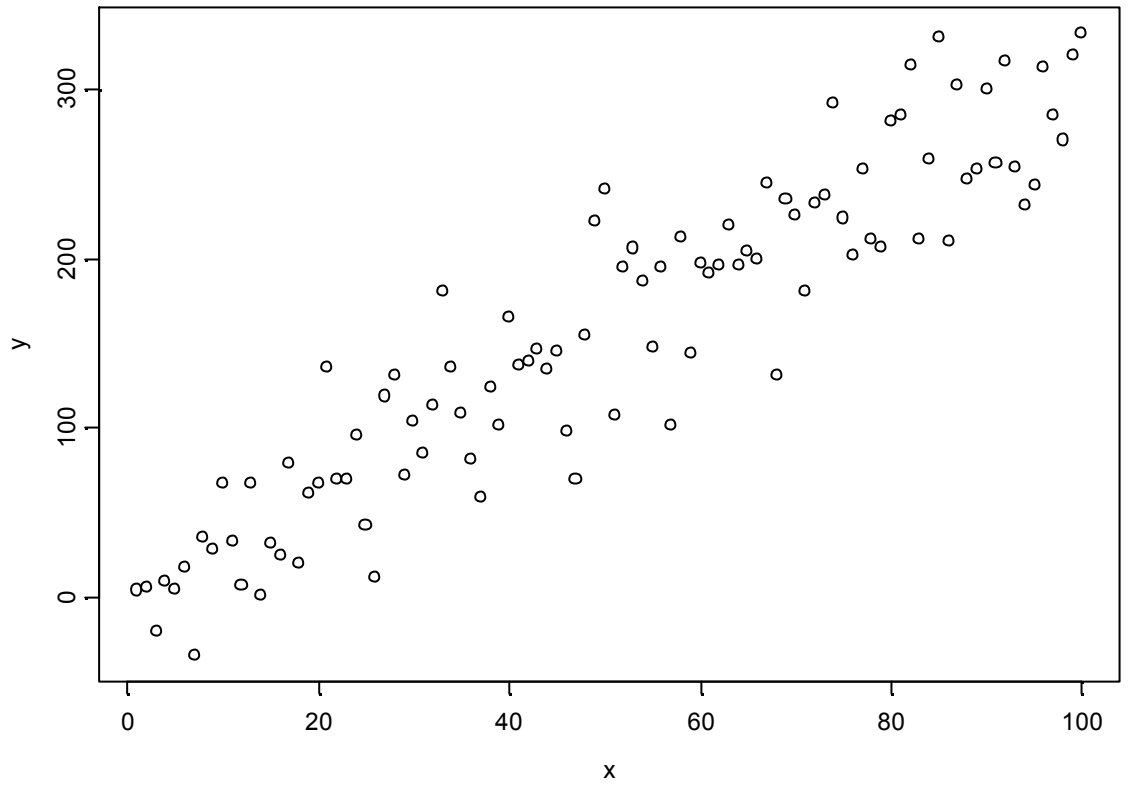
This creates a vector with number from 1 to 100

>x

[1] 1 2 3 4 .... 99 100

>plot(x,y)

This creates a scatter plot of the variables x and y, where x is the horizontal axis and y is the vertical axis.



5) Command history

A log of all commands used in the last session.

6) History log

History log during the last session, more extensive that command history.

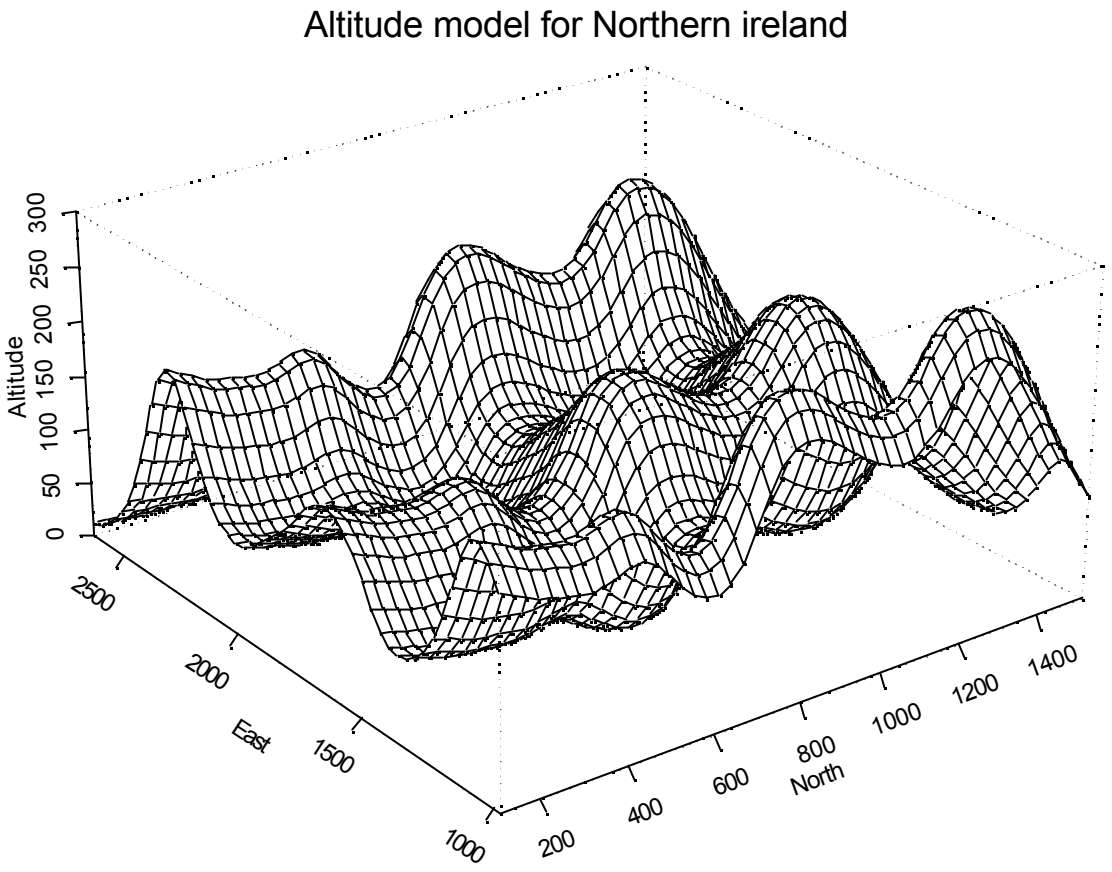
7) Script window

This is a macro-window from where we can create and run macros. In the upper window there is a command window, whereas the results after run will be listed n the lower window. In this window we can create and store large series of commands, and have them run at different times.

In the newer versions of Splus more emphasis has been on developing the script window, whereas in older version this was lacking. However, it is still recommended that we learn the command window.

8) Graphical window

This is a window for plots.



Create a new project

Version 6.0
First time Splus is opened:
Options → General settings ... → startup → mark ”promt for project folder”.

1) Create a folder on c:`... with the name of the project. (c:\skurs)
2) Open Splus-plus
3) In the box find the folder for the project (c:\skurs)
4) When Splus is opened the c:\kurs is automatically ranged as a workspace first in the searchpath.

Import of data:
Filemeny
File → import → From file...

Choose the format of the file, and find the file to be opened. Splus can take many different formats. In Splus incorporated files are stored as “data.frames” (see below), and then it is convenient to use the extension *.df to distinguish these from other files.

R organisation:

1) Command window
2) Graphical window
3) Library

R interface is different from S in that it only provides a command window and a graphical window. R does not have an object explorer, but there is a searchpath were different libraries are attach. The top level of the searchpath is the position were all our objects (datasets, and outputs) will be positioned. This top is named the workspace and can be saved to store any resent developments in our project.

1) Commando window (Important)

It is in this window that the main work is done, here we can do all the analyses and exploit the R flexibility most. Here we can modify data, create plots, and more advanced statistical analyses can be undertaken here. The command window will open automatically, and as for S is starts with the sign:

>

which tells us that we can start giving commands. We can use the same commands in both S and R (some slight differences in syntax occur for special functions).

2) Graphical window

This opens automatically when we write a graphical command within the command window.

>plot(x,y)

will open the graphical window and create a scatterplot of x (horisontal axis) and y (vertical axis), just in the same manner as for S above.

Create a new project

For creating a new project in R we open the program, and after a sessions work we store the workspace containing all objects created during the session. NB! graphs are not stored, these must be exported during the session.

To store the workspace go to the menu bar (top):

File → save workspace

Then give it a sensible name that is easy to recognise.

During the sessions we may write several commands, which we would like to store for later use. A history log can be stored by:

File → save history

This will create a file that includes all commands during the last session.

Import of Data:

As for all statistical tools we need some data. To get data into the workspace of R we need to use the command window (not the menu bar as for S).

The procedure is as follows:

1) Open your data file in a spreadsheet (xl or similar).
2) Copy the content to the clipboard
3) Then within the R command window write the command
4)

>newdata.df<-read.table("clipboard",header=T)

This will create a new object, a data.frame (see below), within the workspace. This object will contain all the data that was copied to the clipboard, with the designated names to each column found within the first row of the spreadsheet matrix. The object on the workspace will be named newdata.df. Here it is of importance to us sensible names that can easily be recognised. Also remember that both S and R are case sensitive.

Now we can use the imported data for statistical analyses.

3) Library

The library is common for both S and R, but is most extensively used within R. A library consists of an extra set of specialised functions. These functions is used to perform particular statistical procedures. For R there is a nearly continuous production of new libraries incorporating the newest statistical techniques. These libraries can be downloaded from the web page. The R comes with a basic package, that in addition contains a set of libraries. To access the function within the different libraries they have to be linked to the searchpath.

>library(MASS)

This opens the library named MASS to the searchpath, and we can access the functions within.

THE FOLLOWING SECTIONS ARE COMMON TO BOTH S AND R.

Data types

Integer – a numerical value

Character – en character

Vector – a series of character or numbers. These cannot be both in the same vector, with the exception of NA that is identified as not available.

Matrix – series of vectors in a system of rows and columns. A matrix can have different number of rows and columns, but each column must be of the same length, and each row must be of the same length. A matrix can either be characters or numerical values but not both.

Array – A multidimensional matrix.

Data.frame – this is an extension of the matrix as it can include both characters and numerical values. However, these cannot be in the same column, only in the same row. The data.frame is column-based, i.e. a matrix consisting of columns. The data.frame can be used as a database as it can be linked to the searchpath, and the columns within the data.frame can be called upon.

List – a list is the most extensive operative unit in both R & Splus. A list can contain all types of data even other lists. We can view a list as a line where we attach the different objects at different positions. In position 1 there may be a vector of characters, whereas in position 2 there may be a matrix of numerical values.

These objects are the operative units in R and Splus, and there are various functions that operate on different objects, i.e. object-oriented programming.

Command window

In the command window we communicate with the data program. Here we can use advanced statistical functions, compute simple calculations, plot graphs, create new functions, etc.

S &R are a function-based program, i.e. it consists of numerous functions that we call upon to perform specific tasks. There are several levels of functions, but the general syntax is similar. In general all functions need information (input), and this information will be given to the specific function within parentheses after the function-name.

>plot(x,y)

Plots y against x, the arguments (information) is x and y.

Some functions are generic, i.e. they read the type of object in the input and perform a proper sequence according to the function.
Say x is a factor or x is a continuous vector this will give different types of plots in the above plot-function.

In many contexts we wish to store some results (output). This we can do by using the ”assign to” sign. This is written as:

<-

>x <- 1:100

Here we ask the program to create a vector containing the numbers from 1 to 100, and to write this in the object x.
The contents of the object x can be viewed simply be calling the object;

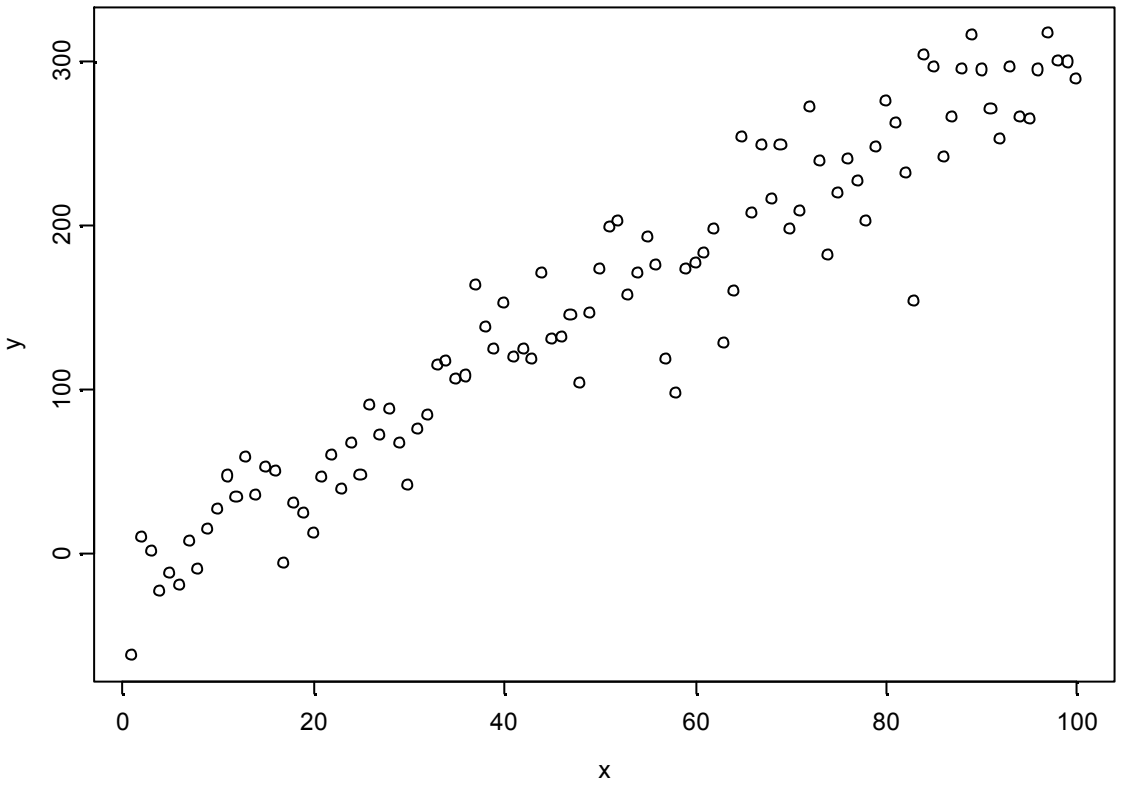
>x
[1] 1 2 3 4 100

We can store results from all kinds of functions, and use these in calculations or other functions within the program.

>x<-1:100
>y<-1+3*x+rnorm(100,0,30)

x is a numerical vector from 1 to 100
y is a vector as a function of x + a normal distributed error.
The function rnorm generates n random numbers from a Normal distribution with the parameters N(0,30).

>plot(x,y)



>fit.lm<-lm(y~x)

Here we call a function lm which performs a least square regression, and writes the results from this analysis to the object fit.lm. This object fit.lm includes a considerable amount of information about the regression. Some of which can be viewed by a simple call:

>fit.lm
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept) x
-15.73301 3.266506

Degrees of freedom: 100 total; 98 residual
Residual standard error: 28.65949

If we want a table of variance we call the anova function, and the argument we use is the object incorporating the results from a earlier used function.

>anova(fit.lm)
Analysis of Variance Table

Response: y

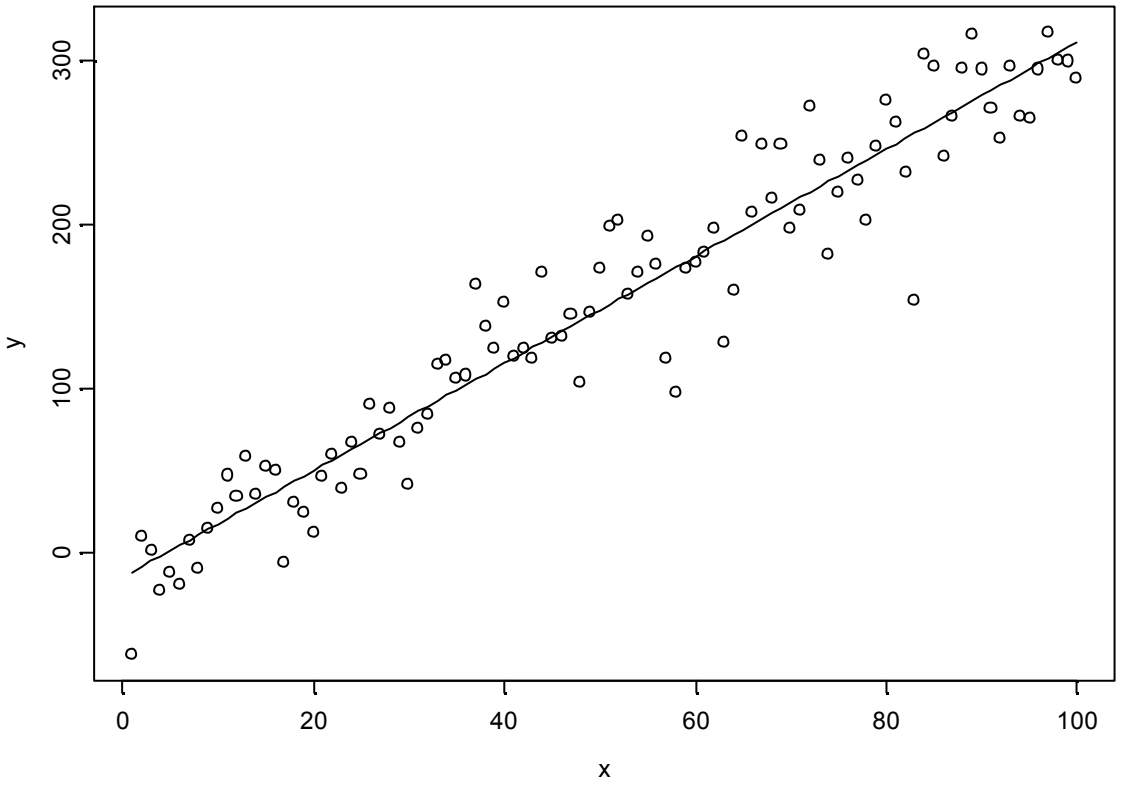
Terms added sequentially (first to last)

	Df	Sum of	Sq	Mean Sq	F	Value	Pr(F)
x	1	889082.8		889082.8	1082.443		0
Residuals	98	80493.9		821.4			

This gives a output of the least square analysis. There may be a slight variation in the output between S and R. See also the function summary for a description of the objects.

>summary(fit.lm)

If we want to view the estimated line in the earlier plot we use the command points.
>points(x,fitted(fit.lm),type="l")



This adds the line to the previous scatterplot. Here we see that within a function we can call another function, as we ask the program to find the fitted values of the lm-model (fitted(fit.lm)) for all the x values, and then subsequent draw a line through these points in the plot. A line is created by the argument type ="l". Note that we use the " " surrounding the letter 1. NB!! This is not number 1 (one).

Some important operations for R & Splus

1) How to create a vector.
This can be undertaken by several ways.

>x<-1:100
>x<-c(1,2,3,4,5)

>x
[1] 1 2 3 4 5

>x<-rep(c(1,2),3)
>x
[1] 1 2 1 2 1 2
>x<-rep(c(1,2),rep(3,2))
>x
[1] 1 1 1 2 2 2

A numerical vector can be converted into a factor or a ordered factor by:

>x<-factor(x)
or
>x<-ordered(factor)

An ordered factor includes a sequence in addition to the categorical classes, i.e. here 1 and 2 is the classes but in addition 1 is smaller than 2.

For simple calculus we use the signs:

- + / * > < >= == **!= exp() log() sqrt() ^**

>x<-1:10
>x1<-10:1
>x-x1
[1] -9 -7 -5 -3 -1 1 3 5 7 9

We can edit a vector by the use of "identification brackets" [ ]
The value within the bracket indicates the number in the vector sequence, i.e. the rank.
>x
[1] 1 2 3 4 5 6 7 8 9 10

The value placed in position 3 along the sequence should be converted to 60

> x[3]<-60
> x
[1] 1 2 60 4 5 6 7 8 9 10

We can also use the [ ] to identify a subset of the vector.
Extract the 3, 4, 5, and 6^{th} number of the vector.

> x[c(3,4,5,6)]
[1] 60 4 5 6

Notice that within [ ] we define a vector that identifies the numbers that should be included in the subset.

2) Matrices
Matrices consist of n rows and m columns.

It is important to notice that in both R and Splus we always identify the row first and then the columns, as mathematical conventions.
A matrix of 10 rows and 20 columns is described as x[10,20]

As for vectors numerous paths can generate a matrix.

>x<-1:10
>x1<-11:20
Two vectors of similar length

From a vector x we can generate a matrix of 2 columns.

>x2<- matrix(x,ncol=2)
>x2
```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

> matrix(x,ncol=2,byrow=T)
```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
```

> matrix(x,nrow=2,byrow=T)
```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

A 2X2 matrix of only zeros.
> matrix(0,2,2)
```
     [,1] [,2]
[1,]    0    0
[2,]    0    0
```

Further we can combine 2 vectors into a matrix by the functions cbind() and rbind(), read column-bind and row.bind, respectively.

> cbind(x,x1)
```
      x x1
 [1,] 1 11
 [2,] 2 12
 [3,] 3 13
 [4,] 4 14
 [5,] 5 15
```

```
 [6,]  6 16
 [7,]  7 17
 [8,]  8 18
 [9,]  9 19
[10,] 10 20
```

> rbind(x,x1)

```
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 x    1    2    3    4    5    6    7    8    9    10
x1   11   12   13   14   15   16   17   18   19    20
```

For transposing a matrix we use t()

>t(rbind(x,x1))

```
      x x1
 [1,]  1 11
 [2,]  2 12
 [3,]  3 13
 [4,]  4 14
 [5,]  5 15
 [6,]  6 16
 [7,]  7 17
 [8,]  8 18
 [9,]  9 19
[10,] 10 20
```

From a matrix we can obtain a subset, of important it is to remember first rows then columns.

>x2<-cbind(x,x1)
>x2

```
      x x1
 [1,]  1 11
 [2,]  2 12
 [3,]  3 13
 [4,]  4 14
 [5,]  5 15
 [6,]  6 16
 [7,]  7 17
 [8,]  8 18
 [9,]  9 19
[10,] 10 20
```

Extract first row
> x2[1,]

```
 x x1
 1 11
```

Extract row number 4
> x2[4,]

```
 x x1
```

4 14

Extract column 1
> x2[,1]
[1] 1 2 3 4 5 6 7 8 9 10

Extract column 2
> x2[,2]
[1] 11 12 13 14 15 16 17 18 19 20

Extract row 3 and column 2
> x2[3,2]
x1
13

Extract rows 2 to 4 and both columns
> x2[2:4,]
x x1
[1,] 2 12
[2,] 3 13
[3,] 4 14

Extract row 2 to 4 and column 1
> x2[2:4,1]
[1] 2 3 4

Extract rows 2, 4 and 9 in column 1.
> x2[c(2,4,9),1]
[1] 2 4 9

We cab use the identification brackets to manipulate the data.

We see that the numbers in row 4 and column 2 is wrong it should be 60 instead of 14.
> x2[4,2]<-60
> x2
x x1
[1,] 1 11
[2,] 2 12
[3,] 3 13
[4,] 4 60
[5,] 5 15
[6,] 6 16
[7,] 7 17
[8,] 8 18
[9,] 9 19
[10,] 10 20

We can find all units within a matrix with a specific value and exchange these.

> x2[x2>5&x2<16]_1
> x2

```
      x x1
 [1,] 1  1
 [2,] 2  1
 [3,] 3  1
 [4,] 4 60
 [5,] 5  1
 [6,] 1 16
 [7,] 1 17
 [8,] 1 18
 [9,] 1 19
[10,] 1 20
```

Here we change the values between 5 and 16 with 1.

Linear algebraic operations can be performed in both R & Splus
>t(x2)%*%x2

```
    x   x1
 x  385 1119
x1 1119 5889
```

This gives the inner product.

3) Array

An array is a multidimensional matrix where each entity within a column and row contains a vector.
x3[1,2,3]
This identification bracket identifies a 3-dimensional matrix, or Array.

4) Data.frame

In a similar way as for matrices can we create data.frames.
>data.frame(x,x1)

>x2.df_data.frame(x2)

Here the name of the columns are ”x” and ”x1”
> names(x2.df)

```
[1] "x"  "x1"
```

The advantage with a data.frame is that we can add it to the searchpath as a data-base. This is done through the function attach().

>attach(x2.df)

Now the data.frame x2.df is attached to the searchpath in position 2, i.e. just after the workspace.
If we then want to call a specific column in the data.frame we just write its name.
> x

```
 [1]  1  2 60  4  5  6  7  8  9 10
```

The identification-brackets work in the same manner as for matrices.

>x2.df[3,2]
[1] 13

5) Liste
A list may contain any type of data-object at the same time.

>x3<-list(x2.df,x,x1)
This list contains a data.frame (x2.df) and 2 vectors (x & x1)

If we want a specific attachment to the list we use double identification-brackets:
>x3[[1]]

Extract row 3 and column 2 in the first attachment in the list:
> x3[[1]][3,2]
[1] 13
This is the same as x2.df[3,2] (see above).

Analyses of data in R & S-plus

Basic for most statistical analyses in R and Splus are the formula. This defines the relationship between the response variable and predictors to be used in the modelling.

The models has a basic syntax:
y ~ x

The left side is the response and the right side is the predictor.
Both left and right side can be modified.
The formula is read as y approximated by x.

In writing formulas we use the signs:
- + * ^ : / %in% I() poly() s() lo() bs() ns()

Only intercept
y~+1

Intercept + linear term of x
y~x

Intercept removed
y~x-1

Multiple predictor with the marginal terms for the variables x and x1
y~x+x1

Marginal terms + interaction between x and x1

y~x*x1

y~x+x1 = y~x*x1-x:x1

Here the sign **:** indicates only the interaction term.

Polynomial regression
y~x+I(x^2)

y~poly(x,2)

Both is of second order, the poly() function creates a orthonormal basis for the polynomial order. poly(variabel, n orden polynom).
poly(x,n) is more numerical stable that the use of I(), but the I() provides the term to be analysed in its original scale.

Nested
y~cut(x,2)/x1

A model for x1 in each class of x, x is cut at the mid-point.

y~cut(x,2)/x1 = y~cut(x,2) + x1 %in% cut(x,2)

Nested is used when a relationship between the response and a predictor only has a meaning within the categories of another variable.

Nested vs interaction
Say the vector forest and biomass, the vector forest is a factor (forest or no-forest), whilst biomass is continuous. Say the response is normal distributed.

Formula
y~skog/biomasse (nested)
and
y~skog*biomasse (crossed)

The nested formula is related to
$\mu + \alpha_S + \beta_1$ biomass
and
$\mu - \alpha_S + \beta_2$ biomass
μ intercept, α is contrast for forest vs. non-forest, and β is the slope within forest and outside forest modelled separately.

A crossed model ~skog*biomass represent
$\mu + \alpha_S + \beta$ biomass $+ \gamma$ biomass
and
$\mu - \alpha_S + \beta$ biomass $- \gamma$ biomass
There is used similar amount of degrees of freedom, and the models seem similar, but there are differences in the interpretation. In the crossed example we view the additional information obtained by allowing the to categorise having different slopes.

Other formulas frequently used in Splus.

Beta-spline
bs(x,knots)

, knots is number of knots the variable x is divided by.

Natrural spline
ns(x,df)
, df is reflecting the number of internal knots.

Cubic smooth spline
s(x,df)
, df represents the degree of smoothening allowed. A high number gives a rougher line.

Loess
lo(x,span,degree)
, span is the relative number of observations included within the neighbourhood, and degree represents the polynomial order within each neighbourhood.

Datastructures, correlation and simple plots

Create the variables x,x1 and y

```
>x<-1:100
>x1<-sample(x,replace=F)
```

This function resample vector x without replacement.

```
>y<-2+3*x+4*x1+rnorm(100,0,50)
```

y is a function of x and x1 with a normal distributed addition.

y = 2 + 3x + 4x + error

Now a simple task to start with:

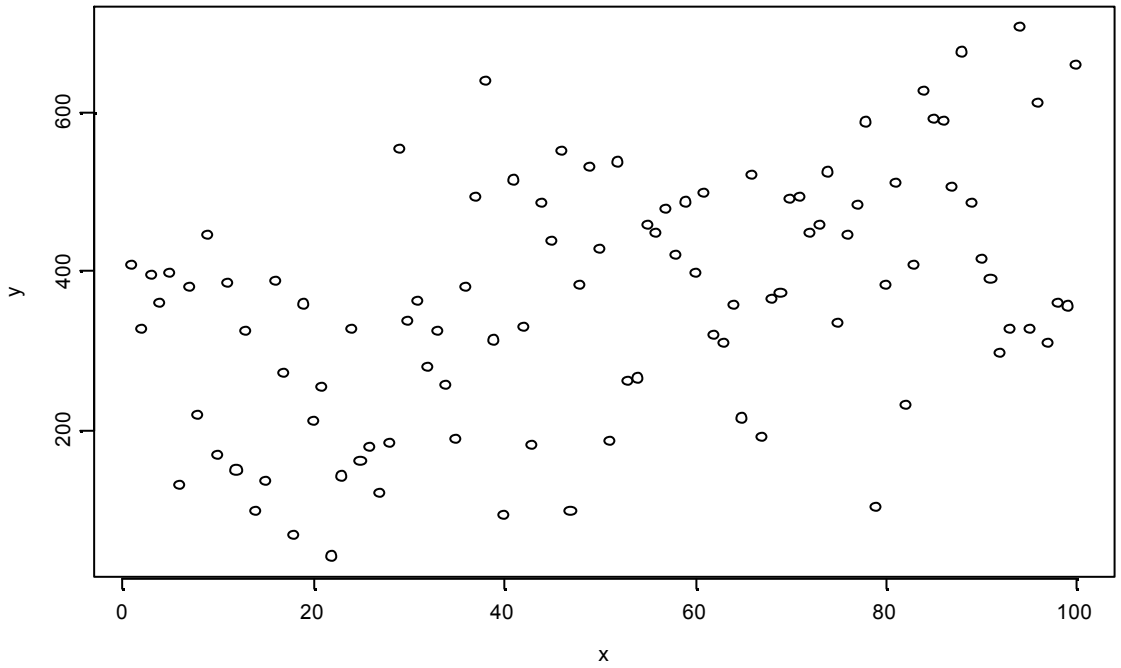
i) Create a matrix from the vectors (y,x,,x1).
ii) Create a data.frame from the vectors, name this data1.df, and implement this in the searchpath.
iii) Find the correlation (cor) between these variables.
iv) Create a scatter-plot.
v) Divide the graph window into 2 by 2 subplots, (hint use par, and mfrow within par)
vi) What does a scatter.plot function do?

> x<-1:100
> x1<-sample(x,replace=T)
> y<-2+3*x+4*x1+rnorm(100,0,50)
> x.ma<-cbind(y,x,x1)
> data1.df<-data.frame(x.ma)
> attach(data1.df.df)

To find the correlation we can search the help for the right command, or if we know the command name we can use the command:
>?cor
This will tell us how we should use the function and which arguments are required, and which are optional.
For a correlation matrix of the data.frame data1.df:
>cor(data1.df)

```
          y          x         x1
y 1.0000000  0.4562431  0.7537193
x 0.4562431  1.0000000 -0.1365726
x1 0.7537193 -0.1365726  1.0000000
```

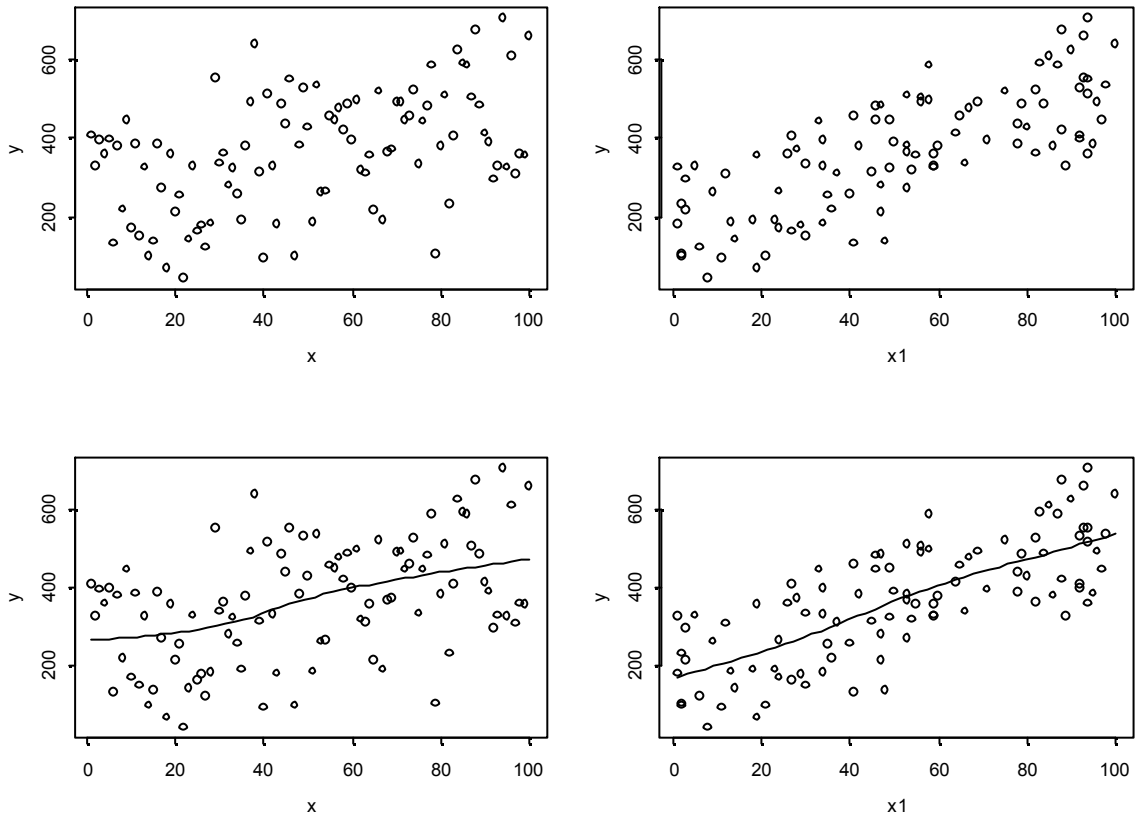
A scatterplot can be made by the function:
>plot(x,y)



To split the graphical window into subsections so we can plot several graphs within the same window we use the function par. par is and important function defining the apperence of the graph, and can take numerous arguments (se >?par for more info).

>par(mfrow=c(2,2))

Deler vinduet inn i 4 deler, 2 rows and 2 columns.



In the lower row we have used the scatter.smooth function which gives an additional LOESS regression line (local regression) to the ordinary scatterplot. This line can often be helpful evaluating the scatter.

>scatter.smooth(x,y)

<u>Test using correlation og rank correlation</u>

The correlation used previous (cor(.)) is a Pearson correlation, and to take this a step further we are now interested in testing its statistical significance, at the same time we are in doubt about the assumptions of correlations therefore we would like to perform a non-parametric or rank correlation.

>?cor.test for mer informasjon.

> cor.test(x,y)

Pearson's product-moment correlation

data: x and y
t = 5.0756, df = 98, p-value = 0

alternative hypothesis: true coef is not equal to 0
sample estimates:
      cor
 0.4562431

This test show that the correlation between x and y is statistically significant, i.e. we may reject the null hypothesis; the correlation between the two variables are zero.

For a non-parametric correlation we chose Spearman rank-correlation.
This can be estimated through numerous ways within S & R:

> cor(rank(x),rank(y))
[1] 0.4288869

Here we see that the spearman rank correlation is a correlation between the rank of x and rank of y. However, simpler we can use the cor.test function to do the same thing, but with additional information:

> cor.test(x,y,method="s")

        Spearman's rank correlation

data: x and y
normal-z = 4.2673, p-value = 0
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
 0.4288869

We used an additional argument within the function that described which method of correlation we should apply, method="s" is the same as method="spearman". The default procedure is a Pearson correlation.

t-test og wilcoxon

Are there a difference between two populations or are there a difference between a population and a specified value.
One-sample t.test gives the difference between a population and a specified value, and two-sample t.test gives the difference between two populations.

First we create a sample; y is the observations and z describes to which population each observatoin belong. to. There are 50 observation of population A and 50 observations of population B.

>z<-rep(c("A","B"),rep(50,2))
In this case we could also have written A=0 and B=1
>z<-rep(c(0,1),rep(50,2))

We generate a new variable y that is normal distributed within both A and B.

>y<-c(rnorm(50,18,5),rnorm(50,15,5))

We are interested in finding whether A and B are different with respect to the variable y. (Here z: A = 0 & B = 1.)

> t.test(y[z==0],y[z==1])

```
        Standard Two-Sample t-Test

data:  y[z == 0] and y[z == 1]
t = 2.523, df = 98, p-value = 0.0132
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.5494216 4.5989398
sample estimates:
 mean of x mean of y
  17.47734  14.90316
```

The two population A and B are different by alpha = 0.05, but not by alpha = 0.01.
Note that within the t.test(...) function create two vectors by splitting y into A population and B population. We use then == and not =.

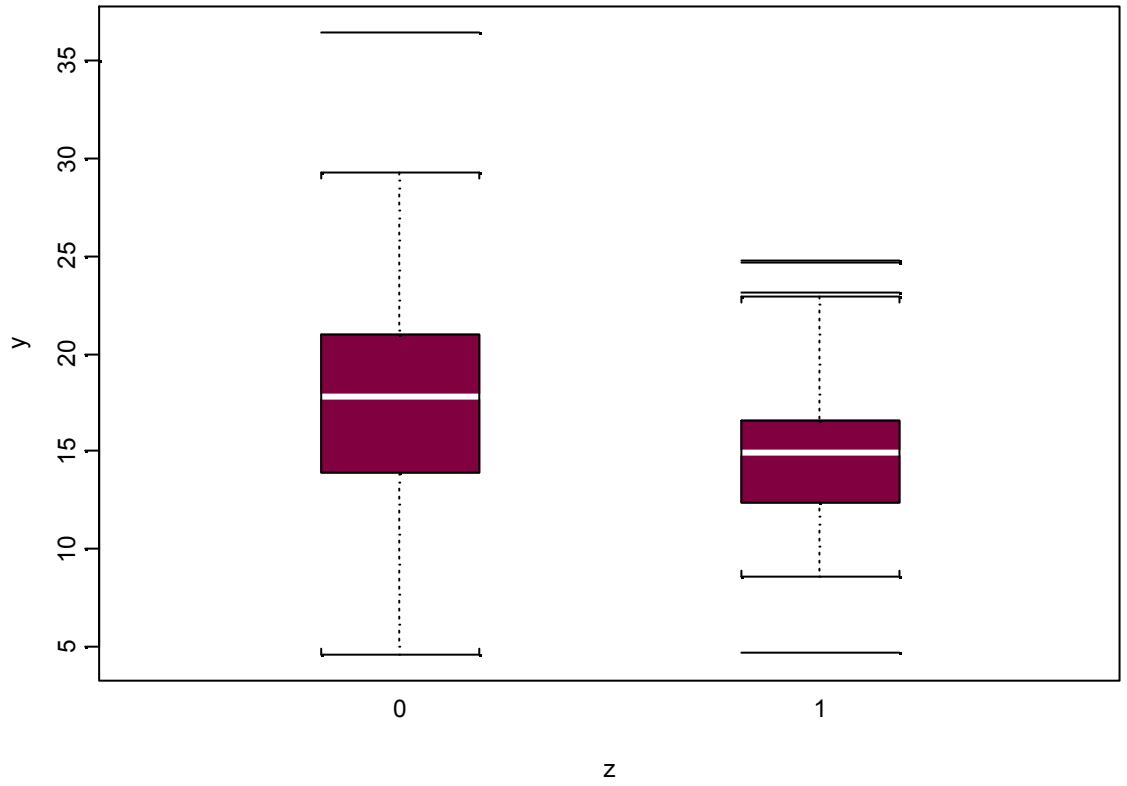
A wilcox.test takes similar arguments as the t.test (see ?wilcox.test)

> wilcox.test(y[z==0],y[z==1])

```
        Wilcoxon rank-sum test

data:  y[z == 0] and y[z == 1]
rank-sum normal statistic with correction Z = 2.6507, p-value = 0.008
alternative hypothesis: true mu is not equal to 0
```

We see that the p-value has been reduced by the wilcoxon test.

The to create a plot of the two populations we first create a factor variable of the z.
>z<-factor(z)
Subsequently we use the familiar plot function to create a boxplot. The plot function is generic and uses information stored about the object to create a suitable plot.
>plot(z,y)



Anova

Analysis of variance is a type of linear model that is frequently used in biological contexts. Within S & R there are two functions that can be used to perform an analysis of variance, i.e. lm and aov.
The general arguments for both these functions are a formula, which defines the relationship to be estimated between the response and the predictor variables. Here it is important that the predictor variable is defined as a factor. In addition, to the formula argument, several optional arguments may be included to specify our analysis. (see ?aov and ?lm).

A one-way anova can be performed by the function:

> fit.aov<-aov(y~z)
> summary(fit.aov)

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
z	1	165.660	165.6602	6.365306	0.01324638
Residuals	98	2550.497	26.0255		

This will give you the same analysis as the previous t.test, and as expected the p-values are identical.

>fit.lm<-lm(y~z)
>anova(fit.lm)
or
>summary(fit.lm)

will give similar result as aov (in general we recommend to use the lm function).

We create an additional factor which is crossed with z,

>z1<-factor(rep(0:1,50))

The following two-way anova:

> fit.aov<-aov(y~z*z1)
> summary(fit.aov)

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
z	1	165.660	165.6602	6.367392	0.0132661
z1	1	46.518	46.5181	1.787994	0.1843306
z:z1	1	6.351	6.3513	0.244121	0.6223740
Residuals	96	2497.628	26.0170		

Her we can see that the interaction between z1 and z is not significant.

If a splitplot design occur we can introduce an additional error-stratum for the split-level. This follows the classic multi-level anova approach.
Say 10 sample areas, and within each we have recorded 10 sample units, 5 of these have been treated and 5 is defined as a control.

>z2<-factor(rep(rep(1:10,rep(5,10)),2))

The factor z2 gives indicates which sample area each observation belongs to.

Now we have two layers of variance, between sample area and within sample area. We are interested in the effect of the treatment and we have to account for potential differences given by the sample area level. This is sorted within the aov function by addition an additional error layer, i.e. +Error(z2)

> fit.aov<-aov(y~z+Error(z2))
> summary(fit.aov)

Error: z2

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
Residuals	1	7.276824	7.276824		

Error: Within

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
z	1	165.66	165.6602	6.318381	0.01359634
Residuals	97	2543.22	26.2188		

Note that the degrees of freedom has changed in relation to a model were we did not make account for the split-plot design. Further we can introduce the design as a factor to the analysis to check if there is a change in the treatment response to between the different areas.

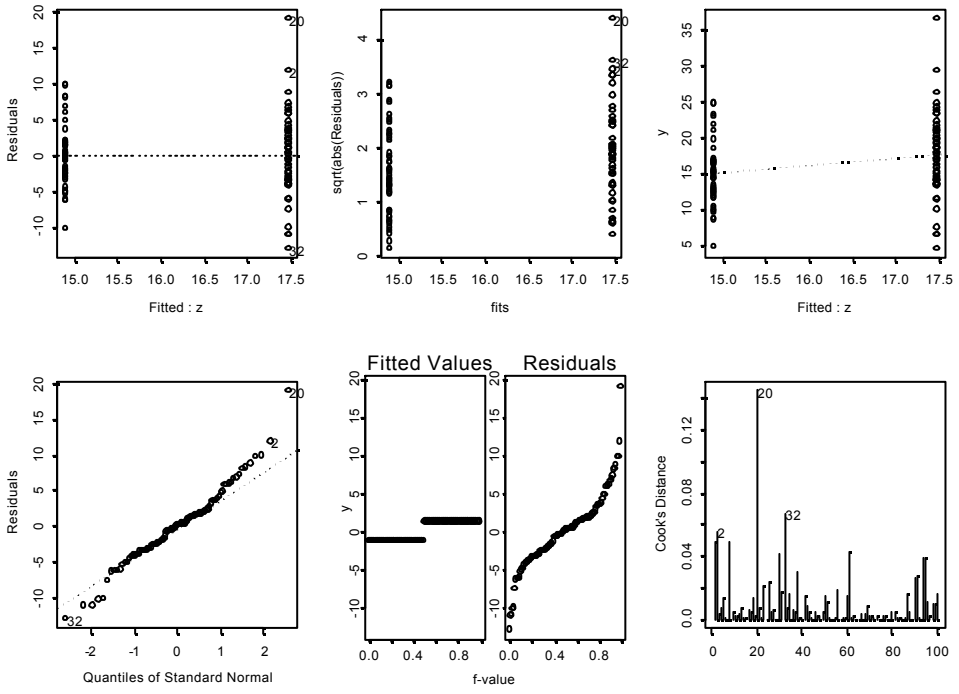
> fit.aov<-aov(y~z*z2+Error(z2))
> summary(fit.aov)
Error: z2
  Df Sum of Sq Mean Sq
z2 1 7.276824 7.276824

Error: Within
        Df Sum of Sq Mean Sq F Value   Pr(F)
      z 1   165.660 165.6602 6.258225 0.0140539
    z:z2 1   2.025   2.0248 0.076492 0.7827037
Residuals 96 2541.196 26.4708

We see the main effect of z2 is at the split-plot level, but the interaction is within split-plot level. The interaction was not significant and we conclude that between the areas there where no differences in how the treatment affected the response.

After an analysis we need to evaluate the assumptions of the model. This can be made through a diagnostic plot. Here we use the generic function plot, and include the modelled object as an argument.

>fit.aov<-aov(y~z)
>par(mfrow=c(2,3))
We use par function because the plot output will give 6 graphs, and I would like to see them on the same page.
>plot(fit.aov)



We see that the variance of the residuals are reasonable similar for the two groups (control and treatment). We also see that the model is reasonably normally distributed. The last plot indicate that the observation number 20 may be an outlier (the 3 most influentil points will always be displayed, but they do not have to be outliers).

Linear mixed models

A continuation of the linear model (and anova), is the linear mixed effect models. Within these models we account for variables being either fixed or random, i.e. contributing fixed effects or random effects. A random variable is a variable generated by a sampling from a greater population of possibilities, whereas a fixed variable is a specified categorisation.
If we have som bushes and for each bush we would like to know something about the effect of being under the canopy and outside the canopy. We select 50 bushes at random from a greater population of bushes, say 50 of all 1000 bushes within an area. For each bush we make 1 observation under the canopy and 1 observations outside the canopy. In this example we can say that the bush is a random factor, and the canopy represent a fixed factor (either it is under the canopy or it is outside).

Let y be the response as earlier, z is the canopy (under the canopy; z=0, and outside the canopy; z = 1), and z2 represent the bushes. z is a fixed factor, and z2 is a random factor.

For a linear mixed effect we need to use the function lme (linear mixed model). This is a very general procedure and procedures like repeated measurement can be solved by using the lme function.

>?lme for mer informasjon.

```
>fit.lme<-lme(y~z*z2,random=~+1|z2)
> summary(fit.lme)
Linear mixed-effects model fit by REML
 Data: NULL
      AIC      BIC    logLik
  591.4999 601.8398 -291.7500

Random effects:
 Formula: ~+1 | z2
       (Intercept) Residual
StdDev:   0.8435942 4.486638

Fixed effects: y ~ z
               Value Std.Error DF    t-value p-value
(Intercept) 17.772028 0.6456249 49 27.526862  <.0001
z           -2.834783 0.8973277 49 -3.159139  0.0027
 Correlation:
  (Intr)
z -0.695
```

Standardized Within-Group Residuals:
```
       Min         Q1        Med         Q3        Max
-2.28918824 -0.60005497 -0.03246693  0.67683978  2.37667063
```

Number of Observations: 100
Number of Groups: 50

Following the output we can divide it into two sections 1) the random effect, and 2) the fixed effect. The fixed effect describes the effect of canopy. However, when testing the fixed effect it is recommended to use the anova function (below).
The random effect describes the standard deviation of the distribution from which the bushes has been selected (intercept), and the standard deviation of the distribution from which the individual observation has been selected.

> anova(fit.lme)

	numDF	**denDF**	**F-value**	**p-value**
(Intercept)	**1**	**49**	**1240.9934**	**<.0001**
z	**1**	**49**	**9.9802**	**0.0027**

Here we see that the main effect of canopy is statistically significant, and from the summary(fit.lme) we see a negative coefficient, which indicates that the effect of being outside the canopy is negative.

Regression

x & x1 are continuous predictor variables observed n = 100.
y is the response variable n = 100.

A regression can be estimated through various ways, such as:

>x.ma<-matrix(rep(1,100),x,x1)
x.ma is a matrix with the first column of the value 1, and the second and third column are te variables x and x1.

The regression coefficients can then be estimated by linear algebra:

$(X^T X)^{-1}(X^T Y) = \beta$

Which in S and R is written as:

> solve(t(x.ma)%*%x.ma)%*%(t(x.ma)%*%y)
```
            [,1]
[1,]  17.8163706249
[2,]  -0.0330354109
[3,]   0.0008166158
```

But much simpler is using the lm function for a least square estimation

>fit.lm_lm(y~x+x1)
> fit.lm
Call:
lm(formula = y ~ x + x1)

Coefficients:
(Intercept) x x1
17.81637 -0.03303541 0.0008166158

Degrees of freedom: 100 total; 97 residual
Residual standard error: 5.201635

To check the significance we can use the anova function (anova function only creates an analysis of table).

> anova(fit.lm)
Analysis of Variance Table

Response: y

Terms added sequentially (first to last)

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
x	1	91.570	91.56997	3.384335	0.0688771
x1	1	0.058	0.05798	0.002143	0.9631745
Residuals	97	2624.529	27.05700		

Here we see that x1 is not significant in the presence of x, and that x is on the border of significance in the presence of the variable x1.

> summary(fit.lm)

Call: lm(formula = y ~ x + x1)
Residuals:
Min 1Q Median 3Q Max
-12.27 -3.119 -0.01845 2.555 19.35

Coefficients:

	Value	Std. Error	t value	Pr(>|t|)
(Intercept)	17.8164	1.4741	12.0864	0.0000
x	-0.0330	0.0182	-1.8161	0.0724
x1	0.0008	0.0176	0.0463	0.9632

Residual standard error: 5.202 on 97 degrees of freedom
Multiple R-Squared: 0.03373
F-statistic: 1.693 on 2 and 97 degrees of freedom, the p-value is 0.1893

Correlation of Coefficients:
(Intercept) x
x -0.7076
x1 -0.7031 0.1366

As usual the summary gives us additional information.

We create a new response variable were we include the interaction term between x and x1.

>y<-2+(0.6*x)-(0.5*x1)+0.2*(x*x1)+rnorm(100,0,500)

We would now use a stepwise procedure to select the ”best-fit” model, we can then apply the step function (see ?step).

First a model is defined by our variables, and subsequently we call upon the step function to do a stepwise selection of our variables. Within the step function I have created a list which defines the upper and lower complexity of the possible models to chose from.

>fit.lm<-lm(y~x+x1)

>fit1.lm<-step(fit.lm,list(upper=~.^2,lower=~+1))

```
Start:  AIC= 3582347
 y ~ x + x1

Single term deletions

Model:
y ~ x + x1

scale:  34780.07

       Df Sum of Sq      RSS       Cp
<none>              3373666  3582347
     x  1  10727768 14101434 14240554
    x1  1   8406874 11780540 11919661
Single term additions

Model:
y ~ x + x1

scale:  34780.07

       Df Sum of Sq     RSS      Cp
<none>              3373666 3582347
  x:x1  1   3125873  247794  526034

Step:  AIC= 526034.1
 y ~ x + x1 + x:x1

Single term deletions

Model:
y ~ x + x1 + x:x1
```

scale: 34780.07

	Df	Sum of Sq	RSS	Cp
<none>			247794	526034
x:x1	1	3125873	3373666	3582347

In this section we have used Mallow's Cp (AIC) as a criteria for selection. The Mallow's Cp is a selection criteria which uses the sum of square residuals and then add a term for the model complexity, i.e. degrees of freedom used by the particular model. The model with lowest Cp is giving the best fit. By this stepwise procedure we see that we need to include the interaction term.

We now want to create a three dimensional figure of the relationship.
First define for which values of x and x1 the plot should include.

>fit.mar<-list(x=seq(min(x),max(x),len=20),x1=seq(min(x1),max(x1),len=20))

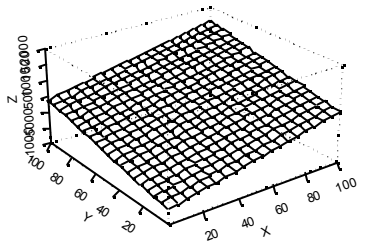
Then we creat som predictions based on the model

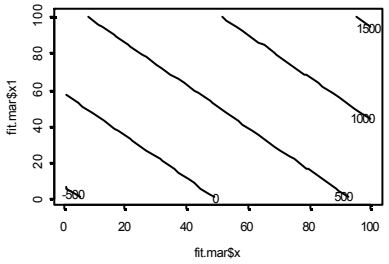
>fit.grid<-predict.gam(fit.lm,expand.grid(fit.mar),type="response")

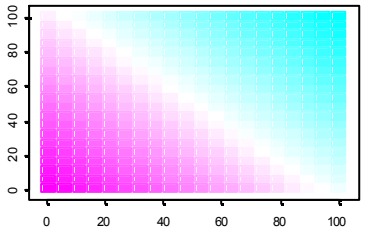
expand.grid is used to find all combinations of x and x1, as defined by fit.mar.

Finally we can make a perspective plot, contour plot, or image plot.

>persp(fit.mar$x,fit.mar$x1,matrix(fit.grid,20))
>contour(fit.mar$x,fit.mar$x1,matrix(fit.grid,20))
>image(fit.mar$x,fit.mar$x1,matrix(fit.grid,20))







Generalized Linear Models (GLM) & Generalized Additive Models (GAM)

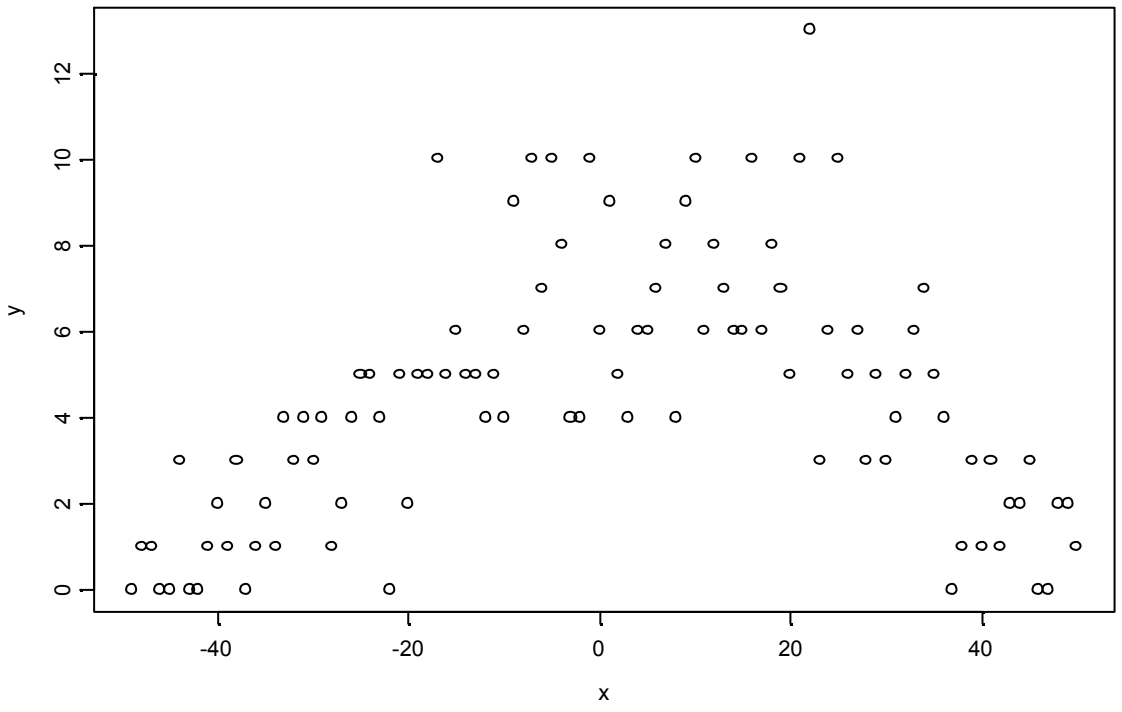
These are model frames that expands the classic linear model procedure. However, instead of using least squares we use maximum likelihood as a maximization criteria for estimating the coefficients of our models. Then we chose the model with the coefficients that gives the highest probability of having generated our observations. By this expansion we may use other distributions than the normal distribution, and we can handle problems like variable variance.

Through the functions glm and gam we can include logarithmic regression and logistic regressions, both of which are important in many biological contexts.

We count the number of species (y) within sites and we would like to predict the average number of species as a function of some predictors. In such cases we can not assume a normal distribution as the variable consist of discrete numbers, and we would normally encounter that the variance increases with the mean, and we can not accept negative predictions.

This can here be solved by assuming a Poisson distribution, and a logarithmic link (transformation of the mean response to the scale of the linear predictor (the right-hand side of a regression equation).

We create the following Poisson distributed variable
>ymean<-2+0.01*x-0.001*x^2
>y<-rpois(100,ymean)
The last creta our observations from a predefined mean response (ymean). A plot of the relationship gives:
>plot(x,y)



We use the glm function which takes the arguments formula, but also an argument of the family. We make 4 models, a null model, and up to third order polynomial of x.

```
> fit.glm<-glm(y~+1,poisson)
> fit1.glm<-glm(y~x,poisson)
> fit2.glm<-glm(y~poly(x,2),poisson)
> fit3.glm<-glm(y~poly(x,3),poisson)
```

The 4 models have increasing complexity, and we would like to see which model gives the "best-fit". Now we can not use the sum of squares anymore, so we change into Deviance. The deviance is likelihood related equivalent to sum of squares, and for normal distirbutions these will be similar.

We chose to use the F-test as a selection criteria, and then starts by comparing the linear termed model against the null model.

```
> anova(fit.glm,fit1.glm,test="F")
Analysis of Deviance Table

Response: y

  Terms Resid. Df Resid. Dev Test Df Deviance  F Value      Pr(F)
1     1       99   236.0295
2     x       98   227.8170       1 8.212511 4.132794 0.04476432
```

The linear term is significant, then we continue to test the second order polynomial model against the linear termed model.

```
> anova(fit1.glm,fit2.glm,test="F")
Analysis of Deviance Table

Response: y

       Terms Resid. Df Resid. Dev     Test Df Deviance  F Value Pr(F)
1          x       98    227.817
2 poly(x, 2)       97     94.885 1 vs. 2  1  132.932 154.0968     0
```

This was also significant, and finally is a higher order polynomial needed?

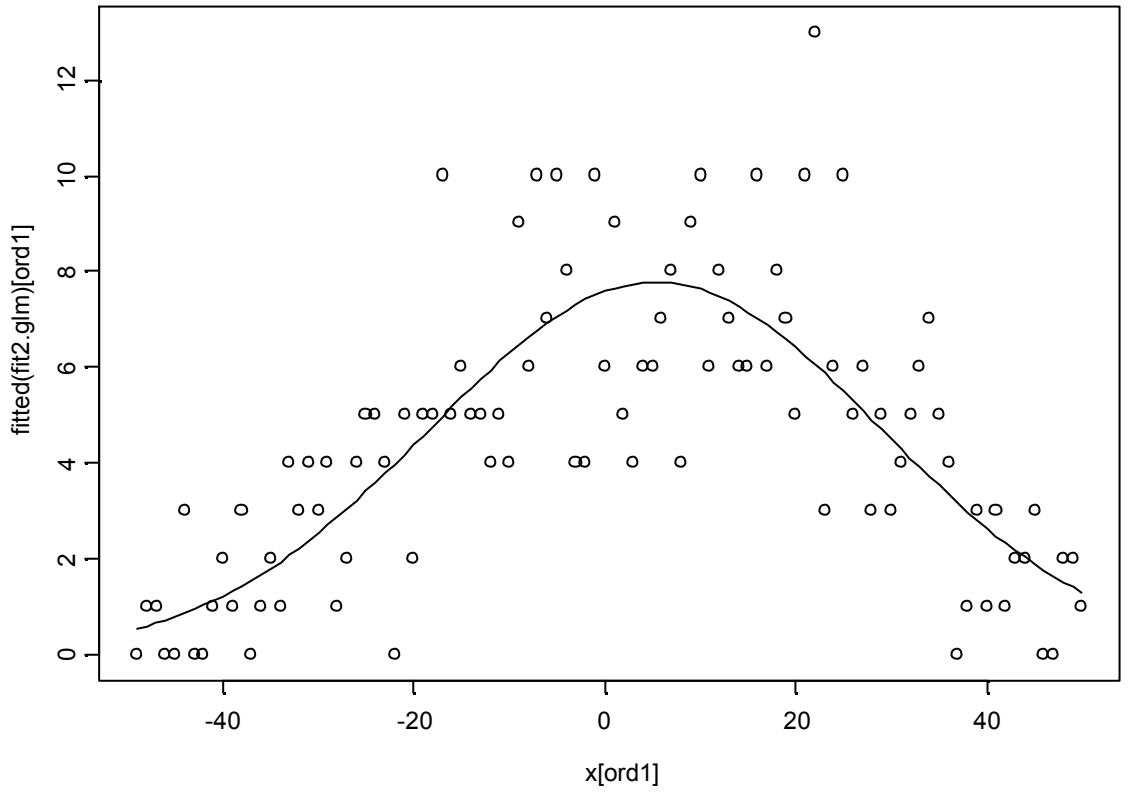
```
> anova(fit2.glm,fit3.glm,test="F")
Analysis of Deviance Table

Response: y

       Terms Resid. Df Resid. Dev     Test Df  Deviance    F Value     Pr(F)
1 poly(x, 2)       97   94.88501
2 poly(x, 3)       96   94.32211 1 vs. 2  1 0.5628992 0.6563739 0.4198468
```

Here we see that the additional third order polynomial term is not significant and then we stop by chosing the model with the second order polynomial term included.

Diagnostic plots can be generated as usual. Note that we can include the argument smooth=T, which will create a helping line for interpretation of the diagnostic plots. The smooth=T is a loess regression.

>plot(fit2.glm,smooth=T)

A graphical output of the chosen relationship can be produced by.
>ord1<-order(x)
>plot(x[ord1],fitted(fit2.glm)[ord1],type="l",ylim=range(y))
>points(x,y)



The summary gives extended information about the relationship.
>summary(fit2.glm)

Call: glm(formula = y ~ poly(x, 2), family = poisson)
Deviance Residuals:
Min 1Q Median 3Q Max
-2.821427 -0.748979 -0.00009974771 0.4697232 2.439547

Coefficients:
Value Std. Error t value
(Intercept) 1.283369 0.05934877 21.624192
poly(x, 2)1 2.505058 0.66238168 3.781895

poly(x, 2)2 -6.682112 0.65809145 -10.153774

(Dispersion Parameter for Poisson family taken to be 1)

Null Deviance: 236.0295 on 99 degrees of freedom

Residual Deviance: 94.88501 on 97 degrees of freedom

Number of Fisher Scoring Iterations: 4

Correlation of Coefficients:
(Intercept) poly(x, 2)1
poly(x, 2)1 -0.2700619
poly(x, 2)2 0.5806944 -0.2158529

Note that the coefficients is on a form of the linear predictor, and to obtain the response values, we need to use the inverse logarithmic transformation (exp). Further, the coefficients is related to the orthonormalised vectors of x, x^2. Hence, we can not use the coefficients directly to our original predictor variables. But by the fitted function and the predict function we can produce the fitted values needed.

To obtain the second order polynomial coefficients on the original scale of our predictor variables we have to use the function I(.) which conserve the arguments within.

>fit.glm<-glm(y~x+I(x^2),poisson)

In the above model we have defined the relationship between y and x as parametric, sometimes there are elements that disfavours such decisions. An option is then to use a non-parametric regression. In S this can made done by using the gam function. This function can also be included in R by opening the library mgcv.

Here we use a cubic smooth spline. This is a regression with local properties (see ?gam). The function s(.) defines the cubic smooth spline process, and the predictor variable is needed together with a parameter describing how rough the estimated relationship should be allowed to be. In this context we often use the degrees of freedom, as higher degrees of freedom within the smooth term will give a rougher estimate.

> fit.gam_gam(y~+1,poisson)
> fit1.gam_gam(y~s(x,1),poisson)
> fit2.gam_gam(y~s(x,2),poisson)
> fit3.gam_gam(y~s(x,3),poisson)
> fit4.gam_gam(y~s(x,4),poisson)
> fit5.gam_gam(y~s(x,5),poisson)

These models can be compared in the same manner as for GLM.

>anova(fit.gam,fit1.gam,test="F")
Analysis of Deviance Table

Response: y

	Terms	Resid. Df	Resid. Dev	Test	Df Deviance	F Value	Pr(F)
1	1	99.00000	236.0295				
2	s(x, 1)	97.99926	227.6749		1.000736	8.354576	4.204176 0.04296644

> anova(fit1.gam,fit2.gam,test="F")
Analysis of Deviance Table

Response: y

	Terms	Resid. Df	Resid. Dev	Test	Df Deviance	F Value	Pr(F)
1	s(x, 1)	97.99926	227.6749				
2	s(x, 2)	96.99763	113.4570	1 vs. 2	1.001639	114.2179	119.7873 0

> anova(fit2.gam,fit3.gam,test="F")
Analysis of Deviance Table

Response: y

	Terms	Resid. Df	Resid. Dev	Test	Df Deviance	F Value	Pr(F)
1	s(x, 2)	96.99763	113.4570				
2	s(x, 3)	96.00962	96.4159	1 vs. 2	0.9880045	17.04106	20.39784 0.00001938935

> anova(fit3.gam,fit4.gam,test="F")
Analysis of Deviance Table

Response: y

	Terms	Resid. Df	Resid. Dev	Test	Df Deviance	F Value	Pr(F)
1	s(x, 3)	96.00962	96.41590				
2	s(x, 4)	94.97351	92.20049	1 vs. 2	1.036114	4.215408	4.899929 0.0280607

> anova(fit4.gam,fit5.gam,test="F")
Analysis of Deviance Table

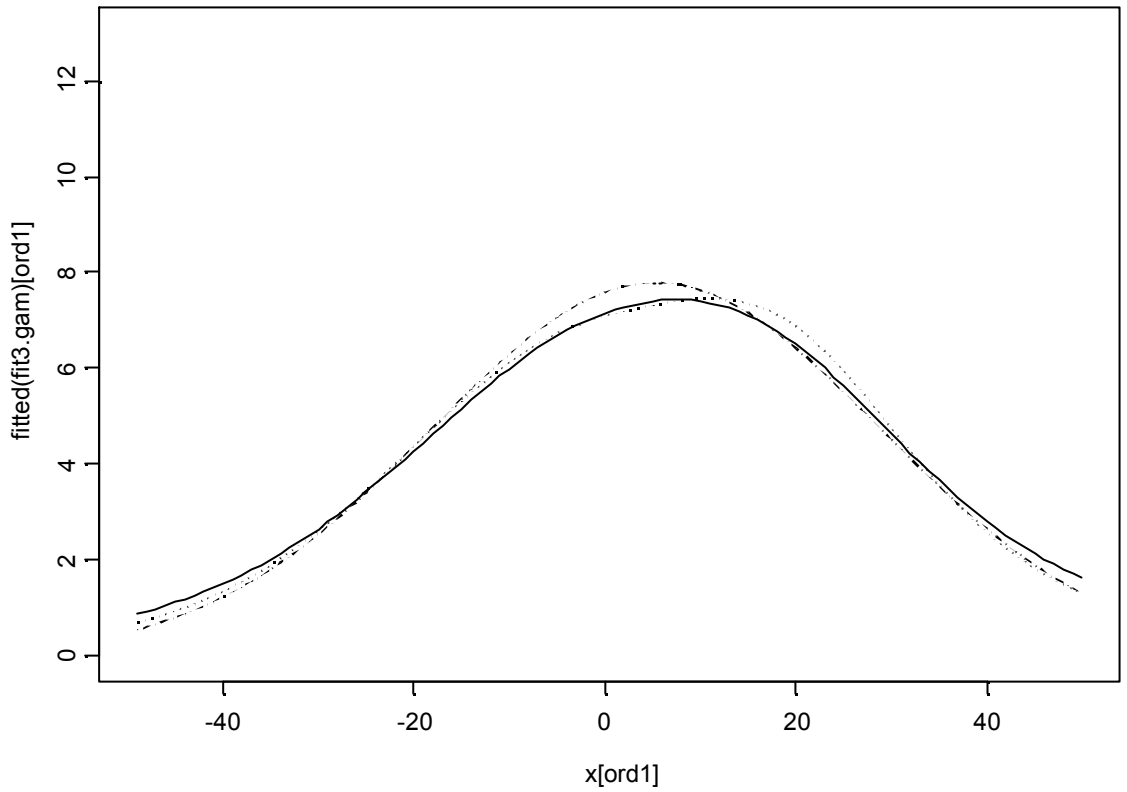
Response: y

	Terms	Resid. Df	Resid. Dev	Test	Df Deviance	F Value	Pr(F)
1	s(x, 4)	94.97351	92.20049				
2	s(x, 5)	93.99493	90.42933	1 vs. 2	0.9785779	1.771162	2.182777 0.1432849

By this procedure we see that model fit4.gam is significant.

The GAM relationships can be plotted in the same way as for GLM.

>plot(x[ord1],fitted(fit3.gam)[ord1],type="l",ylim=range(y))
>points(x[ord1],fitted(fit4.gam)[ord1],type="l",lty=2)
>points(x[ord1],fitted(fit2.glm)[ord1],type="l",lty=3)



Bold line is s(x,3), dotted line is s(x,4), and broken line is poly(x,2). This suggest that the additional information for this particular dataset by a non-papramteric regression is not considerable.

A summary for the gam models can be found by the usual functions:
>summary(fit3.gam)

Call: gam(formula = y ~ s(x, 3), family = poisson)
Deviance Residuals:
Min 1Q Median 3Q Max
-2.799108 -0.7580331 0.001008193 0.4381242 2.384706

(Dispersion Parameter for Poisson family taken to be 1)

Null Deviance: 236.0295 on 99 degrees of freedom

Residual Deviance: 96.4159 on 96.00962 degrees of freedom

Number of Local Scoring Iterations: 4

DF for Terms and Chi-squares for Nonparametric Effects

Df Npar Df Npar Chisq P(Chi)
(Intercept) 1
s(x, 3) 1 2 112.5662 0

Making your own functions

An important feature of both S and R is that it allows us to creat our own functions, libraries, etc.
This can be handy when a model procedure is going to be repeated several times.

A function can be made by the following syntax:

```
> radmiddel.fun_function(x)
+ {x1_length(x[,1])
+ x2_NULL
+ for(i in 1:x1)
+ {x2[i]<-mean(x[i,])}
+ x2}
```

This function calculates the average for each row within a matrix or a data.frame.
Here we write a for loop to tell the software to use the operations for each row.
The average of each row will then be stored as sucessive numbers in the vaector x2.

Instead of using a for loop, which is slow process, the apply functions are recommended, see ?apply.

However, in S and R there are already functions for the row means.
>rowMeans()

This will do exactly the same as our function.

Now good luck with future S-ing and R-ing.