Using R for scientific computing

Karline Soetaert
Royal Institute of Sea Research (NIOZ)
Yerseke
The Netherlands
Januari 2012

Abstract

R (R Development Core Team 2008) is the open-source (read: free-of-charge and free to make derived work) version of the language S. It is best known as a software that performs statistical analysis and graphics. However, R is so much more: it is a high-level language in which one can perform complex calculations, implement new methods, and make high-quality figures.

R has high-level functions to operate on matrices, perform numerical integration, advanced statistics, ... which are easily triggered and which make it ideally suited for data-visualization, statistical analysis and mathematical modeling.

It is the aim of these lecture notes to make you acquainted with the R language. The lecture notes are based on a book (Soetaert and Herman 2009) about ecological modelling in which R is extensively used for developing, applying and visualizing simulation models.

There are many excellent sources for learning the R (or S) language. R comes with several manuals that can be consulted from the main R program (Help/Manuals). R-intro.pdf is a good start. Many other good introductions to R are available, some freely on the web, and accessible via the R web site (www.r-project.org). My favorite is the R introduction by Petra Kuhnert and Bill Venables (Kuhnert and Venables 2005), but beware: this "introduction" comprises more than 300 pages!

Keywords: Variables, Functions, Figures, Interpolation, Fitting, Roots, Ordinary differential equations, R.

1. Introduction

1.1. The R software

Installing R

R is downloadable from the following web site: `http://www.r-project.org/` Choose the precompiled binary distribution. On this website, you will also find useful documentation. To use R for the examples in this course, several packages need to be downloaded.

- **deSolve**. Performs integration. (Soetaert, Petzoldt, and Setzer 2009b)

- **rootSolve**. Finds the root of equations (Soetaert 2009).
- **scatterplot3d**. For 3-D graphics. (Ligges and Machler 2003)
- **seacarb**. Aquatic chemistry. (Proye, Gattuso, Epitalon, Gentili, Orr, and Soetaert 2007)
- **marelac**. Functions and constants from the marine and lacustrine sciences (Soetaert, Petzoldt, and Meysman 2009a).
- **marelacTeaching**. Data sets, used in this tutorial. (Soetaert and Meysman 2009).

If you run R within windows, downloading specific packages can best be done within the R program itself. Select menu item `packages / install packages`, choose a nearby site (e.g. France (Paris)) and select the package you need. If you install package **marelacTeaching** then all other packages will be automatically installed as well.

Other useful software

I prefer to run R from within the Tinn-R editor, which is available for Windows and can be downloaded from URL `http://sourceforge.net/projects/tinn-r` and `http://www.sciviews.org/Tinn-R`. This editor provides R sensitive syntax and help. Download the latest Tinn-R setup file and install it. From within the Tinn-R program, you launch R via the menu (`R/start preferred Rgui`).

1.2. Quick overview of R

R code is highly readable, once you realise that:

- `<-` is the assignment operator.
- everything starting with `#` is considered a comment.
- R is case-sensitive: `a` and `A` are two different objects.

1.3. Console versus scripts

There are two ways in which to work with R.

1. We can type commands into the R console window at the command prompt (>) and use R as a powerful scientific calculator.

   For instance, enter in the console window:

   ```
   >  pi*0.795^2 ; 25*6/sqrt(67) ; log(25)

   [1] 1.985565

   [1] 18.32542

   [1] 3.218876
   ```

Here `sqrt` and `log` are built-in functions in R; `pi` is a built-in constant; the semi-colon (;) is used to separate R-commands.

In the console window, the <UP> and <DOWN> arrow keys are used to navigate through previously typed sentences.

2. Alternatively, we can create R-scripts in an editor (e.g. Tinn-R) and save them in a file (`filename.R`) for later re-use. R-scripts are sequences of R-commands and expressions. These scripts should be submitted to R before they are executed. This can be done in several ways:

   - by typing, in the R-console window:

     ```
     > source ("filename.R")
     ```

   - by opening the file, copying the R script to the clipboard (ctrl-C) and pasting it (ctrl-V) into the R console window
   - If you do not use the tinn-R editor, the file is opened as an R script from within the R console. After selecting the script, and pressing the "send" button the statements are executed and the cursor moved to the next line.
   - If you do use the Tinn-R editor,  you can either submit the entire file (buttons 1,2), selected parts of the text (buttons 3,4), submit marked blocks (buttons 5,6) or line-by-line (last button).

Throughout these notes, the following convention is used:

```
> 3/2
```

denotes input to the console window (> is the prompt)

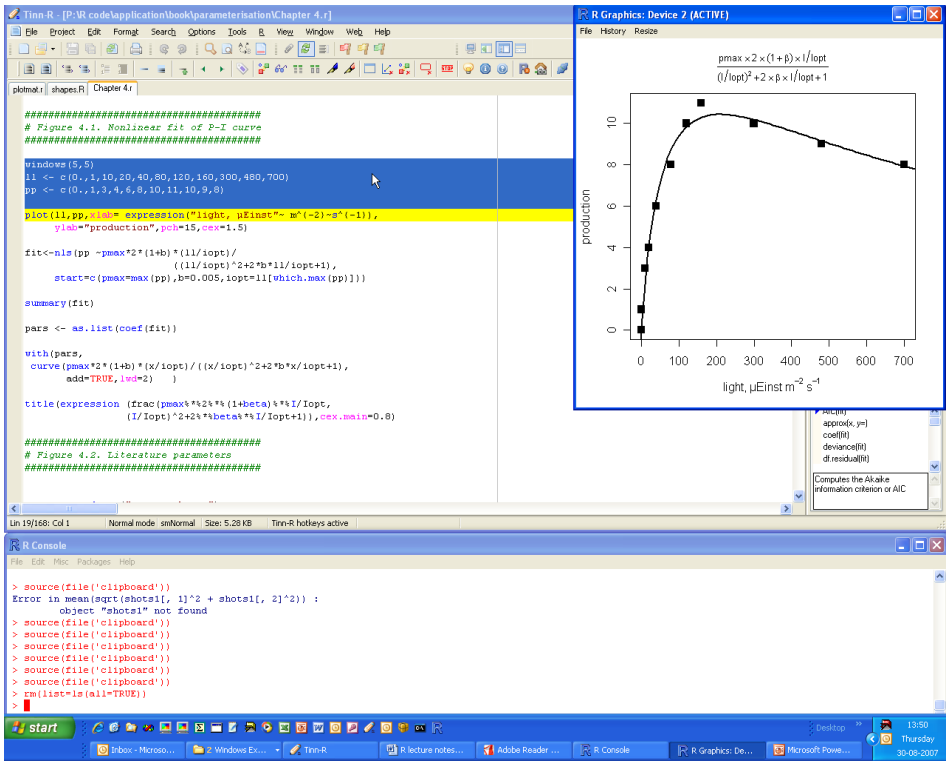
```
[1] 1
```

is R output, as written in the console window

```
getwd()
```

is an R statement in a script file (it lacks the prompt).

A screen capture of a typical Tinn-R session, with the Tinn-R editor (upper window) and the R console (lower window) is given below. A script file is opened in the Tinn-R editor. Note the context-sensitive syntax (green=comments, blue= reserved words, rose = R parameters). Several lines of R-code have been selected (blue area) and sent to the R-console, which has produced the graphics window that floats independently from the other windows.



Getting help, examples, demonstrations

R has an extensive help facility. Apart from the Help window launched from the Help menu, or the HTML help facility, it is also available from the command line prompt. For instance, typing

```
> ?log
> ?sin
> ?sqrt
> ?round
> ?Special
```

will explain about logarithms and exponential functions, trigonometric functions, and other functions.

```
> ?Arithmetic
```

lists the arithmetic operations in R.

```
> help.search("factor")
```

will list occurrences of the word <factor> in R-commands.

Sometimes the best help is provided by the very active mailing list. If you have a specific problem, just type `R: <problem>` on your search engine. Chances are that someone encountered the problem and it was already solved.

Most of the help files also include examples. You can run all of them by using R-statement `example`.

For instance, typing into the console window:

```
> example(matrix)
```

will run all the examples from the matrix help file.

```
> example(pairs)
```

will run all the examples from the pairs help file. (! try this ! `pairs` is a very powerful way of visualizing pair-wise relationships).

Alternatively, you may select one example, copy it to the clipboard (ctrl-C for windows users) and then paste it (ctrl-V) in the console window. In addition, the R main software and many R-packages come with demonstration material. Typing:

```
> demo()
```

will give a list of available demonstrations in the main software.

```
> demo(graphics)
```

will demonstrate some simple graphical capabilities.

Small things to remember

- Pathnames in R are written with forward slashes ("/") , although in windows, back-slashes, (\), are used. Thus, to set a working directory in R:

  ```
  setwd("C:/R code/")
  ```

- If a sentence on one line is syntaxically correct, R will execute it, even if it is the intention that it proceeds on the next line. For instance if we write:

  ```
  > A <- 3 + cos(pi)
  >     - sqrt(5)

  [1] -2.236068
  ```

  then A will get the value $(3+cos(\pi))$ and R will print the value of $(-\sqrt{(}5))$. In contrast, in the next lines:

  ```
  > 3 + cos(pi) -
  >     sqrt(5)
  [1] -0.236068
  ```

R will print the value of $3 + cos(\pi) - \sqrt{(}5)$: as the sentence on the first line was not syntaxically finished, R has (correctly) assumed that it continued on the next line.

Be careful if you want to split a complex statement over several lines ! These errors are very difficult to trace, so it is best to avoid them.

Exercises: Using R as a calculator

It is very convenient to use R as a powerful calculator. This can best be done from within the R console.

1. Use the console to calculate the value of:
   - $(4/6 * 8 - 1)^{2/3}$
   - $\ln(20)$
   - $log_2(4096)$
   - $2 * \pi * 3$
   - $\sqrt{2.3^2 + 5.4^2 - 2 * 2.3 * 5.4 * \cos(\pi/8)}$

   Tip: you may need to look at the help files for some of these functions. typing `?"+"` will open a help file with the common arithmetic operators.

2. Now write the R statements in a script file, using the Tinn-R editor. Try the various ways in which to submit the statements to R .

2. R variables

R calculates as easily with vectors, matrices and arrays as with single numbers.

R also includes more complex structures such as data frames and lists, which allow to combine several types of data.

Learning how to create these variables, how to address them and modify them is essential if you want the make good use of the R software.

2.1. Numbers, vectors, matrices and arrays

Assignment

When variables are used, they need to be initialised with numbers.

```
> A <- 1
> B <- 2
> A + B

[1] 3
```

R can take as arguments for its functions single numbers, vectors, matrices, or arrays.

```
> V <- factorial(10)
```

Calculates $10!(= 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot \ldots \cdot 10)$. The operator `<-` assigns the result of this calculation to variable V. V can then be used in subsequent calculations:

```
> V/10

[1] 362880
```

Note that the assignment of a value to V does not display it on the window. To display V we simply write:

```
>  V

[1] 3628800
```

Alternatively, we may assign the result of calculations to a variable AND view the results, by embracing the statement between brackets:

```
>  (X <- sin(3/2*pi) )

[1] -1
```

Apart from integers, real and complex numbers, R also recognizes infinity (`Inf`) and Not a Number (`NaN`). Try:

```
> 1/0
> 0/0
> 1e-8 * 1000
```

(where the `e-8` notation denotes 10^{-8}).

Vectors

Vectors can be created in many ways:

- Using R function `vector`,
- The function `c()` combines numbers into a vector [1],
- The operator ":" creates a sequence of values, each 1 larger (or smaller) than the previous one,
- A more general sequence can be generated by R function `seq`,
- The same quantity is repeated using R function `rep`.

For instance, the commands:

```
> c(0, pi/2, pi, 3*pi/2, 2*pi)

[1] 0.000000 1.570796 3.141593 4.712389 6.283185

> seq(from = 0,to = 2*pi, by = pi/2 )

[1] 0.000000 1.570796 3.141593 4.712389 6.283185

> seq(0, 2*pi, pi/2 )

[1] 0.000000 1.570796 3.141593 4.712389 6.283185
```

will all create a vector, consisting of: $0, \pi, \ldots 2 * \pi$.

Note that R function `seq` takes as input (amongst others) parameters `from`, `to` and `by` (2^{nd} example). If the order is kept, they not be specified by name (3^{rd} example).

The next command calculates the sine of this vector and outputs the result:

```
> sin( seq(0, 2*pi, pi/2 ))
```

[1] This is perhaps THE most important function in R

```
[1]  0.000000e+00  1.000000e+00  1.224647e-16 -1.000000e+00
[5] -2.449294e-16
```

Function `rep` is used to repeat elements:

```
> rep(1,times=5)

[1] 1 1 1 1 1

> rep(c(1,2),times=5)

 [1] 1 2 1 2 1 2 1 2 1 2

> c(rep(1,5),rep(2,5))

 [1] 1 1 1 1 1 2 2 2 2 2
```

The next statements:

```
> V <- 1:20
> sqrt(V)

 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
 [8] 2.828427 3.000000 3.162278 3.316625 3.464102 3.605551 3.741657
[15] 3.872983 4.000000 4.123106 4.242641 4.358899 4.472136
```

create a sequence of integers between 1 and 20 and take the square root of all of them, displaying the result to the screen. The operator `<-` assigns the sequence to V.

Some other examples of the `:` operator are:

```
> (V <- 0.5:10.5)

 [1]  0.5  1.5  2.5  3.5  4.5  5.5  6.5  7.5  8.5  9.5 10.5

> 6:1

[1] 6 5 4 3 2 1
```

Finally, the statements:

```
> V <- vector(length=10)
> FF<- vector()
```

generate a vector `V` comprising 10 elements, and a vector FF of unknown length.

Note: a peculiar feature of R is that the elements of a vector can also be given `names`:

```
> (fruit <- c(banana = 1, apple = 2, orange = 3))

banana  apple orange
     1      2      3

> names(fruit)

[1] "banana" "apple"  "orange"
```

Matrices

Matrices can also be created in several ways:

- By means of R function `matrix`,
- By means of R function `diag` which constructs a diagonal matrix,
- The functions `cbind` and `rbind` add columns and rows to an existing matrix, or to another vector.

The statement:

```
> A <-matrix(nrow = 2, data = c(1, 2, 3, 4))
```

creates a matrix A, with two rows, and, as there are four elements, two columns. Note that the data are inputted as a vector (using the `c()` function).
The next two statements display the matrix followed by the square root of its elements:

```
> A

     [,1] [,2]
[1,]    1    3
[2,]    2    4

> sqrt(A)

         [,1]     [,2]
[1,] 1.000000 1.732051
[2,] 1.414214 2.000000
```

By default, R fills a matrix column-wise (see the example above). However, this can easily be overruled, using parameter `byrow`:

```
> (M <- matrix(nrow = 4, ncol = 3, byrow = TRUE, data = 1:12))
```

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

The unity matrix (I) is created using R function diag:

```
> diag(1, nrow = 2)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

The *names* of columns and rows are set as follows:

```
> rownames(A) <- c("x", "y")
> colnames(A) <- c("c", "b")
> A
```

```
  c b
x 1 3
y 2 4
```

Note that we also use the `c()` function here! Row names and column names are in fact vectors containing strings.

Matrices can also be created by combining (binding) vectors, e.g. rowwise:

```
> V <- 0.5:5.5
> rbind(V,sqrt(V))
```

```
       [,1]     [,2]     [,3]     [,4]    [,5]     [,6]
V 0.5000000 1.500000 2.500000 3.500000 4.50000 5.500000
  0.7071068 1.224745 1.581139 1.870829 2.12132 2.345208
```

`t(A)` will transpose matrix A (interchange rows and columns).

```
> t(A)
```

```
  x y
c 1 2
b 3 4
```

Arrays

Arrays are multidimensional generalizations of matrices; matrices and arrays in R are actually vectors with a dimension attribute. A multi-dimensional array is created as follows:

```
> AR <- array(dim = c(2, 3, 2), data = 1)
```

In this case AR is a 2*3*2 array, and its elements are all 1.

2.2. Dimensions

The commands

```
> length(V)
> dim(A)
> ncol(M)
> nrow(M)
```

Will return the length (total number of elements) of (vector or matrix) V, the dimension of matrix or array A, and the number of columns and rows of matrix M respectively.

2.3. Selecting and extracting elements

To select subsets of vectors or matrices, we can either

- specify the numbers of the elements that we want (simple indexing),
- specify a vector of logical values (`TRUE`/`FALSE`) to indicate which elements to include (`TRUE`) and which not to include (`FALSE`). This uses logical expressions.

Simple indexing

The elements of vectors, matrices and arrays are indexed using the `[]` operator:

```
M[1 , 1]
M[1 , 1:2]
M[1:3 , c(2,4)]
```

Takes the element on the first row, first column of a matrix M (1^{st} line), then selects the entries in the first row and first two columns (2^{nd} line) and then the elements on the first three rows, and 2^{nd} and 4^{th} column of matrix M (3^{rd} line).

If an index is omitted, then all the rows (1^{st} index omitted) or columns (2^{nd} index omitted) are selected. In the following:

```
M[   ,2] <-0
M[1:3, ] <- M[1:3, ] * 2
```

first all the elements on the 2^{nd} column (1^{st} line) of M are zeroed and then the elements on the first three rows of M multiplied with 2 (2^{nd} line). Similar selection methods apply to vectors:

```
V[1:10]
V[seq(from=1,to=5,by=2)]
```

The statement on the 1^{st} line takes the first 10 elements of vector V, whilst on the 2^{nd} line, the 1^{st}, 3^{rd} and 5^{th} element of vector V are selected.

Logical expressions

Logical expressions are often used to select elements from vectors and matrices that obey certain criteria.

R distinguishes logical variables `TRUE` and `FALSE`, represented by the integers 1 and 0.

```
> ?Comparison
> ?Logic
```

will list the relational and logic operators available in R. The following will return `TRUE` for values of sequence V that are positive:

```
> (V <- seq(-2, 2, 0.5))

[1] -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0

> V > 0

[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

while

```
> V[V > 0]

[1] 0.5 1.0 1.5 2.0
```

will select the positive values from V,

```
> V[V > 0] <- 0
```

will zero all positive elements in V,

```
> sum(V < 0)

[1] 4
```

will return the number of negative elements: it sums the TRUE (=1) values, and

```
> V [V != 0]
```

```
[1] -2.0 -1.5 -1.0 -0.5
```

will display all nonzero elements from V ("!" is the "not" operator). Logical tests can also be combined, using | (the "or" operator), and & ("and").

```
> V[V < (-1) | V > 1]

[1] -2.0 -1.5
```

will display all values from V that are < -1 and > 1. Note that we have enclosed "-1" between brackets (can you see why this is necessary?) Finally,

```
> which (V == 0)

[1] 5 6 7 8 9

> which.min (V)

[1] 1
```

will return the element index of the 0-value, and of the minimum.

2.4. Removing elements

When the index is preceded by a "-", the element is removed.

```
> M[ ,-1]
```

will show the contents of matrix M, except the first column.

```
> x <- x[-1]
> M <- M[-1, ]
> V <- V[-V >= 0]
```

will delete the 1^{st} element of x, (1^{st} line), the 1^{st} row of M (2^{nd} line), and the positive elements of V (3^{rd} line). For more information, type:

```
> ?Extract
```

2.5. More complex data structures

R also allows creating more complex structures such as data frames and lists.

lists

A list is a combination of several objects; each object can be of different length:

```
> list(Array = AR, Matrix = M)
```

will combine the previously defined array AR and matrix M.

data.frames

These are combinations of different data types (e.g. characters, integers, logicals, reals), arranged in tabular format:

```
> genus <- c("Sabatieria", "Molgolaimus")
> dens  <- c(1,2)
> Nematode <-data.frame(genus = genus, density = dens)
> Nematode

        genus density
1  Sabatieria       1
2 Molgolaimus       2
```

In the example above, the data.frame `Nematode` contains two columns, one with strings (the genus name), one with values (the densities). Data.frames are in fact special cases of lists, consisting of vectors with equal length. Many matrix-operations work on data.frames with a single data type, but there exist also special operations on data.frames.

Selecting data from data.frames and lists

Data.frames and lists can be accessed by their names, or by the `[ ]` or `[[ ]]` operator. The object resulting from a selection using single brackets `[ ]`, will be a data.frame respectively a list itself; with double brackets `[[ ]]`, one obtains a vector (data.frames) or a variable data-type (lists).
For instance:

```
> Nematode$density/sum(Nematode$density)

[1] 0.3333333 0.6666667
```

will divide all density values (1,2) by the summed density.

```
> mean(Nematode[,2])

[1] 1.5
```

will calculate the mean of Nematode density (the 2^{nd} column).
Try:

```
> Nematode$genus
> Nematode[1]
> Nematode[[1]]
```

These statements will all output the two genus names, but in a different format.

```
> ?Extract
```

also explains the various ways in which to extract elements from lists and data frames.

2.6. Data Conversion

Conversion from one type of data structures to another can easily be done, e.g. by:

```
> as.data.frame(M)
> as.vector(A)
```

If unsure about the type, you can write:

```
> is.data.frame(M)
> is.vector(A)
```

Or you can display the data type by:

```
> class(M)
```

2.7. Data import from external sources

In all previous examples, data was entered from the console (or from a script file). There are ample facilities to import data from external sources. Most often, we will use functions `read.table`, `read.csv`, or `read.delim` to read matrices or data frames written in tabular form as text files. R also has plenty of built-in data sets. They are listed by:

```
> data()
```

2.8. Exercises

Creating and manipulating matrices and vectors is essential if we want to use R as a mathematical tool. Although this has been implemented in a consistent way in R , it is not simple for novice users! Practice is the best teacher, so you will get plenty of exercise.

Most of the exercises can be answered with one single R statement. However, as these statement smay be quite complicated, it is often simpler to first break them up into smaller parts, after which they are merged into one.

Vectors, sequences

- Use R function `mean` to estimate the mean of two numbers, 9 and 17. (you may notice that this is not as simple as you might think!).

- Vector V
  - Create a vector, called V, with even numbers, between 16 and 56. Do not use loops. (tip: use R function `seq`)
  - Display this vector
  - What is the sum of all elements of V? Do not use loops; there exists an R function that does this; the name of this function is trivial.
  - Display the first 4 elements of V
  - Calculate the product of the first 4 elements of V
  - Display the 4^{th}, 9^{th} and 11^{th} element of V . (tip: use the `c()` function).
- Vector W
  - Create a new vector, W, which equals vector V, multiplied with 3; display its content.
  - How many elements of W are smaller than 100?

    First create a new vector that contains only the elements from W<100 (call it W100), then calculate the length of this new vector.
  - Now perform the same calculation, in one R statement.
- Create a sequence that contains the values (1,1/2,1/3,1/4, ..., 1/10)
- Compute the square root of each element
- Compute the square (^{2}) of each element
- Create a sequence with values (0/1,1/2,2/3,3/4, ..., 9/10)
- Vector U
  - Create a vector, U, with 100 random numbers, uniformly distributed between -1 and 1.

    tip: R statement `runif` generates uniformly distributed random numbers; use `?runif` to see how it works.
  - Check the range of U; all values should be within -1 and +1.

    tip: there exists an R function to do that-its name is trivial.
  - Calculate the sum and the product of the elements of U
  - How many elements of U are positive?
  - Zero all negative values of U.
  - Sort U
- Vectors x, y
  - Create two vectors: vector x, with the elements: 2,9,0,2,7,4,0 and vector y with the elements 3,5,0,2,5,4,6 (in that order). (tip: use the c() function).
  - Divide all the elements of y by the elements of x.

  - Type in the following commands; try to understand:
    * x>y
    * x==0
  - Select all values of y that are larger than the corresponding values of x
  - Select all values of y for which the corresponding values of x are 0.
  - Remove all values of y for which the corresponding values of x equal 0.
  - Zero all elements of x that are larger or equal than 7. Show x.

Matrices

- Use R function `matrix` to create a matrix with the following contents:

$$\begin{bmatrix} 3 & 9 \\ 7 & 4 \end{bmatrix}$$

- display it to the screen
- Use R function `matrix` to create a matrix called "A":

$$\begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/4 & 1/5 & 1/6 \\ 1/7 & 1/8 & 1/9 \end{bmatrix}$$

  - Take the transpose of A.
  - Create a new matrix, B, by extracting the first two rows and first two columns of A. Display it to the screen.

- Use `diag` to create the following matrix, called "D":

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

- Use `cbind` and `rbind` to augment this matrix, such that you obtain:

$$\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 2 & 0 & 4 \\ 0 & 0 & 3 & 4 \\ 5 & 5 & 5 & 5 \end{bmatrix}$$

  It is simplest to do this in two statements (but it can be done in one!)

- Remove the second row and second column of the previous matrix

Diversity of deep-sea nematodes

We will now work on a data set consisting of nematode species densities, found in Mediterranean deep-sea sediments, at depths ranging from 160 m to 1220 m. The densities are expressed in number of individuals per 10 cm2.

Nematodes are small (< 1 mm) worms, and they are generally very abundant in all marine sediments.

The data (from (Soetaert, Heip, and Vincx 1991), originally present in an ACCESS database, have been exported as a table in a comma-separated-values (so-called csv) format. This format has the advantage that it can easily be read by text editors (such as the TINN-R editor) as well as in spreadsheet programs.

Open the file `nemaspec.csv` in Tinn-R[2]

Check its structure.

You may also open the file in EXCEL, but do not forget to close it before proceeding. EXCEL is very territorial, and will not allow another program, such as R , to access a file that is open in EXCEL.

On the first line is the heading (the names of the stations), the first column contains the species names. Before importing the file in R , check the working directory:

```
> getwd()
```

If the file called `nemaspec.csv` is not in this directory, you may need to change the working directory:

```
> setwd("directory name")
```

Ddo not forget that R requires "/" where windows uses "\".

Make a script file in which you write the next steps; submit each line to R to check its correctness. Read the comma-delimited file, using R command `read.csv`. Type `?read.csv` if you need help.

Specify that the first row is the heading (`header=TRUE`) and the first column contains the rownames (`row.names=1`).

Put the data in data.frame `Nemaspec`.

```
Nemaspec <- read.csv("nemaspec.csv", header = TRUE, row.names = 1)
```

Check the contents of `Nemaspec`. As the dataset is quite substantial, it is best to output only the first part of the data:

```
head(Nemaspec)
```

The rest is up to you:

[2]: if you do not have this file, it can be found in the **marelacTeaching** package directory. Within R, type: browseURL(paste(system.file(package="marelacTeaching"), "/doc/lecture/", sep="")).

- Select the data from station M160b (the 2^{nd} column of Nemaspec); put these data in a vector called `dens`.

  (remember: to select a complete column, you select all rows by leaving the first index blanc).

- Remove from vector dens, the densities that are 0. Display this vector on the screen. (Answer: [1] 6.580261 5.919719 etc...)

- Calculate N, the total nematode density of this station. The total density is simply the sum of all species densities (i.e. the sum of values in vector dens). What is the value of N ? (Answer :699).

- Divide the values in vector dens by the total nematode density N. Put the results in vector p, which now contains the relative proportions for all species. The sum of all values in p should now equal 1. Check that.

- Calculate S, the number of species: this is simply the length of p; call this value S. (Answer: S=126)

- Estimate the values of diversity indices N1 and N2 and Ni, given by the following formulae:

$$N1 = e^{\sum -p_i \cdot \log_e(p_i)}$$
$$N2 = 1/\left(\sum p_i^2\right)$$
$$Ni = 1/\max(p_i)$$

  You can calculate each of these values using only one R statement ! (A: 90.15358, 66.77841, 22.56157)

- The 126 nematode species per 10 cm2 were obtained by looking at all 699 individuals. Of course, the fewer individuals are determined to species, the fewer species will be encountered. Some researchers determine 100 individuals, other 200 individuals. To standardize their results, the expected number of species in a sample can be recalculated based on a common number of individuals. The expected number of species in a sample with size n, drawn from a population which size N, which has S species is given by:

$$ES(n) = \sum_{i=1}^{S} \left[1 - \frac{\binom{N-Ni}{n}}{\binom{N}{n}}\right]$$

  where Ni is the number of individuals in the ith species in the full sample and is the so-called "binomial coefficient", the number of different sets with size n that can be chosen from a set with total size N.

  In R, binomial coefficients are estimated with statement `choose(N,n)`.

  What is the expected number of species per 100 individuals ? (n=100,N=699). (A: ES(100) = 60.68971).

- Print all diversity indices to the screen, which should look like:

```
        N        N0        N1        N2        Ni       ESS
699.00000 126.00000  90.15358  66.77841  22.56157  60.68971
```

3. R functions

One of the strengths of R is that one can make user-defined functions that add to R s built-in functions.

3.1. Function definition

Typically, complex functions are written in R script files, which are then submitted to R . For instance,

```
Circlesurface <- function (radius) pi*radius^2
```

defines a function (called `Circlesurface`) which takes as input argument a variable called `radius` and which returns the value $\pi * radius^2$ (which is the surface of a circle).

After submitting this function to R , we can use it to calculate the surfaces of circles with given radius:

```
>Circlesurface(10)

[1] 314.1593

>Circlesurface(1:20)

 [1]    3.141593   12.566371   28.274334   50.265482   78.539816
 [6]  113.097336  153.938040  201.061930  254.469005  314.159265
[11]  380.132711  452.389342  530.929158  615.752160  706.858347
[16]  804.247719  907.920277 1017.876020 1134.114948 1256.637061
```

the latter statement will calculate the surface of circles with radiuses 1, 2, ... ,20.

More complicated functions may return more than one element:

```
Sphere <- function(radius)
{
 volume  <- 4/3*pi*radius^3
 surface <- 4 *pi*radius^2
 return(list(volume=volume,surface=surface))
}
```

Here we recognize

- the function heading (1^{st} line), specifying the name of the function (Sphere) and the input parameter (radius)
- the function specification. As the function comprises multiple statements, the function specification is embraced by curly braces {...}.

- The return values (last line). `Sphere` will return the volume and surface of a sphere, as a list.

The earth has approximate radius 6371 km, so its volume (km^3) and surface (km^2) are:

```
>Sphere(6371)

$volume
[1] 1.083207e+12

$surface
[1] 510064472
```

The next statement will only display the volume of spheres with radius 1, 2, ... 5

```
>Sphere(1:5)$volume

[1]   4.18879  33.51032 113.09734 268.08257 523.59878
```

Sometimes it is convenient to provide default values for the input parameters.

For instance, the next function estimates the density of "standard mean ocean water" (in $kg\ m^{-3}$), as a function of temperature, T, (and for salinity=0, pressure = 1 atm) (Millero and Poisson 1981); the input parameter T is by default equal to 20 °C:

```
Rho_W <- function(T = 20) {
  999.842594 + 0.06793952 * T - 0.00909529 * T^2 +
  0.0001001685 * T^3 - 1.120083e-06 * T^4 + 6.536332e-09 * T^5
}
```

By ending the first sentence with a `+` we made clear that the statement is not finished and continues on the next line. It would have been wrong to put the `+` on the next line.

Calling the function without specifying temperature, uses the default value:

```
>Rho_W()

[1] 998.2063

>Rho_W(20)

[1] 998.2063

>Rho_W(0)

[1] 999.8426
```

3.2. Programming

R has all the features of a high-level programming language:

If, else, ifelse constructs

Try to understand the following:

```
Dummy <- function (x) {
  if ( x<0  ) string <- "x<0"     else
  if ( x<2  ) string <- "0>=x<2"  else
              string <- "x>=2"
  print(string)
}

>Dummy(-1)

[1] "x<0"

>Dummy(1)

[1] "0>=x<2"

>Dummy(2)

[1] "x>=2"
```

Note that we have specified the `else` clause on the same line as the `if` part so that R knows that the statement is continued on the next line!

If and else constructs involving only one statement can be combined:

```
>x<-2
>ifelse (x>0, "positive", "negative,0")

[1] "positive"
```

Loops

Loops allow a set of statements to be executed multiple times.

The `for` loop iterates over a specified set of values. In the example below, the variable "i" takes on the values (1,2,3):

```
>for (i in 1:3) print(c(i,2*i,3*i))

[1] 1 2 3
[1] 2 4 6
[1] 3 6 9
```

`while` and `repeat` will execute until a specified condition is met:

```
>i<-1 ; while(i<3) {print(i); i<-i+1}

[1] 1
[1] 2
```

`break` exits the loop,

`next` stops the current iteration and advances to the next iteration.

```
>i <- 1
>repeat {
+   print(i)
+   i <-i + 1
+   if(i > 2) break
+ }

[1] 1
[1] 2
```

The curly braces {...} embrace multiple statements that are executed in each iteration.

Note: loops are implemented very inefficiently in R and should be avoided as often as possible. Fortunately, R offers many high-level commands that operate on vectors and matrices. These should be used as much as possible!

For more information about if constructs and loops, type

```
> ?Control
```

3.3. R packages

A package in R is a file containing many functions that perform certain related tasks. Packages can be downloaded from the R website.

Once installed, we generate a list of all available packages, we load a package and we obtain a list with its contents by the following commands:

```
> library()
> library(deSolve)
> library(help = deSolve)
> help(package = deSolve)
```

3.4. Exercises

R function sphere

Extend the `Sphere` function with the circumference of the sphere at the place of maximal radius. The formula for estimating the circumference of a circle with radius r is: $2 \cdot \pi \cdot r$. What is the circumference of the earth near the equator?

An R function to estimate saturated oxygen concentrations

The saturated oxygen concentration in water (μmolkg^{-1}), as function of temperature (T), and salinity (S) can be calculated by: $SatOx = e^A$ where:

A= -173.9894 + 25559.07/T + 146.4813* loge(T/100) -22.204*T/100 + S *

(-0.037362+0.016504*T/100-0.0020564 *T/100*T/100)

and T is temperature in Kelvin (Tkelvin = Tcelsius + 273.15).

Tasks:

- Make a function that implements this formula; the default values for temperature and salinity are 20°C and 35 respectively.
- What is the saturated oxygen concentration at the default conditions? (A: 225.2346)
- Estimate the saturated oxygen concentration for a range of temperatures from 0 to 30^{o}C, and salinity 35. (Tip: no need to use loops).

Loops

The Fibonacci numbers are calculated by the following relation: $F_n = F_{n-1} + F_{n-2}$

with $F_1 = F_2 = 1$

Tasks:

- Compute the first 50 Fibonacci numbers; store the results in a vector (use R command `vector` to create it). You have to use a loop here
- For large n, the ratio F_n/F_{n-1} approaches the "golden mean": $(1 + \sqrt{5})/2$
- What is the value of F_{50}/F_{49}; is it equal to the golden mean?
- When is n large enough? (i.e. sufficiently close ($<1e^{-6}$) to the golden mean)

Diversity of deep-sea nematodes for all stations

- Starting from your code to estimate diversity indices for deepsea station M160b, now write a loop that does so for all the stations in Nemaspec.
- First create a matrix called div, with the number of rows equal to the number of deepsea stations, and with 6 columns, one for each diversity index. This matrix will contain the diversity values.
- The column names of div are: "N", "N0", "N1", "N2", "Ninf", "ESS"

  The row names of matrix div are the station names (= the column names of Nemaspec). Tip: Use R command `colnames()`, `rownames()`
- Now loop over all columns of data frame Nemaspec, estimate the diversity indices and put the results in the correct row of matrix div:

```
for (i in 1:ncol(Nemaspec))
 {
   # you have to write this part of the code
 }
```

- Show matrix div to the screen

Diversity indices as a function

Based on the results obtained in previous section, make a function that will calculate the diversity indices for any data matrix.

Rarefaction diversity

[3] If you still have time and the courage: try an alternative way of estimating the number of species per 100 individuals by taking random "subsamples" of 100 individuals and estimating the number of species from this subsample.

If the procedure is repeated often enough, the mean value should converge to the expected number of species, ESS(100); this is the rarefaction method of (Sanders 1968).

You may need the following R functions:

- `round` (converting reals to integers),
- `cumsum` (take a cumulative sum),
- `sample` (take random selection of elements),
- `table` (to make a table of counts),

as well as `length`, `mean`.

(Hurlbert 1971) showed that rarefaction generally overestimates the true estimated number of species ; can you corroborate this finding?

[3] This question requires significant thought and imagination; there are several ways to do this.

4. Statistics

R originated as a statistical package, and it is still predominantly used for this purpose.

You can do virtually any statistical analysis in R .

As there exist many documents that may help you with statistical analyses in R, we will not deal with the subject here.

Statistics is used just to show you how to use efficiently use R , in cases where you have no clue where to begin!

4.1. Using R in four steps

Suppose you want to perform a hierarchic clustering and plot the dendrogram of a multivariate data set.

If you have never done that in R here are the steps:

1. Find a function that performs the requested task.

   for instance, use `help.search ("cluster")` to help you.

   Depending on the number of packages that you have installed, R will list a number of possible functions whose help file contains the word "cluster". Use the function from the package **stats** (part of the core of R).

2. Open the help file (?<name-of-the-function>) and look up the syntax for this function. If you have no time to read it completely, at least read (part of) the section "Description", "Usage", and "Examples".

3. Try the examples in the help file. You may:

   - Try them all at once (example(<name-of-the-function>).
   - Alternatively, you may select the statements in the Examples section that look applicable to your problem, copy-paste them into your script file (Ctrl-C / Ctrl-V) and execute them; e.g. line by line. Chances are real that, if they are suited for your case, you will transform them anyway.

4. Transform a promising example so that it suits your problem.

4.2. Exercise: multivariate statistics on the nematode species data.

Use R to perform a multivariate statistical analysis of the nematode data.

Beware: the nematode data have stations as columns and species as rows.

- Perform a hierarchic clustering and plot the dendrogram
- Perform a principal component analysis (PCA) and plot the results; you may also repeat the PCA analysis, with the first two stations removed!

5. Graphics

R has extensive graphical capabilities, and allows making simple (1-D, x-y), image-like (2-D) and perspective (3-D) figures.

Try:

```
> demo(graphics)
> demo(image)
> demo(persp)
```

to obtain a display of R s simple (1-D, x-y), image-like (2-D) and perspective (3-D) capabilities.

Graphics are plotted in the figure window which floats independently from the other windows. If not already present, it is launched by writing (in windows):

```
> windows()
```

or

```
> x11()
```

A figure consists of a plot region surrounded by 4 margins, which are numbered clockwise, from 1 to 4, starting from the bottom. R distinguishes between:

1. high-level commands. By default, these create a new figure, e.g.
   - `hist`, `barplot`, `pie`, `boxplot`, ... (1-D plot)
   - `plot`, `curve`, `matplot`, `pairs`,... ((x-y)plots)
   - `image`, `contour`, `filled.contour`,... (2-D surface plots)
   - `persp`, `scatterplot3d`,... (3-D plots) [4].
2. low-level commands that add new objects to an existing figure, e.g.
   - `lines`, `points`, `segments`, `polygon`, `rect`, `text`, `arrows`, `legend`, `abline`, `locator`, `rug`, ... These add objects within the plot region
   - `box`, `axis`, `mtext` (text in margin), `title`, ... which add objects in the plot margin
3. graphical parameters that control the appearance of.
   - plotting objects: `cex` (size of text and symbols), `col` (colors), `font`, `las` (axis label orientation), `lty` (line type), `lwd` (line width), `pch` (the type of points), ...
   - graphic window: `mar` (the size of the margins), `mfrow` (the number of figures on a row), `mfcol` (number figures on a column),...

```
> ?plot.default
> ?par
> ?plot.window
> ?points
```

[4] scatterplot3d is in R package **scatterplot3d** which has to be loaded first

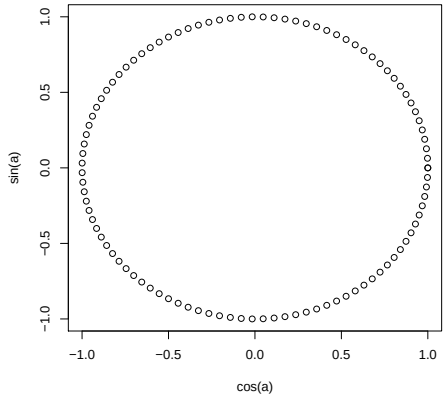


Figure 1: Simple figure with `plot` - see text for R code

will open the help files, while

```
> example(plot.default)
> example(points)
```

will run the examples, displaying each new graph, after you have pressed "<ENTER>" (try it!)

5.1. X-Y plots

A circle can be plotted by (x,y) points, where $x = r \cdot \cos(a)$ and $y = r \cdot \sin(a)$, with a the angle, from 0 to 2π, and r the radius. In the following script, we first generate a sequence of angle values, a, from 0 to 2π, comprising 100 values (`length.out`) and then plot a circle with unit radius:

```
>a <- seq(0,2*pi, length.out=100)
>plot(x=cos(a),y=sin(a))
```

As `plot` is a high-level command, it starts a new figure.

By default, R adds axes, and labels, and represents the (x,y) data as small dots (points). Note that the graph is not symmetrical.

We will now make a more complex figure that resembles a "target face", e.g. for practicing archery or to throw darts. We first use the same command (`plot`) as above, but we add a number of graphical parameters that specify that:

- Rather than dots, the points should be connected by lines (`type`).
- The line should be twice as wide as the default (`lwd`)
- The x- and y-axes labels (`xlab`,`ylab`) have to be empty

- The axes and axes annotations (`axes`) are removed
- The graph has to be symmetrical, i.e. the x/y aspect ratio = 1 (`asp`).

```
plot(cos(a), sin(a), type = "l", lwd = 2, xlab = "", ylab="",
  axes = FALSE, asp = 1)
```

To this figure, we can now add several low-level objects:

- a series of lines, representing smaller and smaller circles (`lines`).

  ```
  for (i in seq( 0.1,0.9,by=0.1)) lines(i*sin(a), i*cos(a))
  ```

- an innermost red polygon (`polygon`).

  ```
  polygon(sin(a)*0.1,cos(a)*0.1,col="red")
  ```

- point marks as text labels, ranging from from 10 to 1 (`text`). The closer to the centre, the higher the score

  ```
  for (i in 1:10) text(x=0,y=i/10-0.025,labels=11-i,font=2)
  ```

  Now two archers take 10 shots at the target face.

  We mimic their arrows by generating normally distributed (x,y) numbers, with mean=0 (the centre!) and where the experience of the archer is mimicked by the standard deviation. The more experienced, the closer the arrows will be to the centre, i.e. the lower the standard deviation.

- R statement `rnorm` generates normally distributed numbers; we need 20 of them, arranged as a matrix with 2 columns.

  ```
  shots1 <- matrix(ncol=2, data=rnorm(n=20,sd=0.2))
  shots2 <- matrix(ncol=2, data=rnorm(n=20,sd=0.5))
  ```

- The shots are added to the plot as points, colored darkblue (experienced archer) and darkgreen (beginners level). Note that we choose a 50% enlarged point size (cex), and we choose a circular shaped point (`pch=16`)

  ```
  points(shots1,col="darkblue",pch=16,cex=1.5)
  points(shots2,col="darkgreen",pch=16,cex=1.5)
  ```

- Finally, we add a legend, explaining who has done the shooting:

  ```
  legend("topright",legend=c("A","B"),pch=16,
          col=c("darkblue","darkgreen"),pt.cex=1.5)
  ```

  Note that the legend text and the colors are inputted as a vector of strings, using the `c()` function (e.g. `c("A", "B")`).

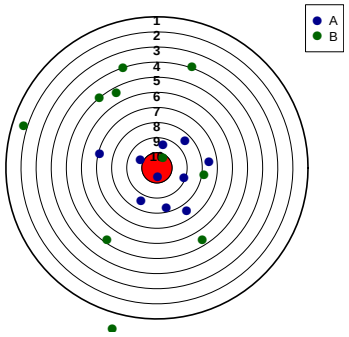


Figure 2: Figure with several low-level objects - see text for R code

5.2. X-Y plots; conditional plotting

As a more sophisticated demonstration of the use of symbols in R graphs, we work on a biological example, from the R data set called "Orange". This data set contains the circumference (in mm, at breast height) measured at different ages for five orange trees. We start by looking at the data (only part is displayed).

```
>head(Orange)
```

```
  Tree  age circumference
1    1  118            30
2    1  484            58
3    1  664            87
4    1 1004           115
5    1 1231           120
6    1 1372           142
```

```
>tail(Orange)
```

```
   Tree  age circumference
30    5  484            49
31    5  664            81
32    5 1004           125
33    5 1231           142
34    5 1372           174
35    5 1582           177
```

and make a rough plot of circumference versus age:

```
>plot(Orange$age, Orange$circumference,xlab="age, days",
+      ylab="circumference, mm", main= "Orange tree growth")
```

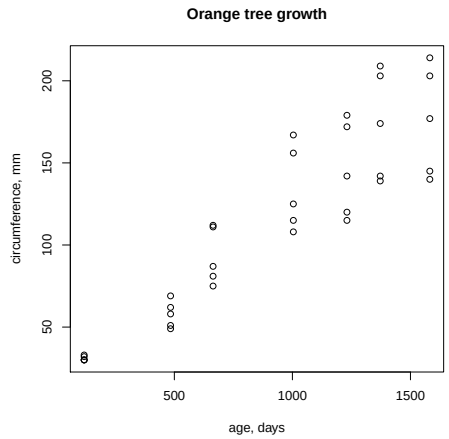


Figure 3: Simple plot of the `orange` dataset - see text for R code

(as Orange is a dataframe, columns can be addressed by their names, `Orange$age` and `Orange$circumference`).

The output (figure) shows that there is a lot of scatter, which is due to the fact that the five trees did not grow at the same rate.

It is instructive to plot the relationship between circumference and age differently for each tree. In R, this is simple: we can make some graphical parameters (symbol types, colors, size,...) conditional to certain "factors".

Factors play a very important part in the statistical applications of R; for our application, it suffices to know that the factors are integers, starting from 1.

In the R statement below, we simply use different symbols (`pch`) and colors (`col`) for each tree: `pch=(15:20)[Orange$Tree]` means that, depending on the value of `Orange$Tree` (i.e. the tree number), the symbol (`pch`) will take on the value 15 (tree=1), 16 (tree=2),... 20 (tree=5). `col=(1:5) [Orange$Tree]` does the same for the point color. The final statement adds a legend, positioned at the bottom, right.

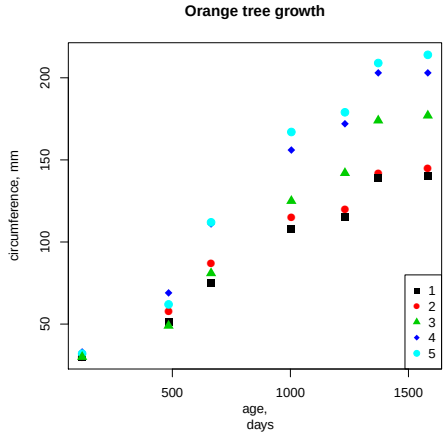
```
>plot(Orange$age, Orange$circumference,xlab="age,
+    days",ylab="circumference, mm", main= "Orange tree growth",
+    pch=(15:20)[Orange$Tree],col=(1:5) [Orange$Tree],cex=1.3)
>legend("bottomright",pch=15:20,col=1:5,legend=1:5)
```

The output shows that tree number 5 grows fastest, tree number 1 is slowest growing. (note: it is also instructive to run the examples in the Orange help file.)

5.3. Zooplankton growth rates

`Zoogrowth` from package **marelacTeaching** is a literature data set, compiled by Hansen et al. (1997) with measurements of maximal growth rates of zooplankton organisms as a function of body volume.

Run the example for this data set (you will need to load package **marelacTeaching** first):




```
> require(marelacTeaching)
> example(Zoogrowth)
```

5.4. Images and contour plots

R has some very powerful functions to create images and add contours. For example, the data set `Bathymetry` from the **marelac** package can be used to generate the bathymetry (and hypsometry) of the world oceans (and land):

```
>require(marelac)
>image(Bathymetry$x,Bathymetry$y,Bathymetry$z,col=femmecol(100),
+       asp=TRUE,xlab="",ylab="")
>contour(Bathymetry$x,Bathymetry$y,Bathymetry$z,add=TRUE)
```

Note the use of `asp=TRUE`, which maintains the aspect ratio.

5.5. Plotting a mathematical function

Plot curves for mathematical functions are quickly generated with R-command `curve`:

```
>curve(sin(3*pi*x))
>curve(sin(3*pi*x),from=0,to=2,col="blue",
+       xlab="x",ylab="f(x)",main="curve")
>curve(cos(3*pi*x),add=TRUE,col="red",lty=2)
>abline(h=0,lty=2)
>legend("bottomleft",c("sin","cos"),text.col=c("blue","red"),lty=1:2)
```

The first command will plot the curve for $y = sin(3 \cdot \pi \cdot x)$, using the default settings (Fig. left), while the next commands afirst draw a graph of $y = sin(3 \cdot \pi \cdot x)$, in blue (`col`), and for x values ranging between 0 and 2 (`from`, `to`), adding a main title (`main`) and x- and y-axis labels (`xlab`, `ylab`) (1^{st} sentence).

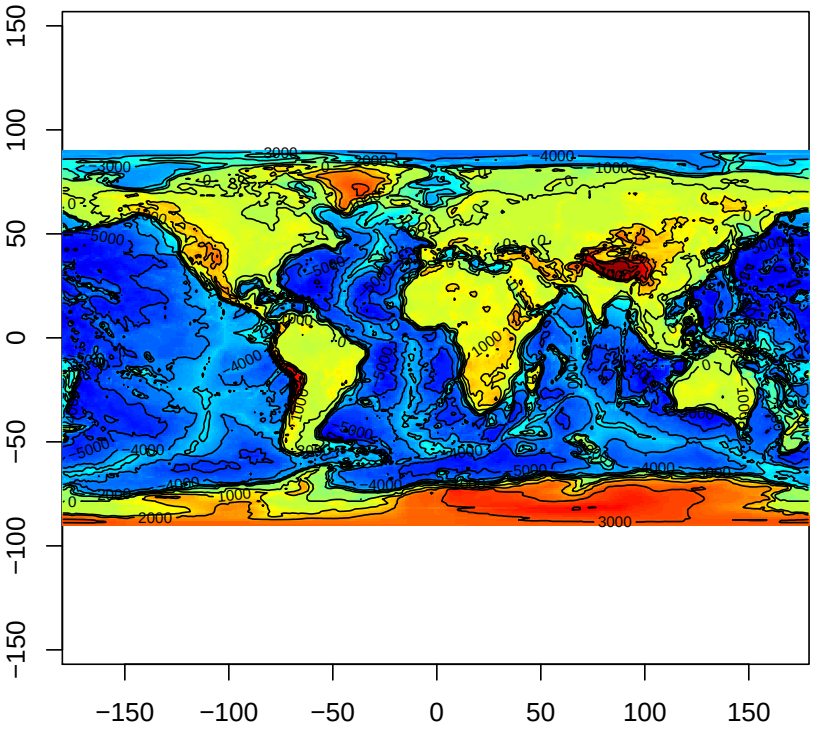


Figure 4: Image plot of ocean bathymetry - see text for R code

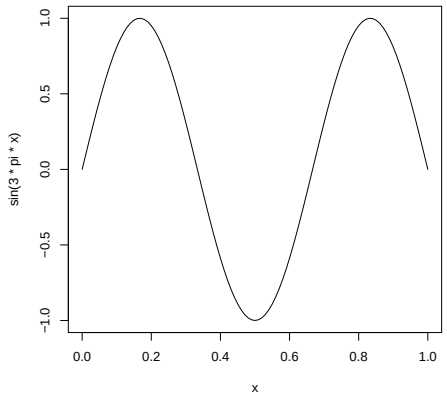


Figure 5: Plotting a mathematical function - see text for R code

The 2^{nd} R sentence adds the function $y = cos(3 \cdot \pi \cdot x)$, as a red (`col`) dashed line (`lty`). Note the use of parameter `add=TRUE`, as by default curve creates a new plot.

The final statements adds the x-axis, i.e. a horizontal, dashed (`lty = 2`), line (`abline`) at y=0 and a legend.

5.6. Multiple figures

There are several ways in which to arrange multiple figures on a plot.

1. The simplest is by specifying the number of figures on a row (mfrow) and on a column (mfcol):

   ```
   > par(mfrow=c(3,2))
   ```

   will arrange the next plots in 3 rows, 2 columns. Graphs will be plotted row-wise.

   ```
   > par(mfcol=c(3,2))
   ```

   will arrange the plots in 3 columns, 2 rows, in a columnwise sequence. Note that both mfrow and mfcol must be inputted as a vector. Try:

   ```
   > par(mfrow=c(2,2))
   > for ( i in 1:4) curve(sin(i*pi*x),0,1,main=i)
   ```

2. R function `layout` allows much more complex plot arrangements.

5.7. Exercises

Simple curves

- Create a script file which draws a curve of the function $y = x^3 sin^2(3\pi x)$ in the interval [-2, 2].
- Make a curve of the function $y = 1/cos(1 + x^2)$ in the interval [-5,5].

Human population growth

The human population (N, millions of people) at a certain time t, can be described as a function of time (t), the initial population density at t=t0 (Nt0), the carrying capacity "K" and the rate of increase "a" by the following equation [5]:

$$N(t) = \frac{K}{1 + [\frac{K - N_{t0}}{N_{t0}}]e^{-a \cdot (t - t0)}}$$

For the US, the population density in 1900 (N0) was 76.1 million; the population growth can be described with parameter values: a=0.02 yr^{-1}, K = 500 million of people.

Actual population values are:

[5] This is the solution of a so-called logistic differential equation (Verhulst 1838)

```
1900  1910  1920  1930  1940  1950  1960  1970  1980
76.1 92.4 106.5 123.1 132.6 152.3 180.7 204.9 226.5
```

Tasks:

1. Plot the population density curve as a thick line, using the US parameter values.
2. Add the measured population values as points. Finish the graph with titles, labels etc...

Toxic ammonia

Ammonia nitrogen is present in two forms: the ammonium ion (NH_4^+) and unionized ammonia (NH_3). As ammonia can be toxic at sufficiently high levels, it is often desirable to know its concentration.

The relative importance of ammonia, (the contribution of ammonia to total ammonia nitrogen, $NH_3/(NH_3 + NH_4^+)$) is a function of the proton concentration [H^+] and a parameter KN, the so-called stoichiometric equilibrium constant:

$$p_{[NH_3]} = \frac{K_N}{K_N + [H^+]}$$

Tasks:

- Plot the relative fraction of toxic ammonia to the total ammonia concentration as a function of pH, where $pH = -log_{10}([H^+])$ and for a temperature of 30°C. Use a range of pH from 4 to 9.

  The value of KN is 810^{-10} at a temperature of 30°C.

- Add to this plot the relative fraction of ammonia at 0°C; the value of KN at that temperature is 8 10^{-11} mol kg^{-1}.

The iris data set

A famous data set that is part of R is the "iris" data set (Fisher, 1936), which we will explore next.

It gives measurements, in centimeters for sepal length and width and petal length and width, respectively, for 50 flowers of the species *Iris setosa*, *Iris versicolor* and *Iris virginica*.

Tasks:

- Have a look at the data:
- What is the class of the data set? why?
- What are the dimensions of the data set? (number of rows, columns)
- Produce a scatter plot of petal length against petal width; produce an informative title and labels of the two axes.

- Repeat the same graph, using different symbol colours for the three species.
- Add a legend to the graph. Copy-paste the result to a WORD document. If you do not have WORD, make a PDF file of the graph.
- Create a box-and whisker plot for sepal length where the data values are split into species groups; use as template the first example in the "boxplot" help file.
- Now produce a similar box-and whisker plot for all four morphological measurements, arranged in two rows and two columns. First specify the graphical parameter that arranges the plots two by two.

6. Matrix algebra

Matrix algebra is very simple in R. Practically everything is possible! Here are the most important R functions that operate on matrices:

- %*% Matrix multiplication
- t(A) transpose of A
- diag(A) diagonal of A
- solve(A) inverse of A
- solve(A,B) solving Ax=B for x
- eigen(A) eigenvalues and eigenvectors for A
- det(A) determinant of A

For instance the following first inverts matrix A (`solve(A)`), and then multiplies the inverse with A , giving the unity matrix:

```
>(A <-matrix(nrow=2,data=c(1,2,3,4)))

     [,1] [,2]
[1,]    1    3
[2,]    2    4

>solve(A) %*% A

     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

whilst `t(A)` will transpose matrix `A` (interchange rows and columns).

```
>t(A)

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

The next set of statements will solve the linear system Ax=B for the unknown vector x:

```
>B <- c(5,6)
>solve(A,B)

[1] -1  2
```

Finally, the eigenvalues and eigenvectors of A are estimated using R function `eigen`. This function returns a list that contains both the eigenvalues (`$values`) and the eigenvectors (`$vectors`), (the columns).

```
>eigen(A)

$values
[1]  5.3722813 -0.3722813

$vectors
           [,1]       [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736
```

6.1. Exercises

Matrix algebra exercise 1

- Use R function `matrix` to create the matrices called `A` and `B`:
  $A = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 4 & 1 \\ -2 & 1 & -1 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$
- Take the inverse of A and the transpose of A.
- Multiply A with B.
- Estimate the eigenvalues and eigenvectors of A.
- For a matrix A, x is an eigenvector, and λ the eigenvalue of a matrix A, if $A \cdot x = \lambda \cdot x$. Test it!

Matrix algebra exercise 2

- Create a matrix, called P:

$$\begin{bmatrix} 0 & 0.0043 & 0.1132 & 0 \\ 0.9775 & 0.9111 & 0 & 0 \\ 0 & 0.0736 & 0.9534 & 0 \\ 0 & 0 & 0.0452 & 0.9804 \end{bmatrix}$$

- What is the value of the largest eigenvalue (the so-called dominant eigenvalue) and the corresponding eigenvector?.

- Create a new matrix, T, which equals P, except for the first row, where the elements are 0.
- Now estimate N= (I-T)-1, where I is the identity matrix [6].

System of linear equations

- Solve the following system of linear equations for the unknown x_i:

```
3x1 + 4x2 + 5x3 = 0
6x1 + 2x2 + 7x3 = 5
7x1 +  x2       = 6
```

- You will first need to rewrite this problem in the form: Ax=B, where A contains the coefficients, x the unknowns, and B the right-hand side values. Then you solve the system, using R function `solve`
- Check the results (i.e. is Ax = B ?)

[6] Note: this is a stage-model of a killer whale (Caswell 2001). The eigenvalue-eigenvectors estimate the rate of increase and stable age distribution, the matrix N contains the mean time spent in each stage.

7. Roots of functions

7.1. Roots of a simple function

Suppose we want to solve the following problem: $\cos(x) = 2 \cdot x$ for x.

Mathematically, we seek the root of the function $y = \cos(x) - 2 \cdot x$, this is the value of x for which $y = 0$.

As the function is quite complex, it is not possible to find an exact solution (an explicit expression) for this root.

It is always a good idea to plot the equation (1^{st} line), and add the x-axis (2^{nd} line).

```
curve(cos(x) - 2 * x, -10, 10)
abline(h = 0, lty = 2)
```

This figure shows that there indeed exists a value x, for which $y = 0$.

Now R function `uniroot` can be used to locate this value.

Functions that seek a root from a nonlinear equation generally work "iteratively", i.e. they move closer and closer to the root in successive steps (iterations).

It is usually not feasible to find this root exactly, so it is approximated, i.e. up to a certain accuracy (tol, a very small number) [7].

For the method to work, there should be at least one root in the interval.

The statement below solves for the root; it returns several values, as a list.

```
>(rr<-uniroot(f = function(x) cos(x)-2*x, interval=c(-10,10)))

$root
[1] 0.4501686

$f.root
[1] 3.655945e-05

$iter
[1] 5

$estim.prec
[1] 6.103516e-05
```

The most important value is the root itself (`$root`), which is 0.45103686;

the function value at the root was 3.66e-5, the function performed 5 iterations.

In this example, the function was simple enough to include it in the call to `uniroot`.

The next chapter gives a more complex example from aquatic chemistry, where the equation to solve is significantly more complex.

Finally, we add the root to the figure:

[7] More specifically: the root of $y = \cos(x) - 2 \cdot x$ is the value x for which |cos(x)-2*x | < tol or for which successive changes of x are < tol

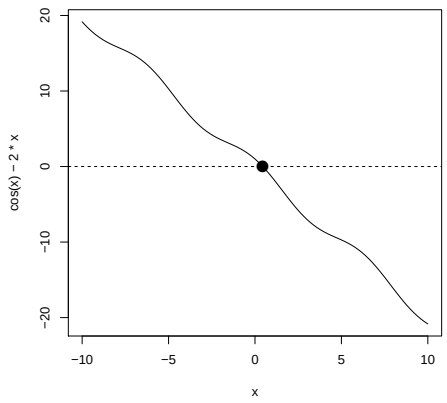


Figure 6: Function drawn with `curve` and the root of the function plotted - see text for R code

```
points(rr$root,0,pch=16,cex=2)
```

7.2. Root of a complex function: solving for the pH

In aquatic systems, the buffering capacity of dissolved inorganic carbon (DIC) species: carbon dioxide (CO_2), bicarbonate ions (HCO_3^-) and carbonate ions (CO_3^{2-}) keep pH in a narrow range. The buffering capacity is measured by total alkalinity, TA, (defined below).

If alkalinity and inorganic carbon concentration (DIC) are known, it is possible to calculate pH (Zeebe and Wolf-Gladrow 2003), by solving the following equations for the unknown proton concentration $[H^+]$ [8].

$$[\mathrm{HCO_3^-}] = \frac{K_{C1} \cdot [H^+]}{[H^+] \cdot [H^+] + K_{C1} \cdot [H^+] + K_{C1} \cdot K_{C2}} \cdot DIC$$

$$[\mathrm{CO_3^{2-}}] = \frac{K_{C1} \cdot K_{C2}}{[H^+] \cdot [H^+] + K_{C1} \cdot [H^+] + K_{C1} \cdot K_{C2}} \cdot DIC$$

$$TA = 2[\mathrm{CO_3^{2-}}] + [\mathrm{HCO_3^-}] - [\mathrm{H^+}]$$

Here is how to solve for the proton concentration $[H^+]$ (or the pH value) in R .

The trick is to estimate alkalinity based on a guess of proton concentration, using equation (3) and compare that with the measured alkalinity value. If both are equal within the tolerance level, the proton concentration has been found.

In the implementation below, the dissociation constants for carbonate (`kc1`, `kc2`) and at salinity 0, temperature 20, and pressure 0 are calculated in R s package **seacarb**, which has to be loaded first (`require`).

We then define a function whose root has to be solved (`pHfunction`). In this function we estimate total alkalinity, based on the guess of pH, the dissociation constants (`kc1`,`kc2`) and

[8] in practice, it is possible to merge these 3 equations such that only one equation is obtained, but this is neither didactically clearer nor computationally more efficient

the DIC concentration. The difference of this calculated alkalinity (`EstimatedAlk`) with the true alkalinity is then returned; if pH is correctly estimated, then true and estimated alkalinity will be equal, and the difference will be zero. So, to find the pH, we need to find the root of this function.

Note that the conversion from pH to $[H^+]$ gives the proton concentration in $mol\ kg^{-1}$. As the concentrations of the other substances are in $\mu mol\ kg^{-1}$, we convert using a factor 10^6.

We restrict the region of the pH root in between 0 and 12 (which is more than large enough), and we set the tolerance (tol) to a very small number to increase precision.

```
require(seacarb)
kc1 <- K1(S=0,T=20,P=0)    # Carbonate k1
kc2 <- K2(S=0,T=20,P=0)    # Carbonate k2

pHfunction <- function(pH, kc1,kc2, DIC, Alkalinity ) {
   H    <- 10^(-pH)
   HCO3 <- H*kc1  /(H*kc1 + H*H + kc1*kc2)*DIC
   CO3  <- kc1*kc2 /(H*kc1 + H*H + kc1*kc2)*DIC

   EstimatedAlk  <- -H *1.e6  + HCO3 + 2*CO3
   return ( EstimatedAlk  - Alkalinity)
}
```

```
>uniroot(pHfunction, interval=c(0, 12), tol=1.e-20, kc1=kc1, kc2=kc2,
+         DIC=2100, Alkalinity=2200)

$root
[1] 9.093965

$f.root
[1] 4.547474e-13
attr(,"pH scale")
[1] "total scale"
attr(,"unit")
[1] "mol/kg-soln"
attr(,"method")
[1] "Millero (2010)"

$iter
[1] 15

$estim.prec
[1] 3.552714e-15
```

7.3. Exercises

Simple functions

- Find the root of the equation $e^x = 4x^2$ in the interval [0,1].

  First draw the function curve.

- Solve the equations $1000 = y * (3 + x) * (1 + y)^4$ for y and with x varying over the range from 1 to 100. Plot the root as a function of x.

  Tip: first make a sequence of `x`-values, then loop over each `x` value, each time estimating the root and putting it in a vector.

Chemistry: pCO2 rises increase acidity

pH can also be estimated based on the measured alkalinity and pCO_2, the partial pressure of CO_2. To solve this equation, it is simplest to use another (equivalent) way to write the relationships between the DIC species:

$$[HCO_3^-] = K_{C1} \cdot \frac{[CO_2]}{[H^+]}$$

$$[CO_3^{2-}] = K_{C2} \cdot \frac{[HCO_3^-]}{[H^+]}$$

pCO_2 relates to $[CO_2]$ through Henrys constant, Kh, which can also be estimated as a function of salinity, temperature and pressure, using R package **seacarb**:

$$pCO_2 = \frac{[CO_2]}{Kh}$$

- Estimate the pH at equilibrium with alkalinity 2300 $\mu mol\ kg^{-1}$ and the current pCO_2 of 360 ppm.

  Use package `seacarb` to estimate the dissociation constants and Henrys constants at temperature 20°C, salinity 0, and pressure 0. (A: pH=8.19)

- The Intergovernmental Panel on Climate Change predicts for 2100 an atmospheric CO_2 concentration ranging between 490 and 1250 ppmv, depending on the socio-economic scenario (IPCC, 2007). These increases of pCO_2 make the water more acid. Make a plot of pH as a function of these increased atmospheric pCO_2 levels. (Assume that the pCO_2 of the ocean is at equilibrium with the atmospheric pCO_2). What is the maximal drop of pH ? (A: at pCO_2 of 1250 ppmv, pH=7.68).

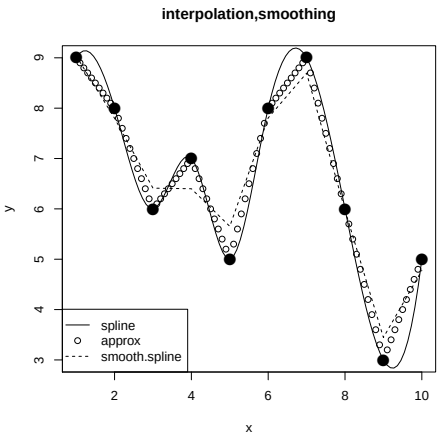


Figure 7: smoothing and interpolation - see text for R code

8. Interpolation, smoothing

Interpolating and smoothing in R can be done in several ways:

- `approx` linearly interpolates through points,
- `spline` uses spline interpolation, which is smoother,
- `smooth.spline` smoothens data sets; this means that it does not connect the original points.

The use of these functions is exemplified in the following script and corresponding output:

```
>x <- 1:10
>y <- c(9, 8, 6, 7, 5, 8, 9, 6, 3, 5)

>plot(x, y, pch=16, cex = 2, main = "interpolation,smoothing")
>lines (spline(x, y, n = 100), lty = 1)
>points(approx(x, y, xout = seq(1, 10, 0.1)), pch = 1)
>lines (smooth.spline(x, y), lty = 2)
>legend("bottomleft", lty = c(1, NA, 2), pch = c(NA, 1, NA),
+        legend = c("spline", "approx", "smooth.spline"))
```

8.1. Curve fitting

R also has several curve fitting procedures. Depending on whether the function to be fitted is linear, or non-linear, you may use:

- `lm` and `glm` for fitting linear models and generalised linear models

- `nls`, `nlm`, `optim`, `constrOptim` for nonlinear models.

As an example, we now fit the US population density values, at 10-year intervals, with the logistic growth model (see previous chapter). The model was: $N(t) = \frac{K}{1+[\frac{K-N_{t0}}{N_{t0}}].e^{-a\cdot(t-t_0)}}$, and the data:

```
1900 1910 1920 1930 1940 1950 1960 1970 1980
76.1 92.4 106.5 123.1 132.6 152.3 180.7 204.9 226.5
```

We start by inputting the data.

```
>year<- seq(1900, 1980, by = 10)
>pop <- c(76.1, 92.4, 106.5, 123.1, 132.6, 152.3, 180.7, 204.9, 226.5)
```

The simplest method for non-linear curve fitting is by using R function `nls`.

This functions requires as input the formula (`y ~ f(x, parameters)`) and starting values of the parameters.

In the example, `y` are the population values, `f` is the logistic growth formulation.

As starting conditions, we use: $K = 500, N0 = 76.1, a = 0.02$.

```
>fit <- nls(pop ~ K/(1 + (K - N0) / N0 * exp(-a * (year - 1900))),
+            start = list(K = 500, N0 = 76.1, a = 0.02))
```

We end the fitting by printing a summary of the fitting parameters, which shows the estimates of the parameters and their standard errors. Clearly, it is not possible to obtain reliable estimates of the value of K based on the data.

```
>summary(fit)

Formula: pop ~ K/(1 + (K - N0)/N0 * exp(-a * (year - 1900)))

Parameters:
    Estimate Std. Error t value Pr(>|t|)
K  1.008e+03  8.932e+02   1.129  0.30209
N0 7.866e+01  2.531e+00  31.084 7.36e-08 ***
a  1.550e-02  2.505e-03   6.188  0.00082 ***
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

Residual standard error: 3.685 on 6 degrees of freedom

Number of iterations to convergence: 6
Achieved convergence tolerance: 4.843e-06
```

The values of the coefficients themselves are retrieved using R function `coef`.

```
>(cc<-coef(fit))
```

```
           K           N0            a
1.008227e+03 7.866365e+01 1.550343e-02
```

8.2. Exercises

Smoothing

An anemometer measures wind-velocity at three hourly intervals. On a certain day, these velocities are: 5,6,7,9,4,6,3,7,9 at time 0, 3, ... 24 o'clock respectively. In order to estimate air-sea exchange, we need hourly measures.

Tasks:

- Interpolate the three-hourly measurements to hourly measurements.
- Make a plot of the interpolated values

Fitting

Primary production is measured by ^{14}C incubations from phytoplankton samples, at different light intensities.

The data are:

```
>ll <- c(0.,1,10,20,40,80,120,160,300,480,700)
>pp <- c(0.,1,3,4,6,8,10,11,10,9,8)
```

Fit the resulting production estimates (pp), as a function of light intensity (ll) with the 3-parameter Eilers-Peeters equation. The primary production is calculated as:

$$pp = p\max \cdot \frac{2 \cdot (1+\beta) \cdot {}^{I}/_{Iopt}}{({}^{I}/_{Iopt})^2 + 2 \cdot \beta \cdot {}^{I}/_{Iopt} + 1}$$

where I is light and pmax, β and Iopt are parameters.

- First plot primary production (pp) versus light (ll). Use large symbols.
- Then use R function `nls` to fit the model to the data
- Add the best-fit line to the graph. (note: use `coef` to retrieve the best parameter values).

9. Differential equations

Differential equations express the rate of change of a constituent (C) along one or more dimensions, usually time and/or space.

Consider the following set of two differential equations:

$$\frac{dA}{dt} = r \cdot (x - A) - k \cdot A \cdot B$$
$$\frac{dB}{dt} = r \cdot (y - B) + k \cdot A \cdot B$$

- A and B are called differential variables (or state variables),
- $\frac{dA}{dt}$ is the derivative (or the rate of change),
- r,x,y and k are parameters (constants).

To solve sets of differential equations in R we define a function (here called `model`), and which has as input the time (`t`), the values of the state variables (`state`) and the values of the parameters (`pars`). This function simply calculates the rate of change of the state variables (`dA` and `dB`) and returns those as a list.

The R statement `with (as.list (c(state,pars)),` ensures that the state variables and parameters can be addressed by their names.

```
model <- function(t, state, pars) {
  with (as.list(c(state, pars)), {
    dA <- r * (x - A) - k * A * B
    dB <- r * (y - B) + k * A * B
    return (list(c(dA, dB)))
  })
}
```

Before we can solve this model, we

- generate a sequence of time values at which we want output (`times`),
- assign initial conditions to the state variables (`state`) and
- give values to the parameters (`parms`):

```
times <- seq(0, 300, 1)
state <- c(A = 1, B = 1)
parms <- c(x = 1, y = 0.1, k = 0.05, r = 0.05)
```

The model can now be solved. To do so, we use R s integration routine `ode`; which can be found in R -package **deSolve**. This package is loaded first.

```
require(deSolve)
```

At each time `t`, `ode` will call function `model`, with the current values of the state variables and the parameter values.

The output is stored in a data.frame, called `out`.

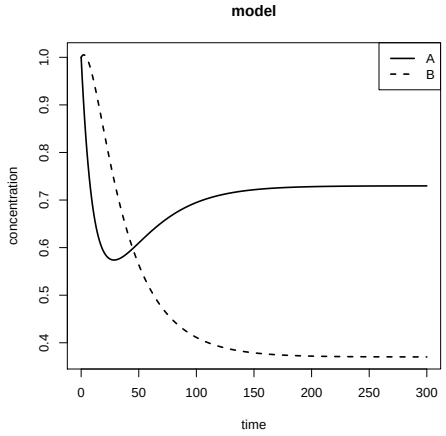


Figure 8: ode model - see text for R code

```
out   <- as.data.frame(ode(state,times,model,parms))
```

All we need to do now is to plot the model output. Before we do so, we have a look at data.frame out:

```
>head(out)
```

```
  time         A         B
1    0 1.0000000 1.0000000
2    1 0.9523189 1.0037869
3    2 0.9090687 1.0052854
4    3 0.8699226 1.0047151
5    4 0.8345728 1.0022854
6    5 0.8027203 0.9982009
```

The data are arranged in three columns: first the time, then the concentrations of A and B. As out is a data frame we can extract the data using their names (`out$time`, `out$A`, `out$B`).

Before plotting the model output, the range of concentrations of substances A and B is estimated; this is used to set the limits of the y-axis (`ylim`).

R function `plot` creates a new plot; `lines` adds a line to this plot; `lty` selects a line type; `lwd = 2` makes the lines twice as thick as the default. Finally a `legend` is added.

```
ylim   <- range(c(out$A,out$B))
plot(out$time,out$A,xlab="time",ylab="concentration",
      lwd=2,type="l",ylim=ylim,main="model")
lines(out$time,out$B,lwd=2,lty=2)
legend("topright",legend=c("A","B"),lwd=2,lty=c(1,2))
```

9.1. Exercises

Lotka-Volterra model

- Write a script file that solves the following system of ODEs [9]:

$$\frac{dx}{dt} = a \cdot x \cdot (1 - \frac{x}{K}) - b \cdot x \cdot y$$
$$\frac{dy}{dt} = g \cdot b \cdot x \cdot y - e \cdot y$$

  for initial values $x = 300, y = 10$ and parameter values: $a = 0.05, K = 500, b = 0.0002, g = 0.8, e = 0.03$.

- Make three plots, one for x and one for y as a function of time, and one plot expressing y as a function of x (this is called a phase-plane plot). Arrange these plots in 2 rows and 2 columns.

- Now run the model with other initial values ($x = 200, y = 50$); add the (x, y) trajectories to the phase-plane plot.

Butterfly

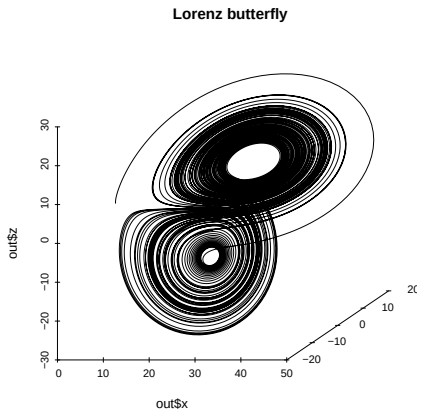
The Lorenz equations (Lorenz 1963) were the first chaotic system of differential equations to be discovered. They are three differential equations that were derived to represent idealized behavior of the earths atmosphere.

$$\frac{dx}{dt} = -\frac{8}{3} \cdot x + y \cdot z$$
$$\frac{dy}{dt} = -10 \cdot (y - z)$$
$$\frac{dz}{dt} = -x \cdot y + 28y - z$$

- It takes about 10 lines of R code to generate the solutions and plot them.

- Function `scatterplot3d` from the package scatterplot3d generates 3-D scatterplots. Can you recreate the following "butterfly" ? Use as initial conditions $x = y = z = 1$; create output for a time sequence ranging from 0 to 100, and with a time step of 0.005.

[9] The Lotka-Volterra models are a famous type of models that either describe predator-prey interactions or competitive interactions between two species. A.J. Lotka and V. Volterra formulated the original model in the 1920's almost simultaneously ((Lotka 1925), (Volterra 1926)).



10. Finally

10.1. The questions

These lecture notes have been generated with LaTeX and making use of R package **Sweave** (Leisch 2002), which allows to merge LaTeX with R code.

If you do not like the layout a PDF version ("ScientificComputing.pdf") made with WORD (Microsoft) can be found in the `/doc/lecture` subdirectory of package **marelacTeaching**.

10.2. The answers

The answers to the questions in this course are present as an R vignette in package **marelacTeaching**. From within R, type:

```
> vignette("Answers")
```

Or, you can find the file "Answers.pdf" in the `/doc` subdirectory of package **marelacTeaching**.

References

Caswell H (2001). *Matrix population models: construction, analysis, and interpretation*. Sinauer, Sunderland, second edition edition.

Hurlbert SH (1971). "The nonconcept of species diversity: critique and alternative parameters." *Ecology*, **52**, 577–586.

Kuhnert P, Venables W (2005). *An introduction to R: software for statistical modelling & computing*. URL www.r-project.org.

Leisch F (2002). "Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), "Compstat 2002 - Proceedings in Computational Statistics," pp. 575–580. Physica Verlag, Heidelberg. ISBN 3-7908-1517-9, URL http://www.stat.uni-muenchen.de/~leisch/Sweave.

Ligges U, Machler M (2003). "Scatterplot3d - an R Package for Visualizing Multivariate Data." *Journal of Statistical Software*, **8(11)**, 1–20.

Lorenz E (1963). "Deterministic non-periodic flows." *J. Atmos. Sci*, **20**, 130–141.

Lotka AJ (1925). *Elements of Physical Biology*. Williams & Wilkins Co., Baltimore.

Millero F, Poisson A (1981). "International one-atmosphere equation of state for seawater." *Deep-Sea Research*, **28(6)**, 625–629.

Proye A, Gattuso JP, Epitalon JM, Gentili B, Orr J, Soetaert K (2007). *seacarb: Calculates parameters of the seawater carbonate system*. R package version 1.2.3, URL http://www.obs-vlfr.fr/~gattuso/seacarb.php.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Sanders H (1968). "Marine benthic diversity: a comparative study." *Amercian Naturalist*, **102**, 243–282.

Soetaert K (2009). *rootSolve: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations*. R package version 1.4.

Soetaert K, Heip C, Vincx M (1991). "Diversity of nematode assemblages along a mediterranean deep-sea transect." *Marine Ecology Progress Series*, **75**, 275–282.

Soetaert K, Herman PMJ (2009). *A Practical Guide to Ecological Modelling. Using R as a Simulation Platform*. Springer. ISBN 978-1-4020-8623-6.

Soetaert K, Meysman F (2009). *marelacTeaching: Datasets and tutorials for use in the MArine, Riverine, Estuarine, LAcustrine and Coastal sciences*. R package version 1.1.

Soetaert K, Petzoldt T, Meysman F (2009a). *marelac: Constants, conversion factors, utilities for the MArine, Riverine, Estuarine, LAcustrine and Coastal sciences*. R package version 2.0.

Soetaert K, Petzoldt T, Setzer RW (2009b). *deSolve: General solvers for initial value problems of ordinary differential equations (ODE), partial differential equations (PDE) and differential algebraic equations (DAE)*. R package version 1.5.

Verhulst PF (1838). "Notice sur la loi que la population pursuit dans son accroissement." *Correspondance mathematique et physique*, **10**, 113–121.

Volterra V (1926). "Variazioni e fluttuazioni del numero d'individui in specie animali conviventi." *Mem. R. Accad. Naz. dei Lincei. Ser. VI*, **2**, 31–113.

Zeebe R, Wolf-Gladrow D (2003). *CO2 in Seawater: Equilibrium, kinetics, isotopes*. Elsevier.

Affiliation:

Karline Soetaert
Centre for Estuarine and Marine Ecology (CEME)
Netherlands Institute of Ecology (NIOO)
4401 NT Yerseke, Netherlands E-mail: karline.soetaert@nioz.nl
URL: http://www.nioz.nl/