Sprint session: NetCDF functionality in the R statistical software

by Tom Van Engeland & Karline Soetaert

September 14, 2010

Centre for Estuarine and Marine Ecology (CEME)
Netherlands Institute of Ecology (NIOO-KNAW)
P.O.Box 140
4400 AC Yerseke
The Netherlands

e-mail: t.vanengeland@nioo.knaw.nl

Contents

1 The R statistical software **2**
1.1 The R environment . . . 2
1.2 Learning R . . . 2
1.3 Installing R . . . 2
1.4 Talking to R . . . 3
1.5 Some language basics . . . 3
1.5.1 Asking for help . . . 3
1.5.2 R data structures, types, modes, classes . . . 4
1.5.3 Some operators . . . 11
1.5.4 Managing your R environment . . . 12
1.5.5 Importing and exporting data . . . 14
1.5.6 Installing and loading packages . . . 17
1.5.7 Statistics and graphics . . . 17
1.5.8 Writing your own functions . . . 22

2 Using NetCDF files in R **24**
2.1 Reading NetCDF files . . . 24
2.2 Writing NetCDF files . . . 33

3 Remarks **38**

1 The R statistical software

1.1 The R environment

R is a language statistical for computing and graphics, similar to the S language and environment, but open source and freely distributable under the GNU license agreement. It is basically a different implementation of S. A lot of S code can also be used in R (sometimes with some minor corrections).
It is highly extensible through a system of add-on package, and provides a strong platform for scientific computing, and publication, because of the excellent connectivity with database, text processing, GIS and visualization software, and other programming languages (C, C++ and Fortran code). It compiles and runs on a UNIX-like, MacOS and Windows platforms.

1.2 Learning R

As with many open source applications, learning to use the R environment takes somewhat more time than for commercial software packages, since documentation is provided by software developers themselves rather than by a dedicated team, and quality control on these help files is not as severe as for commercial packages [1]. Nevertheless, due to strong efforts of project contributors quite some facilities (search engine, mailing lists, reference manuals, books, R help files, bug reports and fixes, ...) are available to learn how to use R effectively. In addition, due to the large (and ever growing) community of R users, a lot of information on specific problems is available on the internet and is accessed most easily using a standard search engine (posts on mailing lists, forums, small tutorials on specific packages, ...).

1.3 Installing R

Procedures to install R depend on the operating system you use. Windows and MacOS users can install the compiled version after downloading it from CRAN (Comprehensive R Archive Network). Linux users can try to install R through their standard package management tool (Yum, RPM, apt, ...). If the package

[1] Note that this does certainly not hold for the quality and functionality of the software itself

manager does not find the repository, it is probably easiest to look for the packages on the CRAN website. If no suitable packages are found, R has to be compiled and installed from source. For more information we refer to CRAN.

1.4 Talking to R

Communication with R usually occurs through principle of question and answer, much like any text-based command shell (e.g. bash or Windows' command prompt). Commands are interpreted by R and executed. In many cases output is written to the standard output and visualized in the command shell. R comes with a stardard GUI (graphical user interface; Rgui), which allows you to put commands in a script and run either individual commands or an entire script. Plots are by default drawn in a seperate window, but can be written to a specific file format as well.
Although the Rgui is basically the only tool you need to talk to R, other more advanced GUIs and IDEs (integrated development environments) are available with a.o. text highlighting, extra help functionality, and facilities for file manipulation and project building. Examples include

- **Windows**
  - Tinn-R
  - WinEdt + RWinEdt
  - TextPad
- **Unix-based**
  - Kate
  - gedit
- **Multiplatform**
  - Komodo Edit + SciViews-K
  - ESS for Emacs
  - Eclipse
  - jEdit
  - Alpha

1.5 Some language basics

1.5.1 Asking for help

The first most important aspect of R is to know where and how to find the proper information. The most standard help functionality in R, are the R help files that come with any R package. The command `?<command>` will invoke a help page in a browser or in the R console with what `<command>` does, which parameters are available, what kind of output or data is generated, and some examples on how to use the command:

```
> ?(help)
> ?(ls)
> help(ls)
```

A second command is the fuzzy version of `?<command>`, which searches for string matches in help files. The function names are returned:

```
> ??(plot)
> ??(line)
> help.search("line")
```

R help can be launched in a web browser as well. The webpage shows a number of links to manuals, package-specific help files, the R-Site search engine, and other potentially useful material.

```
> help.start()
```

The R Site search engine can also be invoked from the R console by:

```
> RSiteSearch("some mysterious thing you want to known something about")
```

If you want to know more about a particular installed R-package, invoke the command `library(help="some package name")`:

```
> library(help="stats")
```

1.5.2 R data structures, types, modes, classes

Data is in R stored in variables. An assignment of a value to a variable is in R accomplished as follows:

```
> var1 <- "some value"
> counter <<- 1
> temp = TRUE
```

These different assignment operators have slightly different functionality. To learn about them type `help(assignOps)`. To retrieve/view the content of these variables just type the variable name:

```
> var1

[1] "some value"

> counter

[1] 1

> temp

[1] TRUE
```

R knows several data types with an internal storage mode that often has the same name:

types	modes
logical	logical
integer	numeric
double	numeric
complex	complex
character	character
symbol	name
raw	raw
list	list
NULL	NULL
...	

Data of these types are combined in several possible kinds of data structures. There are data structures that can have information of only one single mode (vectors, matrices, arrays) and data structures that can contain information of differnt modes (lists, data frames, externally defined classes).
The simplest kind of variable is a single valued vector:

```
> a <- 1
> a
```

```
[1] 1
```

This vector has length:

```
> length(a)
```

```
[1] 1
```

Obviously a vector can have length larger than 1. Some commands to construct vectors are:

```
> b <- 1:10                         # a sequence
> b
```

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

```
> c <- seq(from=1,to=12,by=0.5) # another sequence
> c
```

```
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0
[18]  9.5 10.0 10.5 11.0 11.5 12.0
```

```
> d <- rep(x=b,times=2)         # repetition of a sequence
> d
```

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

```
> e <- rep(x=b,each=2)          # repetition by element
> e
```

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

```
> f <- c(b,c,d,e)               # concatenation
> f
```

```
 [1]  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0 10.0  1.0  1.5  2.0  2.5  3.0  3.5  4.0
[18]  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0  1.0
[35]  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0 10.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0
[52]  9.0 10.0  1.0  1.0  2.0  2.0  3.0  3.0  4.0  4.0  5.0  5.0  6.0  6.0  7.0  7.0  8.0
[69]  8.0  9.0  9.0 10.0 10.0
```

```
> length(f)
```

```
[1] 73
```

```
> f[9:15]                       # vector subsetting
```

```
[1]  9.0 10.0  1.0  1.5  2.0  2.5  3.0
```

```
> rev(e)                        # vector e in opposite order
```

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

```
> g <- c(FALSE,TRUE)            # a logical vector
> g
```

```
[1] FALSE  TRUE
```

```
> c(g,e)                        # automatic type conversion
```

```
 [1]  0  1  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10
```

```
> as.logical(c(g,e))            # but I want boolean values ...
```

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

One can query variables for their data type and storage mode by:

```
> typeof(f)
```

```
[1] "double"
```

```
> mode(f)
```

```
[1] "numeric"
```

Beside 1-dimensional vectors one can create 2-dimensional vectors, matrices:

```
> mat.a <- matrix(data=1:20,ncol=4)
> mat.a                         # a 5x4 matrix
```

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

```
> dim(mat.a)                    # the matrix dimension (size along the two dimensions)
```

```
[1] 5 4
```

```
> mat.a[1,]                     # the first row
```

```
[1]  1  6 11 16
```

```
> mat.a[,1]                     # the first column
```

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

```
> mat.a[3,2]                    # one specific element by index
```

```
[1] 8
```

```
> mat.a[1:2,1:2]                # a 2x2 subset
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
```

```
> mat.b <- matrix(data=rep(c(F,T,F,T,F,T,F,T,F,T),2),ncol=4)
>                               # a matrix of boolean values
> mat.b
```

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

```
> mat.a[mat.b]                   # selection of matrix elements by boolean values
```

```
 [1]  2  4  6  8 10 12 14 16 18 20
```

```
> colnames(mat.a) <- c("first","second","third","4")
> rownames(mat.a) <- paste("row",1:5,sep="_")
> mat.a
```

```
      first second third  4
row_1     1      6    11 16
row_2     2      7    12 17
row_3     3      8    13 18
row_4     4      9    14 19
row_5     5     10    15 20
```

```
> colnames(mat.b) <- mat.a[1,]  # use the first row of mat.a as column names for mat.b
> colnames(mat.b)               # with automatic type conversion.....
```

```
[1] "1"  "6"  "11" "16"
```

```
> as.numeric(colnames(mat.b))   # type conversion to numeric values
```

```
[1]  1  6 11 16
```

What about more dimensions (3- or n-dimensional):

```
> mat3d <- array(data=1:200,dim=c(25,4,2))
> mat3d
```

```
, , 1

      [,1] [,2] [,3] [,4]
 [1,]    1   26   51   76
 [2,]    2   27   52   77
 [3,]    3   28   53   78
 [4,]    4   29   54   79
 [5,]    5   30   55   80
 [6,]    6   31   56   81
 [7,]    7   32   57   82
 [8,]    8   33   58   83
 [9,]    9   34   59   84
[10,]   10   35   60   85
[11,]   11   36   61   86
[12,]   12   37   62   87
[13,]   13   38   63   88
[14,]   14   39   64   89
[15,]   15   40   65   90
[16,]   16   41   66   91
[17,]   17   42   67   92
[18,]   18   43   68   93
[19,]   19   44   69   94
[20,]   20   45   70   95
[21,]   21   46   71   96
[22,]   22   47   72   97
[23,]   23   48   73   98
[24,]   24   49   74   99
[25,]   25   50   75  100

, , 2

      [,1] [,2] [,3] [,4]
 [1,]  101  126  151  176
 [2,]  102  127  152  177
 [3,]  103  128  153  178
```

```
 [4,]  104  129  154  179
 [5,]  105  130  155  180
 [6,]  106  131  156  181
 [7,]  107  132  157  182
 [8,]  108  133  158  183
 [9,]  109  134  159  184
[10,]  110  135  160  185
[11,]  111  136  161  186
[12,]  112  137  162  187
[13,]  113  138  163  188
[14,]  114  139  164  189
[15,]  115  140  165  190
[16,]  116  141  166  191
[17,]  117  142  167  192
[18,]  118  143  168  193
[19,]  119  144  169  194
[20,]  120  145  170  195
[21,]  121  146  171  196
[22,]  122  147  172  197
[23,]  123  148  173  198
[24,]  124  149  174  199
[25,]  125  150  175  200
```

```
> dim(mat3d)                      # size of the 3-dimensional matrix
```

```
[1] 25  4  2
```

```
> length(mat3d)                   # mat3d is actually a vector with a dimension attribute
```

```
[1] 200
```

```
> mat3d[2:3,1:2,]                 # select rows 2 and 3 of columns 1 and 2 in all "depths"
```

```
, , 1

     [,1] [,2]
[1,]    2   27
[2,]    3   28

, , 2

     [,1] [,2]
[1,]  102  127
[2,]  103  128
```

Sometimes it is convenient to keep data of different types together: lists.

```
> employee1 <- list(name = "Van Engeland",
+                   firstname = "Tom",
+                   shoesize = c(42,43),
+                   hardworking = T,
+                   gamestatus = matrix(rep(c("O","X"),len=9),ncol=3)
+                  )              # lists can contain strings, numbers, booleans, matrices,
>                                 # vectors, ...
>
> employee1
```

```
$name
[1] "Van Engeland"

$firstname
[1] "Tom"
```

```
$shoesize
[1] 42 43

$hardworking
[1] TRUE

$gamestatus
     [,1] [,2] [,3]
[1,] "O"  "X"  "O"
[2,] "X"  "O"  "X"
[3,] "O"  "X"  "O"

> employee1$shoesize            # two different shoes

[1] 42 43

> employee1[[3]] <- NA          # suppose that I do not know the size
>                               # (NA; not available)
> employee1

$name
[1] "Van Engeland"

$firstname
[1] "Tom"

$shoesize
[1] NA

$hardworking
[1] TRUE

$gamestatus
     [,1] [,2] [,3]
[1,] "O"  "X"  "O"
[2,] "X"  "O"  "X"
[3,] "O"  "X"  "O"

> employee1$shoesize <- NULL    # ... or I'm not waring any shoes...
> employee1

$name
[1] "Van Engeland"

$firstname
[1] "Tom"

$hardworking
[1] TRUE

$gamestatus
     [,1] [,2] [,3]
[1,] "O"  "X"  "O"
[2,] "X"  "O"  "X"
[3,] "O"  "X"  "O"
```

Data frames: lists with elements of equal length....

```
> experiment <- data.frame( treatment = rep(c("a","b","c","d"),each=3),
+                           replicates = rep(1:3, times=4),
+                           measurement = c(5,4,6,1,2,4,12,9,15,8,6,10)
+                         )
> experiment
```

```
   treatment replicates measurement
1          a          1           5
2          a          2           4
3          a          3           6
4          b          1           1
5          b          2           2
6          b          3           4
7          c          1          12
8          c          2           9
9          c          3          15
10         d          1           8
11         d          2           6
12         d          3          10
```

```
> experiment$measurement         # one variable (column) from the data frame
```

```
 [1]  5  4  6  1  2  4 12  9 15  8  6 10
```

```
> subset(experiment,             # subsetting in analogy to SQL queries
+        treatment=="a" | treatment =="b",
+        select=c(measurement))
```

```
  measurement
1           5
2           4
3           6
4           1
5           2
6           4
```

```
> factorLevels <- unique(experiment[,1])
>                                # unique values from a column (treated as a vector)
>
> experiment[order(experiment$replicate,experiment$treatment),]
```

```
   treatment replicates measurement
1          a          1           5
4          b          1           1
7          c          1          12
10         d          1           8
2          a          2           4
5          b          2           2
8          c          2           9
11         d          2           6
3          a          3           6
6          b          3           4
9          c          3          15
12         d          3          10
```

```
>                                # sorting according to replicate and then by treatment
>
> experiment[order(experiment$replicate,experiment$treatment),"measurement"]
```

```
 [1]  5  1 12  8  4  2  9  6  6  4 15 10
```

```
>                                # another way of selecting columns (works for rows,
>                                # lists, vectors and matrices as well)
>
> names(experiment)              # get column headers
```

```
[1] "treatment"   "replicates"  "measurement"
```

```
> colnames(experiment)             # same result

[1] "treatment"   "replicates"  "measurement"

> rownames(experiment) <- paste("obs",1:nrow(experiment))
>                                  # you can assign names with all three commands
> rownames(experiment)

 [1] "obs 1"  "obs 2"  "obs 3"  "obs 4"  "obs 5"  "obs 6"  "obs 7"  "obs 8"  "obs 9"
[10] "obs 10" "obs 11" "obs 12"

> experiment

       treatment replicates measurement
obs 1          a          1           5
obs 2          a          2           4
obs 3          a          3           6
obs 4          b          1           1
obs 5          b          2           2
obs 6          b          3           4
obs 7          c          1          12
obs 8          c          2           9
obs 9          c          3          15
obs 10         d          1           8
obs 11         d          2           6
obs 12         d          3          10
```

1.5.3 Some operators

Here we summarize some mathematical operators on numbers, vectors and matrices.

```
> x <- 2:4
> y <- 3:1                         # cbind and rbind paste vectors
>                                  # together to form matrices...
>
> cbind(x,y,  x   +   y)           # addition

     x y
[1,] 2 3 5
[2,] 3 2 5
[3,] 4 1 5

> cbind(x,y,  x   -   y)           # subtraction

     x y
[1,] 2 3 -1
[2,] 3 2  1
[3,] 4 1  3

> cbind(x,y,  x   *   y)           # multiplication

     x y
[1,] 2 3 6
[2,] 3 2 6
[3,] 4 1 4

> cbind(x,y,  x   /   y)           # division

     x y
[1,] 2 3 0.6666667
[2,] 3 2 1.5000000
[3,] 4 1 4.0000000
```

```
> cbind(x,y,  x   ^   y)         # power

    x y
[1,] 2 3 8
[2,] 3 2 9
[3,] 4 1 4

> cbind(x,y,  x   %%  y)         # modulo

    x y
[1,] 2 3 2
[2,] 3 2 1
[3,] 4 1 0

> cbind(x,y,  x  %/%  y)         # integer division

    x y
[1,] 2 3 0
[2,] 3 2 1
[3,] 4 1 4

> x  %*%  y                      # dot product / inner product

     [,1]
[1,]   16

> x  %*%  t(y)                   # outer product, using the transpose of y

     [,1] [,2] [,3]
[1,]    6    4    2
[2,]    9    6    3
[3,]   12    8    4

> matx <- matrix(1:9,nrow=3,byrow=T)
> maty <- matrix(5:13,nrow=3,byrow=T)
> matx * maty                    # multiplication by element

     [,1] [,2] [,3]
[1,]    5   12   21
[2,]   32   45   60
[3,]   77   96  117

> matx %*% maty                  # matrix multiplication

     [,1] [,2] [,3]
[1,]   54   60   66
[2,]  126  141  156
[3,]  198  222  246
```

Other unary operators include `abs()`, `sign`, `sqrt`, ... More information can be obtain by typing `?Arithmetic`, `?Special`, and `?Syntax` in the R console.

1.5.4 Managing your R environment

To see which objects you have created in the current environment, the `ls` function without arguments can be used. If a function is supplied as first argument, `ls` returns the local variables in this function.

```
> ls()                           # get all objects in the current environments
```

```
 [1] "a"                  "b"                          "c"
 [4] "centre"             "CNratio"                    "CNratioExt"
 [7] "coastalData"        "counter"                    "crs_var"
[10] "d"                  "depthProfile"               "Dmodel"
[13] "e"                  "employee1"                  "ETM3"
[16] "ETM4"               "experiment"                 "f"
[19] "factorLevels"       "g"                          "getInfo"
[22] "grid.nc"            "i"                          "iterator"
[25] "linearmodel"        "makeRweaveLatexCodeRunner" "mat.a"
[28] "mat.b"              "mat3d"                      "matx"
[31] "maty"               "NDVI"                       "ndvi_var"
[34] "ndvi.nc"            "Noordwijk"                  "NoordwijkTransect"
[37] "nutrientdata"       "Pmodel"                     "rawresiduals"
[40] "residualvalues"     "Rtangle"                    "RtangleFinish"
[43] "RtangleRuncode"     "RtangleSetup"               "RtangleWritedoc"
[46] "RweaveChunkPrefix"  "RweaveEvalWithOpt"          "RweaveLatex"
[49] "RweaveLatexFinish"  "RweaveLatexOptions"         "RweaveLatexRuncode"
[52] "RweaveLatexSetup"   "RweaveLatexWritedoc"        "RweaveTryStop"
[55] "selection"          "sequence"                   "Stangle"
[58] "Sweave"             "SweaveGetSyntax"            "SweaveHooks"
[61] "SweaveParseOptions" "SweaveReadFile"             "SweaveSyntaxLatex"
[64] "SweaveSyntaxNoweb"  "SweaveSyntConv"             "temp"
[67] "time"               "transect"                   "url_grid"
[70] "var.add.ncdf"       "var.def.ncdf.no_dim"        "var1"
[73] "x"                  "x_dim"                      "y"
[76] "y_dim"              "z"
```

R has two search functions to lists all the objects in your environment, matching a search string: `find` and `apropos`. They basically perform the same job, but have slightly different parameterization.

```
> apropos("line")               # returns objects containing "line" in their name
```

One can remove objects by means of the `rm` function. Objects to remove can be provided by naming them at the beginning of the parameter list, or as a list or as a character vector.

```
> ls()
```

```
 [1] "a"                  "b"                          "c"
 [4] "centre"             "CNratio"                    "CNratioExt"
 [7] "coastalData"        "counter"                    "crs_var"
[10] "d"                  "depthProfile"               "Dmodel"
[13] "e"                  "employee1"                  "ETM3"
[16] "ETM4"               "experiment"                 "f"
[19] "factorLevels"       "g"                          "getInfo"
[22] "grid.nc"            "i"                          "iterator"
[25] "linearmodel"        "makeRweaveLatexCodeRunner" "mat.a"
[28] "mat.b"              "mat3d"                      "matx"
[31] "maty"               "NDVI"                       "ndvi_var"
[34] "ndvi.nc"            "Noordwijk"                  "NoordwijkTransect"
[37] "nutrientdata"       "Pmodel"                     "rawresiduals"
[40] "residualvalues"     "Rtangle"                    "RtangleFinish"
[43] "RtangleRuncode"     "RtangleSetup"               "RtangleWritedoc"
[46] "RweaveChunkPrefix"  "RweaveEvalWithOpt"          "RweaveLatex"
[49] "RweaveLatexFinish"  "RweaveLatexOptions"         "RweaveLatexRuncode"
[52] "RweaveLatexSetup"   "RweaveLatexWritedoc"        "RweaveTryStop"
[55] "selection"          "sequence"                   "Stangle"
[58] "Sweave"             "SweaveGetSyntax"            "SweaveHooks"
[61] "SweaveParseOptions" "SweaveReadFile"             "SweaveSyntaxLatex"
[64] "SweaveSyntaxNoweb"  "SweaveSyntConv"             "temp"
[67] "time"               "transect"                   "url_grid"
[70] "var.add.ncdf"       "var.def.ncdf.no_dim"        "var1"
[73] "x"                  "x_dim"                      "y"
[76] "y_dim"              "z"
```

```
> rm(a)                          # remove the vector "a"
> ls()
```

```
 [1] "b"                          "c"                     "centre"
 [4] "CNratio"                    "CNratioExt"            "coastalData"
 [7] "counter"                    "crs_var"               "d"
[10] "depthProfile"               "Dmodel"                "e"
[13] "employee1"                  "ETM3"                  "ETM4"
[16] "experiment"                 "f"                     "factorLevels"
[19] "g"                          "getInfo"               "grid.nc"
[22] "i"                          "iterator"              "linearmodel"
[25] "makeRweaveLatexCodeRunner" "mat.a"                 "mat.b"
[28] "mat3d"                      "matx"                  "maty"
[31] "NDVI"                       "ndvi_var"              "ndvi.nc"
[34] "Noordwijk"                  "NoordwijkTransect"     "nutrientdata"
[37] "Pmodel"                     "rawresiduals"          "residualvalues"
[40] "Rtangle"                    "RtangleFinish"         "RtangleRuncode"
[43] "RtangleSetup"               "RtangleWritedoc"       "RweaveChunkPrefix"
[46] "RweaveEvalWithOpt"          "RweaveLatex"           "RweaveLatexFinish"
[49] "RweaveLatexOptions"         "RweaveLatexRuncode"    "RweaveLatexSetup"
[52] "RweaveLatexWritedoc"        "RweaveTryStop"         "selection"
[55] "sequence"                   "Stangle"               "Sweave"
[58] "SweaveGetSyntax"            "SweaveHooks"           "SweaveParseOptions"
[61] "SweaveReadFile"             "SweaveSyntaxLatex"     "SweaveSyntaxNoweb"
[64] "SweaveSyntConv"             "temp"                  "time"
[67] "transect"                   "url_grid"              "var.add.ncdf"
[70] "var.def.ncdf.no_dim"        "var1"                  "x"
[73] "x_dim"                      "y"                     "y_dim"
[76] "z"
```

```
rm(list=ls())                  # remove all objects listed by ls
ls()                           # i.e. all objects in the current environment
```

```
character(0)
```

1.5.5 Importing and exporting data

Reading and writing data to files provides a flexible means of data exchange between different software application such as spreadsheet programmes. Many programmes are able to handle text files. Here this is demonstrated for two file types. For both read and write commands exist. By changing parameters we can choose separators, quotation, header inclusion, ...

```
> # read data from a tab-delimited file
> nutrientdata <- read.table(file="Data/nutrientdata.txt",sep="\t",header=T)
> head(nutrientdata,5)           # show the first 5 lines
```

```
  stID     station     date year month    lon     lat distance      NH4      NO3
1  558 Noordwijk 2 1992.000 1992     1 4.4048 52.2606        2 4.497847 51.40397
2  558 Noordwijk 2 1992.083 1992     2 4.4048 52.2606        2 8.495934 69.18118
3  558 Noordwijk 2 1992.167 1992     3 4.4048 52.2606        2 5.854341 71.10883
4  558 Noordwijk 2 1992.250 1992     4 4.4048 52.2606        2 3.569720 79.96173
5  558 Noordwijk 2 1992.333 1992     5 4.4048 52.2606        2 4.569242 43.12222
        DON      DOC Chla      PON      POC SPM
1 10.709161 113.2324 0.82 6.496891 71.60282  27
2  8.353145 101.5761 0.98 7.639201 74.10059  24
3  4.854819 108.2368 2.46 3.998087 41.62955  11
4 10.709161 129.8842 3.00 3.783903 34.96882  11
5 20.418800 147.3686 8.50 9.209878 49.95546   5
```

```
> # obtain a subset using regular expression functionality (cf. ?grep)
> NoordwijkTransect <- nutrientdata[grep("Noordwijk",nutrientdata$station),]
> # write the transect data to a comma-delimited file
> write.csv(x=NoordwijkTransect,file="Data/NoordwijkTransect.csv",row.names=F,quote=F)
```

It is also possible to write data objects or an entire environment to a R-specific binary data format (.RData):

```
>                                    # obtain a subset of coastal stations
>                                    # list all data objects
>                                    # save the subsets of nutrient data to a file
> coastalData <- subset(nutrientdata, distance < 10)
> ls()

 [1] "b"                          "c"                     "centre"
 [4] "CNratio"                    "CNratioExt"            "coastalData"
 [7] "counter"                    "crs_var"               "d"
[10] "depthProfile"               "Dmodel"                "e"
[13] "employee1"                  "ETM3"                  "ETM4"
[16] "experiment"                 "f"                     "factorLevels"
[19] "g"                          "getInfo"               "grid.nc"
[22] "i"                          "iterator"              "linearmodel"
[25] "makeRweaveLatexCodeRunner" "mat.a"                 "mat.b"
[28] "mat3d"                      "matx"                  "maty"
[31] "NDVI"                       "ndvi_var"              "ndvi.nc"
[34] "Noordwijk"                  "NoordwijkTransect"     "nutrientdata"
[37] "Pmodel"                     "rawresiduals"          "residualvalues"
[40] "Rtangle"                    "RtangleFinish"         "RtangleRuncode"
[43] "RtangleSetup"               "RtangleWritedoc"       "RweaveChunkPrefix"
[46] "RweaveEvalWithOpt"          "RweaveLatex"           "RweaveLatexFinish"
[49] "RweaveLatexOptions"         "RweaveLatexRuncode"    "RweaveLatexSetup"
[52] "RweaveLatexWritedoc"        "RweaveTryStop"         "selection"
[55] "sequence"                   "Stangle"               "Sweave"
[58] "SweaveGetSyntax"            "SweaveHooks"           "SweaveParseOptions"
[61] "SweaveReadFile"             "SweaveSyntaxLatex"     "SweaveSyntaxNoweb"
[64] "SweaveSyntConv"             "temp"                  "time"
[67] "transect"                   "url_grid"              "var.add.ncdf"
[70] "var.def.ncdf.no_dim"        "var1"                  "x"
[73] "x_dim"                      "y"                     "y_dim"
[76] "z"

> save(coastalData,NoordwijkTransect,file="Data/subsets.RData")
> rm(coastalData,NoordwijkTransect)     # remove the objects created
> ls()                                  # the result ....

 [1] "b"                          "c"                     "centre"
 [4] "CNratio"                    "CNratioExt"            "counter"
 [7] "crs_var"                    "d"                     "depthProfile"
[10] "Dmodel"                     "e"                     "employee1"
[13] "ETM3"                       "ETM4"                  "experiment"
[16] "f"                          "factorLevels"          "g"
[19] "getInfo"                    "grid.nc"               "i"
[22] "iterator"                   "linearmodel"           "makeRweaveLatexCodeRunner"
[25] "mat.a"                      "mat.b"                 "mat3d"
[28] "matx"                       "maty"                  "NDVI"
[31] "ndvi_var"                   "ndvi.nc"               "Noordwijk"
[34] "nutrientdata"               "Pmodel"                "rawresiduals"
[37] "residualvalues"             "Rtangle"               "RtangleFinish"
[40] "RtangleRuncode"             "RtangleSetup"          "RtangleWritedoc"
[43] "RweaveChunkPrefix"          "RweaveEvalWithOpt"     "RweaveLatex"
[46] "RweaveLatexFinish"          "RweaveLatexOptions"    "RweaveLatexRuncode"
[49] "RweaveLatexSetup"           "RweaveLatexWritedoc"   "RweaveTryStop"
[52] "selection"                  "sequence"              "Stangle"
[55] "Sweave"                     "SweaveGetSyntax"       "SweaveHooks"
[58] "SweaveParseOptions"         "SweaveReadFile"        "SweaveSyntaxLatex"
[61] "SweaveSyntaxNoweb"          "SweaveSyntConv"        "temp"
[64] "time"                       "transect"              "url_grid"
[67] "var.add.ncdf"               "var.def.ncdf.no_dim"   "var1"
```

```
[70] "x"                          "x_dim"                       "y"
[73] "y_dim"                      "z"
```

```
> load(file="Data/subsets.RData")       # load the data file from the working directory
> ls()                                  # the result ....
```

```
 [1] "b"                          "c"                           "centre"
 [4] "CNratio"                    "CNratioExt"                  "coastalData"
 [7] "counter"                    "crs_var"                     "d"
[10] "depthProfile"               "Dmodel"                      "e"
[13] "employee1"                  "ETM3"                        "ETM4"
[16] "experiment"                 "f"                           "factorLevels"
[19] "g"                          "getInfo"                     "grid.nc"
[22] "i"                          "iterator"                    "linearmodel"
[25] "makeRweaveLatexCodeRunner"  "mat.a"                       "mat.b"
[28] "mat3d"                      "matx"                        "maty"
[31] "NDVI"                       "ndvi_var"                    "ndvi.nc"
[34] "Noordwijk"                  "NoordwijkTransect"           "nutrientdata"
[37] "Pmodel"                     "rawresiduals"                "residualvalues"
[40] "Rtangle"                    "RtangleFinish"               "RtangleRuncode"
[43] "RtangleSetup"               "RtangleWritedoc"             "RweaveChunkPrefix"
[46] "RweaveEvalWithOpt"          "RweaveLatex"                 "RweaveLatexFinish"
[49] "RweaveLatexOptions"         "RweaveLatexRuncode"          "RweaveLatexSetup"
[52] "RweaveLatexWritedoc"        "RweaveTryStop"               "selection"
[55] "sequence"                   "Stangle"                     "Sweave"
[58] "SweaveGetSyntax"            "SweaveHooks"                 "SweaveParseOptions"
[61] "SweaveReadFile"             "SweaveSyntaxLatex"           "SweaveSyntaxNoweb"
[64] "SweaveSyntConv"             "temp"                        "time"
[67] "transect"                   "url_grid"                    "var.add.ncdf"
[70] "var.def.ncdf.no_dim"        "var1"                        "x"
[73] "x_dim"                      "y"                           "y_dim"
[76] "z"
```

```
> save.image(file="Data/all.RData")     # save all objects in the global environment
> rm(list=ls())                         # remove all objects from the environment
> ls()                                  # the result ....
```

```
character(0)
```

```
> load(file="Data/all.RData")           # load the environment again
> ls()                                  # objects loaded ....
```

```
 [1] "b"                          "c"                           "centre"
 [4] "CNratio"                    "CNratioExt"                  "coastalData"
 [7] "counter"                    "crs_var"                     "d"
[10] "depthProfile"               "Dmodel"                      "e"
[13] "employee1"                  "ETM3"                        "ETM4"
[16] "experiment"                 "f"                           "factorLevels"
[19] "g"                          "getInfo"                     "grid.nc"
[22] "i"                          "iterator"                    "linearmodel"
[25] "makeRweaveLatexCodeRunner"  "mat.a"                       "mat.b"
[28] "mat3d"                      "matx"                        "maty"
[31] "NDVI"                       "ndvi_var"                    "ndvi.nc"
[34] "Noordwijk"                  "NoordwijkTransect"           "nutrientdata"
[37] "Pmodel"                     "rawresiduals"                "residualvalues"
[40] "Rtangle"                    "RtangleFinish"               "RtangleRuncode"
[43] "RtangleSetup"               "RtangleWritedoc"             "RweaveChunkPrefix"
[46] "RweaveEvalWithOpt"          "RweaveLatex"                 "RweaveLatexFinish"
[49] "RweaveLatexOptions"         "RweaveLatexRuncode"          "RweaveLatexSetup"
[52] "RweaveLatexWritedoc"        "RweaveTryStop"               "selection"
[55] "sequence"                   "Stangle"                     "Sweave"
[58] "SweaveGetSyntax"            "SweaveHooks"                 "SweaveParseOptions"
[61] "SweaveReadFile"             "SweaveSyntaxLatex"           "SweaveSyntaxNoweb"
[64] "SweaveSyntConv"             "temp"                        "time"
```

```
[67] "transect"              "url_grid"              "var.add.ncdf"
[70] "var.def.ncdf.no_dim"   "var1"                  "x"
[73] "x_dim"                 "y"                     "y_dim"
[76] "z"
```

1.5.6 Installing and loading packages

All R functions and datasets are stored in packages. Only when they are loaded their content is available. This way optional code and attached documentation is loaded and taking memory only when it is needed. The R distribution itself includes about 25 packages.
Other packages are generally managed and available on the CRAN website. One can install binary packages directly from CRAN by issueing the following command:

```
# install the nlme package (mixed models) from a CRAN mirror
> install.packages("nlme",dependencies=T)
```

First R asks to select a mirror site to download from. This will by default remain the same for the rest of the R session. After installation, packages can be loaded via the command:

```
> library(nlme)
```

An alternative function is `require`, which issues a warning when the package is not found, rather than a error as `library` does. Require is mainly intended to be called inside other functions, to allow for exception handling and a decent termination of the relevant function.
Other functionalities include updating, removing and listing packages (update.packages, available.packages, installed.packages, remove.packages). We refer to the help files for more information.

1.5.7 Statistics and graphics

Now that we know how to import data and handle objects we can start to analyse them. A large part of R's strength lies in the flexibility with which publication quality graphs can be produced.
A simple scatterplot of dissolved organic carbon data as a function of dissolved organic nitrogen data at a coastal station:

```
> Noordwijk <- subset(coastalData, station=="Noordwijk 2", select=c(DON,DOC))
> plot(Noordwijk$DON,Noordwijk$DOC,
+      main="Noordwijk coast", xlab="DON",ylab="DOC",type="p",pch=19,cex=0.5,col="blue")
> text(x=7,y=200,label="A TREND !!!")
```

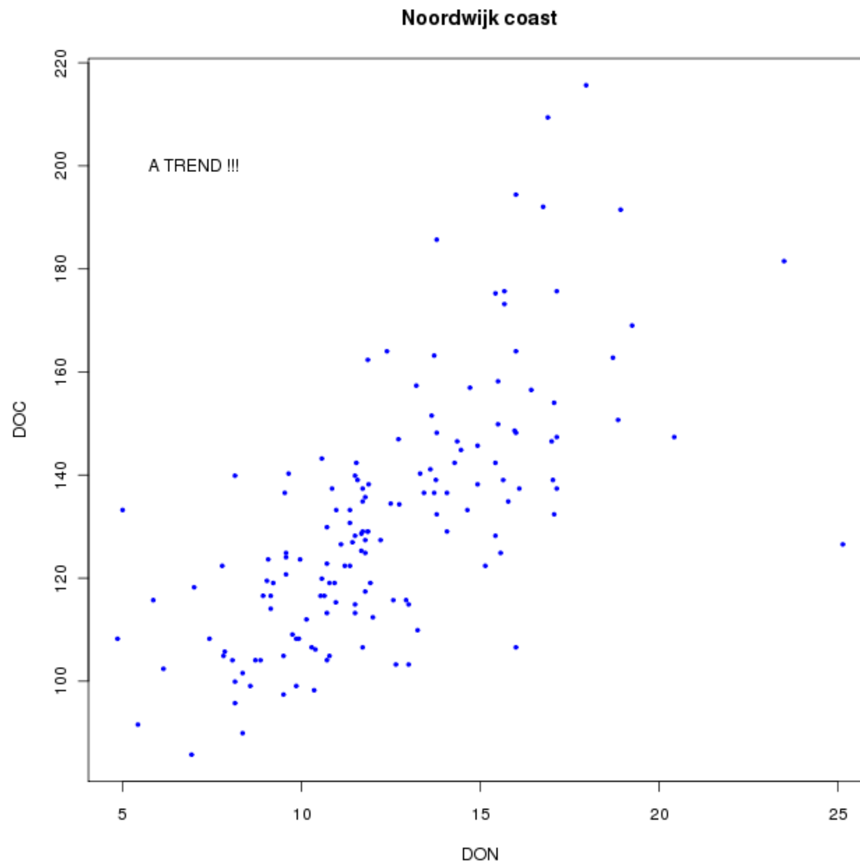

Figure 1: Result of the simple `plot` function.

We could have a look at the particulate component of the organic matter as well. To plot both of them the following code is run:

```
>                                         # we need to add PON and POC to our subset
> Noordwijk <- subset(coastalData, station=="Noordwijk 2", select=c(DON,DOC,PON,POC))
> par(mfrow=c(1,2))
> plot(Noordwijk$DON,Noordwijk$DOC,
+     main="dissolved", xlab="DON",ylab="DOC",type="p",pch=19,cex=0.5,col="blue")
> plot(Noordwijk$PON,Noordwijk$POC,
+     main="particulate", xlab="PON",ylab="POC",type="p",pch=19,cex=0.5,col="blue")
> mtext(side=1,outer=T,line= 1, at=0.5,text="organic matter C/N ratios")
```

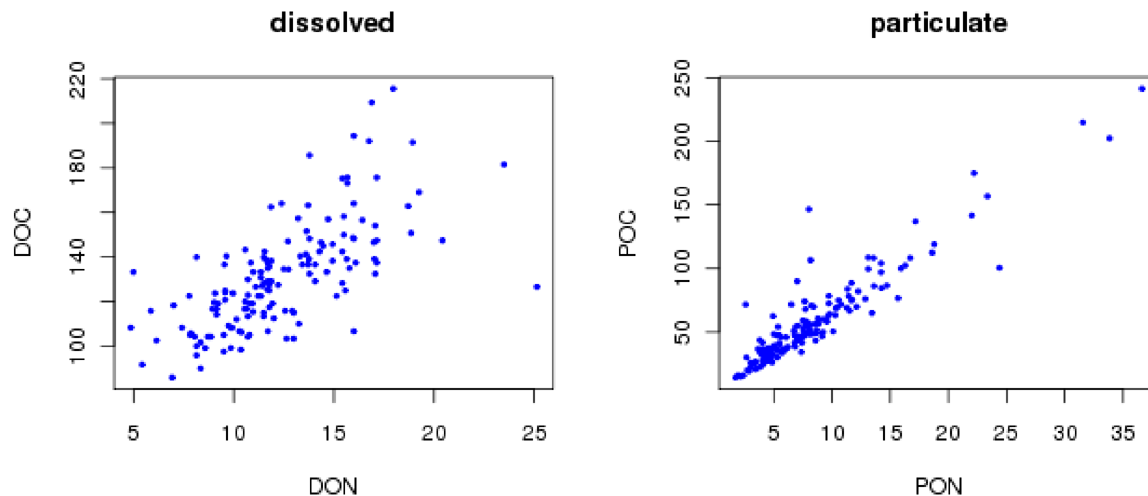

Figure 2: Two scatterplot on a same window device with a margin text.

Since the plots show a clear linearity, we could think of doing a linear regression analysis. For now we will disregard the interdependence of the residuals (time series). The function `lm` performs a linear regression analysis and returns the output. The object can be queried using a set of generic functions that are often redefined in different packages and for different classes of objects. We only demonstrate `summary` and `plot`. Furthermore, anova tables are often generated for linear model. This is possible in R with the `anova` function, which could have different functionality in different situations as well (cf. ?anova.lme, after loading the package "nlme").

```
> linearmodel <- lm(DOC ~ DON, data=Noordwijk) # construction of a linear model object
> linearmodel                                  # this output is usually not enough....

Call:
lm(formula = DOC ~ DON, data = Noordwijk)

Coefficients:
(Intercept)          DON
     73.072        4.708

> names(linearmodel)                           # what does this object contain?

 [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values"
 [6] "assign"        "qr"            "df.residual"   "na.action"     "xlevels"
[11] "call"          "terms"         "model"

> summary(linearmodel)                         # summarize the relevant output of the

Call:
lm(formula = DOC ~ DON, data = Noordwijk)

Residuals:
      Min        1Q    Median        3Q       Max
-64.83509 -11.03870  -0.02278   7.07841  58.03300

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  73.0719     5.3849   13.57   <2e-16 ***
```

```
DON           4.7080     0.4197   11.22   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 17.65 on 148 degrees of freedom
  (30 observations deleted due to missingness)
Multiple R-squared: 0.4595,       Adjusted R-squared: 0.4558
F-statistic: 125.8 on 1 and 148 DF,  p-value: < 2.2e-16

>                                              # linear regression (e.g. t-table)
> coef(linearmodel)                            # to extract the coefficients or....

(Intercept)         DON
  73.071870    4.708043

> coef(summary(linearmodel))                   # to extract the entire t-table

             Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 73.071870  5.3849100 13.56975 9.197175e-28
DON          4.708043  0.4197349 11.21671 1.618762e-21

> anova(linearmodel)                           # to extract the anova table

Analysis of Variance Table

Response: DOC
           Df Sum Sq Mean Sq F value    Pr(>F)
DON         1  39204   39204  125.81 < 2.2e-16 ***
Residuals 148  46118     312
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Plotting some diagnostics on the linear regression residuals (graphs not shown here):

```
> plot(linearmodel)
```

We could have a look at the normality of the residuals:

```
> rawresiduals <- resid(linearmodel)        # linearmodel$residuals works just as well but
>                                           # using a dedicated function is preferable
>                                           # when accessing objects to keep programming
>                                           # transparent and structured
>
> residualvalues <- rstandard(linearmodel) # standardized residuals
> par(mfrow=c(2,2), mar=c(4,4,1,0.1))      # we produce 4 graphs with altered margins
> plot(linearmodel$model$DON,residualvalues,xlab="independent",ylab="residuals")
> plot(residualvalues, main="residuals in order of appearance")
> qqnorm(residualvalues)                    # normal Q-Q plot
> abline(a=0,b=1,col="red")                 # with theoretically expected line y=a+bx
> hist(residualvalues)                      # a histogram of the residuals
```

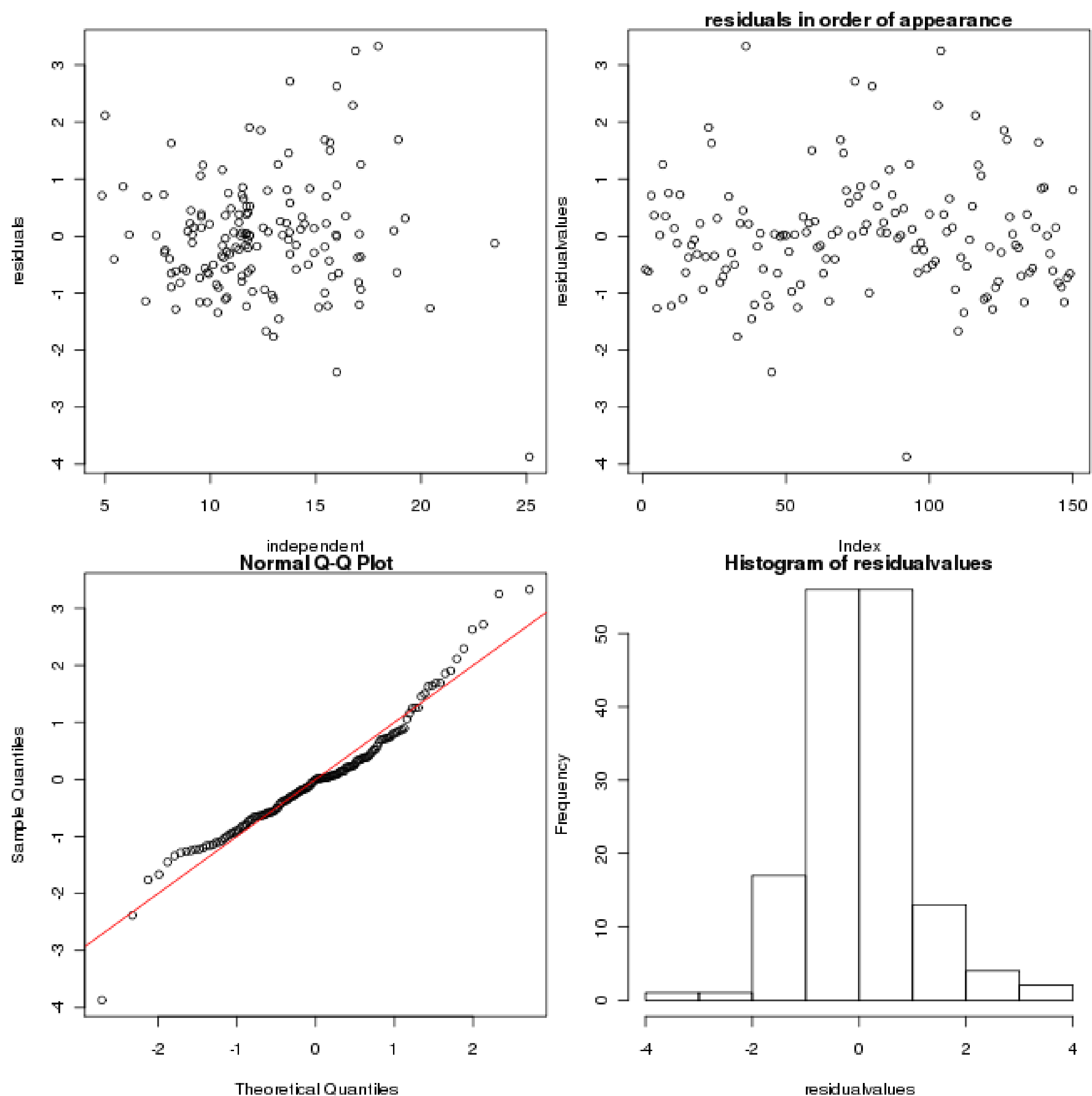

Figure 3: Diagnostics for the residuals of the regression analysis.

Plotting the regression lines on the scatterplots.

```
> Dmodel <- lm(DOC ~ DON, data=Noordwijk)
> Pmodel <- lm(POC ~ PON, data=Noordwijk)
> par(mfrow=c(1,2))
> plot(Noordwijk$DON,Noordwijk$DOC,
+      main="dissolved", xlab="DON",ylab="DOC",type="p",pch=19,cex=0.5,col="blue")
> abline(coef(Dmodel),col="red",lwd=2)
> plot(Noordwijk$PON,Noordwijk$POC,
+      main="particulate", xlab="PON",ylab="POC",type="p",pch=19,cex=0.5,col="blue")
```

```
> abline(coef(Pmodel),col="red",lwd=2)
> mtext(side=1,outer=T,line= 1, at=0.5,text="organic matter C/N ratios")
```

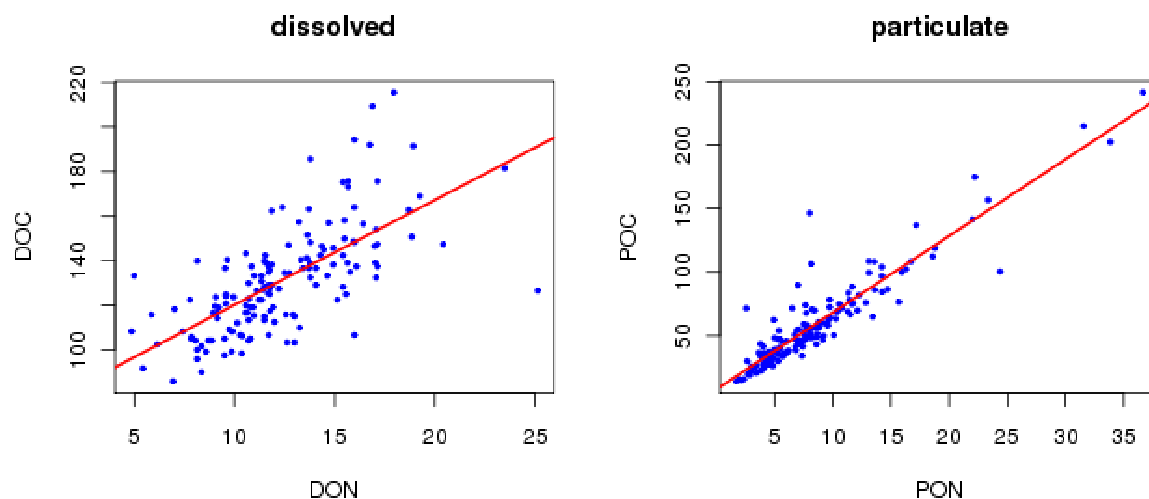

Figure 4: Putting the regression lines on the scatterplots, using the linear models.

1.5.8 Writing your own functions

Using the `function` command one can write his/her own functions. A number of parameters can be defined within braces, followed by a code block ( ). An object can be returned by the function, either as the last output/object generated in the function, or an object returned by the `return` command. The return command allows to exit the function without having to reach the last line of the code block.

```
>                                       # We define a function named CNratio that
>                                       # calculates the carbon-to-nitrogen ratio
>                                       # of the input parameters
> CNratio <- function(carbon, nitrogen) {
+                 carbon/nitrogen
+            }
>                                       # We now extend this to include an N/C calculation
>                                       # in addition to the C/N calculation
> CNratioExt <- function(carbon, nitrogen, inverse=F) {
+               if(!(is.vector(carbon) & is.vector(nitrogen))) {
+                 print("This function only accepts vectors")
+                 return()
+               }                       # Test if carbon and nitrogen are vectors
+               if(length(carbon) != length(nitrogen)) {
+                 print("carbon and nitrogen dataset not of same size")
+                 return()
+               }                       # We check the equality of their lengths
+               if(inverse) {
+                  return(nitrogen/carbon)
+               }
```

```
+               else {
+                  return(carbon/nitrogen)
+               }
+            }
> CNratio(carbon=rep(16,10),nitrogen=3:12)

 [1] 5.333333 4.000000 3.200000 2.666667 2.285714 2.000000 1.777778 1.600000 1.454545
[10] 1.333333

> CNratioExt(carbon=rep(16,10),nitrogen=3:12)

 [1] 5.333333 4.000000 3.200000 2.666667 2.285714 2.000000 1.777778 1.600000 1.454545
[10] 1.333333

> CNratioExt(carbon=rep(16,10),nitrogen=3:12,inverse=T)

 [1] 0.1875 0.2500 0.3125 0.3750 0.4375 0.5000 0.5625 0.6250 0.6875 0.7500

> CNratioExt(carbon=rep(16,10),nitrogen=3:13,inverse=T)

[1] "carbon and nitrogen dataset not of same size"
NULL

> CNratioExt(carbon=rep(16,10),nitrogen=data.frame(nitrogen=3:12),inverse=T)

[1] "This function only accepts vectors"
NULL
```

In this piece of code a first control-flow structure is presented: `if() expr else expr`. Other control-flow constructs are `for(var in seq) expr`,`while(cond) expr`,and `repeat expr`. Information on these constructs is available with `?Control`.

```
> sequence <- 1:10
> for(iterator in sequence) print(iterator)

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

> i <- 1
> while(i <= max(sequence)) {
+    print(i)
+    i <- i + 1
+ }

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

```
> i <- 1
> repeat  {
+    print(i)
+    i <- i + 1
+    if(i > 10) break
+ }

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

> centre <- function(x, type) {                        # a function using the switch construct
+              switch(type,
+                mean = mean(x),
+                median = median(x),
+                trimmed = mean(x, trim = .1))
+            }
> centre(x=c(1:10,20),type="mean")                     # some results

[1] 6.818182

> centre(x=c(1:10,20),type="median")

[1] 6
```

2 Using NetCDF files in R

2.1 Reading NetCDF files

To use the NetCDF functionality in R we use the ncdf packages which has to be loaded every time R is started. Installation in Windows is accomplished by `install.packages("ncdf",dependencies=T)`.

```
> library(ncdf)
> library(fields)
```

Suppose we are interested in bathymetry data from the Oosterschelde (The Netherlands). The Open Earth project provides such a dataset at the following url: http://opendap.deltares.nl/opendap/rijkswaterstaat/vaklodingen/. The relevant file is downloadable straight from the R environment:

```
url_grid <-  "http://opendap.deltares.nl:8080/thredds/dodsC/opendap/
                       rijkswaterstaat/vaklodingen/vaklodingenKB117_4746.nc"
#download.file(url=url_grid, destfile="../vaklodingenKB117_4746.nc",
#              method =  "auto", quiet = FALSE, mode="wb", cacheOK = TRUE)
grid.nc <- open.ncdf("../vaklodingenKB117_4746.nc")

> grid.nc                         # for a first look

[1] "file ../vaklodingenKB117_4746.nc has 3 dimensions:"
[1] "time   Size: 60"
[1] "y   Size: 625"
```

```
[1] "x   Size: 500"
[1] "------------------------"
[1] "file ../vaklodingenKB117_4746.nc has 4 variables:"
[1] "int crs[]  Longname:crs Missval:NA"
[1] "double z[x,y,time]  Longname:z Missval:1e+30"
[1] "double lon[x,y]  Longname:lon Missval:1e+30"
[1] "double lat[x,y]  Longname:lat Missval:1e+30"
```

This gives us the most important information. We have three dimensions: x (of length 625), y (of length 500), and time (of length 60). There are 4 variables of which one is 0-dimensional (crs), two are 2-dimensional (625 x 500 = 312500 observations), and one is 3-dimensional dataset (625 x 500 x 60 = 18.75 x 10^6 observations). But what about the side (meta-)information? How do we access the data? `class(grid.nc)` shows that this is an object of the class "ncdf", but `typeof(grid.nc)` tells us that we are essentially working with a list. Hence, we could see what is in this list by using the `names` and `str` command:

```
> names(grid.nc)
```

```
 [1] "id"           "ndims"        "natts"        "unlimdimid"   "filename"
 [6] "varid2Rindex" "writable"     "dim"          "nvars"        "var"
```

Each object of the class ncdf will contain data substructures with these names. To see what the names stand for you could look in the help files of the `open.ncdf` command (only accessible, once the package is loaded). Here we give a short overview.

name	description
id	a file id
ndims	number of dimensions
natts	number of global attributes
unlimdimid	id of unlimited dimension or -1 if not present
filename	file name on disk
writable	file opened for writing of not
dim	dimensions (named list of objects of class dim.ncdf)
nvars	number of non-coordinate variables
var	variables (named list of objects of class var.ncdf)

The ncdf package is a bit limited in the sence that it can be difficult to obtain a good overview of what is exactly described in a NetCDF file. First of all, the file has to be downloaded before one can retrieve information from the file, since OpenDAP (Open-source Project for a Network Data Access Protocol) capability is currently not supported in this package. Secondly, one can query an attribute, but there is no clear way of getting an overview of which attributes are present. However, if the file has to be downloaded from an OpenDAP server, one can use the url followed by a certain suffix to retrieve a description of the datastructure, including the attributes (e.g. url.das). This can be done via your web browser or can be directly imported in R using the following function.

```
> getInfo <- function(url,what=c("das","dds","asc","html","dods")){
+   urlConnection <- url(paste(url,what,sep="."))
+   if(what=="asc") {
+     temp <- read.table(urlConnection,fill=T,sep=",")
+   }
+   if(what=="dds" | what=="das" | what=="html") {
+     temp <- cat(paste(readLines(urlConnection),"\n"))
+   }
+   if(what=="dods") {
+     filename <- strsplit(url,"/")[[1]]; filename <- filename[length(filename)]
```

```
+     download.file(urlConnection, filename, method =  "auto",
+                    quiet = FALSE, mode="wb", cacheOK = TRUE)
+   }
+   return(temp)
+ }
```

This function allows you to retrieve information on the file in the das, dds, and html format. Furthermore, it combined retrieval of information on the file with retrieval of the file itself, either in binary (NetCDF; dods) format or in ASCII format. In the latter case the data is stored in memory, whereas in the binary case the data is stored on disk. The function is used as follows:
For the das (attribute information) format:

```
> getInfo(url_grid,what="das")

Attributes {
    crs {
        String spatial_ref "PROJCS[\"Amersfoort / RD New\",
    GEOGCS[\"Amersfoort\",
        DATUM[\"Amersfoort\",
            SPHEROID[\"Bessel 1841\",6377397.155,299.1528128,
                AUTHORITY[\"EPSG\",\"7004\"]],
            AUTHORITY[\"EPSG\",\"6289\"]],
        PRIMEM[\"Greenwich\",0,
            AUTHORITY[\"EPSG\",\"8901\"]],
        UNIT[\"degree\",0.01745329251994328,
            AUTHORITY[\"EPSG\",\"9122\"]],
        AUTHORITY[\"EPSG\",\"4289\"]],
    UNIT[\"metre\",1,
        AUTHORITY[\"EPSG\",\"9001\"]],
    PROJECTION[\"Oblique_Stereographic\"],
    PARAMETER[\"latitude_of_origin\",52.15616055555555],
    PARAMETER[\"central_meridian\",5.38763888888889],
    PARAMETER[\"scale_factor\",0.9999079],
    PARAMETER[\"false_easting\",155000],
    PARAMETER[\"false_northing\",463000],
    AUTHORITY[\"EPSG\",\"28992\"],
    AXIS[\"X\",EAST],
    AXIS[\"Y\",NORTH]]";
        Float64 latitude_of_projection_origin 52.15616055555555;
        Float64 false_easting 155000.0;
        Float64 false_northing 463000.0;
        Float64 scale_factor 0.9999079;
        Float64 central_meridian 5.387638888888889;
        String grid_mapping_name "Oblique_Stereographic";
        String comment "value set to the EPSG code";
    }
    lat {
        String standard_name "latitude";
        String units "degrees_north";
    }
    z {
        String grid_mapping "crs";
        Float64 _FillValue -9999.0;
        String positive "up";
        String coordinates "lat lon";
        String standard_name "altitude";
        String units "meter";
    }
    lon {
        String standard_name "longitude";
        String units "degrees_east";
    }
    y {
```

```
        String actual_range "387500.0 400000.0";
        String standard_name "projection_y_coordinate";
        String units "meter";
        String long_name "y coordinate of projection";
        Int32 _Netcdf4Dimid 1;
    }
    x {
        String actual_range "60000.0 70000.0";
        String long_name "x coordinate of projection";
        Int32 _Netcdf4Dimid 2;
        String standard_name "projection_x_coordinate";
        String units "meter";
    }
    time {
        String units "days since 1970-01-01 00:00:00 +1:00";
        String standard_name "time";
        Int32 _Netcdf4Dimid 0;
    }
    NC_GLOBAL {
        String title "Vaklodingen";
        String source "The measurements are described in http://public.deltares.nl/
                       download/attachments/8553849/AGI-2005-GSMH-012.pdf";
        String references "http://public.deltares.nl/download/attachments/8553849/AGI-2005-GSMH-012.pdf";
        String institution "Rijkswaterstaat";
        String history "Converted with $Id$ from $HeadURL$, current time 2010-06-25 18:13:59.00,
                       current machine d00553, current user None";
        String comments "() are skipped because it is not aligned with the other grids, it is 10
                       meters below the other grids";
        String disclaimer "This data is made available in the hope that it will be useful, but
                       WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
                       or FITNESS FOR A PARTICULAR PURPOSE.'";
        String terms_of_use "Unknown";
    }
    DODS_EXTRA {
        String Unlimited_Dimension time;
    }
}
NULL
```

And for the dds (datastructure) format:

```
> getInfo(url_grid,what="dds")
```

```
Dataset {
    Int32 crs;
    Grid {
     ARRAY:
        Float64 lat[y = 625][x = 500];
     MAPS:
        Float64 y[y = 625];
        Float64 x[x = 500];
    } lat;
    Grid {
     ARRAY:
        Float64 z[time = 10][y = 625][x = 500];
     MAPS:
        Float64 time[time = 10];
        Float64 y[y = 625];
        Float64 x[x = 500];
    } z;
    Grid {
     ARRAY:
        Float64 lon[y = 625][x = 500];
     MAPS:
        Float64 y[y = 625];
```

```
        Float64 x[x = 500];
    } lon;
    Float64 y[y = 625];
    Float64 x[x = 500];
    Float64 time[time = 10];
} opendap/rijkswaterstaat/vaklodingen/vaklodingenKB117_4746.nc;
NULL
```

There exist other suffixes which are not implemented in the function, but are given in the following list. You can try these in a web browser. More information on this subject is available on, for instance, http://www.mohid.com/wiki/index.php?title=OpenDAP, where this list came from.

- .help : generates a web based help document
- .html : generates a web page containing information and action buttons over the netcdf file
- .info : returns the netcdf file metadata
- .dds : returns information about the variables
- .das : returns information about the variable attributes
- .ver : returns the OpenDAP server version
- .asc : returns the netcdf file in ASCII format

Now that we have an overview of what information can be retrieved from the file, we can start retrieving data. This is with the ncdf package accomplished by means of the `get.var.ncdf`. A list of variables is obtained from the NetCDF file as follows:

```
> names(grid.nc$var)

[1] "crs" "z"   "lon" "lat"
```

The var-field contains an object of the var.ncdf class, which essentially constitutes a named list. We can now chose to retrieve the values of x, y, and time variables, all coordinate variables (the same name appears in the list of dimensions (`names(grid.nc$dim)`). We also retrieve the actual bathymetry data (z), which is essentially a 3-dimensional cube.

```
> x <- get.var.ncdf(nc=grid.nc,varid="x")
> y <- get.var.ncdf(grid.nc,varid="y")
> time <- get.var.ncdf(grid.nc,"time")
> z <- get.var.ncdf(grid.nc,"z")
> list(x=dim(x),y=dim(y),z=dim(z))

$x
[1] 500

$y
[1] 625

$z
[1] 500 625  60
```

All these variables are stored as arrays (`class(x)`) with dimensions as indicated above. A convenient plotting function for this type of data is provided in the fields package (You know now how to install it). The `image.plot` function, however, requires ascending values for the independent variables. So, we have to reverse the order of y (and z to match those of y).

```
> y <- rev(y)
> z <- z[,(ncol(z):1),]
```

Because we can only plot a two-dimensional matrix or array, we start by plotting the first time step.

```
> image.plot(x,y,z[,,1])
```

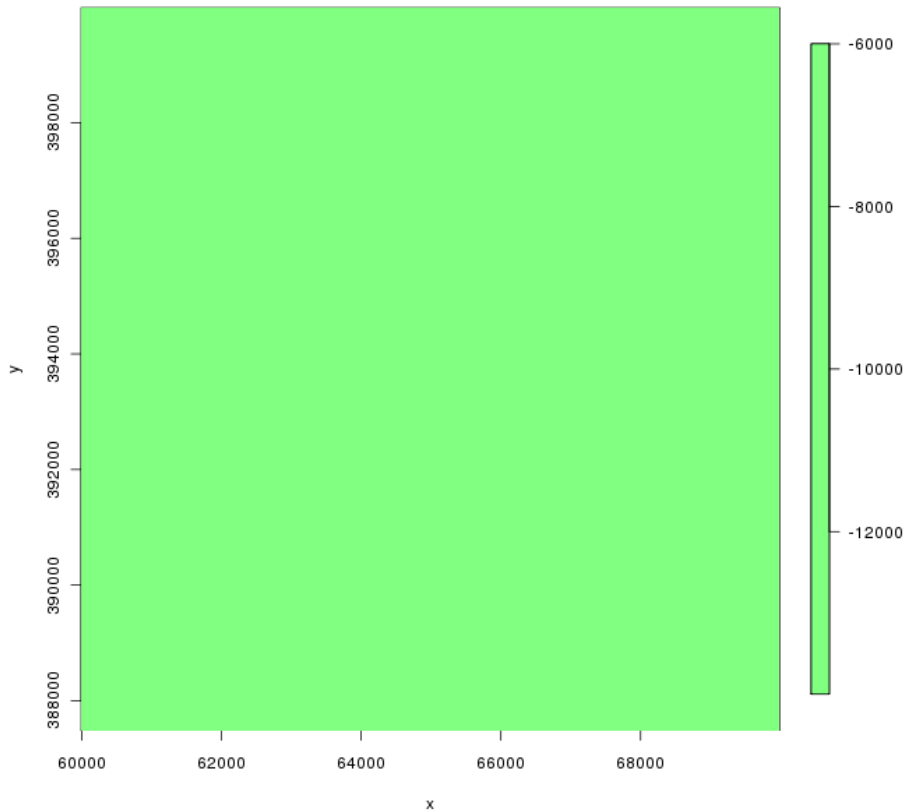

Figure 5: Image plot of the first slice of the bathymetry cube.

Woops, this is empty.... All values in this part of the array are -9999. Is this right?

```
> range(z[,,1])

[1] -9999 -9999

> att.get.ncdf(nc=grid.nc,varid="z",attname="_FillValue")

$hasatt
[1] TRUE

$value
[1] -9999
```

So which time slices are not empty? For empty slices the minimum and maximum value should be the same, hence the difference zero.

```
> selection <- which(apply(z,MARGIN=3,function(a) diff(range(a,na.rm=T))) != 0,arr.ind=T)
> selection
```

```
[1] 29 30 31 32 33 36 46 49 52
```

```
> z <- z[,,selection]
> z[z == -9999] <- max(z,na.rm=T)+1         # replace -9999 with some value closer

> image.plot(x,y,z[,,1],col = c(tim.colors(),"black"))          # the first non-empty time slice
```

By replacing the fill value with a value close but outside the range of actual relevant values, the colour scale is adjusted such that it ranges from the minimum to the maximum of this range, instead of from -9999 to the maximum. By changing the default colour spectrum black can be assigned to the lowest value (the fill value) in the entire range. Remember to exclude these values from future calculations...

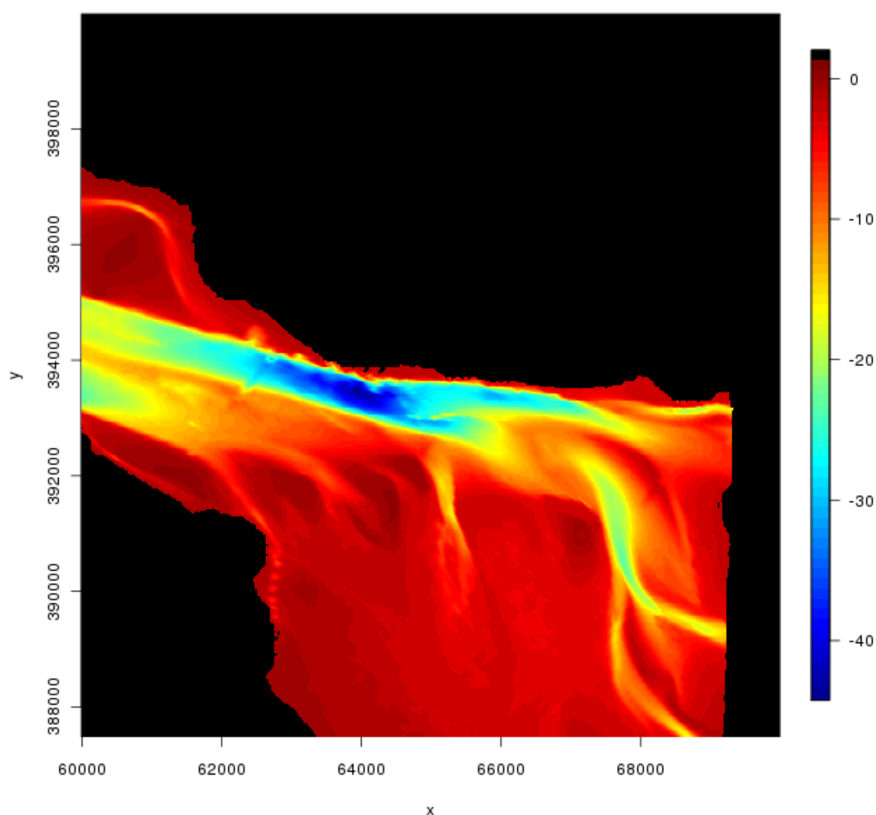

Figure 6: Second attempt for an image plot.

At the moment there are no headers in the arrays. However, headers may be interesting to write proper code, since parts can be selected based on a more informative value than an index. For that reason we assign the independent variable values to the relevant headers of the array with water depths (z). For x and y these are coordinates in the RD system (Dutch national grid). The time variable, however, is not very informative as it is.

```
> att.get.ncdf(grid.nc,"time","units")

$hasatt
[1] TRUE

$value
[1] "days since 0001-01-01 00:00:00 +1:00"

> dimnames(z) <- list(x,y,round(time[selection]/365.242))
>       # Note that the headers have to be unique. If the rounding off
>       # of the time values (in years) would have resulted in repetition of
>       # values (i.e. multiple samplings per year), this would not have been
>       # possible.
> str(dimnames(z))

List of 3
 $ : chr [1:500] "60000" "60020" "60040" "60060" ...
 $ : chr [1:625] "387500" "387520" "387540" "387560" ...
 $ : chr [1:9] "1985" "1986" "1987" "1988" ...
```

We want to plot a latitudinal depth transect along the longitudinal coordinate 62000, and a longitudinal transect along latitude

```
> par(mfrow=c(1,3))
> image.plot(x,y,z[,,1],col = c(tim.colors(),"black"))
> abline(v=62000,h=393600,col="white",lwd=2)
> plot(y,z["62000",,1],type="l",xlab="latitude",ylab="depth [m]")
> plot(x,z[,"393600",1],type="l",xlab="longitude",ylab="depth [m]")
```

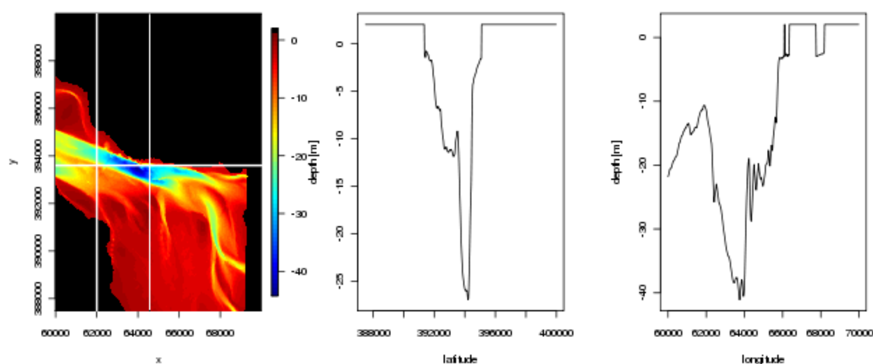

Figure 7: Longitudinal and latitudinal depth transects.

But with this approach we cannot look at transects other than longitudinal or lateral. For that reason we write a function that selects according to a line between two endpoints.

```
> # This function returns an object containing the necessary information for a 2d-plot
> # of a transect (e.g. depth profile). It is assumed that the dimensions are given in
> # the headers of the data matrix.
> transect <- function(datamatrix, startpoint, endpoint, method=c("index","dimension")) {
+     # in case of index values
```

```
+     if(method=="index") {
+       xs <- startpoint[1] ; ys <- startpoint[2]
+       xe <- endpoint[1]   ; ye <- endpoint[2]
+     }
+     if(method=="dimension") {
+       xvalues <- as.numeric(rownames(datamatrix))
+       yvalues <- as.numeric(colnames(datamatrix))
+       xs <- which.min(abs(xvalues-startpoint[1]))
+       ys <- which.min(abs(yvalues-startpoint[2]))
+       xe <- which.min(abs(xvalues-endpoint[1]))
+       ye <- which.min(abs(yvalues-endpoint[2]))
+     } else {
+     print("no appropriate method")
+     return()
+     }
+     x_orig <- xs
+     y_orig <- ys
+     xrange <- abs(xs-xe)
+     yrange <- abs(ys-ye)
+     nbcell <- ceiling(sqrt(xrange^2+yrange^2))
+     xTransect <- seq(from=xs,to=xe,length.out=nbcell)
+     yTransect <- seq(from=ys,to=ye,length.out=nbcell)
+     coordTransect <- unique(round(cbind(xTransect,yTransect)))
+     colnames(coordTransect) <- c("x","y")
+     distFromStart <- apply(coordTransect,MARGIN=1,function(x) sqrt(x[1]^2+x[2]^2))
+     values <- datamatrix[coordTransect]
+     names(startpoint) <- c("x","y")
+     names(endpoint) <- c("x","y")
+   return(list(start=startpoint,end=endpoint,indexTransect=coordTransect,
+               distance=distFromStart,values=values)
+         )
+ }
> # Application of the function
> depthProfile <- transect(datamatrix=z[,,"1988"],
+                          startpoint = c(62000,391500),endpoint = c(64000,394000),
+                          method="dimension")
> par(mfrow=c(1,2))
> image.plot(x,y,z[,,"1988"],col = c(tim.colors(),"black"))
> segments(x0=depthProfile$start[1],y0=depthProfile$start[2],x1=depthProfile$end[1],
+          y1=depthProfile$end[2],lwd=2,col="white")
> plot(depthProfile$distance,depthProfile$values,
+      xlab="Distance from origine [m]",ylab="Depth [m]")
```

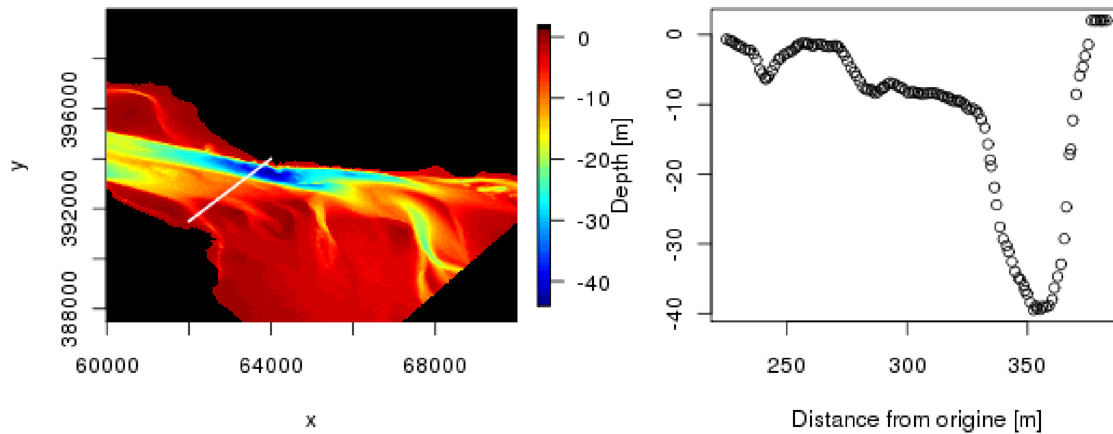

Figure 8: Depth transects in an arbitrary direction.

2.2 Writing NetCDF files

Suppose now that we want to calculate and store an NDVI map, derived from a Landsat image cube. We have downloaded the relevant bands as ArcInfo ASCII raster files. We can store this locally as a NetCDF file. To import the ArcInfo file we need another package, called ”adehabitat”, which can be installed from the CRAN website. The `import.asc` function reads the ASCII file and constructs an object of the class ”asc”, which contains an array of data values.

```
> library(adehabitat)
> ETM3 <- import.asc(file="/home/tom/Desktop/ETM3.asc")
> ETM4 <- import.asc(file="/home/tom/Desktop/ETM4.asc")
> NDVI <- (ETM4-ETM3)/(ETM4+ETM3)    # NDVI = (NIR - red) / (NIR + red)
> image.plot(NDVI,col=topo.colors(64),main="Chlorophyll in the Delaware Estuary intertidal area
+                                           (NDVI)",xlab="Longitude",ylab="Lattitude")
```

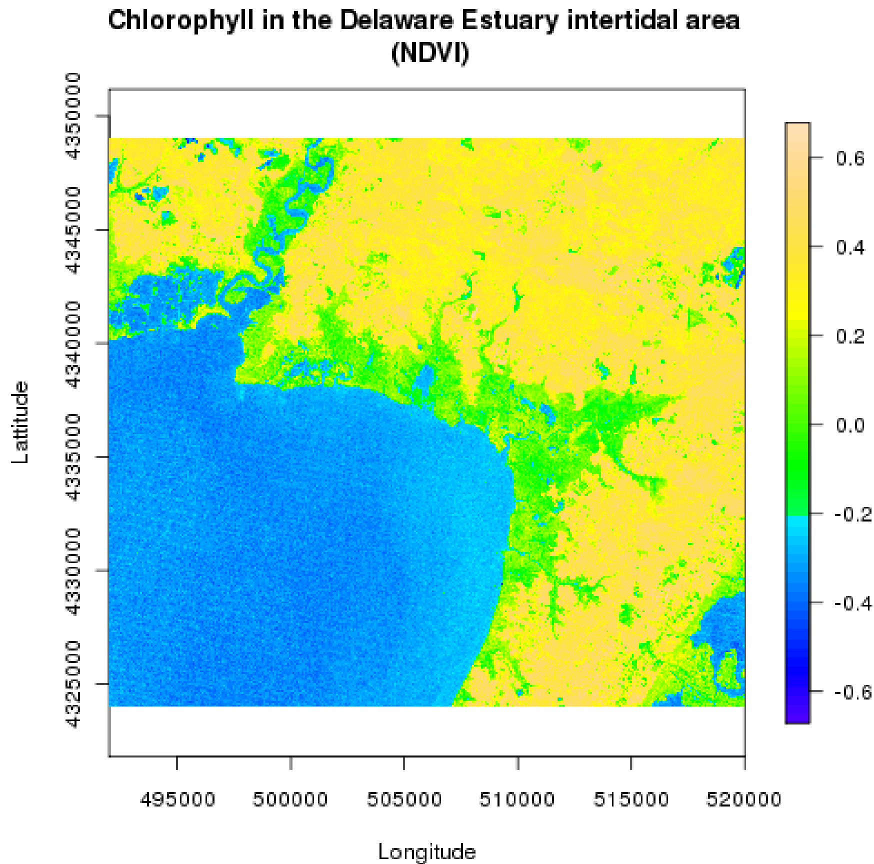

Figure 9: NDVI map of an intertidal area in the Delaware Estuary.

The first thing to do before writing the netCDF file is to construct the variables that represent the dimensions of the image. The information is stored as attributes in the NDVI object (and the original ETM3 and ETM4 objects). Attribute values are retrieved with the function `attr`. To obtain a list of attributes from the object the `str` command is useful. For an overview of the datastructure of an asc object, we refer to the help files of the `import.asc` function.

```
> x <- seq(attr(NDVI,"xll"),length.out=dim(NDVI)[1],by=attr(NDVI,"cellsize"))
> y <- seq(attr(NDVI,"yll"),length.out=dim(NDVI)[2],by=attr(NDVI,"cellsize"))
```

Next we define the dimensions for our netCDF file (dim.ncdf objects):

```
> x_dim <- dim.def.ncdf(name="x",units="m", vals=x, unlim=F, create_dimvar=T)
> y_dim <- dim.def.ncdf(name="y",units="m", vals=y, unlim=F, create_dimvar=T)
```

Two variables are created. The first is a 0-dimensional variable which contains georeferencing information (see below for important remarks). This data was not imported by the `import.asc` function, but retrieved via the QGIS software application (freely available for various software platforms; http://www.qgis.org). This strategy is used to comply with the CF conventions for metadata (Climate & Forecasts), which can be found at http://cf-pcmdi.llnl.gov/. The second ncdf variable is the actual NDVI map.

```
> crs_var <- var.def.ncdf.no_dim(name="crs",units="",dim=numeric(0),
+                                longname="UTM 18N",prec="integer",
```

```
+                                   missval=NA)           # see remark 2 for function (below)
> ndvi_var <- var.def.ncdf(name="ndvi",units="",dim=list(x_dim,y_dim),
+                             longname="Normalized Difference Vegetation Index",
+                             prec="double",missval=-9999)
> ndvi.nc <- create.ncdf("/home/tom/Dropbox/NetCDF/CourseMaterial/Data/NDVI.nc",
+                          vars=list(crs_var,ndvi_var),verbose=T)
```

```
[1] "create.ncdf: input was a list of vars"
[1] "Calling R_nc_create for file  /home/tom/Dropbox/NetCDF/CourseMaterial/Data/NDVI.nc"
[1] "back from R_nc_create for file  /home/tom/Dropbox/NetCDF/CourseMaterial/Data/NDVI.nc"
[1] "create.ncdf: about to create the dims"
[1] "var.add.ncdf: entering with indefine= TRUE"
[1] "var.add.ncdf: ncid of file to add to= 786432
        filename= /home/tom/Dropbox/NetCDF/CourseMaterial/Data/NDVI.nc
        writable= TRUE"
[1] "var.add.ncdf: varname to add= crs"
[1] "var.add.ncdf: creating 0 dims for var crs"
[1] "create.ncdf: creating integer precision var crs"
[1] "create.ncdf: C call returned value 0"
[1] "var.add.ncdf: entering with indefine= TRUE"
[1] "var.add.ncdf: ncid of file to add to= 786432
        filename= /home/tom/Dropbox/NetCDF/CourseMaterial/Data/NDVI.nc
        writable= TRUE"
[1] "var.add.ncdf: varname to add= ndvi"
[1] "var.add.ncdf: creating 2 dims for var ndvi"
[1] "var.add.ncdf: working on dim > x < (number 1 ) for var ndvi"
[1] "create.ncdf: creating dim x"
[1] "dim.create.ncdf: entering for ncid= 786432"
[1] "dim.create.ncdf: entering for dim x"
[1] "dim.create.ncdf: about to call R_nc_def_dim for dim x"
[1] "dim.create.ncdf: making dimvar for dim x"
[1] "dim.create.ncdf: about to call R_nc_def_var_double for dimvar x"
[1] "dim.create.ncdf: about to call R_nc_put_vara_double dimvals for dimvar x"
[1] "dim.create.ncd: exiting for ncid= 786432   dim= x"
[1] "var.add.ncdf: working on dim > y < (number 2 ) for var ndvi"
[1] "create.ncdf: creating dim y"
[1] "dim.create.ncdf: entering for ncid= 786432"
[1] "dim.create.ncdf: entering for dim y"
[1] "dim.create.ncdf: about to call R_nc_def_dim for dim y"
[1] "dim.create.ncdf: making dimvar for dim y"
[1] "dim.create.ncdf: about to call R_nc_def_var_double for dimvar y"
[1] "dim.create.ncdf: about to call R_nc_put_vara_double dimvals for dimvar y"
[1] "dim.create.ncd: exiting for ncid= 786432   dim= y"
[1] "create.ncdf: creating double precision var ndvi"
[1] "create.ncdf: C call returned value 0"
```

```
> put.var.ncdf(nc=ndvi.nc, varid="ndvi",vals=NDVI,start=c(1,1),count=c(-1,-1))
> close.ncdf(ndvi.nc)
```

```
[[1]]
[1] 786432
```

Using the var.ncdf and the dim.ncdf objects an ncdf object is created, holding the connection to a newly created empty netCDF file. The actual non-coordinate variables are added by the `put.var.ncdf` function. By closing the file the data is written to disk.
After creation of the file a number of extra attributes are needed to comply with the CF conventions for metadata. First the file is opened in writable mode.

```
> ndvi.nc <- open.ncdf("/home/tom/Dropbox/NetCDF/CourseMaterial/Data/NDVI.nc",write=T)
```

The following global attributes are added by specifying the variable id as 0:

```
> att.put.ncdf(nc=ndvi.nc,varid=0,attname="title",
+              attval="NDVI of the Delaware Estuary Intertidal Area")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid=0,attname="institution",attval="NIOO-CEME")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid=0,attname="source",
+              attval="Landsat 7 Enhanced Thematic Mapper")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid=0,attname="history",
+              attval="The image cube was downloaded from the NASA website
+                      and processed using GRASS 6.3")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid=0,attname="references",attval="http://www.nasa.gov/")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid=0, attname="comment",
+              attval='The original Landsat image cube, downloaded as Erdas Imagine
+                      format (.img), was imported into GRASS 6.3 using the GDAL libraries
+                      for conversion. The bands 3 (red) and 4 (NIR; near infra-red)
+                      were selected for export as ESRII ArcInfo ASCII files. These were
+                      read into the R Statistical Software via the import.asc function of
+                      the adehabitat package. The NDVI was calculated from these bands,
+                      and exported as this NetCDF file.')

[[1]]
[1] 851968
```

The only attribute that is added to the ndvi variable is the reserved attribute _FillValue. According to CF guidelines this should be provided every time, whereas the missing_value attribute is deprecated. However, the ncdf package requires that a missing value is specified, which by default is 10^{30} (1e+30). Here we chose to specify both the _FillValue and missing_value attributes as -9999, well outside the range. When the netCDF file is loaded using the ncdf package, the variable values equal to the missing_value (not the _FillValue) attribute are replaced by NA.

```
> att.put.ncdf(nc=ndvi.nc,varid="ndvi",attname="_FillValue",attval=-9999)

[[1]]
[1] 851968
```

The crs (coordinate reference system) variable is only intended to provide attributes, it has no value of its own. The following attributes are added according to the CF conventions:

```
> att.put.ncdf(nc=ndvi.nc,varid="crs",attname="spatial_ref",attval='
+ "PROJCS["WGS 84 / UTM zone 18N",
+     GEOGCS["WGS 84",
+         DATUM["WGS_1984",
+             SPHEROID["WGS 84",6378137,298.257223563,
+                 AUTHORITY["EPSG","7030"]],
+             AUTHORITY["EPSG","6326"]],
+         PRIMEM["Greenwich",0,
+             AUTHORITY["EPSG","8901"]],
+         UNIT["degree",0.01745329251994328,
+             AUTHORITY["EPSG","9122"]],
+         AUTHORITY["EPSG","4326"]],
+     UNIT["metre",1,
+         AUTHORITY["EPSG","9001"]],
+     PROJECTION["Transverse_Mercator"],
+     PARAMETER["latitude_of_origin",0],
+     PARAMETER["central_meridian",-75],
+     PARAMETER["scale_factor",0.9996],
+     PARAMETER["false_easting",500000],
+     PARAMETER["false_northing",0],
+     AUTHORITY["EPSG","32618"],
+     AXIS["Easting",EAST],
+     AXIS["Northing",NORTH]]"
+ ')    # nesting of ' and "

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid="crs",attname="grid_mapping_name",attval="transverse_mercator")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid="crs",attname="scale_factor_at_central_meridian",attval="0.9996")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid="crs",attname="longitude_of_central_meridian",attval="-75")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid="crs",attname="latitude_of_projection_origin",attval="0")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid="crs",attname="false_easting",attval="500000")

[[1]]
[1] 851968

> att.put.ncdf(nc=ndvi.nc,varid="crs",attname="false_northing",attval="0")

[[1]]
[1] 851968
```

```
> att.put.ncdf(nc=ndvi.nc,varid="crs",attname="comment",attval="EPSG = 32618")
[[1]]
[1] 851968
```

The spatial_ref attribute is not obligatory but simplifies the use of the data because the content can be interpreted by most GIS software packages. The grid_mapping_name is fixed for a particular coordinate system, just as the parameters that have to be specified (cf. CF conventions documentation on their website). We decided to add the EPSG code as a comment. Many GIS applications can use this code to setup the reference coordinate system.
Again, according to CF regulations the standard_name attributes of the coordinate variables have to be set to a fixed value determined by the type of coordinate system. _FillValue attributes are provided here as well but in our case not necessary because all dimensions are limited and do not contain open spaces.

```
> # x coordinate
> att.put.ncdf(ndvi.nc,varid="x",attname="standard_name",attval="projection_x_coordinate")
[[1]]
[1] 851968

> att.put.ncdf(ndvi.nc,varid="x",attname="_FillValue",attval="-9999")
[[1]]
[1] 851968

> # y coordinate
> att.put.ncdf(ndvi.nc,varid="y",attname="standard_name",attval="projection_y_coordinate")
[[1]]
[1] 851968

> att.put.ncdf(ndvi.nc,varid="y",attname="_FillValue",attval="-9999")
[[1]]
[1] 851968

> close.ncdf(ndvi.nc)
[[1]]
[1] 851968
```

The file is closed again, and the metadata is written to disk.

3 Remarks

1.) It may be interesting to install some additional software for visualization of netCDF files. One possible application is ncview, written by David Pierce (author of the ncdf package). This is freely downloadable from the UNIDATA website, and ideal for a quick look at the content of your files.

2.) The ncdf package has a few odd features. One of them is the requirement to provide at least one dimension. Given the crs requirements of the CF convention, we need to construct a 0-dimensional variable (which takes less space than a 1-dimensional variable...). To allow for this, two functions of the ncdf package have to be changed. The function code is easily retrieved by typing in the function name without braces. This can be copied to a script file and edited to your liking. After loading the function, the original function is overwritten, unless the name is changed. For the interested reader we provide the altered versions of the functions `var.add.ncdf` and `var.def.ncdf` (with changed name). Since the `var.add.ncdf` function is called by the `create.ncdf` function, the name had to remain the same and the original function is overruled by this version upon loading of the latter. When the new version is removed (`rm(var.add.ncdf)`), the old version is still present.

```
var.def.ncdf.no_dim <- function (name, units, dim, missval, longname = name,
                                 prec = "single") {
    if (!is.character(name)) {
        stop("Passed a var name that is NOT a string of characters!")
    }
    if (storage.mode(missval) == "character")
        prec <- "char"
    var <- list()
    var$name <- name
    var$units <- units
    var$missval <- missval
    var$longname <- longname
    var$id <- -1
    attr(var, "class") <- "var.ncdf"
    if (prec == "float")
        prec <- "single"
    if ((prec != "short") && (prec != "single") && (prec != "double") &&
        (prec != "integer") && (prec != "char") && (prec != "byte"))
        stop(paste("var.def.ncdf: error: unknown precision specified:",
            prec, ". Known values: short single double integer char byte"))
    var$prec <- prec
    if (is.character(class(dim)) && (class(dim) == "dim.ncdf"))
        dim <- list(dim)
    var$dim <- dim
    var$ndims <- length(var$dim)
    varunlimited <- FALSE
    if (var$ndims != 0) {
    for (i in 1:var$ndims) {
        if (class(var$dim[[i]]) != "dim.ncdf") {
            print(var)
            stop("Error, passed variable has a dim that is NOT of class dim.ncdf!")
        }
    }
        for (i in 1:var$ndims) {
            if (var$dim[[i]]$unlim)
                varunlimited <- TRUE
        }
    }
    var$unlim <- varunlimited
    return(var)
}

var.add.ncdf <- function (nc, v, verbose = FALSE, indefine = FALSE) {
    if (verbose)
        print(paste("var.add.ncdf: entering with indefine=",
            indefine))
    if (class(nc) != "ncdf")
        stop("var.add.ncdf: passed nc NOT of class ncdf!
              The first arg to var.add.ncdf must be the return value from a call
              to open.ncdf(...,write=TRUE)")
    if (verbose)
        print(paste("var.add.ncdf: ncid of file to add to=",
```

```
            nc$id, "   filename=", nc$filename, "    writable=",
            nc$writable))
    if (class(v) != "var.ncdf")
        stop("var.add.ncdf: passed var NOT of class var.ncdf!
              The second arg to var.add.ncdf must be the return value from a call to var.def.ncdf")
    if (verbose)
        print(paste("var.add.ncdf: varname to add=", v$name))
    if (!indefine) {
        if (verbose)
            print(paste("var.add.ncdf: about to redef ncid=",
                nc$id))
        redef.ncdf(nc)
    }

    nd <- v$ndims
    dimvarids <- array(0, nd)
    if (verbose)
        print(paste("var.add.ncdf: creating", nd, "dims for var",
            v$name))
if(nd > 0) {
    for (idim in 1:nd) {
        d <- v$dim[[idim]]
        if (verbose)
            print(paste("var.add.ncdf: working on dim >", d$name,
                "< (number", idim, ") for var", v$name))
        place <- -1
        if (length(nc$dim) > 0) {
            for (ii in 1:length(nc$dim)) {
                if (nc$dim[[ii]]$name == d$name) {
                  if (!dim.same.ncdf(nc$dim[[ii]], d)) {
                    paste("Error, when trying to add variable named",
                      v$name, "to file", nc$filename,
                      "I found this variable has a dim named",
                      d$name)
                    paste("However, the file ALREADY has a dim named",
                      nc$dim[[ii]]$name,
                      "with different characteristics than the new dim with the same name!")
                    stop("This is not allowed.")
                  }
                  place <- ii
                  break
                }
            }
        }
        if (place == -1) {
            if (verbose)
                print(paste("create.ncdf: creating dim", d$name))
            ids <- dim.create.ncdf(nc, d, verbose)
            dimid <- ids[1]
            dimvarid <- ids[2]
            newel <- list()
            attr(newel, "class") <- "dim.ncdf"
```

```
            newel$name <- d$name
            newel$units <- d$units
            newel$vals <- d$vals
            newel$lene <- d$len
            newel$id <- dimid
            newel$unlim <- d$unlim
            newel$dimvarid <- dimvarid
            nc$ndims <- nc$ndims + 1
            nc$dim[[nc$ndims]] <- newel
        } else dimid <- nc$dim[[place]]$id
        dimvarids[idim] <- dimid
    }
    dimids <- dimvarids
    dimids <- dimids[length(dimids):1]
} else dimids <- 0

    newvar <- list()
    newvar$id <- -1
    newvar$error <- -1
    if (verbose)
        print(paste("create.ncdf: creating", v$prec, "precision var",
            v$name))
    if ((v$prec == "integer") || (v$prec == "int"))
        funcname <- "R_nc_def_var_int"
    else if (v$prec == "short")
        funcname <- "R_nc_def_var_short"
    else if ((v$prec == "single") || (v$prec == "float"))
        funcname <- "R_nc_def_var_float"
    else if (v$prec == "double")
        funcname <- "R_nc_def_var_double"
    else if (v$prec == "char")
        funcname <- "R_nc_def_var_char"
    else if (v$prec == "byte")
        funcname <- "R_nc_def_var_byte"
    else stop(paste("internal error in create.ncdf: var has unknown precision:",
        v$prec, ". Known vals: short single double integer char byte"))
    newvar <- .C(funcname, as.integer(nc$id), v$name, as.integer(v$ndims),
        as.integer(dimids - 1), id = as.integer(newvar$id), error = as.integer(newvar$error),
        PACKAGE = "ncdf")
    if (verbose)
        print(paste("create.ncdf: C call returned value", newvar$error))
    if (newvar$error != 0)
        stop("Error in create.ncdf, defining var!")
    newvar$id <- newvar$id + 1
    v$id <- newvar$id
    nc$nvars <- nc$nvars + 1
    nc$var[[nc$nvars]] <- v
    nc$varid2Rindex[newvar$id] <- nc$nvars
    if ((!is.null(v$units)) && (!is.na(v$units)))
        att.put.ncdf(nc, newvar$id, "units", v$units, definemode = TRUE)
    if (is.null(v$missval) && ((v$prec == "single") || (v$prec ==
        "float") || (v$prec == "double"))) {
```

```
        att.put.ncdf(nc, newvar$id, "missing_value", default.missval.ncdf(),
            definemode = TRUE)
    }
    else {
        if (!is.na(v$missval)) {
            att.put.ncdf(nc, newvar$id, "missing_value", v$missval,
                definemode = TRUE)
        }
    }
    if (v$longname != v$name)
        att.put.ncdf(nc, newvar$id, "long_name", v$longname,
            definemode = TRUE)
    if (!indefine)
        enddef.ncdf(nc)
    return(nc)
}
```