R notes

Vygantas Butkus

May 12, 2014

Contents

1 Basic

```
### Time
  proc.time()
  system.time()

### debug # c, n, where, Q;
  # options(error = recover)
  debug()
  browser()
  trace()
  recover()
  traceback()

### Search in environment:
  ?ls; ?objects  # List Objects
  ?search   # Give Search Path for R Objects
  ?apropos; ?find  # Find Objects by (Partial) Name
  ?get      # Return the Value of a Named Object

### Object analysis:
  x <- matrix(1:4-0.5,2,2)
  class(x)        # matrix
  mode(x)         # numeric
  storage.mode(x) # double
  typeof(x)       # double
  attributes(x)   # dim [1] 2 2
  names(x)        # NULL
  terms(lm(Sepal.Length ~ Sepal.Width, data=iris)) # For models

  # S3
  methods(class="matrix")                  # the list of all S3 methods
  methods("summary")                 # functions of `summary` by class
  getS3method("summary", "matrix")  # Get the body of the function
  getClass("matrix")

  # S4
  library("Matrix")
  mat <- Matrix(1:6, 3, 2)  # S4 object
  getClass(class(mat))  # or `showClass(class(mat))`
  getSlots(class(mat))  # or `slotNames(mat)`
  mat@Dim
  methods(class = "Matrix")
  showMethods(class = "Matrix")

### Set operations:
  union(x, y)
  intersect(x, y)
  setdiff(x, y)
  setequal(x, y)
  is.element(el, set)
  subset(old,logical)

### Index, match
  a %in% b
  match()
  which()

### Loopless
  tapply()    # Apply a Function Over a Ragged array. General, good and fast
  by()        # `tapply` for `data.frame` (or matrix)
  aggregate() # Compute Summary Statistics of Data Subsets
  apply()     # Apply Functions Over Array Margins
  mapply()    # Apply a Function to Multiple List or Vector Arguments
  outer()     # Outer Product of Arrays
  Vectorize() # Vectorize a Scalar Function
  # Apply a Function over a List or Vector:
  lapply()
  sapply()
  replicate()
```

```
### Sequences
  rep(1:4, each = 2)   # 1 1 2 2 3 3 4 4
  rep(1:4, c(2,1,2,1)) # 1 1 2 3 3 4
  seq(1, 9, by = pi)     # 1.000000 4.141593 7.283185
  sequence(c(3,2))         # 1 2 3 1 2
  seq_along(5:10)      # 1 2 3 4 5 6
  unique()
  duplicated()

### Combinatorics, possible combinations, brudforce:
  combn(x, m, FUN = NULL)       # All Combinations of n Elements, Taken m at a Time (with sunction)
  expand.grid()             # All combinations (with simetry)
  choose(n, k)  #pasirinkimu skaicius (C is n po K)
  factorial(n)
  combinations()        #from gtools
  permutations()        #from gtools

### Vector, matrix
  # manipulation
  subset()
  with();within()

  # Ordering
  order()
  sort.list()
  sort()
  arrange() # drom plyr package
  rank()
  # appending
  cbind()
  rbind()
  append()
  # calculation
  A %*% B               #matricu daugyba
  tcrossprod()          # fast product
  outer(X, Y, FUN="*", ...)                     # outer product {%o%}
  kronecker(X, Y, FUN = "*")            # kronecer product {%x%}
  # Misc
  colSums(); rowSums()
  split(); cut()  # grouping
  rev()       # reverse
  embed(x,2)    # laged matrix

### Functions, solutions
  optim()       # General optimisation
  optimize()      # one dimension optimisation
  constrOptim()   # constrain optimisation
  uniroot()     # the root of monotone functions
  nlm()         # Non-linear minimisation
  nlminb()              # More robust (non-)constrained non-linear minimisation

### Strings:
  paste()
  format()
  formatC()
  # Substrings (vectorised, can raplece)
  substr(x, start, stop)
  substring(text, first, last = 1000000L)
  # replacments
  sub(pattern, replacement, x)
  gsub(pattern, replacement, x)
  # Find patern
  grep(pattern, x) ;
  grepl(pattern, x)
  # find expresion
  regexpr(pattern, text)
  gregexpr(pattern, text)
  # interpretation
  eval(parse(text = "(d <- 4 + 7)"))
  substitute(expr, env)
  # misc
  strsplit("a,b;c", ",|;") # a, b, c
  nchar() # the length of string
  chartr() # traslete: L -> L
  tolower()
  toupper()
  as.numeric(as.character(f)) # factor - > number

### Files
  ?files         # Help on low-level interface to file system
  list.files()   # List files in a give directory
  file.info()    # Get information about files


### Misc:
  interaction()   # Joining factors
  x %% y          #mod
  x %/% y       #div
  identical()
  suppressMessages

### Built-in constants:
  pi;letters;LETTERS   # Pi, lower & uppercase letters, e.g. letters[7] = "g"
  month.abb;month.name # Abbreviated & full names for months


### Graphics
  help(package=graphics) # List all graphics functions

  plot()                # Generic function for plotting of R objects
  par()                 # Set or query graphical parameters
  curve(5*x^3,add=T)    # Plot an equation as a curve
  points(x,y)           # Add another set of points to an existing graph
  arrows()              # Draw arrows [see errorbar script]
  abline()              # Adds a straight line to an existing graph
  lines()               # Join specified points with line segments
  segments()            # Draw line segments between pairs of points
  hist(x)               # Plot a histogram of x
  pairs()               # Plot matrix of scatter plots
  matplot()             # Plot columns of matrices
  boxplot()

  ?identify

  ?device               # Help page on available graphical devices
  ?dev.control
  postscript()          # Plot to postscript file
  pdf()                 # Plot to pdf file
  png()                 # Plot to PNG file
  jpeg()                # Plot to JPEG file
  X11()                 # Plot to X window
  persp()               # Draws perspective plot
  contour()             # Contour plot
  image()               # Plot an image
  dev.off()

  x= recordPlot()                           #save the current plot device output in the object x
  replayPlot(x)                            #replot object x
  layout(mat)                         #specify where multiple graphs go on the page
                                      #experiment with the magic code from Paul Murrell to do fancy graphi
  layout(rbind(c(1, 1, 2, 2, 3, 3),
               c(0, 4, 4, 5, 5, 0)))
  for (i in 1:5) {
    plot(i, type="n")
    text(1, i, paste("Plot", i), cex=4)
  }

### Statistical
  help(package=stats)   # List all stats functions

  ?Chisquare            # Help on chi-squared distribution functions
  ?Poisson              # Help on Poisson distribution functions
  help(package=survival) # Survival analysis

  cor.test()            # Perform correlation test
  cumsum(); cumprod(); cummin(); cummax()   # Cumuluative functions for vectors
  density(x)            # Compute kernel density estimates
  ks.test()             # Performs one or two sample Kolmogorov-Smirnov tests
  loess(); lowess()     # Scatter plot smoothing
  mad()                 # Calculate median absolute deviation
  mean(x); weighted.mean(x); median(x); min(x); max(x); quantile(x)
  rnorm(); runif()      # Generate random data with Gaussian/uniform distribution
  splinefun()           # Perform spline interpolation
  smooth.spline()       # Fits a cubic smoothing spline
  sd()                  # Calculate standard deviation
  summary(x)            # Returns a summary of x: mean, min, max etc.
  t.test()              # Student's t-test
  var()                 # Calculate variance
  sample()              # Random samples & permutations
  ecdf()                # Empirical Cumulative Distribution Function
  qqplot()              # quantile-quantile plot

### Help
  ?Control     # Help on control flow statements (e.g. if, for, while)
  ?Extract     # Help on operators acting to extract or replace subsets of vectors
  ?Logic       # Help on logical operators
  ?Mod         # Help on functions which support complex arithmetic in R
  ?Paren       # Help on parentheses
  ?regex       # Help on regular expressions used in R
  ?Syntax      # Help on R syntax and giving the precedence of operators
  ?Special     # Help on special functions related to beta and gamma functions
```

2 Data

2.1 Input-Output

Main base functions:

```
?read.table  # ..., Reads a file in table format and creates a data frame from it
?scan        # Read data into a vector or list from the console or file
?readLines   # Read Text Lines from a Connection
?readBin     # read binary data

?connections    # ..., Functions to Manipulate Connections
?textConnection # Text Connections

?save; load         # Reload Saved Datasets
?readRDS; ?saveRDS; # Functions to write a single R object to a file, and to restore it.
```

Table 1: Usefull packages of ‘data, input-output‘.

package	rating	heading
foreign	9.0	Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase, ..
XML2R	8.0	EasieR XML data collection
xlsx	7.0	Read, write, format Excel 2007 and Excel 97/2000/XP/2003 files
XLConnect	5.0	Excel Connector for R

See also: Database

Special examples

```
text <- "
1 2 3
4 5 6
7 8 9
"
read.table(textConnection(text))

##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
## 3  7  8  9
```

2.2 Manipulation

Table 2: Usefull packages of ‘data, manipulation‘.

package	rating	heading
plyr	9.1	Tools for splitting, applying and combining data
reshape2	9.0	Flexibly reshape data: a reboot of the reshape package
sqldf	7.0	Perform SQL Selects on R Data Frames
gdata	5.0	Various R programming tools for data manipulation
abind	5.0	Combine multi-dimensional arrays

Special examples

```
# list -> matrix
list2mt <- function(lst){
  return(do.call(rbind, lst))
}

# matrix -> list
mt2list <- function(mt){
        return(split(mt, row(mt)))
}

# sort matrix by colums.
msort <- function(mt, sortnr=1, desc=FALSE){
  if(desc) mt <- -mt

  if(length(sortnr)==1){
    return(mt[order(mt[,sortnr]),])
  }else{
    return(mt[do.call(order,mt2list(t(mt[,sortnr]))),])
  }
}

# sort DF. See also: arrange {plyr}
esort <- function(df, sortvar, ...) {
        attach(df,warn.conflicts = FALSE )
        df <- df[with(df,order(sortvar,...)),]
        detach(df)
        return(df)
}
```

2.3 Database

Table 3: Usefull packages of ‘data, database‘.

package	rating	heading
RMySQL	7.0	R interface to the MySQL database
RODBC	7.0	ODBC Database Access
RSQLite	7.0	SQLite interface for R

```
library("RSQLite")  # databese with out server

## Loading required package:  DBI

drv    <- dbDriver("SQLite")
con <- dbConnect(drv, dbname=":memory:")  # if for saving: dbConnect(drv,"newexample.db")


n = ceiling(1e6/26^2) # 1 million rows
DF = data.frame(x=rep(LETTERS,each=26*n),
  y=rep(letters,each=n),
  v=rnorm(n*26^2),
  stringsAsFactors=FALSE
)

res = dbWriteTable(con,"data",DF)

dbListTables(con)
```

```
## [1] "data"

dbListFields(con, "data")

## [1] "row_names" "x"         "y"         "v"

res <- dbSendQuery(con, "select x, sum(v) as sum_v from data group by x")
head(fetch(res, -1))  # or use dbGetQuery to do everything

##   x sum_v
## 1 A 111.2
## 2 B 200.9
## 3 C 293.6
## 4 D 304.7
## 5 E 116.7
## 6 F 187.7

dbClearResult(res)

## [1] TRUE
```

2.4 Big-Data

Table 4: Usefull packages of ‘data, big-data‘.

package	rating	heading
bigmemory	7.0	Manage massive matrices with shared memory and memory-mapped files
ff	7.0	memory-efficient storage of large data on disk and fast access functions
biglm	7.0	bounded memory linear and generalized linear models
data.table	7.0	Extension of data.frame

- Package ‘bigmemory‘ is from the family of ‘big data‘. The whole list: bigmemory, biganalytics, bigtabulate, bigalgebra.
- Nowadays it is in very active development MapReduce jobs in Hadoop, that is implimented in ‘rmr‘ package.

```
library("data.table") # it is very fast for big data (in proper ussing)
# ?data.table

# create
DT = data.table(x=rep(c("a","b","c"),each=3), y=c(1,3,6), v=1:9)
setkey(DT,x,y)
DT

# colums
head(DT[,v])
head(DT[,2,with=FALSE])          # 2nd column

# rows
DT[2]
DT[2,]
DT[2:3, sum(v)]  # sum(v) over rows 2 and 3
```

```
DT[c(FALSE,TRUE)]

# subseting
DT[x=="b" & y==3,] # works but is using data.table badly
DT[J("b", 3)]      # goodway
DT[!J("b", 3)]     # revers

DT[x=="b" & y<5,]   # works but is using data.table badly
DT[J("b", 1:4)]     # wiht NA
DT[J("b", 1:4),roll=TRUE]     # fill NA
DT[J("b", 1:4),nomatch=0]     # remuve NA

# BY
DT[,sum(v),by=x]
DT[,sum(v),by=list(y%%2)] # by expression
DT[,.SD[2],by=x]           # 2nd row of each group
DT[,list(MySum=sum(v),
         MyMin=min(v),
         MyMax=max(v)),
    by=list(x,y%%2)]

# compound query
DT[,sum(v),x][V1<20]

# adding computed colum
DT[,m:=mean(v),by=x]
```

3 Programing

3.1 Debuging

```
##### Debuging. RStudio has quite good debuging tools for deep debuging.
    # The strategy there to start
    # 1) Then the error occurs use
    traceback()
    # 2)Use
    options(error=recover);
    # or
    options(error=browser);
    # (after debuging set `options(error=NULL);` )
    # and rerun the code. In error you can start exploring.
    # 3) Finily use
    with_debug(install()) # in package developing (with devtools)
    debug()
    browser()
    # or break points in RStudio for deap investigation.
    # 4) for source code and scripts use
    findLineNum()
    setBreakpoint()
```

3.2 Compiling

```
##### Compiling. Good for looping or smth.
    # If caling external function - have no speed benefit.
    # Be aware of recursion - must compile with same name (prefered) or resursion sould use `Rcall`
    # Compiling with same name is OK

library("compiler")
# ?cmpfun

### Matrix multiplication with a lot of loops
MM <- function(A, B){
  if(dim(A)[2]!=dim(B)[1]) stop("incompatable")
  v <- dim(A)[2]
  ans <- matrix(NA, dim(A)[1], dim(B)[2])
  for(i in 1:dim(ans)[1]){
    for(j in 1:dim(ans)[2]){
      dum <- 0
      for(k in 1:v){
        dum <- dum + A[i, k]*B[k, j]
      }
      ans[i,j] <-  dum
    }
  }
  ans
}
MMC <- cmpfun(MM)

### test
A <- matrix(1:6, 2, 3)
B <- matrix(1:12, 3, 4)
all.equal(A %*% B, MM(A, B))

## [1] TRUE

all.equal(A %*% B, MMC(A, B))

## [1] TRUE

### speed
require("rbenchmark")

## Loading required package:  rbenchmark

A <- matrix(rnorm(20*10), 20, 10)
B <- matrix(rnorm(10*30), 10, 30)
benchmark(MM(A, B), MMC(A, B), A%*%B)

##        test replications elapsed relative user.self sys.self user.child sys.child
## 3   A %*% B          100   0.001        1     0.004    0.000          0         0
## 1  MM(A, B)          100   0.709      709     0.696    0.012          0         0
## 2 MMC(A, B)          100   0.282      282     0.276    0.000          0         0

### Function compiling it self - it is OK
MM <- cmpfun(MM)
benchmark(MM(A, B), MMC(A, B), A%*%B)

##        test replications elapsed relative user.self sys.self user.child sys.child
## 3   A %*% B          100   0.001        1     0.004        0          0         0
## 1  MM(A, B)          100   0.259      259     0.260        0          0         0
## 2 MMC(A, B)          100   0.235      235     0.236        0          0         0
```

3.3 Recursion

```
##### Compiling. Good for looping or smth.
    # 1. Avoid it,  if you know alternatives
    # 2. Do not use Recall (unless you really need renaming)
    # 3. If possible use some kind memoise
    # 4. Compile (with same name)

# Very simple Fibonachi functin
fibonacci1 <- function(seq) {
  if (seq == 1) return(1);
  if (seq == 2) return(1);
  return (fibonacci1(seq - 1) + fibonacci1(seq - 2));
}

# same function using `Recall`
fibonacci2 <- function(seq) {
  if (seq == 1) return(1);
  if (seq == 2) return(1);
  return (Recall(seq - 1) + Recall(seq - 2)); # Do not use! it jus slow down
}

# Fibonachi with memose
library("memoise")
fibonacci1M <- memoise(fibonacci1)  # must be with new name

# best alternative is use your own memose
fibonacci_M <- local({
  memo <- c(1, 1, rep(NA, 100000))
  f <- function(x) {
    if(x == 0) return(0)
    if(x < 0) return(NA)
    if(x > length(memo))
      stop("x too big for implementation")
    if(!is.na(memo[x])) return(memo[x])
    ans <- f(x-2) + f(x-1)
    memo[x] <<- ans
    ans
  }
})

require("rbenchmark")
n <- 20
benchmark(fibonacci1(n), fibonacci2(n), fibonacci1M(n), fibonacci_M(n), order=NULL)

##             test replications elapsed relative user.self sys.self user.child sys.child
## 1  fibonacci1(n)          100   1.445     1445     1.440        0          0         0
## 2  fibonacci2(n)          100   1.870     1870     1.864        0          0         0
## 3 fibonacci1M(n)          100   0.007        7     0.008        0          0         0
## 4 fibonacci_M(n)          100   0.001        1     0.000        0          0         0
```

3.4 Rcpp(C++)

The package `Rcpp` allows to impliment C++ code easily and with minimal knowledge. Main links:

- http://adv-r.had.co.nz/Rcpp.html
- http://dirk.eddelbuettel.com/code/rcpp/Rcpp-quickref.pdf

- http://cran.r-project.org/web/packages/Rcpp/index.html
- http://www.rcpp.org/
- http://dirk.eddelbuettel.com/code/rcpp/html/index.html

All R types are supported (vectors, functions, environment, etc ...):
IntegerVector, NumericVector, LogicalVector, CharacterVector,
IntegerMatrix, NumericMatrix, LogicalMatrix, CharacterMatrix.

The types that do not have C types gos only with capital letter:
List, Function, Environment,..

Good reference could be fould in `Rcpp-quickref.pdf`. Here is very basic

```
// Geting values
  x[i]
  x(i,j)

//geting info
  .size()
  .nrow()
  .ncol()
  .length()

//Very importas is iterator, see in examples
  ::iterator
  .begin()
  .end()

//basic procedures
  .create
  .fill
  .import
  .insert

//misc
  .erase
  .eval
  .get_na
  .is_na
  .offset
  .sort
```

Very first example of making C++ in R. It is jus like 'Hellow world':

```
library("Rcpp")
cppFunction('
  int add(int x, int y, int z) {
    int sum = x + y + z;
    return sum;
  }'
)
add(1, 2, 3)

## [1] 6
```

An example of misc basics. Writing mean function in several ways. See comments:

```
sourceCpp(code=
















































 )
x <- rnorm(10^2)

# having error form C++:
meanC(x, type = 0)

## Error:  There are no such type.  (stop).

# Comparing
all.equal(mean(x), meanC(x, type=1), meanC(x, type=2), meanC(x, type=3))

## [1] TRUE

library("microbenchmark")
microbenchmark(mean(x), meanC(x, type = 1), meanC(x, type = 2), meanC(x, type = 3))
```

```
## Unit: microseconds
##                expr   min    lq median    uq   max neval
##             mean(x) 5.898 6.685  6.989 7.497 39.02   100
##  meanC(x, type = 1) 1.969 2.177  2.510 2.971  6.50   100
##  meanC(x, type = 2) 1.946 2.240  2.516 3.312 13.85   100
##  meanC(x, type = 3) 1.958 2.268  2.654 3.131 15.55   100

head(x)

## [1] 1000.00000    0.18463   -0.01969   -1.08211    0.55824   -1.41652
```

A quickref for matrix

```
SEXP x;
NumericMatrix xx(x);

// Matrix of 4 rows & 5 columns (filled with 0)
NumericMatrix xx(4, 5);

// Fill with value
int xsize = xx.nrow() * xx.ncol();
for (int i = 0; i < xsize; i++) {
  xx[i] = 7;
}
// Same as above, using STL fill
std::fill(xx.begin(), xx.end(), 8);

// Assign this value to single element
// (1st row, 2nd col)
xx(0,1) = 4;

// Reference the second column
// Changes propagate to xx (same applies for Row)
NumericMatrix::Column zzcol = xx( _, 1);
zzcol = zzcol * 2;
// Copy the second column into new object
NumericVector zz1 = xx( _, 1);
// Copy the submatrix (top left 3x3) into new object
NumericMatrix zz2 = xx( Range(0,2), Range(0,2));
```

The example of ussing C++ librarys and `iterator`. The eqvivalent of the function `findInterval`.

```
sourceCpp(code=
 )
x <- rnorm(10^1)
breaks <- c(-Inf, -3, -2, -1, 0, 1, 2, 3, Inf)
all.equal(findInterval(x, breaks), findInterval2(x, breaks))

## [1] TRUE

microbenchmark(findInterval(x, breaks), findInterval2(x, breaks))

## Unit: microseconds
##                      expr   min    lq median    uq   max neval
##   findInterval(x, breaks) 2.344 2.562  2.764 2.907 15.93   100
##  findInterval2(x, breaks) 1.526 1.704  1.946 2.269 23.58   100
```

An example of best abstraction: List and Function, without knowing anything C code wors just fine (but slow).

```
# Function and List: works in CPP, so you can actualy use R function in CPP,
# but it acually quite slow. Therefore, if posible use R function ir R and C funciton in C.
# But list is quite usefull - in  fact it can contain any object with out knowing.
cppFunction(code='
List lapply1(List input, Function f) {
  int n = input.size();
  List out(n);
  for(int i = 0; i < n; i++) {
    out[i] = f(input[i]);
  }
  return out;
}
')
foo <- function(n) 2^n + 1
microbenchmark(lapply1(1:3, foo), lapply(1:3, foo))

## Unit: microseconds
##               expr    min     lq median    uq    max neval
##  lapply1(1:3, foo) 58.390 60.315 61.624 67.79 143.06   100
##   lapply(1:3, foo)  4.672  5.423  6.621  7.21  16.42   100
```

An example of regular arrays(C) and compatability with R.

```
# Technicly, array in R is jus a vector with atrributes. So C eqvivalnet is Vector.
# There is no eqvivalnet to get C array (but we have matrix, or 3D cube - they are special calsses)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;
// REturning a list with all our toys
// [[Rcpp::export]]
List arr(NumericVector input, IntegerVector dim, IntegerVector dummy) {

  Dimension d(dim);                  // get the dim object

  // making R array with dimensions
  NumericVector array(d);            // create vec. with correct dims
  std::copy(input.begin(), input.end(), array.begin());  // copy - it is not optimal, it just an example
  // now `array` is  NumericVector, but in R it will be an array with dimesnions

  // much better way is to assign dim attribute
  input.attr("dim") = d;

  return List::create(
    _["input"] = input,
    array     // no name
  );
}
')

x = 1:8
y = arr(x, c(2,2,2), 1:5)
```

From C to R, From R to C

```
sourceCpp(code='
#include <iterator>
#include <vector>
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
List RC(NumericVector input) {

  // From C to R #1
  double mynum[] = {0.5,1776,7,4};   // Creating regular C array
  std::vector<double> foo;           // Creating regualr C vector
  foo.assign(mynum, mynum + sizeof mynum / sizeof mynum[0]); //assigning values (could be done in creation
  SEXP bar = wrap (foo);


  // From C to R #2 - directly in R eqvivanlent
  int myint[] = {1776,7,4};
  NumericVector bar2(myint, myint + sizeof myint / sizeof myint[0]);

  // From R to C #1
  double* a = &input[0];
  std::cout << a[0] << ", " << a[1]<< ", ...\\n";

  // From R to C #2
  double a2[100];
  std::copy(input.begin(), input.end(), a2);
  std::cout << a2[0] << ", " << a2[1]<< ", ...\\n";


  return List::create(
    bar,
    bar2
  );
}
')

x = 1:8
RC(x)
```

Fast linar algebra, ussing extra package (and it tipes, that are compatable with R)

```
sourceCpp(code='
#include <RcppArmadillo.h>
//[[Rcpp::depends(RcppArmadillo)]]

#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
List fastLm(NumericVector yr, NumericMatrix Xr) {

    int n = Xr.nrow(), k = Xr.ncol();

    arma::mat X(Xr.begin(), n, k, false);       // reuses memory and avoids extra copy
    arma::colvec y(yr.begin(), yr.size(), false);

    arma::colvec coef = arma::solve(X, y);      // fit model y ~ X
    arma::colvec resid = y - X*coef;            // residuals

    double sig2 = arma::as_scalar( arma::trans(resid)*resid/(n-k) );
                                                // std.error of estimate
    arma::colvec stderrest = arma::sqrt( sig2 * arma::diagvec( arma::inv(arma::trans(X)*X)) );

    return Rcpp::List::create(
        Rcpp::Named("coefficients") = coef,
        Rcpp::Named("stderr")       = stderrest
    ) ;
}

//[[Rcpp::export]]
arma::mat mult(arma::mat A, arma::mat B) {
  return A*B;
}

')

A <- matrix(1:9, 3, 3);
B <- matrix(9:1, 3, 3);

microbenchmark(A%*%B, mult(A, B))

## Unit: microseconds
##        expr   min    lq median    uq   max neval
##     A %*% B 1.113 1.370  1.825 1.958 64.32   100
##  mult(A, B) 2.782 3.625  4.040 4.427 30.11   100
```

3.5 Developing

Creating and developing packages.

Table 5: Usefull packages of ‘programing, developing‘.

package	rating	heading
devtools	7.0	Tools to make developing R code easier
formatR	7.0	Format R Code Automatically
roxygen2	7.0	In-source documentation for R
testthat	7.0	Testthat code. Tools to make testing fun :)
profr	7.0	An alternative display for profiling information
rbenchmark	7.0	Benchmarking routine for R

```
library("devtools")
help(package="devtools")

##### 1.  Creating package folder with necesary infrastructure (use one folder up)
    #  And put the content that you allready have.
    # Sugestions:
    # a) basic in-buld sunction
    package.skeleton()
    # b) devtools function
    create()
    # c) RStudio meniu
    # d) From other packages. Very usefull for optional failes.

##### 2.  Codding. Start coding in R folder.
    # The loan can be done by
    load_all()  # devtools
    # If you allready have some version of the package that is curently in use you
    # can isolate developing code in
    dev_mode()  # devtools

##### 3. Then having starting code you should write tests to make sure everything works as
    # intended to be.
    library("testthat")
    ?test
    ?test_that

##### 4. Profiling
    Rprof
    summaryRprof
    library("profr")
    help(package="profr")

##### 5. Preparing code. Making well format, commenting (+documentation in comment with roxygen2)
    library("formatR")
    help(package="formatR")


##### 6. Documentation
  ### For functions. use one of the strategies (you can mix them, but it is not rezomended):
    # a) manual documentation
    ?prompt #Produce Prototype of an R Documentation File
    check_doc()
    # b) in-code documentation
    library("roxygen2")
    ?document  # Use roxygen to make documentation
    ?roxygenize
  ### Do not foget to
    # Writing ReadMe
    # writing vinigete.
    # demo
    # package description
    dev_help("Vmisc")
    build_vignettes()

##### 7. local install
    install(quick=TRUE)
    library("Vmisc")
    help(package="Vmisc")


##### 8. final scheck, and bild
    check()
    build()
    # ?release
```

3.5.1 Vignettes

- A vignettes should be write in *vignettes* package. All necessary files should be in this directory (or sub-directory).
- The vignettes could be Rn(pd) or Rd(html). Let assume it is Rnw.
- Meta data should be included in the comments. Main list:

  ```
  %\VignetteEngine{knitr::knitr}
  %\VignetteIndexEntry{Just a pdf example}
  %\VignetteDepends{}
  %\VignetteKeywords{string, misc}
  %\VignettePackage{Vmisc}
  ```

- If Vignette source file is Rwn, then it could be clasical Sweave or other (e.g. *knitr*). If using *knitr*, then
  - it sould be decleard in meta-date comment (note the first line).
  - it sould be declear in DESCRIPTION file, line 'VignetteBuilder: knitr'
  - knitr package should be included in DESCRIPTION file 'Suggests' filed, e.g : Suggests: knitr, ...

4 Special topics

4.1 Search in environment

```
### Search in environment, base functions
?ls; ?objects     # List Objects
?search           # Give Search Path for R Objects
?apropos; ?find   # Find Objects by (Partial) Name
?get              # Return the Value of a Named Object
```

Special examples

```
head(ls("package:base", pattern="str"))

## [1] "default.stringsAsFactors" "R.version.string"         "strftime"
## [4] "strptime"                 "strsplit"                 "strtoi"

head(apropos("str"))

## [1] "austres"                  ".__C__C++Constructor"     ".__C__nonStructure"
## [4] "constrOptim"              ".__C__structure"          "default.stringsAsFactors"
```

4.2 Strings

```
### Strings:
  paste()
  format()
  formatC()
  # Substrings (vectorised, can raplece)
  substr(x, start, stop)
  substring(text, first, last = 1000000L)
  # replacments
  sub(pattern, replacement, x)
  gsub(pattern, replacement, x)
  # Find pattern
  grep(pattern, x) ;
  grepl(pattern, x)
  # find expresion
  regexpr(pattern, text)
  gregexpr(pattern, text)
  # interpretation
  eval(parse(text = "(d <- 4 + 7)"))
  substitute(expr, env)
  # misc
  strsplit("a,b;c", ",|;") # a, b, c
  nchar() # the length of string
  chartr() # traslete: L -> L
  tolower()
  toupper()
  as.numeric(as.character(f)) # factor - > number
```

Table 6: Usefull packages of ‘strings‘.

package	rating	heading
evaluate	7.0	Parsing and evaluation tools that provide more details than the default
stringr	7.0	Make it easier to work with strings

Special examples

```
### Basic operations
text <- "Hellow, # comment

     The line after empty line       with     lots    of    sapces.
```

```r
"
doc <- readLines(textConnection(text))
# remove comments
doc <- gsub("#.+$", "", doc)
# remove dublicate space
doc <- gsub("\\s+", " ", doc)
# Trim the rest
doc <- gsub("^\\s+|\\s+$", "", doc)
# remove empty lines
doc <- doc[nchar(doc)>0]
doc




### reverse
(a = "this is a string"); paste(rev(substring(a,1:nchar(a),1:nchar(a))),collapse="")



### vecorise string substitution
(x <- c("aaaa","bbbb","cccc")); substring( x  , 2) <- c("..", "+++"); x;



### Binary string concat
"%.%" <- function(x,y) paste(x,y,sep="")
"I love " %.% "R."



### String matrix spliting
SplitLines <- function(x, chMode, sep=" "){
  ats = do.call(rbind, strsplit(x, sep))
  if (!missing(chMode)){
    storage.mode(ats)<-chMode
  }
  return(ats)
}
text <-c(
   "1 2 3"
  ,"4 5 6"
  ,"7 8 9"
)
X <- SplitLines(text,"double")

# Alternative from data input:
X <- read.table(textConnection(text))
```

5 Graphics

Table 7: Usefull packages of ‘graph‘.

package	rating	heading
igraph	7.0	Network analysis and visualization
ggplot2	7.0	An implementation of the Grammar of Graphics
colorspace	7.0	Color Space Manipulation
RColorBrewer	7.0	ColorBrewer palettes
scales	7.0	Scale functions for graphics
labeling	7.0	Axis Labeling
rgl	7.0	3D visualization device system (OpenGL)
lattice	7.0	Lattice Graphics
gplots	7.0	Various R programming tools for plotting data
vcd	7.0	Visualizing Categorical Data
scatterplot3d	7.0	3D Scatter Plot
plotrix	7.0	Various plotting functions
aplpack	7.0	Another Plot PACKage: stem.leaf, bagplot, faces, spin3R, plotsummary, plothulls
latticeExtra	7.0	Extra Graphical Utilities Based on Lattice
munsell	7.0	Munsell colour system
iplots	7.0	iPlots - interactive graphics for R

6 Data science

6.1 Machine and Statistical Learning

Table 8: Usefull packages of ‘learning‘.

package	rating	heading
e1071	7.0	Misc Functions of the Department of Statistics (e1071), TU Wien
Metrics	7.0	Evaluation metrics for machine learning
dtw	7.0	Dynamic time warping algorithms
randomForest	7.0	Breiman and Cutler’s random forests for classification and regression
FNN	7.0	Fast Nearest Neighbor Search Algorithms and Applications
nnet	7.0	Feed-forward Neural Networks and Multinomial Log-Linear Models
neuralnet	7.0	Training of neural networks
RSNNS	7.0	Neural Networks in R using the Stuttgart Neural Network Simulator (SNNS)
kernlab	7.0	Kernel-based Machine Learning Lab
tree	7.0	Classification and regression trees
rpart	7.0	Recursive Partitioning and Regression Trees
cluster	7.0	Cluster Analysis Extended Rousseeuw et al
FactoMineR	7.0	Multivariate Exploratory Data Analysis and Data Mining with R
rattle	5.0	Graphical user interface for data mining in R

```r
# randomForest
library("randomForest")
rf <- randomForest(train, labels)
```

```r
### SVM
library("kernlab")  # pasirupina gerais defoltais ir yra daugiau metodu
sv <- ksvm(train, labels)

library("e1071")  # siame pakete yra visko, cia svm yra esmine realizacija
sv <- svm(train, labels, kernel="sigmoid") # linear sigmoid polynomial radial




# k-nearest nabahood
library(FNN)
PredTest_knn = knn(train, test, labels, k = 5)

# recursive partition
library("rpart")
rp <- rpart(labels ~ ., data = TrainDF)

# neuron networks
library("nnet")
nn <- nnet(labels ~ ., data = TrainDF[,1:20, 980:1001], size = 40)


library(neuralnet)
?neuralnet
```

A Drafts