Department of Statistics, LMU
Chair of Financial Econometrics

Risk Management Using R

Part I: An Introduction to R

Nikolay Robinzonov

14th June 2013

Abstract

This computer-lab course provides useful routines which aid our decision making in the field of Risk Management—without claiming to cover all topics of this broad subject. Our tool for analysis and for visualization is the R system for statistical computing which is freely available under http://www.r-project.org/. Therefore, the R language is presented first on a level suitable for absolute beginners.

Contents

"R is really important to the point that it's hard to overvalue it"
Daryl Pregibon, a research scientist at Google

1 Getting Started With R

The introductory steps covered here are based on Fritz Leisch' script[1] and on the official R manual "An Introduction to R".[2] The gray boxes contain the input commands and the respective output is preceded by two hash symbols ##. This simplifies the reproduction of the code examples throughout this manuscript.

One strength of R is its versatility. Anyone can extend R's capabilities by contributing additional packages, provided minimal standards are fulfilled. At the time of writing this introduction, the CRAN package repository featured 4447 available *addon* packages. A helpful guide through the vast number of these packages are the so called Task Views. Task Views are one-page summaries of packages and functions useful in certain disciplines. Since many users are not even aware of the existence of established routines in their research field, these Task Views give an impression on what is available out there. Particularly, the following Task Views could be relevant for this course:

- Computational Econometrics
- Empirical Finance
- Time Series Analysis.

1.1 Basic calculations

Almost every textbook on R starts with a simple example of its usage as a calculator and we do likewise here.

```
2+5*7

## [1] 37


5/3+2

## [1] 3.667


5/(3+2)

## [1] 1
```

[1] LMU Munich, Department of Statistics: http://www.stat.uni-muenchen.de/institut/ag/leisch/teaching/cim1213/tutorium/Rintro.R.

[2] The Comprehensive R Archive Network: http://cran.r-project.org/doc/manuals/r-release/R-intro.html.

```
5/3^2

## [1] 0.5556


(5/3)^2

## [1] 2.778


5^18

## [1] 3.815e+12


5/0

## [1] Inf


0/0  ##NaN: Not a Number

## [1] NaN
```

1.2 Help

```
## Specific help
help(Syntax)
?"+"
help("bs", try.all.packages=TRUE)
help("bs", package = "splines")
apropos("help")

## Web search (see also http://www.rseek.org/)
RSiteSearch("egarch")          # search on http://search.r-project.org

## Package tutorials
vignette(package="xtable")              # available vignettes
vignette("xtableGallery", package="xtable") # specific vignette

## General help
getOption("browser")
options(browser="/usr/bin/firefox")
help.start()                            # general HTML help version
```

1.3 Assignment

R has a special operator assignment operator <-, which consists of the two characters < (“less than”) and - (“minus”) occurring strictly side-by-side and it ‘points’ to the object

receiving the value of the expression. It also works in the other directions `->`. Note that the equal sign `=` would also work in most circumstances but it is mainly used for assigning default argument values in functions. For example `myfunc(x=2) {x+3}`. It is advisable to use the `<-` operator.

```
# Assignment operator '<-'
x <- 7

# Print variable values on the screen
print(x)

## [1] 7

x

## [1] 7
```

1.4 Variables

```
# Numeric
x <- 7
length(x)

## [1] 1

mode(x)

## [1] "numeric"

is.integer(x)

## [1] FALSE

is.double(x)

## [1] TRUE


# Logical
y <- is.numeric(x)
y

## [1] TRUE


# Character
z <- "A"
z

## [1] "A"
```

```
z <- 'test'
z

## [1] "test"


# Factor
f <- factor(c("a", "b", "a", "b", "b","c"))
f

## [1] a b a b b c
## Levels: a b c

contrasts(f)

##   b c
## a 0 0
## b 1 0
## c 0 1

str(f)

##  Factor w/ 3 levels "a","b","c": 1 2 1 2 2 3


as.numeric(f)

## [1] 1 2 1 2 2 3

as.character(f)

## [1] "a" "b" "a" "b" "b" "c"

paste(f)

## [1] "a" "b" "a" "b" "b" "c"

paste(f, collapse="")

## [1] "ababbc"

paste(x, f, sep="-")

## [1] "7-a" "7-b" "7-a" "7-b" "7-b" "7-c"
```

1.5 Working Space

```
name <- "Knut"; n1 <- 10; n2 <- 100; m <- 0.5

ls()
```

```
##  [1] "command"  "encoding" "f"        "file"     "m"        "n1"
##  [7] "n2"       "name"     "od"       "x"        "y"        "z"

ls.str()

## command : function (input, output = NULL, tangle = FALSE, text = NULL, quiet = FALSE,
##     envir = parent.frame(), encoding = getOption("encoding"))
## encoding :  chr "utf-8"
## f :  Factor w/ 3 levels "a","b","c": 1 2 1 2 2 3
## file :  chr "/home/robinzoni/lmu/teaching/risk_management/01_handout/RiskRintro.Rnw"
## m :  num 0.5
## n1 :  num 10
## n2 :  num 100
## name :  chr "Knut"
## od :  chr "/home/robinzoni/lmu/teaching/risk_management/01_handout"
## x :  num 7
## y :  logi TRUE
## z :  chr "test"


save(x, name, m, file="somedata.RData")
rm(name, n1)
ls()

##  [1] "command"  "encoding" "f"        "file"     "m"        "n2"
##  [7] "od"       "x"        "y"        "z"

load("somedata.RData")
ls()

##  [1] "command"  "encoding" "f"        "file"     "m"        "n2"
##  [7] "name"     "od"       "x"        "y"        "z"


## working directory
getwd()

## [1] "/home/robinzoni/lmu/teaching/risk_management/01_handout"

## On Windows:
## setwd("c:\\temp") or setwd("/tmp")
```

1.6 Vectors

R operates on named data structures. The simplest such structure is the numeric vector, which is a single entity consisting of an ordered collection of numbers.

```
# Combine elements
x <- c(5, 2, 8, 99)
x

## [1]  5  2  8 99
```

```
x <- c(x, 3, 4)
x
```

```
## [1]  5  2  8 99  3  4
```

```
y <- c("A", "B")
y
```

```
## [1] "A" "B"
```

```
z <- c(TRUE,FALSE,TRUE)
```

```
xz <-c(x,z)    # implicit type conversion
xz
```

```
## [1]  5  2  8 99  3  4  1  0  1
```

```
c(1, 2, "a", "b") # implicit type conversion
```

```
## [1] "1" "2" "a" "b"
```

```
as.numeric(z) # explicit type conversion
```

```
## [1] 1 0 1
```

```
as.character(1:2)# explicit type conversion
```

```
## [1] "1" "2"
```

```
# type and length
mode(z)
```

```
## [1] "logical"
```

```
mode(x)
```

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

```
is.numeric(x)
```

```
## [1] TRUE
```

```
is.logical(x)
```

```
## [1] FALSE
```

```
is.character(y)
```

```
## [1] TRUE
```

```
length(z)

## [1] 3


# sequences
seq(from=0, to=10, by=2)

## [1]  0  2  4  6  8 10

seq(from=0, to=10, length=5)

## [1]  0.0  2.5  5.0  7.5 10.0

seq(to=10)

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

1:10

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


# repetitions
rep(0, times=5)

## [1] 0 0 0 0 0

rep(1:3, times=5)

##  [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3

rep(1:3, length=5)

## [1] 1 2 3 1 2

rep(1:3, each=5)

##  [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3


# access elements
x <- 2:7
x

## [1] 2 3 4 5 6 7

x[2]

## [1] 3

x[-3]

## [1] 2 3 5 6 7
```

```
x[2:4]
## [1] 3 4 5
x[x > 3]
## [1] 4 5 6 7
x[c(1, 3:5)]
## [1] 2 4 5 6
x[2:4] <- 10
x
## [1]  2 10 10 10  6  7

# vector computations
x <- 1:3
y <- 3:1
x+1
## [1] 2 3 4
x+y
## [1] 4 4 4
sum(x)
## [1] 6
cumsum(x)
## [1] 1 3 6
prod(y)
## [1] 6
cumprod(y)
## [1] 3 6 6
x*y
## [1] 3 4 3
exp(x)
## [1]  2.718  7.389 20.086
x^2
## [1] 1 4 9

# missing values (Not Available)
x <- c(7, NA, 3, 5)
max(x)
## [1] NA
max(x, na.rm=TRUE)
## [1] 7
```

1.7 Matrices

```
# matrix creation
m <- matrix(1:12, ncol=4, nrow=3)
m

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

m <- matrix(m, ncol=4, nrow=3, byrow=T)
m

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


# access elements
m[1,2]

## [1] 2

m[,1]

## [1] 1 5 9

m[2,]

## [1] 5 6 7 8

m[2:3, -4]

##      [,1] [,2] [,3]
## [1,]    5    6    7
## [2,]    9   10   11


# type and dimension
mode(m)

## [1] "numeric"

class(m)

## [1] "matrix"

length(m)

## [1] 12

dim(m)
```

```
## [1] 3 4

attributes(m)

## $dim
## [1] 3 4

rownames(m) <- c("but", "or", "and")
attributes(m)

## $dim
## [1] 3 4
##
## $dimnames
## $dimnames[[1]]
## [1] "but" "or"  "and"
##
## $dimnames[[2]]
## NULL

m

##     [,1] [,2] [,3] [,4]
## but    1    2    3    4
## or     5    6    7    8
## and    9   10   11   12


# combining vectors and matrices
x <- 1:3
y <- 3:1
m <- cbind(x, y)
m

##      x y
## [1,] 1 3
## [2,] 2 2
## [3,] 3 1

m <- rbind(x, y)
m

##   [,1] [,2] [,3]
## x    1    2    3
## y    3    2    1


# transpose and matrix-multiplication
t(m)

##      x y
## [1,] 1 3
## [2,] 2 2
## [3,] 3 1
```

```
m2 <- m %*% t(m)
m2

##    x  y
## x 14 10
## y 10 14

x <- rep(1, 2)
m2 %*% x

##   [,1]
## x   24
## y   24

x %*% m2

##       x  y
## [1,] 24 24


# diagonal matrix
diag(1, nrow=4)

##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1

diag(rep(1,4))

##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

1.8 Dates and Times

Dates and Times are notably important in econometrics. After all, the majority of our data represent time series. For a concise overview of the "Date and Time Classes in R" consider the R Help Desk article by Grothendieck and Petzoldt (2004). The Task View: Times Series Analysis gives a summary of the available functionality in R with regard to this discipline.

R provides several formats for handling date and date/time objects. The general rule is to use the simplest class possible. If you have only dates, use `as.Date`. If you in addition have minutes go for the *chron*[3] package and if also timezones are important use

[3]Note, however, that we do not include any *chron* (James and Hornik, 2012) examples here since the `Date` and the `POSIX` classes have conform format codes. Another useful–but also omitted in this

the `POSIX`[4] classes. When handling dates and times one usually starts with a conversion of existing character strings into date (or date/time) objects. In order to avoid ambiguity we need to precisely indicate the time units through the `format` argument. In the following table we see some important abbreviations and below we demonstrate their usage through examples.

Format code	Value
`%d`	Day of the month (decimal number)
`%u`	Weekday as a decimal number (1-7, Monday is 1)
`%A`	Weekday as character
`%a`	Abbreviated weekday
`%e`	Day of the month as decimal number (1-31)
`%m`	Month (decimal number)
`%b`	Month (abbreviated)
`%B`	Month (full name)
`%y`	Year (2 digit)
`%Y`	Year (4 digit)

```
## date
as.Date("2013-06-14")

## [1] "2013-06-14"

as.Date("06/14/2013", format = "%m/%d/%Y")

## [1] "2013-06-14"

d <- as.Date(c("2013-06-14", "2013-06-23"))
format(d, "%A")

## [1] "Friday" "Sunday"

weekdays(d) # quarters(d); months(d)

## [1] "Friday" "Sunday"



## convert characters into date/time: strptime
(a <- strptime("14.06.2013 10:15", format = "%d.%m.%Y %H:%M"))

## [1] "2013-06-14 10:15:00"

strptime("20130614 10:15", format = "%Y%m%d %H:%M")
```

introduction–package is the *zoo* package (Zeileis and Grothendieck, 2005) which allows convenient computations with monthly and quarterly observations.

[4]Note that the `POSIXlt` class (a list) and the `POSIXct` class (the number of seconds since 1 January 1970, the epoch) differ in the way the values are stored internally.

```
## [1] "2013-06-14 10:15:00"

strptime("06/14/13 10.15", format = "%m/%d/%y %H.%M")

## [1] "2013-06-14 10:15:00"

strptime("14.06.2013 10:15", format = "%d.%m.%Y %H:%M")

## [1] "2013-06-14 10:15:00"


## the POSIX* format
strptime("2013-06-14 10:15")   # error (provide format)

## Error:  argument "format" is missing, with no default

as.POSIXlt("2013-06-14 10:15") # tries to be intelligent

## [1] "2013-06-14 10:15:00"

as.POSIXct("2013-06-14 10:15") # also tries to be intelligent

## [1] "2013-06-14 10:15:00 CEST"

## convert seconds (since the epoc) into POSIXct
x <- 1371197700
class(x) <- c("POSIXt","POSIXct") # structure(x, class=c('POSIXt','POSIXct'))
x

## [1] "2013-06-14 10:15:00 CEST"


## convert date/time objects into character
format(a, "%A")                          # weekday

## [1] "Friday"

strftime(a, format="%A")

## [1] "Friday"

weekdays(a)

## [1] "Friday"

a$wday

## [1] 5


## various calculations (date)
d[1] < d[2]

## [1] TRUE
```

```
d[2] - d[1]

## Time difference of 9 days

d - 10

## [1] "2013-06-04" "2013-06-13"

seq(d[1], d[2], by = "day")

##  [1] "2013-06-14" "2013-06-15" "2013-06-16" "2013-06-17" "2013-06-18"
##  [6] "2013-06-19" "2013-06-20" "2013-06-21" "2013-06-22" "2013-06-23"

seq(d[1], d[2], by = "week")

## [1] "2013-06-14" "2013-06-21"

seq(as.Date("2013-4-15"), to=as.Date("2013-6-14"), by = "2 weeks")

## [1] "2013-04-15" "2013-04-29" "2013-05-13" "2013-05-27" "2013-06-10"

range(d)

## [1] "2013-06-14" "2013-06-23"

mean(d)

## [1] "2013-06-18"


## various calculations (date/time)
b <- strptime("14.06.2013 11:00", format = "%d.%m.%Y %H:%M")
a - b

## Time difference of -45 mins

difftime(a, b, units="sec")

## Time difference of -2700 secs

seq(a, b, by="sec")[1:7]

## [1] "2013-06-14 10:15:00 CEST" "2013-06-14 10:15:01 CEST"
## [3] "2013-06-14 10:15:02 CEST" "2013-06-14 10:15:03 CEST"
## [5] "2013-06-14 10:15:04 CEST" "2013-06-14 10:15:05 CEST"
## [7] "2013-06-14 10:15:06 CEST"

seq(a, by="sec", length=7)

## [1] "2013-06-14 10:15:00 CEST" "2013-06-14 10:15:01 CEST"
## [3] "2013-06-14 10:15:02 CEST" "2013-06-14 10:15:03 CEST"
## [5] "2013-06-14 10:15:04 CEST" "2013-06-14 10:15:05 CEST"
## [7] "2013-06-14 10:15:06 CEST"

as.Date(b)                                  # convert into Date

## [1] "2013-06-14"
```

1.9 Lists

```
# create lists
div <- list(vector=1:5, char="ABC", mat=matrix(1:4, ncol=2, nrow=2))
div

## $vector
## [1] 1 2 3 4 5
##
## $char
## [1] "ABC"
##
## $mat
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4

str(div)

## List of 3
##  $ vector: int [1:5] 1 2 3 4 5
##  $ char  : chr "ABC"
##  $ mat   : int [1:2, 1:2] 1 2 3 4

attributes(div)

## $names
## [1] "vector" "char"   "mat"


# access elements
div$char

## [1] "ABC"

div[[2]]

## [1] "ABC"


div[[1]][-1]

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

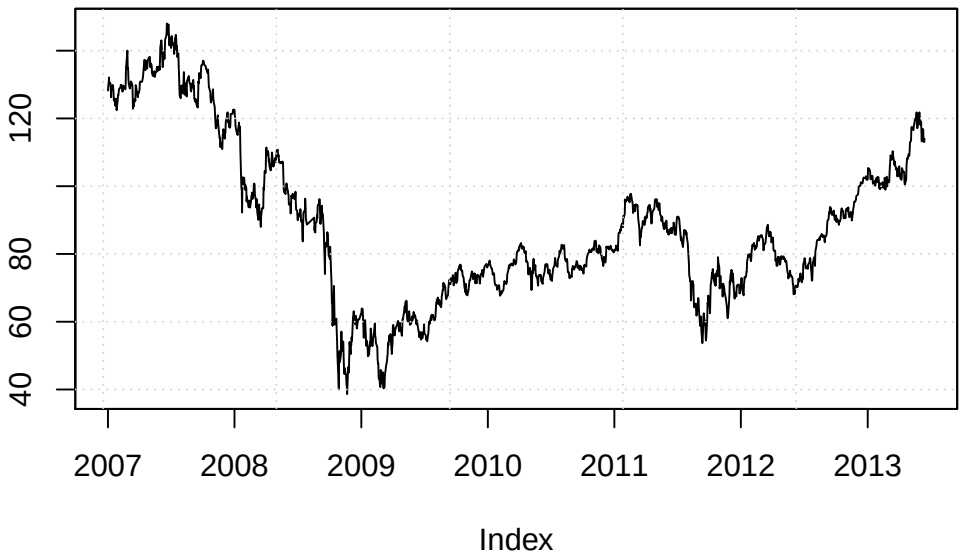
1.10 Import/Export Data

```
dat <- read.table("Rweek.csv", header=TRUE, sep=",", dec=".")
## uncomment next lines to load it from the Internet
# dat <- read.table("http://www.statistik.lmu.de/~robinzoni/riskman/data/Rweek.csv",
#                   header=TRUE, sep="\t", dec=".")
names(dat)
```

```
## [1] "date"  "sp500" "ftse"  "dax"

str(dat)

## 'data.frame': 1127 obs. of  4 variables:
##  $ date : Factor w/ 1127 levels "1984-01-05","1984-01-12",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ sp500: num  -0.628 -0.423 -1.676 -0.536 -4.86 ...
##  $ ftse : num  0.291 4.236 0.602 -1.083 -3.328 ...
##  $ dax  : num  -2.934 3.054 0.929 2.255 -4.09 ...

dat$date  <- as.Date(dat$date)
str(dat)

## 'data.frame': 1127 obs. of  4 variables:
##  $ date : Date, format: "1984-01-05" "1984-01-12" ...
##  $ sp500: num  -0.628 -0.423 -1.676 -0.536 -4.86 ...
##  $ ftse : num  0.291 4.236 0.602 -1.083 -3.328 ...
##  $ dax  : num  -2.934 3.054 0.929 2.255 -4.09 ...

dim(dat)

## [1] 1127    4

dat[1:2,]

##         date   sp500   ftse    dax
## 1 1984-01-05 -0.6279 0.2914 -2.934
## 2 1984-01-12 -0.4232 4.2359  3.054

dat$date[1:2]

## [1] "1984-01-05" "1984-01-12"

write.table(dat, file="Rweek.dat", row.names=FALSE)
write.csv(dat, file="Rweek2.csv", row.names=FALSE)

## Import MATLAB data
library(R.matlab, quietly=TRUE) # ?R.matlab
datei <- readMat("Rweek.mat")
rweek <- data.frame(datei$Rweek)
```

Getting historical financial data over the internet is also straightforward.

```
library("tseries", quietly=TRUE)
x <- get.hist.quote(start = "2007-01-01", end = "2013-06-14",
                    instrument = "alv.de", provider = "yahoo",
                    quote="AdjClose")   # adjusted for splits and dividends
plot(x$AdjClose, ylab="")
grid()
```



1.11 Functions & Packages

```
install.packages("fGarch") # .libPaths()
install.packages("fGarch", lib="~/software/Library/Rlibs",
                 repos="http://cran.at.r-project.org/")
sessionInfo()
```

```
library("splines")
search()
```

```
##  [1] ".GlobalEnv"          "package:splines"     "package:tseries"
##  [4] "package:R.utils"     "package:R.matlab"    "package:R.oo"
##  [7] "package:R.methodsS3" "package:knitr"       "package:stats"
## [10] "package:graphics"    "package:grDevices"   "ESSR"
## [13] "package:utils"       "package:datasets"    "package:methods"
## [16] "Autoloads"           "package:base"
```

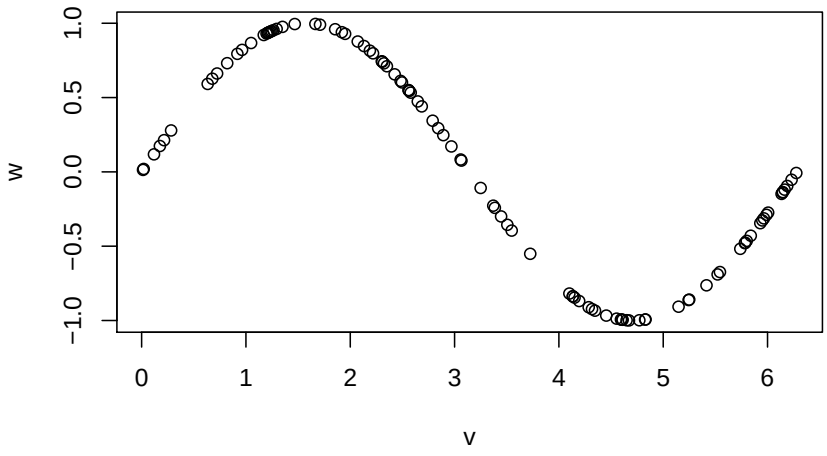
```
detach("package:splines")
search()
```

```
##  [1] ".GlobalEnv"          "package:tseries"     "package:R.utils"
##  [4] "package:R.matlab"    "package:R.oo"        "package:R.methodsS3"
##  [7] "package:knitr"       "package:stats"       "package:graphics"
## [10] "package:grDevices"   "ESSR"                "package:utils"
## [13] "package:datasets"    "package:methods"     "Autoloads"
## [16] "package:base"
```

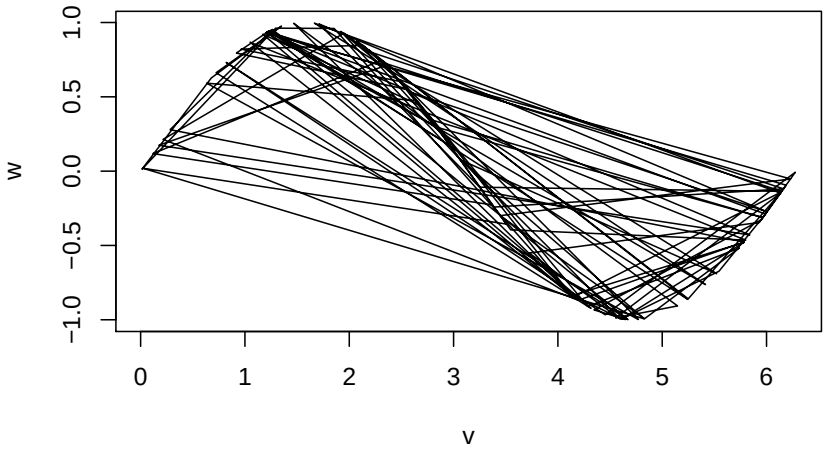
```
head(dat)

##         date   sp500    ftse     dax
## 1 1984-01-05 -0.6279  0.2914 -2.9336
## 2 1984-01-12 -0.4232  4.2359  3.0544
## 3 1984-01-19 -1.6762  0.6024  0.9291
## 4 1984-01-26 -0.5358 -1.0828  2.2549
## 5 1984-02-02 -4.8604 -3.3282 -4.0905
## 6 1984-02-09  0.4568  3.6422  3.7390

attach(dat)
sp500[1:8]

## [1] -0.6279 -0.4232 -1.6762 -0.5358 -4.8604  0.4568 -1.1785  2.5277

detach(dat)
```

1.12 Graphics

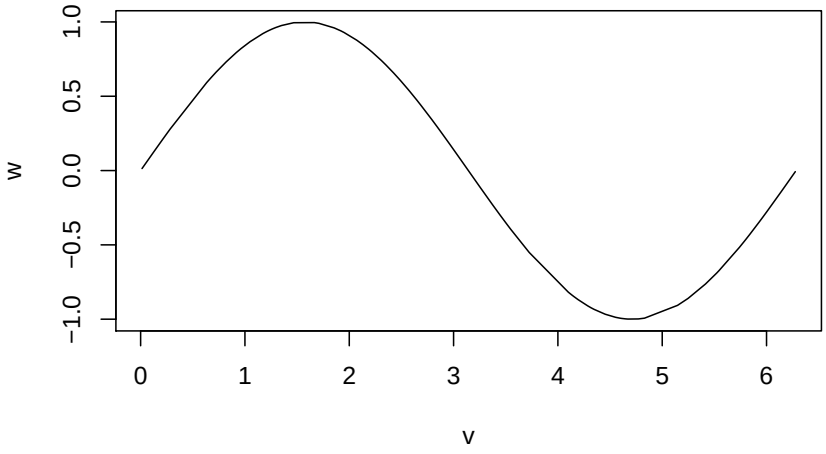
```
# create figures
v <- runif(100, 0, 2*pi)
w <- sin(v)
plot(v, w)
```



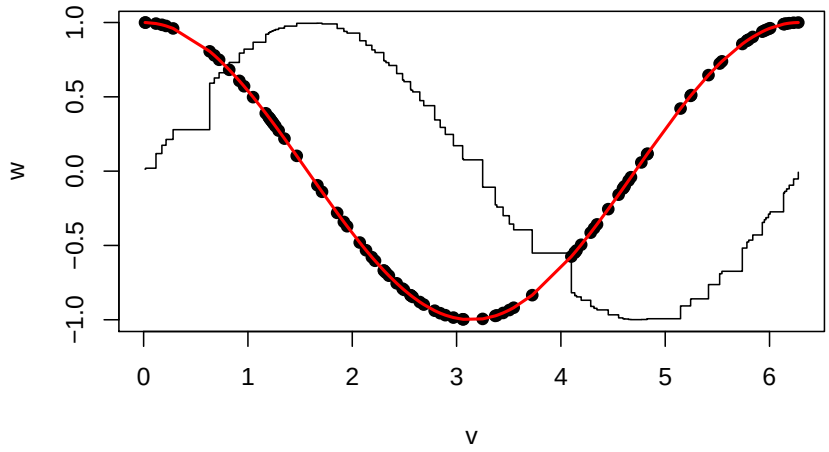

```
if (.Platform$OS.type == "unix") x11() else windows()
plot(v, w, type="l")
```



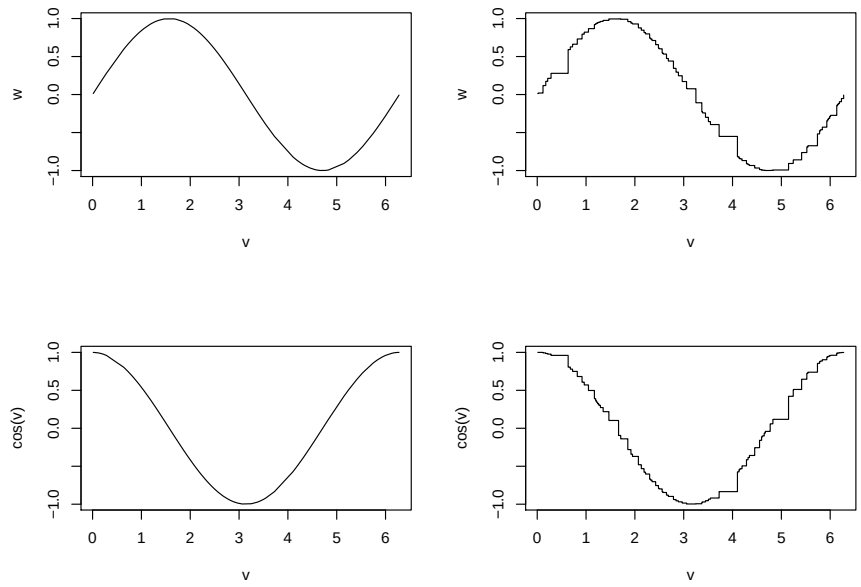

```
o <- order(v)
v <- v[o]
w <- w[o]
plot(v, w, type="l")
```




```
plot(v, w, type="s")

# add information to figures
points(v, cos(v), pch=19)
lines(v, cos(v), col="red", lwd=2)
```




```
# multiple plots in a single figure
if (.Platform$OS.type == "unix") x11() else windows()
par(mfrow=c(2, 2))
plot(v, w, type="l")
plot(v, w, type="s")
plot(v, cos(v), type="l")
plot(v, cos(v), type="s")
```



```
graphics.off()

# save as PDF
pdf(file="sinus.pdf", width=5, height=5, paper="special", pointsize=12)
plot(v, w, type="l")
dev.off()

## null device
##           1
```

1.13 Programming

```
# functions
square <- function(x) {
  return(x*x)
}

square(3)

## [1] 9


pow <- function(x, power) {
  return(x^power)
}

pow(2, 2)

## [1] 4

pow(power=3, x=2)

## [1] 8


# for-loops
forfun1 <- function(x=rep(0, 10)) {
  for(i in seq(along=x)) {
    x[i] <- x[i] + i
  }
  return(x)
}

forfun1()

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

forfun1(5:3)

## [1] 6 6 6
```

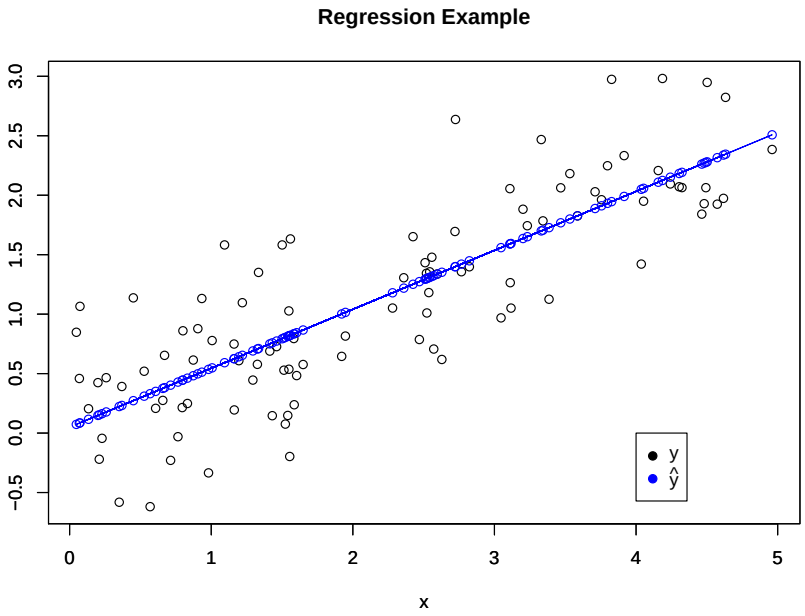
```
forfun2 <- function(x=rep(0, 10)) {
  for(i in seq(2, 10, 2)) {
    x[i] <- x[i] + i
  }
  invisible(x)
}

forfun2()
nx <- forfun2()
nx

##  [1]  0  2  0  4  0  6  0  8  0 10


# while-loops
x <- 1
while( x < 10 ) {
  x <- 2*x
  print(x)
}

## [1] 2
## [1] 4
## [1] 8
## [1] 16


# if-queries
mybernoulli <- function(prob=0.5) {
  u <- runif(1)
    if (u < prob)
      return(1)
    else
      return(0)
}
mybernoulli()     ## equal to: rbinom(1, 1, 0.5)

## [1] 0
```

1.14 Regression and Formula Objects

```
set.seed(1234)
x <- runif(100, 0, 5)
y <- 0.5*x + rnorm(100, sd=0.5)

formel <- y ~ x
formel

## y ~ x
## <environment: 0xa029bb0>
```

```
lm1 <- lm(formel)
lm1

##
## Call:
## lm(formula = formel)
##
## Coefficients:
## (Intercept)            x
##      0.0496       0.4956

is.list(lm1)

## [1] TRUE

## str(lm1)
attributes(lm1)

## $names
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
##
## $class
## [1] "lm"

lm1$coefficients

## (Intercept)           x
##     0.04959     0.49556

coef(lm1)

## (Intercept)           x
##     0.04959     0.49556

summary(lm1)

##
## Call:
## lm(formula = formel)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.017 -0.330 -0.043  0.290  1.237
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.0496     0.0892    0.56     0.58
## x             0.4956     0.0344   14.39   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.478 on 98 degrees of freedom
## Multiple R-squared:  0.679,Adjusted R-squared:  0.675
## F-statistic:  207 on 1 and 98 DF,  p-value: <2e-16
```

```
yhat <- lm1$fitted.values
yhat.test <- lm1$coef[1] + lm1$coef[2]*x

identical(yhat, yhat.test)

## [1] FALSE

range(yhat - yhat.test)

## [1] -6.661e-16  4.441e-16

all.equal(yhat, yhat.test)

## [1] "names for target but not for current"

names(yhat) <- NULL                          # yhat <- as.numeric(yhat)
all.equal(yhat, yhat.test)

## [1] TRUE


# plot the regression
plot(x, y, ylim=range(y, yhat), ylab="", main="Regression Example")
par(new=TRUE)
plot(x, yhat, type="o", col="blue", ylim=range(y, yhat), ylab="")
legend(4, 0, c("y", expression(hat(y))), col=c("black", "blue"), pch=c(19,19))
```




```
?plot.lm
```

References

Grothendieck G, Petzoldt T (2004). "R help desk: Date and time classes in R." R News, **4**(1), 29–32. URL http://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf.

James D, Hornik K (2012). chron: Chronological Objects which Can Handle Dates and Times. R package version 2.3-43. S original by David James, R port by Kurt Hornik., URL http://CRAN.R-project.org/package=chron.

Zeileis A, Grothendieck G (2005). "zoo: S3 Infrastructure for Regular and Irregular Time Series." Journal of Statistical Software, **14**(6), 1–27. URL http://www.jstatsoft.org/v14/i06/.