COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

R Programming for Quantitative Finance

Guy Yollin

Applied Mathematics
University of Washington

Outline

Lecture references

J. Adler.
R in a Nutshell: A Desktop Quick Reference.
O'Reilly Media, 2010.

- Chapters 14

W. N. Venables and D. M. Smith.
An Introduction to R.
R Core Team, 2013.

- Section 12, 8

Outline

Basic plotting functions

Function	Description
`plot`	generic function to plot an R object
`lines`	adds lines to the current plot
`points`	adds points to the current plot
`text`	adds text to the current plot
`segments`	adds lines line segments between point pairs
`arrows`	adds arrows between pairs of points
`abline`	adds straight lines to the current plot
`curve`	plot a function over a range
`legend`	adds a legend to the current plot
`barplot`	creates a bar plot with vertical or horizontal bars
`matplot`	plot all columns of a matrix
`par`	sets graphics parameters

The plot function

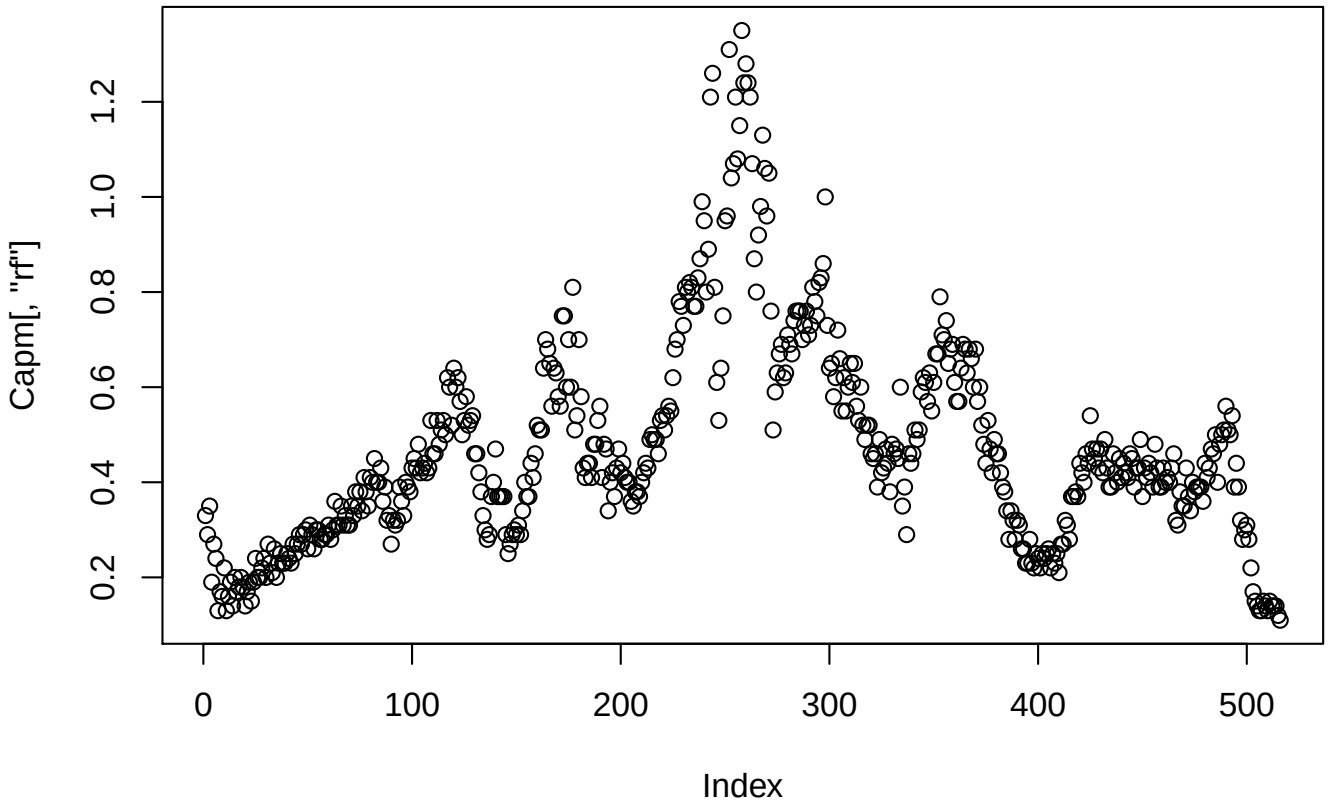
The plot function is a generic function for plotting of R objects

```
args(plot.default)

## function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
##     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
##     ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
##     panel.last = NULL, asp = NA, ...)
## NULL
```

x	vector to be plotted (or index if y given)
y	vector to be plotted
xlim/ylim	x & y limited
xlab/ylab	x & y axis labels
main	plot title (can be done with title function
type	"p" = points (default), "l" = lines, "h" = bars, "n" = no plot
col	color or bars
asp	control the aspect ratio

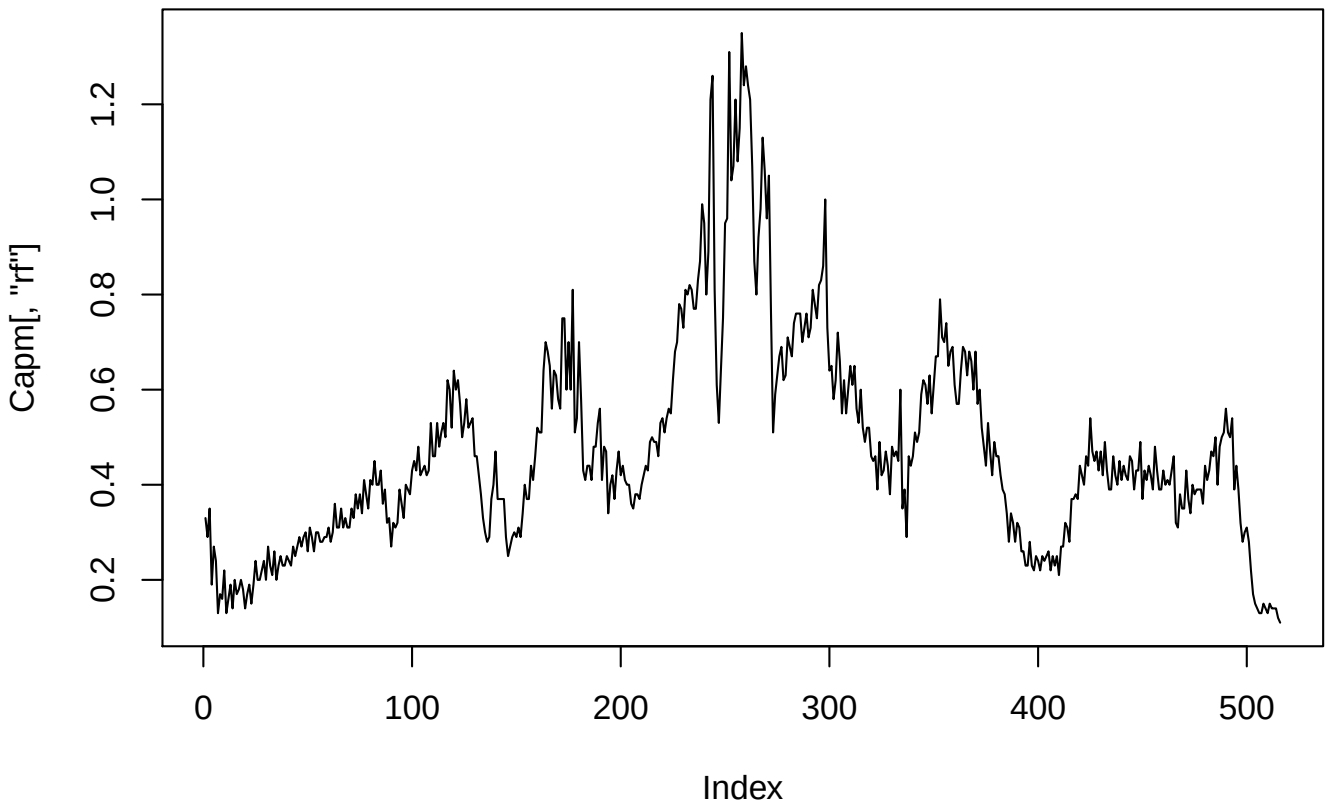
The plot function

```
library(Ecdat)
data(Capm)
plot(Capm[,"rf"])
```



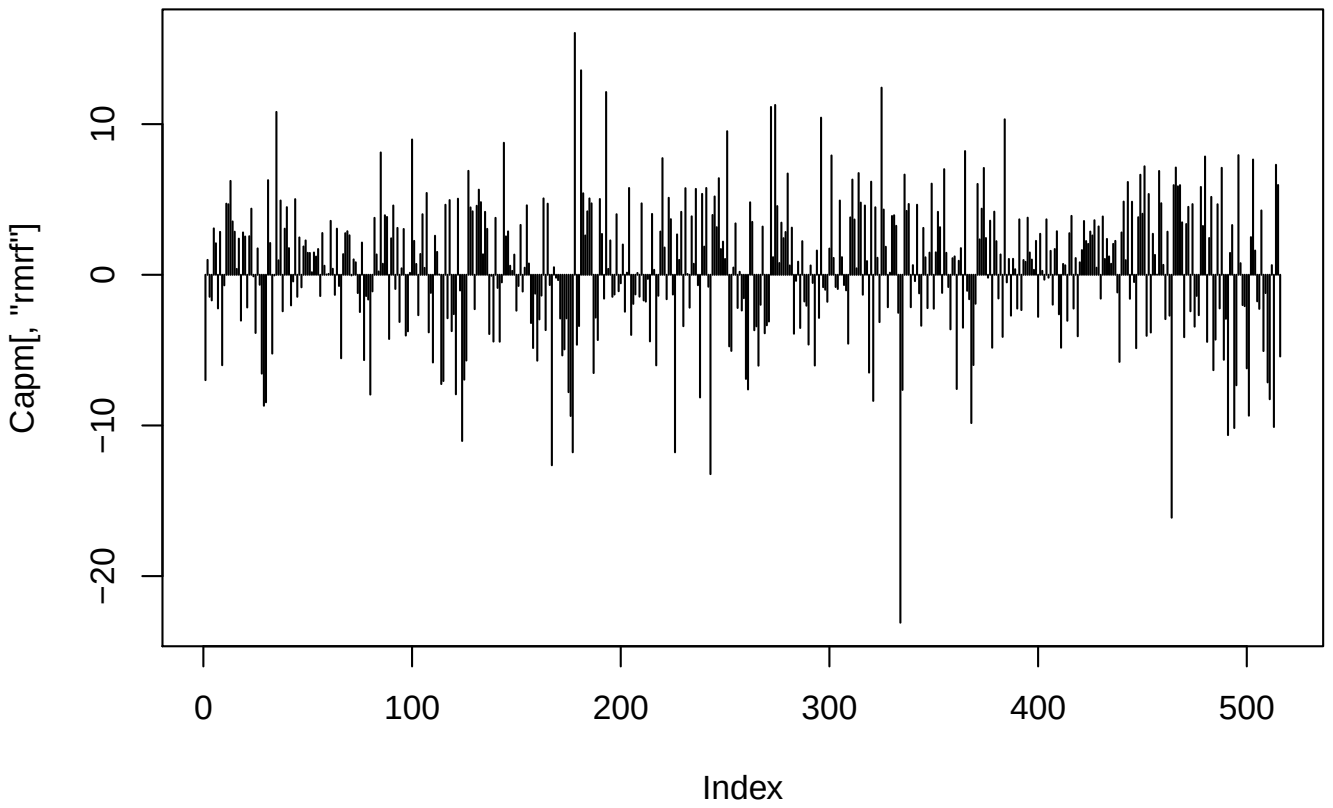
The plot function

```
plot(Capm[,"rf"],type="l")
```



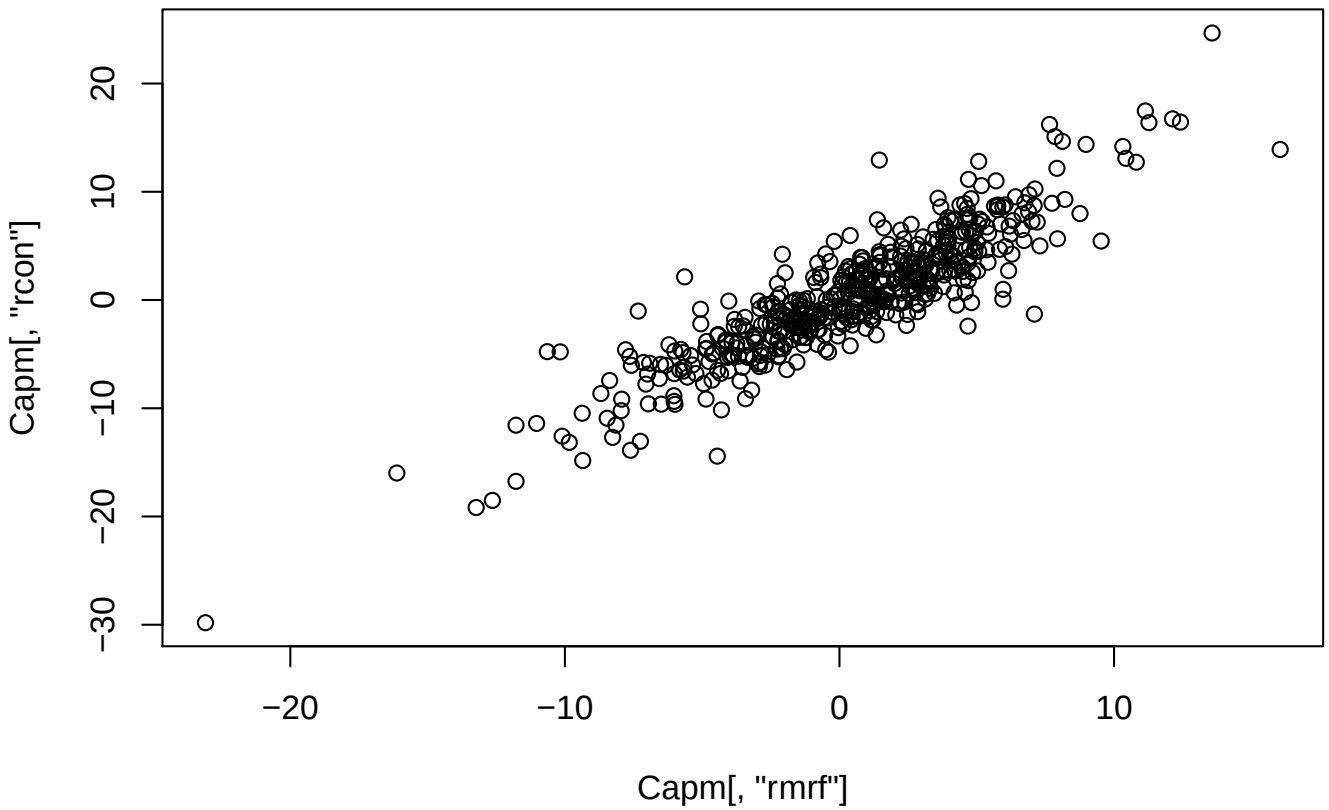
The plot function

```
plot(Capm[,"rmrf"],type="h")
```



The plot function

```
plot(Capm[,"rmrf"],Capm[,"rcon"])
```



The points function

The points function adds points to the current plot at the given x, y coordinates

```
args(points.default)

## function (x, y = NULL, type = "p", ...)
## NULL
```

x vector of x coordinates

y vector of y coordinates

The lines function

The lines function adds connected line segments to the current plot

```
args(lines.default)

## function (x, y = NULL, type = "l", ...)
## NULL
```

- x vector of x coordinates
- y vector of y coordinates

The text function

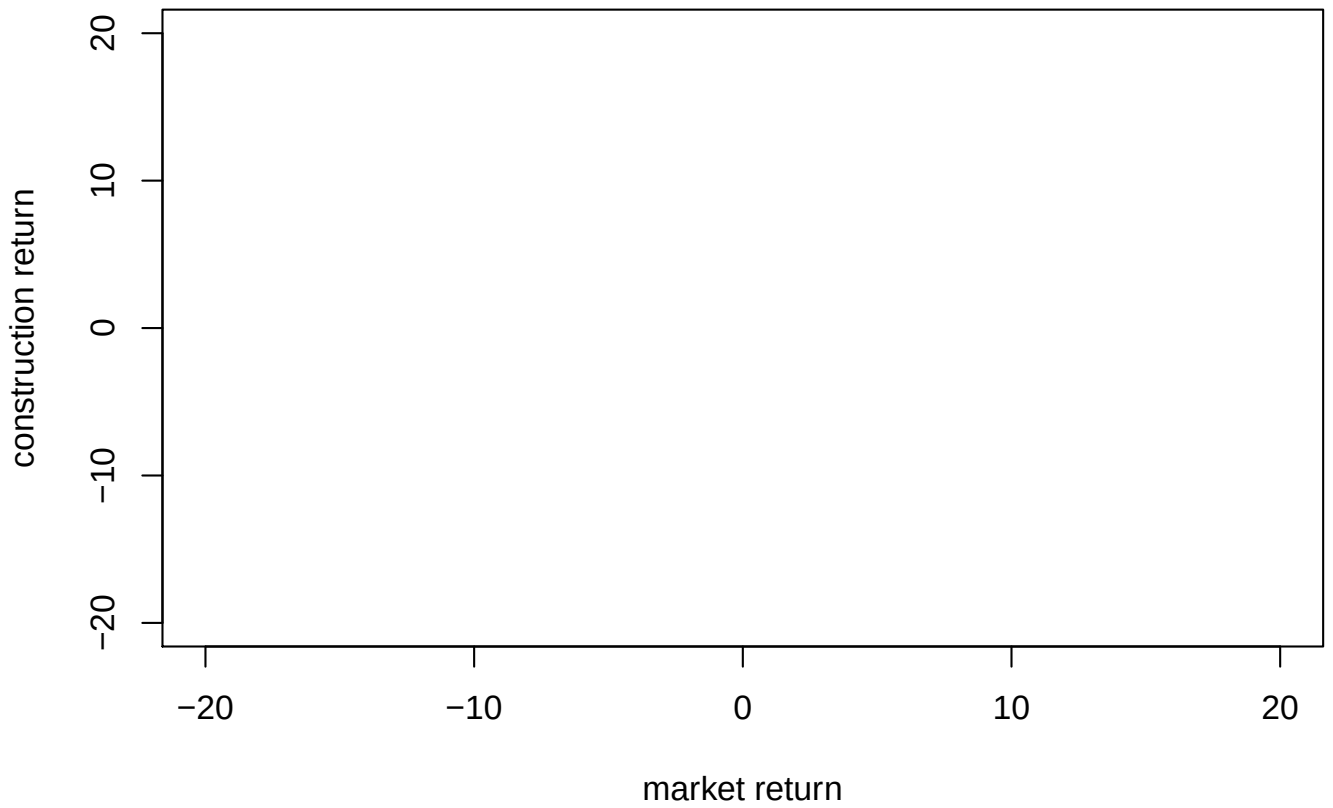
The text function adds text labels to a plot at given x, y coordinates

```
args(text.default)

## function (x, y = NULL, labels = seq_along(x), adj = NULL, pos = NULL,
##     offset = 0.5, vfont = NULL, cex = 1, col = NULL, font = NULL,
##     ...)
## NULL
```

x/y	location to place text
labels	text to be display
adj	adjustment of label at x, y location
pos	position of text relative to x, y
offset	offset from pos

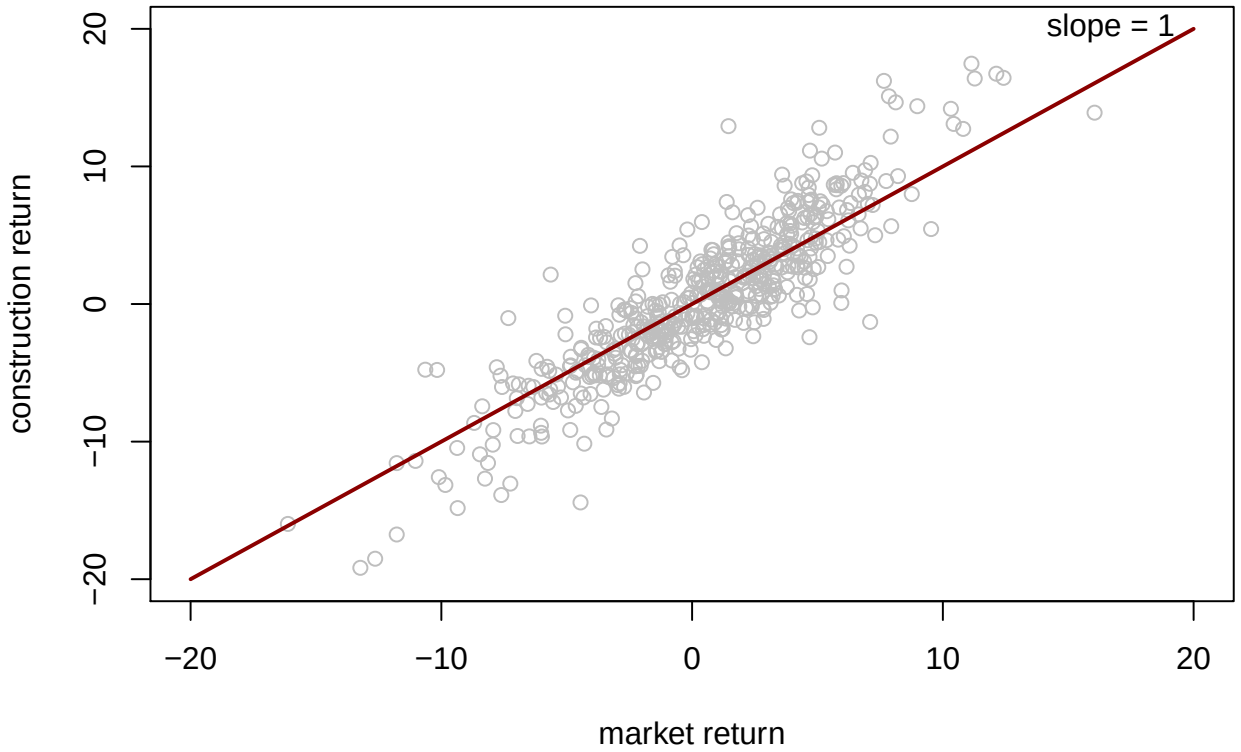
Plotting a blank frame

```
plot(0,xlim=c(-20,20),ylim=c(-20,20),type="n",
  xlab="market return",ylab="construction return")
```



A blank frame with points, lines, and text added

```
plot(0,xlim=c(-20,20),ylim=c(-20,20),type="n",
  xlab="market return",ylab="construction return")
points(x=Capm[,"rmrf"],y=Capm[,"rcon"],col="gray")
lines(x=c(-20,20),y=c(-20,20),lwd=2,col="darkred")
text(20,20,labels="slope = 1",pos=2)
```



Graphical parameters controlled via the par function

R is capable of producing publication quality graphics by allowing (*requiring*) fine-grained control of a number of graphics parameters

```
names(par())
```

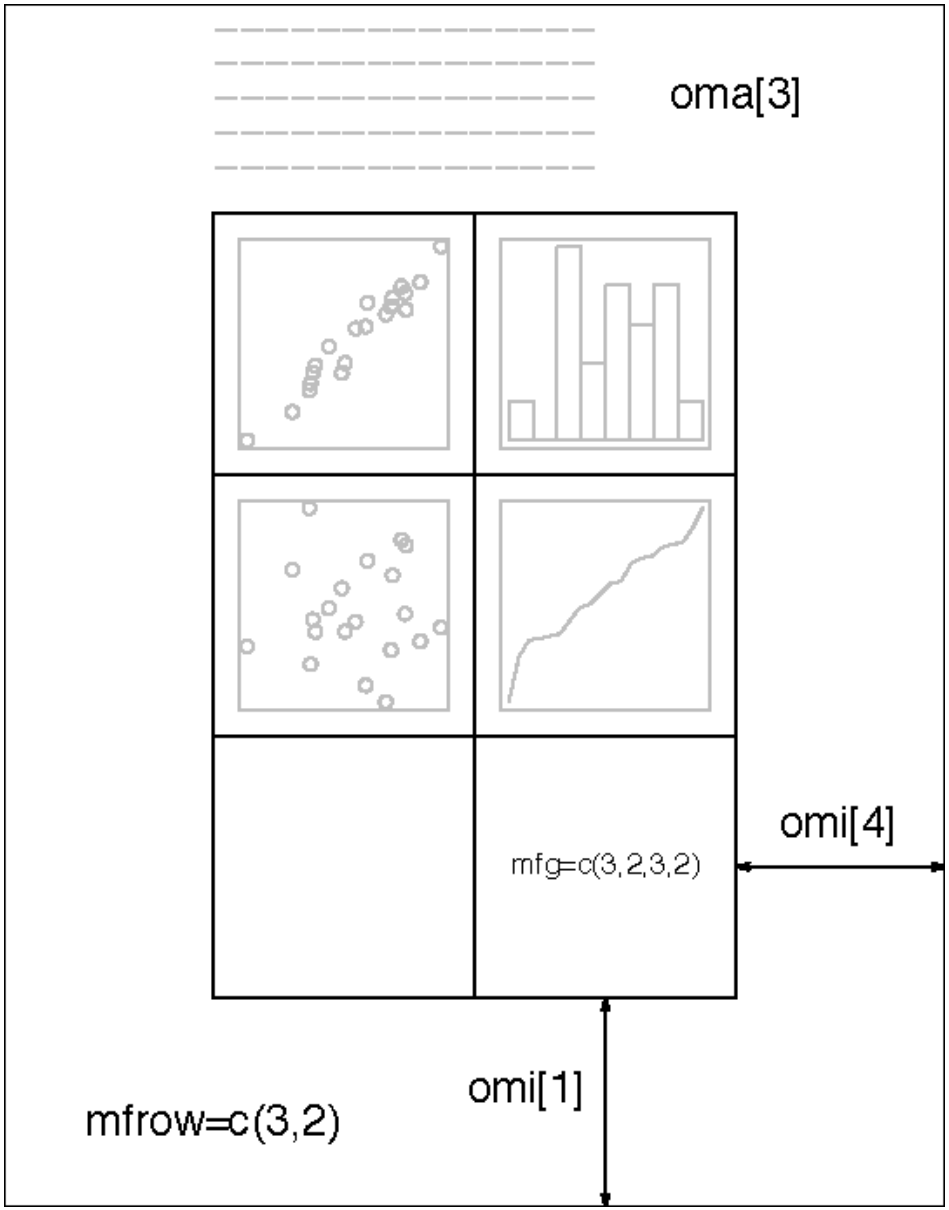
```
##  [1] "xlog"      "ylog"      "adj"       "ann"       "ask"       "bg"
##  [7] "bty"       "cex"       "cex.axis"  "cex.lab"   "cex.main"  "cex.sub"
## [13] "cin"       "col"       "col.axis"  "col.lab"   "col.main"  "col.sub"
## [19] "cra"       "crt"       "csi"       "cxy"       "din"       "err"
## [25] "family"    "fg"        "fig"       "fin"       "font"      "font.axis"
## [31] "font.lab"  "font.main" "font.sub"  "lab"       "las"       "lend"
## [37] "lheight"   "ljoin"     "lmitre"    "lty"       "lwd"       "mai"
## [43] "mar"       "mex"       "mfcol"     "mfg"       "mfrow"     "mgp"
## [49] "mkh"       "new"       "oma"       "omd"       "omi"       "page"
## [55] "pch"       "pin"       "plt"       "ps"        "pty"       "smo"
## [61] "srt"       "tck"       "tcl"       "usr"       "xaxp"      "xaxs"
## [67] "xaxt"      "xpd"       "yaxp"      "yaxs"      "yaxt"      "ylbias"
```

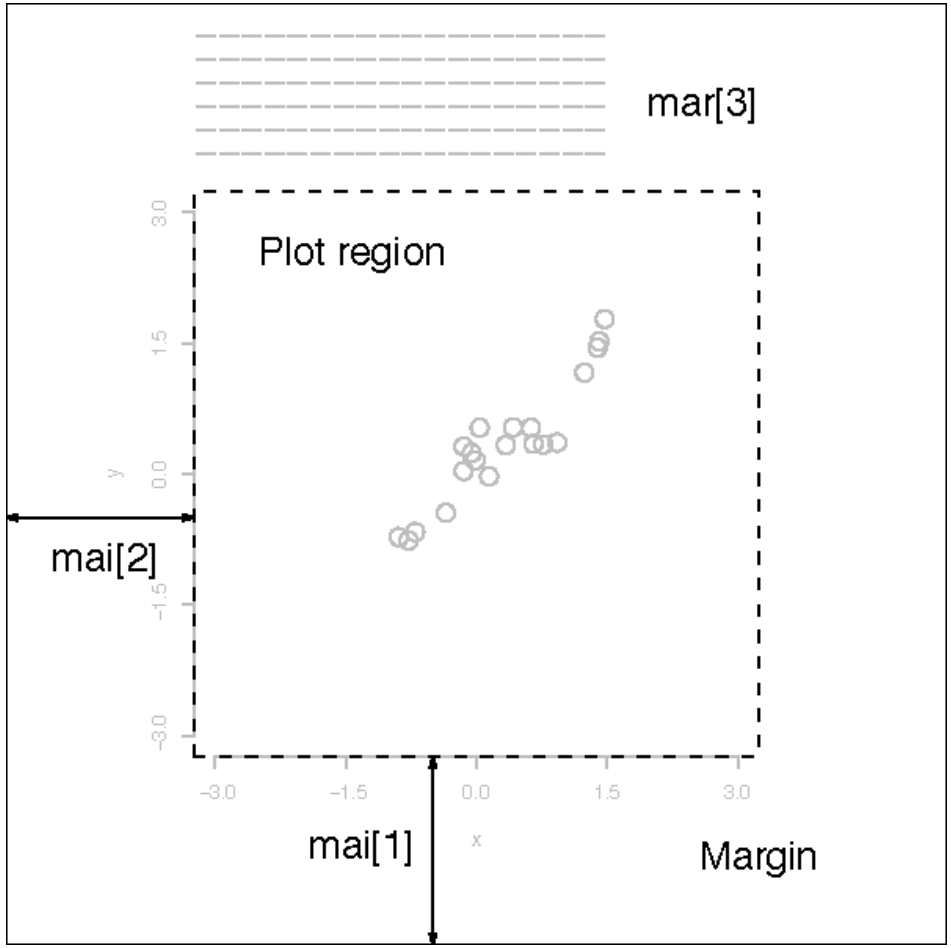
Commonly used par parameters

Parameter	Description
col	plot color
lwd	line width
lty	line type
mfrow	set/reset multi-plot layout
cex.axis	character expansion - axis
cex.lab	character expansion - labels
cex.main	character expansion - main
pch	point character
las	axis label orientation
bty	box type around plot or legend

- some parameters can be passed in a `plot` function (e.g. col, lwd)
- some parameters can only be changed by a call to `par` (e.g. mfrow)

Plot margins





The barplot function

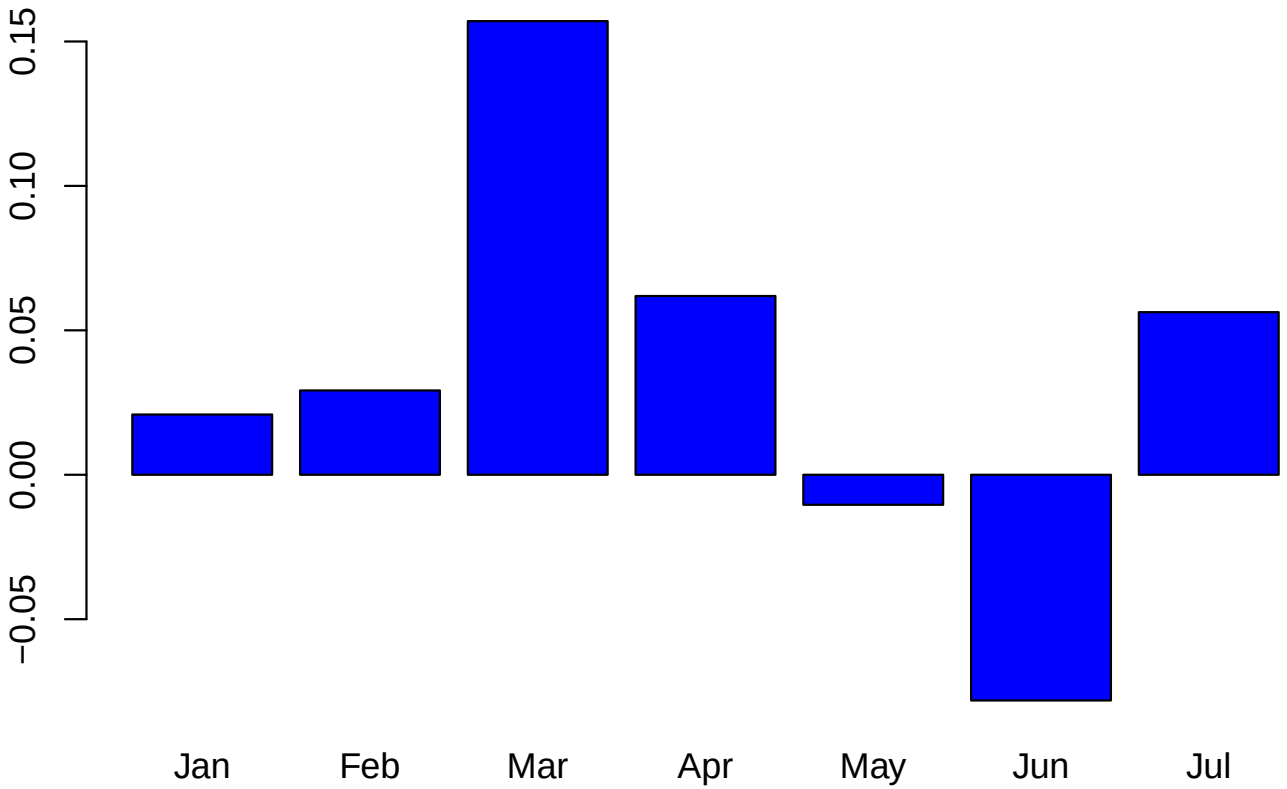
The barplot function can create vertical or horizontal barplots

```
args(barplot.default)

## function (height, width = 1, space = NULL, names.arg = NULL,
##     legend.text = NULL, beside = FALSE, horiz = FALSE, density = NULL,
##     angle = 45, col = NULL, border = par("fg"), main = NULL,
##     sub = NULL, xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
##     xpd = TRUE, log = "", axes = TRUE, axisnames = TRUE, cex.axis = par("cex.axi
##     cex.names = par("cex.axis"), inside = TRUE, plot = TRUE,
##     axis.lty = 0, offset = 0, add = FALSE, args.legend = NULL,
##     ...)
## NULL
```

height	vector or matrix (stacked bars or side-by-side bars) of heights
names.arg	axis labels for the bars
beside	stacked bars or side-by-side if height is a matrix
legend	vector of labels for stacked or side-by-side bars

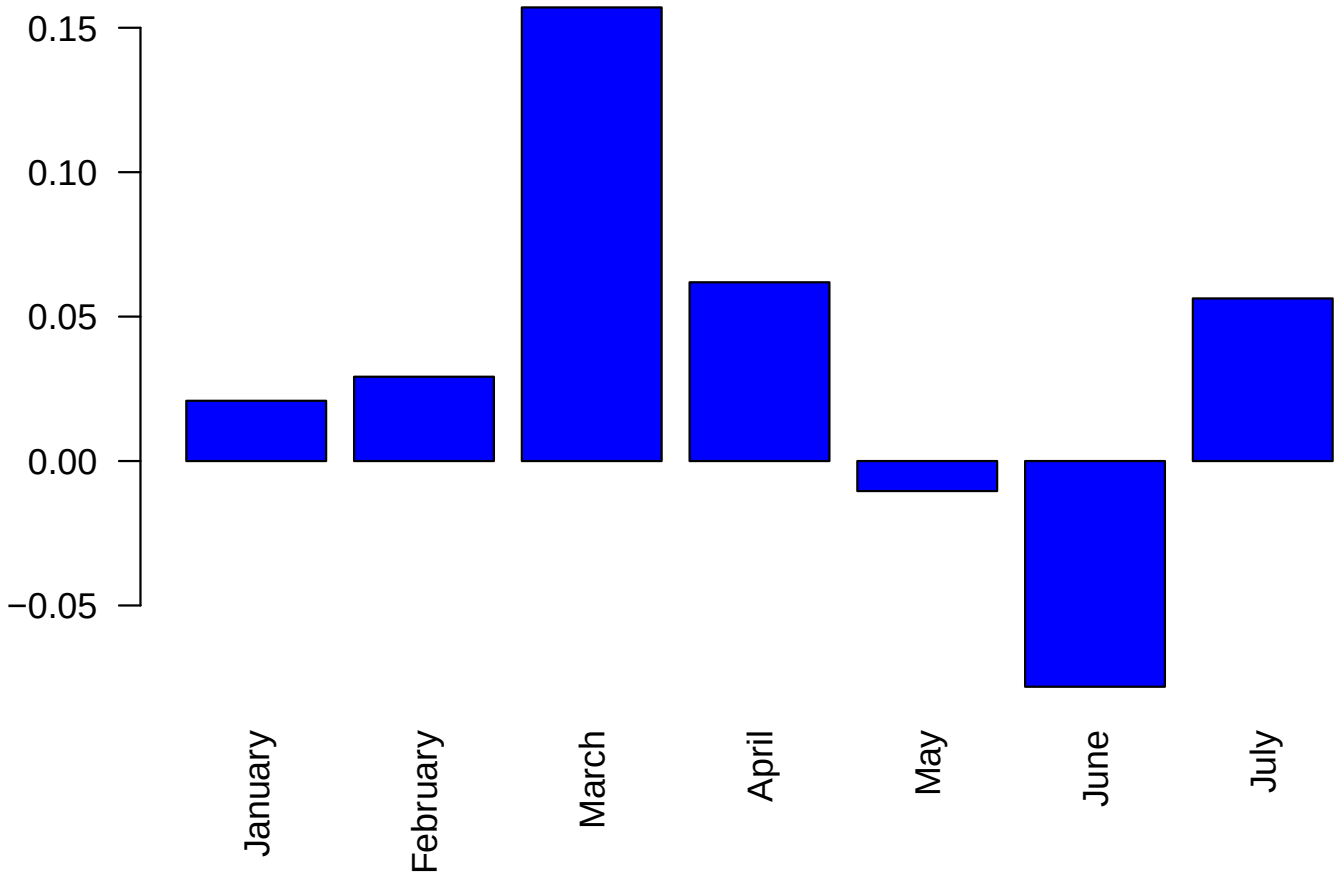
Barplot example

```
msft <- c(26.85,27.41,28.21,32.64,34.66,34.30,31.62,33.40)
msft.returns <- msft[-1] / msft[-length(msft)] - 1
names(msft.returns) <- month.abb[1:length(msft.returns)]
barplot(msft.returns,col="blue")
```



Barplot example

```
barplot(msft.returns,names.arg=month.name[1:length(msft.returns)],
  col="blue",las=2)
```



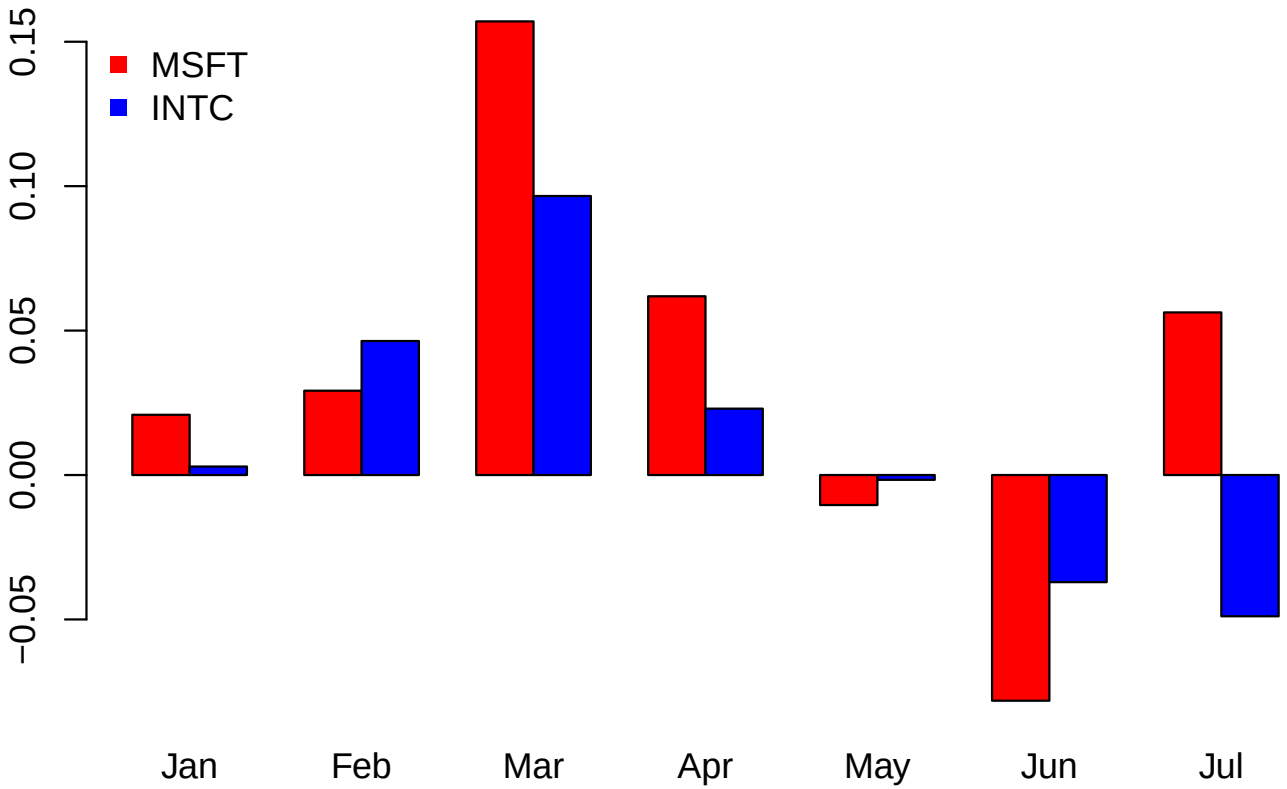
The legend function

```
args(legend)

## function (x, y = NULL, legend, fill = NULL, col = par("col"),
##     border = "black", lty, lwd, pch, angle = 45, density = NULL,
##     bty = "o", bg = par("bg"), box.lwd = par("lwd"), box.lty = par("lty"),
##     box.col = par("fg"), pt.bg = NA, cex = 1, pt.cex = cex, pt.lwd = lwd,
##     xjust = 0, yjust = 1, x.intersp = 1, y.intersp = 1, adj = c(0,
##         0.5), text.width = NULL, text.col = par("col"), text.font = NULL,
##     merge = do.lines && has.pch, trace = FALSE, plot = TRUE,
##     ncol = 1, horiz = FALSE, title = NULL, inset = 0, xpd, title.col = text.col,
##     title.adj = 0.5, seg.len = 2)
## NULL
```

`x/y`	location of the legend (can be give as a position name)
`legend`	vector of labels for the legend
`col`	vector of colors
`lty`	line type
`lwd`	line width
`pch`	character

Legend example

```
intc <- c(20.42,20.48,21.43,23.50,24.04,24.00,23.11,21.98)
intc.returns <- intc[-1] / intc[-length(intc)] - 1
barplot(rbind(msft.returns,intc.returns),beside=T,col=c(2,4))
legend(x="topleft",legend=c("MSFT","INTC"),pch=15,col=c(2,4),bty="n")
```



The matplot function

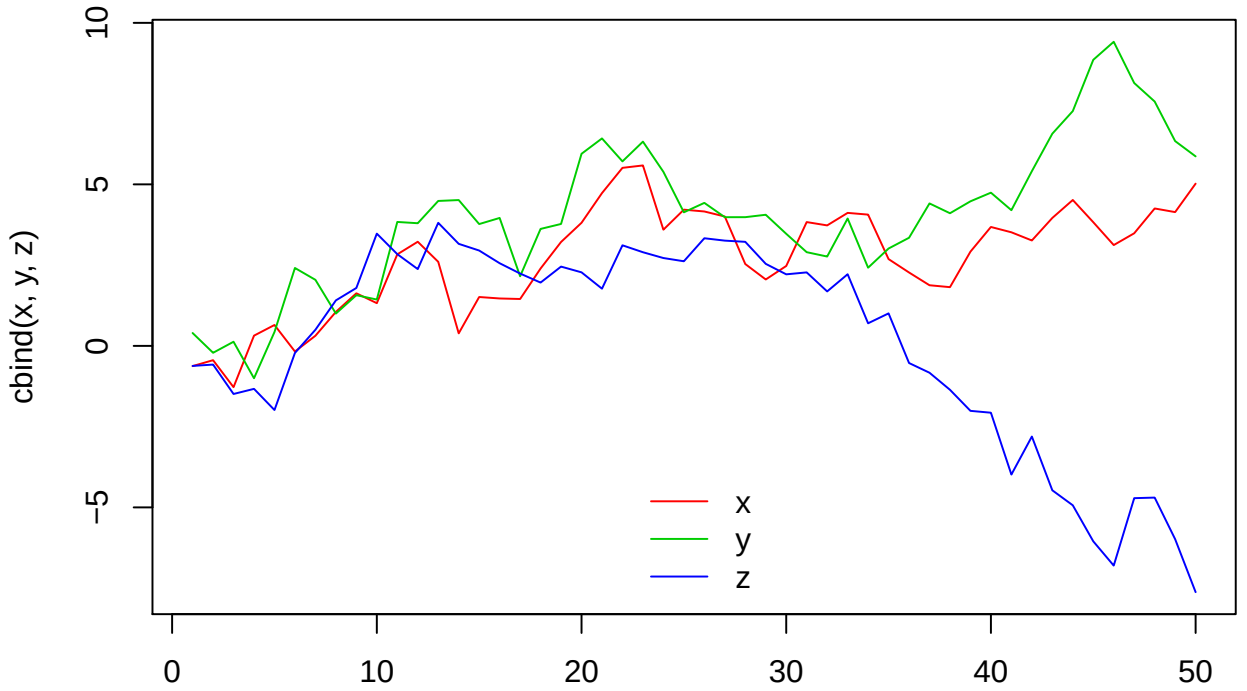
The matplot function plots multiple columns of a matrix versus an index

```
args(matplot)

## function (x, y, type = "p", lty = 1:5, lwd = 1, lend = par("lend"),
##     pch = NULL, col = 1:6, cex = NULL, bg = NA, xlab = NULL,
##     ylab = NULL, xlim = NULL, ylim = NULL, ..., add = FALSE,
##     verbose = getOption("verbose"))
## NULL
```

x/y matrices or vectors to be plotted

matplot example

```
set.seed(1)
x <- cumsum(rnorm(50))
y <- cumsum(rnorm(50))
z <- cumsum(rnorm(50))
matplot(cbind(x,y,z),col=2:4,type="l",lty=1)
legend("bottom",legend=c("x","y","z"),lty=1,col=2:4,bty="n")
```



Outline

Probability distributions

- Random variable

  A *random variable* is a quantity that can take on any of a set of possible values but only one of those values will actually occur
  - *discrete* random variables have a finite number of possible values
  - *continuous* random variables have an infinite number of possible values

- Probability distribution

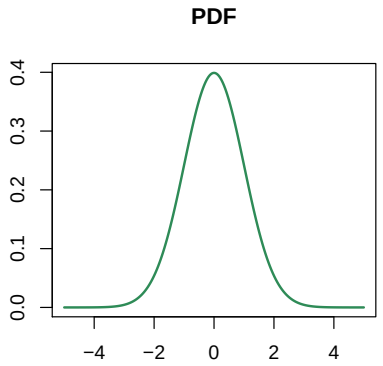
  The set of all possible values of a random variable along with their associated probabilities constitutes a *probability distribution* of the random variable

PDFs and CDFs

- Probability density function (PDF)

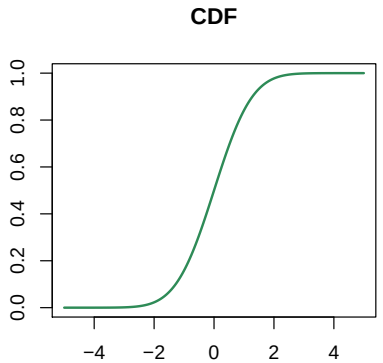
$$Pr(a < Y < b) = \int_a^b f_Y(y)$$

$$\int_{-\infty}^{\infty} f_Y(y)dy = 1$$

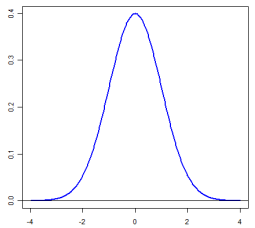
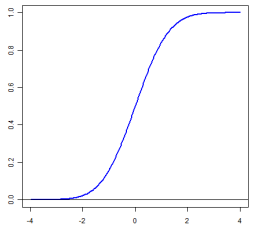
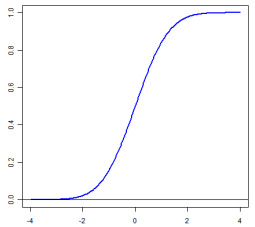


- Cumulative distribution function (CDF)

$$F_Y(y) = Pr(Y \leq y) = \int_{-\infty}^{y} f_Y(y)$$



PDF, CDF, quantile functions

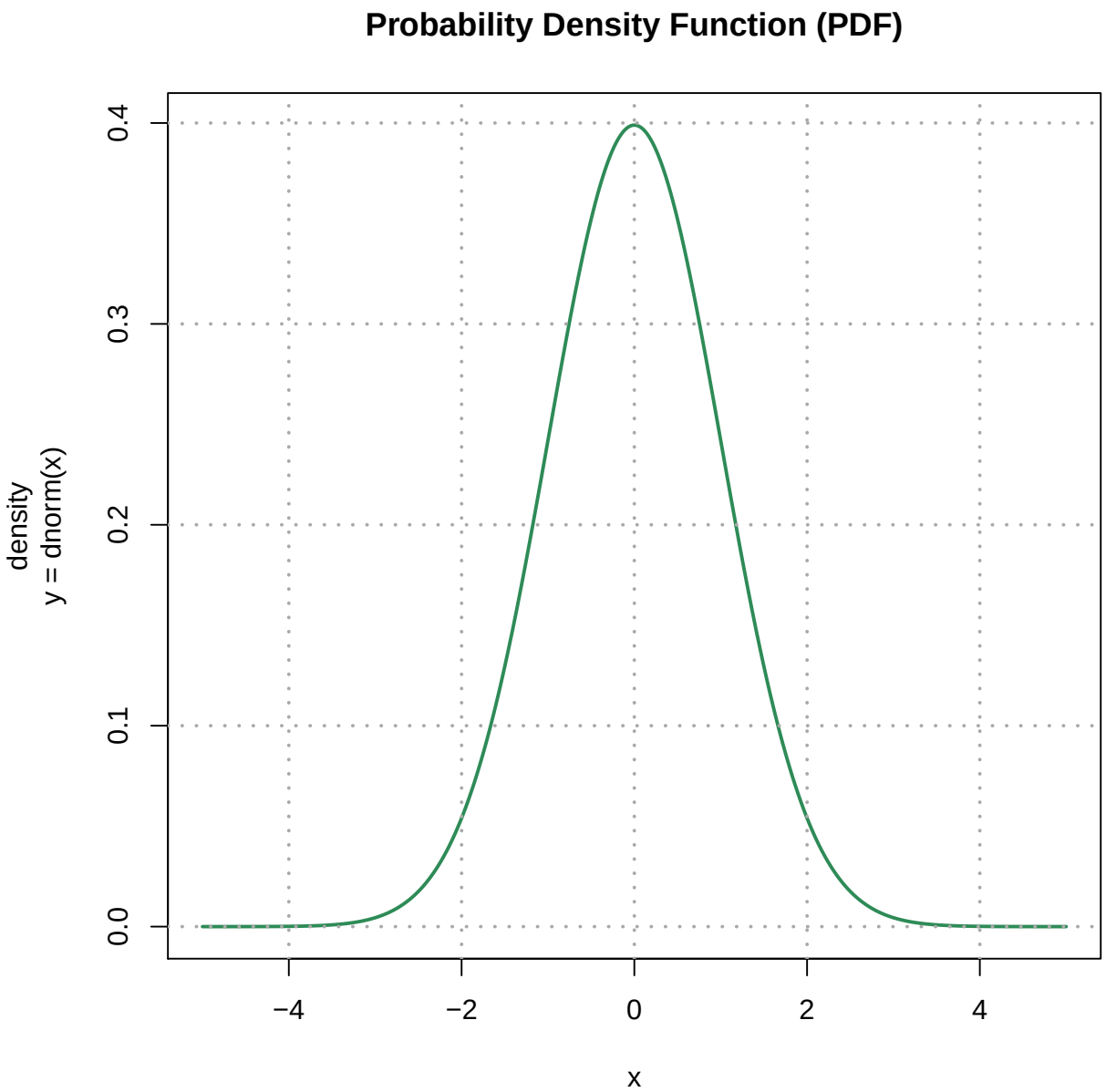
Function	General Notation	Normal Notation	R	Excel	Graph
pdf	$f(x)$	$\phi(z)$	dnorm	NORMDIST	
cdf	$F(x)$	$\Phi(z)$	pnorm	NORMDIST	
quantile	$F^{-1}(x)$	$\Phi^{-1}(z)$	qnorm	NORMINV	

Normal distribution PDF function: dnorm

dnorm computes the normal PDF: $\phi(z)$

```
args(dnorm)

## function (x, mean = 0, sd = 1, log = FALSE)
## NULL

x <- seq(from = -5, to = 5, by = 0.01)
x[1:10]

##  [1] -5.00 -4.99 -4.98 -4.97 -4.96 -4.95 -4.94 -4.93 -4.92 -4.91

y <- dnorm(x)
y[1:5]

## [1] 1.4867195e-06 1.5628671e-06 1.6427506e-06 1.7265445e-06 1.8144312e-06

par(mar = par()$mar + c(0,1,0,0))
plot(x=x,y=y,type="l",col="seagreen",lwd=2,
  xlab="x",ylab="density\ny = dnorm(x)")
grid(col="darkgrey",lwd=2)
title(main="Probability Density Function (PDF)")
```

Normal distribution PDF function: dnorm



Others:

dt

dstd

dsstd

dged

dsged

dst

dmst

dct

Normal distribution CDF functions: pnorm and qnorm

pnorm computes the normal CDF:

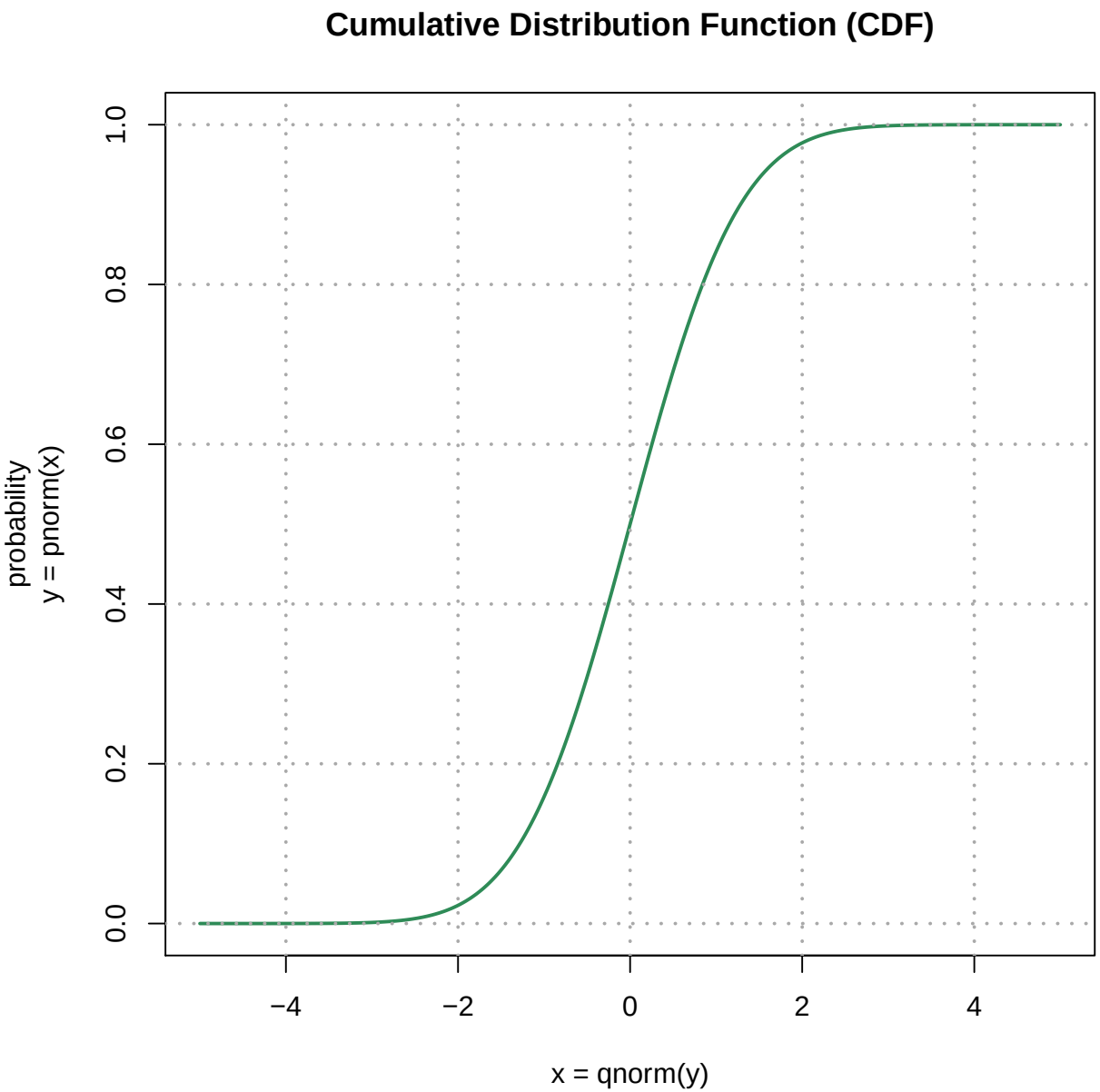
$$\Pr(X \leq z) = \Phi(z)$$

qnorm computes the inverse of the normal CDF (i.e. quantile):

$$z_\alpha = \Phi^{-1}(\alpha)$$

```
args(pnorm)

## function (q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
## NULL

args(qnorm)

## function (p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
## NULL

y <- pnorm(x)
par(mar = par()$mar + c(0,1,0,0))
plot(x=x,y=y,type="l",col="seagreen",lwd=2, xlab="x = qnorm(y)",
   ylab="probability\ny = pnorm(x)")  ; grid(col="darkgrey",lwd=2)
title(main="Cumulative Distribution Function (CDF)")
```

Normal distribution CDF functions: pnorm and qnorm



Others:

pt

pstd

psstd

pged

psged

pst

pmst

pct

Generating normally distributed random numbers

The function `rnorm` generates random numbers from a normal distribution

```
args(rnorm)

## function (n, mean = 0, sd = 1)
## NULL

x <- rnorm(150)
x[1:5]

## [1] -0.62645381  0.18364332 -0.83562861  1.59528080  0.32950777

y <- rnorm(50,sd=3)
y[1:5]

## [1]  1.350561304 -0.055679498 -0.954205124 -2.788086442 -4.462380930
```

`n`	number of observations
`mean`	mean of distribution
`sd`	standard deviation of distribution

Histograms

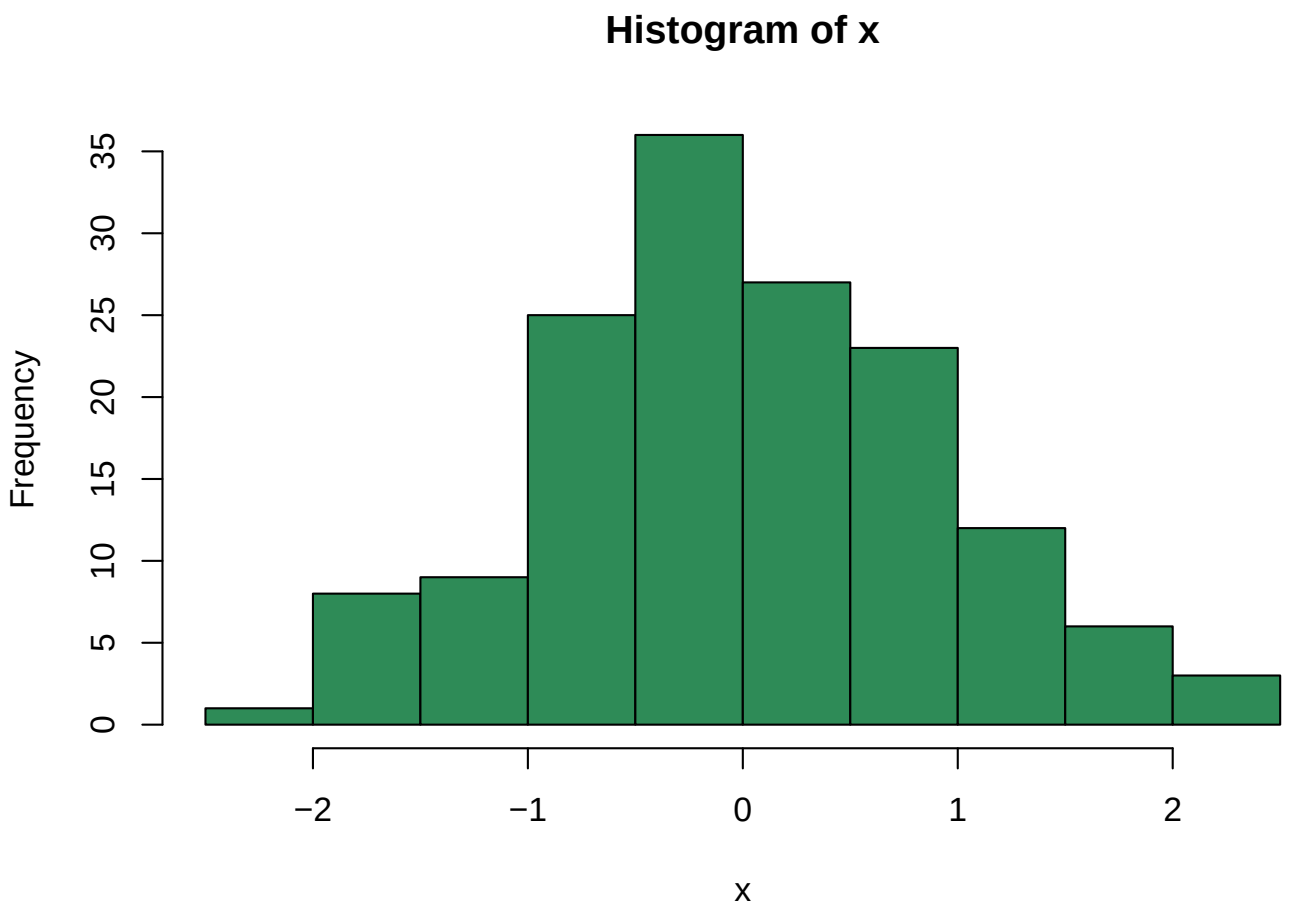
The generic function `hist` computes a histogram of the given data values

```
args(hist.default)

## function (x, breaks = "Sturges", freq = NULL, probability = !freq,
##     include.lowest = TRUE, right = TRUE, density = NULL, angle = 45,
##     col = NULL, border = NULL, main = paste("Histogram of", xname),
##     xlim = range(breaks), ylim = NULL, xlab = xname, ylab, axes = TRUE,
##     plot = TRUE, labels = FALSE, nclass = NULL, warn.unused = TRUE,
##     ...)
## NULL
```

`x`	vector of histogram data
`breaks`	number of breaks, vector of breaks, name of break algorithm, break function
`prob`	probability densities or counts
`ylim`	y-axis range
`col`	color or bars

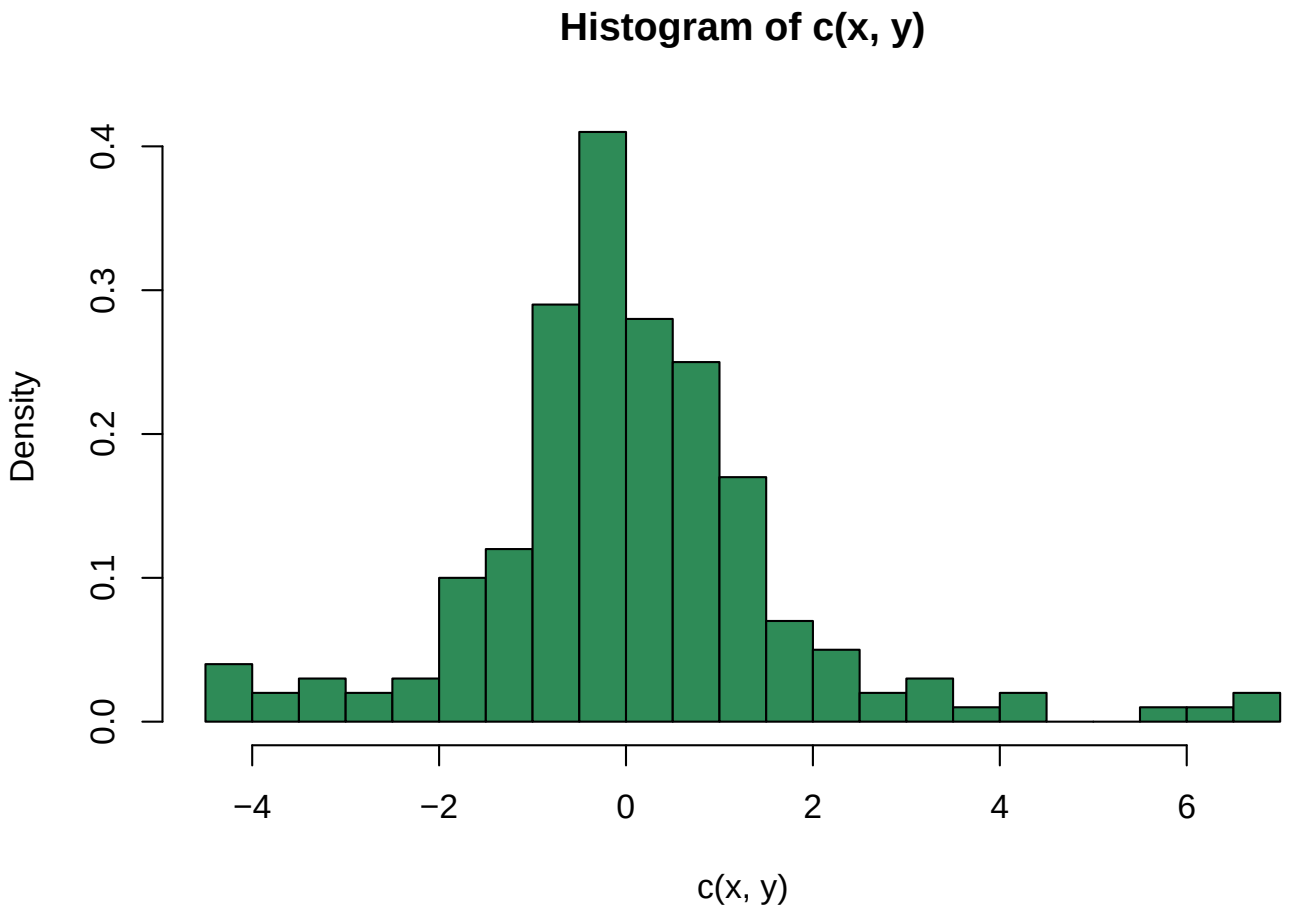
Plotting histograms

```
hist(x,col="seagreen")
```



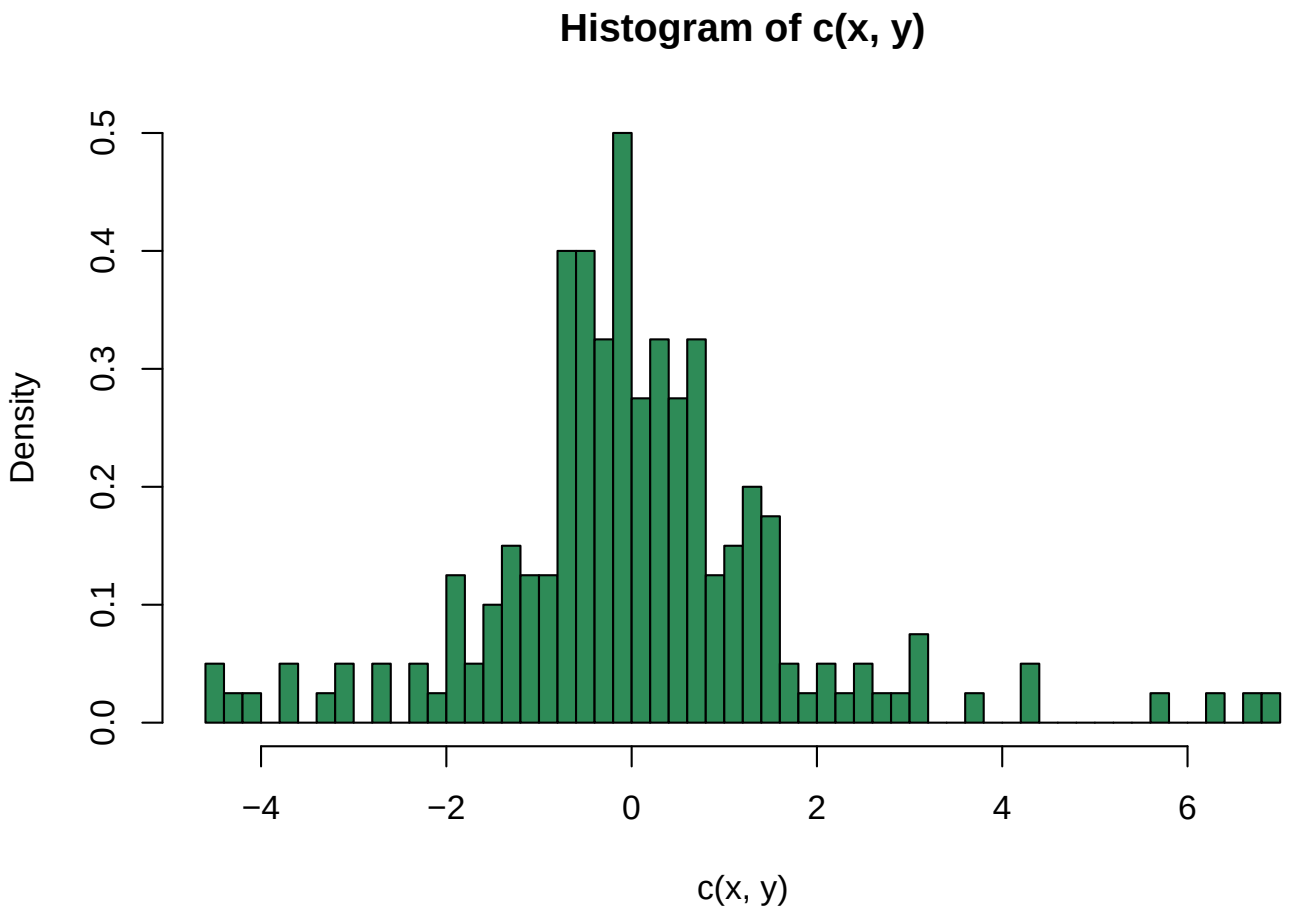
Plotting histograms

```
hist(c(x,y),prob=T,breaks="FD",col="seagreen")
```



Plotting histograms

```
hist(c(x,y),prob=T,breaks=50,col="seagreen")
```

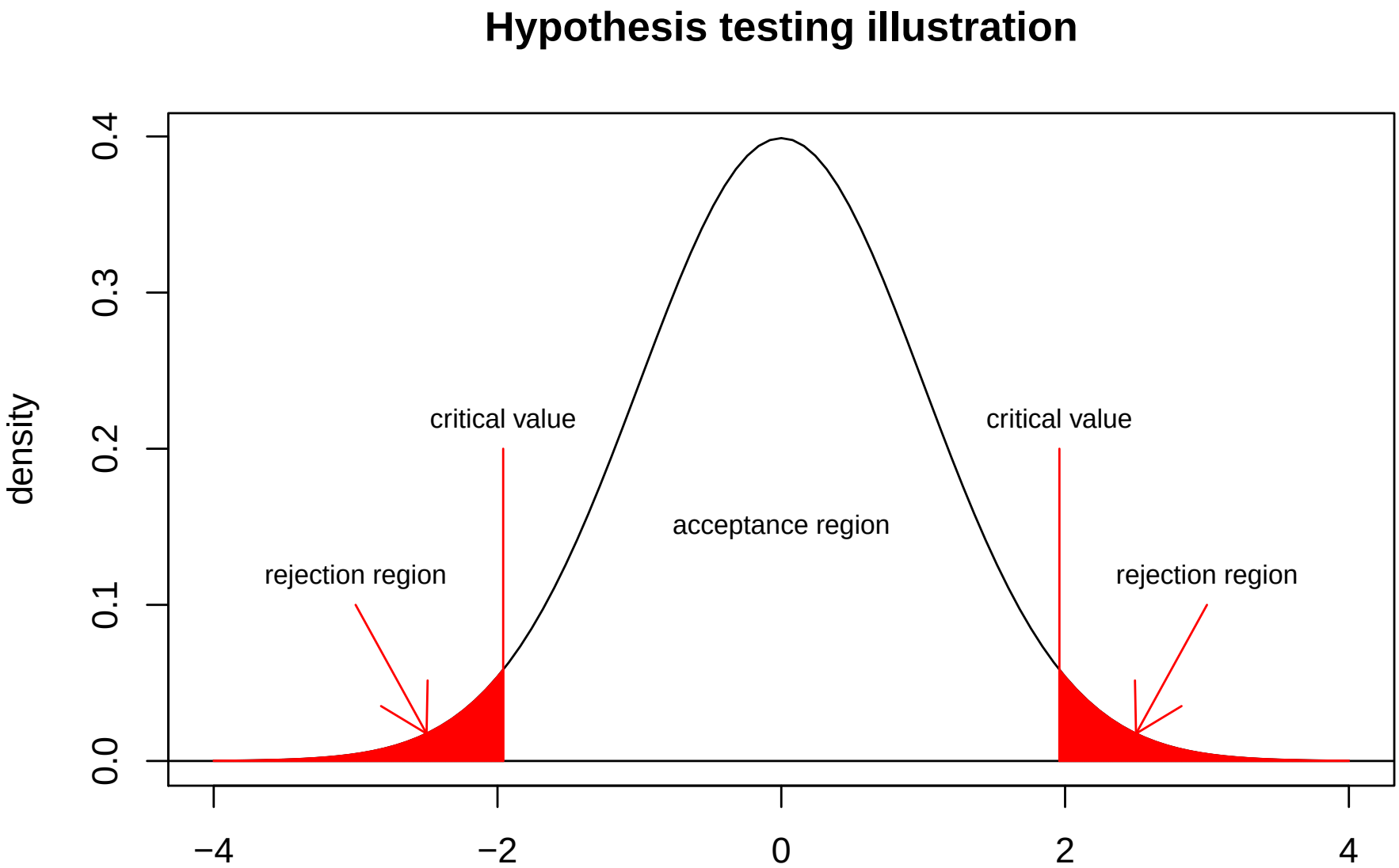


Basic stats functions

Short list of some common statistics and math functions:

`mean`	mean of a vector or matrix
`median`	median of a vector or matrix
`mad`	median absolute deviation of a vector or matrix
`var`	variance of a vector or matrix
`sd`	standard deviation of a vector
`cov`	covariance between vectors
`cor`	correlation between vectors
`diff`	difference between elements in a vector
`log`	log of a vector or matrix
`exp`	exponentiation of a vector or matrix
`abs`	absolute value of a vector or matrix

Plot with curve, segments, arrows, text



The curve function

The curve function draws a curve of a function or expression over a range

```
args(curve)

## function (expr, from = NULL, to = NULL, n = 101, add = FALSE,
##     type = "l", xname = "x", xlab = xname, ylab = NULL, log = NULL,
##     xlim = NULL, ...)
## NULL
```

expr	function or expression of x
from	start of range
to	end of range
n	number of points over from/to range
add	add to current plot (T/F)

The abline function

The abline function adds one or more straight lines through the current plot

```
args(abline)

## function (a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL,
##     coef = NULL, untf = FALSE, ...)
## NULL
```

- h/v vertical or horizontal coordinate of line
- a/b intercept and slope of line

The segments function

The segments function draws line segments between point pairs

```
args(segments)

## function (x0, y0, x1 = x0, y1 = y0, col = par("fg"), lty = par("lty"),
##     lwd = par("lwd"), ...)
## NULL
```

- `x0, y0` point coordinates from which to draw
- `x1, y1` point coordinates to which to draw

The arrows function

The arrows function draws line segments between point pairs

```
args(arrows)

## function (x0, y0, x1 = x0, y1 = y0, length = 0.25, angle = 30,
##     code = 2, col = par("fg"), lty = par("lty"), lwd = par("lwd"),
##     ...)
## NULL
```

x0, y0	point coordinates from which to draw
x1, y1	point coordinates to which to draw
length	length of the edges of the arrow head (in inches)
angle	angle from the shaft of the arrow to the edge of the arrow head

Plot with curve, segments, arrows, text

```
curve(dnorm,-4,4,main="Hypothesis testing illustration",xlab="",ylab="density")
abline(h=0)
x <- seq(-4,qnorm(0.025),len=500)
y <- dnorm(x)
for(i in 1:length(x)) {
  segments(x[i],0,x[i],y[i],col="red")
}
x <- seq(qnorm(0.975),4,len=500)
y <- dnorm(x)
for(i in 1:length(x)) {
  segments(x[i],0,x[i],y[i],col="red")
}
text(0,0.15,"acceptance region",cex=0.75)
t.stat = 2.3256
arrows(-3,0.1,-2.5,dnorm(-2.5),col="red")
text(-3,0.1,"rejection region",pos=3,cex=0.75)
arrows(3,0.1,2.5,dnorm(2.5),col="red")
text(3,0.1,"rejection region",pos=3,cex=0.75)
segments(qnorm(0.975),0,qnorm(0.975),0.2,col="red")
text(qnorm(0.975),0.2,"critical value",pos=3,cex=0.75)
segments(qnorm(0.025),0,qnorm(0.025),0.2,col="red")
text(qnorm(0.025),0.2,"critical value",pos=3,cex=0.75)
```

Outline

Time series data

Most financial data can be characterized as a time series:

- market data
- economic data
- corporate earnings data
- trading activity data

Time series data

Time series

A *time series* is a sequence of *ordered* data points measured at specific points in time

Time series object

A time series object in R is a *compound data structure* that includes a data matrix as well as a vector of associated time stamps

class	**package**	**overview**
`ts`	stats	regularly spaced time series
`mts`	stats	multiple regularly spaced time series
`zoo`	zoo	reg/irreg and arbitrary time stamp classes
`xts`	xts	an extension of the zoo class

Time series methods

Time series classes in R will typically implement the following methods:

`start`	return start of time series
`end`	return end of time series
`frequency`	return frequency of time series
`window`	Extract subset of time series
`index`	return time index of time series
`time`	return time index of time series
`coredata`	return data of time series
`diff`	difference of the time series
`lag`	lag of the time series
`aggregate`	aggregate to lower resolution time series
`cbind`	merge 2 or more time series together
`merge`	merge 2 or more time series together

The zoo package

The zoo package provides an infrastructure for regularly-spaced and irregularly-space time series

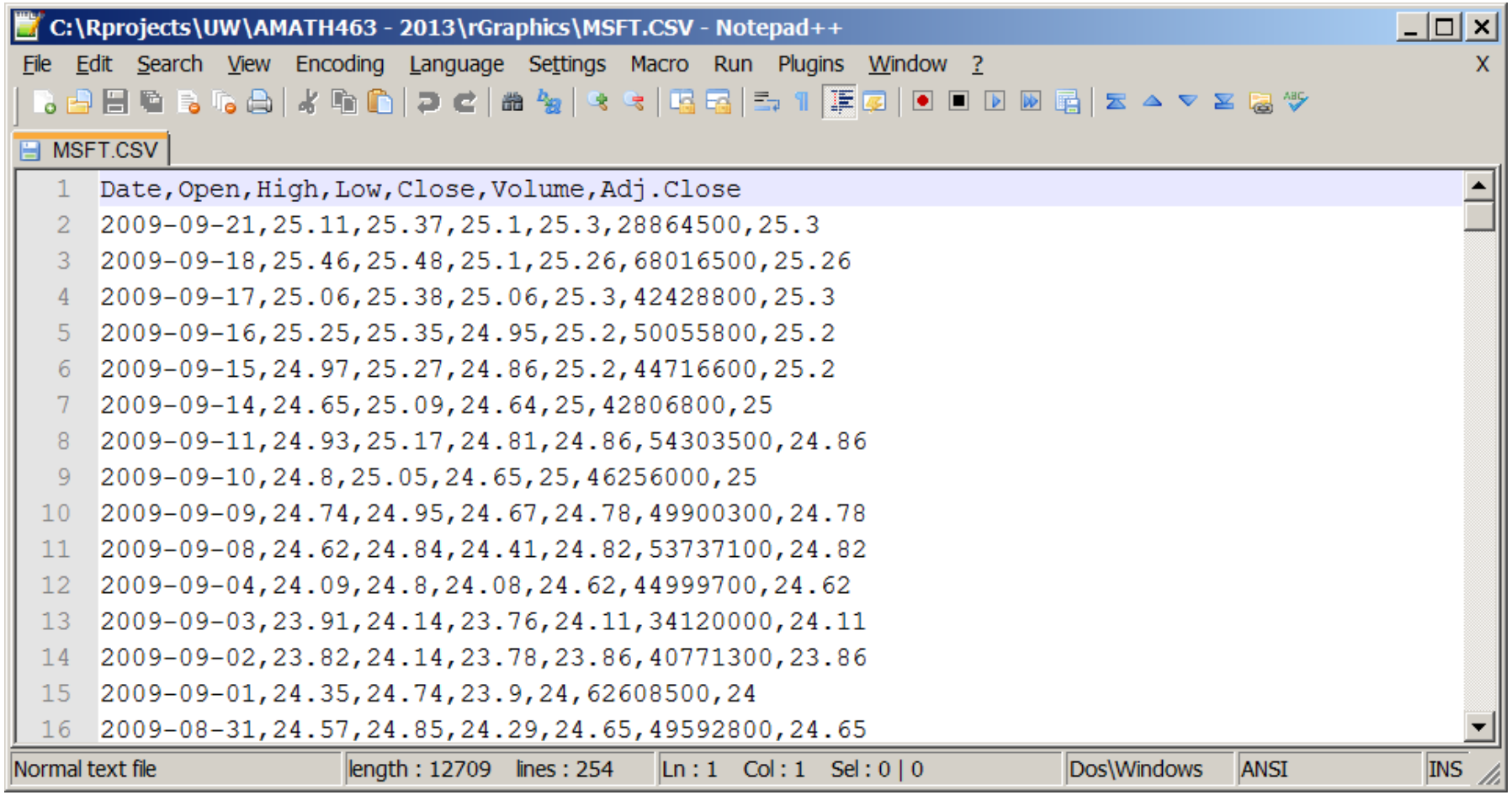
Key functions:

zoo	create a zoo time series object
merge	merges time series (automatically handles of time alignment)
aggregate	create coarser resolution time series with summary statistics
rollapply	calculate rolling window statistics
read.zoo	read a text file into a zoo time series object

Authors:

- Achim Zeileis
- Gabor Grothendieck

Creating a zoo object



Creating a zoo object

```
library(zoo)
msft.df <- read.table("MSFT.CSV", header = TRUE, sep = ",", as.is = TRUE)
head(msft.df,2)

##         Date  Open  High  Low Close   Volume Adj.Close
## 1 2009-09-21 25.11 25.37 25.1 25.30 28864500     25.30
## 2 2009-09-18 25.46 25.48 25.1 25.26 68016500     25.26

args(zoo)

## function (x = NULL, order.by = index(x), frequency = NULL)
## NULL

msft.z <- zoo(x=msft.df[,"Close"],order.by=as.Date(msft.df[,"Date"]))
head(msft.z)

## 2008-09-22 2008-09-23 2008-09-24 2008-09-25 2008-09-26 2008-09-29
##      25.40      25.44      25.72      26.61      27.40      25.01
```

Inspecting a zoo object

```
class(msft.z)
```

```
## [1] "zoo"
```

```
start(msft.z)
```

```
## [1] "2008-09-22"
```

```
end(msft.z)
```

```
## [1] "2009-09-21"
```

```
frequency(msft.z)
```

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

```
class(coredata(msft.z))
```

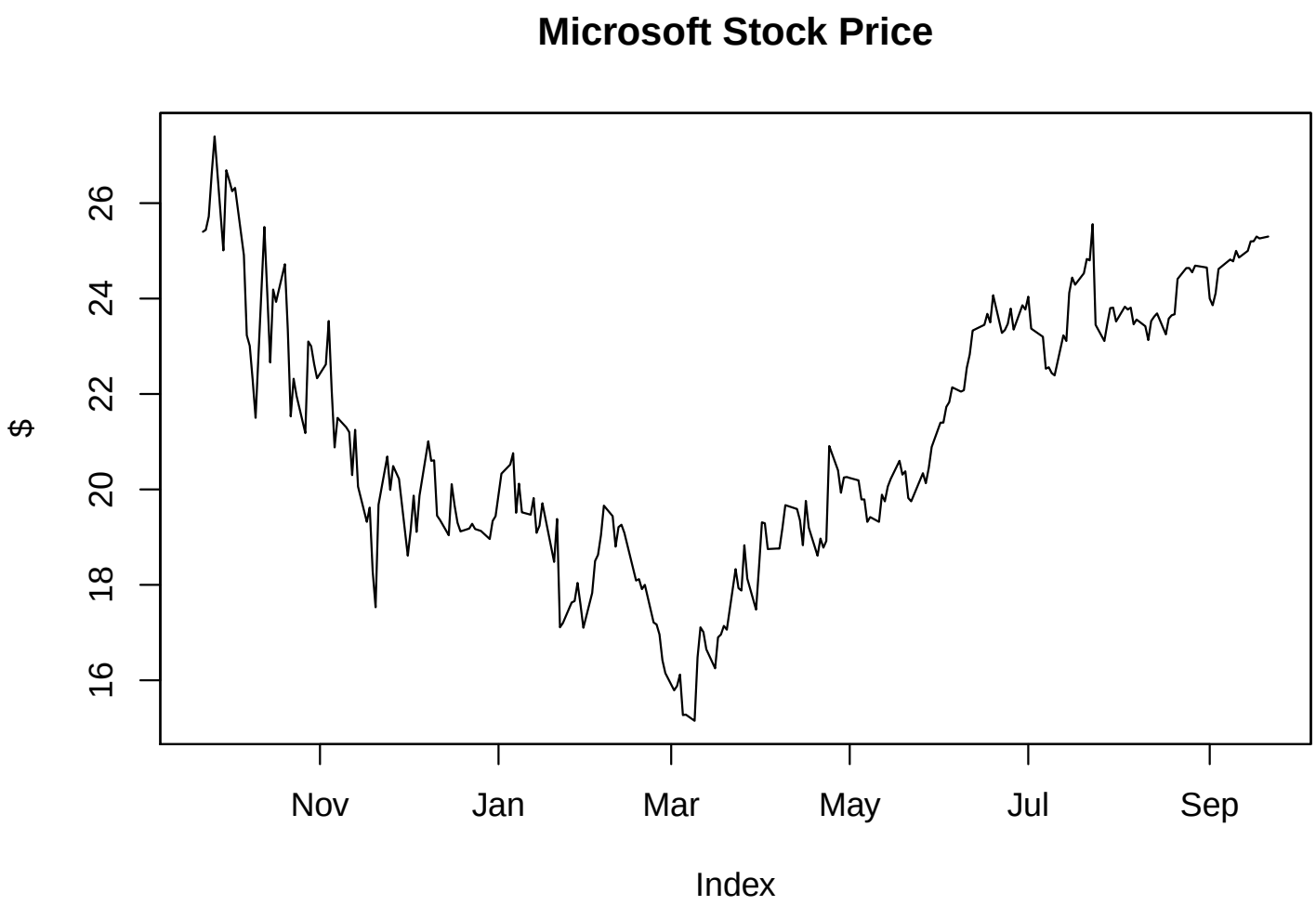
```
## [1] "numeric"
```

```
class(time(msft.z))
```

```
## [1] "Date"
```

Plotting a zoo object

```
plot(msft.z,ylab="$",main="Microsoft Stock Price")
```



Date class

A `Date` object is stored internally as the number of days since 1970-01-01

```
myStr <- "2013-07-04"
class(myStr)
```

```
## [1] "character"
```

```
args(getS3method("as.Date","character"))
```

```
## function (x, format = "", ...)
## NULL
```

```
myDate <- as.Date(myStr)
myDate
```

```
## [1] "2013-07-04"
```

```
class(myDate)
```

```
## [1] "Date"
```

```
as.numeric(myDate)
```

```
## [1] 15890
```

Date format string for as.Date and format.Date

%Y Year with century
%y Year without century (00-99)
%m Month as decimal number (01-12)
%d Day of the month as decimal number (01-31)

```
format(myDate,"%m/%d/%Y")

## [1] "07/04/2013"

format(myDate,"%m/%d/%y")

## [1] "07/04/13"

format(myDate,"%Y%m%d")

## [1] "20130704"
```

- For comprehensive list of date/time conversion specifications, see help for `strptime` function

Multi-column time series object

```
data(Garch)
head(Garch,3)
```

```
##     date       day     dm            ddm     bp     cd       dy     sf
## 1 800102 wednesday 0.5861             NA 2.2490 0.8547 0.004206 0.6365
## 2 800103  thursday 0.5837 -0.00410327127 2.2365 0.8552 0.004187 0.6357
## 3 800104    friday 0.5842  0.00085623774 2.2410 0.8566 0.004269 0.6355
```

```
garch.z <- zoo(x=Garch[,-(1:2)],
          order.by=as.Date(x=as.character(Garch[,"date"]),format="%y%m%d"))
head(garch.z,3)
```

```
##                dm            ddm     bp     cd       dy     sf
## 1980-01-02 0.5861             NA 2.2490 0.8547 0.004206 0.6365
## 1980-01-03 0.5837 -0.00410327127 2.2365 0.8552 0.004187 0.6357
## 1980-01-04 0.5842  0.00085623774 2.2410 0.8566 0.004269 0.6355
```

```
class(coredata(garch.z))
```

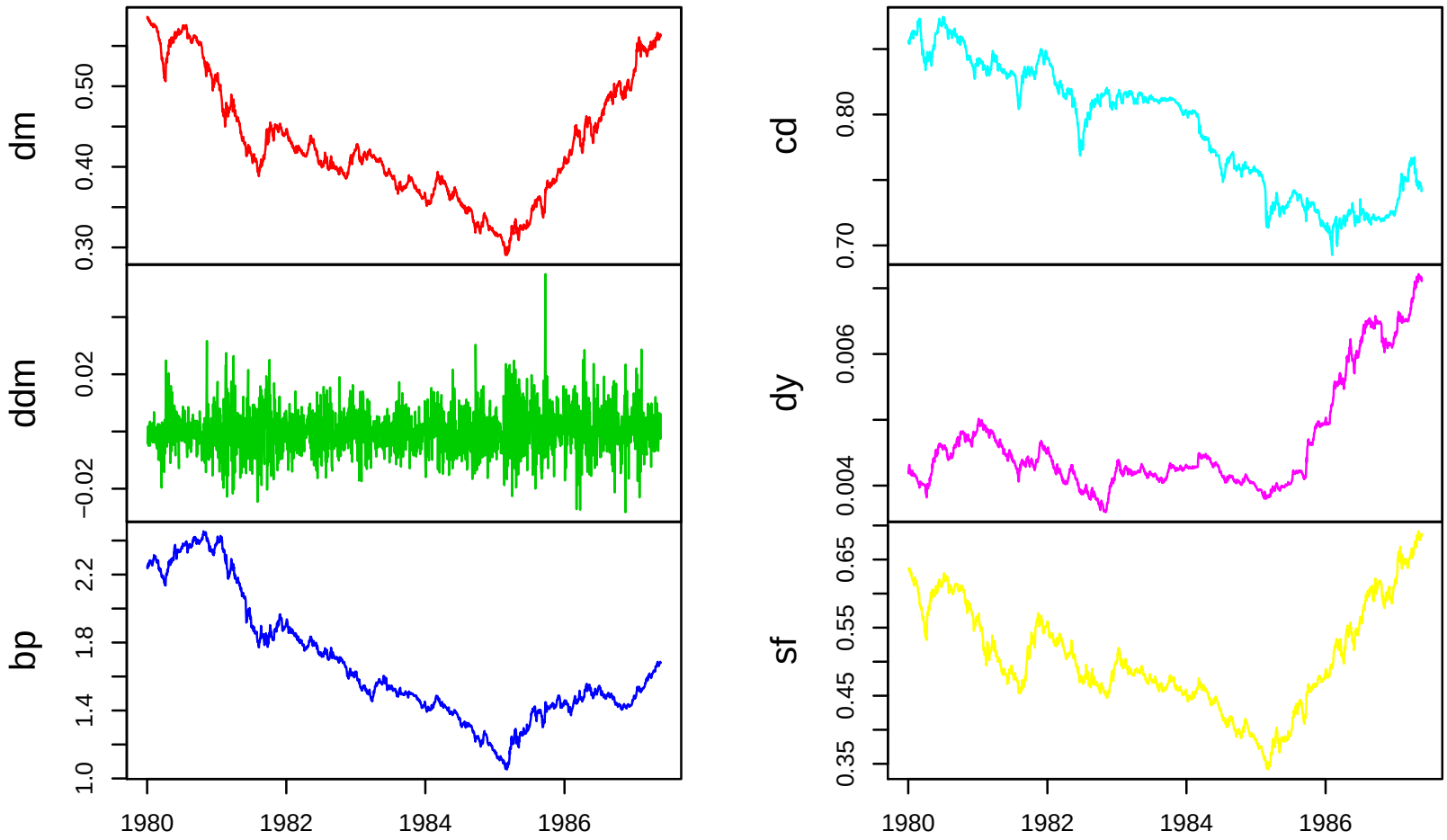
```
## [1] "matrix"
```

```
class(index(garch.z))
```

```
## [1] "Date"
```

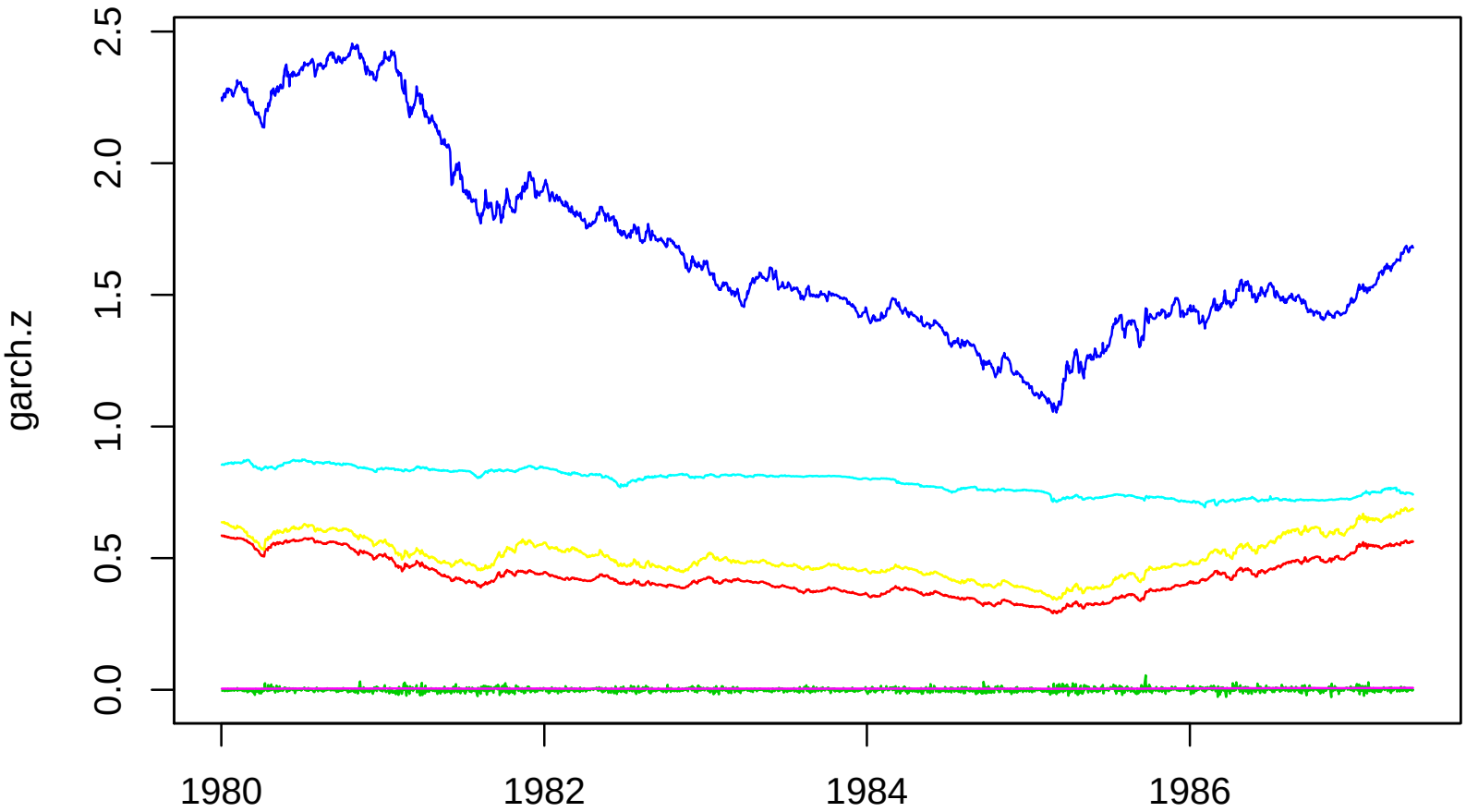
The plot.zoo function

```
plot(garch.z,col=2:7,main="",xlab="")
```



The plot.zoo function

```
plot(garch.z,plot.type="single",col=2:7,main="",xlab="")
```



Zoo object helper function

```
zinfo <- function (zobj)
{
    print(start(zobj))
    print(end(zobj))
    d <- dim(zobj)
    if( is.null(d) ) {
      nr <- length(zobj)
      nc <- 1
    } else {
      nr <- d[1]
      nc <- d[2]
    }
    print(paste(nr, "x", nc))
    print(paste(sum(is.na(zobj)), " NAs", sep = ""))
    x = as.numeric(end(zobj) - start(zobj))
    y = x/365
    opy = nr/y
    print(paste(round(y,1),"years"))
    print(paste(round(opy,1),"obs/years"))
}
```

Zoo object helper function

```
zinfo(msft.z)

## [1] "2008-09-22"
## [1] "2009-09-21"
## [1] "252 x 1"
## [1] "0 NAs"
## [1] "1 years"
## [1] "252.7 obs/years"

zinfo(garch.z)

## [1] "1980-01-02"
## [1] "1987-05-21"
## [1] "1867 x 6"
## [1] "1 NAs"
## [1] "7.4 years"
## [1] "252.8 obs/years"
```

The read.zoo function

```
args(read.zoo)
```

```
## function (file, format = "", tz = "", FUN = NULL, regular = FALSE,
##     index.column = 1, drop = TRUE, FUN2 = NULL, split = NULL,
##     aggregate = FALSE, ..., text)
## NULL
```

```
soft <- read.zoo(file="MSFT.CSV",header=TRUE,sep=",")
head(soft,2)
```

```
##             Open  High   Low Close    Volume Adj.Close
## 2008-09-22 26.22 26.32 25.32 25.40 105207700     24.76
## 2008-09-23 25.66 26.17 25.34 25.44  92181300     24.80
```

```
class(soft)
```

```
## [1] "zoo"
```

```
class(coredata(soft))
```

```
## [1] "matrix"
```

```
class(index(soft))
```

```
## [1] "Date"
```

The zooreg object

```
head(Capm,3)
```

```
##   rfood  rdur  rcon  rmrf   rf
## 1 -4.59  0.87 -6.84 -6.99 0.33
## 2  2.62  3.46  2.78  0.99 0.29
## 3 -1.67 -2.28 -0.48 -1.46 0.35
```

```
args(zooreg)
```

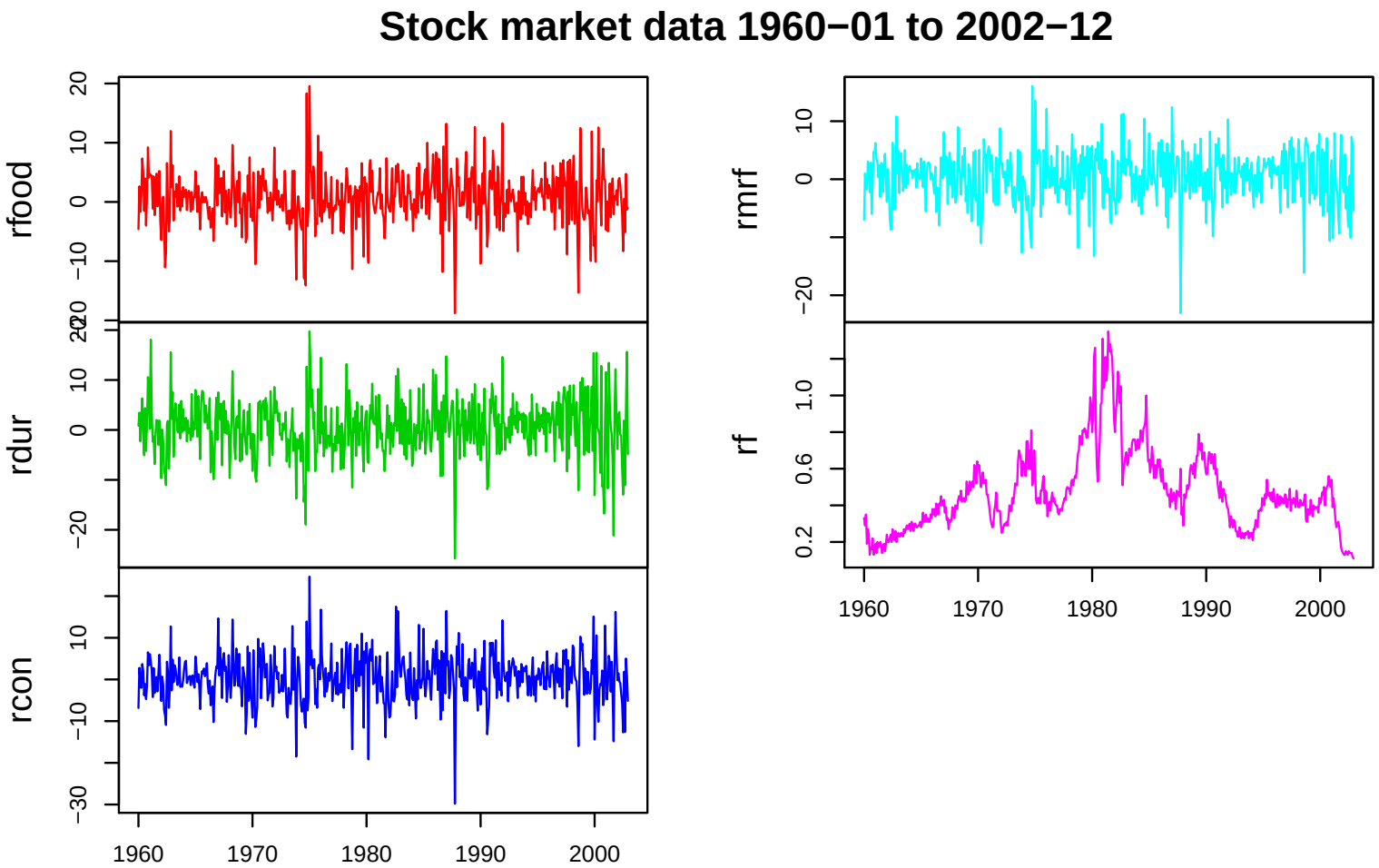
```
## function (data, start = 1, end = numeric(), frequency = 1, deltat = 1,
##     ts.eps = getOption("ts.eps"), order.by = NULL)
## NULL
```

```
capm.z <- zooreg(Capm, frequency = 12, start = c(1960, 1),end=c(2002,12))
head(capm.z,4)
```

```
##         rfood  rdur  rcon  rmrf   rf
## 1960(1) -4.59  0.87 -6.84 -6.99 0.33
## 1960(2)  2.62  3.46  2.78  0.99 0.29
## 1960(3) -1.67 -2.28 -0.48 -1.46 0.35
## 1960(4)  0.86  2.41 -2.02 -1.70 0.19
```

```
plot(capm.z,main="",xlab="",col=2:6)
title("Stock market data 1960-01 to 2002-12")
```

Capm market data



The xts package

The `xts` package extends the zoo time series class with fine-grained time indexes, interoperability with other R time series classes, and user defined attributes

Key functions:

xts	create an xts time series object
align.time	align time series to a coarser resolution
to.period	convert time series data to an OHLC series
[.xts	subset time series

Authors:

- Jeffrey Ryan
- Josh Ulrich

Creating a xts object

```
library(xts)
args(xts)

## function (x = NULL, order.by = index(x), frequency = NULL, unique = TRUE,
##     tzone = Sys.getenv("TZ"), ...)
## NULL

msft.x <- xts(x=msft.df[,"Close"],order.by=as.Date(msft.df[,"Date"]))
head(msft.x)

##             [,1]
## 2008-09-22 25.40
## 2008-09-23 25.44
## 2008-09-24 25.72
## 2008-09-25 26.61
## 2008-09-26 27.40
## 2008-09-29 25.01
```

Inspecting a xts object

```
class(msft.x)
```

```
## [1] "xts" "zoo"
```

```
start(msft.x)
```

```
## [1] "2008-09-22"
```

```
end(msft.x)
```

```
## [1] "2009-09-21"
```

```
frequency(msft.x)
```

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

```
class(coredata(msft.x))
```

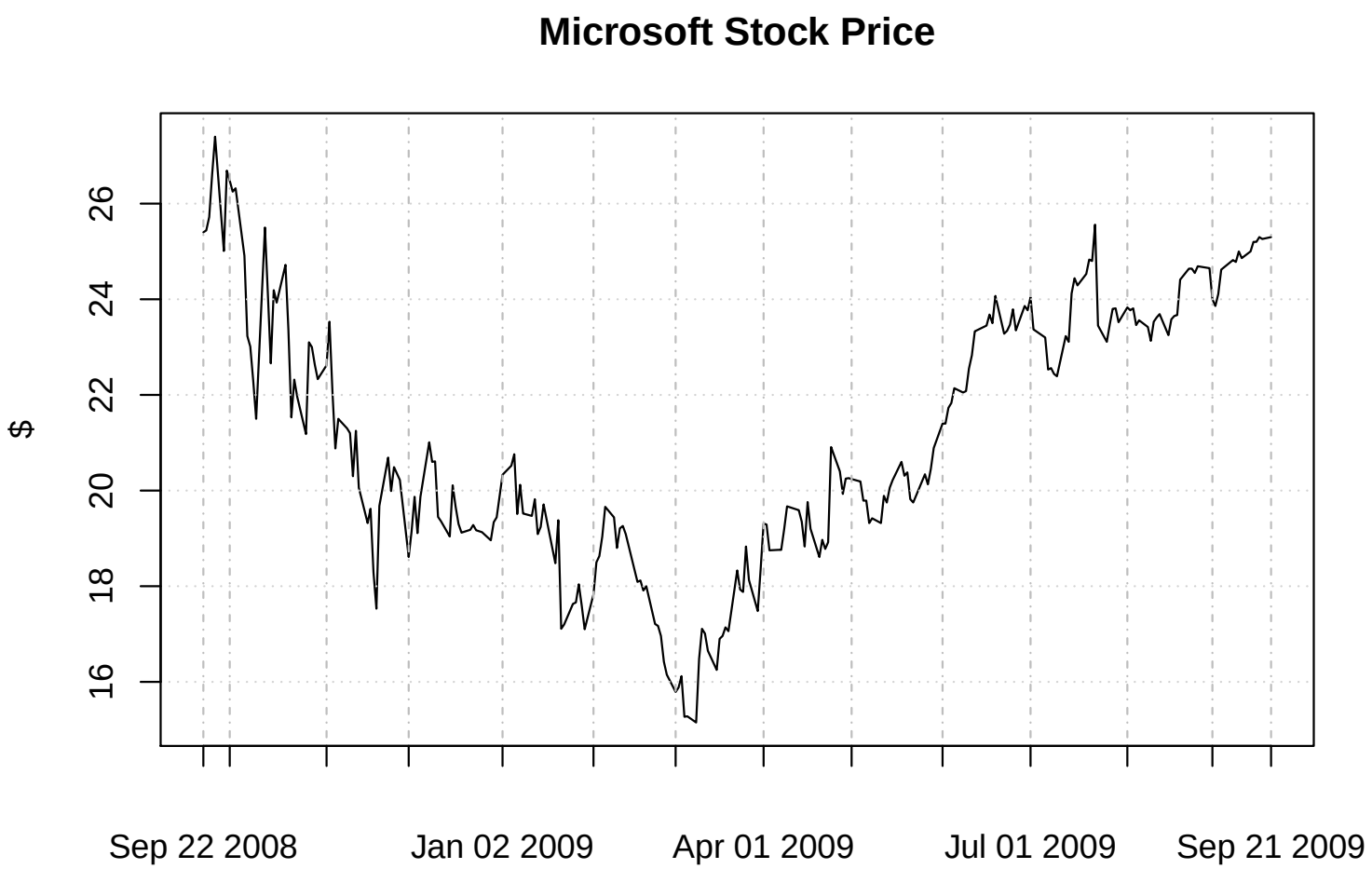
```
## [1] "matrix"
```

```
class(time(msft.x))
```

```
## [1] "Date"
```

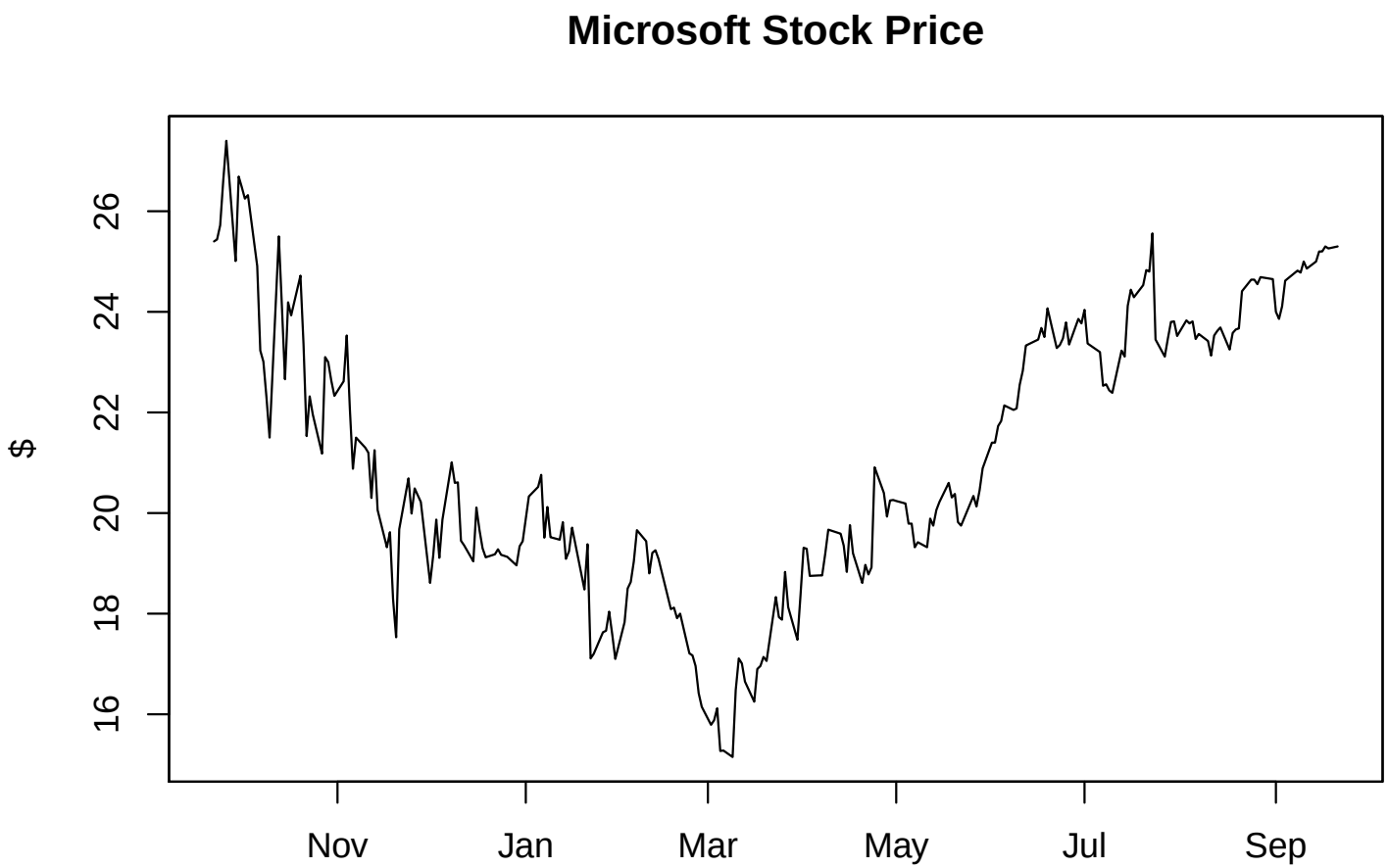
The plot.xts function

```
plot(msft.x,ylab="$",main="Microsoft Stock Price",minor.ticks=FALSE)
```



plot.zoo with an xts object

```
plot.zoo(msft.x,ylab="$",main="Microsoft Stock Price",xlab="")
```



ts objects

```
data(Tbrate)
class(Tbrate)
```

```
## [1] "mts" "ts"
```

```
window(Tbrate,start=start(Tbrate),end=c(1950,4))
```

```
##              r         y   pi
## 1950 Q1 0.510 11.538320 0.50
## 1950 Q2 0.510 11.538320 4.06
## 1950 Q3 0.550 11.561030 5.50
## 1950 Q4 0.623 11.601046 9.85
```

```
tsp(Tbrate)
```

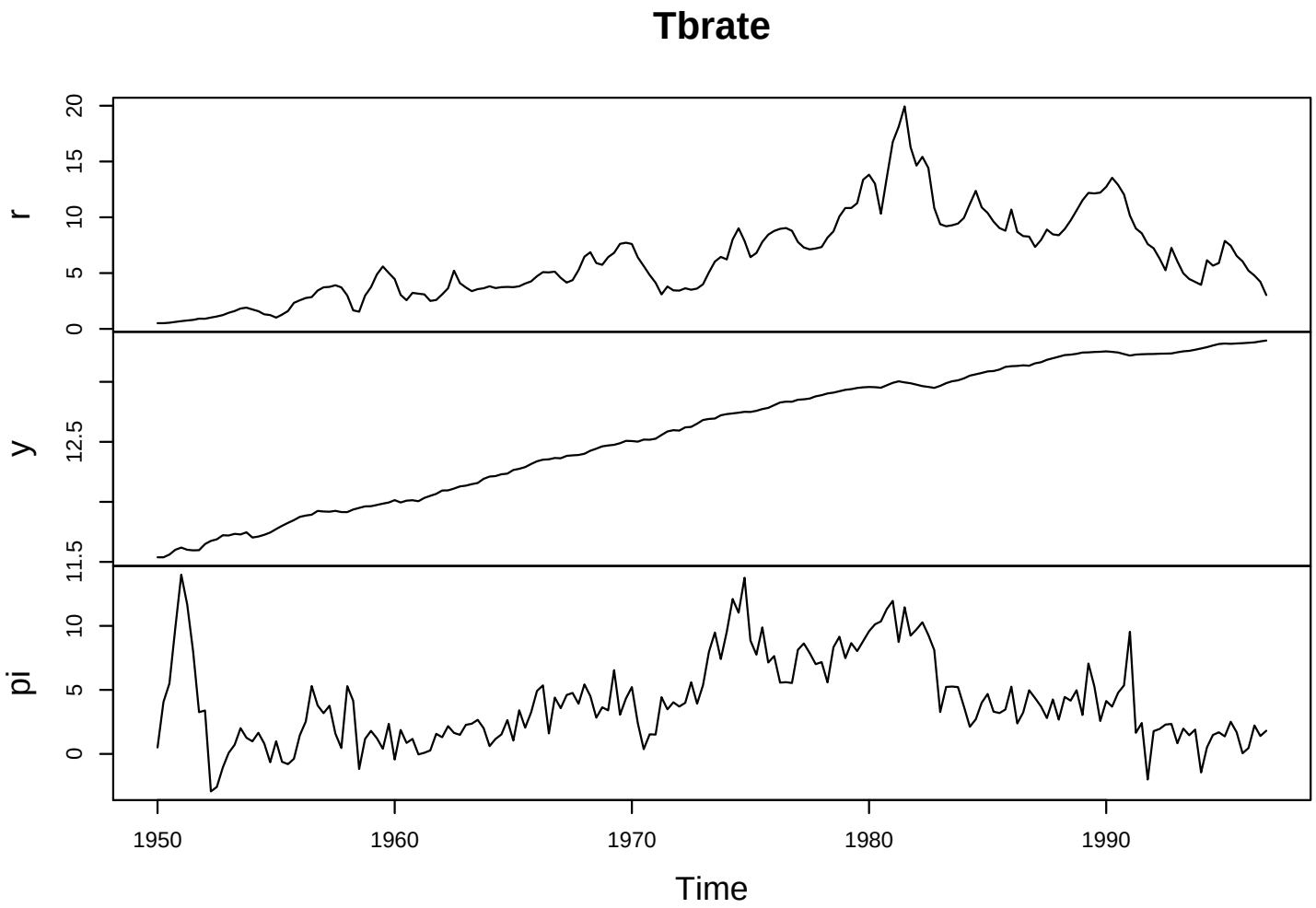
```
## [1] 1950.00 1996.75    4.00
```

```
args(plot.ts)
```

```
## function (x, y = NULL, plot.type = c("multiple", "single"), xy.labels,
##     xy.lines, panel = lines, nc, yax.flip = FALSE, mar.multi = c(0,
##         5.1, 0, if (yax.flip) 5.1 else 2.1), oma.multi = c(6,
##         0, 5, 0), axes = TRUE, ...)
## NULL
```

The plot.ts function

```
plot(Tbrate)
```



The ts.plot function

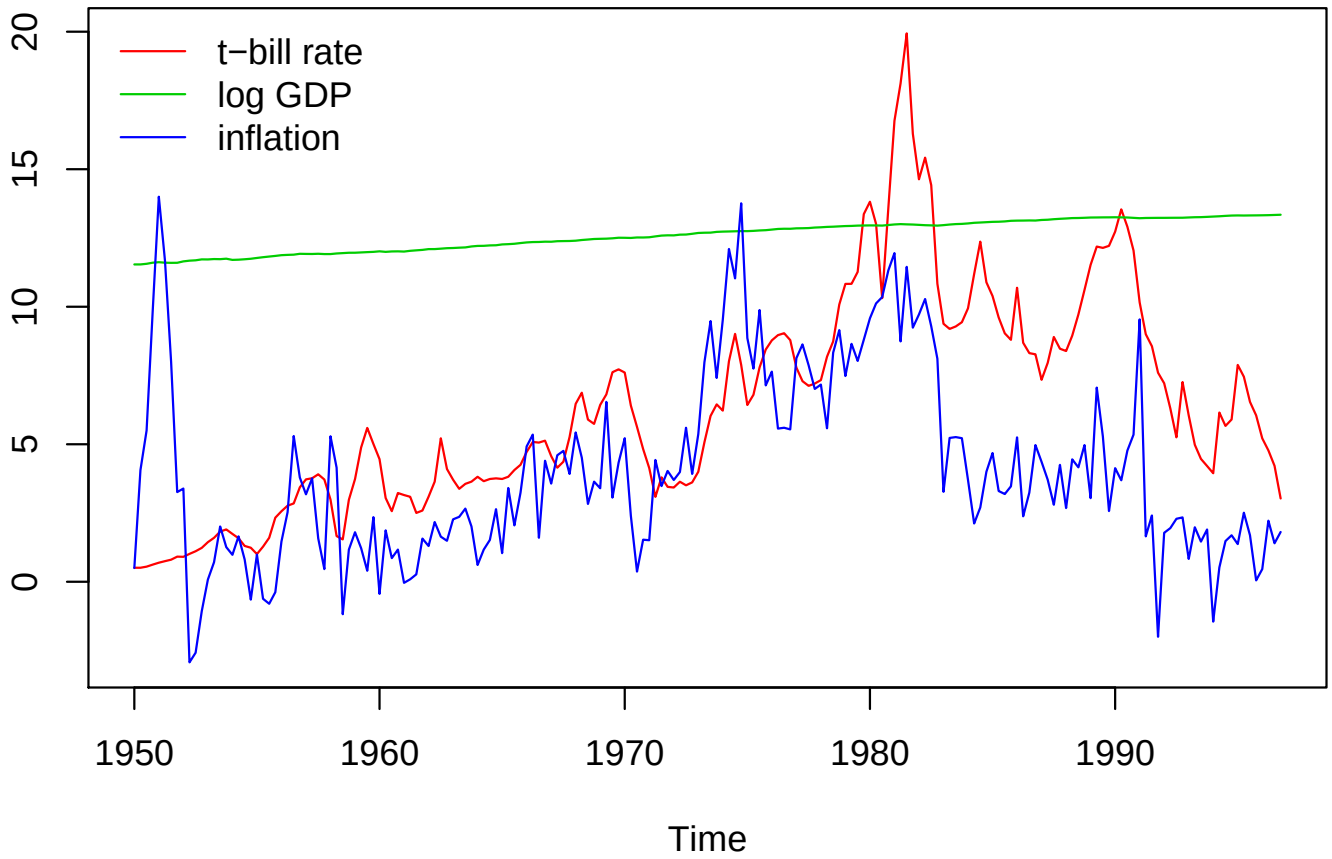
The ts.plot function plots several time series on a common plot. Unlike plot.ts the series can have a different time bases, but they should have the same frequency.

```
args(ts.plot)

## function (..., gpars = list())
## NULL
```

...	one or more univariate or multivariate time series
gpars	list of named graphics parameters to be passed to plotting functions

The ts.plot function

```
ts.plot(Tbrate,col=2:4)
legend(x="topleft",legend=c("t-bill rate","log GDP","inflation"),
  lty=1,col=2:4,bty="n")
```



COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

`http://depts.washington.edu/compfin`