The Magic in R

Stefan Glogger

August 2017

Overview

Please look at the titles to get an overview of what is done when. You can also refer to the introducing sentences of each main title.

Packages, Functions and Parameters

First we load all the relevant packages (are saved separetly in *libraries.R*). Then we show the global parameters (*parameters.R*) and also load the functions (which are shown when needed).

```
source("parameters.R", echo = T)
```

```
##
## > targetRpa <- 0.06
##
## > targetVolpa <- 0.04
##
## > targetDisp <- 0.58
##
## > w <- rep(1, 3)
```

```
# source("functions.R")
```

Data Import

We import the sentiment data. We also import the prices of each index over the relevant time frame.

Sentix

Read the raw sentiment data and save it in the list *sentixRaw* with each list element containing the results of the survey for the different indices. As the number of rows (dates of observation) in data differ, we extract the unique dates (*datesSentix*) and reduce the data to it. We also determine *dateMin* and *dateMax*, which we use lateron to get the stock data.

```
library(openxlsx)
```

```
folderSentix <- (file.path(getwd(), "Data", "Sentix"))

sheets <- c("DAX","DAXm","TEC","TECm","ESX50","ESX50m","SP5","SP5m","NASDAQ","NASDAQm","NIKKEI","NIKKEIm
relevant_rows <- c("Datum","P+","Pn","P-","I+","In","I-","G+","Gn","G-")
```

```
sentixRaw <- list()

for(i in sheets){
  sentixRaw[[i]] <- read.xlsx(file.path(folderSentix, "sentix_anzahlen_bis_02092016xlsx.xlsx"),sheet=i,
  sentixRaw[[i]] <- sentixRaw[[i]][,relevant_rows]
  sentixRaw[[i]] <- sentixRaw[[i]][order(sentixRaw[[i]][,1]),]
}
```

```
unlist(lapply(sentixRaw, nrow))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##     803     803     803     803     803     803     803     803     803
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm   TBOND  TBONDm
##     803     803     803     802     802     802     802
```

```
datesSentix <- unique(sentixRaw[[1]]$Datum)
for(i in names(sentixRaw)[2:length(sentixRaw)]){
  if(!(setequal(datesSentix, sentixRaw[[i]]$Datum)))
    stop("Sentix Data of different indices have not same dates. Handle manually.")
}

for(i in names(sentixRaw)){
  sentixRaw[[i]] <- unique(sentixRaw[[i]])
}
unlist(lapply(sentixRaw, nrow))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##     802     802     802     802     802     802     802     802     802
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm   TBOND  TBONDm
##     802     802     802     802     802     802     802
```

```
(dateMin <- min(datesSentix))
```

```
## [1] "2001-02-23"
```

```
(dateMax <- max(datesSentix))
```

```
## [1] "2016-09-02"
```

```
rm(folderSentix, sheets, relevant_rows, i)
detach("package:openxlsx", unload = T)
```

Stocks

We take data mainly from Yahoo Finance. We take closing course from *dateMin* to *dateMax* for several indexes and store in the data frame *stocks* the closing stock price at each date of the sentiment data (*datesSentix*).

We take the following as sources of the data:

- DAX *ˆGDAXI*
- TEC *ˆTECDAX*
- ESX50 *ˆSTOXX50E*
- SP500 *ˆGSPC*
- NASDAQ *ˆNDX*
- NIKKEI *ˆN225*
- BUND from Sebastian: Den Bund-Future habe ich bei onvista in 5-Jahresst?cken geladen und zusammengebaut. Dezimaltrennzeichen umgestellt im .csv —- not from yahoo, manually from bundesbank *BBK01.WT0557*
- TBOND from Sebastian: Beim T-Bond ist es die 10 Year Treasury Note, auf welche das TBOND Sentiment abzielt. Diese habe ich bei FRED geladen: https://fred.stlouisfed.org/series/DGS10

```
# install.packages("quantmod")
library(quantmod)
# ?getSymbols
```

```
stocks <- data.frame(Datum = datesSentix)

# DAX
dax <- new.env()
getSymbols("^GDAXI", env = dax, src = "yahoo", from = dateMin, to = dateMax)
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).
```

```
## Warning: ^GDAXI contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.
```

```
## [1] "GDAXI"
```

```
DAX <- data.frame(dax$GDAXI[datesSentix,"GDAXI.Close"])
colnames(DAX) <- "Close" # somehow the column name cannot be given directly
```

```r
DAX$Datum <- as.Date(row.names(DAX))

stocks$DAX <- merge(stocks, DAX, by = "Datum", all.x = T)$Close


# TEC
tec <- new.env()
getSymbols("^TECDAX", env = tec, src = "yahoo", from = dateMin, to = dateMax)
```

```
## Warning: ^TECDAX contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.
```

```
## [1] "TECDAX"
```

```r
TEC <- data.frame(tec$TECDAX[datesSentix, "TECDAX.Close"])
colnames(TEC) <- "Close"
TEC$Datum <- as.Date(row.names(TEC))

stocks$TEC <- merge(stocks, TEC, by = "Datum", all.x = T)$Close


# ESX50
esx50 <- new.env()
getSymbols("^STOXX50E", env = esx50, src = "yahoo", from = dateMin, to = dateMax)
```

```
## Warning: ^STOXX50E contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.
```

```
## [1] "STOXX50E"
```

```r
ESX50 <- data.frame(esx50$STOXX50E[datesSentix,"STOXX50E.Close"])
colnames(ESX50) <- "Close"
ESX50$Datum <- as.Date(row.names(ESX50))

stocks$ESX50 <- merge(stocks, ESX50, by = "Datum", all.x = T)$Close


# SP500
sp500 <- new.env()
getSymbols("^GSPC", env = sp500, src = "yahoo", from = dateMin, to = dateMax)
```

```
## [1] "GSPC"
```

```r
SP500 <- data.frame(sp500$GSPC[datesSentix,"GSPC.Close"])
colnames(SP500) <- "Close"
SP500$Datum <- as.Date(row.names(SP500))
# sum(is.na(SP500$Close))

stocks$SP5 <- merge(stocks, SP500, by = "Datum", all.x = T)$Close


# NASDAQ
nasdaq <- new.env()
getSymbols("^NDX", env = nasdaq, src = "yahoo", from = dateMin, to = dateMax)
```

```
## [1] "NDX"
```

```
NASDAQ <- data.frame(nasdaq$NDX[datesSentix,"NDX.Close"])
# sum(is.na(NASDAQ[,"NDX.Close"]))
colnames(NASDAQ) <- "Close"
NASDAQ$Datum <- as.Date(row.names(NASDAQ))

stocks$NASDAQ <- merge(stocks, NASDAQ, by = "Datum", all.x = T)$Close


# NIKKEI
nikkei <- new.env()
getSymbols("^N225", env = nikkei, src = "yahoo", from = dateMin, to = dateMax)
```

```
## Warning: ^N225 contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.
```

```
## [1] "N225"
```

```
NIKKEI <- data.frame(nikkei$N225[datesSentix,"N225.Close"])
colnames(NIKKEI) <- "Close"
NIKKEI$Datum <- as.Date(row.names(NIKKEI))

stocks$NIKKEI <- merge(stocks, NIKKEI, by = "Datum", all.x = T)$Close
```

Bund

```
BUND <- read.csv(file.path(getwd(), "Data", "Bundfuture", "Bundfuture2001-2017.csv"), sep = ";")
BUND[,1] <- as.Date(BUND[,1], format = "%d.%m.%Y")
BUND <- BUND[BUND[,1] %in% datesSentix,]
BUND <- as.data.frame(BUND)

stocks$BUND <- merge(stocks, BUND, by = "Datum", all.x = T)$Schluss
```

Treasury bond

```
TBOND <- read.csv(file.path(getwd(), "Data", "10 year T-Notes", "DGS10.csv"), sep = ",")
TBOND[,1] <- as.Date(TBOND[,1], format = "%Y-%m-%d")
TBOND[,2] <- as.numeric(as.character(TBOND[,2])) # was a factor first and factors are stored via index
```

```
## Warning: NAs durch Umwandlung erzeugt
```

```
colnames(TBOND) <- c("Datum", "DGS10")
TBOND <- TBOND[TBOND[,1] %in% datesSentix,]
TBOND <- as.data.frame(TBOND)

stocks$TBOND <- merge(stocks, TBOND, by = "Datum", all.x = T)$DGS10
```

```
rm(BUND, DAX, ESX50, NASDAQ, NIKKEI, SP500, TBOND, TEC,
   dax, esx50, nasdaq, nikkei, sp500, tec, i)
```

```
## Warning in rm(BUND, DAX, ESX50, NASDAQ, NIKKEI, SP500, TBOND, TEC, dax, :
## Objekt 'i' nicht gefunden
```

Data Preparation

We look at how many people participated in the survey on average and remove TBOND.

We look at the number of dates on which not all stocks report prices and remove those to end up with the dates on which all data is available *datesAll*.

Sentix - number of participants in survey

```
cols <- 8:10
colnames(sentixRaw[[1]])[cols]
```

```
## [1] "G+" "Gn" "G-"
```

```
unlist(lapply(sentixRaw, function(x) {round(mean(rowSums(x[cols])), 0)}))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##     701     698     677     674     696     692     694     690     683
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm   TBOND  TBONDm
##     680     647     643     628     625     160     160
```

```
rm(cols)
```

We remove TBOND, as just very few people voted for it over time in comparison to the other indices.

```
sentixRaw[["TBOND"]] <- NULL
sentixRaw[["TBONDm"]] <- NULL
stocks <- stocks[,-which(colnames(stocks)=="TBOND")]
```

```
unlist(lapply(sentixRaw, function(x) {sum(is.na.data.frame(x))}))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##       0       0       0       0       0       0       0       0       0
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm
##       0       0       0       0       0
```

Stocks - na's

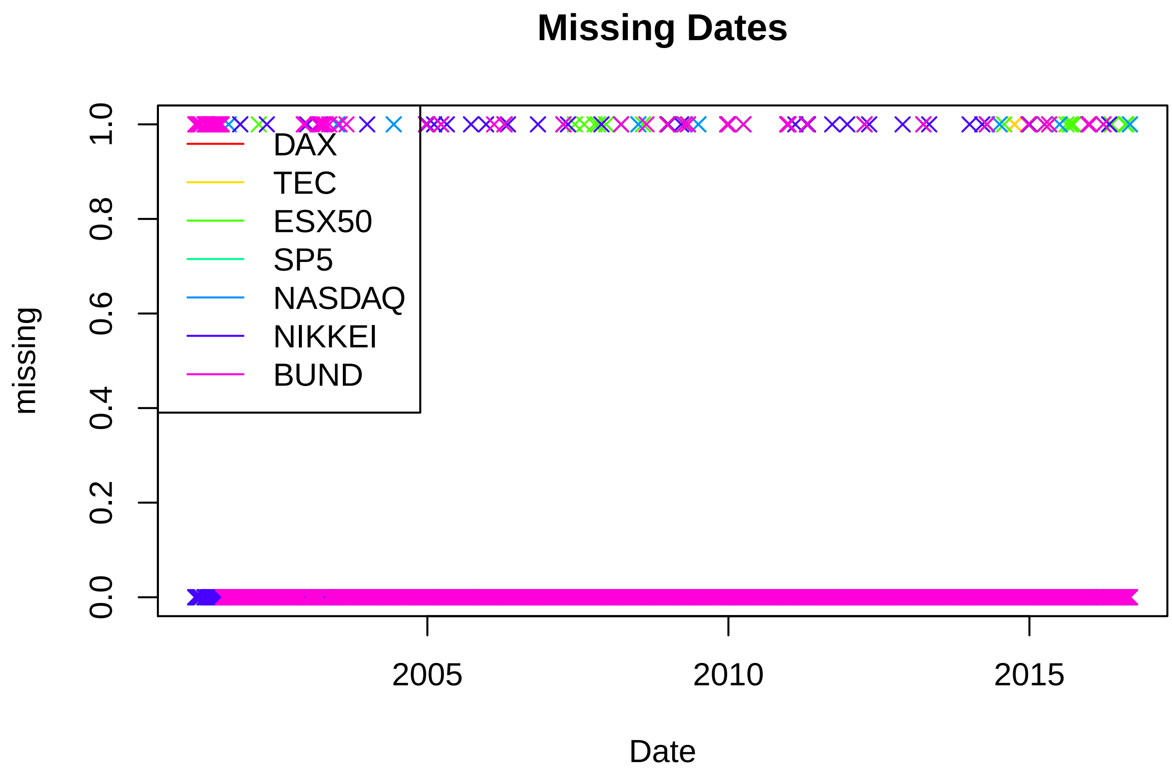
There might be dates missing (we just have to look at stocks as we found the *datesSentix* as those dates, for which all sentiment is there).

```
colSums(is.na.data.frame(stocks))
```

```
##  Datum    DAX    TEC  ESX50    SP5 NASDAQ NIKKEI   BUND
##      0     25     22     41     26     26     32     56
```

Visualize the missing dates (missing date = 1, not missing date = 0 on y-axis).

```
cols <- rainbow(ncol(stocks)-1)

plot(stocks[,1], is.na(stocks[,2]), main = "Missing Dates", ylab = "missing", xlab = "Date", col = cols
for(i in 2:(ncol(stocks)-1)){
  par(new=T)
  plot(stocks[,1], is.na(stocks[,i+1]), col = cols[i], axes = F, xlab = "", ylab = "", pch = 4)
}
legend("topleft", legend = colnames(stocks)[2:ncol(stocks)], col = cols, lty = 1)
```




```r
rm(cols, i)
```

Determine, how many dates do have all data available.

```r
nrow(stocks)
```

```
## [1] 802
```

```r
nrow(stocks[complete.cases(stocks),])
```

```
## [1] 695
```

```r
nrow(stocks) - nrow(stocks[complete.cases(stocks),])
```

```
## [1] 107
```

```r
(nrow(stocks) - nrow(stocks[complete.cases(stocks),]))/nrow(stocks)
```

```
## [1] 0.1334165
```

So we would delete 13.3416459 % of the data.

delete

We delete dates with missing values.

```r
stocks <- stocks[complete.cases(stocks),]

datesAll <- stocks[,1]
rm(datesSentix)
```

```r
sentixRaw <- lapply(sentixRaw, function(x) {x[(x[,1] %in% datesAll),]})

unlist(lapply(sentixRaw, nrow))
```

```
##     DAX    DAXm     TEC    TECm   ESX50  ESX50m     SP5    SP5m  NASDAQ
##     695     695     695     695     695     695     695     695     695
## NASDAQm  NIKKEI NIKKEIm    BUND   BUNDm
##     695     695     695     695     695
```

approach

One way of approaching this might be via linear regression of the stock data when no stock price is available. but this assumes a linear relationship and might cause trouble.

Data Derivations

We calculate dispersion and herfindah for the sentix data.

Sentix

Dispersion

We measure dispersion of the results of the survey (at each date) as its variance.

Fix one date. Let X_i be the respond of participant i to the future state of the stock with $X_i = 1$ representing, he has positive opinion, $X_i = 0$ neutral, $X_i = -1$ negative.

Then we calculate the dispersion of X as:

$$\text{disp}(X) = \text{Var}(X), \text{where} X = (X_1, ...X_n)$$

In alignment to Dominik's code, we perform the calculation for each index, each group of persons (private, institutional and all), and both time periods (1 month, 6 month).

We produce a list named *sDisp*. Each list element (e.g. P1, P6, I1, ...) contains a data frame with the dispersion for each index (column) at each date (row).

```
sDisp <- list()

colnames(sentixRaw[[1]])
```

```
##  [1] "Datum" "P+"    "Pn"    "P-"    "I+"    "In"    "I-"    "G+"
##  [9] "Gn"    "G-"
```

```
groupP <- c("P+", "Pn", "P-")
groupI <- c("I+", "In", "I-")
groupG <- c("G+", "Gn", "G-")
sDispColumn <- function(dat, group){
  res <- numeric(nrow(dat))
  for(i in 1:length(res)){
    res[i] <- var(c(rep(1, dat[i, group[1]]), rep(0, dat[i, group[2]]), rep(-1, dat[i, group[3]])))
  }
  return(res)
}

names(sentixRaw)
```

```
##  [1] "DAX"      "DAXm"     "TEC"      "TECm"     "ESX50"    "ESX50m"   "SP5"
##  [8] "SP5m"     "NASDAQ"   "NASDAQm"  "NIKKEI"   "NIKKEIm"  "BUND"     "BUNDm"
```

```
(period1 <- names(sentixRaw)[2*((0:(length(sentixRaw)/2-1)))+1])
```

```
## [1] "DAX"    "TEC"    "ESX50"  "SP5"    "NASDAQ" "NIKKEI" "BUND"
```

```
(period6 <- names(sentixRaw)[2*((0:(length(sentixRaw)/2-1)))+2])
```

```
## [1] "DAXm"    "TECm"    "ESX50m"  "SP5m"    "NASDAQm" "NIKKEIm" "BUNDm"
```

```
sDispDataFrame <- function(period, group){
  res <- data.frame(Datum = datesAll)

  res$DAX <- sDispColumn(sentixRaw[[period[1]]], group)
```

```
  res$TEC <- sDispColumn(sentixRaw[[period[2]]], group)
  res$ESX50 <- sDispColumn(sentixRaw[[period[3]]], group)
  res$SP5 <- sDispColumn(sentixRaw[[period[4]]], group)
  res$NASDAQ <- sDispColumn(sentixRaw[[period[5]]], group)
  res$NIKKEI <- sDispColumn(sentixRaw[[period[6]]], group)
  res$BUND <- sDispColumn(sentixRaw[[period[7]]], group)

  return(res)
}

sDisp[["P1"]] <- sDispDataFrame(period1, groupP)
sDisp[["P6"]] <- sDispDataFrame(period6, groupP)
sDisp[["I1"]] <- sDispDataFrame(period1, groupI)
sDisp[["I6"]] <- sDispDataFrame(period6, groupI)
sDisp[["G1"]] <- sDispDataFrame(period1, groupG)
sDisp[["G6"]] <- sDispDataFrame(period6, groupG)


# we get a problem as the helping formulas are hard coded
if((ncol(sDisp[[1]])-1) != length(period1))
  stop("Fatal error. Check 'sDispDataFrame'. number of Indices changed")

rm(groupP, groupI, groupG, sDispColumn,
   period1, period6, sDispDataFrame)
```

herfindah

We compute a weighted negative Herfindahl Index, which is a measure of dispersion as given in https://www.federalreserve.gov/pubs/feds/2014/201435/201435pap.pdf. Negative value lets higher values indicate greater dispersion.

At each fixed date, the weighted negative Herfindahl Index is computed by:

$$\operatorname{herf}(X) = -\left[\left(\frac{|\{X_i : X_i = 1\}|}{|\{X_1, ..., X_n\}|}\right)^2 + 2\left(\frac{|\{X_i : X_i = 0\}|}{|\{X_1, ..., X_n\}|}\right)^2 + \left(\frac{|\{X_i : X_i = -1\}|}{|\{X_1, ..., X_n\}|}\right)^2\right]$$

Code in analogy to Dominik's.

We produce a list named *sHerf*. Each list element (e.g. P1, P6, I1, ...) contains a data frame with the dispersion for each index (column) at each date (row).

```
sHerf <- list()

colnames(sentixRaw[[1]])
```

```
##  [1] "Datum" "P+"    "Pn"    "P-"    "I+"    "In"    "I-"    "G+"
##  [9] "Gn"    "G-"
```

```
groupP <- c("P+", "Pn", "P-")
groupI <- c("I+", "In", "I-")
groupG <- c("G+", "Gn", "G-")
sHerfColumn <- function(dat, group){
  res <- numeric(nrow(dat))
  for(i in 1:length(res)){
```

```r
    s <- sum(dat[i, group])
    res[i] <- -1*( (dat[i, group[1]]/s)^2 + 2*(dat[i, group[2]]/s)^2 + (dat[i, group[3]]/s)^2 )
  }
  return(res)
}

names(sentixRaw)
```

```
##  [1] "DAX"     "DAXm"    "TEC"     "TECm"    "ESX50"   "ESX50m"  "SP5"
##  [8] "SP5m"    "NASDAQ"  "NASDAQm" "NIKKEI"  "NIKKEIm" "BUND"    "BUNDm"
```

```r
(period1 <- names(sentixRaw)[2*((0:(length(sentixRaw)/2-1)))+1])
```

```
## [1] "DAX"    "TEC"    "ESX50"  "SP5"    "NASDAQ" "NIKKEI" "BUND"
```

```r
(period6 <- names(sentixRaw)[2*((0:(length(sentixRaw)/2-1)))+2])
```

```
## [1] "DAXm"    "TECm"    "ESX50m"  "SP5m"    "NASDAQm" "NIKKEIm" "BUNDm"
```

```r
sHerfDataFrame <- function(period, group){
  res <- data.frame(Datum = datesAll)

  res$DAX <- sHerfColumn(sentixRaw[[period[1]]], group)
  res$TEC <- sHerfColumn(sentixRaw[[period[2]]], group)
  res$ESX50 <- sHerfColumn(sentixRaw[[period[3]]], group)
  res$SP5 <- sHerfColumn(sentixRaw[[period[4]]], group)
  res$NASDAQ <- sHerfColumn(sentixRaw[[period[5]]], group)
  res$NIKKEI <- sHerfColumn(sentixRaw[[period[6]]], group)
  res$BUND <- sHerfColumn(sentixRaw[[period[7]]], group)

  return(res)
}

sHerf[["P1"]] <- sHerfDataFrame(period1, groupP)
sHerf[["P6"]] <- sHerfDataFrame(period6, groupP)
sHerf[["I1"]] <- sHerfDataFrame(period1, groupI)
sHerf[["I6"]] <- sHerfDataFrame(period6, groupI)
sHerf[["G1"]] <- sHerfDataFrame(period1, groupG)
sHerf[["G6"]] <- sHerfDataFrame(period6, groupG)


# we get a problem as the helping formulas are hard coded
if((ncol(sHerf[[1]])-1) != length(period1))
  stop("Fatal error. Check 'sHerfDataFrame'. number of Indices changed")

rm(groupP, groupI, groupG, sHerfColumn,
   period1, period6, sHerfDataFrame)
```

Stocks

We calculate discrete returns for each date and each stock.

returns

Discrete returns. Be aware that we “loose” the first date now, as we have no idea of the return on day one. Therefore we might also exclude the first date for the other (sentix) variables. We will go on with carefully matching the dates to always consider information ot the actual day.

```
ret <- as.matrix(stocks[2:nrow(stocks),2:ncol(stocks)]/stocks[1:(nrow(stocks)-1),2:ncol(stocks)] - 1)
rownames(ret) <- stocks[2:nrow(stocks), 1]

mu <- colMeans(ret)
C <- cov(ret)
```

```
# sentixRaw <- lapply(sentixRaw, function(x) {x <- x[2:nrow(x), ]})
# sDisp <- lapply(sDisp, function(x) {x <- x[2:nrow(x), ]})
# sHerf <- lapply(sHerf, function(x) {x <- x[2:nrow(x), ]})
#
# stocks <- stocks[2:nrow(stocks), ]
# datesAll <- datesAll[2:nrow(datesAll)]
```

time window

bull and bear

Fix length of time window (l). Calculate return for all stocks (*retWindow*) for all possible time windows (1, l+1, l+2, . . . , T). Equal weights for all returns (of the different indices). Calculate (arithmetic) average of all returns in each possible time window (*retTotal*). Choose the one with lowest (*datesEvalBear*) and highest (*datesEvalBull*).

$$\text{retWindow}_{\text{stock}} = \prod_{k=1}^{l} (1 + \text{ret}_{\text{stock}}(k)) - 1$$

As we calculate with closing prices, we assume that the return is actually of that day (or better spoken of that week). We investment at the very beginning to the opening price, which should be rathly the closing price of the day (week) before).

```
l <- 50

retWindow <- matrix(0, nrow = nrow(ret)-l+1, ncol = ncol(ret))
rownames(retWindow) <- rownames(ret)[l:nrow(ret)]
class(rownames(retWindow)) <- "Date"

for(i in 1:nrow(retWindow)){
    retWindow[i,] <- apply(ret[i:(i+l-1),]+1, 2, function(x) prod(x)-1) # 2 -> columnwise
}

retTotal <- numeric(nrow(retWindow))
retTotal <- apply(retWindow, 1, mean) # 1 -> rowwise
names(retTotal) <- rownames(retWindow)

iMin <- which(retTotal==min(retTotal))
iMax <- which(retTotal==max(retTotal))

# dates of which the returns have been calculated
datesEvalBear <- rownames(ret)[(iMin):(iMin+l-1)]
```

```
datesEvalBull <- rownames(ret)[(iMax):(iMax+l-1)]
class(datesEvalBear) <- "Date"
class(datesEvalBull) <- "Date"
```

additional visualization of the resturns over each time window

```
plot(retTotal, type = "l", axes = FALSE, main = "returns over the time window")
abline(v = iMin, col = "red", lwd = 2)
abline(v = iMax, col = "green", lwd = 2)
axis(1, pretty(1:length(retTotal)), names(retTotal)[pretty(1:length(retTotal))+1])
axis(2)
```

last data

We also look at the most actual data.

```
datesEvalLast <- rownames(ret)[(nrow(ret)-l+1):nrow(ret)]
class(datesEvalLast) <- "Date"
```

used later for storing results. trick *deparse(substitute())* to get an error when a window is deleted.

```
datesNames <- c(deparse(substitute(datesEvalBear)), deparse(substitute(datesEvalBull)), deparse(substitu
```

remove variables

```
rm(l, i)
rm(retWindow, retTotal)
rm(iMin, iMax)
```

TODO further consideration

For the moment, I (Stefan) don't think that the regressing is thoroughly based, so this (updating of code) is skipped for now.

regress Sentiment

We first regress each sentiment on the other sentiments and just go with the non-explained intercept. From these, we calculate the covariance matrix.

```
i <- sentixDataNames[1]
parse(text = paste0(i, "Reg", " <- ", "regSent(", i, ")"))
for (i in sentixDataNames){
    eval(parse(text = paste0(i, "Reg", " <- ", "regSent(", i, ")")))
}

sentixDataNamesReg <- c()
i = 1
parse(text = paste0("sentixDataNamesReg <- ", "c(sentixDataNamesReg, \"", sentixDataNames[i], "Reg\")"))
for(i in sentixDataNames){
    eval(parse(text = paste0("sentixDataNamesReg <- ", "c(sentixDataNamesReg, \"", i, "Reg\")")))
}
```

```
i <- sentixDataNames[i]
parse(text = paste0(i, "RegCov", " <- ", "cov(", i, "Reg)"))
for(i in sentixDataNames){
    eval(parse(text = paste0(i, "RegCov", " <- ", "cov(", i, "Reg)")))
}
```

regression

regress one on all others

We regress one sentiment variable on all other sentiment variables and take the residuals.

```
regSentResidual
```

```
sentixI1dispResiduals50 <- regSentResidual(sentixI1disp, consider = 50, func = mean)
summary(sentixI1dispResiduals50)

sentixI1dispResiduals10 <- regSentResidual(sentixI1disp, consider = 10, func = mean)
summary(sentixI1dispResiduals10)
```

That is not useful! The values differ after the 16th position after decimal point.

Look at what causes this good explanation of one variable by its others:

```
dat <- sentixI1disp
for(k in colnames(dat)){
    # generate formula (regress one column on all the others while using 'consider' previous points)
    print(form <- as.simple.formula(setdiff(colnames(dat), k), k))
    print(summary(lm(form, data = dat[max((200-50),1):200,])))
}
```

do (correct?) adoptation

get Covariance to 0 by regressing one on all before and so on (compare to Portfolio Analysis Theorem 3.5)

Data Visualization

We visualize the data (stocks and sentix). For consistency, we first specify general parameters on how to display each index and the time periods.

overall parameters

Lines with data

```
geomLineDataDAX <- function(x){
    parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = DAX, colour = \"DAX\"))"))
}
geomLineDataTEC <- function(x){
    parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = TEC, colour = \"TEC\"))"))
}
geomLineDataESX50 <- function(x){
    parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = ESX50, colour = \"ESX50\"))"))
}
geomLineDataSP5 <- function(x){
    parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = SP5, colour = \"SP5\"))"))
}
geomLineDataNASDAQ <- function(x){
    parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = NASDAQ, colour = \"NASDAQ\"))"))
}
geomLineDataNIKKEI <- function(x){
    parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = NIKKEI, colour = \"NIKKEI\"))"))
}
geomLineDataBUND <- function(x){
    parse(text = paste0("geom_line(data = ", x, ", aes(x = Datum, y = BUND, colour = \"BUND\"))"))
}
```

probierer, funktioniert nicht (wollte alle linien auf einmal plotten)

```
# geomLineData <- function(x){
#     parse(text = paste0("eval(geomLineDataDAX(\"", x , "\")) + eval(geomLineDataTEC(\"", x , "\"))"))
# }
#
# ggplot() +
#     eval(geomLineData("retPlot")) +
#     eval(geomRectDateLast) +
#     labs(x = "Time", y = "Value")
```

Rectangle for Date periods

store as function to keep structure similar to above (and store at same Place in environment)

```
geomRectDateLast <- function(){
    parse(text = "geom_rect(aes(xmin = min(datesEvalLast), xmax = max(datesEvalLast), ymin = -Inf, ymax

geomRectDateBear <- function(){
    parse(text = "geom_rect(aes(xmin = min(datesEvalBear), xmax = max(datesEvalBear), ymin = -Inf, ymax
```

```
geomRectDateBull <- function(){
    parse(text = "geom_rect(aes(xmin = min(datesEvalBull), xmax = max(datesEvalBull), ymin = -Inf, ymax
```

Function for plotting

```
plotData <- function(x, title = "Indices"){
    ggplot() +
        eval(geomLineDataDAX(x)) +
        eval(geomLineDataTEC(x)) +
        eval(geomLineDataESX50(x)) +
        eval(geomLineDataNASDAQ(x)) +
        eval(geomLineDataNIKKEI(x)) +
        eval(geomLineDataBUND(x)) +
        eval(geomRectDateLast()) +
        eval(geomRectDateBear()) +
        eval(geomRectDateBull()) +
        labs(x = "Time", y = "Value") +
        labs(title = title) +
        theme(plot.title = element_text(hjust = 0.5)) # align title in center
}

## if a special name is given, take it, otherwise take x (plot sentix by using same dataframe (adopted)
plotDataPDF <- function(x, xName = x){
    pdf(file.path(getwd(), "Plot Data", paste0(xName, ".pdf")), width = 10, height = 4)
    plot(plotData(x))
    dev.off()
}
```

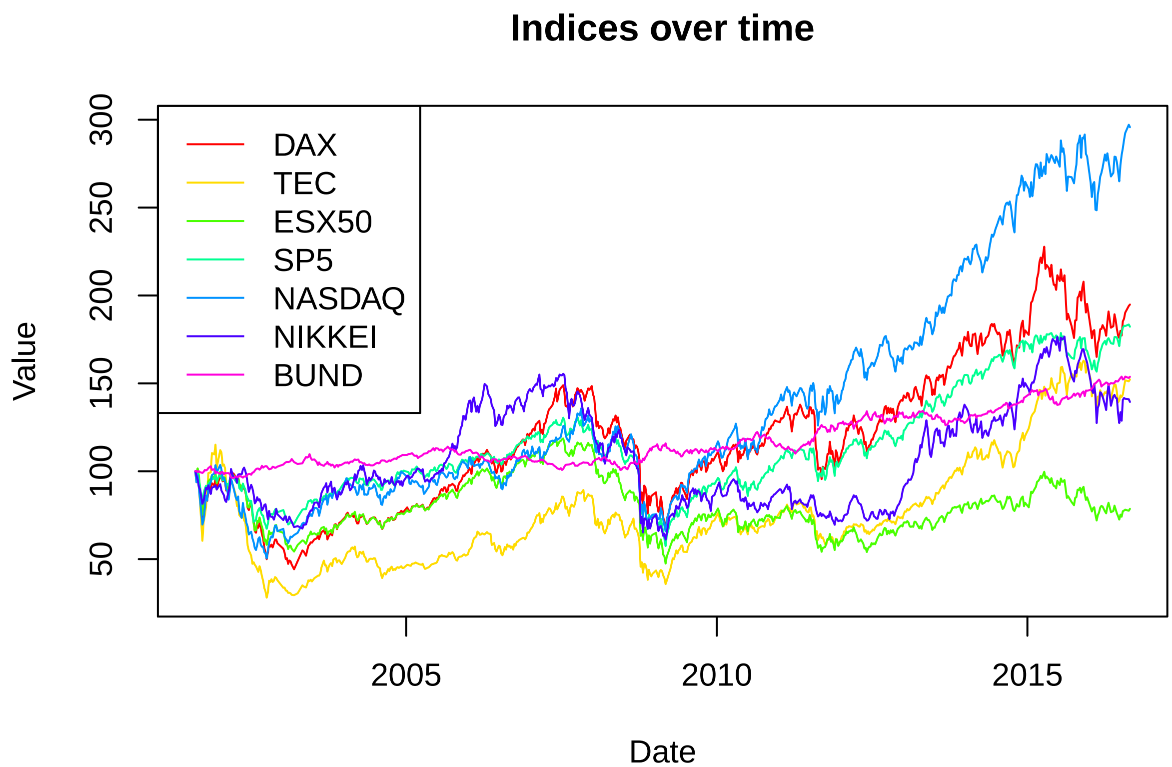
Stocks

Start of with a value of 100 for each stock and then plot the evolvment of this stock.

plot()

```
retPlot <- matrix(100, nrow = nrow(stocks), ncol = ncol(stocks)-1)
retPlot[2:nrow(stocks), ] <- 1+ret # to multiply lateron, we have to add 1
retPlot <- apply(retPlot, 2, cumprod)
rownames(retPlot) <- stocks[,1]

xNames <- rownames(retPlot)
class(xNames) <- "Date"   # convert to date

cols <- rainbow(ncol(retPlot))
ylim <- c(min(retPlot), max(retPlot))
plot(xNames, retPlot[,1], type = "l", xlab = "Date", ylab = "Value", main = "Indices over time",
     col = cols[1], ylim = ylim)
for(i in 2:ncol(retPlot)){
    par(new=T)
    plot(xNames, retPlot[,i], type = "l", col = cols[i], axes = F, xlab="", ylab="", ylim = ylim)
```

```
}
legend("topleft", legend = colnames(stocks)[2:ncol(stocks)], col = cols, lty = 1)
```



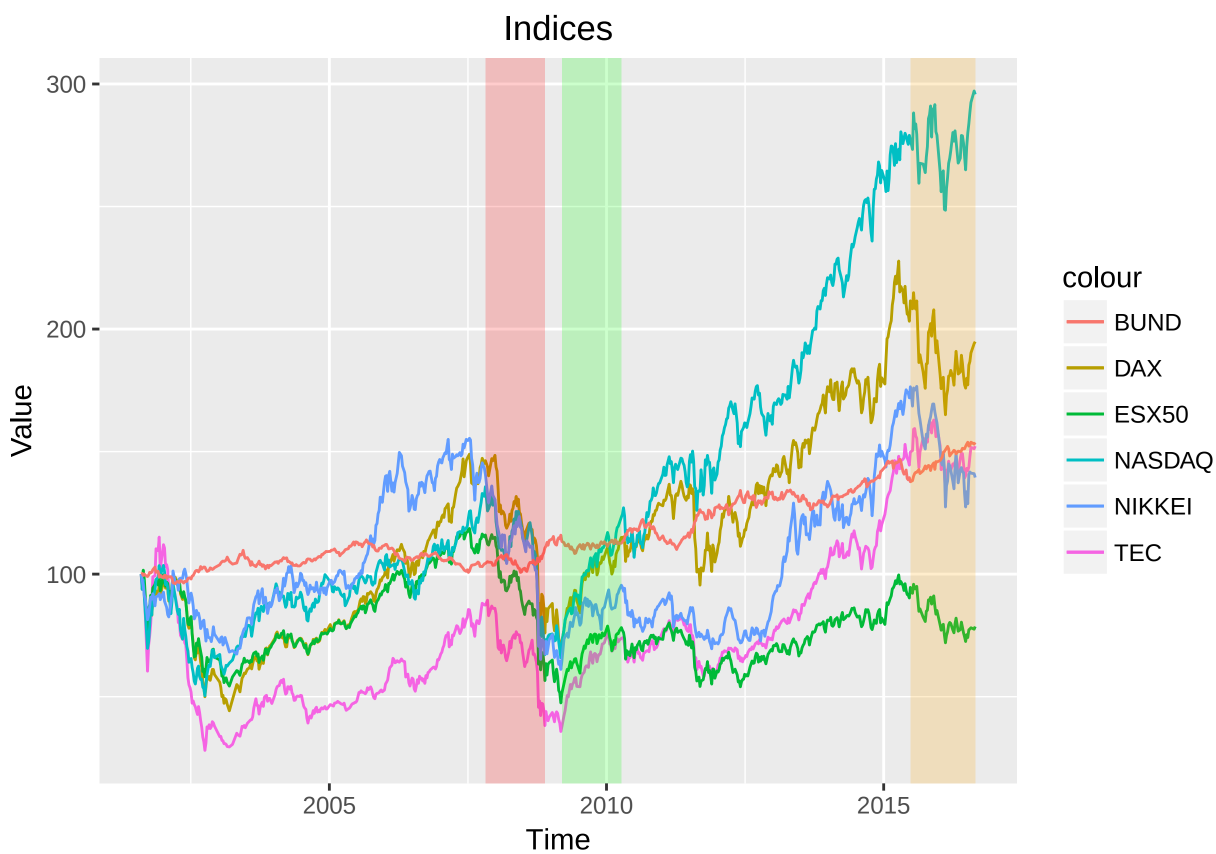

```
rm(retPlot, xNames, ylim, i)
```

ggplot()

```
library(ggplot2)
```

need data frame as input for ggplot

```
retPlot <- matrix(100, nrow = nrow(stocks), ncol = ncol(stocks)-1)
retPlot[2:nrow(stocks), ] <- 1+ret # to multiply lateron, we have to add 1
retPlot <- apply(retPlot, 2, cumprod)

retPlot <- as.data.frame(retPlot)
colnames(retPlot) <- colnames(stocks)[2:ncol(stocks)]
retPlot$Datum <- stocks[,1]
class(retPlot$Datum) <- "Date"   # convert to date

cols <- rainbow(ncol(retPlot))
ylim <- c(min(retPlot[,1:(ncol(retPlot)-1)]), max(retPlot[,1:(ncol(retPlot)-1)]))


plotData("retPlot")
```




```
plotDataPDF("retPlot")
```

```
## pdf
##   2
```

Dispersion

Graphs can be found in "\R-Research Project Statistics\Plot Data".

```
for(i in names(sDisp)){
    sPlot <- sDisp[[i]]
    plotDataPDF("sPlot", paste("sDisp", i))
}
```

And we provide summary statistics.

```
lapply(sDisp, function(x) {base::summary(x[,-1], digits = 2)})
```

```
## $P1
##       DAX            TEC            ESX50           SP5
##  Min.   :0.39   Min.   :0.39   Min.   :0.39   Min.   :0.39
##  1st Qu.:0.55   1st Qu.:0.54   1st Qu.:0.53   1st Qu.:0.51
##  Median :0.58   Median :0.57   Median :0.56   Median :0.55
##  Mean   :0.58   Mean   :0.57   Mean   :0.56   Mean   :0.55
##  3rd Qu.:0.62   3rd Qu.:0.60   3rd Qu.:0.59   3rd Qu.:0.58
##  Max.   :0.76   Max.   :0.74   Max.   :0.75   Max.   :0.73
##      NASDAQ          NIKKEI           BUND
```

```
##  Min.   :0.42   Min.   :0.31   Min.   :0.15  
##  1st Qu.:0.53   1st Qu.:0.48   1st Qu.:0.37  
##  Median :0.56   Median :0.51   Median :0.41  
##  Mean   :0.56   Mean   :0.51   Mean   :0.40  
##  3rd Qu.:0.59   3rd Qu.:0.54   3rd Qu.:0.45  
##  Max.   :0.74   Max.   :0.71   Max.   :0.57  
## 
## $P6
##       DAX             TEC            ESX50            SP5       
##  Min.   :0.49   Min.   :0.46   Min.   :0.49   Min.   :0.47  
##  1st Qu.:0.63   1st Qu.:0.62   1st Qu.:0.62   1st Qu.:0.61  
##  Median :0.66   Median :0.65   Median :0.65   Median :0.64  
##  Mean   :0.66   Mean   :0.65   Mean   :0.64   Mean   :0.64  
##  3rd Qu.:0.69   3rd Qu.:0.68   3rd Qu.:0.68   3rd Qu.:0.67  
##  Max.   :0.76   Max.   :0.75   Max.   :0.75   Max.   :0.75  
##      NASDAQ          NIKKEI          BUND      
##  Min.   :0.49   Min.   :0.37   Min.   :0.38  
##  1st Qu.:0.62   1st Qu.:0.56   1st Qu.:0.49  
##  Median :0.65   Median :0.60   Median :0.52  
##  Mean   :0.65   Mean   :0.59   Mean   :0.52  
##  3rd Qu.:0.68   3rd Qu.:0.62   3rd Qu.:0.55  
##  Max.   :0.75   Max.   :0.71   Max.   :0.66  
## 
## $I1
##       DAX             TEC            ESX50            SP5       
##  Min.   :0.30   Min.   :0.34   Min.   :0.30   Min.   :0.33  
##  1st Qu.:0.55   1st Qu.:0.53   1st Qu.:0.53   1st Qu.:0.51  
##  Median :0.59   Median :0.58   Median :0.58   Median :0.55  
##  Mean   :0.59   Mean   :0.58   Mean   :0.58   Mean   :0.56  
##  3rd Qu.:0.63   3rd Qu.:0.62   3rd Qu.:0.62   3rd Qu.:0.60  
##  Max.   :0.85   Max.   :0.80   Max.   :0.83   Max.   :0.81  
##      NASDAQ          NIKKEI          BUND      
##  Min.   :0.31   Min.   :0.27   Min.   :0.29  
##  1st Qu.:0.51   1st Qu.:0.46   1st Qu.:0.44  
##  Median :0.56   Median :0.50   Median :0.49  
##  Mean   :0.56   Mean   :0.51   Mean   :0.49  
##  3rd Qu.:0.61   3rd Qu.:0.55   3rd Qu.:0.54  
##  Max.   :0.79   Max.   :0.78   Max.   :0.78  
## 
## $I6
##       DAX             TEC            ESX50            SP5       
##  Min.   :0.41   Min.   :0.40   Min.   :0.39   Min.   :0.44  
##  1st Qu.:0.61   1st Qu.:0.61   1st Qu.:0.60   1st Qu.:0.59  
##  Median :0.66   Median :0.65   Median :0.65   Median :0.63  
##  Mean   :0.65   Mean   :0.65   Mean   :0.64   Mean   :0.63  
##  3rd Qu.:0.70   3rd Qu.:0.69   3rd Qu.:0.69   3rd Qu.:0.68  
##  Max.   :0.82   Max.   :0.80   Max.   :0.81   Max.   :0.77  
##      NASDAQ          NIKKEI          BUND      
##  Min.   :0.43   Min.   :0.36   Min.   :0.28  
##  1st Qu.:0.60   1st Qu.:0.53   1st Qu.:0.49  
##  Median :0.63   Median :0.58   Median :0.56  
##  Mean   :0.63   Mean   :0.57   Mean   :0.55  
##  3rd Qu.:0.67   3rd Qu.:0.62   3rd Qu.:0.61  
##  Max.   :0.81   Max.   :0.73   Max.   :0.75  
```

```
##
## $G1
##       DAX            TEC            ESX50            SP5
##  Min.   :0.39   Min.   :0.40   Min.   :0.39   Min.   :0.38
##  1st Qu.:0.55   1st Qu.:0.54   1st Qu.:0.54   1st Qu.:0.52
##  Median :0.59   Median :0.57   Median :0.57   Median :0.55
##  Mean   :0.59   Mean   :0.57   Mean   :0.57   Mean   :0.55
##  3rd Qu.:0.62   3rd Qu.:0.61   3rd Qu.:0.60   3rd Qu.:0.58
##  Max.   :0.78   Max.   :0.75   Max.   :0.76   Max.   :0.75
##      NASDAQ          NIKKEI           BUND
##  Min.   :0.42   Min.   :0.32   Min.   :0.21
##  1st Qu.:0.53   1st Qu.:0.48   1st Qu.:0.39
##  Median :0.56   Median :0.51   Median :0.43
##  Mean   :0.56   Mean   :0.51   Mean   :0.43
##  3rd Qu.:0.59   3rd Qu.:0.54   3rd Qu.:0.47
##  Max.   :0.75   Max.   :0.73   Max.   :0.59
##
## $G6
##       DAX            TEC            ESX50            SP5
##  Min.   :0.52   Min.   :0.48   Min.   :0.49   Min.   :0.49
##  1st Qu.:0.63   1st Qu.:0.62   1st Qu.:0.62   1st Qu.:0.61
##  Median :0.66   Median :0.66   Median :0.65   Median :0.64
##  Mean   :0.66   Mean   :0.65   Mean   :0.65   Mean   :0.64
##  3rd Qu.:0.69   3rd Qu.:0.68   3rd Qu.:0.68   3rd Qu.:0.67
##  Max.   :0.76   Max.   :0.75   Max.   :0.75   Max.   :0.75
##      NASDAQ          NIKKEI           BUND
##  Min.   :0.50   Min.   :0.39   Min.   :0.38
##  1st Qu.:0.62   1st Qu.:0.56   1st Qu.:0.49
##  Median :0.65   Median :0.59   Median :0.53
##  Mean   :0.65   Mean   :0.59   Mean   :0.53
##  3rd Qu.:0.67   3rd Qu.:0.62   3rd Qu.:0.56
##  Max.   :0.74   Max.   :0.71   Max.   :0.67
```

Herfindahl

Graphs can be found in "\R-Research Project Statistics\Plot Data".

```
for(i in names(sHerf)){
    sPlot <- sHerf[[i]]
    plotDataPDF("sPlot", paste("sHerf", i))
}
```

And we provide summary statistics.

```
lapply(sHerf, function(x) {base::summary(x[,-1], digits = 2)})
```

```
## $P1
##       DAX             TEC             ESX50             SP5
##  Min.   :-0.67   Min.   :-0.67   Min.   :-0.76   Min.   :-0.82
##  1st Qu.:-0.53   1st Qu.:-0.54   1st Qu.:-0.55   1st Qu.:-0.57
##  Median :-0.50   Median :-0.51   Median :-0.52   Median :-0.54
##  Mean   :-0.51   Mean   :-0.51   Mean   :-0.52   Mean   :-0.54
##  3rd Qu.:-0.48   3rd Qu.:-0.49   3rd Qu.:-0.49   3rd Qu.:-0.50
##  Max.   :-0.41   Max.   :-0.41   Max.   :-0.41   Max.   :-0.42
##      NASDAQ           NIKKEI            BUND
```

```
##  Min.   :-0.71   Min.   :-0.90   Min.   :-1.45
##  1st Qu.:-0.56   1st Qu.:-0.63   1st Qu.:-0.86
##  Median :-0.52   Median :-0.58   Median :-0.77
##  Mean   :-0.53   Mean   :-0.59   Mean   :-0.78
##  3rd Qu.:-0.49   3rd Qu.:-0.54   3rd Qu.:-0.67
##  Max.   :-0.41   Max.   :-0.42   Max.   :-0.51
##
## $P6
##       DAX              TEC             ESX50              SP5
##  Min.   :-0.61   Min.   :-0.66   Min.   :-0.63   Min.   :-0.65
##  1st Qu.:-0.47   1st Qu.:-0.47   1st Qu.:-0.47   1st Qu.:-0.48
##  Median :-0.45   Median :-0.45   Median :-0.46   Median :-0.46
##  Mean   :-0.45   Mean   :-0.46   Mean   :-0.46   Mean   :-0.47
##  3rd Qu.:-0.43   3rd Qu.:-0.44   3rd Qu.:-0.44   3rd Qu.:-0.45
##  Max.   :-0.40   Max.   :-0.41   Max.   :-0.41   Max.   :-0.41
##      NASDAQ           NIKKEI            BUND
##  Min.   :-0.61   Min.   :-0.87   Min.   :-0.71
##  1st Qu.:-0.47   1st Qu.:-0.51   1st Qu.:-0.58
##  Median :-0.45   Median :-0.49   Median :-0.54
##  Mean   :-0.46   Mean   :-0.50   Mean   :-0.55
##  3rd Qu.:-0.44   3rd Qu.:-0.47   3rd Qu.:-0.52
##  Max.   :-0.41   Max.   :-0.42   Max.   :-0.45
##
## $I1
##       DAX              TEC             ESX50              SP5
##  Min.   :-0.76   Min.   :-0.74   Min.   :-0.73   Min.   :-0.81
##  1st Qu.:-0.53   1st Qu.:-0.56   1st Qu.:-0.54   1st Qu.:-0.58
##  Median :-0.50   Median :-0.51   Median :-0.51   Median :-0.53
##  Mean   :-0.50   Mean   :-0.52   Mean   :-0.51   Mean   :-0.54
##  3rd Qu.:-0.47   3rd Qu.:-0.48   3rd Qu.:-0.47   3rd Qu.:-0.49
##  Max.   :-0.40   Max.   :-0.40   Max.   :-0.40   Max.   :-0.40
##      NASDAQ           NIKKEI            BUND
##  Min.   :-0.76   Min.   :-1.10   Min.   :-1.03
##  1st Qu.:-0.58   1st Qu.:-0.64   1st Qu.:-0.69
##  Median :-0.53   Median :-0.58   Median :-0.61
##  Mean   :-0.54   Mean   :-0.59   Mean   :-0.63
##  3rd Qu.:-0.49   3rd Qu.:-0.53   3rd Qu.:-0.54
##  Max.   :-0.40   Max.   :-0.40   Max.   :-0.42
##
## $I6
##       DAX              TEC             ESX50              SP5
##  Min.   :-0.61   Min.   :-0.68   Min.   :-0.60   Min.   :-0.71
##  1st Qu.:-0.48   1st Qu.:-0.48   1st Qu.:-0.49   1st Qu.:-0.50
##  Median :-0.45   Median :-0.45   Median :-0.45   Median :-0.47
##  Mean   :-0.46   Mean   :-0.46   Mean   :-0.46   Mean   :-0.47
##  3rd Qu.:-0.43   3rd Qu.:-0.43   3rd Qu.:-0.43   3rd Qu.:-0.44
##  Max.   :-0.40   Max.   :-0.40   Max.   :-0.40   Max.   :-0.41
##      NASDAQ           NIKKEI            BUND
##  Min.   :-0.65   Min.   :-0.83   Min.   :-0.97
##  1st Qu.:-0.49   1st Qu.:-0.53   1st Qu.:-0.56
##  Median :-0.47   Median :-0.50   Median :-0.51
##  Mean   :-0.47   Mean   :-0.51   Mean   :-0.52
##  3rd Qu.:-0.44   3rd Qu.:-0.48   3rd Qu.:-0.48
##  Max.   :-0.41   Max.   :-0.41   Max.   :-0.42
```

```
##
## $G1
##       DAX              TEC              ESX50            SP5
##  Min.   :-0.65   Min.   :-0.67   Min.   :-0.67   Min.   :-0.77
##  1st Qu.:-0.53   1st Qu.:-0.54   1st Qu.:-0.54   1st Qu.:-0.57
##  Median :-0.50   Median :-0.51   Median :-0.52   Median :-0.53
##  Mean   :-0.50   Mean   :-0.51   Mean   :-0.52   Mean   :-0.54
##  3rd Qu.:-0.48   3rd Qu.:-0.48   3rd Qu.:-0.49   3rd Qu.:-0.50
##  Max.   :-0.40   Max.   :-0.41   Max.   :-0.41   Max.   :-0.41
##      NASDAQ          NIKKEI           BUND
##  Min.   :-0.71   Min.   :-0.94   Min.   :-1.27
##  1st Qu.:-0.56   1st Qu.:-0.62   1st Qu.:-0.80
##  Median :-0.52   Median :-0.58   Median :-0.72
##  Mean   :-0.53   Mean   :-0.59   Mean   :-0.73
##  3rd Qu.:-0.49   3rd Qu.:-0.54   3rd Qu.:-0.64
##  Max.   :-0.41   Max.   :-0.41   Max.   :-0.49
##
## $G6
##       DAX              TEC              ESX50            SP5
##  Min.   :-0.56   Min.   :-0.63   Min.   :-0.58   Min.   :-0.60
##  1st Qu.:-0.46   1st Qu.:-0.47   1st Qu.:-0.47   1st Qu.:-0.48
##  Median :-0.45   Median :-0.45   Median :-0.45   Median :-0.46
##  Mean   :-0.45   Mean   :-0.46   Mean   :-0.46   Mean   :-0.47
##  3rd Qu.:-0.43   3rd Qu.:-0.44   3rd Qu.:-0.44   3rd Qu.:-0.44
##  Max.   :-0.40   Max.   :-0.41   Max.   :-0.41   Max.   :-0.41
##      NASDAQ          NIKKEI           BUND
##  Min.   :-0.60   Min.   :-0.82   Min.   :-0.68
##  1st Qu.:-0.47   1st Qu.:-0.51   1st Qu.:-0.57
##  Median :-0.46   Median :-0.49   Median :-0.53
##  Mean   :-0.46   Mean   :-0.50   Mean   :-0.54
##  3rd Qu.:-0.44   3rd Qu.:-0.48   3rd Qu.:-0.51
##  Max.   :-0.41   Max.   :-0.42   Max.   :-0.44
```

Optimization of Portfolios

classic portfolio optimization

First of all, we do a classic portfolio optimization. We start of with a mean variance diagram.

notation

Let $x = (x_1, ..., x_p)^T$ represent the portfolio (x_i is percentage of available capital invested in security i). Therefore it holds $\sum_{i=1}^p x_i = 1$. Note, that short selling is allowed.

Let $R = (R_1, ..., R_p)^T$ represent the annual returns (R_i is return of security i). And let $\mu = (\mu_1, ..., \mu_p)^T$ represent the expected returns ($\mu_i = \mathrm{E}[R_i] > 0$).

Furthermore $C = (c_{ij})_{i,j \in \{1,...,p\}}$ denotes the (annual) covariance matrix ($c_{ij} = \mathrm{Cov}(R_i, R_j)$).

Then we have Return $R(x)$ of portfolio x given by $R(x) = \sum_{i=1}^p x_i R_i = x^T R$.

The expected return $\mu(x)$ of portfolio x is given by $\mu(x) = \mathrm{E}[R(x)] = \sum_{i=1}^p x_i \mu_i = x^T \mu$.

The Variance $\sigma^2(x)$ of portfolio x is given by $\sigma^2(x) = \mathrm{Var}(R(x)) = \mathrm{E}[(R(x) - \mathrm{E}(R(x)))^2] = x^T C x$.

We therefore annualize the returns and the variance.

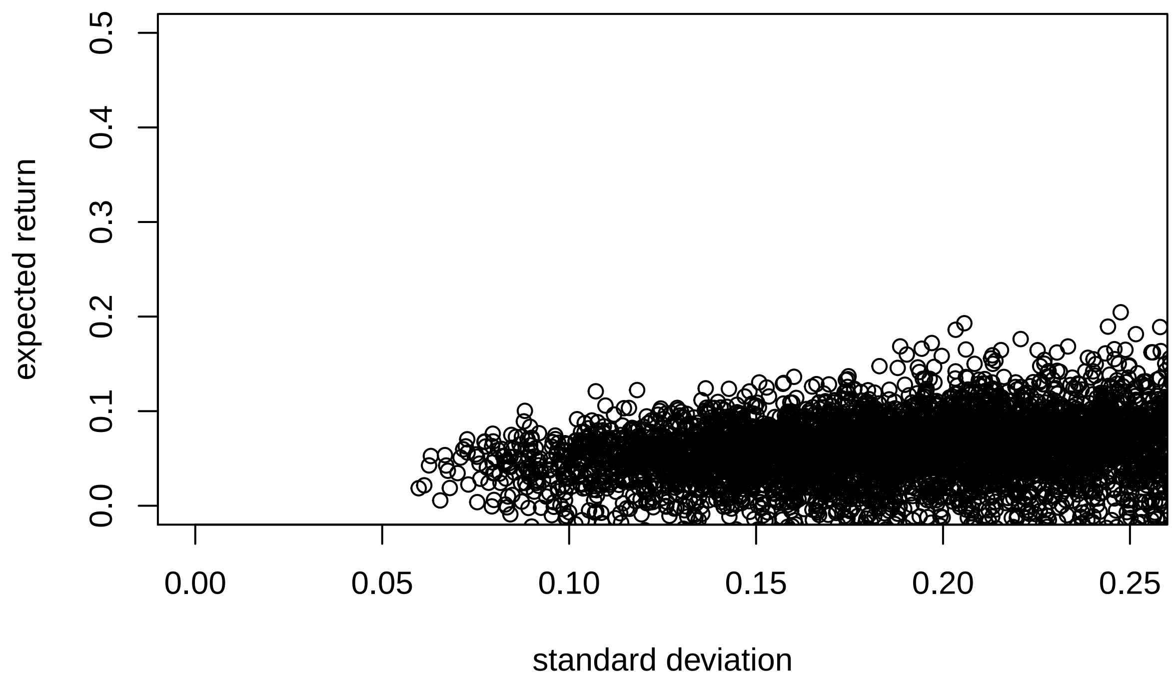
```
anRet <- (1+ret)^52-1
anMu <- (1+mu)^52-1
anC <- C*52
```

mean variance diagram

We plot K random portfolios.

with riskless asset

```
set.seed(1)
K <- 10000

mvRandom <- matrix(0, ncol = 2, nrow = K)
for(i in 1:nrow(mvRandom)){
    x <- rnorm(ncol(ret))
    x <- x/sum(x) # normalize

    mvRandom[i, 1] <- sum(x*anMu)
    mvRandom[i, 2] <- sqrt((x%*%anC)%*%x)
}

plot(mvRandom[,2], mvRandom[,1],
     xlab = "standard deviation", ylab = "expected return",
     xlim = c(0, 0.25), ylim = c(0, 0.5))
```



exclude riskless assets

exclude riskless asset (BUND)

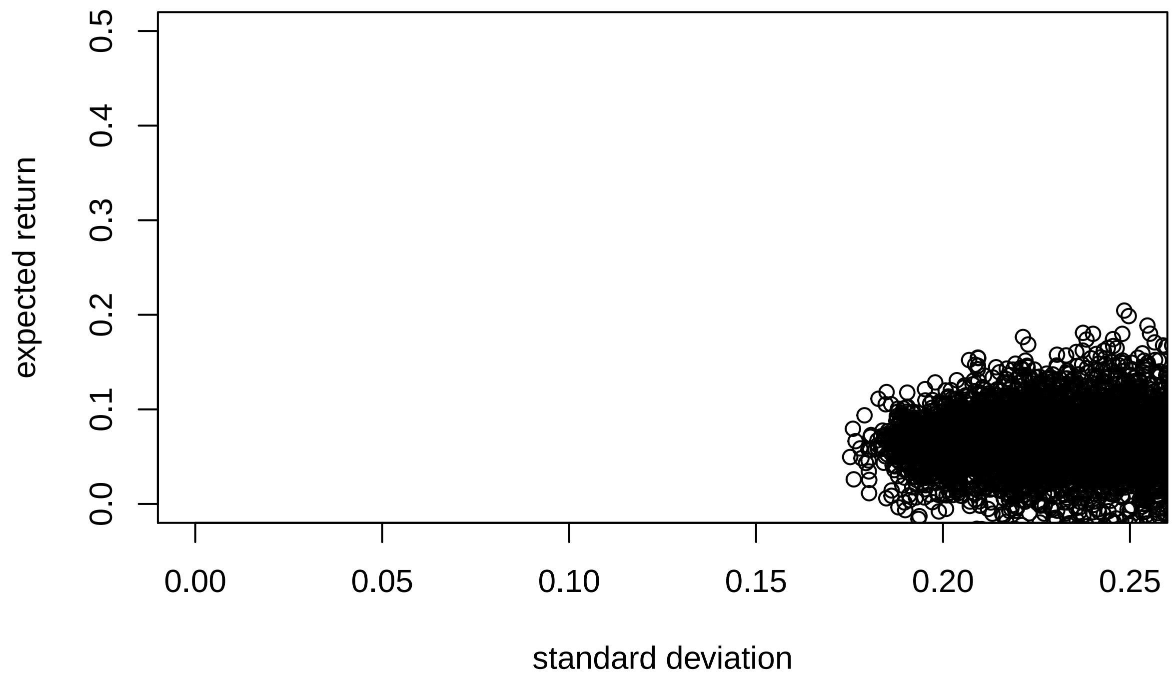
```
retRisky <- ret[,-7]
colnames(retRisky)
```

```
## [1] "DAX"    "TEC"    "ESX50"  "SP5"    "NASDAQ" "NIKKEI"
```

```
muRisky <- colMeans(retRisky)
CRisky <- cov(retRisky)

anRetRisky <- (1+retRisky)^52-1
anMuRisky <- (1+muRisky)^52-1
anCRisky <- CRisky*52
```

```
set.seed(1)
K <- 10000

mvRandom <- matrix(0, ncol = 2, nrow = K)
for(i in 1:nrow(mvRandom)){
    x <- rnorm(ncol(retRisky))
    x <- x/sum(x) # normalize

    mvRandom[i, 1] <- sum(x*anMuRisky)
    mvRandom[i, 2] <- sqrt((x%*%anCRisky)%*%x)
}
```

```
plot(mvRandom[,2], mvRandom[,1],
     xlab = "standard deviation", ylab = "expected return",
     xlim = c(0, 0.25), ylim = c(0, 0.5))
```



efficiency without risk free portfolio

We can use theorem 2.2. of Portfolio Analysis (slide 40). But be careful as C is close to singular.

efficiency line by formula d)

```
det(anC)
```

```
## [1] 1.05804e-13
```

```
det(anCRisky)
```

```
## [1] 3.151767e-11
```

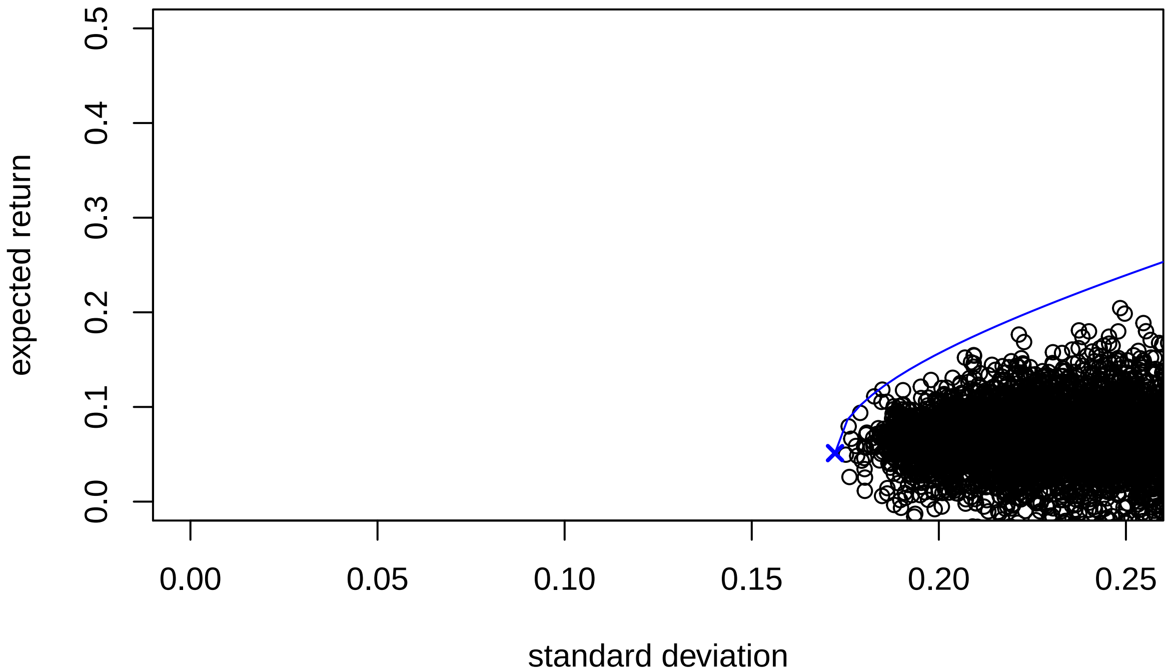
```
anCRisky1 <- solve(anCRisky)
anCRisky %*% anCRisky1
```

```
##                 DAX           TEC         ESX50           SP5
## DAX    1.000000e+00 -5.551115e-17 -2.775558e-16 -4.163336e-16
## TEC    2.126771e-15  1.000000e+00  1.498801e-15 -1.276756e-15
## ESX50  1.491862e-15  4.163336e-16  1.000000e+00 -2.220446e-16
## SP5    1.261144e-15  3.608225e-16  1.665335e-16  1.000000e+00
## NASDAQ 1.065120e-15  3.191891e-16 -3.330669e-16  6.383782e-16
```

```
## NIKKEI 8.326673e-16 -2.220446e-16 -5.551115e-16 -8.326673e-16
##               NASDAQ        NIKKEI
## DAX    -2.359224e-16 -2.220446e-16
## TEC     7.077672e-16 -2.220446e-16
## ESX50  -2.359224e-16 -2.220446e-16
## SP5    -1.734723e-16 -1.110223e-16
## NASDAQ  1.000000e+00  0.000000e+00
## NIKKEI  8.049117e-16  1.000000e+00
```

```r
a <- sum(anCRisky1 %*% anMuRisky)
b <- c((anMuRisky %*% anCRisky1) %*% anMuRisky)
c <- sum(anCRisky1)
d <- b*c - a^2
```

```r
set.seed(1)
K <- 10000

mvRandom <- matrix(0, ncol = 2, nrow = K)
for(i in 1:nrow(mvRandom)){
    x <- rnorm(ncol(retRisky))
    x <- x/sum(x) # normalize

    mvRandom[i, 1] <- sum(x*anMuRisky)
    mvRandom[i, 2] <- sqrt((x%*%anCRisky)%*%x)
}

plot(mvRandom[,2], mvRandom[,1],
     xlab = "standard deviation", ylab = "expected return",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

k <- 100
elWithout <- matrix(0, ncol = 2, nrow = k)
elWithout[,2] <- seq(sqrt(1/c), 0.5, length.out = k)
for(i in 1:nrow(elWithout)){
    elWithout[i,1] <- a/c + sqrt(d/c*(elWithout[i,2]^2 - 1/c))
}
par(new=T)
plot(elWithout[,2], elWithout[,1], type = "l", col = "blue",
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

par(new=T)
plot(sqrt(1/c), a/c,
     col = "blue", pch = 4, lwd = 2,
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))
```




```r
(xMVPwithoutRF <- 1/c*rowSums(anCRisky1))
```

```
##         DAX         TEC       ESX50         SP5      NASDAQ      NIKKEI
## -0.16044087 -0.09906128 -0.09838768  1.31668249 -0.21422886  0.25543620
```

```r
c(a/c, xMVPwithoutRF %*% anMuRisky)
```

```
## [1] 0.0512193 0.0512193
```

```r
c(sqrt(1/c), sqrt( (xMVPwithoutRF%*%anCRisky)) %*% xMVPwithoutRF)
```

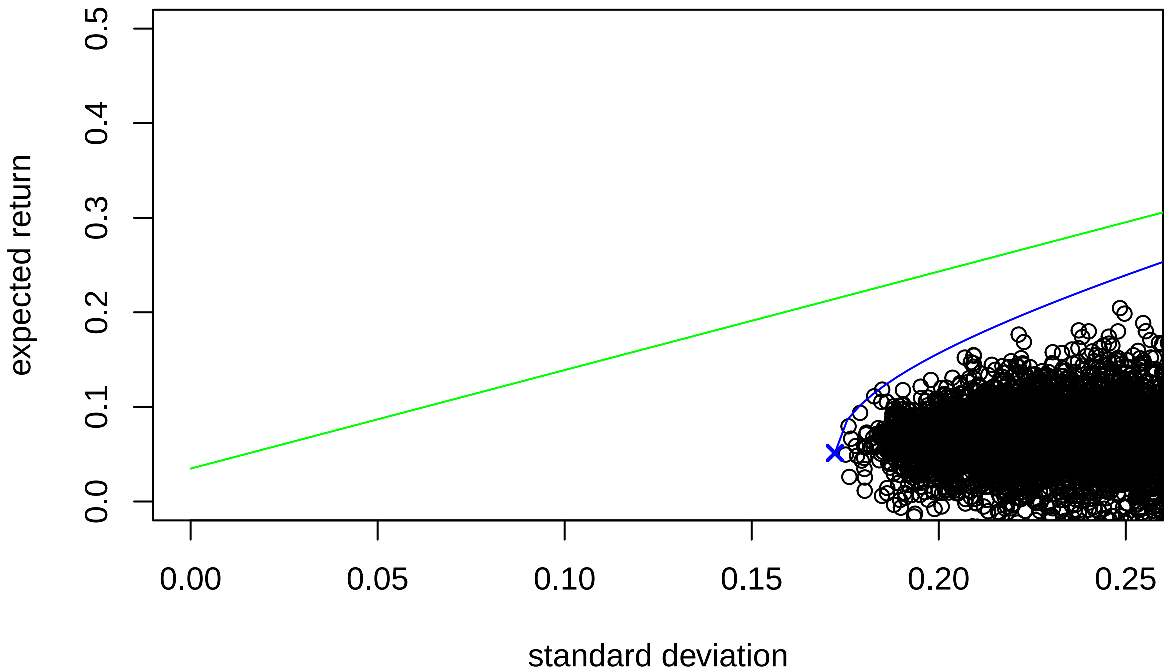
```
## [1] 0.1722548 0.1722548
```

efficiency with risk free portfolio

assume BOND to be risk free

```r
r <- anMu[7]

set.seed(1)
K <- 10000

mvRandom <- matrix(0, ncol = 2, nrow = K)
for(i in 1:nrow(mvRandom)){
    x <- rnorm(ncol(retRisky))
    x <- x/sum(x) # normalize

    mvRandom[i, 1] <- sum(x*anMuRisky)
```

```
    mvRandom[i, 2] <- sqrt((x%*%anCRisky)%*%x)
}

plot(mvRandom[,2], mvRandom[,1],
     xlab = "standard deviation", ylab = "expected return",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

k <- 100
elWithout <- matrix(0, ncol = 2, nrow = k)
elWithout[,2] <- seq(sqrt(1/c), 0.5, length.out = k)
for(i in 1:nrow(elWithout)){
    elWithout[i,1] <- a/c + sqrt(d/c*(elWithout[i,2]^2 - 1/c))
}
par(new=T)
plot(elWithout[,2], elWithout[,1], type = "l", col = "blue",
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

par(new=T)
plot(sqrt(1/c), a/c,
     col = "blue", pch = 4, lwd = 2,
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))

elWith <- matrix(0, ncol = 2, nrow = k)
elWith[,2] <- seq(0, 0.5, length.out = k)
for(i in 1:nrow(elWith)){
    elWith[i,1] <- r + elWith[i,2]*sqrt(c*r^2 - 2*a*r + b)
}
par(new=T)
plot(elWith[,2], elWith[,1], type = "l", col = "green",
     axes = FALSE, xlab = "", ylab = "",
     xlim = c(0, 0.25), ylim = c(0, 0.5))
```




```r
(xMarket <- 1/(a-c*r)*anCRisky1%*%(anMuRisky-r))
```

```
##                 [,1]
## DAX      20.1538293
## TEC      -1.3669675
## ESX50   -24.0025806
## SP5       0.4176436
## NASDAQ    5.0341532
## NIKKEI    0.7639219
```

```r
unname((b-a*r)/(a-c*r))
```

```
## [1] 1.991479
```

```r
unname((c*r^2 - 2*a*r + b)/(a-c*r)^2)
```

```
## [1] 3.52626
```

cleanup

```r
rm(a, anCRisky1, b, c, d, elWith, elWithout, i, k, K, mvRandom, r, retPlot, sPlot, x, ylim)
```

with sentiment (grid search)

IDEE: one could also look at just the previous n dates to calculate the average annual quantities.

general setup

We use several packages for the optimization.

```r
library(Rdonlp2)
```

```r
# library(Rdonlp2) ## needed for donlp2NLP
# library(fPortfolio)
# library(FRAPO) ## mrc (package of Pfaff)
# library(mco) ## mrc
```

Setup Grid. Take care that weights sum up to 1, each weight is at least *wmin* and at most *wmax*.

```r
stepsPerWeight <- 19
wmin <- 0.05
wmax <- 0.95
weights <- seq(wmin, wmax, length.out = stepsPerWeight)
grid <- expand.grid(w1 = weights, w2 = weights, w3 = weights )
grid <- grid[abs(rowSums(grid) - 1.0) < 0.0001,]
rownames(grid) <- 1:nrow(grid)

nrow(grid)
```

```
## [1] 171
```

```r
rm(stepsPerWeight, wmin, wmax, weights)
```

With this setup, we have 171 combinations of weights.

Overview of what data we use.

```r
targetRpa
```

```
## [1] 0.06
```

```r
targetVolpa
```

```
## [1] 0.04
```

```r
targetDisp
```

```
## [1] 0.58
```

```r
IneqA <- matrix(1, nrow = 1, ncol = ncol(ret)) # to take care of investments
```

dispersion direct min

We handle dispersion like return in the first place. Therefore we have the following objective functions:

1. return $\max\left(w_1 \cdot \frac{x^T\mu}{\mu_{target}}\right)$
2. volatility $\min\left(w_2 \cdot \frac{\sqrt{x^TCx}}{\sigma_{\text{target}}}\right))$
3. dispersion $\min\left(w_3 \cdot \frac{x^T\text{d}}{\text{d}_{\text{target}}}\right)$

where d denotes the annualized dispersion of each index, we name it *anDisp*. We furthermore assume that the annual dispersion equals the average dispersion.

```
anDisp <- lapply(sDisp, function(x) {colMeans(x[,-1])})
```

We will minimize the following objective function. Be aware that maximizing something equals minimizing its negative. Furthermore *anDOpt* denotes the annualized dispersion of the indizes. We divide by the target values to have the different components of the objective function comparable (in units of the corresponding target value). We denote *Opt* to be the (newly calculated) data.

```
hDispersionDirectMin <- function(x){
    y <- numeric(3)
    y[1] <- -1.0 * w[1] * drop(crossprod(x, anMuOpt)) / targetRpa
    y[2] <- w[2] * drop(sqrt(t(x) %*% anCOpt %*% x)) * sqrt(12) / targetVolpa
    y[3] <- w[3] * drop(crossprod(x, anDOpt)) / targetDisp
    return(sum(y))
}
```

constant portfolio weights over time window

First, we fix the weights x_i of each security at the beginning of (at the date before) the time window and keep them constant over time.

We store our results in the following data structure (levels of list), while having in mind that we might create a ternary plot lateron (therefore weights inside).

time window -> dispersion (sentixDataNames) -> weights of goal function -> weights of assets

We store the solution (the weights of assets), the objective value and the time needed for the computation (in seconds).

Work in parallel.

```
library(foreach)
library(parallel) # detectCores()
library(doSNOW)
```

We save with saveRDS() to be able to import and compare different results.

```
cores <- detectCores()

if(Sys.getenv("USERNAME") == "Stefan"){
    cl <- makeCluster(cores - 1)
} else if(Sys.getenv("USERNAME") == "gloggest"){
    cl <- makeCluster(cores) # use server fully
} else
    stop("Who are you???")


xDispConst <- list()

registerDoSNOW(cl)
xDispConst <- foreach(t = datesNames, .export = c(datesNames), .packages = c("Rdonlp2")) %dopar%{
    L <- list()
    timeInd <- which(datesAll == min(get(t)))-1 ## one day before start of time window

    retOpt <- ret[1:timeInd,]
    anMuOpt <- (1+colMeans(retOpt))^52-1
```

```r
    anCOpt <- cov(retOpt)*52

    for(i in names(sDisp)){
        anDOpt <- colMeans(sDisp[[i]][1:timeInd,-1])

        for(weightInd in 1:nrow(grid)){
            w <- unlist(grid[weightInd,])

            erg <- donlp2NLP(start = rep(1/ncol(retOpt), ncol(retOpt)), fun = hDispersionDirectMin,
                   par.lower = rep(0, ncol(retOpt)), ineqA = IneqA,
                   ineqA.lower = 1.0, ineqA.upper = 1.0)
            L[[i]][[paste(w, collapse = "-")]] <- list(x = erg$solution, obj = erg$objective, time = as
        }
    }
    L
}
stopCluster(cl)

names(xDispConst) <- datesNames

saveRDS(xDispConst, file = file.path(getwd(), "Optimization", paste0("EDispersionMinConstant_", Sys.get
```

TODO different portfolio weights over time window

We evaluate an optimal portfolio at each date within our time period and assume that we can redistribute our wealth at no cost.

TODO: weights of goal function weiter rein schieben

parallel programming with

```r
library(foreach)
library(doSNOW)
```

```r
# library(doParallel)
```

```r
cores <- detectCores()

if(Sys.getenv("USERNAME") == "Stefan"){
    cl <- makeCluster(cores - 1)
} else if(Sys.getenv("USERNAME") == "gloggest"){
    cl <- makeCluster(cores) # use server fully
} else
    stop("Who are you???")


E <- list()
tt <- numeric(nrow(grid)*length(sentixDataNamesReg)) # track time to evaluate code

# registerDoParallel(cl)
registerDoSNOW(cl)
E <- foreach(weightInd = 1:2, .export = sentixDataNames, .packages = c("fPortfolio", "FRAPO")) %do% {
    w <- as.numeric(grid[weightInd,])
    weightName <- paste(w, collapse = "-") # needed later to store result
```

```
    for(strategy in sentixDataNames){
        SentData <- get(strategy)
        rownames(SentData) <- as.integer(as.Date(rownames(SentData))) # for faster comparison below ->
        erg <- matrix(NA, nrow = length(datesEvalLast)+1, ncol = numAsset) # +1 to lookup every weight
        rownames(erg) <- c("1000-01-01", paste(datesEvalLast))
        erg[1, ] <- rep(1/numAsset, numAsset)

        for(d in datesEvalLast){
            dInd <- which(datesEvalLast==d)


            dispersion <- SentData[which(rownames(SentData) == d)- 1, ] # -1 to just look at the sentim
            rdat <- ret[unique(pmax(which(rownames(ret)<=d) - 1,1)),] # from beginning to one day in pa
            muStock <- colMeans(rdat)
            SStock <- cov(rdat)

            erg[dInd+1,] <- donlp2NLP(start = erg[dInd,], obj = hDispersionDirectMin,
                          par.lower = rep(0, numAsset), ineqA = IneqA,
                          ineqA.lower = 1.0, ineqA.upper = 1.0)$solution
        }

        E[[weightName]][[strategy]] <- erg
        tt[(weightInd-1)*nrow(grid) + which(sentixDataNamesReg == strategy)] <- proc.time()[3]
    }
}
stopCluster(cl)

save(E, file = file.path(folderData, "Optimization", paste0("EDispersionMin_", Sys.getenv("USERNAME"), 
```

without sentiment (classic)

constant portfolio

We also do some classical portfolio optimization, namely

1.	tangency portfolio	fPortfolio	highest return/risk ratio on the efficient frontier (market portfolio)
2.	minimum variance	fPortfolio	portfolio with minimal risk on the efficient frontier
3.	rp	cccp	risk parity solution of long-only portfolio
4.	PGMV	FRAPO (Pfaff)	global minimum variance (via correlation)
5.	PMD	FRAPO (Pfaff)	most diversivied portfolio (long-only)
6.	ew	own	equal weight

safe results in *xClassic* in an anolous manner to above

time window -> portfolio optimizing -> weights of assets

Be aware that the portfolios work with time series and therefore some typecasting is necessary.

```
library(fPortfolio)
library(FRAPO)
```

```
xClassicConst <- list()

# convert rownames back to date format (character!)
t <- rownames(ret)
```

```r
class(t) <- "Date"
rdatTimeSource <- timeSeries(ret, charvec = as.character(t))

# equal weights to start with (maybe)
ew <- rep(1/ncol(ret), ncol(ret))

for(t in datesNames){
    timeInd <- datesAll[which(datesAll == min(get(t)))-1] ## one day before start of time window

    rdatTime <- window(rdatTimeSource, start = start(rdatTimeSource), end = timeInd) # note: first day

    ans <- tangencyPortfolio(rdatTime)
    xClassicConst[[t]][["tanPort"]] <- getWeights(ans)

    ans <- minvariancePortfolio(rdatTime)
    xClassicConst[[t]][["mVaPort"]] <- getWeights(ans)

    C <- cov(rdatTime)
    ans <- rp(ew, C, ew, optctrl = ctrl(trace = FALSE))
    xClassicConst[[t]][["rp"]] <- c(getx(ans))

    ans <- PGMV(rdatTime, optctrl = ctrl(trace = FALSE))
    xClassicConst[[t]][["PGMV"]] <- Weights(ans) / 100

    ans <- PMD(rdatTime, optctrl = ctrl(trace = FALSE))
    xClassicConst[[t]][["PMD"]] <- Weights(ans) / 100

    xClassicConst[[t]][["ew"]] <- ew
}
```

TODO different portfolio weights over time window

IDEA: look at portfolio-rollingPortfolios {fPortfolio}

manually rolling

```r
Wmsr <- matrix(NA, nrow = length(datesEvalLast), ncol = numAsset)
Wmdp <- Wgmv <- Werc <- Wmsr

for(d in datesEvalLast){
    dInd <- which(datesEvalLast==d)
    class(d) <- "Date"
    rdatTime <- window(rdatTimeSource, start = start(rdatTimeSource), end = d-1) # just look at period

    ans <- tangencyPortfolio(rdatTime)
    Wmsr[dInd, ] <- getWeights(ans)


    ### global minimum variance
    ans <- PGMV(rdatTime)
    Wgmv[dInd, ] <- FRAPO::Weights(ans) / 100

    ### most diversified
```

```
    ans <- PMD(rdatTime)
    Wmdp[dInd, ] <- FRAPO::Weights(ans) / 100

    ### risk parity optimization
    SStock <- cov(rdatTime)
    ans <- rp(ew, SStock, ew, optctrl = ctrl(trace = FALSE)) # maybe invisible() makes output silent
    Werc[dInd, ] <- c(getx(ans))
}

Eclassic <- list("MSR" = Wmsr, "MDP" = Wmdp, "GMV" = Wgmv, "ERC" = Werc)
```

—— TODO ——

```
ergSentixNames <- c()
i = 1
parse(text = paste0("ergSentixNames <- ", "c(ergSentixNames, \"erg", sentixDataNames[i], "\")"))
for(i in sentixDataNames){
    eval(parse(text = paste0("ergSentixNames <- ", "c(ergSentixNames, \"erg", i, "\")")))
}
```

mrc

start optimization with equal weights and then start each iteration with result of previous iteration

roughly 30 seconds per strategy and weight (on laptop stefan)

```
nrow(grid)*length(sentixDataNamesReg)*30 # Sekunden
nrow(grid)*length(sentixDataNamesReg)*30/60 # Minuten
nrow(grid)*length(sentixDataNamesReg)*30/60/60 # Stunden
```

roughly 14 seconds per strategy and weight (on laptop stefan)

```
nrow(grid)*length(sentixDataNamesReg)*14 # Sekunden
nrow(grid)*length(sentixDataNamesReg)*14/60 # Minuten
nrow(grid)*length(sentixDataNamesReg)*14/60/60 # Stunden
```

Generate a list holding all data with structure (levels of list) weights of goal function -> strategy -> dates -> weights of assets

```
sentLookback <- 20

E <- list()
tt <- numeric(nrow(grid)*length(sentixDataNamesReg)) # track time to evaluate code

for(weightInd in 1:nrow(grid)){
    w <- as.numeric(grid[weightInd,])
    weightName <- paste(w, collapse = "-") # needed later to store result

    for(strategy in sentixDataNamesReg){
        SentData <- get(strategy)
        rownames(SentData) <- as.integer(as.Date(rownames(SentData))) # for faster comparison below ->
        erg <- matrix(NA, nrow = length(datesEvalLast)+1, ncol = numAsset) # +1 to lookup every weight
        rownames(erg) <- c("1000-01-01", paste(datesEvalLast))
```

```r
        erg[1, ] <- rep(1/numAsset, numAsset)

        for(d in datesEvalLast){
            dInd <- which(datesEvalLast==d)

            SSent <- cov(SentData[(which(rownames(SentData) == d)-sentLookback):
                                  which(rownames(SentData) == d) - 1, ]) # -1 to just look in past
            rdat <- ret[unique(pmax(which(rownames(ret)<=d) - 1,1)),] # from beginning to one day in pa
            muStock <- colMeans(rdat)
            SStock <- cov(rdat)

            erg[dInd+1,] <- donlp2NLP(start = erg[dInd,], obj = hWeighted,
                         par.lower = rep(0, numAsset), ineqA = IneqA,
                         ineqA.lower = 1.0, ineqA.upper = 1.0)$solution
        }

        E[[weightName]][[strategy]] <- erg
        tt[(weightInd-1)*nrow(grid) + which(sentixDataNamesReg == strategy)] <- proc.time()[3]
    }
}
save(E, file = file.path(folderData, "Optimization", paste0("Eserver_", format(Sys.time(), "%Y-%m-%d---
```

Visualization

One Dispersion, different weights

We visualize the different portfolio returns of each time window of each dispersion in a histogram.

The results can (also) be found in "\IR-Phase FIM-Statistik\R-Research Project Statistics\Plot Optimization\Dispersion Const".

on its own

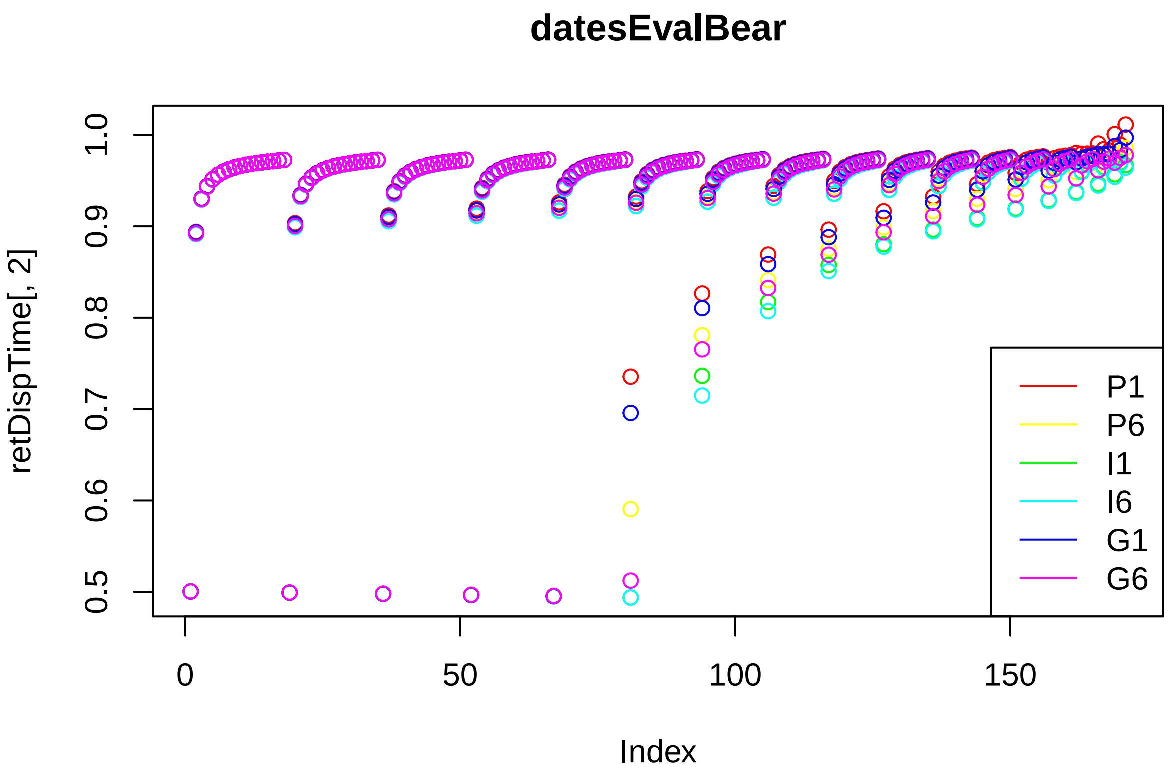
not so interesting, nicer below

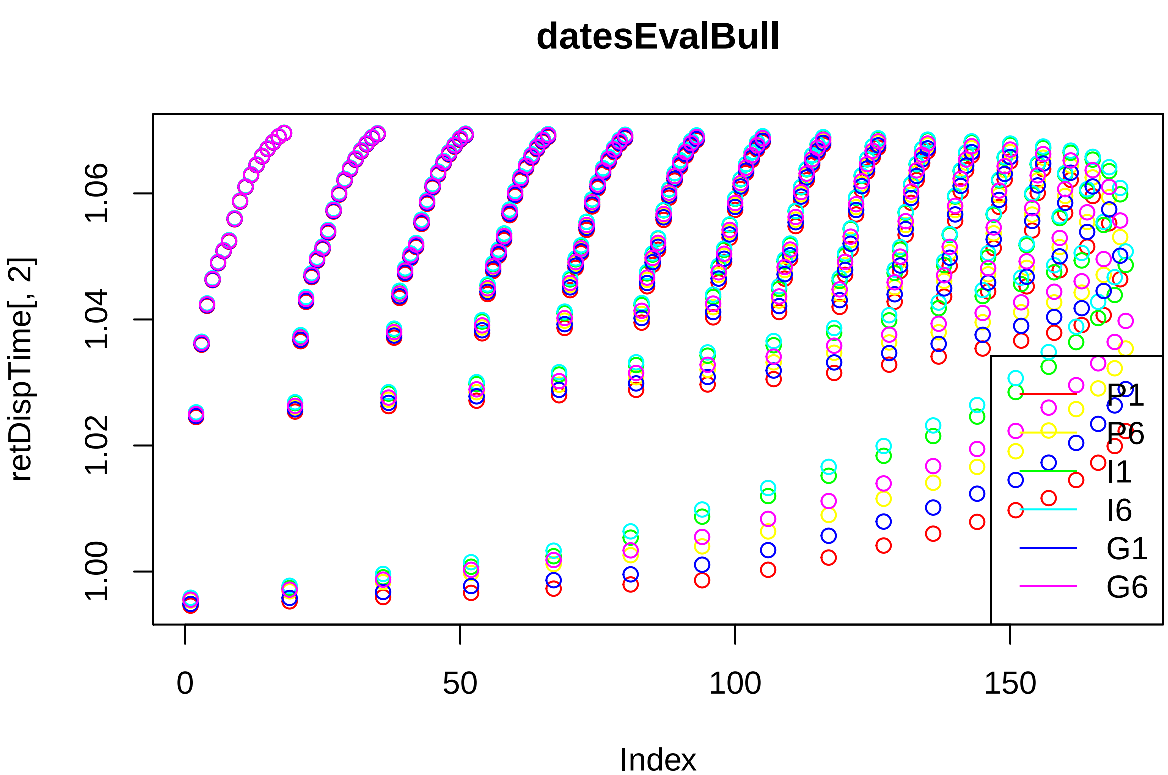
```
for(d in datesNames){
    retOverTime <- apply(1+ret[get(d),], 2, prod)

    for(i in names(xDispConst[[d]])){
        retDispTime <- numeric(length(xDispConst[[d]][[i]]))
        names(retDispTime) <- names(xDispConst[[d]][[i]])
        for(j in 1:length(retDispTime)){
            retDispTime[j] <- crossprod(xDispConst[[d]][[i]][[j]]$x, retOverTime)
        }

        t <- paste(d, i, sep = " - ")
        pdf(file.path(getwd(), "Plot Optimization", "Dispersion Const", paste0(t, ".pdf")), width = 10,
        plot(retDispTime, main = t)
        dev.off()
    }
}
```

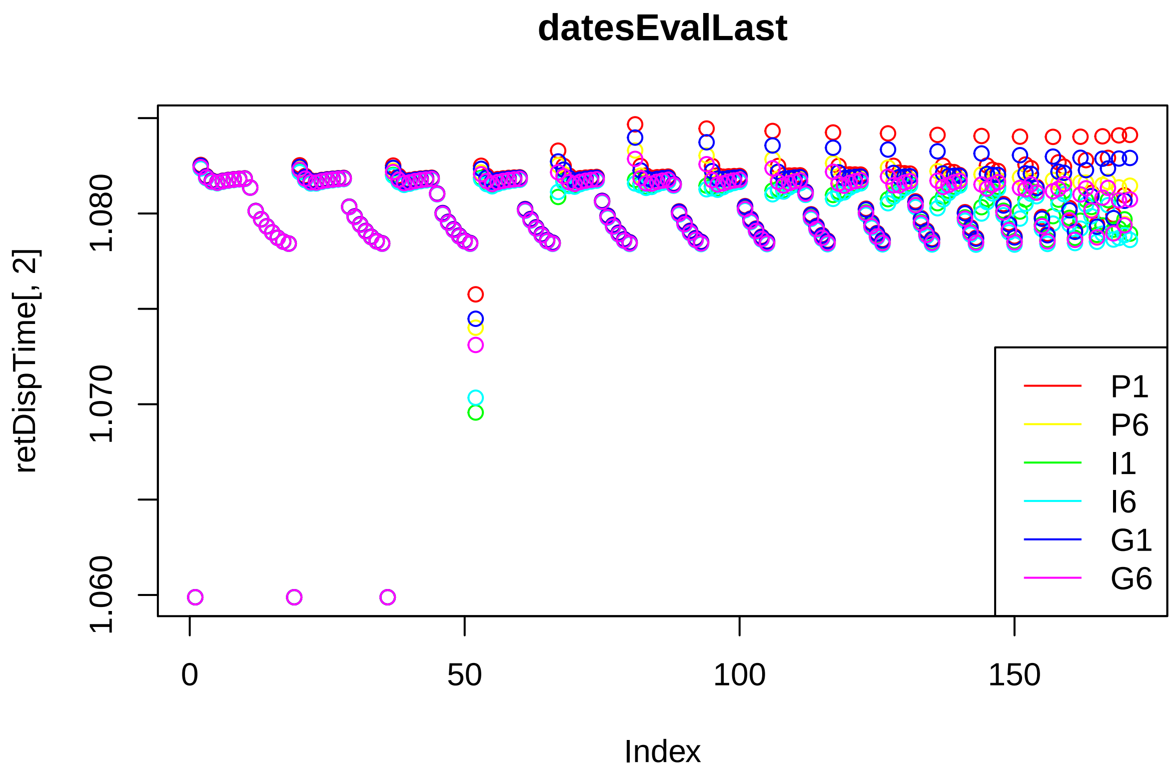
together (all different dispersions)

```
for(d in datesNames){
    cols <- rainbow(length(xDispConst[[d]]))
    retOverTime <- apply(1+ret[get(d),], 2, prod)
    retDispTime <- data.frame(w = names(xDispConst[[d]][[1]]))

    for(i in names(xDispConst[[d]])){
        for(j in 1:nrow(retDispTime)){
            retDispTime[j,i] <- crossprod(xDispConst[[d]][[i]][[j]]$x, retOverTime)
        }
    }

    ylim = c(min(retDispTime[,-1]), max(retDispTime[,-1]))
    plot(retDispTime[,2], ylim = ylim, col = cols[1], main = d)
    for(i in 3:ncol(retDispTime)){
        par(new=T)
        plot(retDispTime[,i], ylim = ylim, axes = F, xlab = "", ylab = "", col = cols[i-1])
    }
    legend("bottomright", legend = names(xDispConst[[d]]), col = cols, lty = 1)

    pdf(file.path(getwd(), "Plot Optimization", "Dispersion Const", paste0("0", d, ".pdf")), width = 10
```

```
    plot(retDispTime[,2], ylim = ylim, col = cols[1], main = d)
    for(i in 3:ncol(retDispTime)){
        par(new=T)
        plot(retDispTime[,i], ylim = ylim, axes = F, xlab = "", ylab = "", col = cols[i-1])
    }
    legend("bottomright", legend = names(xDispConst[[d]]), col = cols, lty = 1)
    dev.off()
}
```


datesEvalBear
retDispTime[, 2]
0.5
0.6
0.7
0.8
0.9
1.0
0
50
100
150
Index
P1
P6
I1
I6
G1
G6


datesEvalBull
retDispTime[, 2]
1.00
1.02
1.04
1.06
0
50
100
150
Index
P1
P6
I1
I6
G1
G6



Classic Optimization

Constant weights over window

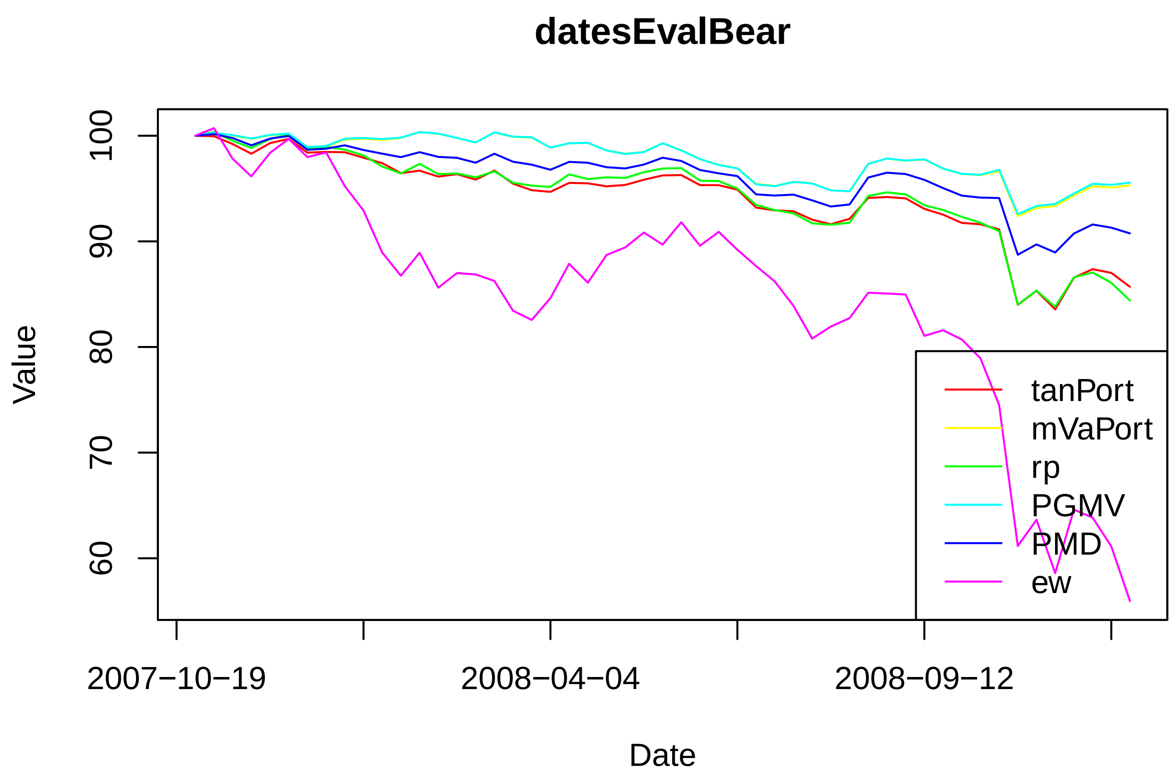
We want to visualize the evolvement of a portfolio over each time window.

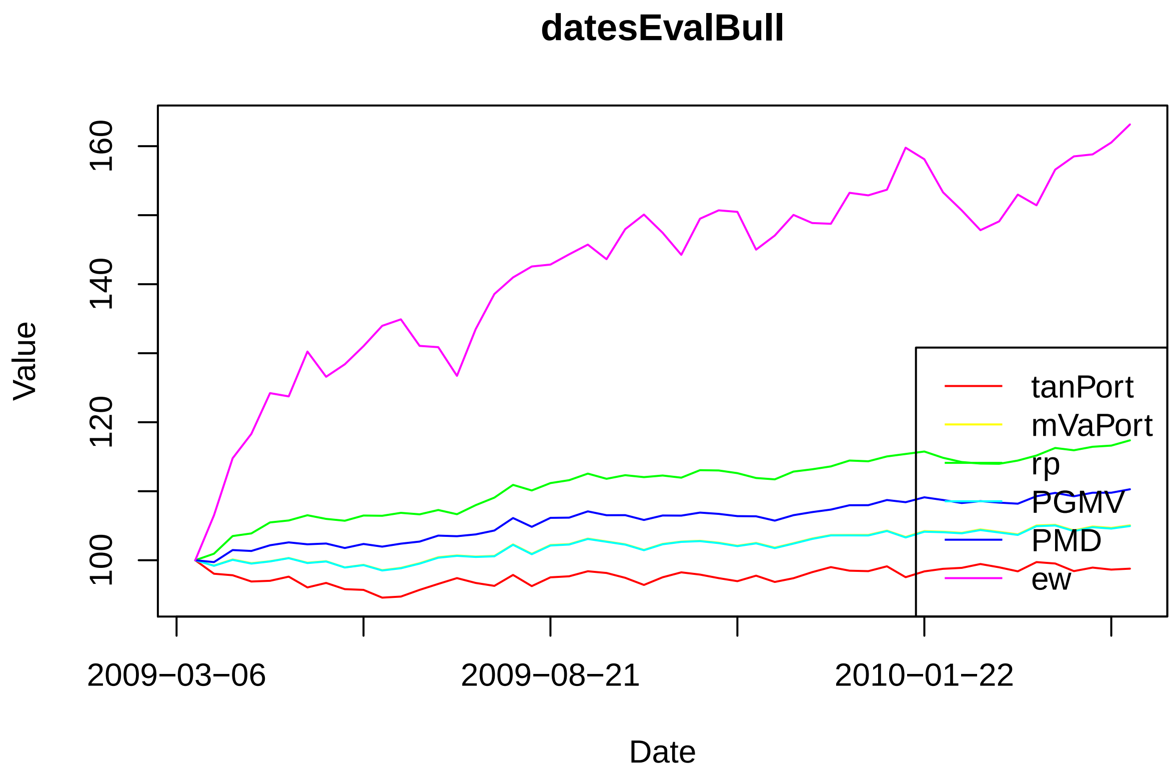
Be aware of the index shifting: retPlot[j-1, i] take wealth of previous day retOverTime[j-1,] take return of today (j is one step ahead)

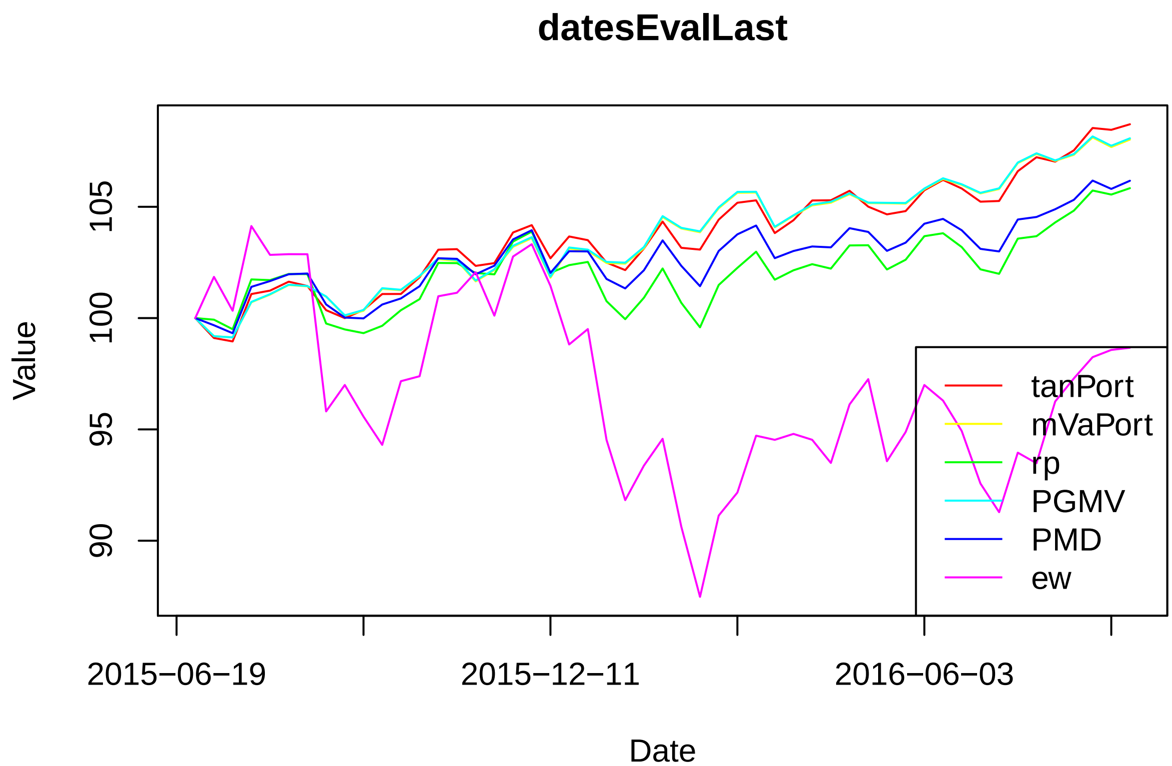
Remove numbering of x-axis by *xaxt='n'*.

```
for(d in datesNames){
    cols <- rainbow(length(xClassicConst[[d]]))
    retOverTime <- 1+ret[get(d),]
    retPlotDates <- get(d)
    retPlotDates <- c(datesAll[which(datesAll==min(retPlotDates))-1], retPlotDates)
    retPlot <- data.frame(Datum = retPlotDates)

    for(i in names(xClassicConst[[d]])){
        retPlot[1,i] <- 100
        for(j in 2:nrow(retPlot)){
            retPlot[j,i] <- retPlot[j-1,i]*crossprod(xClassicConst[[d]][[i]], retOverTime[j-1,])
        }
    }

    ylim = c(min(retPlot[,-1]), max(retPlot[,-1]))
```

```
    plot(retPlot[,2], type = "l", ylim = ylim, col = cols[1], main = d, xlab = "Date", ylab = "Value", :
    for(i in 3:ncol(retPlot)){
        par(new=T)
        plot(retPlot[,i], type = "l", ylim = ylim, axes = F, xlab = "", ylab = "", col = cols[i-1])
    }
    axis(1, at = c(0, 10, 20, 30, 40, 50), labels = retPlot[c(0, 10, 20, 30, 40, 50)+1,1])
    legend("bottomright", legend = names(xClassicConst[[d]]), col = cols, lty = 1)

    pdf(file.path(getwd(), "Plot Optimization", "Classical Const", paste0(d, ".pdf")), width = 10, heig
    plot(retPlot[,2], type = "l", ylim = ylim, col = cols[1], main = d, xlab = "Date", ylab = "Value", :
    for(i in 3:ncol(retPlot)){
        par(new=T)
        plot(retPlot[,i], type = "l", ylim = ylim, axes = F, xlab = "", ylab = "", col = cols[i-1])
    }
    axis(1, at = c(0, 10, 20, 30, 40, 50), labels = retPlot[c(0, 10, 20, 30, 40, 50)+1,1])
    legend("bottomright", legend = names(xClassicConst[[d]]), col = cols, lty = 1)
    dev.off()
}
```




datesEvalBull
Value
160
140
120
100
2009−03−06
2009−08−21
2010−01−22
Date
tanPort
mVaPort
rp
PGMV
PMD
ew


datesEvalLast
Value
105
100
95
90
2015−06−19
2015−12−11
2016−06−03
Date
tanPort
mVaPort
rp
PGMV
PMD
ew