Optimization

Stefan Glogger

August 2017

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
```

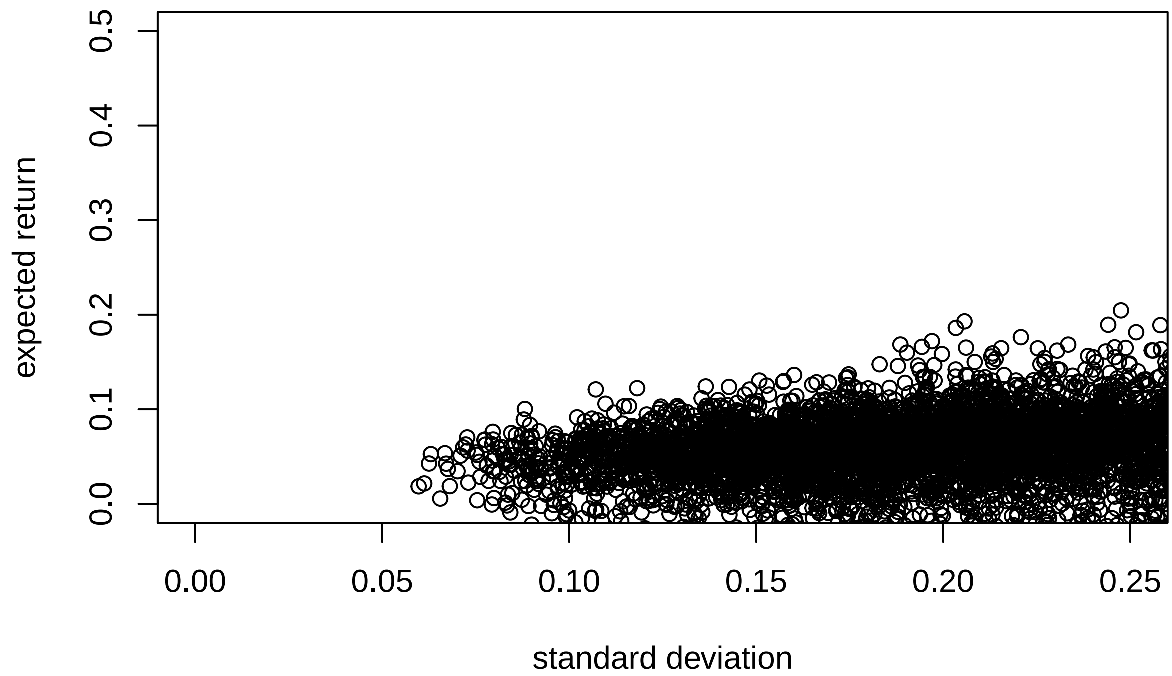
mean variance diagram

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)

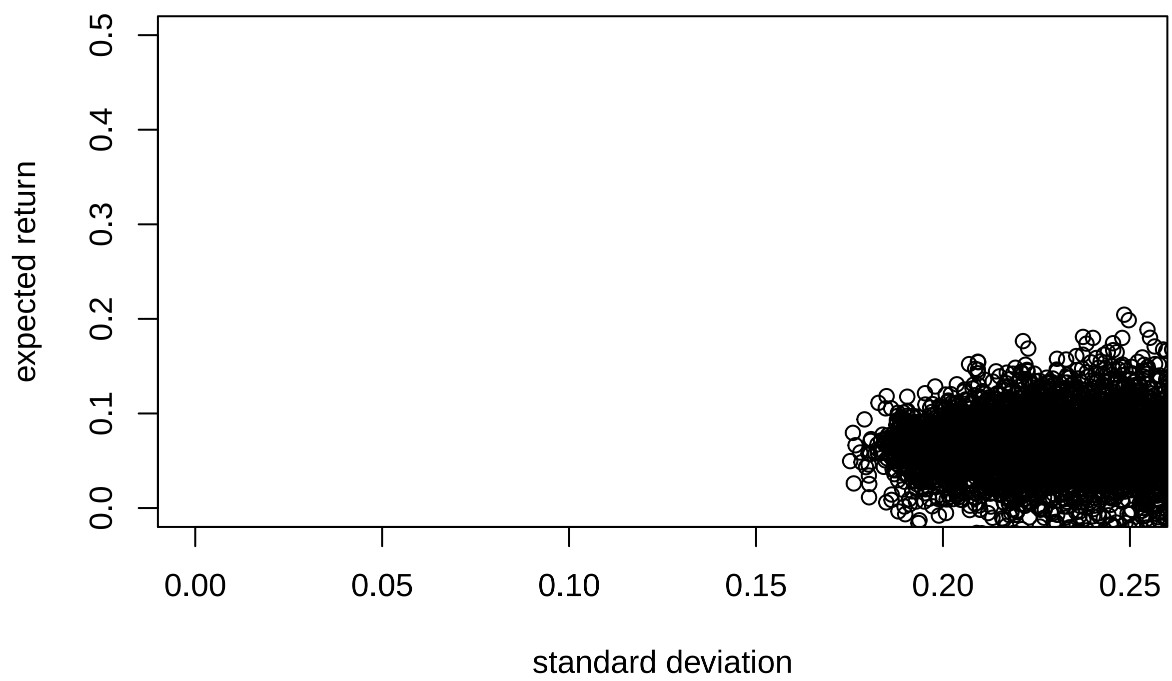
```
retRisky <- ret[,-7]
colnames(retRisky)
```

```
## [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
```

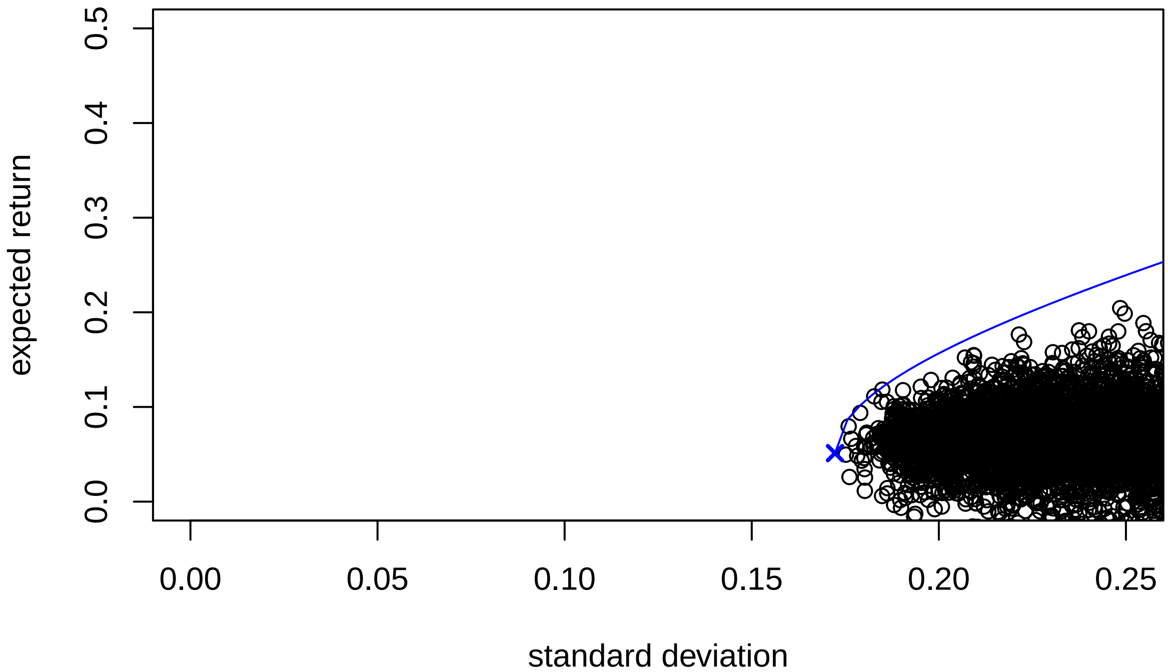
```
anCRisky1 <- solve(anCRisky)
anCRisky %*% anCRisky1
```

```
##                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))
```




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

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

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

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

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

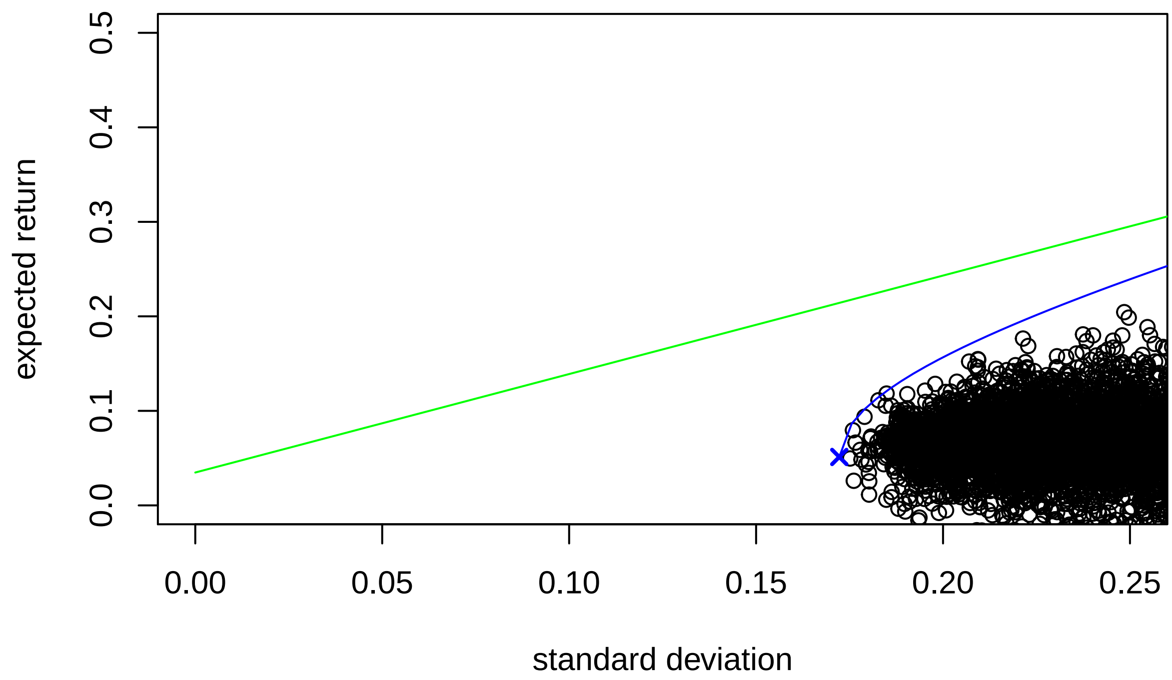
```
## [1] 0.1722548 0.1722548
```

efficiency with risk free portfolio

assume BOND to be risk free

```
r <- 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)
```

```r
    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))
```




```
(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
```

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

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

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

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

cleanup

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

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)
```

```
## Loading required package: iterators

## Loading required package: snow

##
## Attaching package: 'snow'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, clusterSplit, makeCluster,
##     parApply, parCapply, parLapply, parRapply, parSapply,
##     splitIndices, stopCluster
```

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
```

```r
    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
    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.

```r
library(fPortfolio)
```

```
## Loading required package: timeDate
## Loading required package: timeSeries
## Loading required package: fBasics
##
## Rmetrics Package fBasics
## Analysing Markets and calculating Basic Statistics
## Copyright (C) 2005-2014 Rmetrics Association Zurich
## Educational Software for Financial Engineering and Computational Science
## Rmetrics is free software and comes with ABSOLUTELY NO WARRANTY.
## https://www.rmetrics.org --- Mail to: info@rmetrics.org
## Loading required package: fAssets
##
## Rmetrics Package fAssets
## Analysing and Modeling Financial Assets
## Copyright (C) 2005-2014 Rmetrics Association Zurich
## Educational Software for Financial Engineering and Computational Science
## Rmetrics is free software and comes with ABSOLUTELY NO WARRANTY.
## https://www.rmetrics.org --- Mail to: info@rmetrics.org
##
## Rmetrics Package fPortfolio
## Portfolio Optimization
## Copyright (C) 2005-2014 Rmetrics Association Zurich
## Educational Software for Financial Engineering and Computational Science
## Rmetrics is free software and comes with ABSOLUTELY NO WARRANTY.
## https://www.rmetrics.org --- Mail to: info@rmetrics.org
##
## Attaching package: 'fPortfolio'
## The following object is masked from 'package:Rdonlp2':
##
##     donlp2NLP
```

```r
library(FRAPO)
```

```
## Loading required package: cccp
## Loading required package: Rglpk
```

```
## Loading required package: slam
```

```
## Using the GLPK callable library version 4.47
```

```
## Financial Risk Modelling and Portfolio Optimisation with R (version 0.4-1)
```

```
xClassicConst <- list()

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

```
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))
        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---%
```