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
```

We furthermore exclude riskless asset (assume BUND to be risk free)

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

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

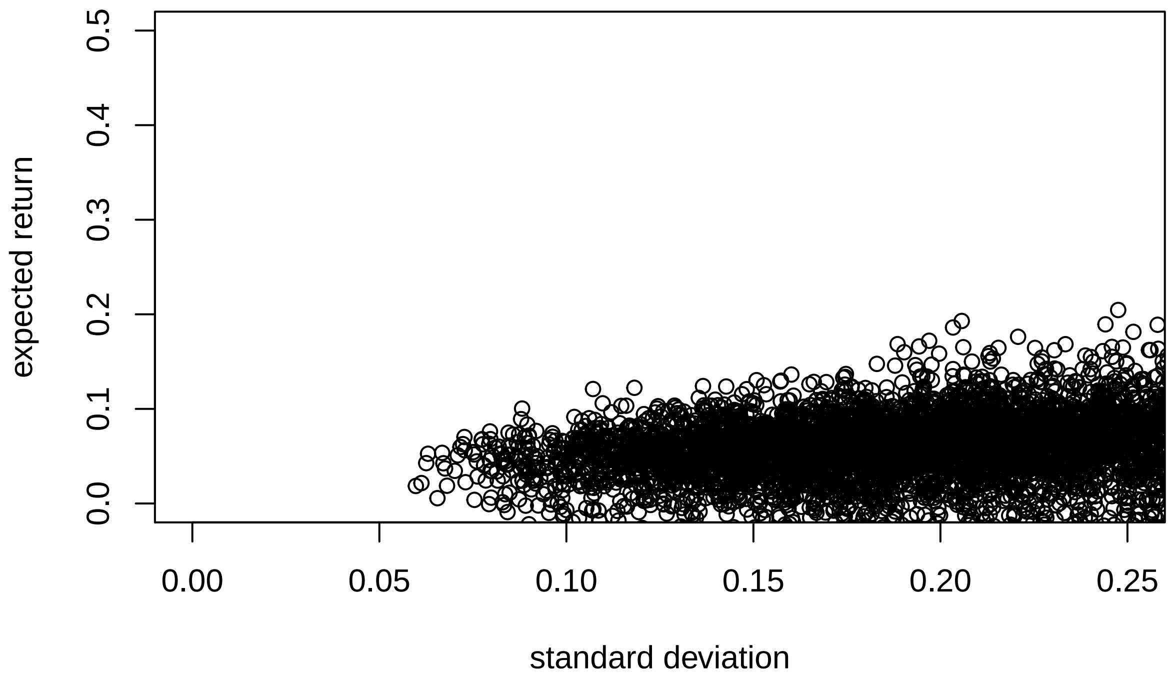
```
muRisky <- colMeans(retRisky)
CRisky <- cov(retRisky)

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

mean variance diagram

We plot K random portfolios.

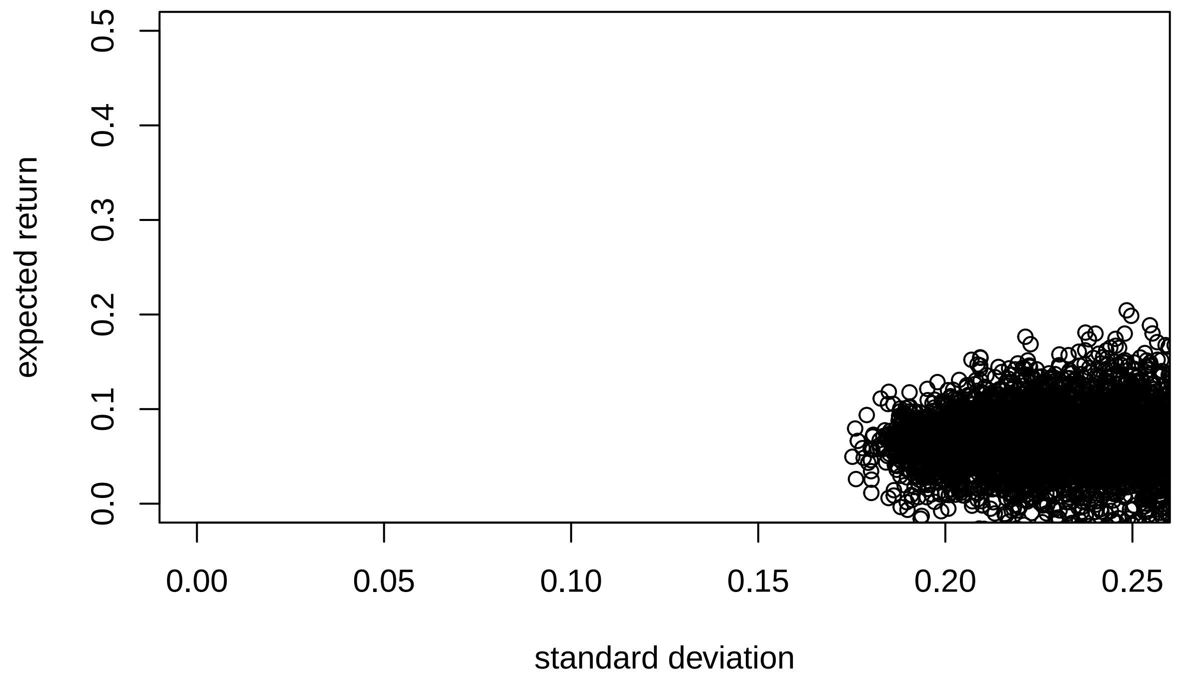
with riskfree asset

```r
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))
```



without risk free asset

```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))
```



efficiency

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)

```r
det(anC)
```

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

```r
det(anCRisky)
```

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

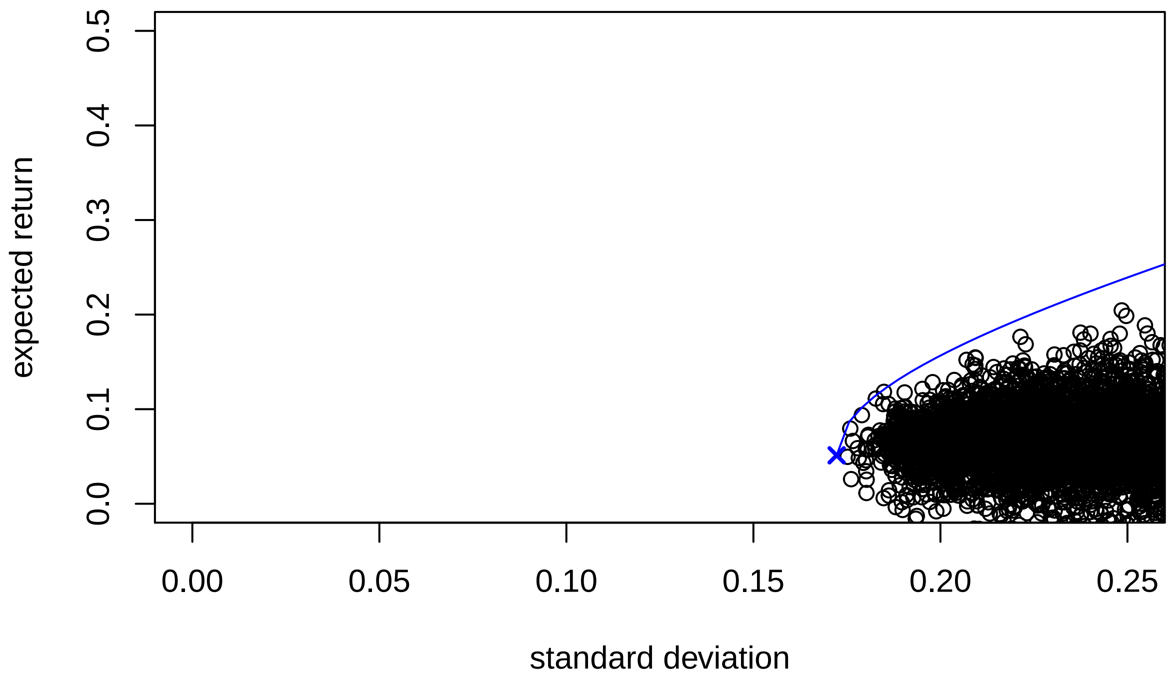
without risk free asset

```r
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
```

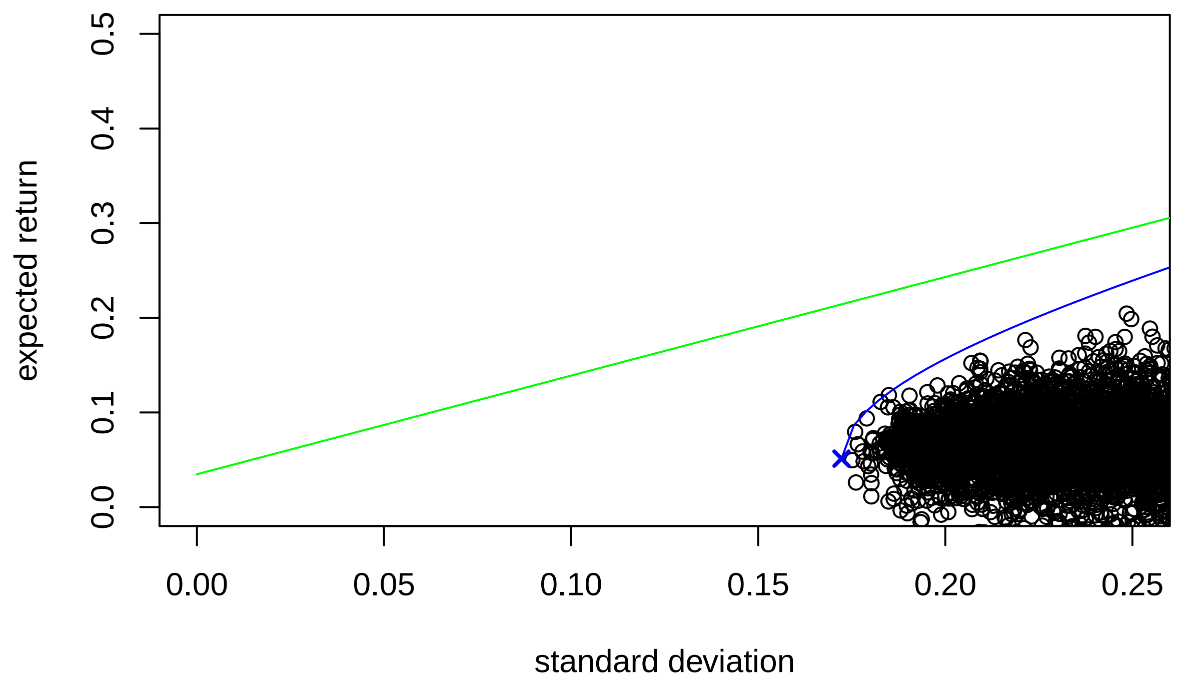
```
c(sqrt(1/c), sqrt( (xMVPwithoutRF%*%anCRisky)) %*% xMVPwithoutRF)
```

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

with risk free asset

assume BUND 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)
    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, x)
rm(anC, anCRisky, anRet, anRetRisky, CRisky, xMarket, anMuRisky, muRisky, retRisky, xMVPwithoutRF)
```

with sentiment

find optimal weights for goal function (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)
```

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
# Return
targetRpa <- 0.06 ## targeted return of 6 % p.a.

# Volatility
targetVolpa <- 0.04 ## % p.a.

# Dispersion
targetDisp <- 0.58 ## found as it looks promising and reachable in the analysis



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 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

NOTE: We keep structure, as we might change lateron the test window to come up with different weights for the different time periods (one test window to get weights for bear market, another test window to get weights for bull market, . . .) NOTE: Then also adopt for dopar

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
    stop("Who are you???")
```

```
xDispConstTest <- list()

registerDoSNOW(cl)
xDispConstTest <- foreach(t = datesTestNames, .export = c(datesTestNames), .packages = c("Rdonlp2")) %do
    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(sentixGroup in names(sDisp)){
        anDOpt <- colMeans(sDisp[[sentixGroup]][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[[sentixGroup]][[paste(w, collapse = "-")]] <- list(x = erg$solution, obj = erg$objective,
                                                             time = as.numeric(erg$elapsed))
        }
    }
    L
}
stopCluster(cl)

names(xDispConstTest) <- datesTestNames

saveRDS(xDispConstTest, file = file.path(getwd(), "Optimization", paste0("EDispersionMinConstantTest_",
```

We now add the returns and the variance over the test time window to lateron calculate the Sharpe Ratio and determine the best weights for each sentiment group.

The function takes the calculated weighted of the assets as inputs and outputs (in the same data structure) the portfolio weights, its return and its variance over the test time window.

```
xDispConstTest <- readRDS(file.path(getwd(), "Optimization", "EDispersionMinConstantTest_gloggest2017-0
```

```
calcEvalTestConst <- function(dat){
    res <- list()
    for(t in names(dat)){
        retTest <- ret[get(t),]
        muTest <- apply((1+retTest), 2, function(x) {prod(x)-1}) # total return (over whole period)
        sigmaTest <- cov(ret) # variance (over whole period)
        rf <- muTest["BUND"]

        res[[t]] <- lapply(dat[[t]], function(x) {
            lapply(x, function(y){
                list(r <- (crossprod(y$x, muTest)-rf), sd <- sqrt(y$x %*% sigmaTest %*% y$x),
                     sr = r/sd, fweight = y$obj)
            })
        })
    }
```

```r
    return(res)
}
temp <- calcEvalTestConst(xDispConstTest)
```

PROBLEM: from *t=1* to *t=50*, we have negative returns of the stocks, therefore, we invest fully in BUND, if we put enough weight on return. This is not, what we want => go directly to different portfolio weights over (test) time window

```r
rm(temp, cl, calcEvalTestConst, xDispConstTest)
```

```
## Warning in rm(temp, cl, calcEvalTestConst, xDispConstTest): Objekt 'cl'
## nicht gefunden
```

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.

We use a moving time window of *k* dates before the actual date to determine mean and variance and therefore to determine the portfolio. Furthermore, we just use the actual dispersion.

We move the parallelization further inside to be sure that we make use of parallelization (might just have one test window).

End result has the following structure: time window -> dispersion (sentixDataNames) -> weights of goal function -> dates in time window -> weights of assets

Last weight for penultimate date of time window (hold until last date).

determine portfolio in test time window (for each gridpoint)

```r
k <- 50

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???")


xDispVarTest <- list()

for(t in datesTestNames){

    xDispVarTest[[t]] <- foreach(i = names(sDisp), .export = c(datesTestNames), .packages = c("Rdonlp2"
        L <- list()

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

            mat <- matrix(NA, nrow = (length(get(t))-1), ncol = ncol(ret))
            colnames(mat) <- colnames(ret)
            rownames(mat) <- get(t)[1:(length(get(t))-1)]
            obj <- numeric(length(get(t))-1)
            tim <- numeric(length(get(t))-1)

            # first separate to then use the previous solution as starting point for next solution
            ### -------------------
            j <- 1
            tInd <- which(datesAll == get(t)[j])
            retOpt <- ret[(tInd-k+1):tInd,]
            anMuOpt <- (1+colMeans(retOpt))^52-1
            anCOpt <- cov(retOpt)*52
            anDOpt <-  as.numeric(sDisp[[i]][tInd,-1])

            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)
            mat[1,] <- erg$solution
            obj[1] <- erg$objective
            tim[1] <- as.numeric(erg$elapsed)
            ### -------------------

            for(j in 2:(length(get(t))-1)){
                tInd <- which(datesAll == get(t)[j])
                retOpt <- ret[(tInd-k+1):tInd,]
                anMuOpt <- (1+colMeans(retOpt))^52-1
                anCOpt <- cov(retOpt)*52
                anDOpt <- as.numeric(sDisp[[i]][tInd,-1])

                erg <- donlp2NLP(start = mat[j-1,], fun = hDispersionDirectMin,
                                 par.lower = rep(0, ncol(retOpt)), ineqA = IneqA,
                                 ineqA.lower = 1.0, ineqA.upper = 1.0)
                mat[j,] <- erg$solution
                obj[j] <- erg$objective
                tim[j] <- as.numeric(erg$elapsed)
            }

            L[[paste(w, collapse = "-")]] <- list(x = mat, obj = obj, time = tim)
            print(weightInd/nrow(grid))
        }
        L
    }
    names(xDispVarTest[[t]]) <- names(sDisp)
}

saveRDS(xDispVarTest, file = file.path(getwd(), "Optimization", paste0("EDispersionMinVaryingTest_", Sy

stopCluster(cl)
```

```
xDispVarTest <- readRDS(file.path(getwd(), "Optimization", "EDispersionMinVaryingTest_gloggest2017-08-2
```

determine optimal goal weights (optimal grid point)

We have the portfolio weights for each date (start) in "dat$datesTest$P1$‘0.9-0.05-0.05‘$x". We hold this for one period, therefore we calculate the portfolio return at each time step.

The datastructure is the following

dat -> datesTest -> P1 -> 0.9-0.05-0.05 -> x

We want the datastrucute

dat -> datesTest -> P1 -> 0.9-0.05-0.05 -> r, sd, sr, fweight

with

r: return (overall) sd: standard deviation (overall) sr: sharpe ratio fweight: mean of goal function

For the calculation of r, we procede as: $\mu = \mathrm{E}[\mathrm{R}] = \sum_{t=1}^{T} \mathrm{R}_t$ and for sd: $\mathrm{sd}(\mathrm{R}) = \sqrt{\mathrm{Var}(\mathrm{R})}$

In *ret* we have how much return was done at the current date (row), so we have to make a one period shift (hold portfolio up to next period and the return we make is given as the next stored return).

```r
calcTestVar <- function(dat){
    res <- list()
    for(timeWindowName in names(dat)){
        timeWindow <- get(timeWindowName)
        retTimeWindow <- ret[timeWindow,]
        retTimeWindow <- retTimeWindow[-1,]
        colnames(retTimeWindow) <- colnames(ret)

        rf <- mean(retTimeWindow[,"BUND"])

        for(sentixGroup in names(dat[[timeWindowName]])){

            for(goalWeight in names(dat[[timeWindowName]][[sentixGroup]])){
                R <- rowSums(dat[[timeWindowName]][[sentixGroup]][[goalWeight]]$x * retTimeWindow)

                r <- mean(R)
                sd <- sd(R)

                anR <- (1+r)^52-1
                anSd <- sqrt((sd^2)*52)

                fweight = mean(dat[[timeWindowName]][[sentixGroup]][[goalWeight]]$obj)

                res[[timeWindowName]][[sentixGroup]][[goalWeight]] <- list(r = r, sd = sd, sr = r/sd,
                                                                           anR = anR, anSd = anSd, a
                                                                           fweight = fweight)
            }
        }
    }
    return(res)
}
```

```r
xDispVarTestCalc <- calcTestVar(xDispVarTest)
```

now, determine optimal weight, for each *sentixGroup* and *timeWindow*, optimal meaning maximum sharpe ratio

Now, we also want to visualize the data. We use a ternary plot for this.

```r
library(ggtern)
```

```
## Loading required package: ggplot2

## --
## Consider donating at: http://ggtern.com
## Even small amounts (say $10-50) are very much appreciated!
## Remember to cite, run citation(package = 'ggtern') for further info.
## --

##
## Attaching package: 'ggtern'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, aes, annotate, calc_element, ggplot, ggplot_build,
##     ggplot_gtable, ggplotGrob, ggsave, layer_data, theme,
##     theme_bw, theme_classic, theme_dark, theme_gray, theme_light,
##     theme_linedraw, theme_minimal, theme_void
```

```
extractWeightsWithValue <- function(dat, value){
    ret <- list()
    for(timeWindowName in names(dat)){
        for(sentixGroup in names(dat[[timeWindowName]])){
            df <- data.frame(w = names(dat[[timeWindowName]][[sentixGroup]])[1], value = dat[[timeWindo
            df$w <- as.character(df$w)
            df$w1 <- as.numeric(unlist(strsplit(df$w, "-"))[1])
            df$w2 <- as.numeric(unlist(strsplit(df$w, "-"))[2])
            df$w3 <- as.numeric(unlist(strsplit(df$w, "-"))[3])

            for(weightsName in names(dat[[timeWindowName]][[sentixGroup]])[2:length(names(dat[[timeWind
                df <- rbind(df, c(weightsName, dat[[timeWindowName]][[sentixGroup]][[weightsName]][[val
            }

            ret[[timeWindowName]][[sentixGroup]] <- data.frame(w1 = as.numeric(df[,"w1"]),
                                                               w2 = as.numeric(df[,"w2"]),
                                                               w3 = as.numeric(df[,"w3"]),
                                                               value = as.numeric(df[,"value"]
        }
    }
    return(ret)
}

srWeightsAn <- extractWeightsWithValue(xDispVarTestCalc, "anSR")
```

```
terntheme <- function(){
    theme_rgbg() +
        theme(legend.position = c(0, 1),
              legend.justification = c(0, 1),
              plot.margin=unit(c(0, 2,0, 2), "cm"),
              tern.panel.background = element_rect(fill = "lightskyblue1"))
}
```

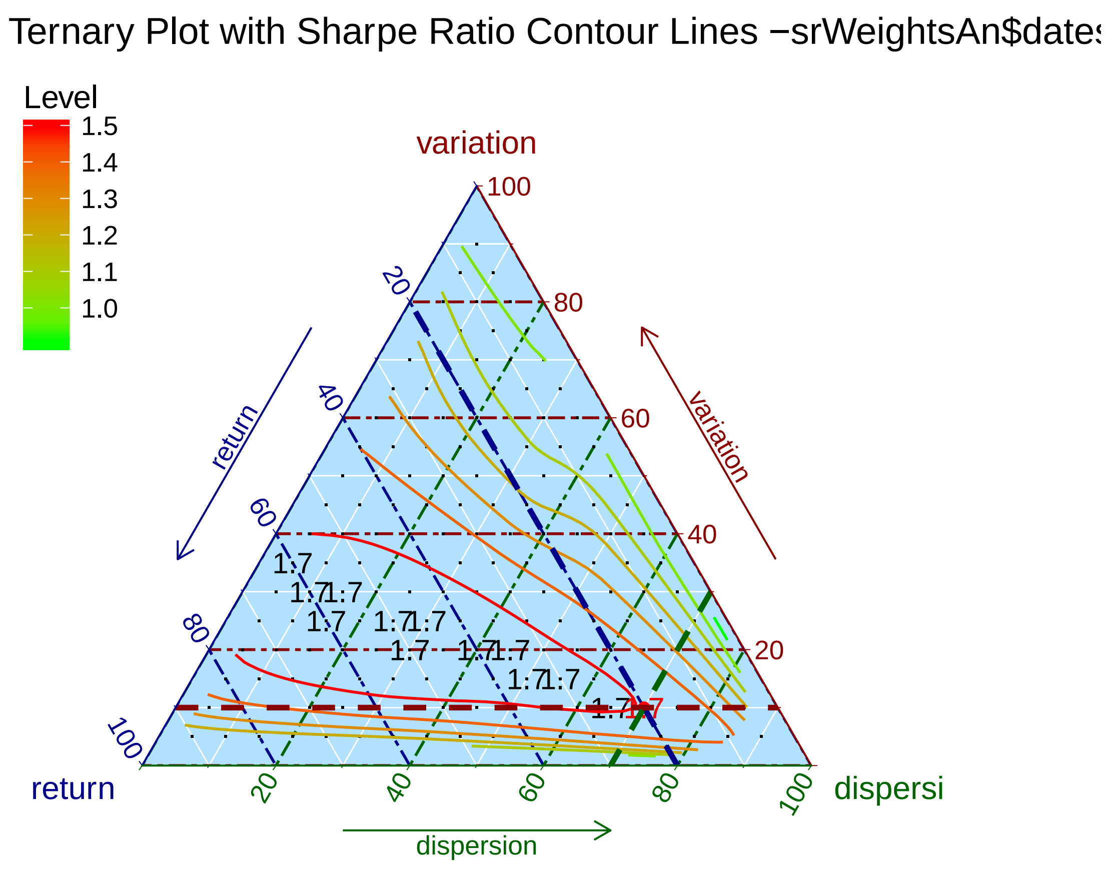
note: “Calling ‘structure(NULL, *)’ is deprecated, as NULL cannot have attributes. Consider ‘structure(list(), *)’ instead.” is just a message, not an error. it is due to strict checks in the newer R-version https://stat.ethz.ch/pipermail/r-devel/2016-December/073554.html

..level.. is calculated and then used inside the function

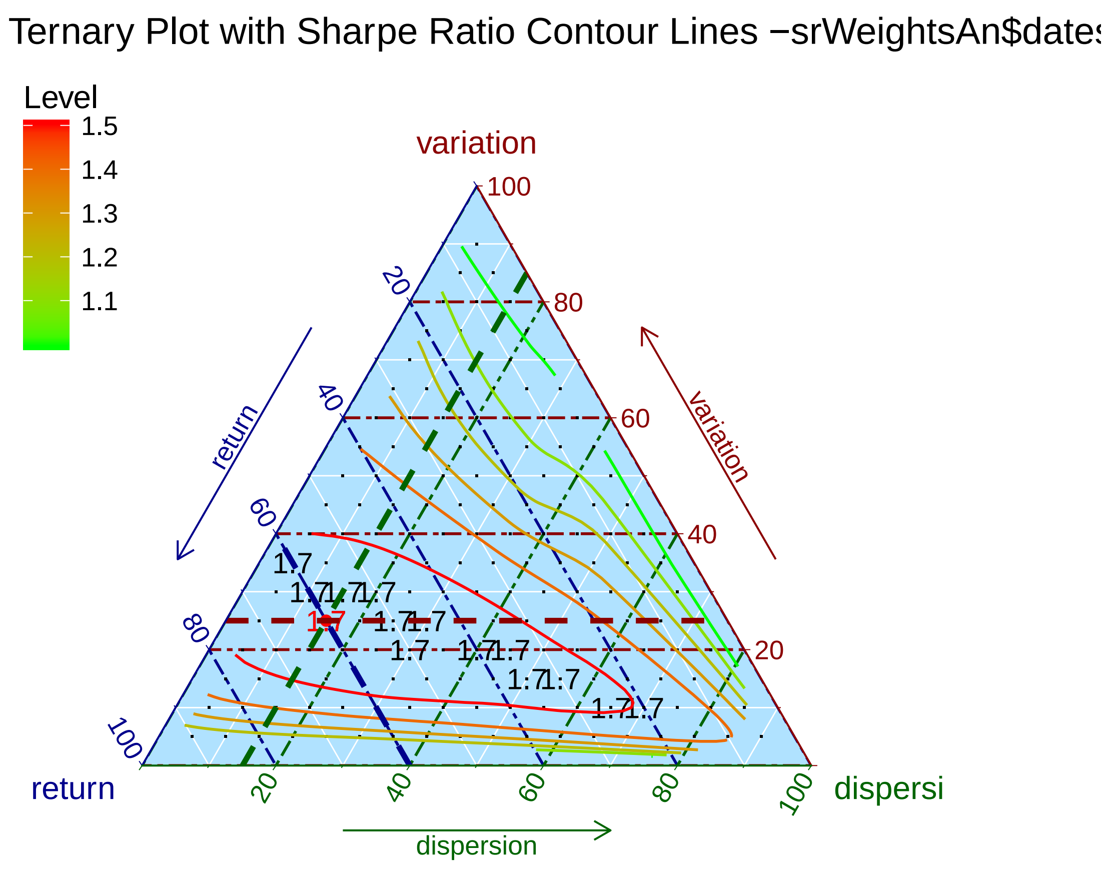
```
plotTernary <- function(dat){
    wmax <- dat[which.max(dat$value), c("w1", "w2", "w3", "value")]

    ggtern(dat, aes(x=w1, y=w2, z=w3, value=value)) +
        geom_point(shape=".")+
        geom_text(aes(x=w1, y=w2, label=round(value,1)), data = dat[dat$value>(dat[(dat$w1==wmax$w1) &
        geom_interpolate_tern(aes(value=value, color = ..level..)) +
        geom_point(aes(x=w1, y=w2), dat = wmax, color = "red") +
        geom_text(aes(x=w1, y=w2, label=round(dat[(dat$w1==wmax$w1) & (dat$w2==wmax$w2),"value"],1)), da
        terntheme() +
        Lline(Lintercept =  wmax$w1, colour = theme_rgbg()$tern.axis.line.L$colour, linetype = 2, lwd=1
        Tline(Tintercept = wmax$w2, colour = theme_rgbg()$tern.axis.line.T$colour, linetype = 2, lwd=1)
        Rline(Rintercept = wmax$w3, color = theme_rgbg()$tern.axis.line.R$colour, linetype = 2, lwd=1)
        scale_color_gradient(low = "green", high = "red") +
        labs(x = "return", y = "variation", z = "dispersion",
             title = paste0("Ternary Plot with Sharpe Ratio Contour Lines -", deparse(substitute(dat)),
             color = "Level")
}
```

```
lateximport <- c(paste0("\\subsection{Ternary}"))

for(t in names(srWeightsAn$datesTest)){
    plot(plotTernary(srWeightsAn$datesTest[[t]]))

    title <- paste0("Ternary-Weights ",t, ".pdf")
    pdf(file.path(getwd(), "Plot", title), width = 10, height = 10)
    plot(plotTernary(srWeightsAn$datesTest[[t]]))
    dev.off()

    lateximport <- c(lateximport, paste0("\\includegraphics[height=0.45\\textheight]{",title,"}\\linebr
}
```

```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

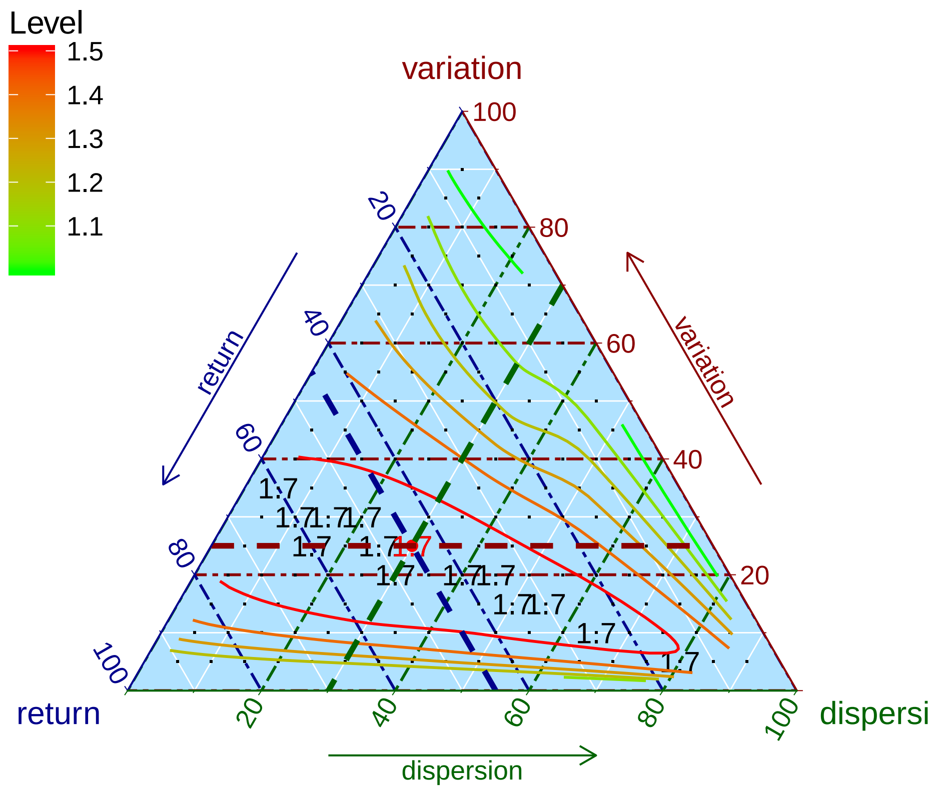


```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

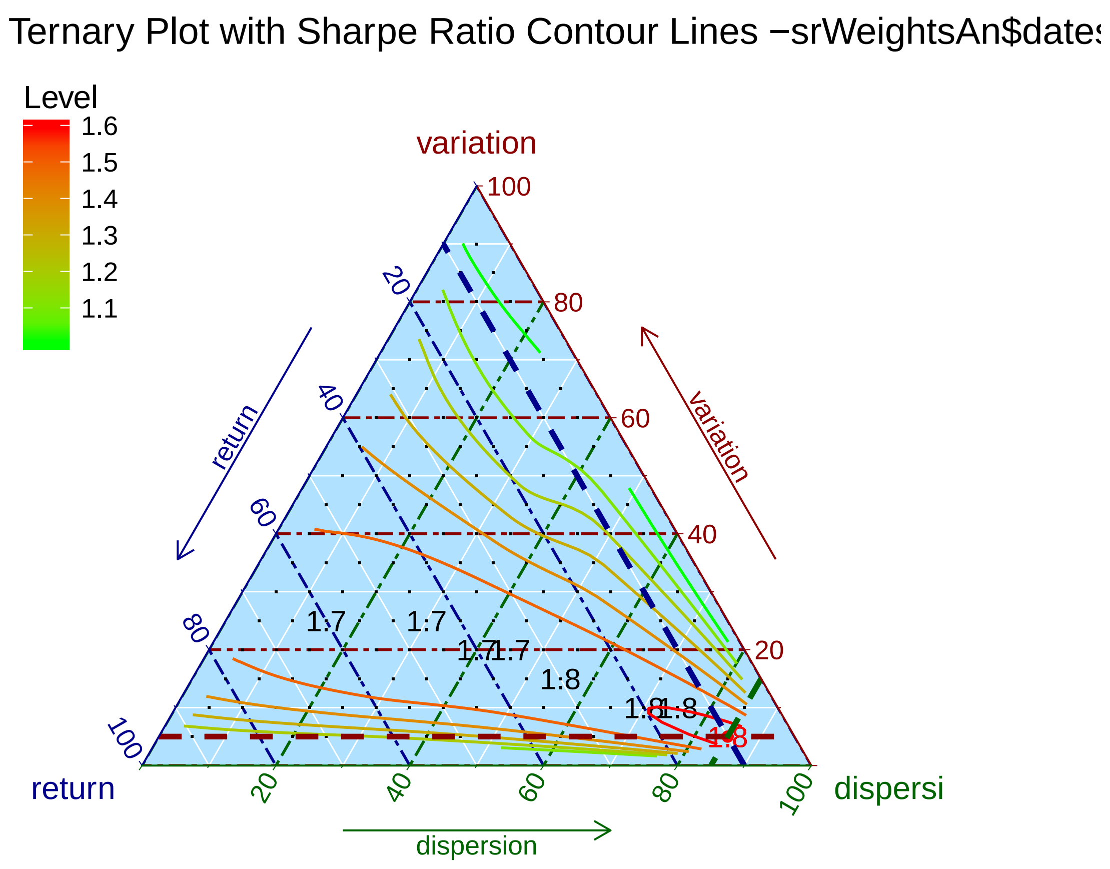


```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

Ternary Plot with Sharpe Ratio Contour Lines −srWeightsAn$dates

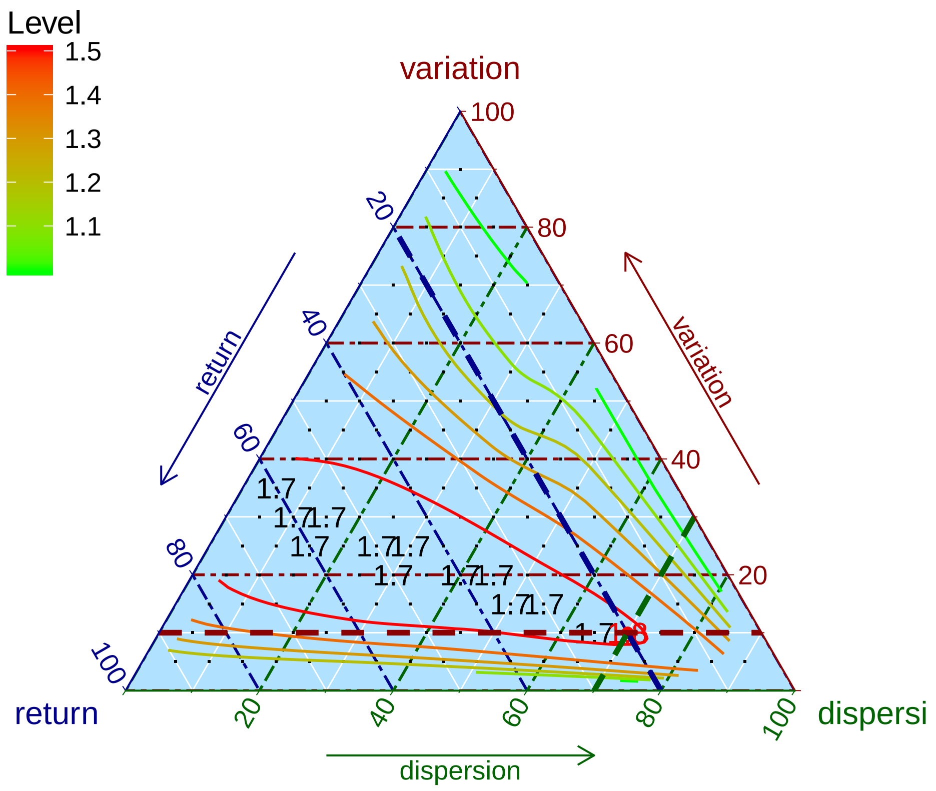


```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

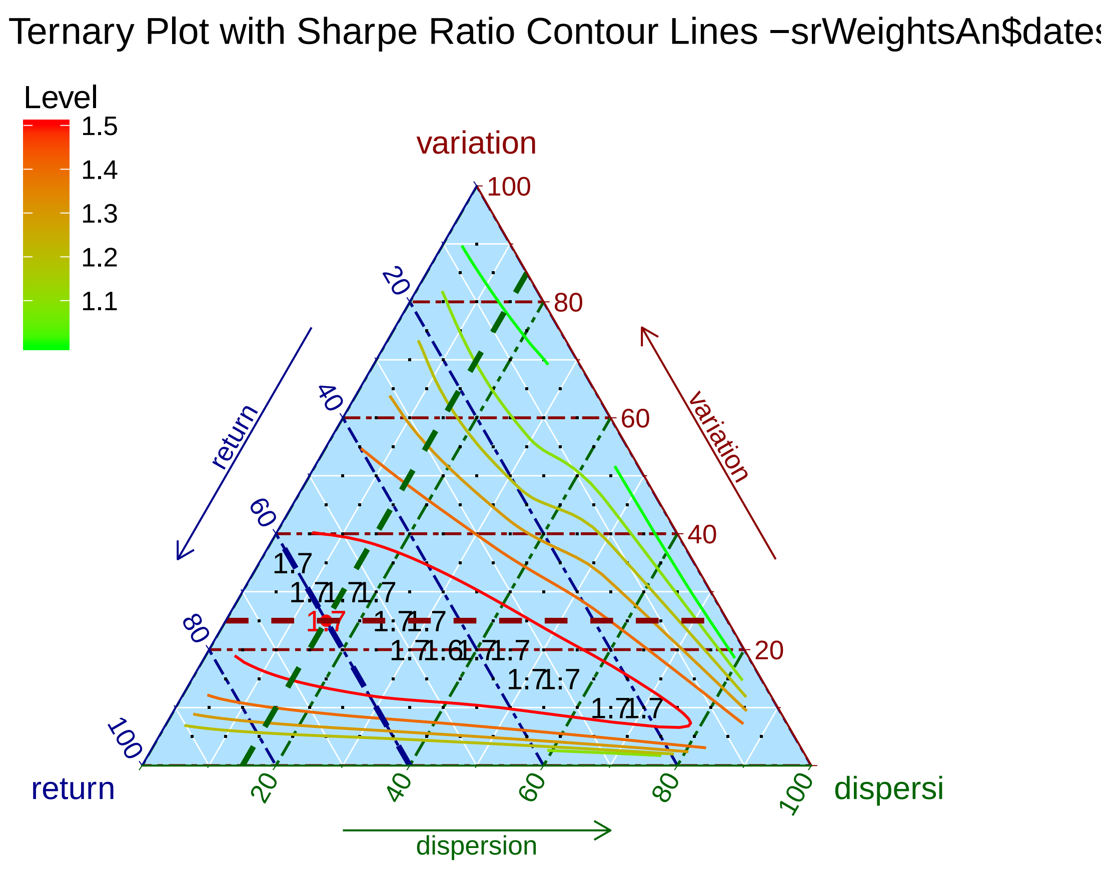


```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```

Ternary Plot with Sharpe Ratio Contour Lines −srWeightsAn$dates



```
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.

## Warning in structure(c(), class = c(class(x), class(y))): Calling 'structure(NULL, *)' is deprecated
##   Consider 'structure(list(), *)' instead.
```



```r
fileConnection <- file(file.path(getwd(), "Plot", paste0("0Ternary.txt")))
writeLines(lateximport, fileConnection)
close(fileConnection)
```

now, determine optimal weight, for each *sentixGroup* and *timeWindow*, optimal meaning maximum sharpe ratio

```r
wOptSRAn <- list()
for(timeWindowName in names(srWeightsAn)){
    for(sentixGroup in names(srWeightsAn[[timeWindowName]])){
        df <- srWeightsAn[[timeWindowName]][[sentixGroup]]

        wOptSRAn[[timeWindowName]][[sentixGroup]] <- df[which.max(df$value), c("w1", "w2", "w3")]
    }
}
wOptSRAn$datesTest
```

```
## $P1
##      w1  w2  w3
## 158 0.2 0.1 0.7
##
## $P6
##     w1   w2   w3
## 40 0.6 0.25 0.15
##
## $I1
##      w1   w2  w3
## 85 0.45 0.25 0.3
##
## $I6
##      w1   w2   w3
## 169 0.1 0.05 0.85
##
## $G1
##      w1  w2  w3
## 158 0.2 0.1 0.7
##
## $G6
##     w1   w2   w3
## 40 0.6 0.25 0.15
```

```
rm(xDispVarTest, xDispVarTestCalc)
rm(plotTernary, terntheme, fileConnection, lateximport)
```

find optimal portfolio

dispersion direct min

Now, we want to determine the portfolio weights over the time Windows. We use the data structure:

time window -> dispersion (sentixGroup) -> weights of assets

NOTE: change the weights of the test data, if different test time windows are used to get weights for different eval time weights

We use a moving time window of k dates before the actual date to determine mean and variance and therefore to determine the portfolio. Furthermore, we just use the actual dispersion.

```
k <- 50

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???")


xDispVarEval <- list()

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

    xDispVarEval[[t]] <- foreach(sentixGroup = names(sDisp), .export = c(datesEvalNames), .packages = c
        L <- list()


        w <- unlist(wOptSRAn$datesTest[[sentixGroup]])

        mat <- matrix(NA, nrow = (length(get(t))-1), ncol = ncol(ret))
        colnames(mat) <- colnames(ret)
        rownames(mat) <- get(t)[1:(length(get(t))-1)]
        obj <- numeric(length(get(t))-1)
        tim <- numeric(length(get(t))-1)

        # first separate to then use the previous solution as starting point for next solution
        ### --------------------
        j <- 1
        tInd <- which(datesAll == get(t)[j])
        retOpt <- ret[(tInd-k+1):tInd,]
        anMuOpt <- (1+colMeans(retOpt))^52-1
        anCOpt <- cov(retOpt)*52
        anDOpt <-  as.numeric(sDisp[[sentixGroup]][tInd,-1])

        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)
        mat[1,] <- erg$solution
        obj[1] <- erg$objective
        tim[1] <- as.numeric(erg$elapsed)
        ### --------------------

        for(j in 2:(length(get(t))-1)){
            tInd <- which(datesAll == get(t)[j])
            retOpt <- ret[(tInd-k+1):tInd,]
            anMuOpt <- (1+colMeans(retOpt))^52-1
            anCOpt <- cov(retOpt)*52
            anDOpt <- as.numeric(sDisp[[sentixGroup]][tInd,-1])

            erg <- donlp2NLP(start = mat[j-1,], fun = hDispersionDirectMin,
                             par.lower = rep(0, ncol(retOpt)), ineqA = IneqA,
                             ineqA.lower = 1.0, ineqA.upper = 1.0)
            mat[j,] <- erg$solution
            obj[j] <- erg$objective
            tim[j] <- as.numeric(erg$elapsed)
        }

        list(x = mat, obj = obj, time = tim)
    }
    names(xDispVarEval[[t]]) <- names(sDisp)
}

stopCluster(cl)

# names(xDispVarTest) <- datesTestNames

saveRDS(xDispVarEval, file = file.path(getwd(), "Optimization", paste0("EDispersionMinVaryingEval_", Sy
```

```
xDispVarEval <- readRDS(file.path(getwd(), "Optimization", "EDispersionMinVaryingEval_Stefan2017-08-29-
```

detach

```r
detach("package:doSNOW", unload = T)
detach("package:parallel", unload = T)
detach("package:foreach", unload = T)

detach("package:ggtern", unload = T)
```

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 *xClassicConst* 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
```

```
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 datesEvalNames){
    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
```

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

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

```r
rm(rdatTime, rdatTimeSource, t, ew, ans)
```

different portfolio weights over time window

IDEA: look at portfolio-rollingPortfolios {fPortfolio}

manually rolling

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

time window -> portfolio (classic) -> weights of assets

NOTE: change the weights of the test data, if different test time windows are used to get weights for different eval time weights

We use a moving time window of k dates before the actual date to determine mean and variance and therefore to determine the portfolio. Furthermore, we just use the actual dispersion.

```r
k <- 50

xClassicVar <- 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(timeWindowName in datesEvalNames){
    datesEvalNow <- get(timeWindowName)

    mat <- matrix(NA, nrow = (length(datesEvalNow)-1), ncol = ncol(rdatTimeSource))
    colnames(mat) <- colnames(ret)
    rownames(mat) <- datesEvalNow[1:(length(datesEvalNow)-1)]

    xClassicVar[[timeWindowName]][["tanPort"]]$x <- mat
    xClassicVar[[timeWindowName]][["mVaPort"]]$x <- mat
    xClassicVar[[timeWindowName]][["rp"]]$x <- mat
    xClassicVar[[timeWindowName]][["PGMV"]]$x <- mat
    xClassicVar[[timeWindowName]][["PMD"]]$x <- mat
    xClassicVar[[timeWindowName]][["ew"]]$x <- mat

    for(d in 1:(length(datesEvalNow)-1)){ # last date no portfolio weights

        timeEndInd <- which(datesAll == datesEvalNow[d]) ## one day before start of time window => NO, 
        timeEnd <- datesAll[timeEndInd]
        timeStart <- datesAll[timeEndInd-k+1]

        rdatTime <- timeSeries::window(rdatTimeSource, start = timeStart, end = timeEnd) # note: first

        ans <- tangencyPortfolio(rdatTime)
        xClassicVar[[timeWindowName]][["tanPort"]]$x[d,] <- getWeights(ans)

        ans <- minvariancePortfolio(rdatTime)
        xClassicVar[[timeWindowName]][["mVaPort"]]$x[d,] <- getWeights(ans)

        C <- cov(rdatTime)
        ans <- rp(ew, C, ew, optctrl = ctrl(trace = FALSE))
        xClassicVar[[timeWindowName]][["rp"]]$x[d,] <- c(getx(ans))

        ans <- PGMV(rdatTime, optctrl = ctrl(trace = FALSE))
        xClassicVar[[timeWindowName]][["PGMV"]]$x[d,] <- Weights(ans) / 100

        ans <- PMD(rdatTime, optctrl = ctrl(trace = FALSE))
        xClassicVar[[timeWindowName]][["PMD"]]$x[d,] <- Weights(ans) / 100

        xClassicVar[[timeWindowName]][["ew"]]$x[d,] <- ew
    }
}
```

```
rm(t, timeEnd, timeEndInd, timeStart, timeWindowName, k, ew, rdatTime, rdatTimeSource, ans)
```

different portfolio weights over time window, just risky assets

IDEA: look at portfolio-rollingPortfolios {fPortfolio}

manually rolling

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

time window -> portfolio (classic) -> weights of assets

NOTE: change the weights of the test data, if different test time windows are used to get weights for different eval time weights

We use a moving time window of k dates before the actual date to determine mean and variance and therefore to determine the portfolio. Furthermore, we just use the actual dispersion.

we exclude the risk free asset *BUND* of the analysis

TODO: risk free rate mit nuller in BUND

```
k <- 50

xClassicVarNoRf <- 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)-1), (ncol(ret)-1))

for(timeWindowName in datesEvalNames){
    datesEvalNow <- get(timeWindowName)

    mat <- matrix(NA, nrow = (length(datesEvalNow)-1), ncol = ncol(rdatTimeSource))
    colnames(mat) <- colnames(ret)[1:ncol(rdatTimeSource)]
    rownames(mat) <- datesEvalNow[1:(length(datesEvalNow)-1)]

    xClassicVarNoRf[[timeWindowName]][["tanPort"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["mVaPort"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["rp"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["PGMV"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["PMD"]]$x <- mat
    xClassicVarNoRf[[timeWindowName]][["ew"]]$x <- mat

    for(d in 1:(length(datesEvalNow)-1)){ # last date no portfolio weights

        timeEndInd <- which(datesAll == datesEvalNow[d]) ## one day before start of time window => NO, 
        timeEnd <- datesAll[timeEndInd]
        timeStart <- datesAll[timeEndInd-k+1]

        rdatTime <- timeSeries::window(rdatTimeSource, start = timeStart, end = timeEnd) # note: first
        rf <- mean(rdatTime[,"BUND"])
        rdatTime <- rdatTime[,setdiff(names(rdatTime), "BUND")] # reduce to all but BUND

        portfolio <- portfolioSpec()
        setRiskFreeRate(portfolio) <- rf

        ans <- tangencyPortfolio(rdatTime, spec = portfolio)
        xClassicVarNoRf[[timeWindowName]][["tanPort"]]$x[d,] <- c(getWeights(ans), 0)

        ans <- minvariancePortfolio(rdatTime, spec = portfolio)
        xClassicVarNoRf[[timeWindowName]][["mVaPort"]]$x[d,] <- c(getWeights(ans), 0)

        C <- cov(rdatTime)
        ans <- rp(ew, C, ew, optctrl = ctrl(trace = FALSE))
        xClassicVarNoRf[[timeWindowName]][["rp"]]$x[d,] <- c(getx(ans), 0)

        ans <- PGMV(rdatTime, optctrl = ctrl(trace = FALSE))
        xClassicVarNoRf[[timeWindowName]][["PGMV"]]$x[d,] <- c(Weights(ans) / 100, 0)

        ans <- PMD(rdatTime, optctrl = ctrl(trace = FALSE))
        xClassicVarNoRf[[timeWindowName]][["PMD"]]$x[d,] <- c(Weights(ans) / 100, 0)

        xClassicVarNoRf[[timeWindowName]][["ew"]]$x[d,] <- c(ew, 0)
    }
}
```

```
rm(t, timeEnd, timeEndInd, timeStart, timeWindowName, k, ew, rdatTime, rdatTimeSource, ans)
```

detach

```
detach("package:FRAPO", unload = T)
detach("package:fPortfolio", unload = T)
```

```r
detach("package:fAssets", unload = T)

unloadNamespace("fCopulae")
unloadNamespace("fMultivar")
detach("package:fBasics", unload = T) # need to unload "fCopulae" and "fMultivar" first, somehow "detac

detach("package:timeSeries", unload = T)
```