Solutions to Exercises

Portfolio Optimization: Theory and Application
Chapter 7 – Modern Portfolio Theory

Daniel P. Palomar (2025). *Portfolio Optimization: Theory and Application.*
Cambridge University Press.

portfoliooptimizationbook.com

Exercise 7.1: Efficient frontier

a. Download market data corresponding to N assets (e.g., stocks or cryptocurrencies) during a period with T observations, $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_T \in \mathbb{R}^N$.
b. Estimate the expected return vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.
c. Plot the mean–volatility efficient frontier computed by solving different mean–variance formulations, namely:

- the scalarized form:

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - \frac{\lambda}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w} \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}; \end{array}$$

- the variance-constrained form:

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} \\ \text{subject to} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w} \leq \alpha, \\ & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}; \end{array}$$

- the expected return-constrained form:

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{minimize}} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w} \\ \text{subject to} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} \geq \beta, \\ & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}. \end{array}$$

d. Discuss the benefits and drawbacks of the three methods for calculating the efficient frontier.

Solution

a. Market data corresponding to N stocks:

```
library(xts)
library(pob)       # Market data used in the book

# Use data from package pob
data(SP500_2015to2020)
stock_prices <- SP500_2015to2020$stocks[
  "2019-10::",
  c("AAPL", "AMZN", "AMD", "GM", "GOOGL", "MGM", "MSFT", "QCOM", "TSCO", "UPS")
  ]
X <- diff(log(stock_prices))[-1]
```

b. Estimate the expected return vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$:

```
mu <- colMeans(X)
Sigma <- cov(X)
```

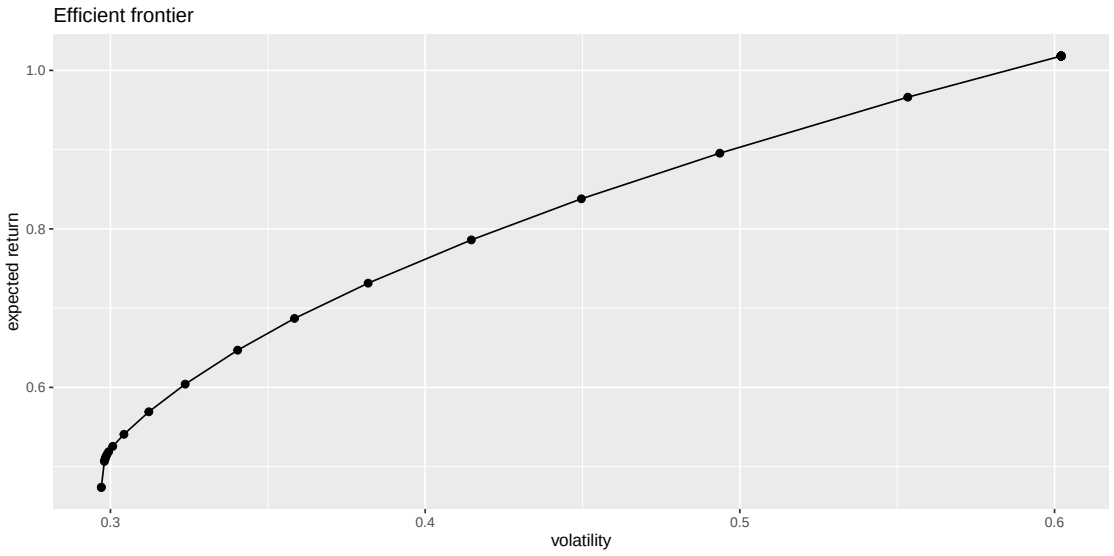
c. Plot the mean–volatility efficient frontier:

- via the scalarized form:

```
library(CVXR)

# Define portfolio formulation
design_MVP_scalarized <- function(mu, Sigma, lmd = 1) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - (lmd/2)*quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1))
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  return(w)
}
```

```r
# Generate grid of lambdas to evaluate the efficient frontier
lmd_sweep <- exp(c(24, 10, seq(4, -4, length.out = 40)))

# Compute Pareto-optimal portfolios
w_frontier <- c()
for (lmd in lmd_sweep)
  w_frontier <- cbind(w_frontier, design_MVP_scalarized(mu, Sigma, lmd))

# Compute the mean and volatility of the efficient frontier
rets <- xts(X %*% w_frontier, index(X))
ret_frontier <- colMeans(rets)
vol_frontier <- apply(rets, 2, sd)

max_mean <- max(ret_frontier)
min_vol <- min(vol_frontier)
```

```r
library(ggplot2)

# Plot
data.frame(x = sqrt(252) * vol_frontier, y = 252 * ret_frontier) |>
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 2) +
  geom_line() +
  labs(title = "Efficient frontier", x = "volatility", y = "expected return")
```

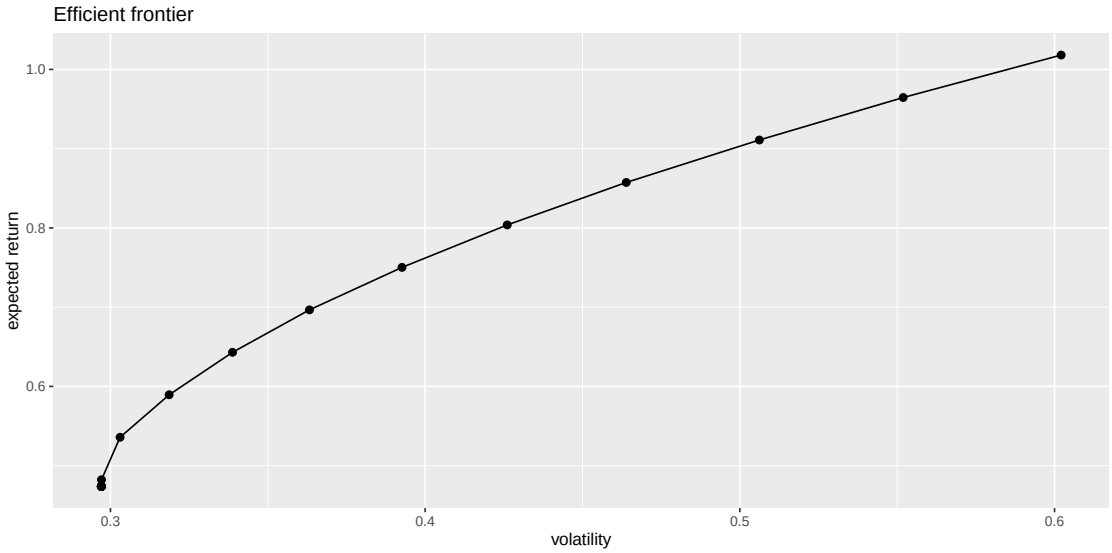


- via the variance-constrained form:

```r
# Define portfolio formulation
design_MVP_var_const <- function(mu, Sigma, alpha = Inf) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w),
                  constraints = list(
                    quad_form(w, Sigma) <= alpha,
                    w >= 0, sum(w) == 1
                    ))
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  return(w)
}
```

```r
# Generate grid of alphas to evaluate the efficient frontier
alpha_sweep <- seq(min_vol, 2*min_vol, length.out = 20)^2

# Compute Pareto-optimal portfolios
w_frontier <- c()
for (alpha in alpha_sweep)
  w_frontier <- cbind(w_frontier, design_MVP_var_const(mu, Sigma, alpha))

# Compute the mean and volatility of the efficient frontier
rets <- xts(X %*% w_frontier, index(X))
ret_frontier <- colMeans(rets)
vol_frontier <- apply(rets, 2, sd)
```

```r
# Plot
data.frame(x = sqrt(252) * vol_frontier, y = 252 * ret_frontier) |>
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 2) +
  geom_line() +
  labs(title = "Efficient frontier", x = "volatility", y = "expected return")
```

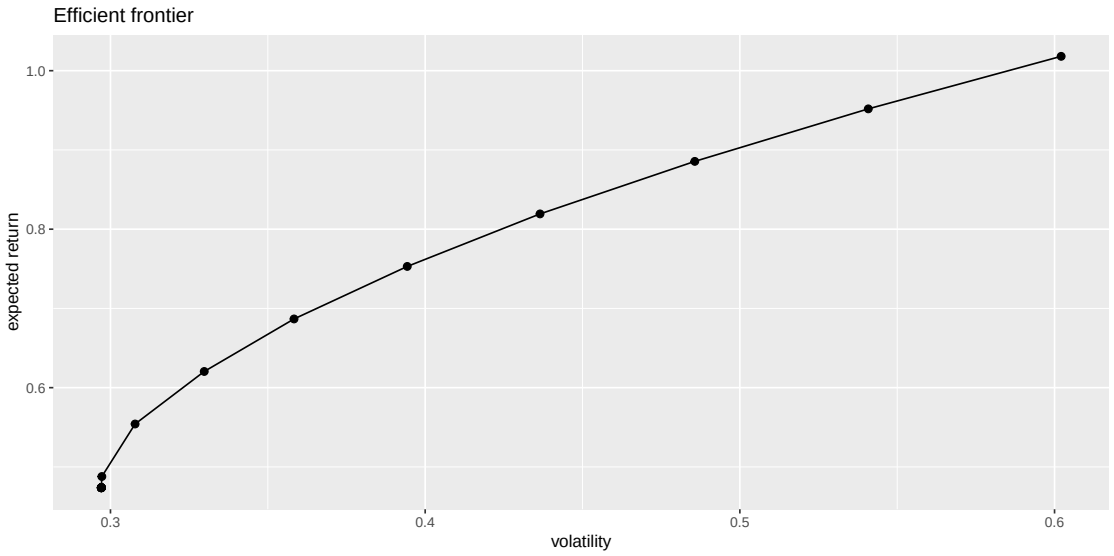


- via the expected return-constrained scalarized form:

```
# Define portfolio formulation
design_MVP_mean_const <- function(mu, Sigma, beta = -Inf) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Minimize(quad_form(w, Sigma)),
                  constraints = list(
                    t(mu) %*% w >= beta,
                    w >= 0, sum(w) == 1
                    ))
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  return(w)
}
```

```
# Generate grid of betas to evaluate the efficient frontier
beta_sweep <- seq(min(mu), max_mean, length.out = 20)

# Compute Pareto-optimal portfolios
w_frontier <- c()
for (beta in beta_sweep)
  w_frontier <- cbind(w_frontier, design_MVP_mean_const(mu, Sigma, beta))

# Compute the mean and volatility of the efficient frontier
rets <- xts(X %*% w_frontier, index(X))
ret_frontier <- colMeans(rets)
vol_frontier <- apply(rets, 2, sd)
```

```r
# Plot
data.frame(x = sqrt(252) * vol_frontier, y = 252 * ret_frontier) |>
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 2) +
  geom_line() +
  labs(title = "Efficient frontier", x = "volatility", y = "expected return")
```



d. Discuss the benefits and drawbacks of the three methods for calculating the efficient frontier:

- The scalarized form is convenient because the problem formulation is always feasible, but it may be challenging to choose appropriate values for the grid of λ values.
- The variance-constrained form leads to a formulation that may be unfeasible, but the choice of the hyperparameter α has a clear physical meaning (maximum variance).
- The expected return-constrained form similarly leads to a formulation that may be unfeasible, but the also choice of the hyperparameter β has a clear physical meaning (minimum expected return).

Exercise 7.2: Efficient frontier with practical constraints

Repeat Exercise 7.1 including different realistic constraints and discuss the differences. In particular:

- leverage constraint: $\|\boldsymbol{w}\|_1 \leq \gamma$
- turnover constraint: $\|\boldsymbol{w} - \boldsymbol{w}_0\|_1 \leq \tau$
- max position constraint: $|\boldsymbol{w}| \leq \boldsymbol{u}$
- market neutral constraint: $\boldsymbol{\beta}^\mathsf{T}\boldsymbol{w} = 0$
- sparsity constraint: $\|\boldsymbol{w}\|_0 \leq K$.

Solution

We will use the scalarized form but similarly the variance-constrained form or expected return-constrained form could be used:

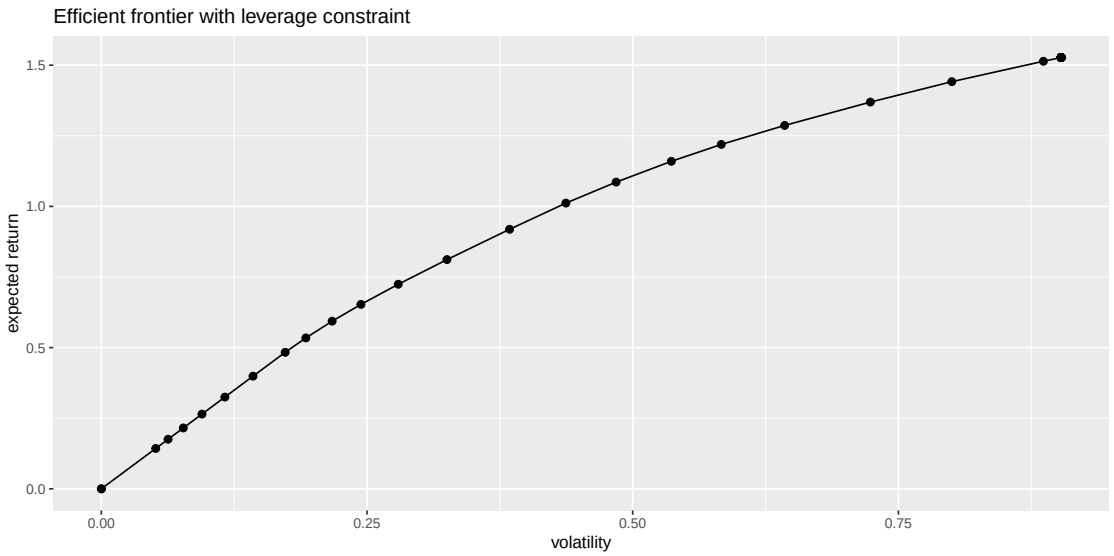
$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - \frac{\lambda}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w} \\ \text{subject to} & \boldsymbol{w} \in \mathcal{W}. \end{array}$$

- leverage constraint: $\|\boldsymbol{w}\|_1 \leq \gamma$

```
library(CVXR)

# Define portfolio formulation
design_MVP_leverage_const <- function(mu, Sigma, lmd = 1) {
  gamma <- 1.5
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - (lmd/2)*quad_form(w, Sigma)),
                  constraints = list(sum(abs(w)) <= gamma))
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  return(w)
}
```

```
# Generate grid of lambdas to evaluate the efficient frontier
lmd_sweep <- exp(c(24, 10, seq(4, -4, length.out = 40)))

# Compute Pareto-optimal portfolios
w_frontier <- c()
for (lmd in lmd_sweep)
  w_frontier <- cbind(w_frontier, design_MVP_leverage_const(mu, Sigma, lmd))

# Compute the mean and volatility of the efficient frontier
rets <- xts(X %*% w_frontier, index(X))
ret_frontier <- colMeans(rets)
vol_frontier <- apply(rets, 2, sd)
```

```
library(ggplot2)

# Plot
data.frame(x = sqrt(252) * vol_frontier, y = 252 * ret_frontier) |>
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 2) +
  geom_line() +
  labs(title = "Efficient frontier with leverage constraint",
       x = "volatility", y = "expected return")
```

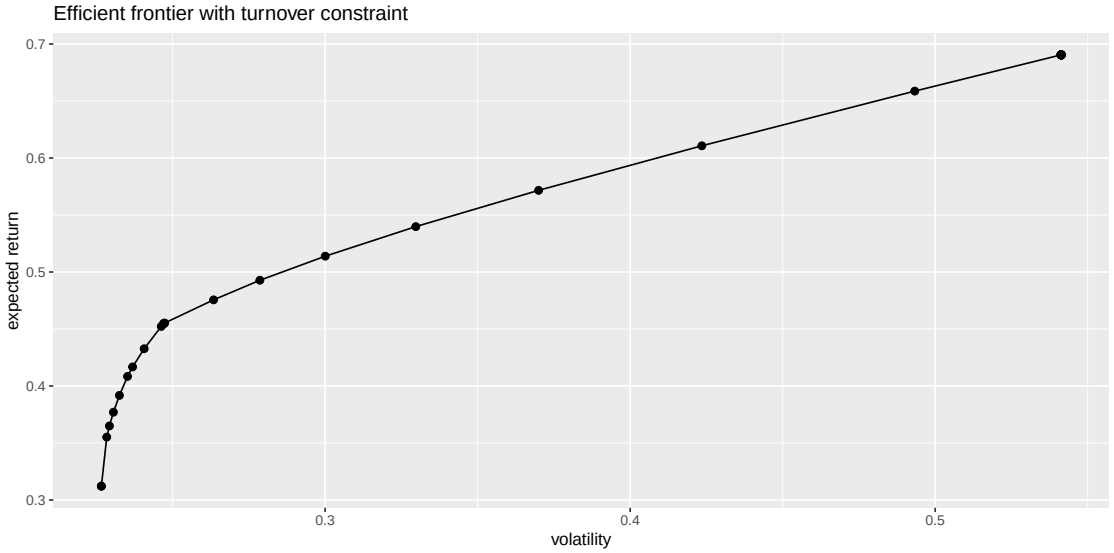


- turnover constraint: $\|\boldsymbol{w} - \boldsymbol{w}_0\|_1 \leq \tau$

```
# Define portfolio formulation
design_MVP_turnover_const <- function(mu, Sigma, lmd = 1) {
  N <- length(mu)
  w0 <- rep(1/N, N)
  tau <- 0.3
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - (lmd/2)*quad_form(w, Sigma)),
                  constraints = list(sum(abs(w - w0)) <= tau))
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  return(w)
}
```

```r
# Generate grid of lambdas to evaluate the efficient frontier
lmd_sweep <- exp(c(24, 10, seq(4, -4, length.out = 40)))

# Compute Pareto-optimal portfolios
w_frontier <- c()
for (lmd in lmd_sweep)
  w_frontier <- cbind(w_frontier, design_MVP_turnover_const(mu, Sigma, lmd))

# Compute the mean and volatility of the efficient frontier
rets <- xts(X %*% w_frontier, index(X))
ret_frontier <- colMeans(rets)
vol_frontier <- apply(rets, 2, sd)
```

```r
# Plot
data.frame(x = sqrt(252) * vol_frontier, y = 252 * ret_frontier) |>
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 2) +
  geom_line() +
  labs(title = "Efficient frontier with turnover constraint",
       x = "volatility", y = "expected return")
```

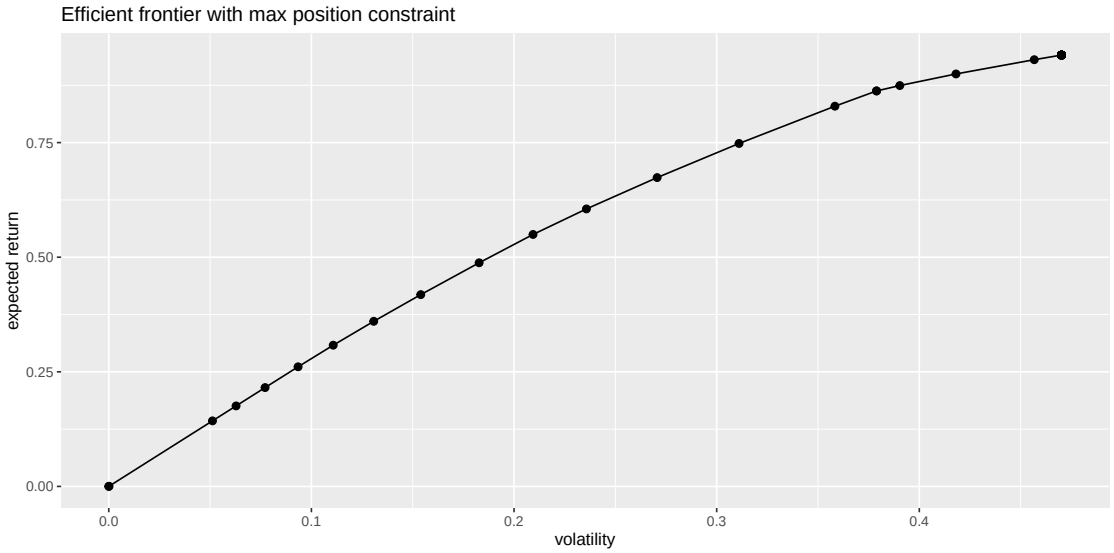


- max position constraint: $|\boldsymbol{w}| \leq \boldsymbol{u}$

```r
# Define portfolio formulation
design_MVP_upperbound_const <- function(mu, Sigma, lmd = 1) {
  N <- length(mu)
  u <- 2/N
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - (lmd/2)*quad_form(w, Sigma)),
                  constraints = list(abs(w) <= u))
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  return(w)
}
```

```r
# Generate grid of lambdas to evaluate the efficient frontier
lmd_sweep <- exp(c(24, 10, seq(4, -4, length.out = 40)))

# Compute Pareto-optimal portfolios
w_frontier <- c()
for (lmd in lmd_sweep)
  w_frontier <- cbind(w_frontier, design_MVP_upperbound_const(mu, Sigma, lmd))

# Compute the mean and volatility of the efficient frontier
rets <- xts(X %*% w_frontier, index(X))
ret_frontier <- colMeans(rets)
vol_frontier <- apply(rets, 2, sd)
```

```r
# Plot
data.frame(x = sqrt(252) * vol_frontier, y = 252 * ret_frontier) |>
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 2) +
  geom_line() +
  labs(title = "Efficient frontier with max position constraint",
       x = "volatility", y = "expected return")
```

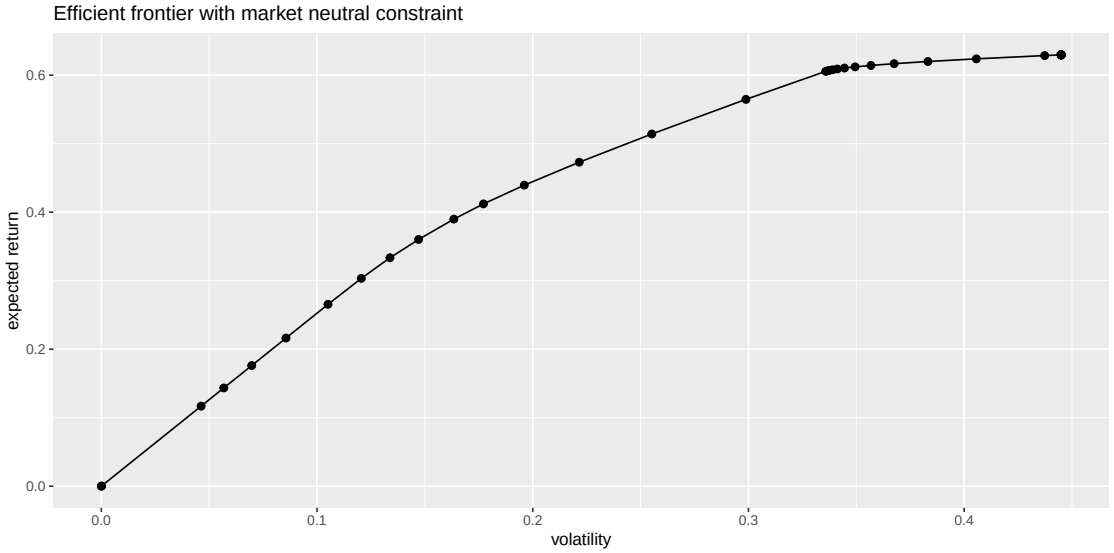


- market neutral constraint: $\boldsymbol{\beta}^{\mathsf{T}}\boldsymbol{w} = 0$

```
# Define portfolio formulation
design_MVP_marketneutral_const <- function(mu, Sigma, lmd = 1) {
  N <- length(mu)
  beta <- rep(1, N)
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - (lmd/2)*quad_form(w, Sigma)),
                  constraints = list(t(beta) %*% w == 0,
                                     sum(abs(w)) <= 1)
                  )
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  return(w)
}
```

```r
# Generate grid of lambdas to evaluate the efficient frontier
lmd_sweep <- exp(c(24, 10, seq(4, -4, length.out = 40)))

# Compute Pareto-optimal portfolios
w_frontier <- c()
for (lmd in lmd_sweep)
  w_frontier <- cbind(w_frontier, design_MVP_marketneutral_const(mu, Sigma, lmd))

# Compute the mean and volatility of the efficient frontier
rets <- xts(X %*% w_frontier, index(X))
ret_frontier <- colMeans(rets)
vol_frontier <- apply(rets, 2, sd)
```

```r
# Plot
data.frame(x = sqrt(252) * vol_frontier, y = 252 * ret_frontier) |>
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 2) +
  geom_line() +
  labs(title = "Efficient frontier with market neutral constraint",
       x = "volatility", y = "expected return")
```



- sparsity constraint: $\|\boldsymbol{w}\|_0 \leq K$

This constraint is tricky because it is not convex. We will reformulate the problem explicitly with

integer (or boolean {0,1}) constraints and then use a mixed-integer solver:

$$\begin{array}{ll} \underset{\boldsymbol{w},\boldsymbol{s}}{\text{maximize}} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - \frac{\lambda}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w} \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0} \\ & |\boldsymbol{w}| \leq \boldsymbol{s}, \quad \mathbf{1}^\mathsf{T}\boldsymbol{s} = K, \quad s_i \in \{0,1\} \quad \forall i. \end{array}$$

```
# Define portfolio formulation
design_MVP_sparsity_const <- function(mu, Sigma, lmd = 1) {
  N <- length(mu)
  K <- round(0.4 * N)
  w <- Variable(N)
  s <- Variable(N, boolean = TRUE)
  prob <- Problem(Maximize(t(mu) %*% w - (lmd/2)*quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1,
                                     abs(w) <= s, sum(s) <= K)
                  )
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  s <- as.vector(result$getValue(s))
  return(list("w" = w, "s" = s))
}
```

```
# Generate grid of lambdas to evaluate the efficient frontier
lmd_sweep <- exp(seq(4, -4, length.out = 40))

# Compute Pareto-optimal portfolios
w_frontier <- c()
for (lmd in lmd_sweep)
  w_frontier <- cbind(w_frontier, design_MVP_sparsity_const(mu, Sigma, lmd)$w)

# Compute the mean and volatility of the efficient frontier
rets <- xts(X %*% w_frontier, index(X))
ret_frontier <- colMeans(rets)
vol_frontier <- apply(rets, 2, sd)
```

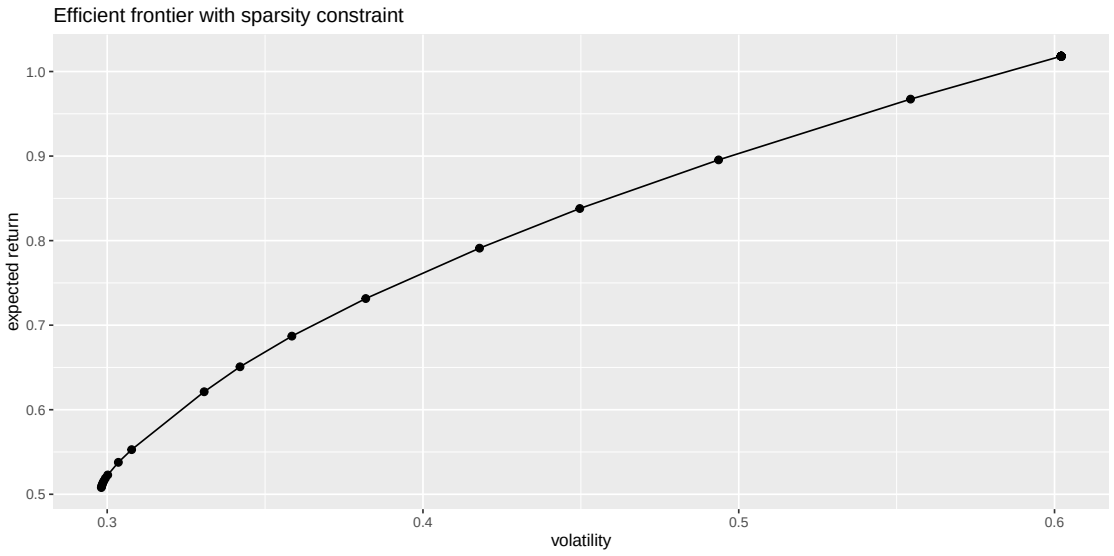
```
# Sanity checks
colSums(w_frontier)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[39] 1 1
```

```
colSums(abs(w_frontier) >= 1e-3)  # recall: K=4
```

```
 [1] 3 3 3 3 3 3 3 4 4 4 4 4 4 3 3 3 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[39] 1 1
```

```r
# Plot
data.frame(x = sqrt(252) * vol_frontier, y = 252 * ret_frontier) |>
  ggplot(aes(x = x, y = y)) +
  geom_point(size = 2) +
  geom_line() +
  labs(title = "Efficient frontier with sparsity constraint",
       x = "volatility", y = "expected return")
```



Exercise 7.3: Efficient frontier out of sample

a. Download market data corresponding to N assets during a period with T observations.
b. Using 70% of the data:
   - estimate the expected return vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$;
   - plot the mean–volatility efficient frontier by solving mean–variance formulations; and
   - plot some randomly generated feasible portfolios.
c. Using the remaining 30% of the data (out of sample):
   - estimate the expected return vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$;
   - plot the new mean–volatility efficient frontier; and
   - re-evaluate and plot the mean and volatility of the previously computed portfolios (the

ones defining the efficient frontier and the random ones).

d. Discuss the difference between the two efficient frontiers, as well as how the portfolios shift from in-sample to out-of-sample performance.

Solution

a. Market data corresponding to N stocks:

```
library(xts)
library(pob)       # Market data used in the book

# Use data from package pob
data(SP500_2015to2020)
stock_prices <- SP500_2015to2020$stocks[
  "2019-10::",
  c("AAPL", "AMZN", "AMD", "GM", "GOOGL", "MGM", "MSFT", "QCOM", "TSCO", "UPS")
  ]
X <- diff(log(stock_prices))[-1]
N <- ncol(X)
T <- nrow(X)
T_trn <- round(0.70*T)
T_tst <- T - T_trn
```

b. Using 70% of the data:

- estimate the expected return vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$;
- plot the mean–volatility efficient frontier by solving mean–variance formulations; and
- plot some randomly generated feasible portfolios.

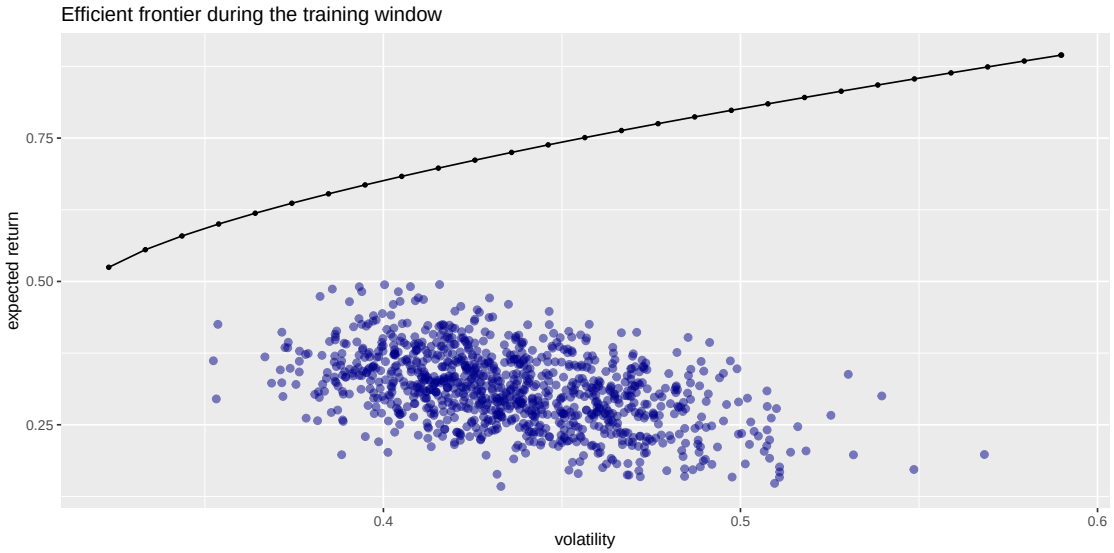
```
# Estimate mu and Sigma from training data
T_trn <- round(0.70*T)
X_trn <- X[c(1:T_trn), ]
mu_hat <- colMeans(X_trn)
Sigma_hat <- cov(X_trn)
```

```r
# Define portfolio formulation
design_MVP_var_const <- function(mu, Sigma, alpha = Inf) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w),
                  constraints = list(
                    quad_form(w, Sigma) <= alpha,
                    w >= 0, sum(w) == 1
                    ))
  result <- solve(prob)
  return(list("feasible" = result$status == "optimal",
              "w" = as.vector(result$getValue(w))))
}
```

```r
# Generate grid of alphas to evaluate the efficient frontier
alpha_sweep <- (seq(0.2, 0.6, length.out = 40)/sqrt(252))^2

# Compute Pareto-optimal portfolios
w_frontier_trn <- NULL
for (alpha in alpha_sweep) {
  result <- design_MVP_var_const(mu_hat, Sigma_hat, alpha)
  if (result$feasible)
    w_frontier_trn <- cbind(w_frontier_trn, result$w)

}

# Compute the mean and volatility of the efficient frontier
rets <- xts(X_trn %*% w_frontier_trn, index(X_trn))
ret_frontier_trn <- colMeans(rets)
vol_frontier_trn <- apply(rets, 2, sd)
```

```r
# Generate random portfolios
design_RandomP <- function(N) {
  w <- runif(N)
  w <- w/sum(w)
  return(w)
}

set.seed(42)
w_random <- NULL
for (i in 1:1000) {
  # N_active <- sample(N, 1)
  # idx <- sample(1:N, N_active)
  w_tmp <- rep(0, N)
  N_active <- N
  idx <- 1:N
  w_tmp[idx] <- design_RandomP(N_active)
  w_random <- cbind(w_random, w_tmp)
}

# Compute the mean and volatility of the efficient frontier
rets <- xts(X_trn %*% w_random, index(X_trn))
ret_random_trn <- colMeans(rets)
vol_random_trn <- apply(rets, 2, sd)
```

```r
# Plot both efficient frontier and random portfolios
data.frame(x = sqrt(252) * vol_frontier_trn, y = 252 * ret_frontier_trn) |>
  ggplot(aes(x = x, y = y)) +
  geom_line() + geom_point(size = 1) +
  geom_point(data = data.frame(x = sqrt(252) * vol_random_trn, y = 252 * ret_random_trn),
             aes(x, y), size = 2, alpha = 0.5, col = "darkblue") +
  labs(title = "Efficient frontier during the training window",
       x = "volatility", y = "expected return")
```



c. Using the remaining 30% of the data (out of sample):

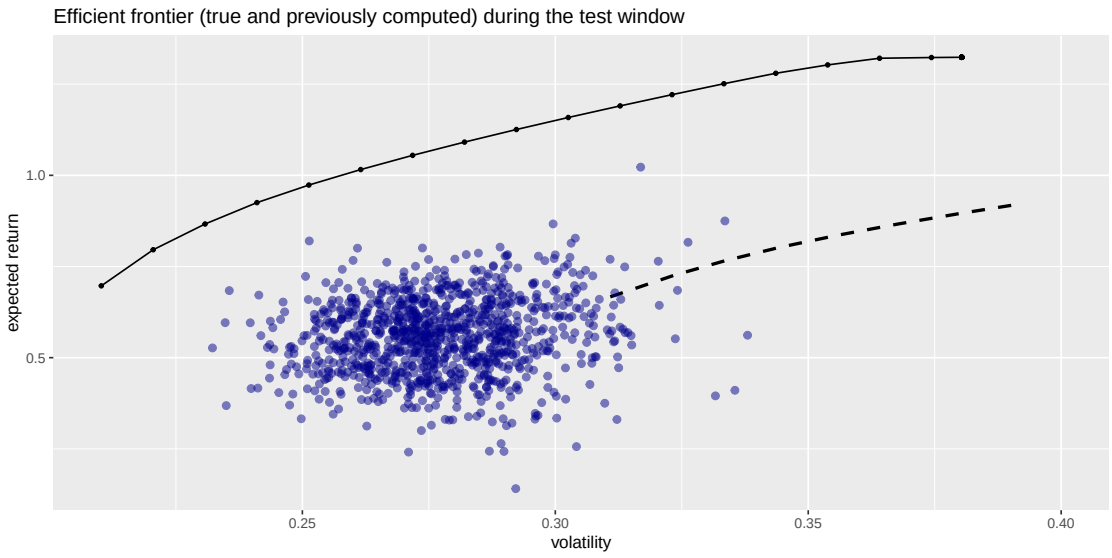
- estimate the expected return vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$;
- plot the new mean–volatility efficient frontier; and
- re-evaluate and plot the mean and volatility of the previously computed portfolios (the ones defining the efficient frontier and the random ones).

```
# Estimate true mu and Sigma from test data
X_tst <- X[-c(1:T_trn), ]
mu_true <- colMeans(X_tst)
Sigma_true <- cov(X_tst)
```

```
# Compute Pareto-optimal portfolios
w_frontier_tst <- NULL
for (alpha in alpha_sweep) {
  result <- design_MVP_var_const(mu_true, Sigma_true, alpha)
  if (result$feasible)
    w_frontier_tst <- cbind(w_frontier_tst, result$w)

}

# Compute the mean and volatility of the efficient frontier
rets <- xts(X_tst %*% w_frontier_tst, index(X_tst))
ret_frontier_tst <- colMeans(rets)
vol_frontier_tst <- apply(rets, 2, sd)
```

```r
# Reevaluate efficient frontier of the training window and random portfolios
rets <- xts(X_tst %*% w_frontier_trn, index(X_tst))
ret_frontier_trn_in_tst <- colMeans(rets)
vol_frontier_trn_in_tst <- apply(rets, 2, sd)

rets <- xts(X_tst %*% w_random, index(X_tst))
ret_random_tst <- colMeans(rets)
vol_random_tst <- apply(rets, 2, sd)
```

```r
# Plot everything
data.frame(x = sqrt(252) * vol_frontier_tst , y = 252 * ret_frontier_tst ) |>
  ggplot(aes(x = x, y = y)) +
  geom_line() + geom_point(size = 1) +
  geom_line(data = data.frame(x = sqrt(252) * vol_frontier_trn_in_tst ,
                              y = 252 * ret_frontier_trn_in_tst),
            aes(x, y), linetype = "dashed", linewidth = 1, col = "black") +
  xlim(NA, 0.4) +
  geom_point(data = data.frame(x = sqrt(252) * vol_random_tst , y = 252 * ret_random_tst),
             aes(x, y), size = 2, alpha = 0.5, col = "darkblue") +
  labs(title = "Efficient frontier (true and previously computed) during the test window",
       x = "volatility", y = "expected return")
```



d. Discuss the difference between the two efficient frontiers, as well as how the portfolios shift from in-sample to out-of-sample performance.

From the plot of the return vs. volatility, we can observe how bad the efficient frontier calculated

during the training window becomes in the test window (out of sample). This explains why the mean–variance portfolio performs poorly out of sample.

Exercise 7.4: Improving the mean–variance portfolio with heuristics

Repeat Exercise 7.3 including the following heuristic constraints to regularize the mean–variance portfolios:

- upper bound constraint: $\|\boldsymbol{w}\|_\infty \leq 0.25$
- diversification constraint: $\|\boldsymbol{w}\|_2^2 \leq 0.25$.

Solution

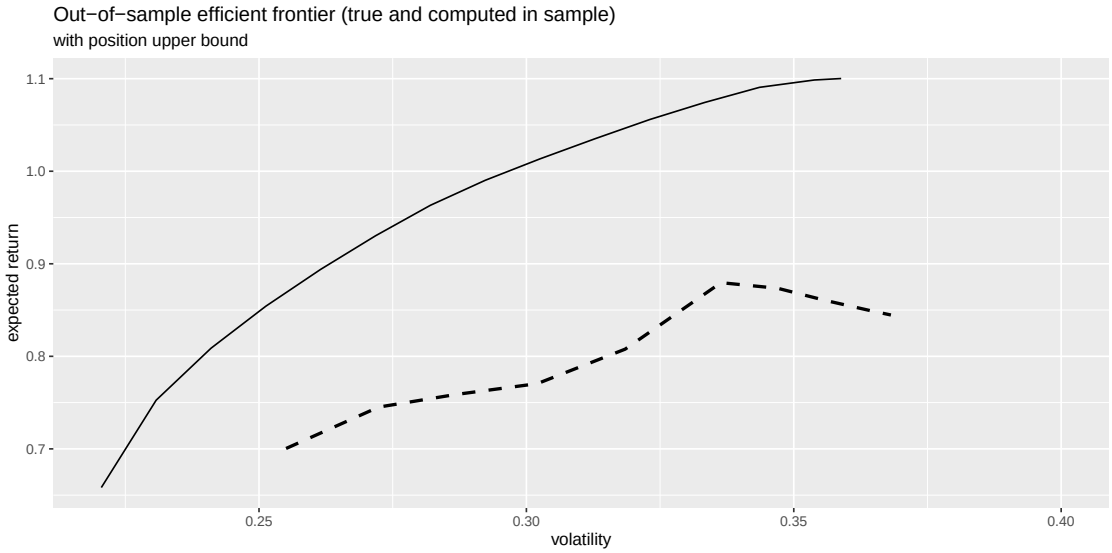
- With upper bound constraint: $\|\boldsymbol{w}\|_\infty \leq 0.25$

```
# Define portfolio formulation
design_MVP_var_const_pos_ub <- function(mu, Sigma, alpha = Inf, pos_ub = 0.25) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w),
                  constraints = list(
                    quad_form(w, Sigma) <= alpha,
                    w >= 0, sum(w) == 1,
                    abs(w) <= pos_ub
                    ))
  result <- solve(prob)
  return(list("feasible" = result$status == "optimal",
              "w" = as.vector(result$getValue(w))))
}
```

```r
# Generate grid of alphas to evaluate the efficient frontier
alpha_sweep <- (seq(0.2, 0.6, length.out = 40)/sqrt(252))^2

# Compute Pareto-optimal portfolios during training window
w_frontier_trn <- NULL
for (alpha in alpha_sweep) {
  result <- design_MVP_var_const_pos_ub(mu_hat, Sigma_hat, alpha)
  if (result$feasible)
    w_frontier_trn <- cbind(w_frontier_trn, result$w)

}

# Compute Pareto-optimal portfolios during the test window
w_frontier_tst <- NULL
for (alpha in alpha_sweep) {
  result <- design_MVP_var_const_pos_ub(mu_true, Sigma_true, alpha)
  if (result$feasible)
    w_frontier_tst <- cbind(w_frontier_tst, result$w)

}
```

```r
# Compute the mean and volatility of both efficient frontiers
rets <- xts(X_tst %*% w_frontier_trn, index(X_tst))
ret_frontier_trn_in_tst <- colMeans(rets)
vol_frontier_trn_in_tst <- apply(rets, 2, sd)

rets <- xts(X_tst %*% w_frontier_tst, index(X_tst))
ret_frontier_tst <- colMeans(rets)
vol_frontier_tst <- apply(rets, 2, sd)
```

```r
# Plot both efficient frontiers
data.frame(x = sqrt(252) * vol_frontier_tst , y = 252 * ret_frontier_tst ) |>
  ggplot(aes(x = x, y = y)) +
  geom_line() + #geom_point(size = 1) +
  geom_line(data = data.frame(x = sqrt(252) * vol_frontier_trn_in_tst ,
                              y = 252 * ret_frontier_trn_in_tst),
            aes(x, y), linetype = "dashed", linewidth = 1, col = "black") +
  xlim(NA, 0.4) +
  labs(title = "Out-of-sample efficient frontier (true and computed in sample)",
       subtitle = "with position upper bound",
       x = "volatility", y = "expected return")
```



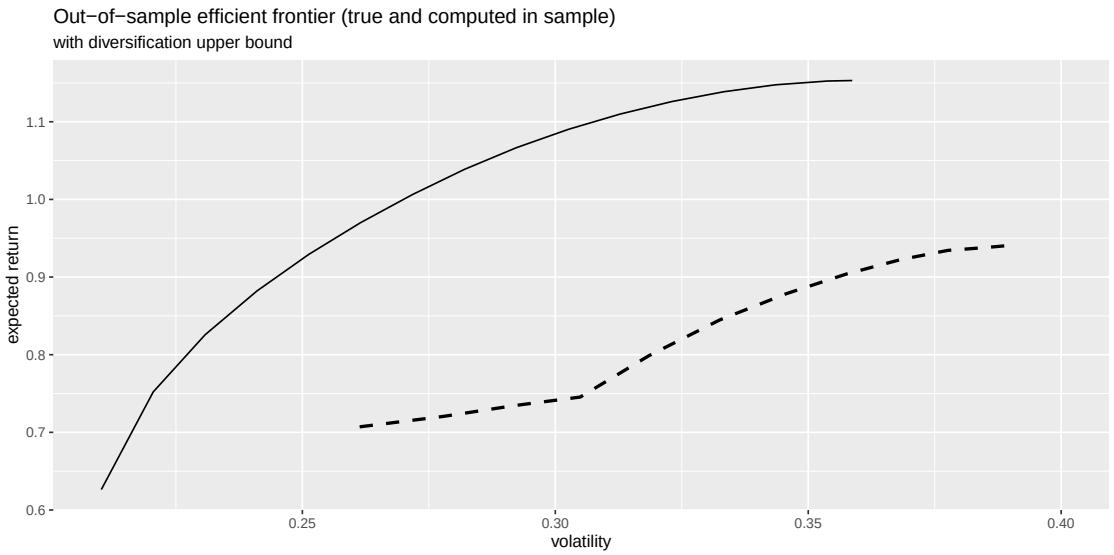
- With diversification constraint: $\|\boldsymbol{w}\|_2^2 \leq 0.25$

```
# Define portfolio formulation
design_MVP_var_const_div_ub <- function(mu, Sigma, alpha = Inf, div_ub = 0.25) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w),
                  constraints = list(
                    quad_form(w, Sigma) <= alpha,
                    w >= 0, sum(w) == 1,
                    sum_squares(w) <= div_ub
                    ))
  result <- solve(prob)
  return(list("feasible" = result$status == "optimal",
              "w" = as.vector(result$getValue(w))))
}
```

```r
# Generate grid of alphas to evaluate the efficient frontier
alpha_sweep <- (seq(0.2, 0.6, length.out = 40)/sqrt(252))^2

# Compute Pareto-optimal portfolios during training window
w_frontier_trn <- NULL
for (alpha in alpha_sweep) {
  result <- design_MVP_var_const_div_ub(mu_hat, Sigma_hat, alpha)
  if (result$feasible)
    w_frontier_trn <- cbind(w_frontier_trn, result$w)

}

# Compute Pareto-optimal portfolios during the test window
w_frontier_tst <- NULL
for (alpha in alpha_sweep) {
  result <- design_MVP_var_const_div_ub(mu_true, Sigma_true, alpha)
  if (result$feasible)
    w_frontier_tst <- cbind(w_frontier_tst, result$w)

}
```

```r
# Compute the mean and volatility of both efficient frontiers
rets <- xts(X_tst %*% w_frontier_trn, index(X_tst))
ret_frontier_trn_in_tst <- colMeans(rets)
vol_frontier_trn_in_tst <- apply(rets, 2, sd)

rets <- xts(X_tst %*% w_frontier_tst, index(X_tst))
ret_frontier_tst <- colMeans(rets)
vol_frontier_tst <- apply(rets, 2, sd)
```

```r
# Plot both efficient frontiers
data.frame(x = sqrt(252) * vol_frontier_tst , y = 252 * ret_frontier_tst ) |>
  ggplot(aes(x = x, y = y)) +
  geom_line() + #geom_point(size = 1) +
  geom_line(data = data.frame(x = sqrt(252) * vol_frontier_trn_in_tst ,
                              y = 252 * ret_frontier_trn_in_tst),
            aes(x, y), linetype = "dashed", linewidth = 1, col = "black") +
  xlim(NA, 0.4) +
  labs(title = "Out-of-sample efficient frontier (true and computed in sample)",
       subtitle = "with diversification upper bound",
       x = "volatility", y = "expected return")
```



Exercise 7.5: Computation of the MSRP

a. Download market data corresponding to N assets during a period with T observations.
b. Estimate the expected return vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.
c. Compute the maximum Sharpe ratio portfolio

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \dfrac{\boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - r_\mathrm{f}}{\sqrt{\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w}}} \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}, \end{array}$$

with the following methods:

- bisection method
- Dinkelbach method
- Schaible transform method.

Solution

a. Market data corresponding to N stocks:

```
library(xts)
library(pob)       # Market data used in the book

# Use data from package pob
data(SP500_2015to2020)
stock_prices <- SP500_2015to2020$stocks[
  "2019-10::",
  c("AAPL", "AMZN", "AMD", "GM", "GOOGL", "MGM", "MSFT", "QCOM", "TSCO", "UPS")
  ]
X <- diff(log(stock_prices))[-1]
```

b. Estimate the expected return vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$:

```
mu <- colMeans(X)
Sigma <- cov(X)
```

c. Compute the maximum Sharpe ratio portfolio:

- Bisection method: we need to solve a sequence of convex feasibility problems are of the form

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{find}} & \boldsymbol{w} \\ \text{subject to} & t\sqrt{\boldsymbol{w}^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{w}} \leq \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\mu} - r_{\mathrm{f}}, \\ & \mathbf{1}^{\mathsf{T}}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}, \end{array}$$

```
# define the inner solver based on an SOCP solver
# (we will simply use CVXR for convenience, see: https://cvxr.rbind.io/cvxr_functions/)
library(CVXR)

# square-root of matrix Sigma
Sigma_12 <- chol(Sigma)
cat("Sanity check of Cholesky decomposition:",
    max(abs(t(Sigma_12) %*% Sigma_12 - Sigma)))
```

```
Sanity check of Cholesky decomposition: 2.168404e-19
```

```r
# create function for MVP
SOCP_bisection <- function(t) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(0),
                  constraints = list(t*cvxr_norm(Sigma_12 %*% w, 2) <= t(mu) %*% w,
                                     sum(w) == 1, w >= 0))
  result <- solve(prob)
  return(list("status" = result$status, "w" = as.vector(result$getValue(w))))
}

# now run the bisection algorithm
t_lb <- 0   # for sure the problem is feasible in this case
t_ub <- 10  # a tighter upper bound could be chosen (10 is a conservative choice)
while(t_ub - t_lb > 1e-6) {
  t <- (t_ub + t_lb)/2  # midpoint
  if(SOCP_bisection(t)$status == "infeasible")
    t_ub <- t
  else
    t_lb <- t
}
w_bisection <- SOCP_bisection(t_lb)$w
```

- Dinkelbach method: we need to solve a sequence of SOCPs of the form ($r_\mathrm{f} = 0$):

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - y^{(k)} \left\|\boldsymbol{\Sigma}^{1/2}\boldsymbol{w}\right\|_2 \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}. \end{array}$$

where

$$y^{(k)} = \frac{\mathbf{w}^{(k)\mathsf{T}}\boldsymbol{\mu}}{\sqrt{\boldsymbol{w}^{(k)\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{w}^{(k)}}}.$$

```r
# define the inner solver based on an SOCP solver
# (we will simply use CVXR for convenience, see: https://cvxr.rbind.io/cvxr_functions/)

# create function for MVP
SOCP_Dinkelbach <- function(y) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - y*cvxr_norm(Sigma_12 %*% w, 2)),
                  constraints = list(sum(w) == 1, w >= 0))
  result <- CVXR::solve(prob)
  return(as.vector(result$getValue(w)))
}

# initial point (has to satisfy t(w_k) %*% mu>=0)
i_max <- which.max(mu)
w_k <- rep(0, N)
w_k[i_max] <- 1

# now the iterative Dinkelbach algorithm
k <- 1
while(k == 1 || max(abs(w_k - w_prev)) > 1e-6) {
  w_prev <- w_k
  y_k <- as.numeric(t(w_k) %*% mu / sqrt(t(w_k) %*% Sigma %*% w_k))
  w_k <- SOCP_Dinkelbach(y_k)
  k <- k + 1
}
w_Dinkelbach <- w_k
cat("Number of iterarions:", k-1)
```

```
Number of iterarions: 5
```

- Schaible transform method ($r_{\mathrm{f}} = 0$):

$$\begin{array}{ll} \underset{\boldsymbol{y}}{\text{maximize}} & \boldsymbol{y}^{\mathsf{T}}\boldsymbol{\mu} \\ \text{subject to} & \boldsymbol{y}^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{y} \leq 1, \\ & \boldsymbol{y} \geq \mathbf{0}, \end{array}$$

from which the original variable $\boldsymbol{w}$ can be easily recovered from $\boldsymbol{y}$, and $t = \mathbf{1}^{\mathsf{T}}\boldsymbol{y}$ as $\boldsymbol{w} = \boldsymbol{y}/\left(\mathbf{1}^{\mathsf{T}}\boldsymbol{y}\right)$. specific QP solver like `quadprog` for speed and stability):

```r
# create function for MSRP
MSRP <- function(mu, Sigma) {
  w_ <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w_),
                  constraints = list(quad_form(w_, Sigma) <= 1,
                                     w_ >= 0))
  result <- solve(prob)
  y <- as.vector(result$getValue(w_))
  w <- y/sum(y)
  names(w) <- colnames(Sigma)
  return(w)
}

# this function can now be used as
w_MSRP <- MSRP(mu, Sigma)
```

Comparison of the three solutions:

```r
# Comparison among three solutions
round(cbind(w_bisection, w_Dinkelbach, w_MSRP), digits = 3)
```

```
      w_bisection w_Dinkelbach w_MSRP
AAPL        0.161         0.161  0.161
AMZN        0.446         0.446  0.446
AMD         0.247         0.247  0.247
GM          0.000         0.000  0.000
GOOGL       0.000         0.000  0.000
MGM         0.000         0.000  0.000
MSFT        0.000         0.000  0.000
QCOM        0.000         0.000  0.000
TSCO        0.146         0.146  0.146
UPS         0.000         0.000  0.000
```

```r
# Sharpe ratio of solutions
fn_SR <- function(w) {
  return(as.numeric(t(w) %*% mu / sqrt(t(w) %*% Sigma %*% w)))
}

cat("Sharpe ratios:\n")
```

```
Sharpe ratios:
```

```r
c("bisection"        = fn_SR(w_bisection),
  "Dinkelbach"       = fn_SR(w_Dinkelbach),
  "Schaible"         = fn_SR(w_MSRP))
```

```
bisection Dinkelbach   Schaible
0.1208336  0.1208338  0.1208338
```

Exercise 7.6: Kelly portfolio

a. Download market data corresponding to N assets during a period with T observations.
b. Compute the Kelly portfolio

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \mathbb{E}\left[\log\left(1+\boldsymbol{w}^{\mathsf{T}}\boldsymbol{r}\right)\right] \\ \text{subject to} & \mathbf{1}^{\mathsf{T}}\boldsymbol{w}=1, \quad \boldsymbol{w}\geq\mathbf{0}, \end{array}$$

with the following methods:

- sample average approximation
- mean–variance approximation
- Levy–Markowitz approximation.

Solution

a. Market data corresponding to N stocks:

```
library(xts)
library(pob)       # Market data used in the book

# Use data from package pob
data(SP500_2015to2020)
stock_prices <- SP500_2015to2020$stocks[
  "2019-10::",
  c("AAPL", "AMZN", "AMD", "GM", "GOOGL", "MGM", "MSFT", "QCOM", "TSCO", "UPS")
  ]
X <- diff(log(stock_prices))[-1]
N <- ncol(X)

# Estimate mu and Sigma
mu <- colMeans(X)
Sigma <- cov(X)
```

b. Compute the Kelly portfolio:

- With sample average approximation:

```
design_Kelly_portfolio <- function(X) {
  w <- Variable(ncol(X))
  prob <- Problem(Maximize(mean(log(1 + as.matrix(X) %*% w))),
                  constraints = list(sum(w) == 1, w >= 0))
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  names(w) <- colnames(X)
  return(w)
}

w_Kelly <- design_Kelly_portfolio(X)
```

- With mean–variance approximation:

```
design_MVP <- function(mu, Sigma, lambda = 1) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - (lambda/2)*quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1))
  result <- solve(prob)
  w <- as.vector(result$getValue(w))
  return(w)
}

w_MVP <- design_MVP(mu, Sigma)
```

- With Levy–Markowitz approximation:

$$\mathbb{E}\left[U(\boldsymbol{w}^\mathsf{T}\boldsymbol{r})\right] \approx U(\boldsymbol{w}^\mathsf{T}\boldsymbol{\mu}) + \frac{U\left(\boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} + \kappa\sqrt{\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w}}\right) + U\left(\boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - \kappa\sqrt{\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w}}\right) - 2U\left(\boldsymbol{w}^\mathsf{T}\boldsymbol{\mu}\right)}{2\kappa^2}.$$

```r
library(nloptr)

# define the nonconvex objective function
U <- function(t) log(1 + t)

kappa <- 1
Levy_Markowitz_obj <- function(w) {
  mu_w <- as.numeric(t(w) %*% mu)
  sqrt_w_Sigma_w <- sqrt(as.numeric(t(w) %*% Sigma %*% w))
  U(mu_w) + (U(mu_w + kappa*sqrt_w_Sigma_w) + U(mu_w - kappa*sqrt_w_Sigma_w)
             - 2*U(mu_w)) / (2*kappa^2)
}
neg_Levy_Markowitz_obj <- function(w) -Levy_Markowitz_obj(w)

# random initial point
w0 <- runif(N)
w0 <- w0/sum(w0)

res <- nloptr::slsqp(w0, neg_Levy_Markowitz_obj,
                     lower = rep(0, N), upper = rep(1, N),
                     heq = function(w) return(sum(w) - 1))    # sum(w) = 1
w_Levy_Markowitz <- res$par
```

Comparison of the three solutions:

```r
# Comparison among three solutions
round(cbind(w_Kelly, w_MVP, w_Levy_Markowitz), digits = 3)
```

```
      w_Kelly w_MVP w_Levy_Markowitz
AAPL        0     0                0
AMZN        0     0                0
AMD         1     1                1
GM          0     0                0
GOOGL       0     0                0
MGM         0     0                0
MSFT        0     0                0
QCOM        0     0                0
TSCO        0     0                0
UPS         0     0                0
```

Exercise 7.7: Expected utility portfolio

a. Download market data corresponding to N assets during a period with T observations.
b. Compute the expected utility portfolio

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \mathbb{E}\left[U(\boldsymbol{w}^\mathsf{T}\boldsymbol{r})\right] \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}, \end{array}$$

with different utilities such as

- $U(x) = \log(1+x)$
- $U(x) = \sqrt{1+x}$
- $U(x) = -1/x$
- $U(x) = -p/x^p$ with $p > 0$
- $U(x) = -1/\sqrt{1+x}$
- $U(x) = 1 - \exp(-\lambda x)$ with $\lambda > 0$.

Solution

a. Market data corresponding to N stocks:

```
library(xts)
library(pob)       # Market data used in the book

# Use data from package pob
data(SP500_2015to2020)
stock_prices <- SP500_2015to2020$stocks[
  "2019-10::",
  c("AAPL", "AMZN", "AMD", "GM", "GOOGL", "MGM", "MSFT", "QCOM", "TSCO", "UPS")
  ]
X <- diff(log(stock_prices))[-1]
N <- ncol(X)

# Estimate mu and Sigma
mu <- colMeans(X)
Sigma <- cov(X)
```

b. Compute the expected utility portfolio:

- With (concave) utility $U(x) = \log(1+x)$:

```
library(CVXR)

U <- function(x) log(1 + x)

w <- Variable(N)
prob <- Problem(Maximize(mean(U(as.matrix(X) %*% w))),
                constraints = list(sum(w) == 1, w >= 0))
result <- solve(prob)
round(as.vector(result$getValue(w)), digits = 3)
```

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

- With (concave) utility $U(x) = \sqrt{1+x}$:

```
U <- function(x) sqrt(1 + x)

w <- Variable(N)
prob <- Problem(Maximize(mean(U(as.matrix(X) %*% w))),
                constraints = list(sum(w) == 1, w >= 0))
result <- solve(prob)
round(as.vector(result$getValue(w)), digits = 3)
```

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

- With (concave) utility $U(x) = -1/x$:

```
U <- function(x) -inv_pos(0.1 + x)

w <- Variable(N)
prob <- Problem(Maximize(mean(U(as.matrix(X) %*% w))),
                constraints = list(sum(w) == 1, w >= 0))
result <- solve(prob)
round(as.vector(result$getValue(w)), digits = 3)
```

```
 [1] 0.000 0.599 0.000 0.000 0.000 0.000 0.000 0.000 0.042 0.359
```

- With utility $U(x) = -p/x^p$ with $p > 0$ (not concave because x can be nonpositive!):

```
# We need a general-purpose solver
library(nloptr)

p <- 2
U <- function(x) -p/x^p
fn <- function(w) -mean(U(as.matrix(X) %*% w))

# random initial point
w0 <- runif(N)
w0 <- w0/sum(w0)

# solve
res <- nloptr::slsqp(w0, fn,
                     lower = rep(0, N), upper = rep(1, N),
                     heq = function(w) return(sum(w) - 1))    # sum(w) = 1

# print
round(as.vector(result$getValue(w)), digits = 3)
```

```
 [1] 0.000 0.599 0.000 0.000 0.000 0.000 0.000 0.000 0.042 0.359
```

- With (concave) utility $U(x) = -1/\sqrt{1+x}$:

```
U <- function(x) -power(1 + x, -0.5)


w <- Variable(N)
prob <- Problem(Maximize(mean(U(as.matrix(X) %*% w))),
                constraints = list(sum(w) == 1, w >= 0))
result <- solve(prob)
round(as.vector(result$getValue(w)), digits = 3)
```

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

- With (concave) utility $U(x) = 1 - \exp(-\lambda x)$ with $\lambda > 0$:

```
lmd <- 1
U <- function(x) 1 - exp(-lmd*x)

w <- Variable(N)
prob <- Problem(Maximize(mean(U(as.matrix(X) %*% w))),
                constraints = list(sum(w) == 1, w >= 0))
result <- solve(prob)
round(as.vector(result$getValue(w)), digits = 3)
```

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

Exercise 7.8: Universal successive mean–variance approximation method

a. Consider the maximum Sharpe ratio portfolio,

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \dfrac{\boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - r_\mathrm{f}}{\sqrt{\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w}}} \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}, \end{array}$$

and the mean–volatility portfolio,

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - \kappa\sqrt{\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w}} \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}, \end{array}$$

both of which lie on the efficient frontier.

b. Solve them with some appropriate method.

c. Solve them via the universal successive mean–variance approximation method, which at each iteration k, solves the mean–variance problem

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \boldsymbol{w}^\mathsf{T}\boldsymbol{\mu} - \dfrac{\lambda^k}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w} \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}, \end{array}$$

with a properly chosen λ^k.

d. Compare the obtained solutions and the computational cost.

Solution

First we download market data corresponding to N stocks to estimate the mean vector and covariance matrix:

```
library(xts)
library(pob)       # Market data used in the book

# Use data from package pob
data(SP500_2015to2020)
stock_prices <- SP500_2015to2020$stocks[
  "2019-10::",
  c("AAPL", "AMZN", "AMD", "GM", "GOOGL", "MGM", "MSFT", "QCOM", "TSCO", "UPS")
  ]
X <- diff(log(stock_prices))[-1]
N <- ncol(X)

# Estimate mu and Sigma
mu <- colMeans(X)
Sigma <- cov(X)
```

b. Resolution of problems via general-purpose solver:

- maximum Sharpe ratio portfolio:

```
library(nloptr)

# define the nonconvex objective function
fn_SR <- function(w)
  as.numeric(t(w) %*% mu / sqrt(t(w) %*% Sigma %*% w))

fn_neg_SR <- function(w) -fn_SR(w)

# initial point
w0 <- rep(1/N, N)

# solve problem
start_time <- Sys.time()
res <- nloptr::slsqp(w0, fn_neg_SR,
                     lower = rep(0, N), upper = rep(1, N),
                     heq = function(w) return(sum(w) - 1))    # sum(w) = 1
cpu_time_MSRP_gensolver <- Sys.time() - start_time
w_MSRP_gensolver <- res$par
```

- mean–volatility portfolio:

```
library(nloptr)

# define the nonconvex objective function
kappa <- 1
fn_neg_mean_vol <- function(w)
  -as.numeric(t(w) %*% mu - kappa * sqrt(t(w) %*% Sigma %*% w))

# initial point
w0 <- rep(1/N, N)

# solve problem
start_time <- Sys.time()
res <- nloptr::slsqp(w0, fn_neg_mean_vol,
                     lower = rep(0, N), upper = rep(1, N),
                     heq = function(w) return(sum(w) - 1))    # sum(w) = 1
cpu_time_mean_vol_gensolver <- Sys.time() - start_time
w_mean_vol_gensolver <- res$par
```

c. Resolution of problems via the universal successive mean–variance approximation method:

- maximum Sharpe ratio portfolio:

$$\lambda^k = \frac{(\boldsymbol{w}^k)^\mathsf{T}\boldsymbol{\mu} - r_\mathrm{f}}{(\boldsymbol{w}^k)^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w}^k}$$

```
library(quadprog)

w <- rep(1/N, N)  # initial point
start_time <- Sys.time()
for (k in 1:5) {
  lmd_k <- as.numeric(t(w) %*% mu / (t(w) %*% Sigma %*% w))
  # # Using CVXR:
  # w <- Variable(N)
  # prob <- Problem(Maximize(t(w) %*% mu - (lmd_k/2) * quad_form(w, Sigma)),
  #                 constraints = list(sum(w) == 1, w >= 0))
  # result <- solve(prob)
  # w <- as.vector(result$getValue(w))
  # Using quadprog:
  w <- solve.QP(Dmat = lmd_k * Sigma, dvec = mu,
           Amat = cbind(matrix(rep(1, N), ncol = 1), diag(N)),
           bvec = c(1, rep(0, N)),
           meq = 1)$solution
}
cpu_time_MSRP_universal <- Sys.time() - start_time
w_MSRP_universal <- w
```

- mean–volatility portfolio:

$$\lambda^k = \kappa / \sqrt{(\boldsymbol{w}^k)^\mathsf{T} \boldsymbol{\Sigma} \boldsymbol{w}^k}$$

```
w <- rep(1/N, N)  # initial point
start_time <- Sys.time()
for (k in 1:3) {
  lmd_k <- as.numeric(kappa/sqrt(t(w) %*% Sigma %*% w))
  # # Using CVXR:
  # w <- Variable(N)
  # prob <- Problem(Maximize(t(w) %*% mu - (lmd_k/2) * quad_form(w, Sigma)),
  #                 constraints = list(sum(w) == 1, w >= 0))
  # result <- solve(prob)
  # w <- as.vector(result$getValue(w))
  # Using quadprog:
  w <- solve.QP(Dmat = lmd_k * Sigma, dvec = mu,
           Amat = cbind(matrix(rep(1, N), ncol = 1), diag(N)),
           bvec = c(1, rep(0, N)),
           meq = 1)$solution
}
cpu_time_mean_vol_universal <- Sys.time() - start_time
w_mean_vol_universal <- w
```

d. Comparison of solutions and the computational cost:

```r
# Comparison of solutions for MSRP
round(cbind(w_MSRP_gensolver, w_MSRP_universal), digits = 3)
```

```
      w_MSRP_gensolver w_MSRP_universal
 [1,]            0.161            0.163
 [2,]            0.446            0.444
 [3,]            0.247            0.253
 [4,]            0.000            0.000
 [5,]            0.000            0.000
 [6,]            0.000            0.000
 [7,]            0.000            0.000
 [8,]            0.000            0.000
 [9,]            0.146            0.140
[10,]            0.000            0.000
```

```r
# Comparison of execution time
cat("\nCPU time for general solver:", cpu_time_MSRP_gensolver)
```

```
CPU time for general solver: 0.004403114
```

```r
cat("\nCPU time for universal algorithm:", cpu_time_MSRP_universal)
```

```
CPU time for universal algorithm: 0.002616882
```

```r
# Comparison of solutions for MSRP
round(cbind(w_mean_vol_gensolver, w_mean_vol_universal), digits = 3)
```

```
      w_mean_vol_gensolver w_mean_vol_universal
 [1,]                0.000                0.000
 [2,]                0.433                0.433
 [3,]                0.000                0.000
 [4,]                0.001                0.001
 [5,]                0.000                0.000
 [6,]                0.000                0.000
 [7,]                0.000                0.000
 [8,]                0.000                0.000
 [9,]                0.315                0.315
[10,]                0.251                0.251
```

```r
# Comparison of execution time
cat("\nCPU time for general solver:", cpu_time_mean_vol_gensolver)
```

```
CPU time for general solver: 0.004465818
```

```r
cat("\nCPU time for universal algorithm:", cpu_time_mean_vol_universal)
```

```
CPU time for universal algorithm: 0.002540827
```