Master-Thesis

Axel Roth

2022-09-05

Contents

Preface

Blah blah blah (soll vor dem TOC kommen denke ich)

Chapter 1

Abstract

Things about this thesis. why and what question should be answered. and what are the answers. (zusammenfassung)

Chapter 2

Software information and usage

wie ich das buch schreibe, R markodwn bookdown und so und welche versionen ich nutze

2.1 R-Version and Packages

2.2 Reproducibility

github und code im bookdown

2.3 R-functions

zb plotly_save

Chapter 3

Open Data Sources

To increase the reproducibility, all data is free and can be loaded with the function `getSymbols()` from the quantmod R-Package. There can be chosen between different data-sources like yahoo-finance (default), alpha-vantage, google and more.

3.1 R-Functions

The following functions are created to increase the simplicity of data-gathering with the quantmod R-package, that can be found in the directory named `R/` in the attached github repository.

3.1.1 `get_yf()`

This function is the main wrapper for gathering data with `getSymbols()` from yahoo-finance and transforms prices to returns with the `pri_to_ret()` function explained in 4.5.1. The output is a list that contains prices and returns as a xts object. The arguments that can be passed to `get_yf()`, are:

- `tickers`: Vector of symbols (asset identifiers e.g. "APPL", "GOOG", ...)
- `from ="2020-01-01"`: R-Date
- `to = "2021-01-01"`: R-Date
- `price_type = "close"`: Type of prices to gather (e.g. "open", "high", "low", "close", "adjusted")
- `return_type = "adjusted"`: Type of returns to gather (e.g. "open", "high", "low", "close", "adjusted")
- `print = F`: Should the function print the return of `getSymbols()`

3.1.2 `buffer()`

To make data reusable and decrease compiling time, this function saves data gathered with `get_yf()`. It takes an R expression, evaluates it and saves it in the `buffer_data/` directory with the given name. If this name already exists, it loads the R-object from the RData-files, without evaluating the expression. Forcing the evaluation and overwriting the existing RData-file can be done with `force=T`.

Chapter 4

Mathematical Fundations

This chapter provides an overview of the mathematical calculations and conventions used in this Thesis. It's important to note that most of the time mathematical formulas are written in matrix notation. In the majority of cases, this will result in a direct translation into R-code. All necessary assumptions needed for the modeled return structure are provided in this chapter to enable each reader to make sense of the stated formulas. It is crucial to note that reality is too complex and can only be partially modeled. Simplistic, basic models are employed that don't hold up in real-world situations, but these models or variations of them are frequently used in finance and have proven to be helpful. The complexity of solving advanced and basic models do not differ for the PSO, because the dimension of the objective function is based on the number of selectable elements, see chapter 6.

4.1 Basic Operators

A compendium, that compares commonly used mathematical symbols to R-code and its meanings, is listed in the table below:

Latex/Displayed	R-Code	Meaning
×	%*%	Matrix or Vector multiplication and Cross-Product
⊗	%*%	Outer Product
A^T	t(A)	Transpose of Matrix or Vector A
·	*	Scalar or elementwise Vector multiplication

4.2 Return Calculation

Any portfolio optimization strategy based on historical data must start with returns. These returns are calculated using adjusted closing prices, which show the percentage change over time. Adjusted closing prices are reflecting dividends and are cleaned of by stock splits and rights offerings. These Returns are essential for comparing assets and for analyzing dependencies.

4.2.1 Simple Returns

The default timeframe for all raw data in this thesis is one workday and only simple returns are used. Suppose there is a asset with prices P on workday t_i and following workday t_{i+1}, then follows that the simple return on t_{i+1} can be calculated as:

$$R_{i+1} = \frac{P_{t_{i+1}}}{P_{t_i}} - 1$$

4.3 Markowitz Modern Portfolio Theory (MPT)

In 1952, Harry Markowitz published his first ground-breaking work, which had a significant influence on modern finance, primarily by outlining the effects of diversification and efficient portfolios. The definition of an efficient portfolio is one that has either the maximum expected return for a given risk target or the minimum risk for a given expected return target. A simple quote to define diversification could be: “A portfolio has the same return but less variance than the sum of its parts”. This is true if the assets are not perfectly correlated, because bad and good movers can make up for each other which will reduce

the likelihood of extreme events. More specific information can be found at (Maringer, 2005).

4.3.1 Assumptions of Markowitz Portfolio Theory

The following list contains all Markowitz assumptions according to (Maringer, 2005):

- Perfect market without taxes or transaction costs.
- Short sales are disallowed.
- Assets are infinitely divisible.
- Expected Returns, Variances and Covariances contain all information.
- Investors are risk-adverse, they will only accept greater risk if they are compensated with a higher expected return.

The assumption that the returns are normally distributed is not required, but it will be assumed in this case to make the problem simpler. It is obvious that these assumptions are unrealistic in real-life. For more details regarding the requirements for utilizing other distributions, have a look at (Maringer, 2005).

4.4 Portfolio Math

Proofs for the fundamental calculations required for portfolio optimization as shown in (Zivot, 2021) will be provided in this section. The returns are presented differently than in most sources, because its the most common data-format used in practice. Suppose there are N assets that are described by a return vector R of random variables and a portfolio weight vector w, respectively:

$$R = \begin{bmatrix} R_1 & R_2 & \cdots & R_N \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}$$

In this thesis, each return is simplified to be normal distributed with $R_i = \mathcal{N}(\mu_i, \sigma_i^2)$. As a result, linear combinations of normally distributed random variables are jointly normal distributed and have a mean, variance, and covariance that can be used to fully describe them.

4.4.1 Expected Returns

The following formula can be used to get the expected returns of a vector with normally distributed random variables $R \in \mathbb{R}^{1 \times N}$:

$$\begin{aligned} E[R] &= \begin{bmatrix} E[R_1] & E[R_2] & \cdots & E[R_N] \end{bmatrix} \\ &= \begin{bmatrix} \mu_1 & \mu_2 & \cdots & \mu_N \end{bmatrix} = \mu \end{aligned}$$

and μ_i can be estimated in R with historical data and the formula for geometric mean returns (also called compound returns). The function to calculate the geometric mean returns from a xts-object can be found in 4.5.3.

4.4.2 Expected Portfolio Returns

The following equation can be used to get the linear combination of expected returns μ and a weighting vector w (e.g. portfolio weights):

$$\begin{aligned} \mu \times w &= \begin{bmatrix} E[\mu_1] & E[\mu_2] & \cdots & E[\mu_N] \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_N \end{bmatrix} \\ &= E[\mu_1] \cdot w_1 + E[\mu_2] \cdot w_2 + \cdots + E[\mu_N] \cdot w_N = \mu_P \end{aligned}$$

4.4.3 Covariance

The general formula of the covariance matrix $\sum$ of a random vector R with N normally distributed elements and $\sigma_{i,j}$ as correlation of two unique assets is described as:

$$\begin{aligned} Cov(R) &= E[(R-\mu)^T \otimes (R-\mu)] \\ &= \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,N} \\ \sigma_{2,1} & \sigma_2^2 & \cdots & \sigma_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{N,1} & \sigma_{N,2} & \cdots & \sigma_N^2 \end{bmatrix} \\ &= \sum \end{aligned}$$

and can be estimated in R with the base-function `cov()` and historical data.

4.4.4 Portfolio Variance

Let R be a random vector with N normally distributed elements and w a weighting vector. Suppose the covariance matrix $\sum$ of R is known, then the

variance of the linear combination of R can be calculated as:

$$\begin{aligned} Var(R \times w) &= E[(R \times w - \mu \times w)^2] \\ &= E[((R - \mu) \times w)^2] \end{aligned}$$

Since $(R - \mu) \times w$ is a scalar, it can be transformed from $((R - \mu) \times w)^2$ to $((R - \mu) \times w)^T \cdot ((R - \mu) \times w)$ and results in:

$$\begin{aligned} Var(R \times w) &= E[((R - \mu) \cdot w)^T \times ((R - \mu) \times w)] \\ &= E[(w^T \times (R - \mu)^T) \cdot ((R - \mu) \times w)] \\ &= w^T \times E[(R - \mu)^T \otimes (R - \mu)] \times w \\ &= w^T \times Cov(R) \times w \\ &= w^T \times \sum \times w \end{aligned}$$

The same holds for a estimation of $\sum$.

4.4.5 Portfolio Returns

Suppose there are N assets which create a portfolio with weights w at time t_0 and the portfolio should perform multiple timesteps till t_T without re-balancing. What are the portfolio returns on each timestep t_i? Its obvious that assets with a positive performance at the current timestep have a higher weight at the next timestep. This can be done by adjusting the weights after each timestep, dependent on the returns. The formula for holding a portfolio with weights $\sum w = 1$ and return matrix $R \in R^{T \times N}$, has the return $Z_i - 1$ on t_i with $i = 0, 1, \cdots, T$ for:

$$Z_i = \begin{cases} (1 + R_i) \cdot w & \text{if } i = 0 \\ (1 + R_i) \cdot \frac{Z_{i-1}}{\sum Z_{i-1}} & \text{if } i > 0 \end{cases}$$

This calculation of portfolio returns is implemented in the `calc_portfolio_returns()` function below.

4.5 R-Functions

asdad

4.5.1 `pri_to_ret()`

asdasdasd ((prices to returns))

4.5.2 ret_to_cumret()

asdasdasdasdasd ((returns to cumulated returns normalized to 100))

4.5.3 ret_to_geomeanret()

Geometric mean returns are a better estimator than the arithmetic mean returns, because they capture the exact mean price changes over a period of time. The variance estimated from daily returns is a daily variance, which is why the returns need to have the same timebase. This can be done by calculating geometric mean returns from multiple daily returns. Suppose there is an asset with returns $r_1 = 0.01$, $r_2 = 0.03$ and $r_3 = 0.02$ then follows that the geometric mean return r^{id} can be calculated as:

$$r^{id} = ((1 + r_1) \cdot (1 + r_2) \cdot (1 + r_3))^{1/3} - 1 = 0.01996732$$

And the benefit is that it's a daily mean return, which will produce exactly the same outcome as the real returns, that means:

$$(1 + r^{id})^3 = (1 + r_1) \cdot (1 + r_2) \cdot (1 + r_3)$$

This isnt the case for arithmetic mean returns. The general formula to calculate the mean geometric return from n days is:

$$r^{id} = (\prod_{i=1}^{n}(1 + r_i))^{\frac{1}{n}} - 1$$

and as R-code:

```
ret_to_geomeanret <- function(xts_ret){
  sapply((1+xts_ret), prod)^(1/nrow(xts_ret))-1
}
```

4.5.4 calc_portfolio_returns()

This is the implementation of a vectorial calculation of portfolio returns over multiple periods with a weighting vector `weights` at t_0 and no re-balancing:

```
calc_portfolio_returns <-
  function(xts_returns, weights, name="portfolio"){
  if(sum(weights)!=1){
    xts_returns$temp___X1 <- 0
    weights <- c(weights, 1-sum(weights))
  }
```

```
  res <- cumprod((1+xts_returns)) * matrix(
    rep(weights, nrow(xts_returns)), ncol=length(weights), byrow=T)
  res <- xts(
    rowSums(res/c(1, rowSums(res[-nrow(xts_returns),])))-1,
    order.by=index(xts_returns)) %>%
    setNames(., name)
  return(res)
}
```

This function has the same results as the `Return.portfolio()` function from the `PortfolioAnalytics` package.

Chapter 5

Activ vs Passiv Investing

The fundation of Asset Management

passiv vs activ studie https://www.scirp.org/journal/paperinformation.aspx?paperid=92983

gut gut file:///C:/Users/Axel/Desktop/Master-Thesis-All/Ziel%20was%20beantwortet%20werden%20soll/Quellen%20nur%20wichtige/Rasmussen2003_Book_QuantitativePortfolioOptimisat.pdf

Chapter 6

Challenges of Passiv Investing

This Chapter will analyse two common challenges of Passiv-Investing and creates simple use-cases to test the PSO. The first one is the mean-variance portfolio (MVP) from the modern portfolio theory of Markowitz which is simply said an optimal allocation of assets regarding risk and return. The second challenge is the index-tracking-problem which tries to construct a portfolio with minimal tracking error to a given benchmark.

6.1 Mean-Variance Portfolio (MVP)

Markowitz has shown that diversifying the risk on multiple assets will reduce the overall risk of the portfolio. This result was the beginning of the widely used modern portfolio theory which uses mathematical models to archive portfolios with minimal variance for a given return target. All these optimal portfolios for a given return target are called efficient and create the efficient frontier.

6.1.1 MVP

Let there be N assets and its returns on T different days which creates a return matrix $R \in \mathbb{R}^{T \times N}$. Each element $R_{t,i}$ contains the return of the i-th asset on day t. The covariance matrix of the returns is $\sum \in \mathbb{R}^{N \times N}$ and the expected returns are $\mu \in \mathbb{R}^N$. The MVP with risk aversion parameter $\lambda \in [0, 1]$ like shown in (Maringer, 2005) can be formalized as follows:

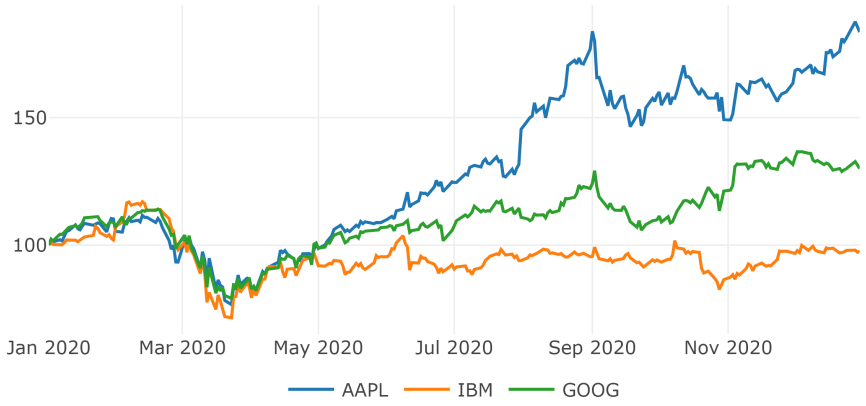
$$\underset{w}{minimize} \quad \lambda \, w^T \sum w - (1 - \lambda) \, \mu^T w \tag{6.1}$$

The risk aversion parameter λ defines the trade-off between risk and return. With $\lambda = 1$, the minimization problem only contains the variance term and so on results in a minimum variance portfolio and $\lambda = 0$ transforms the problem into a minimization of the negative expected returns, which results in a maximum return portfolio. All possible portfolios created by $\lambda \in [0, 1]$ define the efficient frontier.

6.1.2 MVP example

All possible MVPs combined create the efficient frontier, that is analyzed in this section without going into the details of its calculation. This example uses three assets (equitys: IBM, Google, Apple) and calculates the solution of the MVP for each λ. First of all are the daily returns of these three assets are loaded from the year 2020.

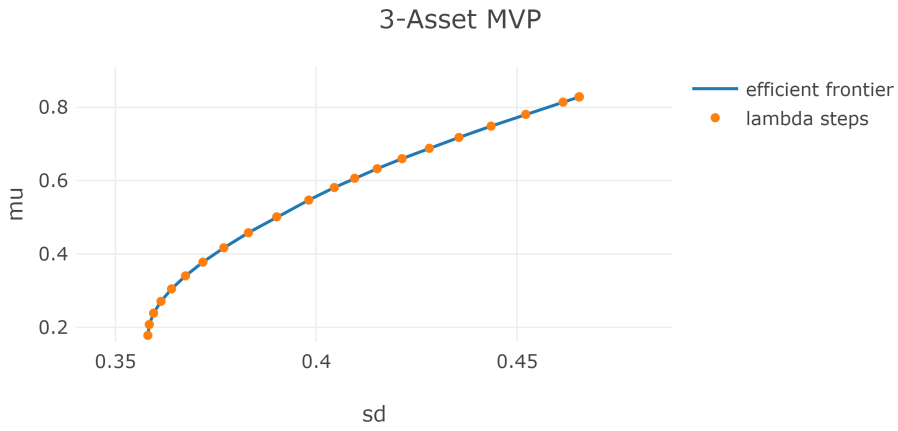
The cummulated daily returns are:



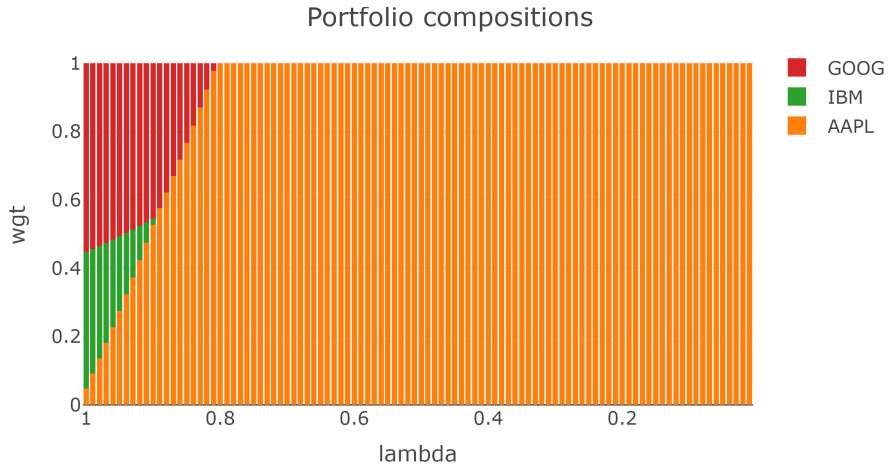
The expected daily returns and the covariance matrix for the 3 assets can be estimated with the formulas from chapter 4:

These are all the necessary data to solve the MVP with $\lambda \in \{0.01, 0.02, ..., 0.99, 1\}$. All 100 portfolios are calculated by solving a quadratic minimization problem with long only constraint and the weights should sum up to 1.

The resulting daily returns and standard deviation are transformed to annual returns and standard deviation and are plotted to create the efficient frontier:



The portfolio compositions for each λ's are:



6.2 Index-Tracking Portfolio (ITP)

Indices are baskets of assets that are used to track the performance of a specific asset group. For example, the well-known Standard and Poor's 500 index (short: S&P 500), tracks the 500 largest companies in the United States. Indices are not purchasable and only serve to visualize the performance of there asset group, without incurring transaction costs. Such indices, or a combination of indices, are used by asset managers as benchmarks to compare the performance of their funds. Each fund has its own benchmark, which contains roughly the same assets that the manager could purchase. If the fund underperforms its benchmark, it may be an indication that the fund manager made poor decisions. That is why fund managers strive to outperform their benchmarks through carefully

chosen investments. The past has proven that this is rarely achieved with active management after costs (Desmond Pace and Grima, 2016). This resulted in a growing popularity of passive managed funds with its goal to track there benchmarks as close as possible. It can be accomplished through either full or sparse replication. A full replication is a portfolio that has all assets contained in the benchmark, with the same weights. The resulting performance with neglecting transaction costs, would be exactly the benchmark performance. The first problem is that a benchmark can contain inliquid or not purchasable assets. The second problem would be the weighting scheme of indices, because they are often weighted by its market capitalization which will change daily. This would result in daily work of rebalancing and increasing transaction costs to mimic the benchmarks performance as close as possible. To prevent this, sparse replications are used, that contain a fraction of its benchmarks assets. To do so, the portfolio manager must define his benchmark, which should overlap with his fund's investment universe. Following that, he will reduce this universe using the investors constraints and availability, to create a pool of possible assets. A example pool to track the S&P 500 could consist of the top hundred highest weighted assets in the S&P 500. Now is the time to optimize a portfolio with the goal of matching the benchmark performance and taking into account the investors constraints like ratings, asset sectors and more. Typically, this is accomplished by lowering the variance between the portfolio and benchmark returns:

$$minimize \quad Var(r_p - r_{bm})$$

To obtain the portfolio weights w, its necessary to substitute r_p as shown below:

$$r_p = R \times w$$

The Variance is then solved up until a quadratic problem dependent on the portfolio weights w is represented:

$$\begin{aligned} Var(r_p - r_{bm}) &= Var(R \times w - r_{bm}) \\ &= Var(R \times w) + Var(r_{bm}) - 2 \cdot Cov(R \times w, r_{bm}) \end{aligned}$$

Now the three terms can be solved, beginning with the easiest.

$$Var(r_{bm}) = \sigma_{bm}^2 = constant$$

The variance of the portfolio can be solved with 4.4.4:

$$Var(R \times w) = w^T \times Cov(R) \times w$$

And the last term can be solved the same way as in (Zivot, 2021):

$$\begin{aligned} Cov(A \times a, b) &= Cov(b, A \times a) \\ &= E[(b - \mu_b)(A \times a - \mu_A \times a)] \\ &= E[(b - \mu_b)(A - \mu_A) \times a] \\ &= E[(b - \mu_b)(A - \mu_A)] \times a \\ &= Cov(A, b) \times a \end{aligned}$$

This results in the final formula of the ITP:

$$\begin{aligned} Var(r_p - r_{bm}) &= Var(R \times w - r_{bm}) \\ &= Var(R \times w) - 2 \cdot Cov(R \times w, r_{bm}) + Var(r_{bm}) \\ &= w^T \times Cov(R) \times w - 2 \cdot Cov(r_{bm}, R)^T \times w + \sigma_{bm}^2 \end{aligned}$$

The minimization problem of the ITP in the general structure needed by many optimizers is:

$$min(\frac{1}{2} \cdot b^T \times D \times b - d^T \times b)$$

Minimization problems can ignore constant terms and global stretching coefficients and still find the same minimum. This results in the general substitution of the ITP as follows:

$$D = Cov(R)$$

and

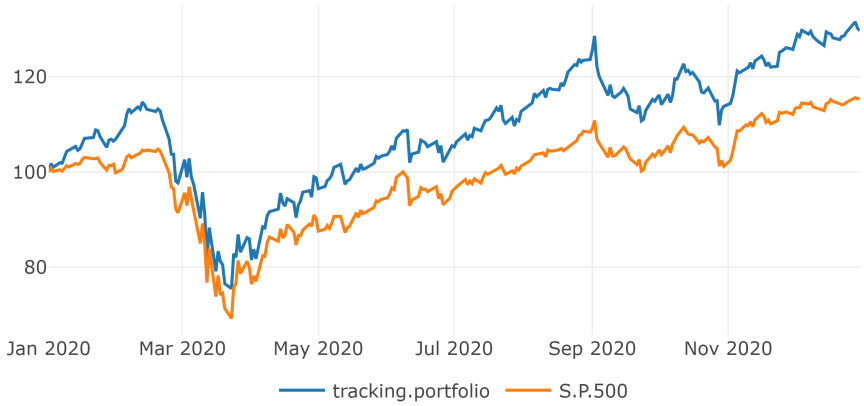
$$d = Cov(r_{bm}, R)$$

Its possible to add some basic constraints like in the MVP, to sum up the weights to 1 and being long only.

6.2.1 Example ITP

This example will show the results of tracking the S&P 500 with a tracking portfolio that can only invest in IBM, Apple and Google. The time frame is the year 2020 and the goal is to minimize the variance between the portfolios and benchmarks historical returns.

```
##       AAPL        IBM       GOOG
## 0.2681058 0.4040352 0.3278591
```



Chapter 7

Analytic Solver for Quadratic Programming Problems

The benefits and drawbacks of analytic solvers for quadratic programming problems will be discussed in this chapter. It would go beyond the scope of this thesis to explain the mathematical underlying principles of how a solver addresses quadratic problems, only the applications and analysis are discussed. The foremost reason of addressing quadratic programming solvers is to use it as a benchmark for the PSO.

7.1 Quadratic Programming (QP)

A quadratic program is a minimization problem of some function that returns a scalar and consists of an quadratic term and an linear term, dependent on the variable of interest. Additionally can the problem be constrained by several linear inequalities that restrict the solution. The general formulation used, is to find x that minimizes the following problem:

$$min\ \frac{1}{2} \cdot x^T \times D \times x - d^T \times x$$

and holds under the linear constraints:

$$A^T \times x >= b_0$$

Some other sources notate the problem with different signs or coefficients that are all exchangeable with the above stated problem. Additionally has the

problem above the same notation that is used in the R-Package `quadprog` which will reduce substitution efforts. All modern programming languages do have many solvers for quadratic problem. They differ mostly in computational time on specific problems and the requirements. Some commercial QP-solvers do additionally accept more complex constraints, like absolute (e.g. $|A^T \times x| >= a_0$) or mixed-integer (e.g. $x \in \mathbb{N}$). Specially the mixed-integer constraint problems will result in a enormously increase of memory.

7.2 QP Solver from quadprog

The most common free QP-solver used in R is from the package quadprog which consists of a single function named `solve.QP()`. Its implementation routine is the dual method of Goldfarb and Idnani that was published in (Goldfarb and Idnani, 1982) and (Goldfarb and Idnani, 1983). It uses the above stated QP with the requirement that D needs to be a symmetric positive definite matrix. It means that $D \in R^{N \times N}$ and $x^T Dx > 0 \; \forall \; x \in R^N$ which is equivalent to, all eigenvalues are bigger than null. In most cases this is not achieved by using the estimation of the covariance matrix $\sum$, but its possible to find the nearest positive definite matrix of $\sum$ with the function `nearPD()` from the Matrix R-Package. The error that occurs is printed and often do not exceed a percentage change of elements above $10^{-15}\%$, which is negligible for the context of this thesis. The `solve.QP()` function for a N dimensional vector of interest, has the following arguments, that can as well be found in the above stated formulation of a QP:

- `Dmat`: Symmetric positive definite matrix $D \in R^{N \times N}$ of the quadratic term.
- `dvec`: Vector $d \in R^N$ of the linear term
- `Amat`: Constraint matrix A
- `bvec`: Constraint vector b_0
- `meq = 1`: means that the first row of A is treated as equality constraint

The return of `solve.QP()` is a list and contains among other things the following attributes of interest:

- `solution`: Vector containing the solution x of the quadratic programming problem. (e.g. portfolio weights)
- `value`: Scalar, the value of the quadratic function at the solution

7.3 Example: Solving MVP with `solve.QP()`

This section provides insights into the effects of diversification and the use of `solve.QP`, by creating ten different efficient frontiers from a pool of ten assets.

Each efficient frontier $i \in \{1, 2, \cdots, 10\}$ consists of $N_i = i$ assets and is created by adding the asset with the next smallest variance first. After loading returns for ten of the biggest equity's in the US market, the variance is calculated to arrange all columns ascending by its variance, like shown in the code bellow:

```
returns_raw <- buffer(
  get_yf(
    tickers = c("IBM", "GOOG", "AAPL", "MSFT", "AMZN",
                "NVDA", "JPM", "META", "V", "WMT"),
    from = "2016-01-01",
    to = "2021-12-31"
  )$returns,
  "AS_10_assets"
)

# re-arrange: low var first
vars <- sapply(returns_raw, var)
returns_raw <- returns_raw[, order(vars, decreasing = F)]
```

The next step is to create a function `mvp()` that has the arguments `return` and `lambda`. It calculates the expected returns `mu` and the estimated positive definite covariance `cov`. Afterwards it solves a MVP with the constraints $\sum w = 1$ and $w >= 0$, which returns key features `mu`, `var` and `composition` of the portfolio.

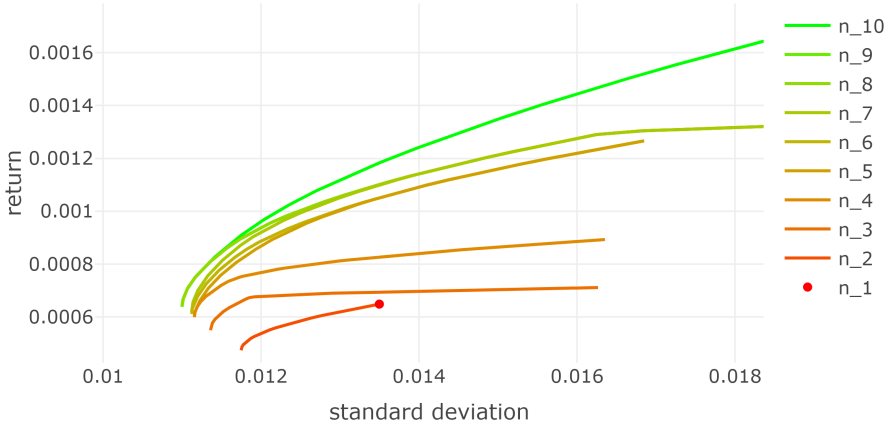
```
mvp <- function(returns, lambda){
  tc <- tryCatch({
    mu <- sapply((1+returns), prod)^(1/nrow(returns))-1

    cov <- as.matrix(nearPD(cov(returns))$mat)

    mat <- list(
      Dmat = lambda * cov,
      dvec = (1-lambda) * mu,
      Amat = t(rbind(
        rep(1, ncol(returns)), # sum up to 1
        diag(1, nrow=ncol(returns), ncol=ncol(returns)) # long only
      )),
      bvec = c(
        1, # sum up to 1
        rep(0, ncol(returns)) # long only
      ),
      meq = 1
    )

    qp <- solve.QP(
```

```
      Dmat = mat$Dmat, dvec = mat$dvec,
      Amat = mat$Amat, bvec = mat$bvec, meq = mat$meq
    )

    res <- list(
      "mu" = mu %*% qp$solution,
      "var" = t(qp$solution) %*% cov %*% qp$solution,
      "composition" = setNames(qp$solution, colnames(returns))
    )
    TRUE
  }, error = function(e){FALSE})


  if(tc){
    return(res)
  }else{
    return(list(
      "mu" = NA,
      "var" = NA,
      "composition" = NA
    ))
  }
}
```

Each $\lambda \in \{0.01, 0.02, \cdots, 1\}$ and each combination of ascending number of assets produces one portfolio that can be generated with two for loops.

```
df <- data.frame("index"=1, "var"=as.numeric(var(returns_raw[, 1])), "return" = prod(1+retu
for(i in 2:ncol(returns_raw)){
  returns <- returns_raw[, 1:i]
  for(lambda in seq(0.01, 1, 0.01)){
    res <- mvp(returns, lambda)

    df <- rbind(df, data.frame("index"=i, "var"=res$var, "return" = res$mu))
  }
}
```

The result gets filtered and names are added that represent the number of assets. Now can the plot be generated:



It can be seen, that each asset added results in a minimum variance portfolio with smaller or equal standard deviation. Nonetheless that it started with the asset that has the smallest standard deviation of 0.0134977. This is the effect of diversification mentioned by Markowitz.

7.4 Example: Solving ITP with `solve.QP()`

This example will analyse how many assets are needed to minimize the variance between the replication and the S&P 500 historical returns from `2016-01-01` to `2021-12-31`. The constraints are set to be long only and the weights should sum up to one. To step-wise reduce the number of assets, the five assets with the least weights get discarded and acts as the new asset pool for the next replication, until only five assets are left. At first the needed data can be downloaded with existing functions from the `R/` directory. The function `get_spx_composition()` uses web scrapping to read the constituents of wikipedia and transforms it in monthly compositions of the S&P 500. The pool is created by all assets that are present in the last month of the timeframe, reduced by assets with missing values. The code below, loads returns of all assets in the pool and the S&P 500:

```
from <- "2016-01-01"
to <- "2021-12-31"

spx_composition <- buffer(
  get_spx_composition(),
  "AS_spx_composition"
)


pool_returns_raw <- buffer(
```

```
  get_yf(
    tickers = spx_composition %>%
      filter(Date<=to) %>%
      filter(Date==max(Date)) %>%
      pull(Ticker),
    from = from,
    to = to
  )$returns,
  "AS_sp500_assets"
)
pool_returns_raw <-
  pool_returns_raw[, colSums(is.na(pool_returns_raw))==0]


bm_returns <- buffer(
  get_yf(tickers = "%5EGSPC", from = from, to = to)$returns,
  "AS_sp500"
) %>% setNames(., "S&P 500")
```

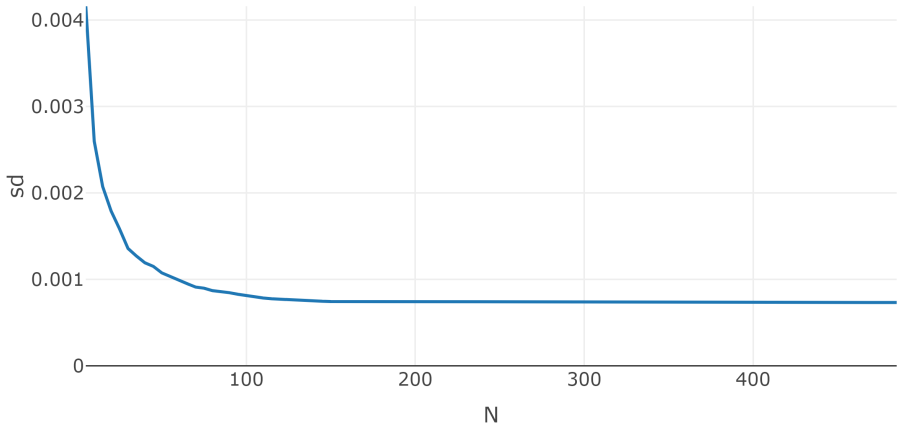
Now is the needed data present and the function for the ITP can be created. It needs `pool_returns` with variable number of columns and the single column matrix `bm_returns`.

```
itp <- function(pool_returns, bm_returns){
  mat <- list(
    Dmat = cov(pool_returns),
    dvec = cov(pool_returns, bm_returns),
    Amat = t(rbind(
      rep(1, ncol(pool_returns)), # sum up to 1
      diag(1,
           nrow=ncol(pool_returns),
           ncol=ncol(pool_returns)) # long only
    )),
    bvec = c(
      1, # sum up to 1
      rep(0, ncol(pool_returns)) # long only
    ),
    meq = 1
  )

  qp <- solve.QP(
    Dmat = mat$Dmat, dvec = mat$dvec,
    Amat = mat$Amat, bvec = mat$bvec, meq = mat$meq
  )
```

```
  res <- list(
    "var" = as.numeric(
      var(pool_returns %*% qp$solution - bm_returns)),
    "solution" = setNames(qp$solution, colnames(pool_returns))
  )
}
```

The replication and successive discarding of assets can begin. The results are saved in `res` and used to plot the results.

```
res <- NULL
for(i in rev(seq(5, ncol(pool_returns_raw), 5))){
  if(i==ncol(pool_returns_raw)){
    temp <- itp(pool_returns_raw, bm_returns)
  }else{
    temp <- itp(pool_returns_raw[, names(sort(temp$solution, decreasing = T)[1:i])], bm_retu
  }
  res <- rbind(res, data.frame("N"=i, "var"=temp$var, "sd"=sqrt(temp$var), row.names = NULL)
}
```



It can be seen that the standard deviation stagnates at about $N = 100$. This results in the conclusion that a sparse replication with hundred assets is, in this specific case, sufficient to track the historical S&P 500 performance of the timeframe.

Chapter 8

Particle Swarm Optimization (PSO)

The PSO was developed by J. Kennedy as a Global Optimization method based on Swarm Intelligence and was introduced to the public in 1995 by Eberhart and Kennedy (James Kennedy, 1995). The initial PSO should resemble a flock of birds, that is flying through the sky without collisions. That is why its first applications were found in particle physics, to analyse moving particles in high dimensional spaces, which is resembled in the name Particle. Afterwards it was adapted in Evolutionary Computation to exploit a set of potential solutions in high dimensions and find the optima by cooperation with other particles in the swarm (Konstantinos Parsopoulos, 2002). The benefits compared to some other Global Optimization methods are the fact that no gradient information is needed. It can find the optimum by considering only the result of the function that needs to be optimized. That means that the function can be arbitrarily complex and its still possible to reach the global optimum. Other benefits are the low computational costs, because only basic mathematical operators are used.

8.1 The Algorithm

Each Particle d with position x_d moves in the search space R^N and has its inherent velocity v_d and remembers its previous best position P_d. After each iteration the velocity changes into the direction of its inherent velocity, its best previous position and the global best position p_g of all particles. A position change from i to $i + 1$ can be calculated by the following two equations (Konstantinos Parsopoulos, 2002):

$$v_d^{i+1} = wv_d^i + c_p r_1^i (P_d^i - x_d^i) + c_g r_2^i (p_g^i - x_d^i)$$
$$x_d^{i+1} = x_d^i + v_d^{i+1}$$

with r_1 and r_2 being uniformly distributed random numbers in [0, 1]. The cognitive parameter c_p acts as the weighting of the direction to its previous best position of the particle. On the contrary is the social parameter c_g, which is a weighting to the direction of the global best position. The inertia weight w is crucial for the convergence behavior by remembering a part of its previous trajectory. A study examined in (Konstantinos Parsopoulos, 2002) showed that these parameters can be set to $c_p = c_g = 0.5$ and w should decrease from 1.2 to 0. However, some problems do benefit from more fine-tuning of these parameters. To provide a effortless translation to code, the formula above can be stated for $d = 1, 2, \cdots, D$ particles in the following matrix notation:

$$V^{i+1} = w \cdot V^i + c_1 \cdot r_1^i \cdot (P^i - X^i) + c_2 \cdot r_2^i \cdot (p_g^i - X^i)$$
$$X^{i+1} = X^i + V^{i+1}$$

with current positions $X \in R^{N \times D}$, current velocity's $V \in R^{N \times D}$, previous best positions $P \in R^{N \times D}$ and the global best position $p_g \in R^N$. The parameters w, c_p, c_g, r_1 and r_2 are stile scalars.

8.2 `pso()` Function

A general purpose PSO function is created in this section by following the structure of other optimization heuristics in R, specially the existing PSO implementation from the R-Package `pso`. The center of everything is a objective function `fn()` which will return a scalar that needs to be minimized. The function itself needs mainly a vector `pos` that describes the position of a particle (e.g. weights). The other main parameters for the PSO function are `par`, which is one position of a particle, that is used to derive the dimension of the problem and used as first position of one particle. The argument can contain only `NA`'s which results in completely randomized starting positions. The last two arguments needed are `lower` and `upper` bounds (e.g. weights bigger than 0 and smaller than 1). All other parameters have default values that can be overwritten by passing a list named `control`. The resulting structure is:

```
pso <- function(
    par,
    fn,
```

```
    lower,
    upper,
    control = list()
  ){

}
```

Before the main data-structure can be initialized, its necessary to create some example inputs for the `pso()` function like below:

```
par <- rep(NA, 2)
fn <- function(x){return(sum(abs(x)))}
lower <- -10
upper <- 10
control = list(
  s = 10, # swarm size
  c.p = 0.5, # inherit best
  c.g = 0.5, # global best
  maxiter = 100, # iterations
  w0 = 1.2, # starting inertia weight
  wN = 0, # ending inertia weight
  save_traces = F # save more information
)
```

Now is the time to initialize the random positions `X`, its fitness `X_fit` and its random velocity's `V` with the function `mrunif()`, which will create a matrix from uniformly distributed random numbers between `lower` and `upper`:

```
X <- mrunif(
  nr = length(par), nc=control$s, lower=lower, upper=upper
)
if(all(!is.na(par))){
  X[, 1] <- par
}
X_fit <- apply(X, 2, fn)
V <- mrunif(
  nr = length(par), nc=control$s,
  lower=-(upper-lower), upper=(upper-lower)
)/4
```

The velocity's are compressed with factor 4 to start with a maximal movement from a quarter of the space in each axis. The personal best positions `P` are the same as `X` and the global best position is the position with the smallest fitness:

```
P <- X
P_fit <- X_fit
p_g <- P[, which.min(P_fit)]
p_g_fit <- which.min(P_fit)
```

The needed data-structure is present and the optimization can start by calculating the new velocity's and the transformation of the old positions. If particles have left the valid space, they get pushed back to the border. Afterwards the fitness is calculated and the personal best and global best positions are saved, if they have improved.

```
trace_data <- NULL
for(i in 1:control$maxiter){
  # move particles
  V <-
    (control$w0-(control$w0-control$wN)*i/control$maxiter) * V +
    control$c.p * runif(1) * (P-X) +
    control$c.g * runif(1) * (p_g-X)
  X <- X + V

  # set velocity to zeros if not in valid space
  V[X > upper] <- 0
  V[X < lower] <- 0

  # move into valid space
  X[X > upper] <- upper
  X[X < lower] <- lower

  # evaluate objective function
  X_fit <- apply(X, 2, fn)

  # save new previews best
  P[, P_fit > X_fit] <- X[, P_fit > X_fit]
  P_fit[P_fit > X_fit] <- X_fit[P_fit > X_fit]

  # save new global best
  if(any(P_fit < p_g_fit)){
    p_g <- P[, which.min(P_fit)]
    p_g_fit <- min(P_fit)
  }

  if(control$save_traces==TRUE){
    trace_data <- rbind(trace_data, data.frame("iter"=i, t(X)))
  }
}
```

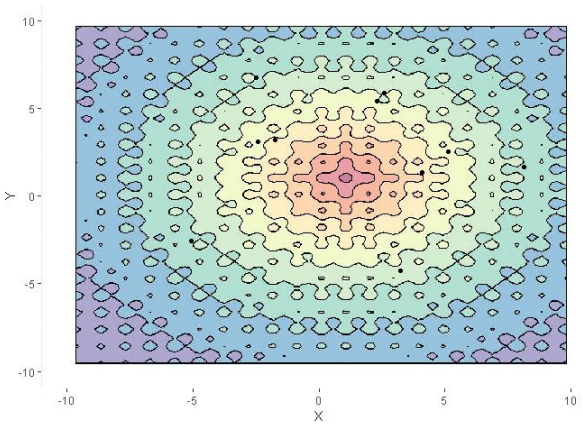
The best fitness after 100 iterations is 0.0000009 and the best possible solution is at 0.

8.3 Animation 2-Dimensional

This section provides insights into the behavior of the PSO by visualizing multiple iterations in a GIF. The GIF only works in Adobe Acrobat DC or in the Markdown/HTML Version of this thesis. The amazing template of the animation is inspired by R'tichoke. The PSO core from the chapter above was used to finish the `pso()` function and is tested here with seed 0. The function `fn` to evaluate can be found in R'tichoke.

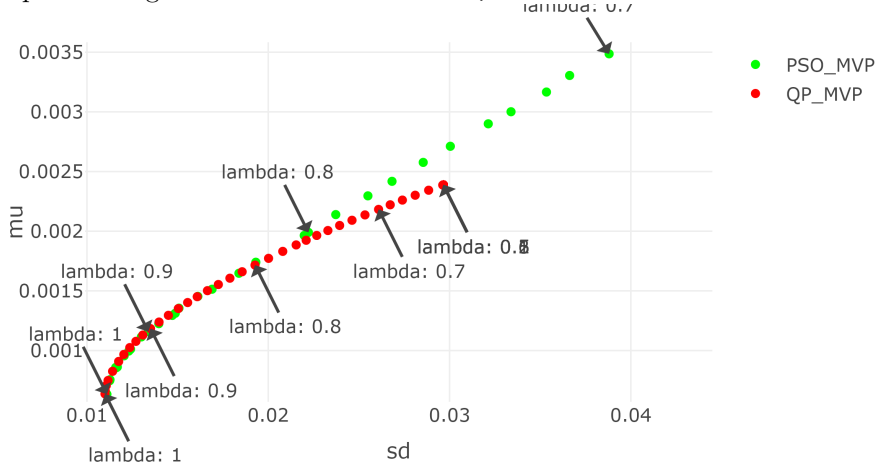
```
set.seed(0)

fn <- function(pos){
  -20 * exp(-0.2 * sqrt(0.5 *((pos[1]-1)^2 + (pos[2]-1)^2))) -
  exp(0.5*(cos(2*pi*pos[1]) + cos(2*pi*pos[2]))) +
  exp(1) + 20
}

res <- pso(
  par = rep(NA, 2),
  fn = fn,
  lower = -10,
  upper = 10,
  control = list(
    s = 10,
    maxiter = 30,
    w0 = 0.8,
    save_traces = T
  )
)
```

The function `fn` has many local minima and a global minima at $(1, 1)$ with value 0. The background-color-scale ranges from 0 as red to 20 as purple. The PSO has 10 particles, iterates 30 times with inertia weight decreasing from 0.8 to 0. The iterations are visualized in the GIF below:



8.4 Example MVP

This example uses the `solve.QP` approach from 7.3 with ten assets as benchmark. Recap, the goal is to create a MVP from ten of the biggest american equity's between 2016 and 2021 for each possible λ. The PSO has 100 particles and 200 iterations for each lambda. The key features of all portfolios generated with the `solve.QP` versus the PSO is shown below:



It can be seen that the analytical approach and the PSO approach have a tiny difference for $\lambda > 0.9$ and a steadily increased difference otherwise. This indicates that minimum variance portfolios are more stable to generate with the PSO.

8.5 Example ITP

The same ITP solved with `solve.QP()` in 7.4 is used as benchmark for the PSO. Recap, the goal is to create a portfolio that minimizes the variance in the returns of it self and the S&P 500 between 2016 and 2021. The pool of assets contain all assets that are present at 2021 and have no missing values. The constraints are long only and the weights should sum up to one. The parameters for the PSO are a swarm size of 100, 100 iterations, inertia weight starts by 0.9, the upper bound is 0.1 and one starting position is the null vector. The PSO was run ten times and the aggregated best and mean runs are compared to the `solve.QP()` approach in the table below for seed 0:

type	sd	var	fitness	constraint break
ITP_QP	0.0007	0,0000005	-0.0000666	0
ITP_PSO_best	0.0033	0,0000106	-0.0000616	0
ITP_PSO_mean	0.0036	0,0000126	-0.0000605	1.18805e-13

It can be seen that a sufficient fitness was achieved in all PSO runs with negligible constraint breaks, but much more computing time needed.

8.6 Pros and Cons

8.7 Functions

Bibliography

Desmond Pace, J. H. and Grima, S. (2016). Active versus passive investing: An empirical study on the us and european mutual funds and etfs.

Goldfarb, D. and Idnani, A. (1982). Dual and primal-dual methods for solving strictly convex quadratic programs.

Goldfarb, D. and Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs.

James Kennedy, R. E. (1995). Particle swarm optimization.

Konstantinos Parsopoulos, M. N. V. (2002). Recent approaches to global optimization problems through particle swarm optimization.

Maringer, D. (2005). *Portfolio Management with Heuristic Optimization.*

Zivot, E. (2021). *Introduction to Computational Finance and Financial Econometrics with R.*