CS/ECE/ISyE 524 — Introduction to Optimization — Summer 2020

On Efficiency and Robustness in Portfolio Optimization

Albert Dorador (albert.dorador@wisc.edu)

Table of Contents

1. Introduction

Portfolio optimization is the process of finding the best possible stock portfolio i.e. the best way to allocate a budget among different possible financial assets, according to some objective. The objective typically maximizes factors such as expected return, and minimizes costs like financial risk.

The above optimization problem (maximizing expected return with minimum risk) is the backbone of Modern portfolio theory, which was introduced in a 1952 essay by Harry Markowitz [1] (further elaborated later in 1959 [2]), who won a Nobel Prize in 1990 for these ideas.

The world's fascination by Markowitz ideas still continues today and many different variations of his model have been proposed over the last few decades trying to correct some of its drawbacks.

A recent paper by Dai and Wang [3] studies one of the main drawbacks that the Markowitz model has: the sensitivity of optimal solutions to changes in the model parameters. In this project we will study this issue as well, and we will propose a **simple way to improve the robustness** of our solution (with a caveat).

Another issue that used to be a significant drawback of the original Markowitz model was the quadratic nature of the model: a Quadratic program may be considerably harder to solve than a Linear one, and a few decades ago the computational constraint was a factor that limited the model's applicability in practice. Konno and Yamazaki [4] study this issue and propose a way to linearize the problem while providing a comparable performance. Hence, at least since the 1990's, portfolio optimization is no longer (exclusively) a quadratic optimization problem, but linear. We will also propose an **alternative linearization** to the quadratic portfolio optimization model, based on maximum drawdown (defined in subsequent sections). Moreover, we will present a Mixed-Integer Linear Programming **(MILP) variation** of our new model to achieve a more refined - and robust - solution at the expense of added problem complexity (in theory). But, as we will see, in practice the MILP formulation of our linearized portfolio optimization program is the fastest model of all the models surveyed in this project.

Besides the above topics, we will discuss the original Markowitz model and two well-known variations. We will present a variation that attempts to simultaneously optimize the expected return and the risk. Notions of **Pareto optimality** will be discussed and a simple **heuristic to find the 'best' Pareto optimal portfolio** in a less arbitrary manner will be considered, as an extension to the model seen in class.

In addition, we will conduct an **empirical analysis** both in-sample and out-of-sample that compares the performance of the Markowitz variations and our two proposed portfolio optimization models, as well as testing how **sensitive** the optimal solutions of each model are to perturbations in the parameter values.

Finally, we will outline some extensions to the model we propose and possible **areas of future research**.

We use recent market data on about 400 US stocks publicly available at Yahoo Finance (https://finance.yahoo.com/) and automatically downloaded through an in-house Python script. Average return vector and covariance matrix are then computed from these data. The Python script and all the necessary data are included as separate attachments.

With larger and larger data banks, and larger and larger risks, finding **faster and more robust** ways to optimize portfolios is more important than ever. We hope that this project may serve as a catalyst of further research that will lead to safer and more efficient portfolio selection, for the benefit of all market participants.

Note: 'variance' and 'standard deviation' are used interchangeably (mutatis mutandis) in this project, since variance is a monotonically increasing function of standard deviation. Similarly, 'average' and 'expected' return are used interchangeably, since the mean sample return is an unbiased estimator of the expected return.

2. Mathematical models

Back to index

In all deterministic portfolio optimization models the general **assumption** is that parameter values are correctly observed (i.e. sample values correspond to true population values) and are fixed, at least to a relevant extent and for sufficiently long. These assumptions are of course naive. However, improving on these assumptions would require much more advanced tools, among them **Machine Learning** and **stochastic programming**. Moreover, it is not clear how much it would be gained by doing that since, at the core, portfolio optimization is about choosing the assets that will statistically perform best jointly, but this inevitably has a component of 'forecasting the future' which, in finance, is an impossible task in abstract, and some assumptions must be made (e.g. the distribution of returns stays constant during the investment horizon, or at most it changes according to a known model).

Having said that, there is of course room for improvement and some relaxation of the assumptions, particularly in the form of using stochastic programming, is likely to lead to an improvement in out-of-sample performance.

2.1 Model 1: Classical Markowitz model

Back to index

This is the model that Markowitz describes in [1] and [2], dating back to 1950's.

$$
\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} \sigma_{i,j} x_i x_j & \\
\text{subject to:} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho & (1) \\
& \sum_{i=1}^{n} x_i = 1 & (2) \\
& 0 \leq x_i \leq 1 \qquad i = 1, \ldots, n & (3)
\end{aligned}
$$

Where the decision variable x_i is the proportion of the funds allocated to asset i, $\sigma_{i,j}$ is the (daily) covariance between asset i and asset j, r_i is the expected return of asset i (computed via the Method of Moments i.e. the arithmetic mean over the sample period), and ρ is a minimum (expected) return that is required per trading day.

The objective function seeks to minimize the standard deviation of the portfolio by considering the variance of each asset as well as its covariance with respect to the other assets in the sample.

Constraint (1) ensures that the optimal asset allocation achieves a minimum acceptable expected return.

Constraint (2) enforces that exactly 100% of the funds are allocated to some combination of assets.

Constraint (3) is a set of n box constraints that ensures each asset is assigned a non-negative proportion of the funds, up to a maximum of 100%.

Therefore, this model has a quadratic objective function (since the decision variables are squared) and a set of linear constraints, so it is a QP.

Since the covariance matrix is, by definition, positive semidefinite, then: this is a **convex quadratic program**, the feasible set is a polyhedron, the solution lies either on the boundary or the interior of the feasible set, and it is relatively easy to solve (compared to a quadratic program with a non positive definite paramater matrix in the objective function).

In that direction, it is worth pointing out that **Slater's conditions** are satisifed:

Let $x^T \Sigma x$ be the matrix form of the objective, where Σ is the covariance matrix.

- The program is convex: $x^T \Sigma x$ is convex since Σ is positive semi-definite, and all constraints are affine, and therefore convex (and concave too)
- The program is strictly feasible: let $x_i = 1$, for a given stock i with an expected return strictly above ρ (there must exist at least 1 such stock, otherwise the program is infeasible); then the inequality constraint is satisfied strictly, and the equality constraint is also satisfied

Therefore, **Strong Duality** holds and, since we know a finite p^* exists (since this is a convex quadratic program), then we are guaranteed a finite d^* exists too and $p^* = d^*$ (where p^* and d^* are the optimal objective values of the Primal and the Dual, respectively).

In our implementation this model has about 400 continuous variables and 400 constraints (including box contraints).

2.2 Model 2: Reverse Markowitz model

Back to index

This is the most basic variation of the classical Markowitz model. Not very popular neither in academia nor industry. One possible reason is that it is much more arbitrary to set a maximum standard deviation than a minimum return (for which you have clear minimum acceptable thresholds like bond yields of maturities similar to one's sample horizon). Furthermore, like in our particular case, the data collected does not allow us to require a very low daily portfolio standard deviation (otherwise the optimization problem is infeasible: the feasible region is empty).

$$
\begin{aligned}
\max_{x \in \mathbb{R}^n} \quad & \sum_{i=1}^{n} r_i x_i \\
\text{subject to:} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} \sigma_{i,j} x_i x_j \leq \sigma_0 \qquad (1) \\
& \sum_{i=1}^{n} x_i = 1 \qquad (2) \\
& 0 \leq x_i \leq 1 \qquad i = 1, \ldots, n \qquad (3)
\end{aligned}
$$

Where the decision variable x_i is the proportion of the funds allocated to asset i, $\sigma_{i,j}$ is the covariance between asset i and asset j, r_i is the expected daily return of asset i (computed via the Method of Moments i.e. the arithmetic mean over the sample period), and σ_0 is a maximum acceptable daily standard deviation over the time horizon of the sample period.

The objective function seeks to maximize the expected return of the portfolio.

Constraint (1) ensures that the standard deviation of the optimal asset allocation does not go beyond a given threshold.

Constraint (2) enforces that exactly 100% of the funds are allocated to some combination of assets.

Constraint (3) is a set of n box constraints that ensures each asset is assigned a non-negative proportion of the funds, up to a maximum of 100%.

Therefore, this model has a linear objective function and a set of quadratic and linear constraints. Note it cannot be considered a linear program because even if the objective is linear, not all constraints are. This is in fact a Quadratically Constrained Quadratic Program (QCQP).

In our implementation this model has about 400 continuous variables and 400 constraints (including box contraints).

2.3 Model 3: Simultaneous mean-variance optimization

Back to index

In both Model 1 and Model 2 one's preferences are heavily constrained by the random data one collects, in the sense that one cannot require a minimum return higher than any return observed in the sample collected, and likewise one cannot require a maximum variance lower than any variance observed in the sample collected.

By optimizing both the expected return and the standard deviation simultaneously we overcome the issue above, in the sense that we will not have an infeasible problem due to 'unrealistic market demands'.

However, this comes at the cost of introducing arbitrariness in the form of the penalty parameter λ, which balances how much we care about each of the two objectives.

In any case, as we will see in a later section, we will present a simple heuristic that attempts to remove part of this arbitrariness.

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{i=1}^{n} r_i x_i + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \sigma_{i,j} x_i x_j & \\ \text{subject to:} \quad & \sum_{i=1}^{n} x_i = 1 & (1) \\ & 0 \leq x_i \leq 1 \qquad i = 1, \ldots, n & (2) \end{aligned}$$

Where the decision variable x_i is the proportion of the funds allocated to asset i, $\sigma_{i,j}$ is the (daily) covariance between asset i and asset j, r_i is the expected daily return of asset i (computed via the Method of Moments i.e. the arithmetic mean over the sample period).

The objective function seeks to minimize the negative of the expected return (i.e. equivalent to maximizing the expected return) while also minimizing λ times the variance of the portfolio.

Constraint (1) enforces that exactly 100% of the funds are allocated to some combination of assets.

Constraint (2) is a set of n box constraints that ensures each asset is assigned a non-negative proportion of the funds, up to a maximum of 100%.

Therefore, this model has a quadratic objective function and a set of linear constraints, so it is a QP.

Finally, it is worth mentioning that a variation of the above three models introducing L_1 **regularization** was considered. However, it was eventually discarded since, by construction, it had no effect irrespective of the size of the penalty parameter $\mu > 0$. We provide a simple **proof** next.

L_1 regularization introduces an extra term in the objective function, which, in case of a minimization problem, is $\mu \sum_{i=1}^{n} u_i$, with $|x_i| \leq u_i$ for all $i \in \{1, \dots, n\}$. But then, since $x_i \geq 0$ for all $i \in \{1, \dots, n\}$, the previous epigraph condition simplifies to $x_i \leq u_i$ for all $i \in \{1, \dots, n\}$, which implies $\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} u_i$. But we require $\sum_{i=1}^{n} x_i = 1$ so $1 \leq \sum_{i=1}^{n} u_i$. Finally, since $\sum_{i=1}^{n} u_i$ is penalized in the objective by $\mu > 0$, and there are no additional constraints on u_i, we then have $\sum_{i=1}^{n} u_i = 1$ at the optimum, irrespective of the particular (positive) value of μ.

In our implementation this model has about 400 continuous variables and 400 constraints (including box contraints).

2.4 Model 4: Mean Absolute Deviation (Konno and Yamazaki 1991)

Back to index

Note: This model has not been implemented in code in this project.

The authors propose in their 1991 paper [4] a way to **turn portfolio optimization into a linear program** based on the **'epigraph trick'** seen in class. Their key idea is to substitute the standard deviation for the mean absolute deviation, which is a very related concept, as we show next:

(Sample) Mean absolute deviation: $\frac{1}{T} \sum_{t=1}^{T} \left| \sum_{i=1}^{n} (r_{i_t} - \overline{r_i}) x_i \right|$

(Sample) Standard deviation: $\sqrt{\frac{1}{T} \sum_{t=1}^{T} [\sum_{i=1}^{n} (r_{i_t} - \overline{r_i}) x_i]^2}$

where $\overline{r_i} = \sum_{t=1}^{T} x_i$

The above two expressions would actually be equal if the square root was placed in the innermost sum (recall that $\sqrt{f(x)^2} = |f(x)|$). In fact, Konno and Yamazaki prove that the two resulting optimization problems are equivalent if the returns are multivariate normally distributed.

Original program (non linear):

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{T} \sum_{t=1}^{T} \left| \sum_{i=1}^{n} (r_{i_t} - \overline{r_i}) x_i \right| & \\ \text{subject to:} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho & (1) \\ & \sum_{i=1}^{n} x_i = 1 & (2) \\ & 0 \leq x_i \leq 1 \qquad i = 1, \dots, n & (3) \end{aligned}$$

Linear program using epigraph trick:

$$\min_{x\in\mathbb{R}^n, y\in\mathbb{R}^T} \quad \frac{1}{n}\sum_{t=1}^{T} y_t$$

$$\text{subject to:} \quad y_t \geq \sum_{i=1}^{n}(r_{i_t} - \overline{r_i})x_i \qquad t = 1, \dots, T \qquad (1)$$

$$y_t \geq -\sum_{i=1}^{n}(r_{i_t} - \overline{r_i})x_i \qquad t = 1, \dots, T \qquad (2)$$

$$\sum_{i=1}^{n} r_i x_i \geq \rho \qquad (3)$$

$$\sum_{i=1}^{n} x_i = 1 \qquad (4)$$

$$0 \leq x_i \leq 1 \qquad i = 1, \dots, n \qquad (5)$$

We see the epigraph variable y_t replacing the mean absolute deviation in the objective function, as well as the two sets of T epigraph constraints. The rest of the constraints are identical to Models 1, 2 and 3.

As the authors discuss, this linearized model has several advantages over the usual quadratic one, and here we highlight the ones we believe are most important, in order of importance:

First, the number of functional constraints remains constant at $2T + 2$ regardless of the number of stocks included in the model, so it allows one to update the optimal portfolio in real time even if the universe of stocks under consideration is thousands of stocks (which is a very standard number in modern captial markets).

Second, an optimal solution contains at most $2T + 2$ non-zero components if short-selling is allowed (non-negativity constraint is dropped). In contrast, an optimal portfolio derived from L_2 risk (instead of L_1 like in this case), may contain as many as n assets with non-zero allocation. This is in general not desirable due to fixed costs (transaction fees) and increased portfolio management difficulty (more companies and exposures to monitor). Note that controling for how many stocks have non-zero allocations via a lower bound on the decision variable (that is activated only if the stock is selected) would turn the problem into a Mixed Integer Program, which is in theory even harder to solve than a QP. We will explore this option in Model 6 and we'll see in practice this is not the case.

Third, there's no need anymore to calculate a covariance matrix, so it is easier and faster to update ones' optimal portfolio when new stocks are added into the universe under consideration, or when new data on individual stock returns becomes available. However, related to the second point above, this model may favor stock concentration (i.e. the opposite of stock diversification) since it does not use covariance information. This may not be desirable.

Overall, the authors report that the computational complexity of solving a L_1-based model is $O(n)$ while that of a traditional L_2-based model is $O(n^2)$, where n is the number of assets under consideration, and so the time savings for large n can be very significant. Or, for a fixed solving time, the LP can consier a much larger universe of stocks, providing a much better optimal solution (at least in theory). As a side note, I believe their linear-time claim is based on empirical evidence since, for example, the complexity of a very common LP algorithm like the Simplex method is exponential in the worst case, but it has been observed to perform linearly in most cases.

But, in addition to the above advantages, we would like to add two key advantages that an LP has over a QP in any setting, not just in the case of portfolio optimization.

First, the optimal solution of an LP, if it exists, is always at a vertex of the polyhedral feasible region, so the optimum of any LP can always be found by checking a finite number of points (which is the essence of the Simplex method). On the other hand, in a QP the optimum may be reached in the interior of the feasible region, and hence there is absolutely no guarantee that it can be found by checking a finite number of points, and hence other, more sophisticaded methods must be used.

Second, the objective function of an LP is by definition always convex (and concave), whereas this is not always the case in a QP (the coefficient matrix may not be positive semi-definite), in which case the problem becomes much harder to solve. In portfolio optimization, though, this second aspect will never be an issue since the covariance matrix is by definition positive semi-definite. But still, it is worth pointing this general advantage of LP compared to QP.

If we were to implement this model, it would have about 450 continuous variables and 520 constraints (including box contraints).

Next we present our proposed linearization of the portfolio optimization problem.

2.5 Model 5: Maximum Drawdown portfolio optimization

Back to index

We propose a new way to optimize a portfolio that relies on **linear programming**. The idea is simple: find a portfolio of stocks that minimizes the maximum drawdown of the portfolio (subject to a minimum expected return). Chekhlov et al. [5] consider a similar approach but they opt for maximizing expected return subject to a maximum drawdown allowed. We instead opt for the reverse formulation.

The 'maximum drawdown' of an asset over a time period is defined as the largest drop in price of the asset in any given period. In our case, the optimal portfolio will combine stocks such that the worst overall performance of the portfolio in any given trading day is as good ('less bad') as possible.

This is a reasonable goal to pursue in building a portfolio, and it has several advantages over the traditional mean-variance portfolio optimization. Most notably, the fact that we consider a downside risk metric instead of a general risk metric like the standard deviation, which also penalizes desirable outcomes like presence of returns above the mean return.

Hence, we wish to maximize the minimum portfolio return over a sample time period.

This is a **discretized version of the maximin model** seen in class (instead of a set of functions we have a set of discrete portfolio returns for $t \in \{1, 2, \dots, T\}$ for all possible portfolio configurations). It can be made linear using the **epigraph trick**. To prevent a trivial solution allocating 100% of the funds to the one stock with the minimum maximum drawdown (largest minimum return in any given trading day of the sample period), we impose a ceiling of 50% of the funds on any given allocation, so there will be at least 2 stocks in the optimal portfolio (and note that it is not necessarily the case that they will be the 2 stocks with largest minimum return, since those may happen on the same day and hence add up to a larger maximum portfolio drawdown than the optimal achievable). Note that by setting the ceiling to 50% we avoid a trivial solution while also avoiding the arbitrariness of imposing a minimum number of $k > 1$ stocks in our optimal portfolio. Finally, note that we don't really lose anything by requiring at least 2 stocks in the optimal portfolio, since basic domain knowledge tells us that some degree of diversification is always good, and with 1 stock you have none.

Original program (non-linear):

Define $\overline{r_t} = \sum_{i=1}^{n} r_{i_t} x_i$, that is, the weighted average return on day t of a portfolio using weights x.

$$\begin{aligned}
\max_{x\in\mathbb{R}^n \;\; t=1,\dots,T} \min \quad & \overline{r_t} \\
\text{subject to:} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \qquad (1) \\
& \sum_{i=1}^{n} x_i = 1 \qquad (2) \\
& 0 \leq x_i \leq 0.5 \qquad i = 1, \dots, n \qquad (3)
\end{aligned}$$

Linear program using epigraph trick:

$$\begin{aligned}
\max_{x\in\mathbb{R}^n, y\in\mathbb{R}} \quad & y \\
\text{subject to:} \quad & y \leq \overline{r_t} \qquad t = 1, \dots, T \qquad (1) \\
& \sum_{i=1}^{n} r_i x_i \geq \rho \qquad (2) \\
& \sum_{i=1}^{n} x_i = 1 \qquad (3) \\
& 0 \leq x_i \leq 0.5 \qquad i = 1, \dots, n \qquad (4)
\end{aligned}$$

The epigraph trick can be interpreted as follows: find the largest possible value of the single variable y such that, for each and every day, y is below (or equal to) the portfolio return given by the weights vector x. Therefore this achieves the goal of finding the budget allocation x that maximizes the minimum portfolio return on any given day. The rest of constraints are the same as in the previous models.

Note our LP has just $T + 2$ constraints, so it's even simpler than Model 4. But even more importantly, the number of constraints is independent of the number of stocks under consideration, so it scales very well to larger sets of stocks.

In addition, our LP has $n + 1$ variables, instead of $n + T$ in Model 4, and just 1 variable more than the standard QP.

Finally, we write this LP in **standard form**. By writing it in standard form, this program does not really gain anything (in fact, it becomes somewhat harder to read) but it may be interesting to show one example of an LP that uses the epigraph trick and is in standard form.

$$\begin{aligned}
\max_{x\in\mathbb{R}^n, y\in\mathbb{R}} \quad & y \\
\text{subject to:} \quad & y - \overline{r_t} \leq 0 \qquad t = 1, \dots, T \qquad (1) \\
& -\sum_{i=1}^{n} r_i x_i \leq -\rho \qquad (2) \\
& \sum_{i=1}^{n} x_i \leq 1 \qquad (3.1) \\
& -\sum_{i=1}^{n} x_i \leq -1 \qquad (3.2) \\
& 0 \leq x_i \leq 0.5 \qquad i = 1, \dots, n \qquad (4)
\end{aligned}$$

In our implementation this model has about 400 continuous variables and 450 constraints (including box contraints).

2.6 Model 6: Maximum Drawdown portfolio optimization with minimum allocation constraint

Back to index

In this variation of Model 5 we impose a minimum allocation size, which in turn implies a maximum number of positive allocations. This is a potentially more refined formulation since having many small allocations is not desirable in practice, as we have already discussed in subsection 2.4. However, this variation is, in theory, much harder to solve, so it may not be worth it depending on the problem structure, the number of stocks (n) and sample time periods (T) under consideration.

Let $M = 0.5$ be a **'big M'** paramater. Note 0.5 is the smallest (i.e. 'best') M can be because x_i could be as large as 0.5 (but not more, since 0.5 is the (least) upper bound on any allocation by construction). In fact, by choosing M this way we obtain the **tightest LP relaxation** possible; this way the feasible region we work with is as close as possible to the **convex hull** using available information from the problem.

We want to model for all $i \in \{1, \dots, n\}$ $x_i > 0 \Rightarrow x_i \geq 5\%$. This way we also ensure that at most 20 stocks (out of about 400) will be picked.

Let z_i $i = 1, \dots, n$ be a **binary variable** that is 1 if and only if $x_i > 0$.

Note that if we impose $x_i \geq 0.05 z_i$ then when $z_i = 1$ indeed $x_i \geq 0.05$, and when $z_i = 0$ then $x_i \geq 0$... but the problem is that if we don't do anything else the solver will dodge this (attempted) **logical constraint** by just setting $z_i = 0$ for all i. We can fix this by also requiring $x_i \leq M z_i$ where $M = 0.5$ is a 'big M' parameter. With this additional constraint, the solver cannot dodge our previous contraint by setting all $z_i = 0$ because that would imply $0 \leq x_i \leq 0$ for all i, which is not even a feasible solution since we require $\sum_{i=1}^{n} x_i = 1$.

MILP using **epigraph trick** and logical constraints:

$$
\begin{aligned}
\max_{x \in \mathbb{R}^n, y \in \mathbb{R}, z \in \{0,1\}^n} \quad & y \\
\text{subject to:} \quad & y \leq \overline{r_t} && t = 1, \dots, T && (1) \\
& \sum_{i=1}^{n} r_i x_i \geq \rho && && (2) \\
& \sum_{i=1}^{n} x_i = 1 && && (3) \\
& x_i \geq 0.05 z_i && i = 1, \dots, n && (4) \\
& x_i \leq M z_i && i = 1, \dots, n && (5) \\
& 0 \leq x_i \leq 0.5 && i = 1, \dots, n && (6)
\end{aligned}
$$

As it can be seen, implementing a very minor refinement like the one in Model 6 required **turning an LP into a MILP*** and increased the number of variables from $n + 1$ to $2n + 1$, and the number of contraints from $T + 2$ to $T + 2 + 2n$, which means the number of functional constraints is no longer independent of the number of stocks.

But more fundamentally, this problem seems similar (if anything, harder) to the example of **NP-complete** problem seen in class (finding a subset of numbers in a set that sum to a given constant, in this case 1, and here the candidates to add up are not given ex-ante), which means that there are 2^n total possible combinations and checking all of them by brute force would would take thousands of times the current age of

the universe for $n = 100$. In our case we have $n = 393$. However, as we will see, the particular structure of our problem and the use of a sophisticated MIP solver like Gurobi will in fact produce a solution in less time than our LP in Model 5.

In our implementation this model has about 400 continuous variables, 400 binary variables, and 1250 constraints (including box contraints).

3. Solution

3.A. Data import and implementation of Models 1, 2, 3

Back to index

In [27]:
```
# Data import, used in all models [data: May 1, 2020 - Aug 1, 2020]
using DataFrames, CSV, LinearAlgebra

# Reading daily returns of S&P 500 stocks
daily_expected_returns = CSV.read("_exp_ret.csv",header=1,delim=',')  #f
ile must be in same directory as this notebook
SP_tickers = daily_expected_returns[:,1]
daily_expected_returns = daily_expected_returns[:,2]  # keep only the 2n
d column (1st column is stock ticker)
r = 100 .*daily_expected_returns # Expected return (in percent)
rr = convert(Array, r) # for convenience, convert the data frame r into
 an Array (rr)
n = length(r)
println(n)  # We have 393 stocks
# Reading covariance matrix
Σ = CSV.read("_sigma_mat.csv",header=1,delim=',')
select!(Σ, Not(:Symbols))
Σ = 10000 .*convert(Matrix,Σ); # Expected variance (in percent squared)
```

```
393
```

In [28]:
```
# MODEL 1: Classical Markowitz's model

# NOTE: if solution not displayed nicely, please run this cell again

ϱ = 0.05  #we require at least 0.05% daily return, approx 250*0.05% = 1
2.5% in a trading year

using JuMP, Gurobi, Statistics

m1 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m1, "OutputFlag", 0)

@variable(m1, x[1:n]>=0)

@objective(m1, Min, dot(x,Σ*x) )  #note: it's equivalent to consider the
variance or std dev = sqrt(variance)

@constraint(m1, dot(rr,x) >= ϱ)
@constraint(m1, budget, sum(x) == 1)

optimize!(m1)
xsol = value.(x)
xsol_clean1 = zeros(n,1)

for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean1[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret = dot(rr,xsol_clean1) # calculate rate of return
sdev = sqrt(dot(xsol_clean1,Σ*xsol_clean1)) # calculate std

println()
println("************ MODEL 1 ************")
println("Optimized expected return: ", ret, " Optimized std dev: ", sdev
)
println(length(ix1)," stocks with a nonzero allocation")
println()
selected_stocks = SP_tickers[ix1]
allocations = 100 .*xsol[ix1]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end

# Note the following benchmarking operation may take up to 30 seconds
k = 1000
t_vector = zeros(k)
for i in 1:k
    t_vector[i] = @elapsed(optimize!(m1))
end
println()
println("Median solving time in ", k, " iterations: " ,round(Statistics.
median(t_vector),digits=4)," seconds.")
```

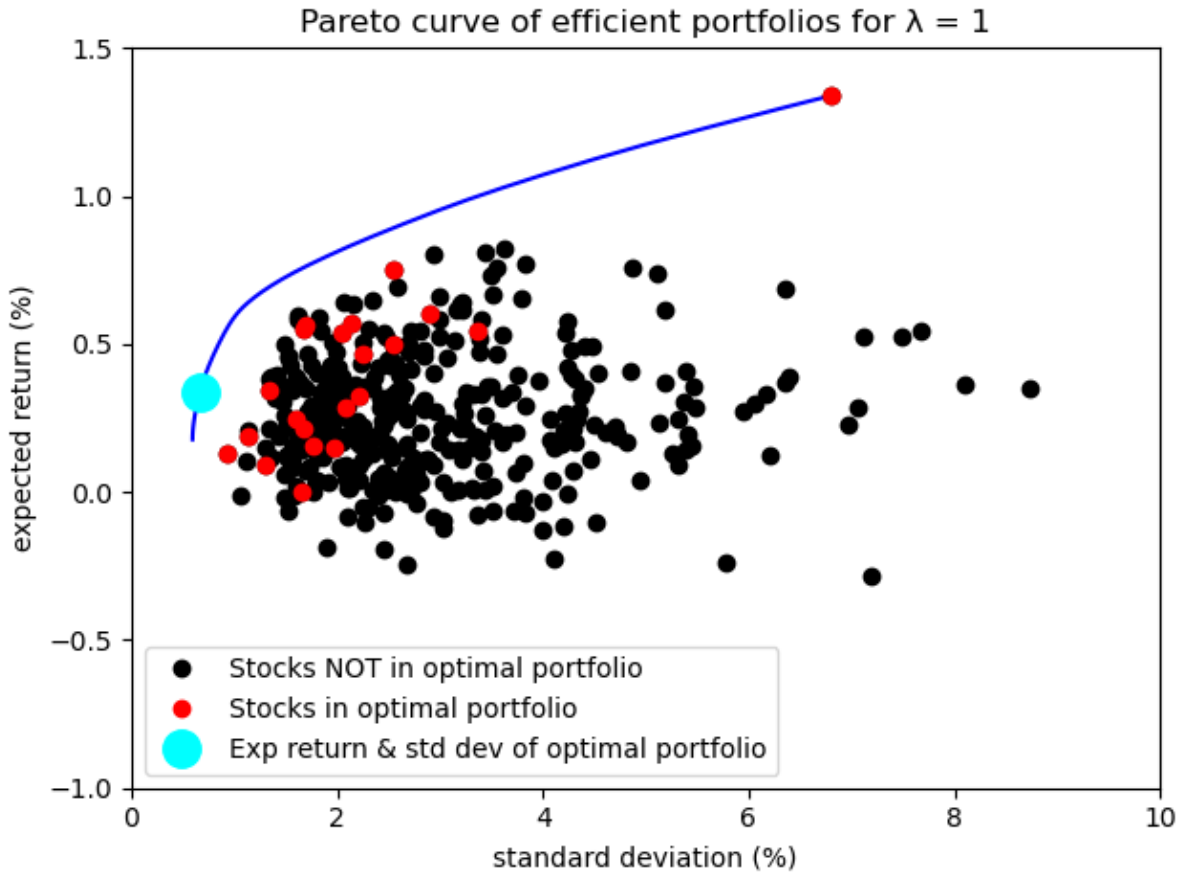
```
Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 1 ************
Optimized expected return: 0.17767929562010903 Optimized std dev: 0.593
0069818947696
16 stocks with a nonzero allocation

Allocate funds in the following way:
ABBV: 1.34%
AMZN: 1.86%
CMG: 1.94%
CLX: 12.77%
CMCSA: 2.84%
ED: 4.96%
COST: 18.34%
D: 4.64%
EBAY: 3.71%
EFX: 7.49%
JNJ: 1.8%
KMB: 7.06%
KR: 6.77%
TIF: 16.39%
FOXA: 0.61%
WMT: 7.46%

Median solving time in 1000 iterations: 0.0224 seconds.
```

In [3]:
```
# MODEL 2: Reverse variation of classical Markowitz's model

sigma2 = 1  #we require at most 1% daily variation, approx 250*1% = 250%
in a trading year

using JuMP, Gurobi
m2 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m2, "OutputFlag", 0)

@variable(m2, x[1:n]>=0)

@objective(m2, Max, dot(rr,x) )

@constraint(m2, dot(x,Σ*x) <= sigma2)  #note: it's equivalent to conside
r the variance or std dev = sqrt(variance)
@constraint(m2, budget, sum(x) == 1)

optimize!(m2)
xsol = value.(x)
xsol_clean2 = zeros(n,1)

for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean2[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret = dot(rr,xsol_clean2) # calculate rate of return
sdev = sqrt(dot(xsol_clean2,Σ*xsol_clean2)) # calculate std

println()
println("************ MODEL 2 ************")
println("Optimized expected return: ", ret, " Optimized std dev: ", sdev
)
println(length(ix1)," stocks with a nonzero allocation")
println()
selected_stocks = SP_tickers[ix1]
allocations = 100 .*xsol[ix1]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end

# Note the following benchmarking operation may take up to 30 seconds
k = 1000
t_vector = zeros(k)
for i in 1:k
    t_vector[i] = @elapsed(optimize!(m2))
end
println()
println("Median solving time in ", k, " iterations: " ,round(Statistics.
median(t_vector),digits=4)," seconds.")
```

```
Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 2 ************
Optimized expected return: 0.5913910141467928 Optimized std dev: 0.9999
999875560069
13 stocks with a nonzero allocation

Allocate funds in the following way:
AMZN: 0.99%
AAPL: 1.05%
CDNS: 6.23%
CHD: 15.17%
CTAS: 6.56%
CLX: 4.8%
EBAY: 26.45%
HOLX: 7.95%
LB: 4.05%
LOW: 0.45%
PDCO: 0.35%
TSCO: 19.29%
UPS: 6.66%

Median solving time in 1000 iterations: 0.0198 seconds.
```

In [4]:
```
# MODEL 3: Class model with Pareto curve

using JuMP, Gurobi
m3 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m3, "OutputFlag", 0)

@variable(m3, x[1:n]>=0)

@constraint(m3, sum(x) == 1)

# compute optimal tradeoff curve (this may take a few seconds)
N = 100
ret_vector = zeros(N)
sdev_vector = zeros(N)
lambda_values = 10 .^(range(-3,stop=4,length=N))  # min lambda = 10 ^-3,
max lambda = 10^4

for (i,λ) in enumerate(lambda_values)
    @objective(m3, Min, -dot(rr,x) + λ*dot(x,Σ*x) )
    optimize!(m3)
    xsol = value.(x)
    xsol_clean = zeros(n,1)
    for i in 1:n
        if xsol[i] > 1e-5
            xsol_clean[i] = xsol[i]
        end
    end
    ret_vector[i] = dot(rr,xsol_clean)
    sdev_vector[i] = sqrt(dot(xsol_clean,Σ*xsol_clean))
end


# Class model solved for λ = 1
m3 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m3, "OutputFlag", 0)

@variable(m3, x[1:n]>=0)

@constraint(m3, budget, sum(x) == 1)

λ = 1 # weight on risk

@objective(m3, Min, -dot(rr,x) + λ*dot(x,Σ*x) )

optimize!(m3)
xsol = value.(x)

xsol_clean3 = zeros(n,1)
for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean3[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret1 = dot(rr,xsol_clean3) # optimal expected return: 0.36
```

```
std1 = sqrt(dot(xsol_clean3,Σ*xsol_clean3)) # optimal standard deviatio
n: 0.7

# plot tradeoff (Pareto) curve
using PyPlot
plot(sdev_vector,ret_vector,"b-")
plot(sqrt.(diag(Σ)), rr, "k.", markersize=12,label="Stocks NOT in optima
l portfolio")
plot(sqrt.(diag(Σ))[ix1], rr[ix1], "r.", markersize=12,label="Stocks in
 optimal portfolio")
plot([std1], [ret1], "o", color="cyan",markersize=14,label="Exp return &
std dev of optimal portfolio")
# note it lies on the Pareto curve, but far from the 'corner' of the cur
ve, roughly at (1.4,0.7)
xlabel("standard deviation (%)")
ylabel("expected return (%)")
axis([0,10,-1,1.5]);
title("Pareto curve of efficient portfolios for λ = 1")
legend()
tight_layout()
```




```
Academic license - for non-commercial use only
Academic license - for non-commercial use only
Academic license - for non-commercial use only
Academic license - for non-commercial use only
```

In [5]:
```
minsdev = minimum(sdev_vector)
println(minsdev)
maxret = maximum(ret_vector)
println(maxret)
```

```
0.593006981616025
1.3397743234544173
```

Back to index

Note that, in our universe of optimal portfolios for 100 different values of λ, the minimum portfolio standard deviation is about 0.59% daily, and the maximum expected portfolio daily return is 1.34%.

We want to find the value of the parameter λ that yields an optimal portfolio with minimum **Euclidean norm** from the point (0.593006981616025,1.3397743234544173). The rationale behind this heuristic is that the top left corner is the ideal situation (high expected return and low risk), and in our universe of optimal portfolios for different values of λ the top left corner is given by the two coordinates we have just mentioned.

This is a natural heuristic to objectively choose the 'best' **Pareto optimal portfolio**, i.e. a way to objectively choose the 'best' balance between the goal of maximizing expected return and minimizing standard deviation. As a brief reminder, a Pareto optimal portfolio is a portfolio in which it is not possible to increase the portfolio expected return without also increasing the portfolio standard deviation, and vice versa (it is not possible to reduce the portfolio standard deviation without also reducing the portfolio expected return). Any portfolio to the left of the Pareto curve (also called 'Efficient frontier' in this context) is infeasible, and any portfolio to the right of the Pareto curve is inefficient i.e. it is possible to improve one of the two metrics without worsening the other.

In [6]:
```
norm2 = zeros(N,1)
for i in 1:N
    norm2[i] = sqrt(((sdev_vector[i]-minsdev)^2 + ret_vector[i]-maxret)^
2)
end
optLambda = lambda_values[argmin(norm2)]
```

Out[6]:
```
0.08111308307896872
```

In [7]:
```
# Repeat class model with optimal lambda

using JuMP, Gurobi
m3 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m3, "OutputFlag", 0)

@variable(m3, x[1:n]>=0)

@constraint(m3, budget, sum(x) == 1)

λ = optLambda  # optimal lambda

@objective(m3, Min, -dot(rr,x) + λ*dot(x,Σ*x) )

optimize!(m3)
xsol = value.(x)

xsol_clean3 = zeros(n,1)
for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean3[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret = dot(rr,xsol_clean3) # calculate rate of return
sdev = sqrt(dot(xsol_clean3,Σ*xsol_clean3)) # calculate std

println()
println("************ MODEL 3 ************")
println("Optimized expected return: ", ret, " Optimized std dev: ", sdev
)
println(length(ix1)," stocks with a nonzero allocation")
println()
selected_stocks = SP_tickers[ix1]
allocations = 100 .*xsol[ix1]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end

# Note the following benchmarking operation may take up to 30 seconds
k = 1000
t_vector = zeros(k)
for i in 1:k
    t_vector[i] = @elapsed(optimize!(m3))
end
println()
println("Median solving time in ", k, " iterations: " ,round(Statistics.
median(t_vector),digits=4)," seconds.")
```

```
Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 3 ************
Optimized expected return: 0.6968543786239834 Optimized std dev: 1.3568
34421442871
12 stocks with a nonzero allocation

Allocate funds in the following way:
AMD: 2.96%
AAPL: 0.02%
CDNS: 4.32%
CHD: 2.36%
EBAY: 31.31%
HOLX: 0.76%
IDXX: 10.47%
LB: 10.33%
LRCX: 4.09%
PDCO: 1.15%
TSCO: 16.85%
UPS: 15.37%

Median solving time in 1000 iterations: 0.0205 seconds.
```

3.B. Data import and implementation of Models 5, 6

Back to index

In [8]:
```
# Data loading for Models 5 and 6

using DataFrames, CSV

# Reading daily returns of S&P 500 stocks
SPstocks_clean_rets = CSV.read("_stocks_SP_clean_daily_returns.csv",head
er=1,delim=',');
SPstocks_clean_rets = SPstocks_clean_rets[2:end,2:end]; #first row is em
pty since no returns in day 1, 1st col: date
# [up]: 63 days, 393 stocks, ij is return on day i for stock j
```

In [29]:
```
# Model 5: Maximum drawdown portfolio optimization [LP]

T = size(SPstocks_clean_rets, 1)  #number of rows (days) in stock return
s dataframe
ϱ = 0.05

using JuMP, Clp  #note we can use a linear solver, which should be faste
r than a more general one like Gurobi
m5 = Model(Clp.Optimizer)
set_optimizer_attribute(m5, "LogLevel", 0)

@variable(m5, 0<=x[1:n]<=0.5)
@variable(m5, y)  # epigraph trick variable

@objective(m5, Max, y)

@constraint(m5,epi[t in 1:T], y <= dot(SPstocks_clean_rets[t,:],x))
@constraint(m5, dot(rr,x) >= ϱ)
@constraint(m5, budget, sum(x) == 1)

optimize!(m5)
xsol = value.(x)
xsol_clean5 = zeros(n,1)

for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean5[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret = dot(rr,xsol_clean5) # calculate rate of return
sdev = sqrt(dot(xsol_clean5,Σ*xsol_clean5)) # calculate std

println()
println("************ MODEL 5 ************")
println("Expected return of optimal allocation: ", ret, " Std dev of opt
imal allocation: ", sdev)
println(length(ix1)," stocks with a nonzero allocation")
println(objective_value(m5)*100)  #multiply by 100 to have the optimal p
ortfolio maximum drawdown in %
println()
selected_stocks = SP_tickers[ix1]
allocations = 100 .*xsol[ix1]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end

# Note the following benchmarking operation may take up to 5 seconds (se
e the speed of an LP)
k = 1000
t_vector = zeros(k)
for i in 1:k
    t_vector[i] = @elapsed(optimize!(m5))
end
```

```
println()
println("Median solving time in ", k, " iterations: " ,round(Statistics.
median(t_vector),digits=4)," seconds.")
```

```
************ MODEL 5 ************
Expected return of optimal allocation: 0.21701108723809875 Std dev of o
ptimal allocation: 0.8380393976081978
9 stocks with a nonzero allocation
-0.8525968035575026

Allocate funds in the following way:
COG: 1.16%
CLX: 38.2%
DLTR: 3.36%
JCI: 3.08%
KMB: 2.22%
KR: 27.48%
ORLY: 2.48%
TGT: 0.89%
TIF: 21.12%

Median solving time in 1000 iterations: 0.0027 seconds.
```

In [10]:
```
# Model 6: Maximum drawdown portfolio optimization with minimum allocati
on [MILP]

using JuMP, Gurobi
m6 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m6, "OutputFlag", 0)

M = 0.5

@variable(m6, 0<=x[1:n]<=0.5)
@variable(m6, y)  # epigraph trick variable
@variable(m6, z[1:n], Bin)

@objective(m6, Max, y)

@constraint(m6,epi[t in 1:T], y <= dot(SPstocks_clean_rets[t,:],x))
@constraint(m6, dot(rr,x) >= ϱ)
@constraint(m6, sum(x) == 1)
@constraint(m6, logic1[i in 1:n], x[i] >= 0.05*z[i])
@constraint(m6, logic2[i in 1:n], x[i] <= M*z[i])

optimize!(m6)
xsol = value.(x)
xsol_clean6 = zeros(n,1)

for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean6[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret = dot(rr,xsol_clean6) # calculate rate of return
sdev = sqrt(dot(xsol_clean6,Σ*xsol_clean6)) # calculate std

println()
println("************ MODEL 6 ************")
println("Expected return of optimal allocation: ", ret, " Std dev of opt
imal allocation: ", sdev)
println(length(ix1)," stocks with a nonzero allocation")
println(objective_value(m6)*100)  #multiply by 100 to have the optimal p
ortfolio maximum drawdown in %
println()
selected_stocks = SP_tickers[ix1]
allocations = 100 .*xsol[ix1]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end

# Note the following benchmarking operation may take up to 'nothing' (ve
ry impressive for a MILP)
k = 1000
t_vector = zeros(k)
for i in 1:k
    t_vector[i] = @elapsed(optimize!(m6))
```

```
end
println()
println("Median solving time in ", k, " iterations: " ,round(Statistics.
median(t_vector),digits=4)," seconds.")
```

```
Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 6 ************
Expected return of optimal allocation: 0.2273254754734032 Std dev of op
timal allocation: 0.8429542141673866
5 stocks with a nonzero allocation
-0.8638366402913557

Allocate funds in the following way:
CLX: 40.34%
DLTR: 5.0%
JCI: 5.01%
KR: 28.92%
TIF: 20.73%

Median solving time in 1000 iterations: 0.0001 seconds.
```

In [11]:
```
# Let's find the Maximum Drawdown (MD) of Models 1, 2, and 3 too

function getMD(M,x)
    #input: T by n matrix of stock returns, vector of weights
    #output: Max Drawdown (float)

    M = Matrix(M) #just in case it's not a matrix
    pr = M*x #vector of portfolio returns per day

    MD = minimum(pr)*100 #the portfolio return in the worst day, in %

    return MD
end

MD1 = getMD(SPstocks_clean_rets,xsol_clean1);
MD2 = getMD(SPstocks_clean_rets,xsol_clean2);
MD3 = getMD(SPstocks_clean_rets,xsol_clean3);

println("MD1 = ",MD1, "MD2 = ",MD2, "MD3 = ",MD3)
```

```
MD1 = -1.995869952609788MD2 = -3.176219519707469MD3 = -4.80227233133059
9
```

3.C. Out-of-sample comparison of models 1, 2, 5, 6

Back to index

In [20]:
```
# Solving again Models 1, 2, 5, 6 with data from February 1, 2020 to May
1, 2020
# Note: FM subscript denotes 'February - May'

# ---------------- Load data ---------------- #
using DataFrames, CSV, LinearAlgebra

# Reading daily returns of S&P 500 stocks
daily_expected_returns_FM = CSV.read("_exp_ret_FebMay.csv",header=1,deli
m=',')  #file must be in same directory as this notebook
SP_tickers_FM = daily_expected_returns_FM[:,1]
daily_expected_returns_FM = daily_expected_returns_FM[:,2]  # keep only
 the 2nd column (1st column is stock ticker)
r_FM = 100 .*daily_expected_returns_FM # Expected return (in percent)
rr_FM = convert(Array, r_FM) # for convenience, convert the data frame r
into an Array (rr)

n_FM = length(r_FM)
println(n_FM)  # We have 374 stocks for this period

# Reading covariance matrix
ΣFM = CSV.read("_sigma_mat_FebMay.csv",header=1,delim=',')
select!(ΣFM, Not(:Symbols))
ΣFM = 10000 .*convert(Matrix,ΣFM); # Expected variance (in percent squar
ed)

# Data collection for Models 5 and 6

# Reading daily returns of S&P 500 stocks
SPstocks_clean_rets_FM = CSV.read("_stocks_SP_clean_daily_returns_FebMa
y.csv",header=1,delim=',');
SPstocks_clean_rets_FM = SPstocks_clean_rets_FM[2:end,2:end];

using JuMP, Gurobi, Statistics, Clp


# ---------------- Model 1 ---------------- #
ϱ = 0.05
m1 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m1, "OutputFlag", 0)

@variable(m1, x[1:n_FM]>=0)

@objective(m1, Min, dot(x,ΣFM*x) )  #note: it's equivalent to consider t
he variance or std dev = sqrt(variance)

@constraint(m1, dot(rr_FM,x) >= ϱ)
@constraint(m1, budget, sum(x) == 1)

optimize!(m1)
xsol = value.(x)
xsol_clean1_FM = zeros(n_FM,1)

for i in 1:n_FM
    if xsol[i] > 1e-5
        xsol_clean1_FM[i] = xsol[i]
```

```
    end
end

ix_FM = findall(xsol .> 1e-5) # pull all xs with positive value
ret_FM = dot(rr_FM,xsol_clean1_FM) # calculate rate of return
sdev_FM = sqrt(dot(xsol_clean1_FM,ΣFM*xsol_clean1_FM)) # calculate std

println()
println("************ MODEL 1 ************")
println("Optimized expected return: ", ret_FM, " Optimized std dev: ", s
dev_FM)
println(length(ix_FM)," stocks with a nonzero allocation")
println()
selected_stocks = SP_tickers_FM[ix_FM]
allocations = 100 .*xsol[ix_FM]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end



# ---------------- Model 2 ---------------- #
sigma2 = 3
m2 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m2, "OutputFlag", 0)

@variable(m2, x[1:n_FM]>=0)

@objective(m2, Max, dot(rr_FM,x) )

@constraint(m2, dot(x,ΣFM*x) <= sigma2)  #note: it's equivalent to consi
der the variance or std dev = sqrt(variance)
@constraint(m2, budget, sum(x) == 1)

optimize!(m2)
xsol = value.(x)
xsol_clean2_FM = zeros(n_FM,1)

for i in 1:n_FM
    if xsol[i] > 1e-5
        xsol_clean2_FM[i] = xsol[i]
    end
end

ix_FM = findall(xsol .> 1e-5) # pull all xs with positive value
ret_FM = dot(rr_FM,xsol_clean2_FM) # calculate rate of return
sdev_FM = sqrt(dot(xsol_clean2_FM,ΣFM*xsol_clean2_FM)) # calculate std

println()
println("************ MODEL 2 ************")
println("Optimized expected return: ", ret_FM, " Optimized std dev: ", s
dev_FM)
println(length(ix_FM)," stocks with a nonzero allocation")
println()
selected_stocks = SP_tickers_FM[ix_FM]
allocations = 100 .*xsol[ix_FM]  #to have them in %
```

```
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end
```

```
# ---------------- Model 5 ---------------- #
T_FM = size(SPstocks_clean_rets_FM, 1)  #number of rows (days) in stock
 returns dataframe

m5 = Model(Clp.Optimizer)
set_optimizer_attribute(m5, "LogLevel", 0)

@variable(m5, 0<=x[1:n_FM]<=0.5)
@variable(m5, y)  # epigraph trick variable

@objective(m5, Max, y)

@constraint(m5,epi[t in 1:T_FM], y <= dot(SPstocks_clean_rets_FM[t,:],x
))
@constraint(m5, dot(rr_FM,x) >= ϱ)
@constraint(m5, budget, sum(x) == 1)

optimize!(m5)
xsol = value.(x)
xsol_clean5_FM = zeros(n_FM,1)

for i in 1:n_FM
    if xsol[i] > 1e-5
        xsol_clean5_FM[i] = xsol[i]
    end
end

ix_FM = findall(xsol .> 1e-5) # pull all xs with positive value
ret_FM = dot(rr_FM,xsol_clean5_FM) # calculate rate of return
sdev_FM = sqrt(dot(xsol_clean5_FM,ΣFM*xsol_clean5_FM)) # calculate std

println()
println("************ MODEL 5 ************")
println("Expected return of optimal allocation: ", ret_FM, " Std dev of
 optimal allocation: ", sdev_FM)
println(length(ix_FM)," stocks with a nonzero allocation")
println(objective_value(m5)*100)  #multiply by 100 to have the optimal p
ortfolio maximum drawdown in %
println()
selected_stocks = SP_tickers_FM[ix_FM]
allocations = 100 .*xsol[ix_FM]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end
```

```
# ---------------- Model 6 ---------------- #
m6 = Model(Gurobi.Optimizer)
```

```
set_optimizer_attribute(m6, "OutputFlag", 0)

M = 0.5

@variable(m6, 0<=x[1:n_FM]<=0.5)
@variable(m6, y)  # epigraph trick variable
@variable(m6, z[1:n_FM], Bin)

@objective(m6, Max, y)

@constraint(m6,epi[t in 1:T_FM], y <= dot(SPstocks_clean_rets_FM[t,:],x
))
@constraint(m6, dot(rr_FM,x) >= ϱ)
@constraint(m6, sum(x) == 1)
@constraint(m6, logic1[i in 1:n_FM], x[i] >= 0.05*z[i])
@constraint(m6, logic2[i in 1:n_FM], x[i] <= M*z[i])

optimize!(m6)
xsol = value.(x)
xsol_clean6_FM = zeros(n_FM,1)

for i in 1:n_FM
    if xsol[i] > 1e-5
        xsol_clean6_FM[i] = xsol[i]
    end
end

ix_FM = findall(xsol .> 1e-5) # pull all xs with positive value
ret_FM = dot(rr_FM,xsol_clean6_FM) # calculate rate of return
sdev_FM = sqrt(dot(xsol_clean6_FM,ΣFM*xsol_clean6_FM)) # calculate std

println()
println("************ MODEL 6 ************")
println("Expected return of optimal allocation: ", ret_FM, " Std dev of
 optimal allocation: ", sdev_FM)
println(length(ix_FM)," stocks with a nonzero allocation")
println(objective_value(m6)*100)  #multiply by 100 to have the optimal p
ortfolio maximum drawdown in %
println()
selected_stocks = SP_tickers_FM[ix_FM]
allocations = 100 .*xsol[ix_FM]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end
```

```
374
Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 1 ************
Optimized expected return: 0.05224090328146336 Optimized std dev: 1.696
413662642256
8 stocks with a nonzero allocation

Allocate funds in the following way:
AKAM: 4.39%
AOS: 5.11%
CHRW: 7.71%
CLX: 6.07%
CAG: 10.76%
GILD: 1.14%
SJM: 13.96%
TIF: 50.86%
Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 2 ************
Optimized expected return: 0.12222617991131493 Optimized std dev: 1.732
049634851219
10 stocks with a nonzero allocation

Allocate funds in the following way:
AKAM: 1.79%
AOS: 3.04%
CHRW: 6.87%
CLX: 11.68%
CAG: 8.21%
EQT: 1.77%
GILD: 4.93%
SJM: 14.02%
REGN: 0.64%
TIF: 47.06%

************ MODEL 5 ************
Expected return of optimal allocation: 0.32213187057943116 Std dev of o
ptimal allocation: 2.257225087355545
7 stocks with a nonzero allocation
-3.2752828459656103

Allocate funds in the following way:
AKAM: 8.65%
CLX: 28.62%
CAG: 3.03%
EA: 3.11%
GILD: 4.1%
RRC: 13.44%
TIF: 39.06%
Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 6 ************
Expected return of optimal allocation: 0.29790999476077257 Std dev of o
```

```
ptimal allocation: 2.1927769393719725
7 stocks with a nonzero allocation
-3.2822751947614783

Allocate funds in the following way:
AKAM: 7.58%
AOS: 5.0%
CLX: 25.66%
CAG: 5.14%
GILD: 5.64%
RRC: 12.37%
TIF: 38.62%
```

3.D. Code for sensitivity analysis

Back to index

The remaining code cells are devoted to the sensitivity analysis portion of the project.

In [13]:
```
using DataFrames, CSV
# Load price (not return) data May - Aug
Prices_MA = CSV.read("_stocks_SP_clean.csv",header=2,delim=',')  #file m
ust be in same directory as this notebook
Prices_MA = Prices_MA[2:end,2:(n+1)]; #drop empty row, date and extra co
lumns

# AGN is in Feb-May but not in May-Aug
AGN_ix = findfirst(isequal("AGN"), SP_tickers_FM)
println(AGN_ix) # so we will need to delete position 22 in xsol_clean ve
ctor
SP_tickers_FM_clean = filter(e->e != "AGN", SP_tickers_FM); #delete AGN
 column
Prices_MA_filt = Prices_MA[SP_tickers_FM_clean];

xsol_clean1_FM_noAGN = xsol_clean1_FM[1:end .!= 22]; #remove position 22
(AGN)
xsol_clean2_FM_noAGN = xsol_clean2_FM[1:end .!= 22]; #remove position 22
(AGN)
xsol_clean5_FM_noAGN = xsol_clean5_FM[1:end .!= 22]; #remove position 22
(AGN)
xsol_clean6_FM_noAGN = xsol_clean6_FM[1:end .!= 22]; #remove position 22
(AGN)
```

```
22

┌ Warning: `getindex(df::DataFrame, col_inds::Union{AbstractVector, Reg
ex, Not})` is deprecated, use `df[:, col_inds]` instead.
│   caller = top-level scope at In[13]:11
└ @ Core In[13]:11
```

In [18]:
```
# Define function to get portfolio return for specified period and portf
olio
function getPortRet(M,x)
    #input: T by n matrix of stock prices, vector of weights
    #output: Portfolio Return at the end of the period(float)

    M = Matrix(M) #just in case it's not a matrix

    Ret_vec = (M[end,:] - M[1,:]) ./ M[1,:]
    Ret_vec = 100 .* Ret_vec  #to have them in %

    pr = dot(Ret_vec,x) #portfolio return of the period

    return pr
end

PR1 = getPortRet(Prices_MA_filt,xsol_clean1_FM_noAGN);
PR2 = getPortRet(Prices_MA_filt,xsol_clean2_FM_noAGN);
PR5 = getPortRet(Prices_MA_filt,xsol_clean5_FM_noAGN);
PR6 = getPortRet(Prices_MA_filt,xsol_clean6_FM_noAGN);
println("PR1 = ",PR1, ", PR2 = ",PR2, ", PR5 = ",PR5, ", PR6 = ",PR6)
```

```
PR1 = 5.67676508412911, PR2 = 5.3177207868025045, PR5 = 9.3765791333089
6, PR6 = 8.537634479497331
```

In [25]:
```
# Data perturbation for empirical sensitivity analysis

using DataFrames, CSV, Statistics, Random, Distributions

SPstocks_clean_rets_mat = Matrix(SPstocks_clean_rets);

# Get variance (in % squared) from original covariance matrix imported a
t the beginning of this big section
variances = zeros(n)
for i in 1:n
    variances[i] = Σ[i,i]
end

sdevs = sqrt.(variances)

new_return_mat = deepcopy(SPstocks_clean_rets_mat)  #copy values, not ad
dress

# Add perturbation to returns proportional to the std deviation of each
 stock
Random.seed!(1234);
for t in 1:T
    for s in 1:n
        perturb = rand(Normal(0,sdevs[s]),1)/1000
        new_return_mat[t,s] = new_return_mat[t,s] + perturb[1]
    end
end

diff_mat = SPstocks_clean_rets_mat - new_return_mat
diff_mean_vec = mean(diff_mat,dims=1) #average perturbation by stock
mean_return_vec = mean(SPstocks_clean_rets_mat,dims=1)
relative_diff_mean_vec = diff_mean_vec ./ mean_return_vec

println(minimum(relative_diff_mean_vec))
println(maximum(relative_diff_mean_vec))
println(mean(relative_diff_mean_vec)) #close to 0 on average

#new expected returns
println(round.(rr[1:9],digits=5))
rr_new = 100 .*mean(new_return_mat,dims=1)
println(round.(rr_new[1:9],digits=5)) #similar but noticeably different
 from original returns
#new covariance matrix
Σnew = 10000 .*Statistics.cov(new_return_mat); #in % squared
rr_new = reshape(rr_new,(n,1))
println(mean(rr_new-rr)) #0.3%
cov_change = Σ - Σnew
pseudoFrobenius = mean(abs.(cov_change)) #0.11755
avg_cov_element = mean(abs.(Σ)) #4.2113
```

```
-540.756589123579
371.2817318971066
-0.35064397810133424
[0.05861, 0.20369, 0.24793, 0.37649, 0.40757, 0.37334, 0.42608, 0.7697
5, 0.43222]
[0.04558, 0.21732, 0.20177, 0.37314, 0.37107, 0.36764, 0.44075, 0.7487
2, 0.45277]
0.003075395129690682
```

Out[25]: 4.211336645860046

In [26]:
```
# Solving again Models 1, 2, 5, 6 with perturbed returns

# ---------------- Model 1 ---------------- #
m1 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m1, "OutputFlag", 0)

@variable(m1, x[1:n]>=0)

@objective(m1, Min, dot(x,Σnew*x) )  #note: it's equivalent to consider
 the variance or std dev = sqrt(variance)

@constraint(m1, dot(rr_new,x) >= ϱ)
@constraint(m1, budget, sum(x) == 1)

optimize!(m1)
xsol_new = value.(x)
xsol_clean1_new = zeros(n,1)

for i in 1:n
    if xsol_new[i] > 1e-5
        xsol_clean1_new[i] = xsol_new[i]
    end
end

ix_new = findall(xsol_new .> 1e-5) # pull all xs with positive value
ret_new = dot(rr_new,xsol_clean1_new) # calculate rate of return
sdev_new = sqrt(dot(xsol_clean1_new,Σnew*xsol_clean1_new)) # calculate s
td

println()
println("************ MODEL 1 ************")
println("Optimized expected return: ", ret_new, " Optimized std dev: ",
sdev_new)
println(length(ix_new)," stocks with a nonzero allocation")
println()
selected_stocks = SP_tickers[ix_new]
allocations = 100 .*xsol_new[ix_new]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end
diff_alloc = 100 .*(xsol_clean1_new - xsol_clean1)  #in %
xsol_active1 = xsol_clean1[xsol_clean1.>0]
println()
println("Diff new-old allocations in pctg points: ", round.(diff_alloc[a
bs.(diff_alloc).>0],digits=2))
println("Average absolute difference = ", round(mean(abs.(diff_alloc[abs
.(diff_alloc).>0])),digits=2), " pctg points")
println("Relative absolute difference in original allocs > 0 = ", round.
(100 .*((xsol_clean1_new[xsol_clean1.>0] - xsol_active1)./xsol_active1),
digits=2))
println("Average of abs value of above vector = ", round.(mean(abs.(100
.*((xsol_clean1_new[xsol_clean1.>0] - xsol_active1)./xsol_active1))),dig
its=2),"%")
println()
```

```
# ---------------- Model 2 ---------------- #
m2 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m2, "OutputFlag", 0)

@variable(m2, x[1:n]>=0)

@objective(m2, Max, dot(rr_new,x) )

@constraint(m2, dot(x,Σnew*x) <= sigma2)  #note: it's equivalent to cons
ider the variance or std dev = sqrt(variance)
@constraint(m2, budget, sum(x) == 1)

optimize!(m2)
xsol = value.(x)
xsol_clean2_new = zeros(n,1)

for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean2_new[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret = dot(rr_new,xsol_clean2_new) # calculate rate of return
sdev = sqrt(dot(xsol_clean2_new,Σnew*xsol_clean2_new)) # calculate std

println()
println("************ MODEL 2 ************")
println("Optimized expected return: ", ret, " Optimized std dev: ", sdev
)
println(length(ix1)," stocks with a nonzero allocation")
println()
selected_stocks = SP_tickers[ix1]
allocations = 100 .*xsol[ix1]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end
diff_alloc = 100 .*(xsol_clean2_new - xsol_clean2)  #in %
xsol_active2 = xsol_clean2[xsol_clean2.>0]
println()
println("Diff new-old allocations in pctg points: ", round.(diff_alloc[a
bs.(diff_alloc).>0],digits=2))
println("Average absolute difference = ", round(mean(abs.(diff_alloc[abs
.(diff_alloc).>0])),digits=2), " pctg points")
println("Relative absolute difference in original allocs > 0 = ", round.
(100 .*((xsol_clean2_new[xsol_clean2.>0] - xsol_active2)./xsol_active2),
digits=2))
println("Average of abs value of above vector = ", round.(mean(abs.(100
.*((xsol_clean2_new[xsol_clean2.>0] - xsol_active2)./xsol_active2))),dig
its=2),"%")
println()



# ---------------- Model 5 ---------------- #
```

```
m5 = Model(Clp.Optimizer)
set_optimizer_attribute(m5, "LogLevel", 0)

@variable(m5, 0<=x[1:n]<=0.5)
@variable(m5, y)  # epigraph trick variable

@objective(m5, Max, y)

@constraint(m5,epi[t in 1:T], y <= dot(new_return_mat[t,:],x))
@constraint(m5, dot(rr_new,x) >= ϱ)
@constraint(m5, budget, sum(x) == 1)

optimize!(m5)
xsol = value.(x)
xsol_clean5_new = zeros(n,1)

for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean5_new[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret = dot(rr_new,xsol_clean5_new) # calculate rate of return
sdev = sqrt(dot(xsol_clean5_new,Σnew*xsol_clean5_new)) # calculate std

println()
println("************ MODEL 5 ************")
println("Expected return of optimal allocation: ", ret, " Std dev of opt
imal allocation: ", sdev)
println(length(ix1)," stocks with a nonzero allocation")
println(objective_value(m5)*100)  #multiply by 100 to have the optimal p
ortfolio maximum drawdown in %
println()
selected_stocks = SP_tickers[ix1]
allocations = 100 .*xsol[ix1]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end
diff_alloc = 100 .*(xsol_clean5_new - xsol_clean5)  #in %
xsol_active5 = xsol_clean5[xsol_clean5.>0]
println()
println("Diff new-old allocations in pctg points: ", round.(diff_alloc[a
bs.(diff_alloc).>0],digits=2))
println("Average absolute difference = ", round(mean(abs.(diff_alloc[abs
.(diff_alloc).>0])),digits=2), " pctg points")
println("Relative absolute difference in original allocs > 0 = ", round.
(100 .*((xsol_clean5_new[xsol_clean5.>0] - xsol_active5)./xsol_active5),
digits=2))
println("Average of abs value of above vector = ", round.(mean(abs.(100
.*((xsol_clean5_new[xsol_clean5.>0] - xsol_active5)./xsol_active5))),dig
its=2),"%")
println()
```

```
# ---------------- Model 6 ---------------- #
m6 = Model(Gurobi.Optimizer)
set_optimizer_attribute(m6, "OutputFlag", 0)

M = 0.5

@variable(m6, 0<=x[1:n]<=0.5)
@variable(m6, y)  # epigraph trick variable
@variable(m6, z[1:n], Bin)

@objective(m6, Max, y)

@constraint(m6,epi[t in 1:T], y <= dot(new_return_mat[t,:],x))
@constraint(m6, dot(rr_new,x) >= ϱ)
@constraint(m6, sum(x) == 1)
@constraint(m6, logic1[i in 1:n], x[i] >= 0.05*z[i])
@constraint(m6, logic2[i in 1:n], x[i] <= M*z[i])

optimize!(m6)
xsol = value.(x)
xsol_clean6_new = zeros(n,1)

for i in 1:n
    if xsol[i] > 1e-5
        xsol_clean6_new[i] = xsol[i]
    end
end

ix1 = findall(xsol .> 1e-5) # pull all xs with positive value
ret = dot(rr_new,xsol_clean6_new) # calculate rate of return
sdev = sqrt(dot(xsol_clean6_new,Σnew*xsol_clean6_new)) # calculate std

println()
println("************ MODEL 6 ************")
println("Expected return of optimal allocation: ", ret, " Std dev of opt
imal allocation: ", sdev)
println(length(ix1)," stocks with a nonzero allocation")
println(objective_value(m6)*100)  #multiply by 100 to have the optimal p
ortfolio maximum drawdown in %
println()
selected_stocks = SP_tickers[ix1]
allocations = 100 .*xsol[ix1]  #to have them in %
println("Allocate funds in the following way: ")
for (i,ticker) in enumerate(selected_stocks)
    println(ticker, ": ", round(allocations[i],digits = 2), "%")
end
diff_alloc = 100 .*(xsol_clean6_new - xsol_clean6)  #in %
xsol_active6 = xsol_clean6[xsol_clean6.>0]
println()
println("Diff new-old allocations in pctg points: ", round.(diff_alloc[a
bs.(diff_alloc).>0],digits=2))
println("Average absolute difference = ", round(mean(abs.(diff_alloc[abs
.(diff_alloc).>0])),digits=2), " pctg points")
println("Relative absolute difference in original allocs > 0 = ", round.
(100 .*((xsol_clean6_new[xsol_clean6.>0] - xsol_active6)./xsol_active6),
digits=2))
println("Average of abs value of above vector = ", round.(mean(abs.(100
```

```
.*((xsol_clean6_new[xsol_clean6.>0] - xsol_active6)./xsol_active6))),dig
its=2),"%")
println()
```

```
Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 1 ************
Optimized expected return: 0.1815324481372167 Optimized std dev: 0.5801
317947470083
17 stocks with a nonzero allocation

Allocate funds in the following way:
ABBV: 2.21%
AMZN: 1.39%
CMG: 2.55%
CLX: 14.51%
CMCSA: 0.08%
ED: 5.75%
COST: 14.32%
D: 3.53%
EBAY: 3.92%
EFX: 8.87%
HOLX: 0.24%
KMB: 4.19%
KR: 7.11%
TIF: 18.31%
TJX: 0.64%
FOX: 2.4%
WMT: 9.98%

Diff new-old allocations in pctg points: [0.87, -0.46, 0.61, 1.74, -2.7
6, 0.79, -4.02, -1.11, 0.22, 1.38, 0.24, -1.8, -2.87, 0.33, 1.91, 0.64,
-0.61, 2.4, 2.52]
Average absolute difference = 1.44 pctg points
Relative absolute difference in original allocs > 0 = [64.67, -25.04, 3
1.29, 13.62, -97.07, 15.91, -21.92, -23.9, 5.91, 18.38, -100.0, -40.7,
4.93, 11.67, -100.0, 33.71]
Average of abs value of above vector = 38.05%

Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 2 ************
Optimized expected return: 0.7776684989702565 Optimized std dev: 1.7320
405421915726
9 stocks with a nonzero allocation

Allocate funds in the following way:
AMD: 0.63%
AAPL: 0.0%
CDNS: 19.33%
EBAY: 19.33%
IDXX: 0.0%
LB: 16.79%
LRCX: 15.73%
TSCO: 14.9%
UPS: 13.29%

Diff new-old allocations in pctg points: [0.63, -0.99, -1.05, 13.11, -1
5.17, -6.56, -4.8, -7.12, -7.95, 0.0, 12.74, 15.73, -0.45, -0.35, -4.4,
```

```
6.63]
Average absolute difference = 6.1 pctg points
Relative absolute difference in original allocs > 0 = [-100.0, -99.89,
210.52, -100.0, -100.0, -100.0, -26.91, -100.0, 314.43, -100.0, -100.0,
-22.79, 99.5]
Average of abs value of above vector = 113.39%


************ MODEL 5 ************
Expected return of optimal allocation: 0.24909977016164828 Std dev of o
ptimal allocation: 0.842466846204954
8 stocks with a nonzero allocation
-0.8331657296043316

Allocate funds in the following way:
CHD: 5.23%
CTAS: 0.79%
CLX: 36.98%
DLTR: 3.06%
JCI: 3.33%
KR: 28.24%
TGT: 0.04%
TIF: 22.33%

Diff new-old allocations in pctg points: [-1.16, 5.23, 0.79, -1.22, -0.
31, 0.25, -2.22, 0.76, -2.48, -0.85, 1.21]
Average absolute difference = 1.5 pctg points
Relative absolute difference in original allocs > 0 = [-100.0, -3.2, -
9.13, 8.0, -100.0, 2.77, -100.0, -95.46, 5.73]
Average of abs value of above vector = 47.14%

Academic license - for non-commercial use only
Academic license - for non-commercial use only

************ MODEL 6 ************
Expected return of optimal allocation: 0.25796826332998307 Std dev of o
ptimal allocation: 0.8406077552025637
6 stocks with a nonzero allocation
-0.8444455614955387

Allocate funds in the following way:
CHD: 7.2%
CLX: 33.62%
DLTR: 5.0%
JCI: 5.0%
KR: 28.53%
TIF: 20.65%

Diff new-old allocations in pctg points: [7.2, -6.73, -0.01, -0.38, -0.
07]
Average absolute difference = 2.88 pctg points
Relative absolute difference in original allocs > 0 = [-16.68, 0.0, -0.
27, -1.33, -0.35]
Average of abs value of above vector = 3.73%
```

4. Results and discussion

4.A. Analysis of Model 6's optimal solution

Back to index

For reasons that will become apparent in the following subsections, Model 6 (MILP formulation of the Maximum dradown model) is our model of choice, at least for a portfolio optimization problem of a similar size to this one (about 400 stocks). Thus, we will focus our solution analysis on the optimal solution this model has found.

According to our proposed model, the three stocks with largest allocation should be CLX (40.34\%), KR (28.92\%), and TIF (20.73\%) (see output of code above or table in next subsection). But what do those tickers mean?

The first one will make a lot of sense to the reader: Clorox. Clorox is a major producer of cleaning products, especially those containing bleach and other strong disinfectants. Needless to say, it seems reasonable that such a stock is picked during a pandemic. Models 1 (12.77\%) and 2 (4.8\%) also pick this stock, but with less emphasis. This article on Yahoo Finance (https://finance.yahoo.com/news/clorox-clx-outperforming-other-consumer-153003179.html) provides more details on the exceptional performance of Clorox so far this year.

The second one is Kroger, the largest supermarket in the US by revenue. There are three main reasons why it makes sense to have a significant allocation on this stock. First, with restaurants being closed or operating at reduced capacity due to the pandemic, supermarkets have seen a rise in demand. Second, Kroger in particular has been able to benefit more than other supermarkets thanks to their investments made in their online sales service since 2017, which allowed Kroger to absorbe the surge in demand (https://www.winsightgrocerybusiness.com/retailers/how-pandemic-accelerated-krogers-transformation), with digital sales climbing 92\% in the three months before May 23. Moreover, Kroger's reputation has surged (https://finance.yahoo.com/news/kroger-ranks-9-2020-axios-134200603.html) in recent times for their COVID-19 response and racial equity support.

Finally, TIF is the ticker for the luxury jewelry company Tiffany & Co., which admittedly seems an odd pick considering the ongoing pandemic. However, there are two reasons that may explain this, and both of them are completely unrelated to the pandemic. First, the plans of French multinational LVMH (sometimes informally called Louis Vuitton) to acquire Tiffany - plans which were approved by Tiffany's stockholders in February (https://finance.yahoo.com/news/tiffany-co-stockholders-approve-acquisition-165835793.html). Second, Tiffany has significantly improved their internal cost structure (https://www.forbes.com/sites/greatspeculations/2020/07/06/how-did-tiffany-stock-manage-to-grow-80-with-weak-revenue-growth/#5f8c996228c6) over the past three years, making the company more profitable.

4.B. In-sample solution comparison

Back to index

In this subsection we provide an exhaustive account of relevant metrics for each model, and discuss the main attributes of each.

The following results are obtained from sample daily data of 393 stocks in the period May 1, 2020 to August 1, 2020.

Model	#cont & bin var, #const	Time (s)	Solution E(r), σ, Max Drawdown	Top 3 stocks
Markowitz 1 [QP]	400 c, 0 b, 400	0.0228	$E(r) = 0.18\%, \sigma = 0.59\%$, MD $= -2.00\%$	COST, TIF, CLX
Markowitz 2 [QP]	400 c, 0 b, 400	0.0184	$E(r) = 0.59\%, \sigma = 1.00\%$, MD $= -3.18\%$	EBAY, TSCO, CHD
Mean-Variance [QP]	400 c, 0 b, 400	0.0200	$E(r) = 0.70\%, \sigma = 1.36\%$, MD $= -4.80\%$	EBAY, TSCO, UPS
Max Drawdown 1 [LP]	400 c, 0 b, 450	0.0026	$E(r) = 0.22\%, \sigma = 0.84\%$, MD $= -0.85\%$	CLX, KR, TIF
Max Drawdown 2 [MILP]	400 c, 400 b, 1250	0.0001	$E(r) = 0.23\%, \sigma = 0.84\%$, MD $= -0.86\%$	CLX, KR, TIF

Our poposed Max Drawdown models seem to strike a good balance return - standard deviation between the different variations of the Markowitz model. Although possibly not the most useful metric given that risk is measured differently in some of the models above, the ratios of expected return / standard deviation are, for each model: 0.31, 0.59, 0.51, 0.26, 0.27. In addition, our proposed models achieve a much lower maximum drawdown, i.e. the worst performance day is not quite as bad as in the other models. All in all, it seems fair to claim that the Max Drawdown models provide an expected return on par with some Markowitz variations but at a lower risk also considering the maximum drawdown. This, coupled with a much faster solving time make our proposed portfolio optimization models so far a superior choice compared to the Markowitz models, at least on paper. See next subsection for an out-of-sample performance comparison.

It is remarkable that the MILP variation is the fastest, by far: 200 times faster than the QPs, and 25 times faster than the LP. As briefly discussed in a previous section, this is due to the fact that Gurobi is a highly optimized solver for MIP and is able to take advantage of the the particular structure of our portfolio optimization problem. According to Gurobi (https://www.gurobi.com/resource/mip-basics/), their sophisticated MIP algorithm (a combination of **Branch-and-Bound** and **Cutting plane method**, i.e. the fancier **Branch-and-cut**) first does some heavy **preprocessing** to reduce the problem dimension, and then solves a **linear programming relaxation** of the problem via Simplex iterations; it seems plausible that by requiring a minimum of a 5\% allocation if a stock is chosen, this might already discard many 'bad' stocks based on the objective in the preprocessing stage, which greatly reduces the combinatorial complexity of the MILP. It seems from the solver's output (see screenshot below) that after adding just 2 cuts to the original LP relaxation and solving that instance, Gurobi managed to recover an integer feasible solution (and hence solving the entire original MILP) without even needing to do any branching. Indeed, the leftmost column in the table of the screenshot below shows that no nodes beyond node 0 (root node) are explored in the branch-and-bound tree, so no branching is really done and hence why the process is so fast.

4.C. Out-of-sample solution comparison

Back to index

In this subsection we report the results obtained by each model (except Model 3, for the sake of clarity in an already-long code section) trained with data from February 1 to May 1 of the present year (three months), and tested on the subsequent three months (May 2 - August 1). This way we can compare the portfolio optimization solution **out of sample**. Of course, the following results are at best a very slight indication of true performance, since a similar experiment should be repeated multiple times to gather enough evidence to conclude which model is likely the 'best'.

The idea is to simulate what would have happened if we had optimized our portfolio according to each of the 4 models above on May 1 with data from the previous three months, and then we had checked on August 1 how well each portfolio did. The reason why we wait until August 1 to check is because we optimize our portfolio with 3-month data so it's only natural the investment horizon should also be 3 months (with the implicit and naive assumption that history repeats itself at least to some relevant extent).

The results are as follows:

Model	Portfolio 3-month return
Markowitz 1 [QP]	+5.68\%
Markowitz 2 [QP]	+5.32\%
Max Drawdown 1 [LP]	+9.38\%
Max Drawdown 2 [MILP]	+8.54\%

The above results seem very good, but there are several qualifying remarks we must make:

First, they're actually not that great compared to the benchmark (the S&P 500, where the portfolio constituents trade), which rose by over 15\% in the same period (https://finance.yahoo.com/quote/%5EGSPC/history?period1=1588291200&period2=1596326400&interval=1d&filter=history&frequency=1d). Just out of curiosity, most of the stocks selected by the models are 'right' except for one big mistake with TIF, which lost slightly (about 1\%) in May - Aug, and it's weighted heavily in all models. Still, even with that mistake, the overall portfolio return is not bad in absolute terms, even if it underperformed the market.

Second, it seems that the Markowitz models and our proposed models form two different clusters (delivering around 5.5\% return compared to around 9\% return). Much more evidence would be needed to make the stronger claim that our proposed models are 'better' than the Markowitz model (or, in statistics terms, the expected return of our models is siginificantly higher than that of the Markowitz models), but at least from this small experiment it does not seem impossible.

Third, we notice that the out-of-sample performance is significantly worse than the in-sample performance i.e. the return that the optimal solution promised. For example, Model 2 'promised' a little over 0.122\% return daily on average, which over a 3-month period (62 trading days) should be about 8\%, not 5.3\%. Or, even worse in this case, Model 6 promised about 0.258\% return daily on average which in 62 trading days would have been over 17\% return, but delivered only half of that, 8.5\%. This is also a sign that the optimal solutions are not very robust to changes in the parameter values and they overfit the data.

Finally, we must stress again that no conclusions can be extracted from this small experiment, beyond the fact that the performance of the models, even if fell short of (naive) expectations, doesn't seem to be terrible. Still, this small experiment is worthwile, if anything, as a sanity check that our models have some hope of being reasonable (both in technical implementation and in performance). It would be interesting to see how the models perform compared to the market in several other periods, in particular in periods when the market did not do so well, but this is beyond the scope of the current project, which is primarily focused on the optimization aspect and not the investment management side of portfolio optimization.

4.D. Sensitivity analysis

Back to index

a) On the right hand side **(RHS) of primal constraints**

In general, in all models considered, the only RHS that is in our opinion potentially interesting to check is the RHS of the constraint $\sum_{i=1}^{n} x_i = 1$, and it can be interpreted as the value of having access to 1 unit more of budget e.g. through borrowing (**shadow price** of this constraint). However, a more careful analysis reveals that, since we work with stock returns and not absolute monetary gains, it is in fact pointless to ask how valuable it would be to have an additional unit of budget. Here's why: stock returns are invariant to budget size (only the different proportions of the budget invested matter), and the standard deviation ('risk') of the portfolio in fact increaseas as the budget increases, which after more carful thought seems obvious because if you have more budget you can, well, lose more budget.

The same argument applies to Model 5, since the maximum drawdown is, at the end of the day, a return, and so invariant to budget size.

For Model 6 no such sensitivity analysis is possible since there's no well-defined concept of "Dual" in MIP.

b) On the **primal objective coefficients**

The primal objective coefficients of Models 4, 5 and 6 are not interesting since they are just auxiliary variables used in the epigraph trick.

On the other hand, it may be interesting to see how sensitive the solution (vector of budget allocations) is to changes in the vector of expected returns and/or (careful) changes in the covariance matrix, for now restricting attention to when they appear in the objective. Thus, these changes leave the original primal **feasible region** unchanged, so the original optimal solution is still feasible. For ease of exposition, let's focus on Model 2 (maximize expected return subject to a maximum allowable variance). In this new scenario, the dot product of the original optimal solution vector and the new vector of expected returns is a **lower bound** to the new maximization problem. To obtain an **upper bound**, though, we would need to find a feasible point of the (new) **Dual** (by **Weak Duality in convex programs**). However, finding the Dual is not trivial since the Primal is a **quadratically constrained quadratic program** and the **Lagrangian** is a function of x, λ, ν where x is an n-dimensional vector, which suggests the use of more sophisticated methods (e.g. matrix differentiation). Even if that was not an issue, the resulting Dual objective is very messy, as the reader can verify below (in un-simplified form):

$$\frac{-r - \nu e}{2\lambda}\Sigma^{-1} r + \frac{-r - \nu e}{2}\left(\frac{-r - \nu e}{2\lambda}\right)^T - \lambda\sigma_0 + \nu\frac{-r - \nu e}{2\lambda}\Sigma^{-1} e - \nu$$

Note e is an n-dimensional vector of ones, and Σ is the covariance matrix. Also note Σ^{-1} exists since the covariance matrix of risky assets is positive definite, so has nonzero (positive) determinant.

At this point we still need to find a feasible dual point. Note that even if we manage to somehow find one by inspection, the upper bound that it would provide to the new primal may be arbitrarily bad.

However, we can still obtain valuable insight by adding a small perturbation to the data on stock returns, which will then automatically add some perturbation to the covariance matrix (and will do so preserving its defining properties since we are not perturbing the covariance matrix directly 'by hand'). Then we can see how the optimal budget allocation of Models 1, 2, 5 and 6 change.

Note that if we go this route we are no longer restricted to making changes in the coefficients of the primal objective or primal RHS (dual objective coefficients), but also in the coefficients of the left hand side (**LHS**) of the primal constraints.

We have performed this empirical **sensitivity analysis** in the last portion of the previous section and here we discuss the results.

We have modeled the **random perturbation** of the return of stock s at time t, $p_{s,t}$ that we mentioned previously, as follows:

$$p_{s,t} = N(0, \sigma_s)/c$$

where N signifies a **Normal distribution** centered at 0 and with standard deviation σ_s, divided by a positive constant c, which in our case is 10^3 to suit the scale of our daily returns.

As it can be seen in the code section above, the new perturbed mean returns are similar to the original ones but noticeably different. For convenience, we show again here (second row) the first 9 perturbed average daily returns, corresponding to the first 9 stocks, compared to the original average daily returns (first row):

$$0.0586, 0.2037, 0.2479, 0.3765, 0.4076, 0.3733, 0.4261, 0.7698, 0.4322$$
$$0.0456, 0.2173, 0.2018, 0.3731, 0.3711, 0.3676, 0.4408, 0.7487, 0.4528$$

Note that some average returns are closer to the original than others, and this is intended: the **size of the random perturbation is proportional to the standard deviation** of each stock, which is a more realistic model assumption compared to a fixed random shock applied to all stocks (or completely arbitrary perturbations) since more volatile stocks should have bigger price fluctuations across time. Also note that some perturbed mean returns are higher than the original ones, while others are lower, further adding to the realism of the experiment.

The new mean returns are on average 0.3\% higher than the original ones, but we found that the more relevant factor is that the covariance matrix has changed too. We tried first adding a constant (stock-dependent) perturbation to the entire time series of each stock to change the expected return but not the covariance matrix, and the optimal allocations did not change. However, now that the covariance has changed the allocations have changed too.

We have decided to measure the change in covariance matrix by computing the average absolute pairwise difference between the original and perturbed covariance matrices. This is similar to the **Frobenius norm** of the difference matrix, but has a more straightforward interpretation: average absolute change in each element of the covariance matrix is $0.11755\%^2$. Considering that the average size of an element in the original covariance matrix is $4.2113\%^2$, the size of the perturbation represents roughly a 2.8\% change in each covariance element, on average.

There are several different ways to capture the change in allocation, and in the code section above the reader can glance at various metrics printed. However, we believe that the most meaningful metric is to see the percentage by which the new allocation differs from the original one, restricting the comparison to the positive original allocations (because if the original allocation is 0 and now is positive, the change would be something like '+infinite \%' which is not very meaningful and would completely dillute all other finite changes). Then we take the average of the absolute value of those percentage differences.

The table below shows that the average absolute allocation change in each model is between 38% and 40% for the Markowitz models, 47% for the linear Maximum Drawdown model, and only 3.7% for the MILP Maximum Drawdown model. With the exception of the last model, and considering that on average each covariance element was changed by 2.8\% only, the sensitivity analysis performed reveals the **fragility** of the optimal solution to a small change in the parameter values.

Model	Avg abs change in allocation
Markowitz 1 [QP]	38.05\%
Markowitz 2 [QP]	39.82\%
Max Drawdown 1 [LP]	47.14\%
Max Drawdown 2 [MILP]	3.73\%

The difference in robustness is striking. The explanation is as follows: the 'pure' Maximum Drawdown model is inherintly less robust than a Markowitz model since it has an extreme metric (the minimum observed return) as the optimization objective. However, the Maximum Drawdown model becomes more robust once we disallow small allocations (below 5% in our case), and so it is expected that at least the choice of (fewer) stocks is more similar to the original model, and so we have a better chance of avoiding -100% changes (stocks dropped). In general, each stock selected in the original model should be 'more strongly selected' than in the 'pure' version of the model, since it is selected knowing that at least we will allocate 5%. Intuitively, we are **reducing overfitting**, and hence increasing the robustness of the model ('out-of-sample performance' in the Machine Learning jargon).

Therefore, it seems that a simple way to **increase the robustness** of any given portfolio optimization model is to require a minimum allocation for each selected stock (at the expense of turning the portfolio optimization model into a MILP, but it seems this is not necessarily an issue for problems with a size and structure similar to ours, especially with today's solvers and computational power). Naturally, the previous claim would require a deeper analysis to be more conclusive, but it seems at least a reasonable conjecture in light of the evidence we have and the intuitive connection with overfitting.

There is no denying that our attempt at increasing the robustness of the optimal solutions by requiring a minimum allocation is less sophisticated than the one Dai and Wang [3] and other authors before them discuss, which is based on modeling parameter uncertainty, venturing into the realm of stochastic programming. This is certainly a possible extension to our project. Another way Dai and Wang [3] try to improve the robustness of the solution is by promoting sparsity through L_1 regularization. This is interesting, because we have proved in subsection 2.4 that L_1 regularization has no effect in portfolio optimization. The answer to this apparent contradiction is that Dai and Wang [3] allow short-selling, i.e. they do not impose $x_i \geq 0$ for all i, while we do, and we use this fact quite crucially in our proof.

Finally, a word of caution: even if in a relevant sensitivity metric like the one discussed above, minimum allocation constraints seem to greatly improve the robustness of the optimal solution, we cannot forget the fact that the MILP model was the one that fell short of their 'promised' return the most in subsection 4.A., which is a sign that it is, in the end, not that robust if robustness is measured by the differencce in promised vs delivered return (although, to be fair, what was 'promised' is an expected return, not a realized return, so in fact it is possible it has stayed true to its promise after all).

5. Conclusion

Back to index

The aim of this project is to analyze different portfolio optimization models and propose a faster, more robust model with a comparable ratio of portfolio expected return to risk (as measured by standard deviation and maximum drawdown). We hope these findings will be helpful in further research on this topic.

To that end, we reviewed the classical Markowitz model together with two well-known variations, while providing some additional analysis on those QP models e.g. confirmed Strong Duality holds by verifying Slater's conditions, proved that L_1 regularization has no effect in portfolio optimization (in the standard case short-selling is not allowed), and devised a heuristic to more objectively choose the λ parameter when simultaneously optimizing mean and variance (this last one included within the Solution section).

Then we introduced a fourth model from 1991 that is one of the first examples of an LP formulation of portfolio optimization, and discussed its similarities and (theoretical) advantages compared to the classical QP formulations. This model relies on the epigraph trick seen in class to turn a non-linear objective (due to the presence of an absolute value) into an LP.

We followed this discussion by presenting a new model that provides an alternative linearization of the classical QP formulation. The model is first presented as a maximin problem (hence non-linear), and then it is turned into an LP by again using the epigraph trick. We observe a solution speed-up by nearly a factor of 10 compared to the classical QPs, while providing similar (if not better) in-sample and out-of-sample performance. The advantages of using a downside risk measure in the model are briefly discussed.

We then propose a variation of this new model, which turns the LP into a MILP by restricting minimum allocations to 5\% if any budget is allocated to a given stock. In theory, this formulation has worse optimization properties as it should be much harder to solve (MIPs are in general NP-complete). However, due to the particular structure of the problem (including choosing the best possible 'big-M' value) and the use of a highly optimized MIP solver, in practice this variation is about 25 times faster than the LP formulation, and thus 200 times faster than the QPs, while providing a level of performance on par with the other models on most metrics. One exception is in terms of robustness: the MILP formulation seems to be much *more* robust, at least based on how much the optimal solution changes when the parameter values change. We argue this is due to the minimum allocation restriction, which helps reduce overfitting. The other models, on the other hand, are shown to be quite sensitive to perturbations in parameter values, especifically to changes in the covariance matrix.

The MILP formulation being the most convincing one, we analyzed its optimal solution in depth, arguing why its top three stock choices seem reasonable in the current times of COVID.

A natural extension to this project is considering a multi-period model using Machine Learning and/or stochastic programming. In a multi-period model, the goal is to optimize the expected return and standard deviation of a portfolio considering more than one period. In our case, one possibility is to forecast to some extent the returns of the stocks under consdieration for some future periods (possibly with the help of Machine Learning techniques), and optimize the current portfolio accordingly. Another, related possibility is instead of trying to predict the future with sophisticated machinery (pun intended), 'just' make (educated) assumptions on the distribution of returns, and then make use of stochastic programming to capture the inherent non-deterministic nature of the future parameter values (scenario modeling). It is unclear, though, how much there is to be gained from increasing the sophistication of the optimization approach by several orders of magnitude in either of the two ways we have just outlined. But it is clearly interesting and potentially useful to find out.

Finally, the results shown by the MILP formulation in terms of increased speed and robustness while prodividng a comparable (if not better) in-sample ratio risk-reward seem promising, but further research is necessary to confirm them. A first step towards that direction would be to repeat, multiple times, the entire empirical analysis conducted in this project (in sample and out of sample) for different stocks (also different number of stocks) and times (both in horizon length and economic cycle).

Legal disclaimer: the stocks mentioned in the solution of the models are just for illustrative and academic purposes. In particular, they are not to be interpreted as stock recommendations, and any investment decision made as a result of reading this project is exclusively at the reader's discretion.

References

Back to index

[1] H. Markowitz, "Portfolio selection", The Journal of Finance Vol. 7, pp. 77-91, March 1952.

[2] H. Markowitz, *Portfolio selection: Efficient diversification of investments*. New York: John Wiley & Sons, 1959.

[3] Z. Dai and F. Wang, "Sparse and robust mean–variance portfolio optimization problems" Physica A Vol. 523, pp. 1371–1378, April 2019.

[4] H. Konno and H. Yamazaki, "Mean-Absolute Deviation Portfolio Optimization Model and Its Applications to Tokyo Stock Market", Management Science Vol. 37, no. 5, pp. 519-531, 1991.

[5] A. Chekhlov, S. Uryasev, and M. Zabarankin, "Portfolio optimization with drawdown constraints" Supply chain and finance, pp. 209-228, 2004.

In [ ]: