Markowitz portfolio optimization using MOSEK. MOSEK Technical report: TR-2009-2

Erling D. Andersen, Joachim Dahl and Henrik A. Friberg*

Revised on March 14th, 2012.

Abstract

In this tutorial paper we introduce different approaches to Markowitz portfolio optimization, and we show how to solve such problems in MATLAB, R and Python using the MOSEK optimization toolbox for MATLAB, the Rmosek package, and the MOSEK Python API, respectively. We first consider conic formulations of the basic portfolio selection problem, and we then discuss more advanced models for transaction costs.

1 Markowitz portfolio selection

We start by reviewing basic Markowitz portfolio optimization, while introducing the necessary notation used in the remaining tutorial. In Markowitz portfolio selection we optimize a portfolio of assets based on a simple probabilistic model of the stock market. Traditionally we assume that the return of n different assets over time can be modeled as a multivariate random variable $r \in \mathbb{R}^n$ with known mean

$$\mathbf{E}\, r = \mu_r$$

and covariance matrix

$$\mathbf{E}(r - \mu_r)(r - \mu_r)^T = \Sigma_r,$$

where the mean is a vector, and Σ_r is the covariance matrix (i.e., $\mu_r \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n \times n}$). We have an initial holding of w_j^0 dollars of asset j, $j = 1, \ldots, n$ and we invest x_j dollars in asset j, i.e., after the investment period our portfolio is $w^0 + x$. The return of our investment is also a random variable,

$$y = r^T(w^0 + x),$$

with mean value (or *expected return*)

$$\mathbf{E}\, y = \mu_r^T(w^0 + x) \tag{1}$$

and variance (or *risk*)

$$(y - \mathbf{E}\, y)^2 = (w^0 + x)^T \Sigma_r (w^0 + x). \tag{2}$$

In Markowitz portfolio selection, we seek to optimize a trade-off between expected return and risk.

*MOSEK ApS, Fruebjergvej 3, Box 16, 2100 Copenhagen, Denmark. Email: support@mosek.com

1.1 Estimating the mean and variance

In practice, the mean return μ_r and covariance Σ_r must be estimated based on empirical data, which is often a non-trivial task. In this tutorial we estimate those entities using simple *sample estimates*; more details are given in Appendix C.

We assume that for all n assets, we have recordings of N stock returns at different times, and we organize those measurements in a matrix $X \in \mathbb{R}^{N \times n}$, where column j represents the return of asset j over time. This is similar to how such data would be represented in a spreadsheet with n columns and N rows. The sample mean $\bar{r}$ and covariance $\bar{\Sigma}_r$ can easily be estimated.

MATLAB

```
% Estimate sample mean and variance of X
mu_r = mean(X, 1);
Sigma_r = cov(X);
```

R

```
# Estimate sample mean and variance of X
mu_r <- colMeans(X);
Sigma_r <- cov(X);
```

A recurring theme in this tutorial is *conic quadratic* formulations (see Appendix A), which are well-suited for the MOSEK optimizer. In the following conic formulations of Markowitz portfolio selection, we need a factorization of the covariance Σ_r. The covariance matrix can be factored as

$$\bar{\Sigma}_r = G^T G,$$

where the G matrix depends on the choice of factorization. For any such factorization, we can then express the risk as a Euclidean norm, i.e.,

$$(w^0 + x)^T \bar{\Sigma}_r (w^0 + x) = (w^0 + x)^T G^T G (w^0 + x) = \|G(w^0 + x)\|^2,$$

which is used in the conic formulations in the following sections.

The sample estimate is defined as a factorization (or a *Grammian matrix*)

$$\bar{\Sigma}_r \triangleq \bar{X}^T \bar{X},$$

where

$$\bar{X} \triangleq \frac{1}{\sqrt{N-1}}(X - e\bar{r}^T)$$

denotes a scaled zero-mean version of the data-matrix, and e is a vector of all ones. We can therefore express the risk directly in terms of the (normalized) data-matrix $\bar{X}$ as

$$(w^0 + x)^T \bar{\Sigma}_r (w^0 + x) = (w^0 + x)^T \bar{X}^T \bar{X} (w^0 + x) = \|\bar{X}(w^0 + x)\|^2. \tag{3}$$

The factored risk expression in (3) is the one most frequently used in Markowitz portfolio optimization, and makes no assumptions about the dimensions or the rank of X - it can be employed independent of whether we have more observations than assets (i.e., whether X has more rows than columns). Moreover, it is not required that X has full rank.

MATLAB

```
% Normalize data-matrix for conic formulations of risk
[N, n] = size(X);
mu_r = mean(X,1);
Xbar = 1/sqrt(N-1)*(X-ones(N,1)*mu_r);   G = Xbar;
```

R

```
# Normalize data-matrix for conic formulations of risk
N <- nrow(X);  n <- ncol(X);
mu_r <- colMeans(X);
Xbar <- 1/sqrt(N-1)*(X-matrix(1,nrow=N)%*%mu_r);  G <- Xbar;
```

Alternatively, we can use a QR factorization on the normalized data-matrix

$$QR = \bar{X},$$

where $Q \in \mathbb{R}^{N\times N}$ is an orthogonal matrix (i.e., $QQ^T = Q^TQ = I$) and $R \in \mathbb{R}^{N\times n}$ is an upper-triangular matrix. Since Q is orthogonal, we have that

$$(w^0 + x)^T\bar{\Sigma}_r(w^0 + x) = (w^0 + x)^T R^T Q^T QR(w^0 + x) = \|R(w^0 + x)\|^2.$$

MATLAB

```
% Compute a factorization of the normalized data-matrix
[Q, R] = qr(Xbar,0);    G = R;
```

R

```
# Compute a factorization of the normalized data-matrix
z <- qr(Xbar);  Q <- qr.Q(z);  R <- qr.R(z);  G <- R;
```

A related approach to the QR factorization is the *singular-value decomposition* (SVD) of $\bar{X}$,

$$\bar{X} = USV^T,$$

where $U \in \mathbb{R}^{N\times N}$ is an orthogonal matrix, $S \in \mathbb{R}^{N\times n}$ is a diagonal matrix with non-negative *singular values* appearing in decreasing order on the diagonal,

$$S_{11} \geq S_{22} \geq \cdots \geq S_{pp} \geq 0,$$

where p is the rank of $\bar{X}$, and $V \in \mathbb{R}^{n\times n}$ is another orthogonal matrix. As for the QR factorization, we get a factored expression for the risk,

$$(w^0 + x)^T\bar{\Sigma}_r(w^0 + x) = (w^0 + x)^T VS^T U^T USV^T(w^0 + x) = \|SV^T(w^0 + x)\|^2.$$

The SVD is often used for creating *low-rank approximations*, i.e., by truncating the smallest singular values to zero, we obtain a low-rank approximation $\hat{X}$ of $\bar{X}$. This is often useful for robust modeling and for removing noise from the measurements, but is beyond the scope of this tutorial.

MATLAB

```
% Compute a singular value decomposition of the normalized data-matrix
[U, S, V] = svd(Xbar,0);  G = S*V';
```

R

```
# Compute a singular value decomposition of the normalized data-matrix
z <- svd(Xbar); U <- z$u; S <- diag(z$d); V <- z$v; G <- tcrossprod(S,V);
```

We conclude this section by mentioning the simpler *Cholesky* factorization, which can be used to factor symmetric positive definite matrices, e.g., if $\bar{\Sigma}_r$ is positive definite, we can factor it as

$$\bar{\Sigma}_r = R^T R,$$

where R is an upper-triangular matrix.

MATLAB

```
% Compute a Cholesky factorization of the covariance matrix
Sbar = Xbar'*Xbar;
R = chol(Sbar);   G = R;
```

R

```
# Compute a Cholesky factorization of the covariance matrix
Sbar <- crossprod(Xbar);  # Same as t(Xbar) %*% Xbar
R <- chol(Sbar);   G <- R;
```

In general, it is not recommended to use the Cholesky factorization approach for factoring Σ_r instead of directly using the (normalized) data matrix, or a QR factorization of the data-matrix. When we have fewer observations than assets, the data matrix has smaller dimensions than the covariance matrix, so from a computational point of view it is more attractive. Furthermore, in that case, the covariance matrix is rank-deficient, and as such the Cholesky factorization is not directly applicable. On the other hand, when we have more observations than assets, the covariance matrix will generally have full rank and its Cholesky factor has smaller dimensions than the data matrix.

The economy size QR factorization unifies the two cases; it yields the smallest factor, and can be used regardless of the rank of the data-matrix and covariance matrix. A more subtle benefit of the QR factor compared to the Cholesky factor is that it avoids the squaring operation for forming the covariance matrix, and subsequently it is better conditioned. However, in Section 1.6 the Cholesky factorization will prove very useful.

1.2 Minimizing the risk

We now turn to the most traditional Markowitz portfolio selection problem, where we minimize the risk (in this case by minimizing the square root of the variance) given a fixed expected return,

$$\begin{array}{lrcl} \text{minimize} & \|R(w^0 + x)\| & & \\ \text{subject to} & \bar{r}^T(w^0 + x) & = & t, \\ & e^T x & = & 0. \end{array} \tag{4}$$

The additional constraint $e^T x = 0$ is a *budget* or *self-financing* constraint stating that the total amount invested in the portfolio remains unchanged after the transaction.

To rewrite (4) as a standard conic quadratic problem, we first use an intuitive reformulation; the problem

$$\text{minimize} \quad g(x)$$

can formulated equivalently in *epigraph form* using one additional variable f as,

$$\begin{array}{lrcl} \text{minimize} & f & & \\ \text{subject to} & g(x) & \leq & f. \end{array}$$

We can therefore rewrite (4) as

$$\begin{array}{lccc} \text{minimize} & f & & \\ \text{subject to} & \bar{r}^T(w^0+x) & = & t, \\ & \|R(w^0+x)\| & \leq & f, \\ & e^T x & = & 0. \end{array}$$

The problem can finally be formulated as a standard conic quadratic optimization problem suited for MOSEK by including an additional equality constraint as

$$\begin{array}{lccc} \text{minimize} & f & & \\ \text{subject to} & \bar{r}^T(w^0+x) & = & t, \\ & R(w^0+x) & = & g, \\ & f & \geq & \|g\|, \\ & e^T x & = & 0, \end{array} \tag{5}$$

in the variables (x, f, g).

Code listing 1 in the appendix for MATLAB, R and Python shows an example program for minimum risk selection for the S&P500 data set described in Appendix C. This data set has 500 stocks with 800 observations (i.e., $X \in \mathbb{R}^{800\times 500}$). We initially have 1 dollar distributed over the portfolio, i.e., $w^0 = (1/n)e$, and we demand an expected daily return of $t = 1.005$. If we run the program, we get an optimal portfolio illustrated in Figure 1a. From the example we observe an investment strategy called *short-selling*. With short-selling, the investor borrows the asset and sells it, and at the end of the period the asset is bought back and returned. Mathematically, short-selling occurs whenever

$$w_j^0 + x_j \leq 0,$$

and if the asset has lost value during the transaction period, the investor makes a profit buying it back.

The potential loss associated with short-selling is infinite, and therefore we often wish to either eliminate or limit short-selling. Short-selling can be completely eliminated from the portfolio selection by adding the linear constraint

$$w^0 + x \geq 0,$$

or we can allow a limited short-selling of s_j for asset j,

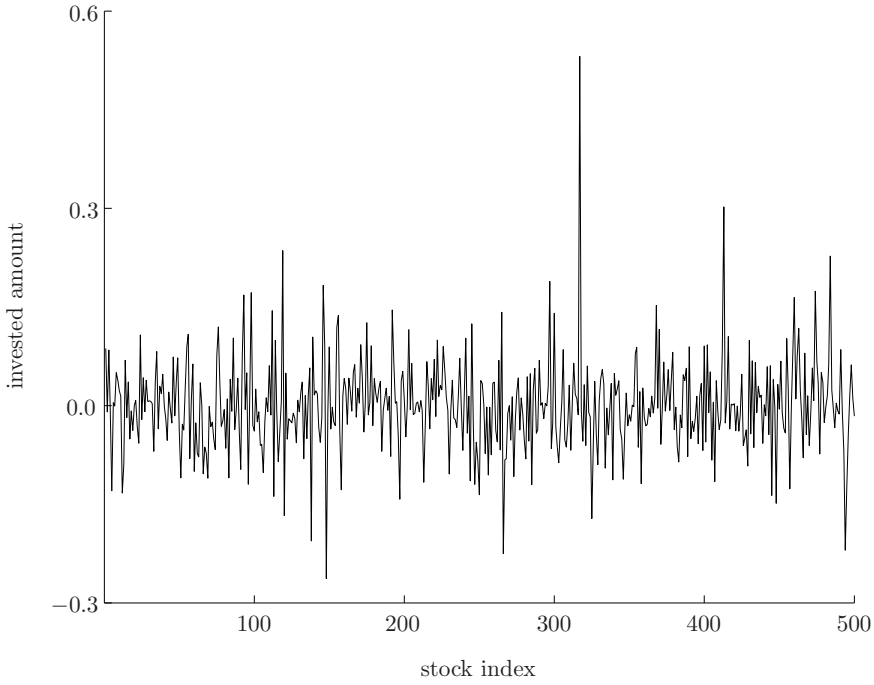
$$w_j^0 + x_j \geq -s_j.$$

If we modify the program in Code listing 1 to exclude short-selling (i.e. by setting variable SHORTSELLING to 0), we get a very different investment strategy illustrated in Figure 1b.

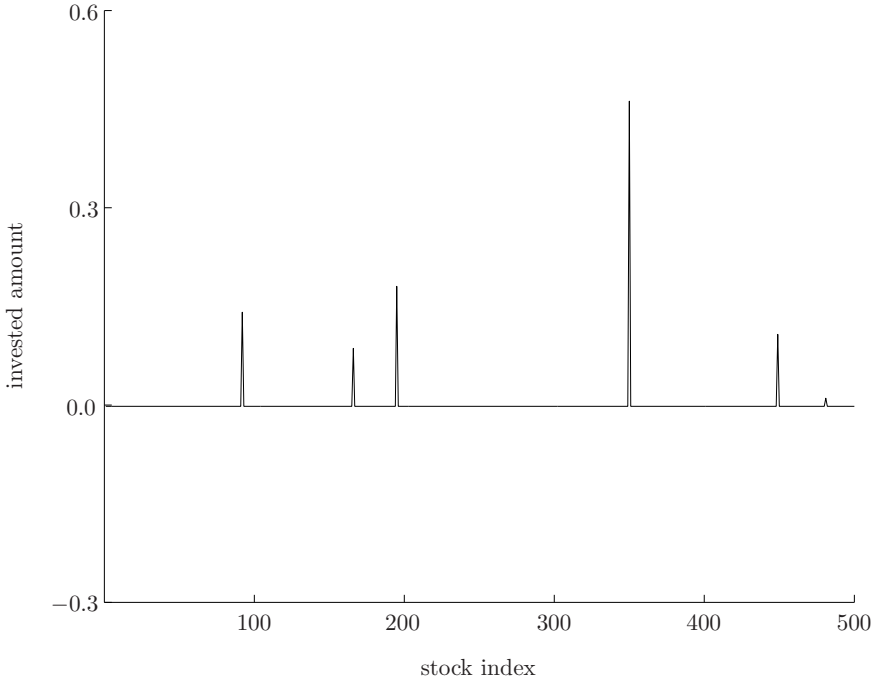
Since we are restricting our feasible domain by adding the constraints $w_j^0 + x_j \geq 0$, we invariably increase the objective value, i.e., the risk. With this investment strategy, we must invest heavily in a few assets to achieve our expected return. This naturally leads to the question: how high an expected return can we get? With short-selling, we can achieve an arbitrarily high expected return. Without short-selling, the largest expected return occurs when we invest exclusively in a single asset - in this example in asset #481. If we change the demanded return to $t = \max(\bar{r}) + 1$, and rerun the example, we get the following message

```
Interior-point solution
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
```

which tells us that the constraints are conflicting (i.e., the feasible set is empty); such a large expected return is incompatible with the self-budget constraint.



(a) With short-selling. The associated risk is $f = 0.0068$.



(b) Without short-selling. The associated risk is $f = 0.0836$.

Figure 1: Optimal minimum-risk solutions.

Another pertinent issue is which factorization of the data matrix do we use. In the example we considered a time-horizon of 800 observations, i.e., $X \in \mathbb{R}^{800\times 500}$ and $\Sigma = X^T X \in \mathbb{R}^{500\times 500}$ has full rank, so the most efficient formulation (in terms of solution time) uses an economy-size QR factorization of the normalized data-matrix, $\bar{X} = QR$ with a triangular 500×500 factor R. If instead we directly use the normalized data matrix $\bar{X}$ as our factor, the solution time for this particular example is around 6 times slower[1].

The situation is different when we have a short time-horizon with fewer observations than number of stocks. In that case, there is no efficiency loss in directly using the data matrix instead of the economy-size QR factorization (since then the dimensions of X and R are identical), and using the data-matrix is simpler and saves a single QR factorization. If we change the time-horizon in the example to 100 observations, then the solution time using either the QR factorization or the data matrix is around 7 times faster than with 800 observations. We summarize the execution times for the different setups in Table 1.

# observations	factor	solution time (secs)
800	QR	1.09
800	norm. data matrix	6.47
100	QR	0.16
100	norm. data matrix	0.16

Table 1: Solution times for different portfolio setups.

As a final experiment, we keep the number of observations fixed at $N = 800$, but we gradually change the number of stocks in the portfolio from 50 to 500 and record the solution. Figure 2 shows the solution time for different problem sizes.

1.3 Maximizing expected return

We can also maximize the expected return given constraints on the largest admissible risk,

$$\|R(w^0 + x)\| \leq \hat{f},$$

where $\hat{f}$ is the acceptable level of risk. This leads to the problem

$$\begin{array}{lrcl} \text{maximize} & \bar{r}^T(w^0 + x) & & \\ \text{subject to} & e^T x & = & 0, \\ & \|R(w^0 + x)\| & \leq & \hat{f}, \end{array}$$

which we can rewrite as a standard conic quadratic problem

$$\begin{array}{lrcl} \text{maximize} & \bar{r}^T(w^0 + x) & & \\ \text{subject to} & e^T x & = & 0, \\ & R(w^0 + x) & = & g, \\ & f & \geq & \|g\|, \\ & f & = & \hat{f}, \end{array} \tag{6}$$

in the variables (x, f, g).

[1] All examples are executed in MATLAB on a quad core Intel Xeon 2.5GHz processor with 16GB memory.

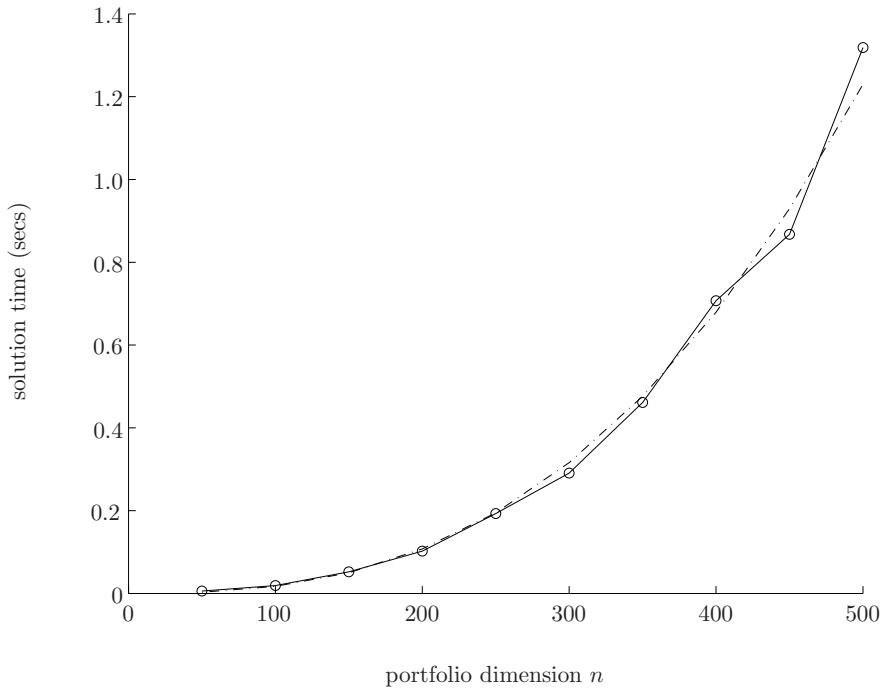


Figure 2: Solution time for different problem sizes. The dashed line shows the fitted curve $f(x) = \gamma x^{2.66}$, where γ depends on the computer (we observed $\gamma \approx 10^{-7}$).

1.4 Computing the efficient frontier

The two problem formulations (5) and (6) produce the same *optimal frontier*, i.e., by varying t and $\hat{f}$ the two formulations give the same optimal trade-off curves between risk and expected return. If our main objective is to compute the efficient frontier, we can use an alternative formulation that directly minimizes a compromise between standard deviation and return using a positive *penalization factor* λ,

$$\begin{array}{ll} \text{maximize} & \bar{r}^T(w^0 + x) - \lambda \|R(w^0 + x)\| \\ \text{subject to} & e^T x = 0, \end{array}$$

which has the following conic quadratic formulation

$$\begin{array}{lrcl} \text{maximize} & \bar{r}^T(w^0 + x) - \lambda f & & \\ \text{subject to} & e^T x & = & 0, \\ & R(w^0 + x) & = & g, \\ & f & \geq & \|g\|, \end{array} \tag{7}$$

in the variables (x, f, g). Alternatively, one may wish to use the classic Markowitz formulation

$$\begin{array}{ll} \text{maximize} & \bar{r}^T(w^0 + x) - (\lambda/2)\|R(w^0 + x)\|^2 \\ \text{subject to} & e^T x = 0. \end{array}$$

Note, however, that the objective function of this formulation uses risk and return with different units, e.g. dollars and dollars^2, in contrary to (7) having same units. The classic Markowitz

formulation can be posed as a *rotated conic quadratic* problem (see Appendix A),

$$\begin{array}{lccc} \text{maximize} & \bar{r}^T(w^0+x)-f & & \\ \text{subject to} & e^T x & = & 0, \\ & R(w^0+x) & = & g, \\ & 2fh & \geq & \|g\|^2, \\ & h & = & 1/\lambda, \end{array} \tag{8}$$

in the variables (x, f, g, h). By varying $\lambda \in [0, \infty)$ in either (7) or (8) we trace the (same) efficient frontier.

1.5 Maximizing the Sharpe ratio

The Sharpe ratio defines an efficiency metric of a portfolio as the expected return per unit risk, i.e.,

$$S(x) = \frac{r^T(w^0+x) - r_f e^T w^0}{\|R(w^0+x)\|},$$

where r_f denotes the return of a risk-free asset, and x is assumed not to include a risk-free asset. Also, by assumption, we have that $\bar{r}^T(w^0+x) > r_f e^T w^0$ (i.e., there exists a risk-associated portfolio with a greater yield than the risk-free asset), so maximizing the Sharpe ratio is equivalent to minimizing $1/S(x)$. In other words, we have the following problem

$$\begin{array}{lcc} \text{minimize} & \frac{\|R(w^0+x)\|}{\bar{r}^T(w^0+x)-r_f e^T w^0} & \\ \text{subject to} & e^T x & = \quad 0. \end{array}$$

We next introduce a variable transformation,

$$y = \gamma(w^0+x), \qquad \gamma \geq 0.$$

Since $\gamma > 0$ can be chosen arbitrarily and $(\bar{r} - r_f e)^T(w^0+x) > 0$, we have without loss of generality that

$$(\bar{r} - r_f e)^T y = 1.$$

Thus, we obtain the following problem for maximizing the Sharpe ratio,

$$\begin{array}{lccc} \text{minimize} & \|Ry\| & & \\ \text{subject to} & e^T y & = & \gamma \cdot e^T w^0, \\ & (\bar{r} - r_f e)^T y & = & 1, \\ & \gamma & \geq & 0, \end{array}$$

or equivalently as a standard conic quadratic problem,

$$\begin{array}{lccc} \text{minimize} & f & & \\ \text{subject to} & e^T y & = & \gamma \cdot e^T w^0, \\ & Ry & = & g, \\ & f & \geq & \|g\|, \\ & (\bar{r} - r_f e)^T y & = & 1, \\ & \gamma & \geq & 0, \end{array}$$

and we recover $x := y/\gamma - w^0$. Eliminating short-selling corresponds to adding the constraints $y \geq 0$ (in fact $y/\gamma \geq 0$). Code listing 2 in the appendix for MATLAB, R and Python shows an implementation for computing the largest Sharpe ratio without short-selling. The referred functions 'minrisk' and 'sharpe' are defined in separate code listings under the appendix of each programming language.

We next illustrate the largest Sharpe ratio with a numerical example. For the data set used in Section 1.2 (i.e., a portfolio of 500 assets and 800 observations) we compute the efficient frontier (using the algorithm in Section 1.2 for different values of t), and we also compute the largest Sharpe ratio, assuming a risk-free asset with a daily return of 1.0005.

The results are shown in Figure 3. As evident from the figure, the maximum Sharpe ratio is the tangent to the optimal frontier, which passes through $(0, r_f e^T w^0)$.

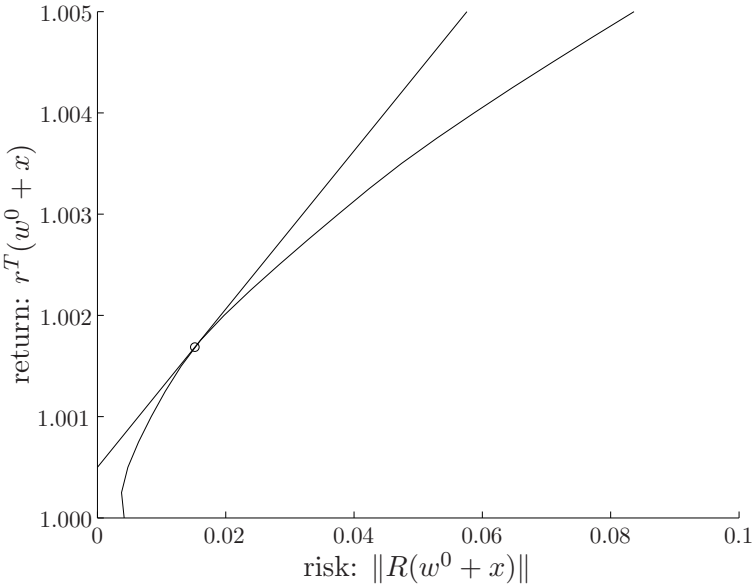


Figure 3: Efficient frontier of a portfolio. The solution that achieves the maximum Sharpe ratio is denoted by '$\circ$'.

1.6 Structured covariance matrices

Sometimes the covariance matrix is assumed to have a specific structure, e.g., in a *factor model* it is often assumed to consist of a diagonal plus a low-rank term,

$$\Sigma_r = D + ABA^T \tag{9}$$

where $D \in \mathbb{R}^{n\times n}$ is a positive definite diagonal matrix, $B \in \mathbb{R}^{l\times l}$ ($l << n$) is a positive definite matrix, and $A \in \mathbb{R}^{n\times l}$ is a matrix that describes the impact of different factors on the covariance. We can think of (9) as a simple *parametric model*, with parameters D, A and B, and since the number of parameters in (9) is smaller than the number of parameters in $\bar{\Sigma}_r$, we expect the parameters in the factor model to be estimated more accurately. Popular factor models are developed by commercial risk analysis companies, and the parameters are estimated using expert knowledge of the respective stock market.

We could use a Cholesky factorization from Section 1.1 to factor Σ_r directly, which would result in a triangular dense factor. Alternatively, we can rewrite the factor model as

$$\Sigma_r = \begin{bmatrix} I & A \end{bmatrix} \begin{bmatrix} D & \\ & B \end{bmatrix} \begin{bmatrix} I & A \end{bmatrix}^T,$$

and factor the smaller matrix $B = R^T R$ using a Cholesky factorization. If we define

$$G \triangleq \begin{bmatrix} D^{1/2} & \\ & R \end{bmatrix} \begin{bmatrix} I & A \end{bmatrix}^T = \begin{bmatrix} D^{1/2} \\ RA^T \end{bmatrix},$$

where $D^{1/2}$ denotes a positive diagonal matrix with $D_{ii}^{1/2} = \sqrt{D_{ii}}$, we get an alternative factorization $\Sigma_r = G^T G$. Thus, exploiting the structure of the factor model leads to a problem

$$\begin{array}{lccc} \text{minimize} & f & & \\ \text{subject to} & \bar{r}^T(w^0 + x) & = & t, \\ & G(w^0 + x) & = & \tilde{g}, \\ & f & \geq & \|\tilde{g}\|, \\ & e^T x & = & 0, \end{array}$$

in the variables $(x, f, \tilde{g})$. The dimensions of this formulation are larger than for the original problem (5), but the constraints are much sparser (and therefore require less storage space), which usually is preferable.

1.7 Choosing the right formulation

In this section we summarize important points made in previous sections about speeding up the solution time.

- In general, one should use the formulation with the smallest memory requirements.
- Exploiting the low-rank structure of a factor model gives a significant speedup in solution time, even though we get additional variables. In the quadratic programming formulation, we can think of the improvement as replacing a completely dense Hessian matrix of the objective with a larger diagonal Hessian plus a number of relatively sparse constraints.
- If we need to estimate the covariance matrix using fewer observations than assets (i.e., we have a relatively short time-horizon), then we should directly use the data-matrix as our factor, or alternative we can use an economy size QR factorization.
- If we need to estimate the covariance matrix, and we have more observations than assets (i.e., we have a relatively long time-horizon), then we should use the economy size QR factorization of the data matrix, or alternative we can use a Cholesky factorization of the estimated covariance matrix, but we should not directly use the data matrix as our factor.

2 Transaction costs with market impact

2.1 Modeling market impact

The previous problem formulations in Section 1 did not include transaction costs, i.e., they assumed no cost associated with a transaction, and that trading does not affect the market price. A more realistic model of the transaction cost (of a single asset) is given in [1], and has the form

$$\text{commission} + \frac{\text{bid}}{\text{ask}} - \text{spread} + \theta\sqrt{\frac{\text{trade volume}}{\text{daily volume}}}.$$

The last term is the *market impact cost*, and captures that the price of an asset increases or decreases if you buy or sell large quantities, respectively. It can be approximated as

$$\theta_j \sqrt{\frac{(\text{trade volume})_j}{(\text{daily volume})_j}} \approx m_j \sqrt{|x_j|}$$

for the jth asset, where θ_j and m_j are parameters that must be estimated. We can include the market impact term into the minimum-risk portfolio formulation as follows

$$\begin{array}{lccll} \text{minimize} & f & & & \\ \text{subject to} & \bar{r}^T(w^0+x) & = & t, & \\ & \|R(w^0+x)\| & \leq & f, & \\ & e^Tx+e^Ty & = & 0, & \\ & |x_j|(m_j|x_j|^{1/2}) & \leq & y_j, & j=1,\ldots,n, \end{array}$$

where the market impact term influences the modified self-budget constraint $e^Tx+e^Ty=0$ to make large transaction more costly. If we introduce the variable change $y_j = m_j\bar{y}_j$, we get the formulation

$$\begin{array}{lccll} \text{minimize} & f & & & \\ \text{subject to} & \bar{r}^T(w^0+x) & = & t, & \\ & \|R(w^0+x)\| & \leq & f, & \\ & e^Tx+m^T\bar{y} & = & 0, & \\ & |x_j|^{3/2} & \leq & \bar{y}_j, & j=1,\ldots,n. \end{array} \tag{10}$$

The formulation (10) is a convex problem, but more interestingly, it can also be written as a standard conic quadratic problem; the derivation is technically involved and deferred to Appendix B. Using the result from Appendix B, we get the following standard conic quadratic formulation of the minimum risk selection with market impact correction,

$$\begin{array}{lccll} \text{minimize} & f & & & \\ \text{subject to} & \bar{r}^T(w^0+x) & = & t, & \\ & e^Tx+m^T\bar{y} & = & 0, & \\ & R(w^0+x) & = & g, & \\ & f & \geq & \|g\|, & \\ & z_j^2 & \leq & 2s_j\bar{y}_j, & j=1,\ldots,n, \\ & w_j^2 & \leq & 2v_jr_j, & j=1,\ldots,n, \\ & z & = & v, & \\ & s & = & w, & \\ & r & = & \frac{1}{8}e, & \\ & x & \leq & z, & \\ & -x & \leq & z, & \\ & s,\bar{y},v,r & \geq & 0, & \end{array} \tag{11}$$

in the variables $(x, f, g, s, \bar{y}, v, r, z)$. The conic formulation contains several additional variables and constraints, but the constraints are all extremely sparse, so they only have a very limited impact on memory requirements and execution time for the MOSEK solver.

2.2 Modeling transaction costs

The transaction cost can be modeled in different ways, e.g., we can assume a linear cost

$$T(x_j) = \begin{cases} c^+x_j, & x_j \geq 0, \\ -c^-x_j, & x_j < 0, \end{cases}$$

or more realistically, we can assume a constant transaction cost plus a linear cost,

$$T(x_j) = \begin{cases} b + c^+x_j, & x_j > 0, \\ 0, & x_j = 0, \\ b - c^-x_j, & x_j < 0. \end{cases}$$

The latter model can be included in the portfolio selection model as

$$
\begin{array}{lrcll}
\text{minimize} & f & & & \\
\text{subject to} & \bar{r}^T(w^0+x) & = & t, & \\
& R(w^0+x) & = & g, & \\
& f & \geq & \|g\|, & \\
& e^T x + e^T y & = & 0, & \\
& bz_j + c^+ x_j & \leq & y_j, & j = 1,\ldots,n, \\
& bz_j - c^- x_j & \leq & y_j, & j = 1,\ldots,n, \\
& x_j & \geq & l_j z_j, & j = 1,\ldots,n, \\
& x_j & \leq & u_j z_j, & j = 1,\ldots,n, \\
& z_j \in \{0,1\} & & & j = 1,\ldots,n,
\end{array}
\tag{12}
$$

in the variables (x, y, f, g, z). In the formulation (12), the variable y_j is the transaction cost for x_j and the binary variable z_j controls the constant transaction cost. The constants l_j and u_j give lower and upper bounds on the allowed range of x_j, respectively. This problem is non-convex because of the integer constraints $z_j \in \{0, 1\}$, and is therefore much more difficult to solve than the problems studied in the previous sections. Tight bounds l_j and u_j generally help to reduce computation-time, however.

Appendix A: Conic quadratic optimization

In this section, we review conic quadratic optimization. Conic quadratic optimization is an established field of convex optimization with close similarities to traditional linear programming. A conic quadratic optimization problem can be formulated as

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{C}_q \end{array} \tag{13}$$

in the variable $x \in \mathbb{R}^n$ with a problem specified by $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. The cone $\mathcal{C}_q$ is the so-called *quadratic cone* or *Lorentz cone*, and $x \in \mathcal{C}_q$ means that

$$x_1 \geq \sqrt{x_2^2 + \cdots + x_n^2}.$$

Note, that this formulation is very similar to a standard linear program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{C}_l, \end{array} \tag{14}$$

where the *linear cone* $\mathcal{C}_l$ is simply the non-negative orthant, i.e., $\mathcal{C}_l = \{x \,|\, x \geq 0\}$.

MOSEK is an industry leader in software for conic optimization, and uses a more general formulation for the conic quadratic problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c \\ & l^x & \leq & x & \leq & u^x \\ & & & x^t \in \mathcal{C}^t & & \end{array} \tag{15}$$

where the variables x^t form a partitioning of $\mathbb{R}^n$, i.e., each element of x belongs to *exactly one* subset x^t. For example, it could be that

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix}, \quad x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Each cone C^t is defined as either,

- the set of *free variables*

$$\mathcal{C}_f^t = \left\{ x \in \mathbb{R}^{n^t} \right\},$$

- the *quadratic cone*

$$\mathcal{C}_q^t = \left\{ x \in \mathbb{R}^{n^t} \;:\; x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\},$$

- the *rotated quadratic cone*

$$\mathcal{C}_r^t = \left\{ x \in \mathbb{R}^{n^t} \;:\; 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2,\; x_1, x_2 \geq 0 \right\}.$$

Note, that the formulation (15) covers the standard conic quadratic formulation (13) by choosing $l^c = u^c = b$, $l^x = -\infty$, $u^x = \infty$ and a single quadratic cone $x \in C_q$. Similarly, it also covers the standard linear programming formulation (14) by choosing $l^c = u^c = b$, $l^x = 0$, $u^x = \infty$ and a single cone $x \in C_f$. More generally, the conic quadratic formulation (15) covers a wide range of different problem formulations and applications, see [2, 4] for a general treatment.

Appendix B: Conic quadratic modeling of $|x|^{3/2}$

In this appendix we show how to model

$$|x|^{3/2} \leq t$$

as a conic quadratic constraint. We initially represent it as

$$-z \leq x \leq z, \qquad \frac{z^2}{\sqrt{z}} \leq t, \qquad z \geq 0$$

using an additional variable z. The only non-linear constraint is $z^2/\sqrt{z} \leq t$, which can be represented equivalently as

$$\begin{array}{rcl} z^2 & \leq & 2st \\ w^2 & \leq & 2vr \\ z & = & v \\ s & = & w \\ r & = & \frac{1}{8} \\ s,t,v,r & \geq & 0, \end{array} \tag{16}$$

in the variables (z, w, s, t). The set (16) is the intersection of a set of linear (in)equalities and rotated quadratic cones (see Appendix A), and thus fits the framework of conic quadratic optimization.

To see that the formulations are equivalent, we observe from (16) that

$$s^2 = w^2 \leq 2vr = \frac{v}{4} = \frac{z}{4}$$

and

$$z^2 \leq 2st \leq 2\sqrt{\frac{z}{4}}t,$$

which shows the desired result

$$z^2/\sqrt{z} \leq t.$$

Thus, a conic quadratic formulation of the inequality $|x|^{3/2} \leq t$ is

$$\begin{array}{rcl} x & \leq & z \\ -x & \leq & z \\ z^2 & \leq & 2st \\ w^2 & \leq & 2vr \\ z & = & v \\ s & = & w \\ r & = & \frac{1}{8} \\ s,t,v,r & \geq & 0. \end{array} \tag{17}$$

More generally, for positive integers p and q the following convex power functions

$$|x|^{p/q} \leq t, \qquad p/q \geq 1,$$

and

$$|x|^{-p/q} \leq t,$$

can be represented using conic quadratic formulations, see [2, 5] for details.

Appendix C: Parameter estimation from historical data

In practice, the parameters of the stochastic portfolio model are not known and must be estimated from historical data. In this tutorial we consider daily prices for the 500 stocks in the S&P500 index [6].

For these 500 stocks, we retrieved historical prices from the Google Finance service [3]. The dataset contains daily observations of the opening, closing, highest, and lowest trading prices, respectively, as well as the traded volume. The number of observations for each stock varies, since stocks might enter or leave the index at different times, and furthermore, there are a missing observations in the dataset.

From the data, we need to estimate the vector of average returns $\bar{r}$ as well as the covariance matrix for the returns. For each stock asset j, we first estimate the kth daily absolute return as

$$X_{j,k} = \frac{(\text{closing price})_{j,k+1}}{(\text{closing price})_{j,k}}.$$

This gives us an $N \times 500$ data matrix with daily absolute returns, where N is the width of our observation window (or time-horizon). Estimating the average returns from the data matrix is by no means trivial. For example, consider two consecutive returns of 2 and $1/2$. A reasonable estimate of the average return is the geometric mean of 1, but the arithmetic mean of $5/4$ is less appropriate. Furthermore, for realistic data spanning a longer time-horizon, we also need to consider other issues such as *discounting*, etc., and possibly use different weights for the observations.

Another issue is how to handle missing data in the observations. We choose to simply interpolate (and extrapolate) missing data, by taking the exact value of the nearest non-missing observation for each stock asset.

To estimate the average return, we merely use the arithmetic mean of each column in X as our estimate, i.e.,

$$\bar{r} = (1/N)X^T e.$$

Similarly, we estimate the covariance of returns as the traditional unbiased estimator

$$\bar{\Sigma}_r = \frac{1}{N-1}(X - e\bar{r}^T)^T(X - e\bar{r}^T).$$

We mention four factorizations G of this covariance matrix (see Section 1.1).

As an illustration of the data matrix, Figure 4 shows the closing price for AAPL for the past 800 observations (or roughly 3.5 years). From the data we estimate an average daily return of 1.0015.

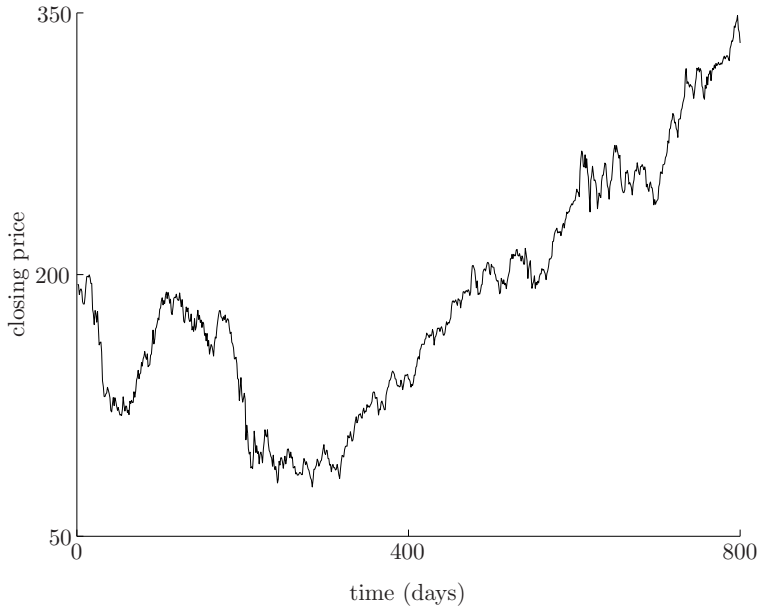


Figure 4: Closing prices for AAPL from 2007-12-07 to 2011-01-20.

Appendix D: MATLAB programs

MATLAB

```
% Minimum risk Markowitz portfolio selection example
%
%   minimize     f
%   subject to   r'*(w0 + x) = t
%                G*(w0 + x) = g,   ||g||_2 < f
%                sum(x) = 0.
%                and optionally w0 + x >= 0
% or equivalently
%
%   minimize      f
%   subject to   [ r'   0    0 ] [ x ]    [ t-r'*w0 ]
%                [ G    0   -I ]*[ f ] = [  -G*w0  ]
%                [ e'   0    0 ] [ g ]    [     0    ]
%                ||g||_2 <= f
%                and optionally w0 + x >= 0

SHORTSELLING = 1;

% Read expected returns and covariance factor
rT = csvread('../data/sp500-ret.csv');
G  = csvread('../data/sp500-Gqr.csv');

[p,n] = size(G);
w0 = ones(n,1)/n;  % initial holdings
t  = 1.005;        % expected return

prob.a = sparse([ rT, zeros(1, 1+p);
                  G, zeros(p,1), -eye(p);
                  ones(1,n), zeros(1,1+p)]);
prob.c = [ zeros(n,1); 1; zeros(p,1)];
prob.cones{1}.type = 'MSK_CT_QUAD';
prob.cones{1}.sub = [n+1:n+1+p];
b = [ t - rT*w0; -G*w0; 0 ];
prob.blc = b;
prob.buc = b;

if SHORTSELLING == 0
  prob.blx = [-w0; -inf*ones(1+p,1)];
end

[err,res]=mosekopt('minimize',prob);
x = res.sol.itr.xx(1:n);
f = res.sol.itr.xx(n+1);
```

Code listing 1: Minimum risk portfolio selection

MATLAB

```
clear; close all;

% Read expected returns and covariance factor
rT = csvread('../data/sp500-ret.csv');
G  = csvread('../data/sp500-Gqr.csv');

n = size(G,2);
w0 = ones(n,1)/n;

tlow  = 1;
thigh = 1.005;      % or max(rT)
RET = linspace(tlow,thigh,21);

X = [];  RISK = zeros(1,length(RET));
for i = 1:length(RET)
    [x, risk] = minrisk(G, rT, RET(i), w0);

    X(:,i)  = x;
    RISK(i) = risk;
end

rf = 1.0005;
[x2, risk2, ret2] = sharpe(G, rT, w0, rf);
sharperatio = (ret2 - rf*sum(w0))/risk2;

plot(RISK, RET, '-')
hold on
plot(risk2,ret2,'o')
line([0, 100],[rf*sum(w0), rf*sum(w0) + 100*sharperatio])
hold off
axis([0, 0.1, tlow, thigh])
```

Code listing 2: Illustrating largest Sharpe ratio without short-selling

MATLAB

```
function [x, risk] = minrisk(G, rT, t, w0)
% Minimum risk Markowitz portfolio selection (assuming no short-selling)
%
%   minimize     f
%   subject to   r'*(w0 + x) = t,
%                G*(w0 + x) = g,   ||g||_2 < f
%                sum(x) = 0,
%                and w0 + x >= 0.
% or equivalently
%
%   minimize      f
%   subject to   [ r'   0    0 ] [ x ]    [ t-r'*w0 ]
%                [ G    0   -I ]*[ f ] = [  -G*w0  ]
%                [ e'   0    0 ] [ g ]    [     0    ]
%                ||g||_2 <= f ,  and w0 + x >= 0.

n = length(rT);
p = size(G,1);

prob.a = sparse([ rT, zeros(1, 1+p);
                  G, zeros(p,1), -eye(p);
                  ones(1,n), zeros(1,1+p)]);
prob.c = [ zeros(n,1); 1; zeros(p,1)];
prob.cones{1}.type = 'MSK_CT_QUAD';
prob.cones{1}.sub = [n+1:n+1+p];
b = [ t - rT*w0; -G*w0; 0 ];
prob.blc = b;
prob.buc = b;
prob.blx = [-w0; -inf*ones(1+p,1)];

[err,res]=mosekopt('minimize',prob);

x    = res.sol.itr.xx(1:n);
risk = res.sol.itr.xx(n+1);
```

Function **minrisk**: Minimum risk portfolio selection without short-selling

MATLAB

```
function [x, risk, ret] = sharpe(G, rT, w0, rf)
% Computes the largest Sharpe ratio (assuming no short-selling),
%
%   minimize     f
%   subject to   (r-e*rf)'*y = 1
%                G*y  = g,   ||g||_2 <= f
%                sum(y) = t*sum(w0)
%                t >= 0
%                y >= 0
%
% or equivalently
%
%   minimize       f
%   subject to   [ (r-e*rf)'  0   0        0        ] [ y ]   [ 1 ]
%                [    G       0  -I        0        ]*[ f ] = [ 0 ]
%                [    e'      0   0   -sum(w0)     ] [ g ]   [ 0 ]
%                                                     [ t ]
%                  t      >= 0
%                  y      >= 0
%                ||g||_2 <= f

n = length(rT);
p = size(G,1);

prob.a = sparse([ rT-rf, zeros(1, 1+p+1);
                  G, zeros(p,1), -eye(p), zeros(p,1);
                  ones(1,n), zeros(1,1+p), -sum(w0)]);
prob.c = [zeros(n,1); 1; zeros(p+1,1)];
prob.cones{1}.type = 'MSK_CT_QUAD';
prob.cones{1}.sub = [n+1:n+1+p];
b = [1; zeros(p,1); 0];
prob.blc = b;
prob.buc = b;
prob.blx = [zeros(n,1); -inf*ones(1+p,1); 0];

[err,res]=mosekopt('minimize',prob);
y    = res.sol.itr.xx(1:n);
f    = res.sol.itr.xx(n+1);
g    = res.sol.itr.xx(n+1+[1:p]);
t    = res.sol.itr.xx(n+1+p+1);
x    = y/t - w0;
risk = f/t;
ret  = rT*(x+w0);
```

Function **sharpe**: Largest Sharpe ratio without short-selling

Appendix E: R programs

R

```
# Minimum risk Markowitz portfolio selection example
#
#   minimize     f
#   subject to   r'*(w0 + x) = t
#                G*(w0 + x) = g,   ||g||_2 < f
#                sum(x) = 0.
#                and optionally w0 + x >= 0
# or equivalently
#
#   minimize      f
#   subject to   [ r'   0    0 ] [ x ]    [ t-r'*w0 ]
#                [ G    0   -I ]*[ f ] = [  -G*w0  ]
#                [ e'   0    0 ] [ g ]    [     0    ]
#                ||g||_2 <= f
#                and optionally w0 + x >= 0

SHORTSELLING <- 1;

# Read expected returns and covariance factor
rT <- as.matrix(read.csv("../data/sp500-ret.csv", header=F))
G  <- as.matrix(read.csv("../data/sp500-Gqr.csv", header=F))

p <- nrow(G); n <- ncol(G);
w0 <- matrix(1/n,nrow=n); # initial holdings
t  <- 1.005;              # expected return

prob <- list(sense="minimize")
prob$A <- rBind(
  cBind(rT, Matrix(0, ncol=1+p)),
  cBind(G, Matrix(0, nrow=p), -diag(p)),
  cBind(Matrix(1,ncol=n), Matrix(0,ncol=1+p))
)
prob$c <- c(rep(0,n), 1, rep(0,p));
prob$cones <- cBind( list('MSK_CT_QUAD', (n+1):(n+1+p)) )
b <- c(t - rT%*%w0 , -G%*%w0 ,  0 );
prob$bc <- rbind(b, b);
prob$bx <- matrix(c(-Inf,Inf), nrow=2, ncol=n+1+p);

if (SHORTSELLING == 0) {
  prob$bx[1,] <- c(-w0, rep(-Inf,1+p));
}

res<-mosek(prob);
x <- res$sol$itr$xx[1:n];
f <- res$sol$itr$xx[n+1];
```

Code listing 1: Minimum risk portfolio selection

R

```
source("minrisk.R")
source("sharpe.R")

# Read expected returns and covariance factor
rT <- as.matrix(read.csv("../data/sp500-ret.csv", header=F))
G  <- as.matrix(read.csv("../data/sp500-Gqr.csv", header=F))

n  <- ncol(G);
w0 <- matrix(1/n,nrow=n);

tlow  <- 1;
thigh <- 1.005;     # or max(rT)
RET <- seq(tlow, thigh, length.out=21)

X    <- Matrix(0,n,length(RET));
RISK <- rep(0, length(RET));
for (i in 1:length(RET)) {
    solrisk <- minrisk(G, rT, RET[i], w0);

    X[,i]    = solrisk$x;
    RISK[i] = solrisk$risk;
}

rf <- 1.0005;
solsharpe <- sharpe(G, rT, w0, rf);
x2 <- solsharpe$x;  risk2 <- solsharpe$risk;  ret2 <- solsharpe$ret;
sharperatio <- (ret2 - rf*sum(w0))/risk2;

plot(RISK, RET, 'l', xlim=c(0, 0.1), ylim=c(tlow, thigh))
points(risk2,ret2)
lines(c(0, 100), c(rf*sum(w0), rf*sum(w0) + 100*sharperatio))
dev.off()
```

Code listing 2: Illustrating largest Sharpe ratio without short-selling

R

```r
minrisk <- function(G, rT, t, w0) {
  # Minimum risk Markowitz portfolio selection (assuming no short-selling)
  #
  #   minimize     f
  #   subject to   r'*(w0 + x) = t,
  #               G*(w0 + x) = g,   ||g||_2 < f
  #               sum(x) = 0,
  #               and w0 + x >= 0.
  # or equivalently
  #
  #   minimize      f
  #   subject to   [ r'   0    0 ] [ x ]    [ t-r'*w0 ]
  #               [ G    0   -I ]*[ f ] = [   -G*w0   ]
  #               [ e'   0    0 ] [ g ]    [     0     ]
  #               ||g||_2 <= f ,   and w0 + x >= 0.

  n <- length(rT);
  p  <- nrow(G);

  prob <- list(sense="minimize")
  prob$A <- rBind(
    cBind(rT, Matrix(0, ncol=1+p)),
    cBind(G, Matrix(0, nrow=p), (-1)*Diagonal(p)),
    cBind(Matrix(1,ncol=n), Matrix(0,ncol=1+p))
  )
  prob$c <- c(rep(0,n), 1, rep(0,p));
  prob$cones <- cBind(
    list('MSK_CT_QUAD', (n+1):(n+1+p))
  )
  b <- c(t - rT%*%w0 , -G%*%w0 ,  0 );
  prob$bc <- rbind(b, b);
  prob$bx <- matrix(c(-Inf,Inf), nrow=2, ncol=n+1+p);
  prob$bx[1,] <- c(-w0, rep(-Inf,1+p));

  res  <- mosek(prob);
  x    <- res$sol$itr$xx[1:n];
  risk <- res$sol$itr$xx[n+1];
  return( list(x=x, risk=risk) )
}
```

Function **minrisk**: Minimum risk portfolio selection without short-selling

R

```
sharpe = function(G, rT, w0, rf) {
  # Computes the largest Sharpe ratio (assuming no short-selling),
  #
  #   minimize     f
  #   subject to   (r-e*rf)'*y = 1
  #                G*y  = g,   ||g||_2 <= f
  #                sum(y) = t*sum(w0)
  #                t >= 0
  #                y >= 0
  #
  # or equivalently
  #
  #   minimize       f
  #   subject to   [ (r-e*rf)'  0   0        0        ] [ y ]   [ 1 ]
  #                [    G       0  -I        0        ]*[ f ] = [ 0 ]
  #                [    e'      0   0   -sum(w0)     ] [ g ]   [ 0 ]
  #                                                     [ t ]
  #                  t      >= 0
  #                  y      >= 0
  #                ||g||_2 <= f

  n <- length(rT);
  p <- nrow(G);

  prob <- list(sense="minimize");
  prob$A <- rBind(
    cBind(rT-rf, Matrix(0,ncol=1+p+1)),
    cBind(G, Matrix(0,nrow=p), (-1)*Diagonal(p), Matrix(0,nrow=p)),
    cBind(Matrix(1,ncol=n), Matrix(0,ncol=1+p), -sum(w0))
  )
  prob$c <- c(rep(0,n), 1, rep(0,p+1));
  prob$cones <- cBind(
    list('MSK_CT_QUAD', (n+1):(n+p+1))
  );

  b <- c(1, rep(0,p+1));
  prob$bc <- rBind(b,b);
  prob$bx <- rBind(c(rep(0,n),rep(-Inf,1+p),0), rep(Inf,n+1+p+1));

  res  <- mosek(prob);
  y    <- res$sol$itr$xx[1:n];
  f    <- res$sol$itr$xx[n+1];
  g    <- res$sol$itr$xx[n+1+(1:p)];
  t    <- res$sol$itr$xx[n+1+p+1];
  x    <- y/t - w0;
  risk <- f/t;
  ret  <- ( rT%*%(x+w0) )[1];
  return( list(x=x, risk=risk, ret=ret) );
}
```

Function **sharpe**: Largest Sharpe ratio without short-selling

Appendix F: Python programs

Python

```
# Minimum risk Markowitz portfolio selection example
#   minimize      f
#   subject to  [ r'  0   0 ] [ x ]   [ t-r'*w0 ]
#               [ G   0  -I ]*[ f ] = [  -G*w0  ]
#               [ e'  0   0 ] [ g ]   [    0    ]
#               ||g||_2 <= f ,  and optionally w0 + x >= 0
import sys, csv, mosek
def streamprinter(x): sys.stdout.write(x); sys.stdout.flush()
env = mosek.Env(); env.set_Stream(mosek.streamtype.log, streamprinter)
task = env.Task(0,0); task.set_Stream(mosek.streamtype.log, streamprinter)
SHORTSELLING = 1

def main ():
  f_itrT = open('../data/sp500-ret.csv','rb')  # expected returns
  rT     = csv.reader(f_itrT, quoting=csv.QUOTE_NONNUMERIC).next()
  f_itG  = open('../data/sp500-Gqr.csv','rb')  # covariance factor
  itG    = csv.reader(f_itG, quoting=csv.QUOTE_NONNUMERIC)
  G      = [[(i,v) for i,v in enumerate(row) if v!=0.0] for row in itG]
  n = len(rT);  p = 0;  w0 = [1.0/n]*n;  t  = 1.005;  inf = 0.0

  task.putobjsense(mosek.objsense.minimize)
  task.append(mosek.accmode.var, n+1+len(G)) # x, f and g
  task.append(mosek.accmode.con, 1+len(G)+1) # constraints
  task.putcj(n,1.0)

  bc = t - sum([rT[i]*w0[i] for i in range(n)])
  task.putavec(mosek.accmode.con, 0, range(n), rT)
  task.putbound(mosek.accmode.con, 0, mosek.boundkey.fx, bc, bc)

  for row in G:
    Gidx = [v[0] for v in row];  Gval = [v[1] for v in row];
    bc = -sum([Gval[i]*w0[Gidx[i]] for i in range(len(Gval))])
    task.putavec(mosek.accmode.con, 1+p, Gidx+[n+1+p], Gval+[-1.0])
    task.putbound(mosek.accmode.con, 1+p, mosek.boundkey.fx, bc, bc)
    p += 1

  task.putavec(mosek.accmode.con, 1+p, range(n), [1.0]*n)
  task.putbound(mosek.accmode.con, 1+p, mosek.boundkey.fx, 0.0, 0.0)

  task.putboundlist(mosek.accmode.var, range(n+1+p),
    [mosek.boundkey.fr]*(n+1+p), [-inf]*(n+1+p), [inf]*(n+1+p))
  if SHORTSELLING==0: task.putboundlist(mosek.accmode.var, range(n),
                        [mosek.boundkey.lo]*n, [-i for i in w0], [inf]*n)

  task.appendcone(mosek.conetype.quad, 0.0, range(n, n+1+p))

  task.optimize();  x = [0.0]*n;  f = [0.0]*1
  task.getsolutionslice(mosek.soltype.itr, mosek.solitem.xx, 0, n, x)
  task.getsolutionslice(mosek.soltype.itr, mosek.solitem.xx, n, n+1, f)
  [prosta,solsta] = task.getsolutionstatus(mosek.soltype.itr)
if __name__ == '__main__': main()
```

Code listing 1: Minimum risk portfolio selection

Python

```
import sys, csv, mosek
from minrisk import minrisk
from sharpe import sharpe

def streamprinter(x): sys.stdout.write(x); sys.stdout.flush()
env = mosek.Env(); env.set_Stream (mosek.streamtype.log, streamprinter)

def newtask():
  task = env.Task(0,0)
  task.set_Stream(mosek.streamtype.log, streamprinter)
  return task

def main ():
  f_itrT = open('../data/sp500-ret.csv','rb')  # expected returns
  rT     = csv.reader(f_itrT, quoting=csv.QUOTE_NONNUMERIC).next()
  f_itG  = open('../data/sp500-Gqr.csv','rb')  # covariance factor
  itG    = csv.reader(f_itG, quoting=csv.QUOTE_NONNUMERIC)
  G      = [[(i,v) for i,v in enumerate(row) if v!=0.0] for row in itG]
  n = len(rT);  w0 = [1.0/n]*n

  [start, stop, num] = [1.0, 1.005, 21]
  RET = [start + float(i)/(num-1)*(stop-start) for i in range(num)]

  RISK = [0.0]*len(RET)
  for i in range(len(RET)):
    [_,RISK[i]] = minrisk(newtask(), G, rT, RET[i], w0)

  rf = 1.0005
  [_,risk2,ret2] = sharpe(newtask(), G, rT, w0, rf)
  sharperatio = (ret2 - rf*sum(w0))/risk2

  for item in zip(RET,RISK):
    print item
  print 'sharpe:\n', sharperatio
  print (ret2,risk2)

if __name__ == '__main__':
  main()
```

Code listing 2: Illustrating largest Sharpe ratio without short-selling

Python

```
import mosek
def minrisk(task, G, rT, t, w0):
  # Minimum risk Markowitz portfolio selection (assuming no short-selling)
  #   minimize      f
  #   subject to  [ r'  0   0 ] [ x ]   [ t-r'*w0 ]
  #               [ G   0  -I ]*[ f ] = [  -G*w0  ]
  #               [ e'  0   0 ] [ g ]   [    0    ]
  #               ||g||_2 <= f ,  and w0 + x >= 0.
  n = len(rT);  p = 0;  inf = 0.0

  task.putobjsense(mosek.objsense.minimize)
  task.append(mosek.accmode.var, n+1+len(G)) # x, f and g
  task.append(mosek.accmode.con, 1+len(G)+1) # constraints
  task.putcj(n,1.0)

  bc = t - sum([rT[i]*w0[i] for i in range(n)])
  task.putavec(mosek.accmode.con, 0, range(n), rT)
  task.putbound(mosek.accmode.con, 0, mosek.boundkey.fx, bc, bc)

  for row in G:
    Gidx = [v[0] for v in row];  Gval = [v[1] for v in row]
    bc = -sum([Gval[i]*w0[Gidx[i]] for i in range(len(Gval))])
    task.putavec(mosek.accmode.con, 1+p, Gidx+[n+1+p], Gval+[-1.0])
    task.putbound(mosek.accmode.con, 1+p, mosek.boundkey.fx, bc, bc)
    p += 1

  task.putavec(mosek.accmode.con, 1+p, range(n), [1.0]*n)
  task.putbound(mosek.accmode.con, 1+p, mosek.boundkey.fx, 0.0, 0.0)

  task.putboundlist(mosek.accmode.var, range(n+1+p),
      [mosek.boundkey.lo]*(n) + [mosek.boundkey.fr]*(1+p),
      [-i for i in w0] + [-inf]*(1+p),
      [inf]*(n+1+p))

  task.appendcone(mosek.conetype.quad, 0.0, range(n, n+1+p))

  task.optimize();  x = [0.0]*n;  f = [0.0]*1
  task.getsolutionslice(mosek.soltype.itr, mosek.solitem.xx, 0, n, x)
  task.getsolutionslice(mosek.soltype.itr, mosek.solitem.xx, n, n+1, f)
  return [x,f[0]]
```

Function **minrisk**: Minimum risk portfolio selection without short-selling

Python

```
import mosek
def sharpe(task, G, rT, w0, rf):
  # Computes the largest Sharpe ratio (assuming no short-selling),
  #   minimize       f
  #   subject to  [ (r-e*rf)'  0   0       0       ] [ y ]   [ 1 ]
  #               [    G       0  -I       0       ]*[ f ] = [ 0 ]
  #               [    e'      0   0   -sum(w0)    ] [ g ]   [ 0 ]
  #                                                  [ t ]
  #               t >= 0,   y >= 0,   ||g||_2 <= f
  n = len(rT);   p = 0;   nvars = n+1+len(G)+1;   inf = 0.0

  task.putobjsense(mosek.objsense.minimize)
  task.append(mosek.accmode.var, nvars)       # y, f, g and t
  task.append(mosek.accmode.con, 1+len(G)+1) # constraints
  task.putcj(n,1.0)

  task.putavec(mosek.accmode.con, 0, range(n), [i-rf for i in rT])
  task.putbound(mosek.accmode.con, 0, mosek.boundkey.fx, 1.0, 1.0)

  for row in G:
    Gidx = [v[0] for v in row];   Gval = [v[1] for v in row]
    task.putavec(mosek.accmode.con, 1+p, Gidx+[n+1+p], Gval+[-1.0])
    task.putbound(mosek.accmode.con, 1+p, mosek.boundkey.fx, 0.0, 0.0)
    p += 1

  task.putavec(mosek.accmode.con, 1+p, range(n) + [n+1+p],
                                       [1.0]*n  + [-sum(w0)])
  task.putbound(mosek.accmode.con, 1+p, mosek.boundkey.fx, 0.0, 0.0)

  task.putboundlist(mosek.accmode.var, range(n+1+p+1),
      [mosek.boundkey.lo]*n+[mosek.boundkey.fr]*(p+1)+[mosek.boundkey.lo],
      [0]*n + [-inf]*(p+1) + [0], # lower bounds
      [inf]*nvars)                # upper bounds

  task.appendcone(mosek.conetype.quad, 0.0, range(n, n+1+p))

  task.optimize();   xx = [0.0]*nvars
  task.getsolutionslice(mosek.soltype.itr, mosek.solitem.xx, 0, nvars, xx)
  y = xx[0:n];   f = xx[n:n+1];   g = xx[n+1:n+1+p];   t = xx[n+1+p:nvars]

  x    = [y[i]/t[0]-w0[i] for i in range(n)]
  risk = f[0]/t[0]
  ret  = sum([rT[i]*(x[i]+w0[i]) for i in range(n)])
  return [x,risk,ret]
```

Function **sharpe**: Largest Sharpe ratio without short-selling

References

[1] Richard C. Grinold abd Ronald N. Kahn. *Active portfolio management*. McGraw-Hill, New York, 2 edition, 2000.

[2] A. Ben-Tal and A Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MPS/SIAM Series on Optimization. SIAM, 2001.

[3] Google Finance. Historical stock data. http://www.google.com/finance/.

[4] M. S. Lobo, L. Vanderberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra Appl.*, 284:193–228, November 1998.

[5] MOSEK ApS, Fruebjergvej 3, Boks 16, 2100 Copenhagen O, Denmark. *The MOSEK optimization tools manual – version 6.0*, 2009.

[6] Wikipedia. List of S&P 500 companies. http://en.wikipedia.org/wiki/List_of_S%26P_500_companies.