An introduction to portfolio optimization using MOSEK

Gábor Balló
Optimization Specialist, MOSEK ApS
Email: gabor.ballo@mosek.com

www.mosek.com

Outline

Section 1

Introduction

Purpose of this talk

- Show that solving **portfolio optimization** problems with Mosek is very easy.
- Show how **you** can do it. (Try it at home!)
- Show some possible extensions.

Mosek is a software package for solving

- Linear and **conic** problems.
- Convex quadratically constrained problems.
- Mixed-integer versions of the above.

Mosek Fusion: High level interface, almost direct mapping of math to code. (You will see later.)

What is a conic problem?

Convex optimization

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & \mathbf{g}(\mathbf{x}) & \leq \mathbf{0}, \\ & \mathbf{h}(\mathbf{x}) & = \mathbf{0}, \end{array}$$

- Old way.
- f, components of $\mathbf{g}$ convex.
- Components of $\mathbf{h}$ affine.
- Needs gradients, Hessians.

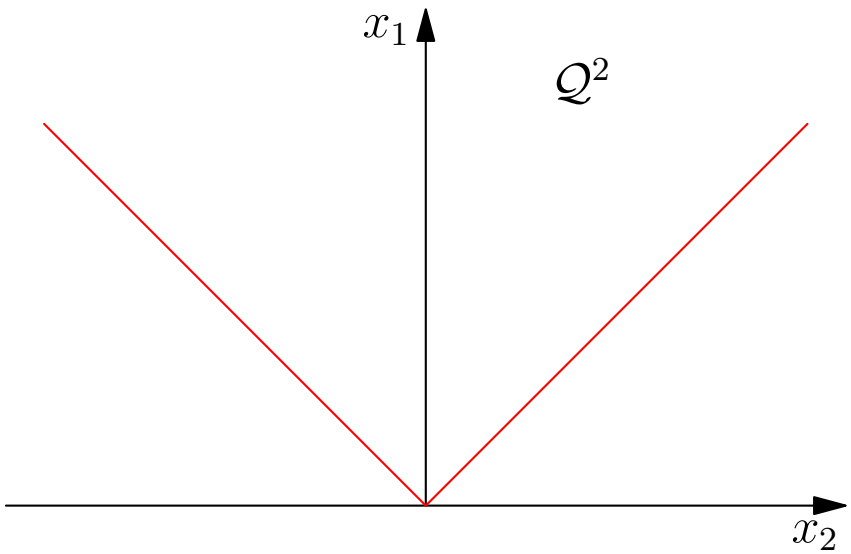
Conic optimization

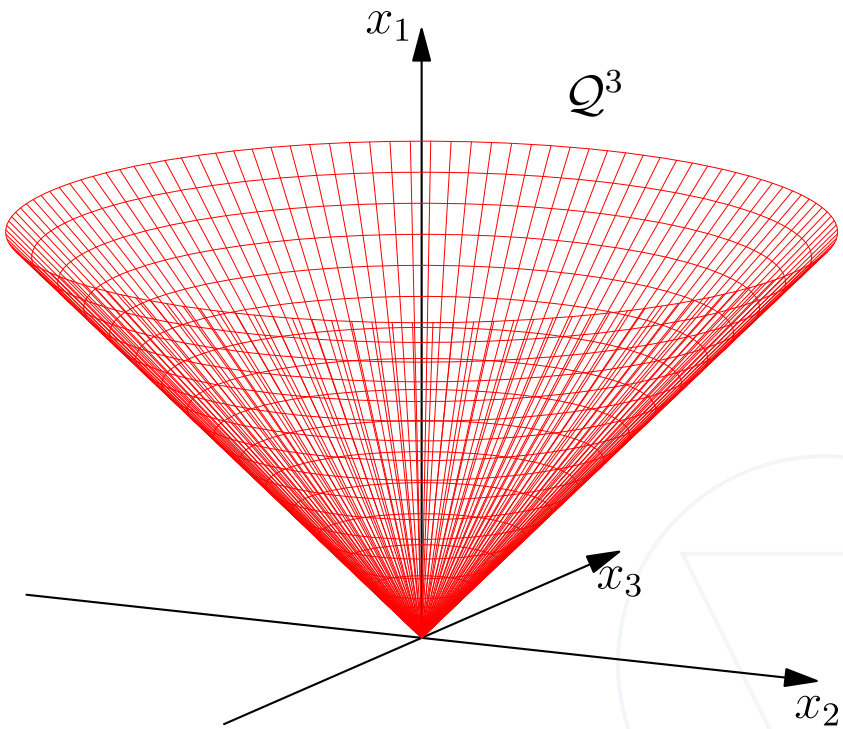
$$\begin{array}{ll} \text{minimize} & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} + \mathbf{b} \in \mathcal{K}, \end{array}$$

- New way.
- $\mathcal{K}$ is a **convex cone**.
- Similar to linear programming.
- Interior-point solvers.
- Easy for Mosek financial customers.

Example: Quadratic cone

- $\mathcal{Q}^n = \{\mathbf{x} \in \mathbb{R}^n | x_1 \geq \|\mathbf{x}_{2..n}\|_2\}$, also called *second-order cone*.
- Used for modeling problems with quadratic functions. (Portfolio optimization!)
- 2D example: $|x_2| \leq x_1 \iff (x_1, x_2) \in \mathcal{Q}^2$.
- 3D example: $\|(x_2, x_3)\|_2 \leq x_1 \iff (x_1, x_2, x_3) \in \mathcal{Q}^3$.





Section 2

Mean-variance portfolio optimization

The mean-variance model

Goal

Maximize the *return* of the investment, while keeping the investment *risk* (variance) acceptable.

- R_i is the *rate of return* of security i at the end of the investment period. (random variable)
- $\mu_i = \mathbb{E}(R_i)$ is the expected return security i.
- $\Sigma_{i,j} = \mathrm{cov}(R_i, R_j)$ is the covariance of security pair (i, j).

Modern Portfolio Theory

- Risk is reduced through diversification, forming a *portfolio*.
- Portfolio weights control the tradeoff between return and risk.
- Find the optimal weights, then invest!

The mean-variance model

- $\mathbf{x}$ is the portfolio weight vector, x_i is the fraction of funds invested into security i.
- $R_{\mathbf{x}} = \sum_i x_i R_i$ is the portfolio return. (random)
- $\mu_{\mathbf{x}} = \mathbb{E}(R_{\mathbf{x}}) = \mathbf{x}^{\mathsf{T}}\mu$ is the expected portfolio return.
- $\sigma^2_{\mathbf{x}} = \operatorname{cov}(R_{\mathbf{x}}) = \mathbf{x}^{\mathsf{T}}\Sigma\mathbf{x}$ is the portfolio variance.

Mean–variance optimization (MVO)

- Find $\mathbf{x}$, that gives the optimal tradeoff between $\mu_{\mathbf{x}}$ and $\sigma^2_{\mathbf{x}}$.
- Variance as risk measure: MVO is a *quadratic optimization* (QO) problem.

Solution of MVO

Data

- $\boldsymbol{\mu}$: The estimate of the vector μ.
- $\boldsymbol{\Sigma}$: The estimate of the matrix Σ.

Simplifications

- Only risky securities in the universe.
- No transaction costs.
- No initial holdings: $\mathbf{x}_0 = \mathbf{0}$.
- Fully invested portfolio: $\sum_i x_i = 1$.

We can formulate four equivalent optimization problems.

Solution of MVO

Minimize the portfolio risk, bound the portfolio return

$$\begin{array}{lrcl} \text{minimize} & \mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}\mathbf{x} & & \\ \text{subject to} & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} & \geq & r_{\min}, \\ & \mathbf{1}^{\mathsf{T}}\mathbf{x} & = & 1. \end{array} \tag{1}$$

QO problem, easy to solve.

Maximize the portfolio return, bound the portfolio risk

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} & & \\ \text{subject to} & \mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}\mathbf{x} & \leq & \gamma^2, \\ & \mathbf{1}^{\mathsf{T}}\mathbf{x} & = & 1. \end{array} \tag{2}$$

QCQO problem, easy after conic reformulation.

Solution of MVO

Maximize the portfolio return with variance penalty

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} - \frac{\delta}{2}\mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}\mathbf{x} & & \\ \text{subject to} & \mathbf{1}^{\mathsf{T}}\mathbf{x} & = & 1. \end{array} \tag{3}$$

- QO problem.
- δ: Tradeoff parameter, no intuitive meaning.

Maximize the portfolio return with std. dev. penalty

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} - \tilde{\delta}\sqrt{\mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}\mathbf{x}} & & \\ \text{subject to} & \mathbf{1}^{\mathsf{T}}\mathbf{x} & = & 1. \end{array} \tag{4}$$

- Not QO, but easy after conic reformulation.
- Penalty term is of the same scale as portfolio return.
- $\tilde{\delta}$: Distance from mean return in std. dev. units. (Intuitive.)

Conic formulation

1. Factorize the covariance matrix

- $\mathbf{\Sigma} = \mathbf{G}\mathbf{G}^{\mathsf{T}}$, where $\mathbf{G} \in \mathbb{R}^{N \times K}$.
- Cholesky factorization: $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$, for $\mathbf{A} \succ \mathbf{0}$, $\mathbf{L}$ lower triang.
- Factor model: Has direct financial interpretation. (see later)

2. Rewrite portfolio variance constraint

- Quadratic constraint: $\mathbf{x}^{\mathsf{T}}\mathbf{\Sigma}\mathbf{x} \leq \gamma^2$
- $\mathbf{x}^{\mathsf{T}}\mathbf{\Sigma}\mathbf{x} = \mathbf{x}^{\mathsf{T}}\mathbf{G}\mathbf{G}^{\mathsf{T}}\mathbf{x} = (\mathbf{G}^{\mathsf{T}}\mathbf{x})^{\mathsf{T}}(\mathbf{G}^{\mathsf{T}}\mathbf{x}) = \|\mathbf{G}^{\mathsf{T}}\mathbf{x}\|_2^2$
- Equivalent, conic representable constraint: $\|\mathbf{G}^{\mathsf{T}}\mathbf{x}\|_2 \leq \gamma$

3. Model with quadratic cone

$$\|\mathbf{G}^{\mathsf{T}}\mathbf{x}\|_2 \leq \gamma \iff (\gamma, \mathbf{G}^{\mathsf{T}}\mathbf{x}) \in \mathcal{Q}^{K+1}$$

Conic formulation

Problem (2) (return maximization)

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} & & \\ \text{subject to} & \mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}\mathbf{x} & \leq & \gamma^2, \\ & \mathbf{1}^{\mathsf{T}}\mathbf{x} & = & 1. \end{array} \tag{2}$$

The conic equivalent of problem (2)

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} & & \\ \text{subject to} & (\gamma, \mathbf{G}^{\mathsf{T}}\mathbf{x}) & \in & \mathcal{Q}^{K+1}, \\ & \mathbf{1}^{\mathsf{T}}\mathbf{x} & = & 1. \end{array} \tag{2'}$$

Conic formulation

Problem (4) (std. dev. penalty)

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^T\mathbf{x} - \tilde{\delta}\sqrt{\mathbf{x}^T\boldsymbol{\Sigma}\mathbf{x}} & & \\ \text{subject to} & \mathbf{1}^T\mathbf{x} & = & 1. \end{array} \tag{4}$$

The conic equivalent of problem (4)

- New variable s: Upper bound of $\sqrt{\mathbf{x}^T\boldsymbol{\Sigma}\mathbf{x}} = \|\mathbf{G}^T\mathbf{x}\|_2$.
- Add constraint $\|\mathbf{G}^T\mathbf{x}\|_2 \leq s$, model using the quadratic cone.

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^T\mathbf{x} - \tilde{\delta}s & & \\ \text{subject to} & (s, \mathbf{G}^T\mathbf{x}) & \in & \mathcal{Q}^{K+1}, \\ & \mathbf{1}^T\mathbf{x} & = & 1. \end{array} \tag{4'}$$

If Markowitz had known this, would he have worked with std. dev.?

Example in MOSEK Fusion: Simple MVO

Assumptions:

- $\boldsymbol{\mu}$ is given, $\boldsymbol{\Sigma}$ is given and is positive definite.
- We create a long only portfolio of eight stocks. $(N = 8)$

Inputs:

```
# Expected returns and covariance matrix
m = np.array(
    [0.0720, 0.1552, 0.1754, 0.0898, 0.4290, 0.3929, 0.3217, 0.1838]
)
S = np.array([
    [0.0946, 0.0374, 0.0349, 0.0348, 0.0542, 0.0368, 0.0321, 0.0327],
    [0.0374, 0.0775, 0.0387, 0.0367, 0.0382, 0.0363, 0.0356, 0.0342],
    [0.0349, 0.0387, 0.0624, 0.0336, 0.0395, 0.0369, 0.0338, 0.0243],
    [0.0348, 0.0367, 0.0336, 0.0682, 0.0402, 0.0335, 0.0436, 0.0371],
    [0.0542, 0.0382, 0.0395, 0.0402, 0.1724, 0.0789, 0.0700, 0.0501],
    [0.0368, 0.0363, 0.0369, 0.0335, 0.0789, 0.0909, 0.0536, 0.0449],
    [0.0321, 0.0356, 0.0338, 0.0436, 0.0700, 0.0536, 0.0965, 0.0442],
    [0.0327, 0.0342, 0.0243, 0.0371, 0.0501, 0.0449, 0.0442, 0.0816]
])
```

Example in MOSEK Fusion: Simple MVO

We choose to solve problem (2) (return maximization).

- Additional constraint to prevent short selling.
- Risk limit of $\gamma^2 = 0.05$.
- Cholesky factorization: $\mathbf{\Sigma} = \mathbf{G}\mathbf{G}^\mathsf{T}$, $\mathbf{G} \in \mathbb{R}^{8\times 8}$.

Conic model of problem (2)

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} & & \\ \text{subject to} & (\sqrt{0.05}, \mathbf{G}^\mathsf{T}\mathbf{x}) & \in & \mathcal{Q}^9, \\ & \mathbf{1}^\mathsf{T}\mathbf{x} & = & 1, \\ & \mathbf{x} & \geq & 0. \end{array} \qquad (2'')$$

Example in MOSEK Fusion: Simple MVO

The Mosek Python Fusion model of problem (2"):

```
with Model("markowitz") as M:
    M.setLogHandler(sys.stdout)  # Send log to stdout.

    # Decision variable (imposes the no short-selling constraint)
    x = M.variable('x', N, Domain.greaterThan(0.0))

    # Budget constraint
    M.constraint('budget', Expr.sum(x), Domain.equalsTo(1))
    # Portfolio risk constraint
    M.constraint('risk', Expr.vstack(np.sqrt(0.05), Expr.mul(G.T, x)),
                 Domain.inQCone())

    # Objective
    M.objective('obj', ObjectiveSense.Maximize, Expr.dot(m, x))

    M.solve()  # Solve optimization

    preturn = M.primalObjValue()
    portfolio = x.level()
```

Optimal $\mathbf{x}$: $[0.0, 0.091, 0.269, 0.0, 0.025, 0.322, 0.177, 0.116]$.

Example in MOSEK Fusion: Efficient frontier

Efficient frontier

- The collection of optimal (expected return, variance) points.
- Easiest to find using problem (4) by varying the $\tilde{\delta}$ parameter.

Conic model of problem (4)

$$\begin{array}{lrcl} \text{maximize} & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} - \tilde{\delta}s & & \\ \text{subject to} & \mathbf{1}^{\mathsf{T}}\mathbf{x} & = & 1, \\ & \mathbf{x} & \geq & 0, \\ & (s, \mathbf{G}^{\mathsf{T}}\mathbf{x}) & \in & \mathcal{Q}^9. \end{array} \qquad (4'')$$

Example in MOSEK Fusion: Efficient frontier

The Mosek Python Fusion model of problem (4”):

```
with Model("efficient_frontier") as M:
    M.setLogHandler(sys.stdout)  # Send log to stdout.

    # Decision variable (imposes the no short-selling constraint)
    x = M.variable('x', N, Domain.greaterThan(0.0))
    # Variable 's' models the portfolio std. dev. term in the objective.
    s = M.variable('s', 1, Domain.unbounded())

    # Budget constraint and portfolio risk constraint
    M.constraint('budget', Expr.sum(x), Domain.equalsTo(1))
    M.constraint('risk', Expr.vstack(s, Expr.mul(G.T, x)), Domain.inQCone()

    # Objective (with model parameter)
    delta = M.parameter()
    M.objective('obj', ObjectiveSense.Maximize,
                Expr.sub(Expr.dot(m, x), Expr.mul(delta, s)))
```

[continued below]

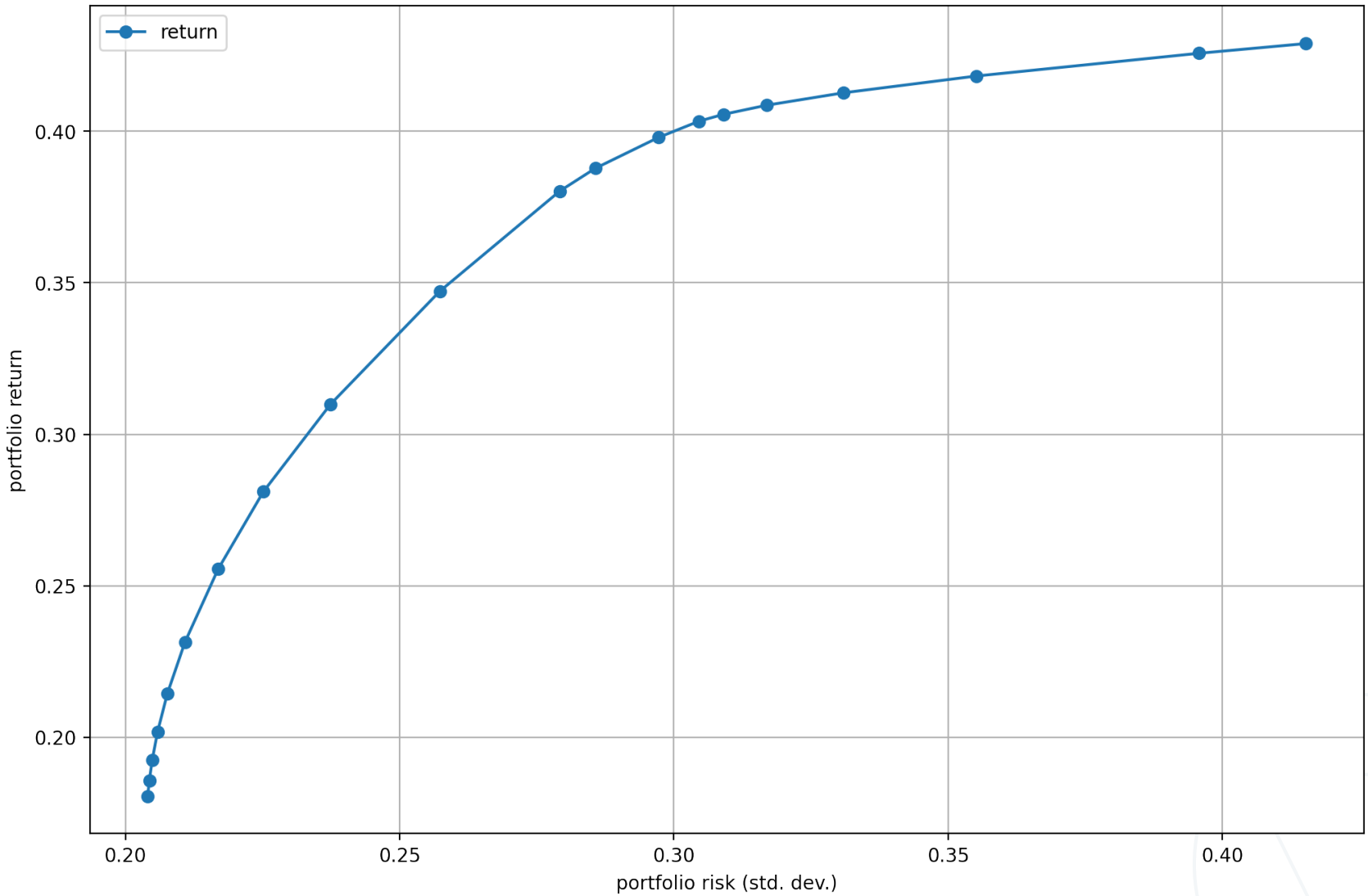
Example in MOSEK Fusion: Efficient frontier

[continued from above]

```
        # Solve the optimization model for all parameter values
        for d in deltas:
            # Update parameter
            delta.setValue(d)

            # Solve optimization
            M.solve()

            # Save results
            portfolio_return = m @ x.level()
            portfolio_risk = s.level()[0]
```

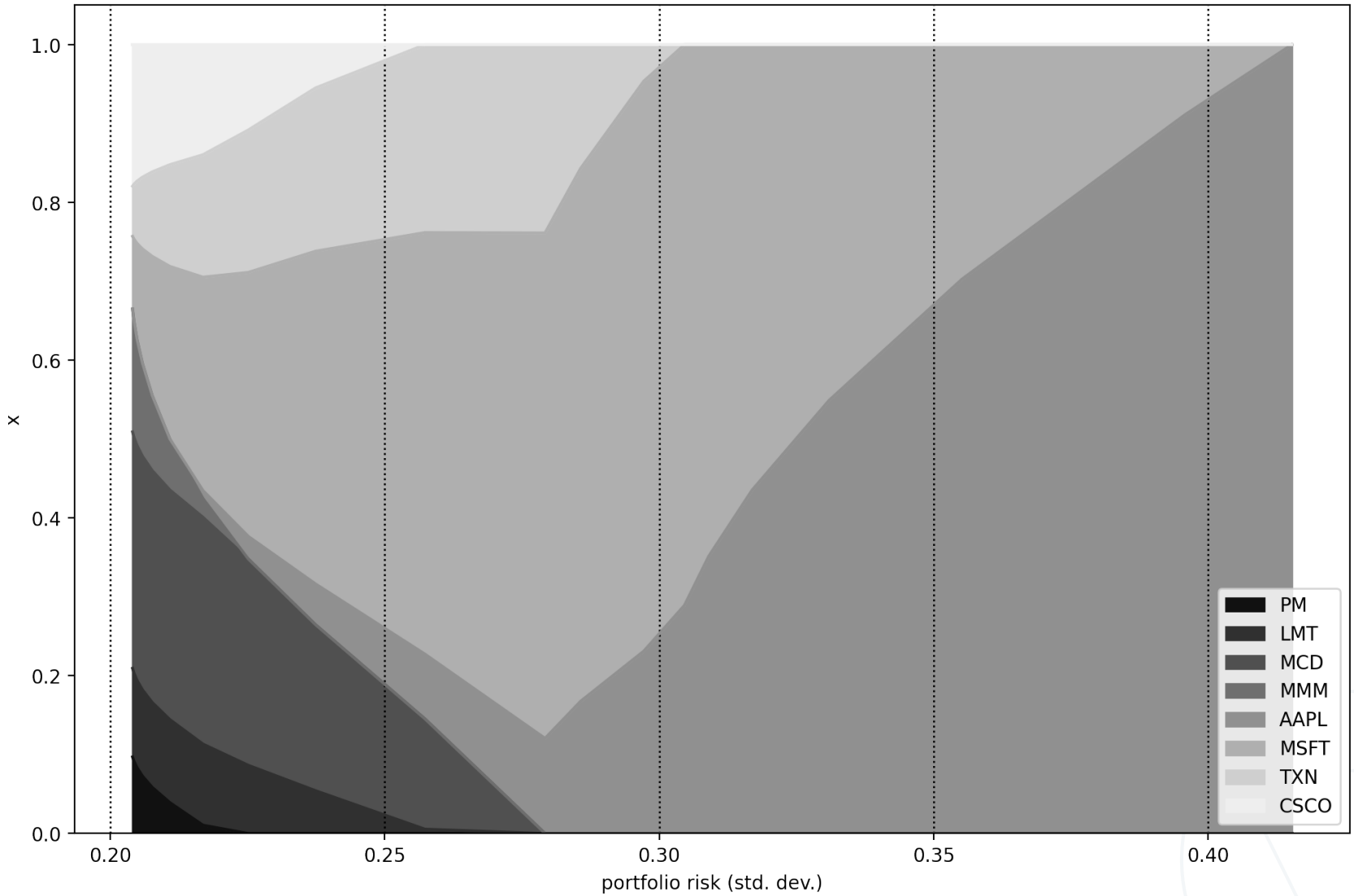
Example in MOSEK Fusion: Efficient frontier

The resulting efficient frontier:



Example in MOSEK Fusion: Efficient frontier

The resulting porfolios:



Summary

Takeaway of this section

1. Conic optimization is a generalization of linear programming.
2. **You** can convert portfolio optimization problems into conic equivalents.
3. **You** can easily map math into code using Mosek Fusion API.
4. **You** can solve conic problems very efficiently.

Section 3

Factor models

Factor models

Definition:

- Explains random variable R_t (N dim.) through a small number of common factors F_t (K dim.).
- $R_t = \beta F_t + \theta_t$.
- β: Factor exposure matrix. ($N \times K$ dim.)
- $\theta_t = \alpha + \varepsilon_t$: Specific component at time t. (N dim.)
- $\alpha = \mathbb{E}(\theta_t)$, ε_t is white noise with covariance Σ_θ.

Factor model[1] structure of the covariance matrix

- $\Sigma = \beta \Sigma_F \beta^{\mathsf{T}} + \Sigma_\theta$, where Σ_θ is diagonal.
- Only $N(K+1) + K(K+1)/2$ parameters (linear in N).
- No factor model: $N(N+1)/2$ covariances (quadratic in N).

[1] Exact, static factor models with weakly stationary F_t.

Conic constrains with factor model

We have the decomposition $\mathbf{\Sigma} = \boldsymbol{\beta}\mathbf{\Sigma}_F\boldsymbol{\beta}^\mathsf{T} + \mathbf{\Sigma}_\theta$, where

- $\mathbf{\Sigma}_F$ is the $K \times K$ sample factor covariance matrix (K factors).
- Typically $K \ll N$, thus $\mathbf{\Sigma}_F$ is very low dimensional.
- Very cheap to compute $\mathbf{\Sigma}_F = \mathbf{F}\mathbf{F}^\mathsf{T}$, and get $\mathbf{\Sigma} = \mathbf{G}\mathbf{G}^\mathsf{T}$ with

$$\mathbf{G} = \begin{bmatrix} \boldsymbol{\beta}\mathbf{F} & \mathbf{\Sigma}_\theta^{1/2} \end{bmatrix}.$$

Sparsity

- This form of $\mathbf{G}$ is typically very sparse.
- Computational efficiency: sparsity is more important than the number of variables and constraints!
- Dimension: $N \times (N + K) > N \times N$.
- Nonzeros: $N(K + 1) \ll N(N + 1)/2$. (storage size)
- Leads to significant reduction in the solution time!

Example in Python: 2-factor model

Assume we have two index ETFs as factors for the eight stocks:

$$R_t = \alpha + \beta \begin{pmatrix} R_{F_1,t} \\ R_{F_2,t} \end{pmatrix} + \varepsilon_t.$$

Estimate $\boldsymbol{\beta}$, $\boldsymbol{\Sigma}_F$, $\boldsymbol{\Sigma}_\theta$ (in code `B`, `S_F`, and `S_theta`):

$$\boldsymbol{\beta} = \begin{bmatrix} 0.4256 & 0.1869 \\ 0.2413 & 0.3877 \\ 0.2235 & 0.3697 \\ 0.1503 & 0.4612 \\ 1.5325 & -0.2633 \\ 1.2741 & -0.2613 \\ 0.6939 & 0.2372 \\ 0.5425 & 0.2116 \end{bmatrix}, \quad \boldsymbol{\Sigma}_F = \begin{bmatrix} 0.0620 & 0.0577 \\ 0.0577 & 0.0908 \end{bmatrix},$$

$$\boldsymbol{\Sigma}_\theta = \mathrm{Diag}([0.0720, 0.0508, 0.0377, 0.0394, 0.0663, 0.0224, 0.0417, 0.0459]),$$

Example in Python: 2-factor model

Compute the matrix $\mathbf{G}$:

```
P = np.linalg.cholesky(S_F)
G = np.block([[B @ P, np.sqrt(S_theta)]])
```

Then $\mathbf{G}$ will be

```
array([
    [0.149, 0.036, 0.268, 0.,    0.,    0.,    0.,    0.,    0.,    0.   ],
    [0.150, 0.075, 0.,    0.225, 0.,    0.,    0.,    0.,    0.,    0.   ],
    [0.141, 0.071, 0.,    0.,    0.194, 0.,    0.,    0.,    0.,    0.   ],
    [0.144, 0.089, 0.,    0.,    0.,    0.199, 0.,    0.,    0.,    0.   ],
    [0.321,-0.051, 0.,    0.,    0.,    0.,    0.258, 0.,    0.,    0.   ],
    [0.257,-0.050, 0.,    0.,    0.,    0.,    0.,    0.150, 0.,    0.   ],
    [0.228, 0.046, 0.,    0.,    0.,    0.,    0.,    0.,    0.204, 0.   ],
    [0.184, 0.041, 0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.214]
])
```

Compare to Cholesky factor:

- Size: 8×10 vs. 8×8.
- Nonzeros: 24 vs. 36.
- For $N = 1000$ and $K = 50$: 51000 vs. half million nonzeros. (See next talk.)

Summary

Takeaway of this section

1. Factor models impose a special structure on the covariance matrix: $\boldsymbol{\Sigma} = \boldsymbol{\beta}\boldsymbol{\Sigma}_F\boldsymbol{\beta}^\mathsf{T} + \boldsymbol{\Sigma}_\theta$, $\boldsymbol{\Sigma}_F \in \mathbb{R}^{K\times K}$, $K \ll N$.
2. If you use a factor model, you can exploit this structure in conic optimization: $\mathbf{G} = [\boldsymbol{\beta}\mathbf{F}\ \boldsymbol{\Sigma}_\theta^{1/2}]$ very sparse.
3. This can make **large scale** portfolio optimization problems run **orders of magnitude faster**!

Section 4

Transaction cost modeling

Transaction costs

Cost model

$\sum_{i=1}^{n} C_i(\tilde{x}_i)$, where $\tilde{x}_i = x_i - x_{0,i}$.

MVO model with transaction cost

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} \\ \text{subject to} \quad & \mathbf{1}^{\mathsf{T}}\mathbf{x} + \sum_{i=1}^{n} C_i(\tilde{x}_i) = 1, \\ & \mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}\mathbf{x} \leq \gamma^2. \end{aligned}$$

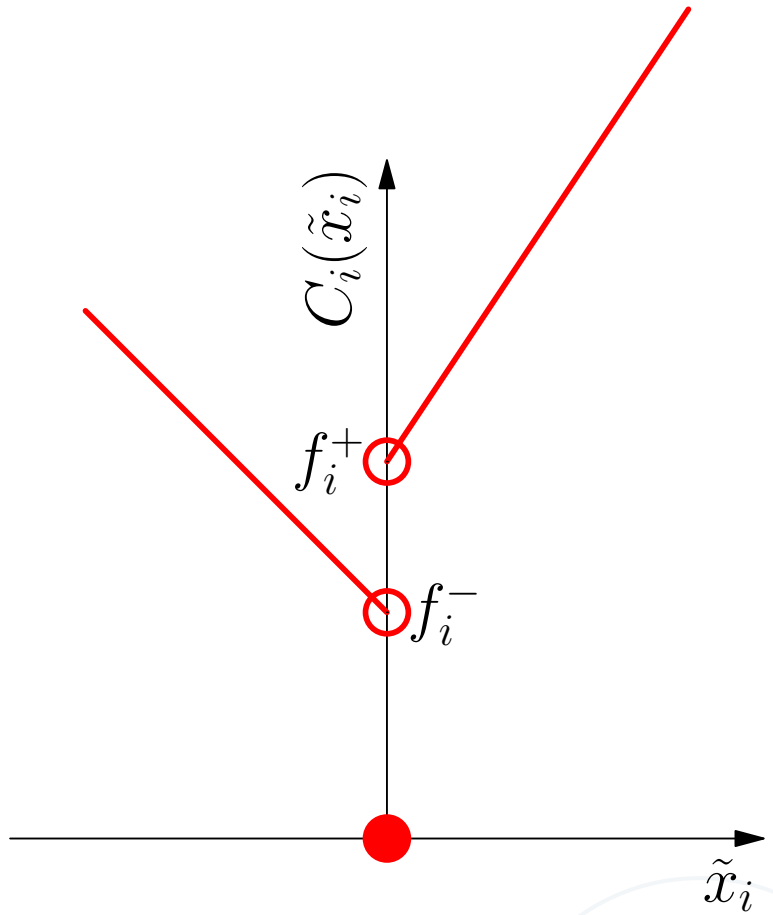
Self financing: No external cash is added to the portfolio.

Fixed + variable cost model

Cost function

$$C_i(\tilde{x}_i) = \begin{cases} 0, & \tilde{x}_i = 0, \\ f_i^+ + v_i^+ \tilde{x}_i, & \tilde{x}_i > 0, \\ f_i^- - v_i^- \tilde{x}_i, & \tilde{x}_i < 0. \end{cases}$$

- $\tilde{x}_i^+, \tilde{x}_i^-$: Pos./neg. part of $\tilde{x}_i$.
- f_i^+, f_i^-: Fixed costs of buying/selling security i.
- v_i^+, v_i^-: Cost rates of buying/selling security i.



Not convex, but solvable as Mixed integer (MIO) problem.

Fixed + variable cost model

$$
\begin{aligned}
\text{maximize} \quad & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} \\
\text{subject to} \quad & \mathbf{1}^{\mathsf{T}}\mathbf{x} + \begin{bmatrix}\mathbf{f}^{+}\\ \mathbf{f}^{-}\end{bmatrix}^{\mathsf{T}} \begin{bmatrix}\mathbf{y}^{+}\\ \mathbf{y}^{-}\end{bmatrix} + \begin{bmatrix}\mathbf{v}^{+}\\ \mathbf{v}^{-}\end{bmatrix}^{\mathsf{T}} \begin{bmatrix}\tilde{\mathbf{x}}^{+}\\ \tilde{\mathbf{x}}^{-}\end{bmatrix} &= 1, \\
& \tilde{\mathbf{x}} &= \tilde{\mathbf{x}}^{+} - \tilde{\mathbf{x}}^{-}, \\
& \tilde{\mathbf{x}}^{+}, \tilde{\mathbf{x}}^{-} &\geq 0, \\
& \tilde{\mathbf{x}}^{+} &\leq \mathbf{u}^{+} \circ \mathbf{y}^{+}, \\
& \tilde{\mathbf{x}}^{-} &\leq \mathbf{u}^{-} \circ \mathbf{y}^{-}, \\
& \mathbf{y}^{+} + \mathbf{y}^{-} &\leq \mathbf{1}, \\
& \mathbf{y}^{+}, \mathbf{y}^{-} &\in \{0,1\}^{N}, \\
& \mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}\mathbf{x} &\leq \gamma^{2}.
\end{aligned}
$$

- $\mathbf{y}^{+}, \mathbf{y}^{-}$: Binary vectors indicating buys/sells. (On/off switch.)
- $u_i^{+} y_i^{+}$ and $u_i^{-} y_i^{-}$: Forces $\tilde{\mathbf{x}}^{+}/\tilde{\mathbf{x}}^{-}$ to 0 if not traded.
- $\mathbf{y}^{+} + \mathbf{y}^{-} \leq \mathbf{1}$: The transaction is either buy or sell.
- Budget constraint: $\tilde{\mathbf{x}}^{+}$ and $\tilde{\mathbf{x}}^{-}$ will not be both positive in any optimal solution.

Example: Fixed + variable cost

Assumptions:

- Short-selling is allowed up to the limit of 30% portfolio size.
- $\tilde{\mathbf{x}} = \mathbf{x}$, because $\mathbf{x}_0 = \mathbf{0}$.

Variables:

```
# Real variables
xp = M.variable("xp", N, Domain.greaterThan(0.0))
xm = M.variable("xm", N, Domain.greaterThan(0.0))

# Buy/sell indicators
yp = M.variable("yp", N, Domain.binary())
ym = M.variable("ym", N, Domain.binary())
```

Constraints:

```
# Constraint assigning xp and xm to the positive and negative part of x.
M.constraint('pos-neg-part', Expr.sub(x, Expr.sub(xp, xm)),
             Domain.equalsTo(0.0))

# Exclusive buy-sell constraint
M.constraint('exclusion', Expr.add(yp, ym), Domain.lessThan(1.0))
```

[continued below]

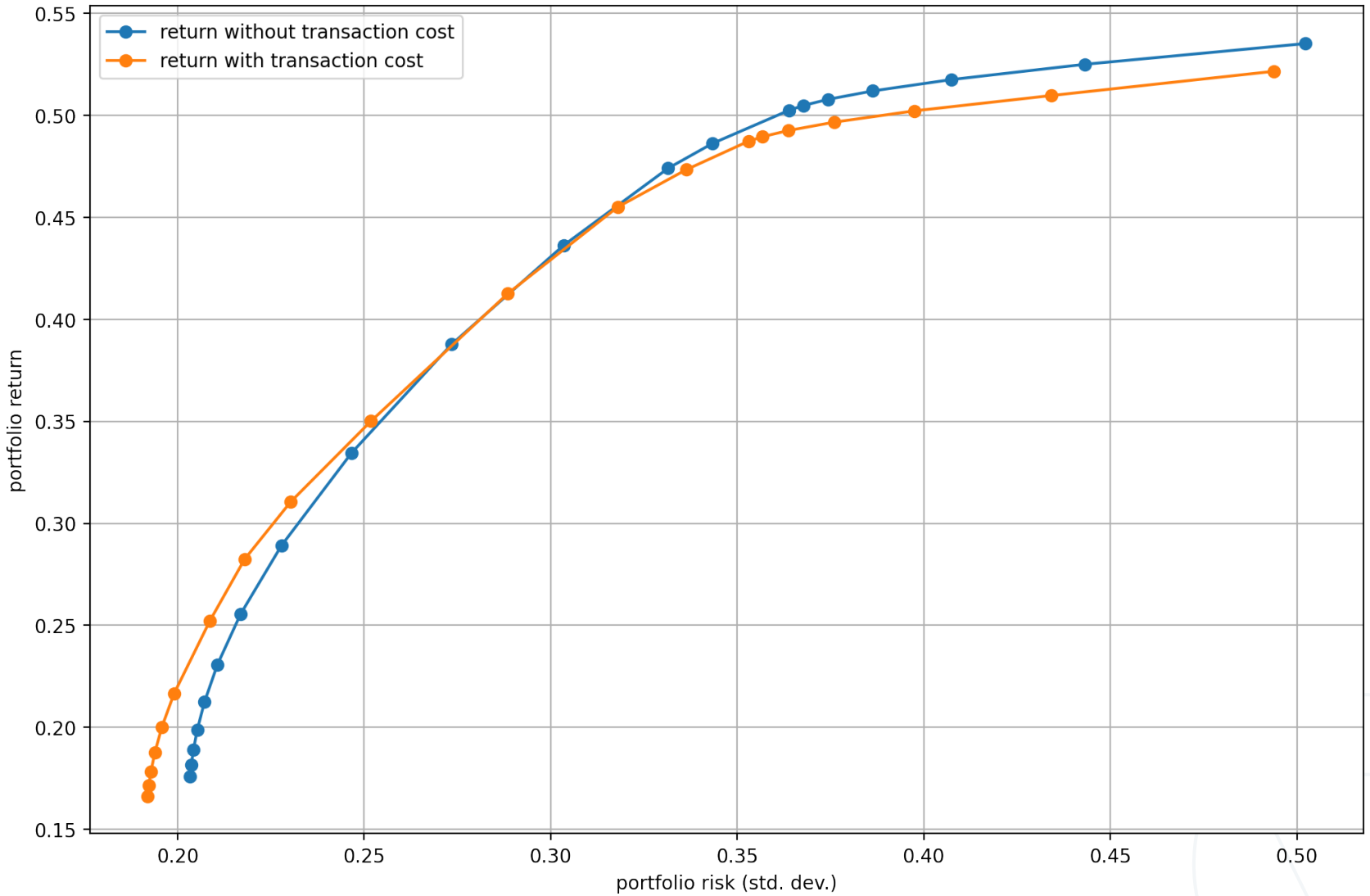
Example: Fixed + variable cost

[continued from above]

```
# Budget constraint with transaction cost terms
fixcost_terms = Expr.add([Expr.dot(fp, yp), Expr.dot(fm, ym)])
varcost_terms = Expr.add([Expr.dot(vp, xp), Expr.dot(vm, xm)])
budget_terms = Expr.add([Expr.sum(x), varcost_terms, fixcost_terms])
M.constraint('budget', budget_terms, Domain.equalsTo(1.0))

# Auxiliary variable for 130/30 leverage constraint
z = M.variable("z", N, Domain.unbounded())

# 130/30 leverage constraint
M.constraint('leverage-gt', Expr.sub(z, x), Domain.greaterThan(0.0))
M.constraint('leverage-ls', Expr.add(z, x), Domain.greaterThan(0.0))
terms = Expr.add([Expr.sum(z), varcost_terms, fixcost_terms])
M.constraint('leverage-sum', terms, Domain.equalsTo(1.6))

# Bound constraints for fixed cost
M.constraint('ub-p', Expr.sub(Expr.mul(up, yp), xp), Domain.greaterThan(0.0
M.constraint('ub-m', Expr.sub(Expr.mul(um, ym), xm), Domain.greaterThan(0.0
M.constraint('lb-p', Expr.sub(xp, Expr.mul(lp, yp)), Domain.greaterThan(0.0
M.constraint('lb-m', Expr.sub(xm, Expr.mul(lm, ym)), Domain.greaterThan(0.0
```

Example: Fixed + variable cost

The resulting efficient frontiers:



Market impact cost

Cost function

$$C_i(\tilde{x}_i) = a_i|\tilde{x}_i|^\beta,$$

where a_i is calibrated and β is typically $3/2$.

Modeling with power cone

- $\mathcal{P}_n^{\alpha,1-\alpha} = \left\{\mathbf{x} \in \mathbb{R}^n \middle| x_1^\alpha x_2^{1-\alpha} \geq \|\mathbf{x}_{3..n}\|_2,\ x_1, x_2 \geq 0\right\}$.
- $|\tilde{x}_i|^\beta \leq t_i \iff (t_i, 1, \tilde{x}_i) \in \mathcal{P}_3^{1/\beta,(\beta-1)/\beta}$.

Total market impact cost model

$$\sum_{i=1}^N a_i|\tilde{x}_i|^\beta \iff \sum_{i=1}^N a_i t_i,\ (t_i, 1, \tilde{x}_i) \in \mathcal{P}_3^{1/\beta,(\beta-1)/\beta} \text{ for } i = 1 \ldots N$$

Market impact cost

Ensuring $|\tilde{x}_i|^\beta = t_i$ at the optimal solution:

- Add a risk-free security to the model.
- x^{f}: The weight of the risk-free security.
- r^{f}: The return of the risk-free security.

Optimization problem with market impact cost

$$
\begin{array}{lrcl}
\text{maximize} & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} + r^{\mathrm{f}}x^{\mathrm{f}} & & \\
\text{subject to} & \mathbf{1}^{\mathsf{T}}\mathbf{x} + \mathbf{a}^{\mathsf{T}}\mathbf{t} + x^{\mathrm{f}} & = & 1, \\
& \mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}\mathbf{x} & \leq & \gamma^2, \\
& (t_i, 1, \tilde{x}_i) & \in & \mathcal{P}_3^{1/\beta,(\beta-1)/\beta}, \ i = 1, \ldots, N.
\end{array}
$$

Example: Market impact cost

We update the Fusion model of (4) with new variables and constraints:

```
with Model("Market impact") as M:
    M.setLogHandler(sys.stdout)  # Send log to stdout.

    # Decision variable (imposes the no short-selling constraint)
    x = M.variable('x', N, Domain.greaterThan(0.0))
    # Variable 's' models the portfolio variance term in the objective.
    s = M.variable('s', 1, Domain.unbounded())
    # Variable for risk-free security (no borrowing)
    xf = M.variable("xf", 1, Domain.greaterThan(0.0))
    # Auxiliary variable to model market impact
    t = M.variable("t", N, Domain.unbounded())

    # Budget constraint, portfolio risk
    budget_terms = Expr.hstack(Expr.sum(x), xf, Expr.dot(a, t))
    M.constraint('budget', Expr.sum(budget_terms), Domain.equalsTo(1))
    M.constraint('risk', Expr.vstack(s, Expr.mul(G.T, x)),
                 Domain.inQCone())
```

[continued below]

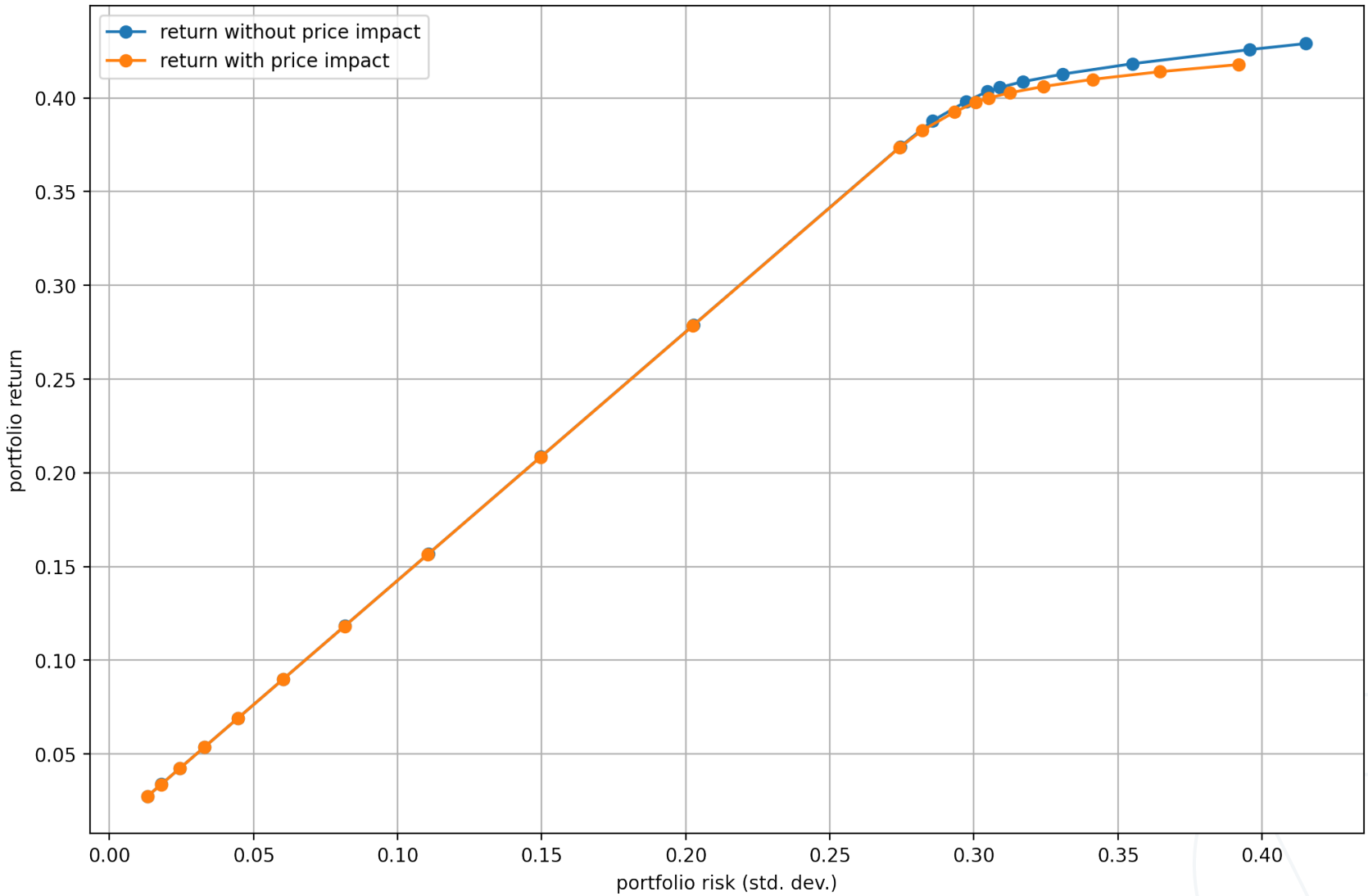
Example: Market impact cost

[continued from above]

```
        # Market impact constraint (x0 assumed to be 100% cash)
        M.constraint('market_impact', Expr.hstack(t, Expr.constTerm(N, 1.0), x)
                     Domain.inPPowerCone(1.0 / beta))

        # Objective
        delta = M.parameter()
        return_terms = Expr.add(Expr.dot(m, x), Expr.mul(rf, xf))
        M.objective('obj', ObjectiveSense.Maximize,
                    Expr.sub(return_terms, Expr.mul(delta, s)))
```

Example: Market impact cost

The resulting efficient frontiers:



Summary

Takeaway of this section

1. We have discussed some examples of **transaction cost models** in portfolio optimization.
2. **You** can also model these as conic and mixed integer conic problems, and solve them using Mosek.

Section 5

Benchmark relative portfolio optimization

Quantities of interest

In benchmark relative setting we use the following quantities:

- Active holdings: $\mathbf{x}_{\mathrm{a}} = \mathbf{x} - \mathbf{x}_{\mathrm{bm}}$.
- Active return (return above benchmark return): $R_{\mathbf{x}_{\mathrm{a}}} = \mathbf{x}_{\mathrm{a}}^{\mathsf{T}} R$.
- Tracking error (std. dev. of active return): $\sigma_{\mathbf{x}_{\mathrm{a}}}(R_{\mathbf{x}}, R_{\mathbf{x}_{\mathrm{bm}}}) = \sqrt{\mathbf{x}_{\mathrm{a}}^{\mathsf{T}} \Sigma \mathbf{x}_{\mathrm{a}}}$, where $\Sigma = \mathrm{Cov}(R)$.

Systematic component of active return

- $R_{\mathbf{x}_{\mathrm{a}}} = (\beta_{\mathbf{x}} - 1) R_{\mathbf{x}_{\mathrm{bm}}} + \theta_{\mathbf{x}}$.
- Active beta: $\beta_{\mathbf{x}} - 1$.
- Alpha: $\alpha_{\mathbf{x}} = \alpha^{\mathsf{T}} \mathbf{x} = \mathbb{E}(\theta_{\mathbf{x}})$.

Optimization problem

- Optimizing tradeoff between **portfolio alpha** and the **squared tracking error**.
- Relevant constrains: bounds on portfolio active beta or on active holdings.

Benchmark relative portfolio optimization

$$
\begin{aligned}
\text{maximize} \quad & \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{x} \\
\text{subject to} \quad & (\mathbf{x}-\mathbf{x}_{\mathrm{bm}})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x}-\mathbf{x}_{\mathrm{bm}}) \leq \gamma_{\mathrm{TE}}^2, \\
& \mathbf{x}-\mathbf{x}_{\mathrm{bm}} \geq \mathbf{l}_{\mathrm{h}}, \\
& \mathbf{x}-\mathbf{x}_{\mathrm{bm}} \leq \mathbf{u}_{\mathrm{h}}, \\
& \boldsymbol{\beta}_{\mathbf{x}}-1 \geq \mathbf{l}_{\beta}, \\
& \boldsymbol{\beta}_{\mathbf{x}}-1 \leq \mathbf{u}_{\beta}.
\end{aligned}
$$

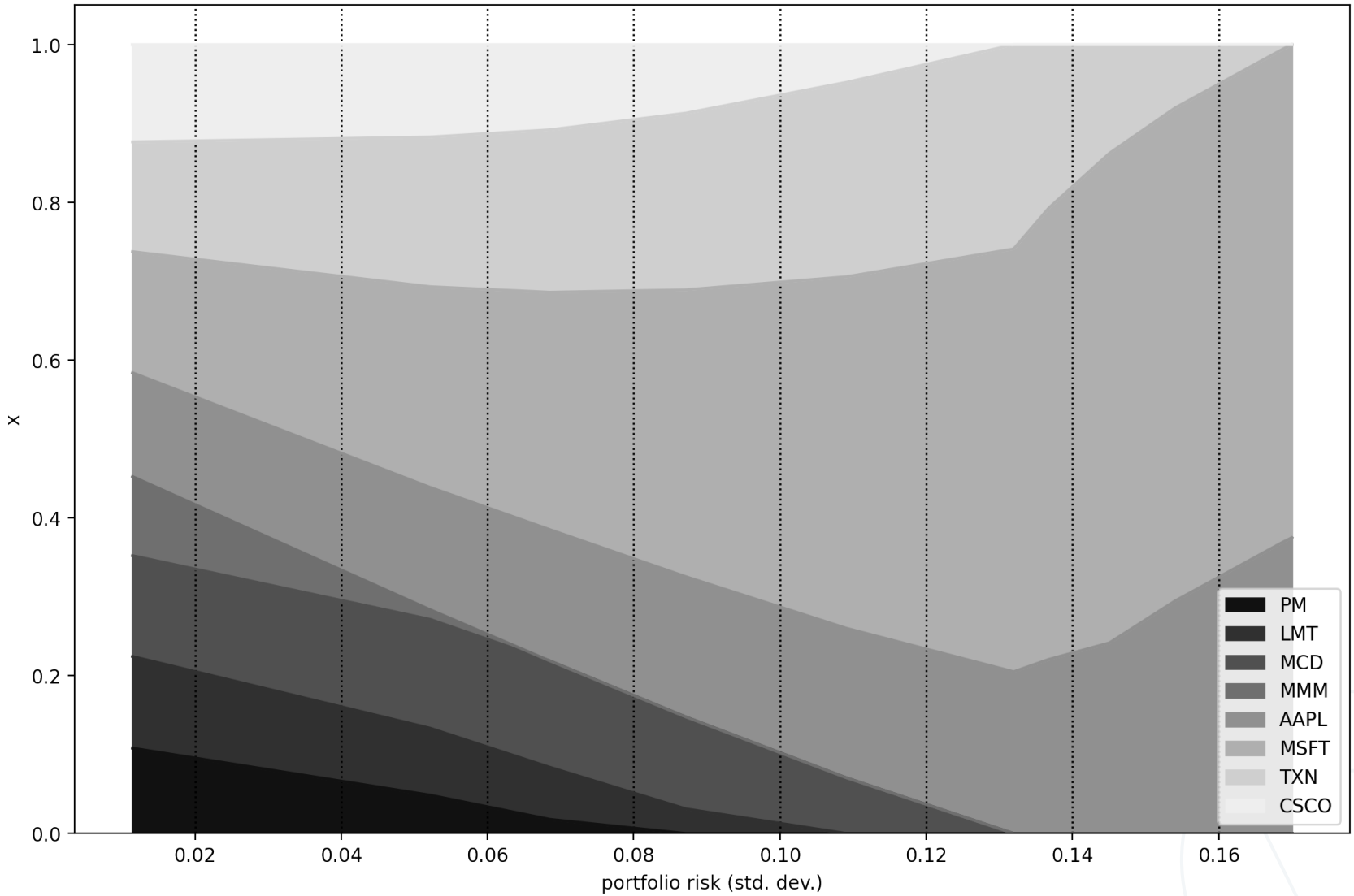
Example: Benchmark relative optimization

The benchmark is the equally weighted portfolio, $\mathbf{x}_{\text{bm}} = \mathbf{1}/N$.
Modifications in the Fusion model of problem (4):

```
# Active holdings
xa = Expr.sub(x, xbm)

# Constraint for the portfolio squared tracking error
M.constraint('risk', Expr.vstack(s, 1, Expr.mul(G.T, xa)),
             Domain.inRotatedQCone())

# Constraint on active holdings and active beta
M.constraint('ub-h', Expr.sub(uh, xa), Domain.greaterThan(0.0))
M.constraint('lb-h', Expr.sub(xa, lh), Domain.greaterThan(0.0))
port_act_beta = Expr.sub(Expr.dot(B, x), 1)
M.constraint('ub-b', Expr.sub(ub, port_act_beta), Domain.greaterThan(0.0))
M.constraint('lb-b', Expr.sub(port_act_beta, lb), Domain.greaterThan(0.0))

# Objective: max portfolio alpha
delta = M.parameter()
M.objective('obj', ObjectiveSense.Maximize,
            Expr.sub(Expr.dot(a, x), Expr.mul(delta, s)))
```

Example: Benchmark relative optimization

The resulting porfolios:



Summary

Takeaway of this section

1. Conic formulation is general so you can model portfolio risk or tracking error the same way.
2. **You** can convert the Mosek Fusion code of an MVO problem into benchmark relative without much effort.

What have we learned

Takeaway of the talk

1. Conic optimization is a **modern framework** allowing the efficient solution of a very diverse set of problems.
2. Mosek is a **general purpose tool** that helps you easily implement and solve conic optimization models.
3. Conic modeling can also be applied to **portfolio optimization**.
4. Mosek Fusion code is a **direct mapping of math** formulas.
5. Factor models can be used to achieve **orders of magnitude** speedup.
6. Adding various **practical constraints** usually requires just a few lines of new code.

Further information

- Mosek `https://mosek.com`
  - Trial and free academic license.
  - Solves linear and conic mixed problems.
  - Interfaces C, Python, Java, Julia, Matlab, R ...
- Documentation at `https://www.mosek.com/documentation/`
  - Modelling cookbook.
  - Portfolio optimization cookbook.
  - Modelling cheat sheet.
  - The MOSEK notebook collection.
- Examples
  - Tutorials at Github: `https://github.com/MOSEK/Tutorials`
  - Distributionally robust optimization notebook : `https://github.com/MOSEK/Tutorials/tree/master/dist-robust-portfolio`