Financial Optimization
ISE 347/447

Lecture 7

Dr. Ted Ralphs

Reading for This Lecture

- C&T Chapter 4

Portfolio Optimization

- We now begin to discuss the *portfolio optimization problem*.
- An investor has a fixed amount of money to invest in a portfolio of n risky assets $S^1, \ldots, S^n$ and a risk-free asset S^0.
- We consider the portfolio's return over a fixed investment period $[0, 1]$.
- The random return of asset i over this period is

$$R_i := \frac{S_1^i - S_0^i}{S_0^i}.$$

- In general, we assume that the vector $\mu = \mathbb{E}[R]$ of expected returns is known.
- Likewise, $Q = \mathrm{Cov}(R)$, the variance-covariance matrix of the return vector R, is also assumed to be known.
- What proportion of wealth should the investor invest in asset i?

Formulating the Portfolio Optimization Problem

Decision variables: x_i, proportion of wealth invested in asset i.

Constraints:

- the entire wealth is assumed invested, $\sum_i x_i = 1$,
- if short-selling of asset i is not allowed, $x_i \geq 0$,
- bounds on exposure to groups of assets, $\sum_{i \in \mathcal{G}} x_i \leq b$, . . .

Objective function: The investor wants to maximize expected return while minimizing risk. What to do?

- Let $R = [R_1 \ \ldots \ R_n]^\top$ be the random vector of asset returns and $\mu = \mathbb{E}[R]$ the vector of their expectations.
- Then the random return of the portfolio y is

$$\frac{\sum_i y_i S_1^i - \sum_i y_i S_0^i}{\sum_i y_i S_0^i} = \sum_i \frac{y_i S_0^i}{\sum_i y_i S_0^i} \cdot \frac{S_1^i - S_0^i}{S_0^i} = R^\top x.$$

The Konno & Yamazaki Model

- The expected portfolio return is

$$\mathbb{E}[R^\top x] = \sum_i x_i \mathbb{E}[R_i] = \mu^\top x.$$

- How do we measure risk?
- Konno & Yamazaki proposed an LP model for portfolio optimization by measuring risk based on the ℓ_1 norm.

$$\ell(x) := \mathbb{E}\left[|(R-\mu)^\top x|\right],$$

- In other words, we consider the mean absolute deviation of the portfolio return from its mean.
- This is a measure of volatility, just like variance.

A Linear Portfolio Optimization Model

- The main motivation for using $\ell(x)$ as a risk measure instead of the variance of the portfolio return

$$\sigma^2(x) := \sigma^2(R^\top x) = \mathbb{E}\left[\left((R-\mu)^\top x\right)^2\right]$$

  is that the resulting model is linear rather than quadratic.

- This allows us to handle a much larger number of assets.

- If $R \sim \mathbb{N}(\mu, Q)$ is a multivariate normal random vector with covariance matrix Q, then $\sigma^2(x) = x^\top Q x$ and

$$\ell(x) = \frac{1}{\sqrt{2\pi\sigma^2(x)}} \int_{-\infty}^{+\infty} |\vartheta| \exp\left(-\frac{\vartheta^2}{2\sigma^2(x)}\right) d\vartheta = \sqrt{\frac{2\sigma^2(x)}{\pi}}.$$

- Thus, minimizing $\sigma^2(x)$ is the same as minimising $\ell(x)$ in this case.

Trading Off Risk and Return

- To set up an optimization model, we must determine what we are going to optimize.
- Recall that there is a tradeoff between risk and return.
- One approach is to set a target for one and then optimize the other.
- Let $r_{\min} \leq \mu^\top x$ be a minimum target for the expected portfolio return.
- Constraints that limit the extent of short-selling, sector allocation and so forth can also be imposed
- We'll discuss such constraints in more detail later.
- Let's assume the constraints have the form inequalities

$$Ax \geq a, \qquad\qquad Bx = b,$$

where A, B are matrices and a, b vectors of conformable dimensions.

The Konno & Yamazaki Model

- We assume the self-financing constraint $\sum_i x_i = 1$ is taken into account among the constraints $Bx = b$
- The target return constraint $r_{\min} \leq \mu^\top x$ can be modelled among the constraints $Ax \geq a$.
- Subject to these constraints, we now want to minimize the risk,

$$\begin{aligned} \text{(KY)} \quad & \min_x \ell(x) \\ \text{s.t.} \quad & Ax \geq a, \\ & Bx = b. \end{aligned}$$

Estimating Parameters

- If the distribution of R_i is not known, we can replace the expectations appearing in the expression

$$\ell(x) := \mathbb{E}\left[|(R-\mu)^\top x|\right]$$

  with empricial estimates.

- We sample the return R_i of each asset by the returns r_{it} $(t = 1, \ldots, T)$ achieved over T recent nonoverlapping investment periods of equal length.

- We can then replace μ_i and $\ell(x)$ by their empirical estimates

$$\widehat{\mu}_i = \frac{1}{T}\sum_{t=1}^{T} r_{it},$$

$$\widehat{\ell}(x) = \frac{1}{T}\sum_{t=1}^{T}\left|\sum_{i=0}^{n}(r_{it}-\widehat{\mu}_i)x_i\right|.$$

The Revised Model

After these replacements problem (KY) becomes

$$
\begin{aligned}
(\text{KY}') \quad & \min_{x} \frac{1}{T} \sum_{t=1}^{T} \left| \sum_{i=0}^{n} (r_{it} - \widehat{\mu}_i) x_i \right| \\
\text{s.t.} \quad & Ax \geq a, \\
& Bx = b.
\end{aligned}
$$

Note that the objective function of problem (KY') is nonlinear because of the absolute value.

Solving the Revised Model

Theorem 1. *Let* (x^*, y^*, z^*) *be an optimal solution for the linear programming problem*

$$
\begin{aligned}
(KY'') \quad & \min_{x,y,z} \sum_{t=1}^{T} (y_t + z_t) \\
& \text{s.t.} \quad y_t - z_t = \sum_{i=0}^{n} (r_{it} - \widehat{\mu}_i) x_i, \quad (t = 1, \ldots, T) \\
& \qquad Ax \geq a, \\
& \qquad Bx = b, \\
& \qquad y_t, z_t \geq 0, \quad (t = 1, \ldots, T).
\end{aligned}
$$

Then x^* *is an optimal solution for* (KY').

Proof: Clearly, x^* satisfies the constraints of problem (KY′).

Furthermore, for each $(t = 1, \ldots, T)$, we have

$$y_t^* = \begin{cases} |\sum_{i=0}^n (r_{it} - \widehat{\mu}_i) x_i^*| & \text{if } \sum_{i=0}^n (r_{it} - \widehat{\mu}_i) x_i^* \geq 0, \\ 0 & \text{if } \sum_{i=0}^n (r_{it} - \widehat{\mu}_i) x_i^* < 0, \end{cases}$$

$$z_t^* = \begin{cases} 0 & \text{if } \sum_{i=0}^n (r_{it} - \widehat{\mu}_i) x_i^* \geq 0, \\ |\sum_{i=0}^n (r_{it} - \widehat{\mu}_i) x_i^*| & \text{if } \sum_{i=0}^n (r_{it} - \widehat{\mu}_i) x_i^* < 0, \end{cases}$$

for otherwise $y_t^* + z_t^*$ could be reduced while still satisfying the constraints

$$y_t - z_t = \sum_{i=0}^n (r_{it} - \widehat{\mu}_i) x_i,$$

$$y_t, z_t \geq 0,$$

contradicting the optimality of (x^*, y^*, z^*) for (KY″).

Therefore,

$$y_t^* + z_t^* = |\sum_{i=0}^{n}(r_{it} - \widehat{\mu}_i)x_i^*|, \quad (t = 1, \ldots, T),$$

and

$$\sum_{t=1}^{T} y_t^* + z_t^* = \sum_{t=1}^{T} \left| \sum_{i=0}^{n}(r_{it} - \widehat{\mu}_i)x_i^* \right|.$$

This shows that

$$\sum_{t=1}^{T} \left| \sum_{i=0}^{n}(r_{it} - \widehat{\mu}_i)\tilde{x}_i \right| \leq \sum_{t=1}^{T} \left| \sum_{i=0}^{n}(r_{it} - \widehat{\mu}_i)x_i^* \right| \tag{1}$$

for any $\tilde{x}$ that optimizes the problem (KY′).

On the other hand, setting

$$\tilde{y}_t = \begin{cases} |\sum_{i=0}^n (r_{it} - \widehat{\mu}_i)\tilde{x}_i| & \text{if } \sum_{i=0}^n (r_{it} - \widehat{\mu}_i)\tilde{x}_i \geq 0, \\ 0 & \text{if } \sum_{i=0}^n (r_{it} - \widehat{\mu}_i)\tilde{x}_i < 0, \end{cases}$$

$$\tilde{z}_t = \begin{cases} 0 & \text{if } \sum_{i=0}^n (r_{it} - \widehat{\mu}_i)\tilde{x}_i \geq 0, \\ |\sum_{i=0}^n (r_{it} - \widehat{\mu}_i)\tilde{x}_i| & \text{if } \sum_{i=0}^n (r_{it} - \widehat{\mu}_i)\tilde{x}_i < 0, \end{cases}$$

we find that $(\tilde{x}, \tilde{y}, \tilde{z})$ is feasible for (KY″) and

$$\sum_{t=1}^T \left| \sum_{i=0}^n (r_{it} - \widehat{\mu}_i)\tilde{x}_i \right| = \sum_{t=1}^T \tilde{y}_t + \tilde{z}_t \geq \sum_{t=1}^T y_t^* + z_t^* = \sum_{t=1}^T \left| \sum_{i=0}^n (r_{it} - \widehat{\mu}_i)x_i^* \right|.$$

Together with (1) this shows

$$\frac{1}{T} \left| \sum_{i=0}^n (r_{it} - \widehat{\mu}_i)\tilde{x}_i \right| = \frac{1}{T} \left| \sum_{i=0}^n (r_{it} - \widehat{\mu}_i)x_i^* \right|$$

Another Python-based Modeling Language: Pyomo

- An algebraic modeling language in Python similar to PuLP.
- Can import data from many sources, including AMPL data files.
- More powerful, includes support for nonlinear modeling.
- Allows development of both concrete models (like PuLP) and abstract models (like AMPL).
- Also include PySP for Stochastic Programming.
- Primary classes
  - `ConcreteModel`, `AbstractModel`
  - `Set`, `Parameter`
  - `Var`, `Constraint`

Pyomo Basics: Concrete Dedication Model

```
model = ConcreteModel()

Bonds, Features, BData, Liabilities = read_data('ded.dat')

Periods = range(len(Liabilities))

model.buy = Var(Bonds, within=NonNegativeReals)
model.cash = Var(Periods, within=NonNegativeReals)
model.obj = Objective(expr=model.cash[0] +
            sum(BondData[b, 'Price']*model.buy[b] for b in Bonds),
            sense=minimize)

def cash_balance_rule(model, t):
    return (model.cash[t-1] - model.cash[t]
            +sum(BData[b, 'Coupon']*model.buy[b] for b in Bonds if BData[b, 'Maturity']>=t)
            +sum(BData[b, 'Principal']*model.buy[b] for b in Bonds if BData[b, 'Maturity']==t)
             == Liabilities[t])
model.cash_balance = Constraint(Periods[1:], rule=cash_balance_rule)

opt = SolverFactory("cbc")
instance = model.create()
results = opt.solve(instance)
instance.load(results)
print "Optimal strategy"
for b in instance.buy:
    if instance.buy[b].value > epsilon:
        print 'Buy \%f of Bond \%s' \%(instance.buy[b].value, b)
```

Pyomo Basics: Abstract Dedication Model

```
model = AbstractModel()

model.Periods = Set()
model.Bonds = Set()
model.Price = Param(model.Bonds)
model.Maturity = Param(model.Bonds)
model.Coupon = Param(model.Bonds)
model.Principal = Param(model.Bonds)
model.Liabilities = Param(range(9))

model.buy = Var(model.Bonds, within=NonNegativeReals)
model.cash = Var(range(9), within=NonNegativeReals)

def objective_rule(model):
    return model.cash[0] + sum(model.Price[b]*model.buy[b] for b in model.Bonds)
model.objective = Objective(sense=minimize, rule=objective_rule)

def cash_balance_rule(model, t):
    return (model.cash[t-1] - model.cash[t]
            +sum(BData[b, 'Coupon']*model.buy[b] for b in Bonds if BData[b, 'Maturity']>=t)
            +sum(BData[b, 'Principal']*model.buy[b] for b in Bonds if BData[b, 'Maturity']==t)
             == Liabilities[t])
model.cash_balance = Constraint(range(1, 9), rule=cash_balance_rule)
```

Pyomo Basics: Importing Data and Solving the Pyomo Model

```
epsilon = .001

opt = SolverFactory("cbc")
instance = model.create('ded.dat')
results = opt.solve(instance)
results.write()
instance.load(results)

print "Optimal strategy"
for b in instance.buy:
    if instance.buy[b].value > epsilon:
        print 'Buy %f of Bond %s' %(instance.buy[b].value, b)
```

Pyomo Portfolio Model: Konno and Yamazaki

```
model.assets = Set()
model.T = Set(initialize = range(1994, 2014))
model.max_risk = Param(mutable = True, initialize = .00305)
model.R = Param(model.T, model.assets)

def mean_init(model, j):
    return sum(model.R[i, j] for i in model.T)/len(model.T)
model.mean = Param(model.assets, initialize = mean_init)

def Q_init(model, i, j):
    return sum((model.R[k, i] - model.mean[i])*(model.R[k, j] - model.mean[j]) for k in model.T)
model.Q = Param(model.assets, model.assets, initialize = Q_init)

model.alloc = Var(model.assets, within=NonNegativeReals)

def risk_bound_rule(model):
    return (
        sum(sum(model.Q[i, j] * model.alloc[i] * model.alloc[j] for i in model.assets)
            for j in model.assets) <= model.max_risk)
model.risk_bound = Constraint(rule=risk_bound_rule)

def tot_mass_rule(model):
    return (sum(model.alloc[j] for j in model.assets) == 1)
model.tot_mass = Constraint(rule=tot_mass_rule)

def objective_rule(model):
    return summation(model.mean, model.alloc)
model.objective = Objective(sense=maximize, rule=objective_rule)
```

Show me the Data

- We now have an abstract model, but where does the data come from?
- In practice, models are easy to come by.
- Data that is "real" and "clean" is the key to good financial decision-making.
- We may need both historical data and predicted future data.
- Getting this data can be the most difficult and expensive part of implementing these models.
- We will do it on the cheap: Yahoo/Google Finance!
  - There are many Python packages for pulling financial data.
  - An internet search will turn up any number of viable alternatives.

Pulling Data with Python

```
from yahoo import Quote, YahooQuote

stocks = ['AA', 'AXP', 'BA', 'BAC', 'CAT', 'CSCO', 'CVX', 'DD', 'DIS', 'GE', 'HD', 'HPQ']
stocks += ['IBM', 'INTC', 'JNJ', 'JPM', 'KO', 'MCD', 'MMM', 'MRK', 'MSFT', 'PFE', 'PG']
stocks += ['T', 'TRV', 'UNH', 'UTX', 'VZ', 'WMT', 'XOM']

price = {}
quotes = {}
returns = {}
for s in stocks:
    print 'Stock', s
    for year in range(1993, 2015):
        try:
            quotes[year, s] = YahooQuote(s,'%s-01-01'%(str(year)), '%s-01-08'%(str(year)))
        except ValueError:
            pass
        for q in str(quotes[year, s]).split('\n'):
            if q.split(',')[0] == s:
                price[year, s] = float(q.split(',')[5])
                break

for s in stocks:
    for year in range(1994, 2015):
        returns[year, s] = price[year, s]/price[year -1, s]
```

Results with DJIA Data

```
Creating model...
Optimizing with minimum return 1.15
Optimal purchase amounts:
CVX                0.275
DD                 0.031
CAT                0.009
BAC                0.078
MSFT               0.013
IBM                0.081
PG                 0.275
WMT                0.030
UNH                0.207
Optimal risk: 1.349
```

Using Biojective Optimization

- Rather than choosing just one value of $r_{\min}$, we can try to generate a range of solutions for different values of $r_{\min}$.
- We can then analyze the tradeoff between return and risk directly.
- We would like to know how much we will have to give up in terms of decreased return in order to reduce the risk by a given amount.
- We can do this by posing (KY') as a *biobjective linear program*.

Biobjective Mathematical Programs

A *biobjective* or *bicriterion mathematical program* is an optimization problem of the form

$$\begin{array}{ll} \text{vmin} & f(x) \\ \text{subject to} & x \in X, \end{array}$$

where

- $f : \mathbb{R}^n \to \mathbb{R}^2$ is the *(bicriterion) objective function*, and
- X is the *feasible region*, usually defined to be

$$\{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \mid g_i(x) \geq 0, i = 1, \ldots, m\}$$

for functions $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$.

The vmin operator indicates that we are interested in generating the *efficient solutions* (defined next).

Some Definitions

- $x^1 \in X$ *dominates* $x^2 \in X$ if $f_i(x_1) \leq f_i(x_2)$ for $i = 1, 2$ and at least one inequality is strict.
- If both inequalities are strict the dominance is *strong* (otherwise *weak*).
- Any $x \in X$ not dominated by any other member of X is said to be *efficient*.
- The set of *outcomes* is defined to be $Y = f(X) \subset \mathbb{R}^2$.
- In outcome space, BMIP can be restated as

$$\begin{array}{ll} \text{vmin} & y \\ \text{subject to} & y \in f(X), \end{array}$$

- If $x \in X$ is efficient, then $y = f(x)$ is *Pareto*.
- For simplicity, we work in outcome space.
- Our goal is to generate the set of all Pareto outcomes.

Plotting Risk and Return (KY Model)

```
instance = model.create('DJIA.dat')
opt = SolverFactory("cbc")
#return_values = [1.17]
return_values = [1.15 + .01*i for i in range(14)]
risk_values = []
for r in return_values:
    instance.min_return = r

    print 'Optimizing with minimum return', r

    results = opt.solve(instance)
    instance.load(results)

    print 'Optimal purchase amounts:'
    for i in instance.assets:
        if instance.alloc[i].value > epsilon:
            print '%-15s %-3.3f' % (i, instance.alloc[i].value)

    print 'Optimal return: %.3f' % (value(instance.objective))
    risk_values.append(value(instance.objective))

plt.plot(risk_values, return_values, 'bs')
plt.show()
```

Plotting Risk and Return (KY Model)

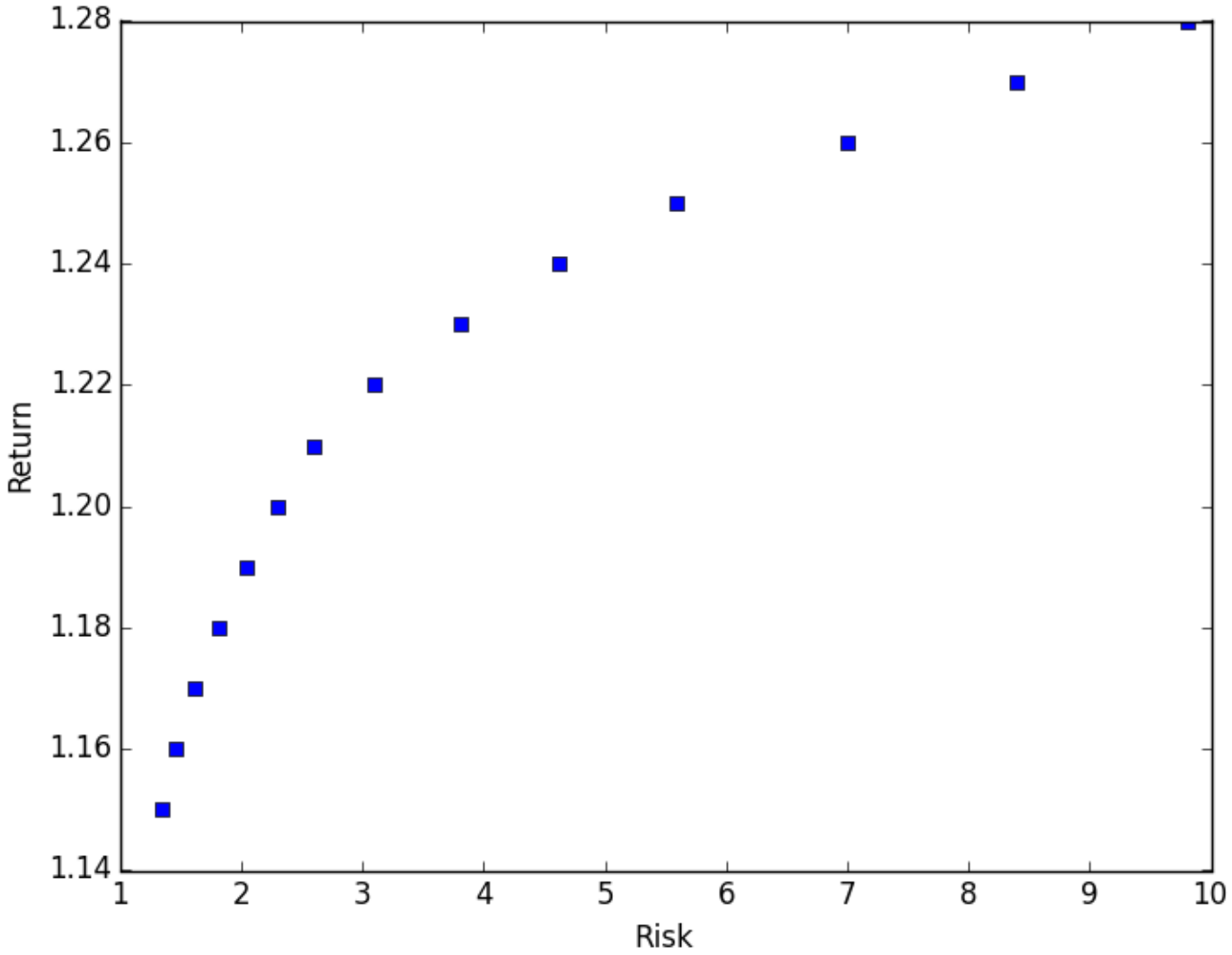

Figure 1: Approximate frontier of risk versus return in KY model

Probing Algorithms

- A wide array of algorithms for generating Pareto outcomes for mathematical programs have been proposed.
- The easiest to understand and implement are *probing algorithms* that *scalarize* the objective, i.e., replace it with a single criterion.
- Such algorithms attempt to reduce solution of a biobjective math programs to a series of single-objective math programs.
- The most obvious scalarization is the *weighted sum objective*.
- We replace the original objective with

$$\max_{y \in f(X)} \beta y_1 + (1 - \beta) y_2$$

to obtain a parameterized family of mathematical programs.

Probing Algorithms

- For convex programs, it is easy to show that any Pareto outcome can be genrated as the solution to some member of the above family.
- In particular, for linear programs, we can therefore generate all Pareto outcome just by solving a sequence of LPs.
- Can we do this efficiently?

Parametric Linear Programming

- For a fixed matrix A, vectors b and c, and direction d, consider the problem

$$\begin{aligned} min\ & (c+\theta d)^\top x \\ s.t.\quad & Ax = b \\ & x \geq 0 \end{aligned}$$

- If $g(\theta)$ is the optimal cost for a given θ, then

$$g(\theta) = \min_{i=1,\ldots,N} (c+\theta d)^\top x^i$$

where $x^1, \ldots, x^N$ are the extreme points of the feasible set.

- Hence, $g(\theta)$ is piecewise linear and concave.
- There is a systematic method for obtaining $g(\theta)$ for all θ.

The Basic Idea

- For a fixed basis B and a fixed θ, the reduced costs are linear functions of θ.
- Hence, we can easily determine the range of values of θ for which the current basis is optimal.
- The ends of this interval determine the nearest breakpoints of $g(\theta)$.
- By changing θ to the next breakpoint and performing a change of basis, we obtain a new interval and a new breakpoint.
- This process can be continued to obtain all breakpoints.

The Parametric Simplex Method

- Determine an initial feasible basis.
- Determine the interval $[\theta_1, \theta_2]$ for which this basis is optimal.
- Determine a variable j whose reduced cost is nonpositive for $\theta \geq \theta_2$.
- If the corresponding column has no positive entries, then the problem is unbounded for $\theta > \theta_2$.
- Otherwise, rotate column j into the basis.
- Determine a new interval $[\theta_2, \theta_3]$ in which the current basis is optimal.
- Iterate to find all breakpoint $\geq \theta_1$.
- Repeat the process to find breakpoints $\leq \theta_1$.

Comments on Parametric Simplex Method

- Note that as long as each interval has a positive length, this process will terminate finitely.
- If some interval has zero length, cycling is possible.
- We can prevent cycling with anticycling rules as in the regular simplex algorithm.
- We can perform the same analysis for a parametrically defined right hand side using dual simplex.

A Parametric Verion of Konno & Yamazaki

If we replace the original LP (KY′) with the parametric program

$$
\begin{aligned}
(\mathsf{KY}'') \quad \max_{x,y,z} \ & \widehat{\mu}^\top x - \theta \sum_{t=1}^{T} (y_t + z_t) \\
\text{s.t.} \quad & y_t - z_t = \sum_{i=0}^{n} (r_{it} - \widehat{\mu}_i) x_i, \quad (t = 1, \ldots, T) \\
& Ax \geq a, \\
& Bx = b, \\
& y_t, z_t \geq 0, \quad (t = 1, \ldots, T),
\end{aligned}
$$

we can now analzye the tradeoff between the two objectives and decide on a solution for which the tradeoff is acceptable.