Computational Integer Programming

Lecture 6: Algebraic Modeling (Part II)

Dr. Ted Ralphs

LEHIGH
UNIVERSITY

COR@L
COMPUTATIONAL OPTIMIZATION
RESEARCH AT LEHIGH

Example: Constructing an Index Fund

- An index is essentially a proxy for the entire universe of investments.
- An index fund is, in turn, a proxy for an index.
- A fundamental question is how to construct an index fund.
- It is not practical to simply invest in exactly the same basket of investments as the index tracks.
  - The portfolio will generally consist of a large number of assets with small associated positions.
  - Rebalancing costs may be prohibitive.
- A better approach may be to select a small subset of the entire universe of stocks that we predict will closely track the index.
- This is what index funds actually do in practice.

A Deterministic Model

- The model we now present attempts to cluster the stocks into groups that are "similar."
- Then one stock is chosen as the representative of each cluster.
- The input data consists of parameters ρ_{ij} that indicate the similarity of each pair (i, j) of stocks in the market.
- One could simply use the correlation coefficient as the similarity parameter, but there are also other possibilities.
- This approach is not guaranteed to produce an efficient portfolio, but should track the index, in principle.

An Integer Programming Model

- We have the following variables:
  - y_j is stock j is selected, 0 otherwise.
  - x_{ij} is 1 if stock i is in the cluster represented by stock j, 0 otherwise.
- The objective is to maximize the total similarity of all stocks to their representatives.
- We require that each stock be assigned to exactly one cluster and that the total number of clusters be q.

An Integer Programming Model

Putting it all together, we get the following formulation

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{j=1}^{n} y_j = q \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{3}$$

$$x_{ij} \leq y_j \qquad \forall i = 1, \ldots, n, j = 1, \ldots, n \tag{4}$$

$$x_{ij}, y_j \in \{0, 1\} \qquad \forall i = 1, \ldots, n, j = 1, \ldots, n \tag{5}$$

Constructing an Index Portfolio (IndexFund-Pyomo.py)

```
model.K = Param()
model.assets = Set()
model.T = Set(initialize = range(1994, 2014))
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.rep    = Var(model.assets, model.assets,
                   within=NonNegativeIntegers)
model.select = Var(model.assets,
                   within=NonNegativeIntegers)
```

Pyomo Model for Constructing an Index Portfolio (cont'd)

```
def representation_rule(model, i):
    return (sum(model.rep[i, j] for j in model.assets) == 1)
model.representation = Constraint(model.assets,
                                  rule=representation_rule)

def selection_rule(model, i, j):
    return (model.rep[i, j] <= model.select[j])
model.selection = Constraint(model.assets, model.assets,
                             rue=selection_rule)

def cardinality_rule(model):
    return (summation(model.select) == model.K)
model.cardinality = Constraint(rule=cardinality_rule)

def objective_rule(model):
    return sum(model.Q[i, j]*model.rep[i, j]
               for i in model.assets for j in model.assets)
model.objective = Objective(sense=maximize, ruke=objective_rule)
```

Interpreting the Solution

- As before, we let $\hat{w}$ be the relative market-capitalized weights of the selected stocks

$$\hat{w}_i = \frac{\sum_{j=1}^{n} z_i S^i x_{ij}}{\sum_{i=0}^{n} \sum_{j=1}^{n} z_i S^i x_{ij}},$$

  where z_i is the number of shares of asset i that exist in the market and S^i the value of each share.

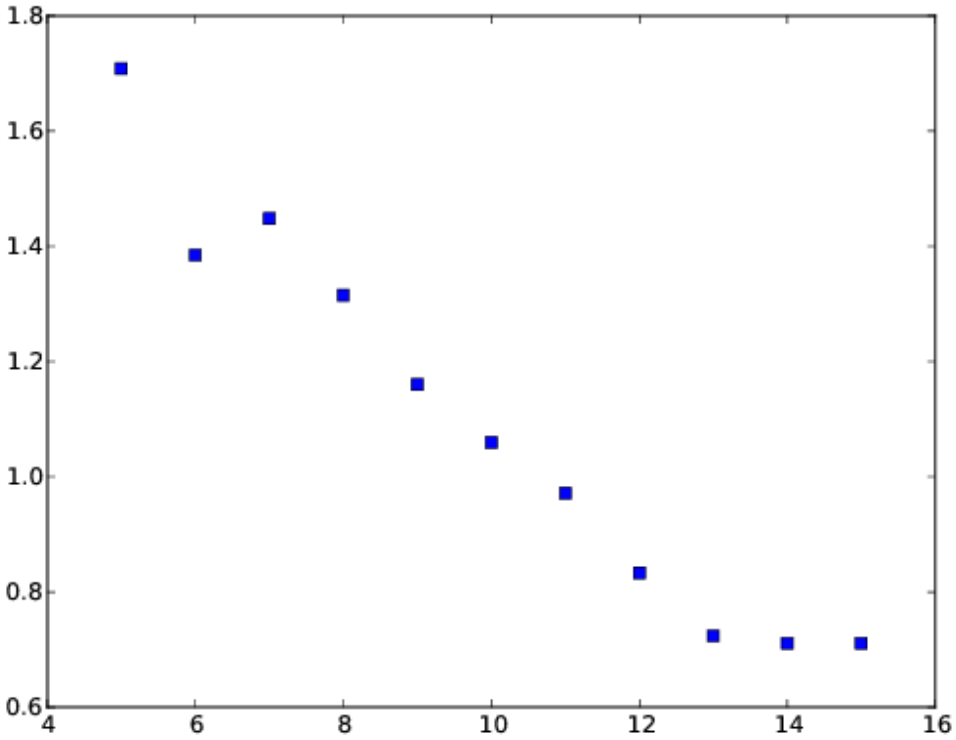
- This portfolio is what we now use to track the index.

- Note that we could also have weighted the objective by the market capitalization in the original model:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} z_i S^i \rho_{ij} x_{ij}$$

Effect of K on Performance of Index Fund

- This is a chart showing how the performance of the index changes as it's size is increased.
- This is for an equal-weighted index and the performance metric is sum of squared deviations.



Example: Facility Location Problem

- We have n locations and m customers to be served from those locations.
- There is a fixed cost c_j and a capacity W_j associated with facility j.
- There is a cost d_{ij} and demand w_{ij} for serving customer i from facility j.
- We have two sets of binary variables.
  - y_j is 1 if facility j is opened, 0 otherwise.
  - x_{ij} is 1 if customer i is served by facility j, 0 otherwise.

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} & \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 & \forall i \\
& \sum_{i=1}^{m} w_{ij} x_{ij} \leq W_j & \forall j \\
& x_{ij} \leq y_j & \forall i, j \\
& x_{ij}, y_j \in \{0, 1\} & \forall i, j
\end{aligned}
$$

PuLP: Facility Location Example

```
from products   import REQUIREMENT, PRODUCTS
from facilities import FIXED_CHARGE, LOCATIONS, CAPACITY

prob = LpProblem("Facility_Location")
ASSIGNMENTS = [(i, j) for i in LOCATIONS for j in PRODUCTS]
assign_vars = LpVariable.dicts("x", ASSIGNMENTS, 0, 1, LpBinary)
use_vars    = LpVariable.dicts("y", LOCATIONS, 0, 1, LpBinary)

prob += lpSum(use_vars[i] * FIXED_COST[i] for i in LOCATIONS)

for j in PRODUCTS:
    prob += lpSum(assign_vars[(i, j)] for i in LOCATIONS) == 1

for i in LOCATIONS:
    prob += lpSum(assign_vars[(i, j)] * REQUIREMENT[j]
                  for j in PRODUCTS) <= CAPACITY * use_vars[i]

prob.solve()

for i in LOCATIONS:
    if use_vars[i].varValue > 0:
        print "Location ", i, " is assigned: ",
        print [j for j in PRODUCTS if assign_vars[(i, j)].varValue > 0]
```

PuLP Basics: Facility Location Example

```
# The requirements for the products
REQUIREMENT = {
    1 : 7,
    2 : 5,
    3 : 3,
    4 : 2,
    5 : 2,
}
# Set of all products
PRODUCTS = REQUIREMENT.keys()
PRODUCTS.sort()
# Costs of the facilities
FIXED_COST = {
    1 : 10,
    2 : 20,
    3 : 16,
    4 : 1,
    5 : 2,
}
# Set of facilities
LOCATIONS = FIXED_COST.keys()
LOCATIONS.sort()
CAPACITY = 8
```

Tradeoff Analysis (Multiobjective Optimization)

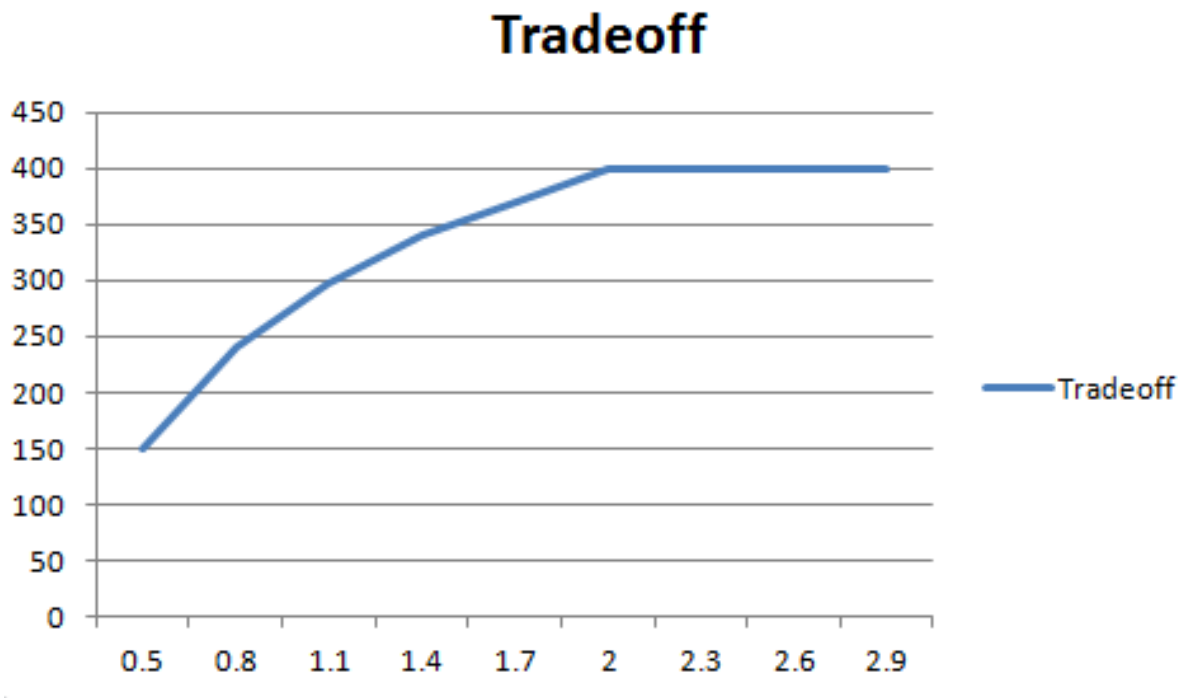
Analysis with Multiple Objectives

- In many cases, we are trying to optimize multiple criteria simultaneously.
- These criteria often conflict (risk versus reward).
- Often, we deal with this by placing a constraint on one objective while optimizing the other.
- Extending the principles from the sensitivity analysis section, we can consider a doing a *parametric analysis*.
- We do this by varying the right-hand side systematically and determining how the objective function changes as a result.
- More generally, we ma want to find all *non-dominated* solutions with respect to two or more objectives functions.
- This latter analysis is called *multiobjective optimization*.

Parametric Analysis with PuLP

(`FinancialModels.xlsx:Bonds-Tradeoff-PuLP`)

- Suppose we wish to analyze the tradeoff between yield and rating in our bond portfolio.
- By iteratively changing the value of the right-hand side of the constraint on the rating, we can create a graph of the tradeoff.



Parametric Analysis with PuLP

```
for r in range_vals:
    if sense[what_to_range] == '<':
         prob.constraints[what_to_range].constant = -max_cash*r
    else:
         prob.constraints[what_to_range].constant = max_cash*r

    status = prob.solve()

    epsilon = .001

    if LpStatus[status] == 'Optimal':
        obj_values[r] = value(prob.objective)
    else:
         print 'Problem is', LpStatus[status]
```

Nonlinear Modeling

Portfolio Optimization

- 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

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

- 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: In general, 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

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

Trading Off Risk and Return

- To set up an optimization model, we must determine what our measure of "risk" will be.
- The goal is to analyze the tradeoff between risk and return.
- One approach is to set a target for one and then optimize the other.
- The classical portfolio model of Henry Markowitz is based on using the variance of the portfolio return as a risk measure:

$$\sigma^2(R^\top x) = x^\top Q x,$$

  where $Q = \mathrm{Cov}(R_i, R_j)$ is the variance-covariance matrix of the vector of returns R.

- We consider three different single-objective models that can be used to analyze the tradeoff between these conflicting goals.

Three Markowitz Models

$$\begin{aligned} \text{(M1)} \quad & \min_{x \in \mathcal{R}^n} \; x^\top Q x \\ & \text{s.t.} \quad \mu^\top x \geq r, \\ & \qquad \sum_{i=1}^{n} x_i = 1, \end{aligned}$$

where r is a targeted minimum expected portfolio return.

$$\begin{aligned} \text{(M2)} \quad & \max_{x \in \mathcal{R}^n} \; \mu^\top x \\ & \text{s.t.} \quad x^\top Q x \leq \sigma^2 \\ & \qquad \sum_{i=1}^{n} x_i = 1, \end{aligned}$$

where σ^2 is the maximum risk the investor is willing to take on.

Three Markowitz Models (cont.)

$$\text{(M3)} \quad \max_{x \in \mathcal{R}^n} \ \mu^\top x - \lambda x^\top Q x$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_i = 1,$$

where $\lambda > 0$ is a risk-aversion parameter.

- All three models are examples of *quadratic optimization problems*,
- Also, since Q is a positive semidefinite symmetric matrix, then $x \mapsto x^\top Q x$ is a convex function.
- Hence, these are actually *convex quadratic programs.*
- Convex quadratic programs can generally be solved efficiently.

Modeling Nonlinear Programs

- Both AMPL and Pyomo support the inclusion of nonlinear functions in the model.
- In both cases, a wide range of built-in functions are available.
- By restricting the form of the nonlinear functions, we ensure that the Hessian can be easily calculated.
- The solvers `ipopt`, `bonmin`, and `couenne` can be used to solve the models.
- See
  - `portfolio-*.mod`,
  - `portfolio-*-Pyomo.py`,
  - `FinancialModels.xlsx:Portfolio-AMPL`, and
  - `FinancialModels.xlsx:Portfolio-Pyomo`.

AMPL model for Portfolio Optimization (portfolio-iid.mod)

```
set assets;                          # asset categories
set T := {1984..1994};               # years

param max_risk default 0.00305;
param R {T,assets};
param mean {j in assets} := (sum{i in T} R[i,j])/card(T);
param Rtilde {i in T, j in assets} := R[i,j] - mean[j];
param Q {i in assets, j in assets} := sum{k in T}
(R[k, i] - model.mean[i])*(model.R[k, j] - model.mean[j]);

var alloc{assets} >=0;

minimize reward: - sum{j in assets} mean[j]*alloc[j] ;

subject to risk_bound: sum{i in assets}
(sum{j in assets} Q[i,j]*alloc[i]*alloc[j]) <= max_risk;
subject to tot_mass: sum{j in assets} alloc[j] = 1;
```

Pyomo model for Portfolio Optimization (portfolio-Pyomo.py)

```
model = AbstractModel()

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

Pyomo model for Portfolio Optimization (cont'd)

```
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 sum(model.alloc[j]*model.mean[j] for j in model.assets)
model.objective = Objective(sense=maximize, rule=objective_rule)
```

Getting the Data

- One of the most compelling reasons to use Python for modeling is that there are a wealth of tools available.
- Historical stock data can be easily obtained from Yahoo using built-in Internet protocols.
- Here, we use a small Python package for getting Yahoo quotes to get the price of a set of stocks at the beginning of each year in a range.
- See `FinancialModels.xlsx:Portfolio-Pyomo-Live`.

```
for s in stocks:
    for year in range(1993, 2014):
        quote[year, s] = YahooQuote(s,'%s-01-01'%(str(year)),
                                    '%s-01-08'%(str(year)))
        price[year, s] = float(quote[year, s].split(',')[5])
            break
```

The Efficient Frontier

- We can assume without loss of generality that $Q \succ 0$, so we have $\sigma_{\min} > 0$, where

$$\begin{aligned} \sigma_{\min}^2 := \min_x \; & x^\top Q x \\ \text{s.t.} \quad & \mu^\top x \geq r, \\ & \sum_{i=1}^n x_i = 1, \end{aligned}$$

- Let

$$\begin{aligned} \text{(R)} \quad r(\sigma) = \max_x \; & \mu^\top x \\ \text{s.t.} \quad & Ax \geq a \\ & Bx = b \\ & x^\top Q x \leq \sigma^2, \end{aligned}$$

and note that for $\sigma \geq \sigma_{\min}$ the function $r(\sigma)$ is well-defined.

The Efficient Frontier

Note that $\mu^\top x \leq r(\sqrt{x^\top Q x})$ for all feasible x, and that it can never make sense to hold a portfolio x for which

$$\mu^\top x < r\left(\sqrt{x^\top Q x}\right),$$

since the portfolio x^* obtained from solving problem (R) with $\sigma^2 = x^\top Q x$ would yield the more desirable expected return

$$\mu^\top x^* = r\left(\sqrt{x^\top Q x}\right).$$

Definition 1. *Portfolios that satisfy the relation*

$$\mu^\top x = r\left(\sqrt{x^\top Q x}\right)$$

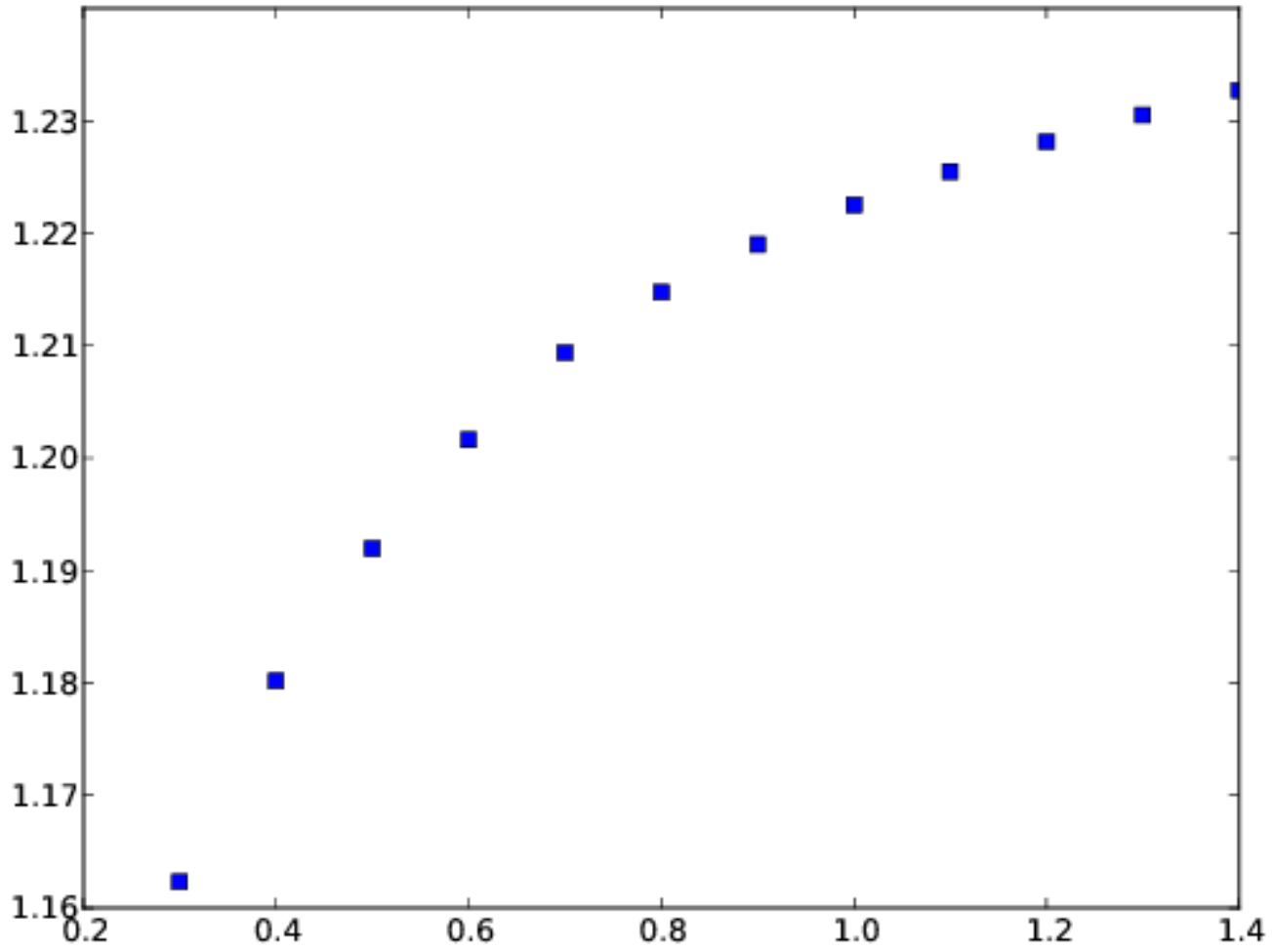
are called efficient. *The curve* $\sigma \mapsto r(\sigma)$, *defined for* $\sigma \geq \sigma_{\min}$, *is called the* efficient frontier.

Constructing the Efficient Frontier

- Since portfolio optimization is a convex program, we can show that the efficient frontier is convex.
- By sampling, we can construct it.

```
opt = SolverFactory("ipopt")
risk_values = [float(i)/10 for i in range(3, 15)]
returns = []
for risk in risk_values:
    instance = model.create('DJIA.dat')
    instance.max_risk.value = risk
    results = opt.solve(instance)
    instance.load(results)
    print 'Optimal return: %.3f' % (value(instance.objective))
    returns.append(value(instance.objective))
plt.plot(risk_values, returns, 'bs')
plt.show()
```

Efficient Frontier for the DJIA Data Set



Exercise: Knapsack Problem

Develop a Model for the Knapsack Problem in as many AMLs as you can!