The COIN-OR Optimization Suite:

Advanced Modeling

Ted Ralphs

COIN fORgery: Developing Open Source Tools for OR

Institute for Mathematics and Its Applications, Minneapolis, MN

Outline

Outline

1. Sensitivity Analysis
2. Tradeoff Analysis (Multiobjective Optimization)
3. Nonlinear Modeling
4. Integer Programming
5. Stochastic Programming

Marginal Price of Constraints

- The dual prices, or *marginal prices* allow us to put a value on "resources" (broadly construed).
- Alternatively, they allow us to consider the sensitivity of the optimal solution value to changes in the input.
- Consider the bond portfolio problem.
- By examining the dual variable for the each constraint, we can determine the value of an extra unit of the corresponding "resource".
- We can then determine the maximum amount we would be willing to pay to have a unit of that resource.
- The so-called "reduced costs" of the variables are the marginal prices associated with the bound constraints.

Marginal Prices in AMPL

Again, recall the simple bond portfolio model from Lecture 3.

```
ampl: model bonds.mod;
ampl: solve;
...
ampl: display rating_limit, cash_limit;
rating_limit = 1
cash_limit = 2
```

- This tells us that the optimal marginal cost of the `rating_limit` constraint is 1.
- What does this tell us about the "cost" of improving the average rating?
- What is the return on an extra $\$1K$ of cash available to invest?

Another Interpretation of Marginal Prices

- Let's consider again the prices for the constraints in the simple bond portfolio model.
- By combining the two constraints with nonzero prices, we can get a third inequality that must be satisfied by any feasible solution:

$$\begin{array}{rl} 2\,[x_1 + x_2 \leq 100] & + \\ 1\,[2x_1 + x_2 \leq 150] & = \\ 4x_1 + 3x_2 & \leq 350 \end{array}$$

- What does this tell us about the optimal solution value?

Economic Interpretation of Optimality

Example: A simple product mix problem.

```
ampl: var X1;
ampl: var X2;
ampl: maximize profit: 3*X1 + 3*X2;
ampl: subject to hours: 3*X1 + 4*X2 <= 120000;
ampl: subject to cash: 3*X1 + 2*X2 <= 90000;
ampl: subject to X1_limit: X1 >= 0;
ampl: subject to X2_limit: X2 >= 0;
ampl: solve;
...
ampl: display X1;
X1 = 20000
ampl: display X2;
X2 = 15000
```

Shadow Prices in Product Mix Model

```
ampl: model simple.mod
ampl: solve;
...
ampl: display hours, cash;
hours = 0.5
cash = 0.5
```

- This tells us that increasing the hours by 2000 will increase profit by $(2000)(0.5) = \$1000$.
- Hence, we should be willing to pay up to $.50/hour for additional labor hours (as long as the solution remains feasible).
- We can also see that the availability of cash and man hours are contributing equally to the cost of each product.

Economic Interpretation of Optimality

- In the preceding example, we can use the shadow prices to determine how much each product “costs” in terms of its constituent “resources.”
- The reduced cost of a product is the difference between its selling price and the (implicit) cost of the constituent resources.
- If we discover a product whose “cost” is less than its selling price, we try to manufacture more of that product to increase profit.
- With the new product mix, the demand for various resources is changed and their prices are adjusted.
- We continue until there is no product with cost less than its selling price.
- This is the same as having the reduced costs nonpositive (recall this was a maximization problem).
- Complementary slackness says that we should only manufacture products for which cost and selling price are equal.
- This can be viewed as a sort of multi-round auction.

AMPL: Displaying Auxiliary Values with Suffixes

- In AMPL, it's possible to display much of the auxiliary information needed for sensitivity using suffixes.
- For example, to display the reduced cost of a variable, type the variable name with the suffix *.rc*.
- Recall again the short term financing example (`short_term_financing.mod`).

```
ampl: display credit.rc;
credit.rc [*] :=
 0  -0.003212
 1   0
 2  -0.0071195
 3  -0.00315
 4   0
 5   0
;
```

- How do we interpret this?

AMPL: Sensitivity Ranges

- AMPL does not have built-in sensitivity analysis commands.
- AMPL/CPLEX does provide such capability, however.
- To get sensitivity information, type the following

```
ampl: option cplex_options 'sensitivity';
```

- Solve the bond portfolio model:

```
ampl: solve;
...
suffix up OUT;
suffix down OUT;
suffix current OUT;
```

AMPL: Accessing Sensitivity Information

Access sensitivity information using the suffixes *.up* and *.down*. This is from the model `bonds.mod`.

```
ampl: display cash_limit.up, rating_limit.up, maturity_limit.up;
cash_limit.up = 102
rating_limit.up = 200
maturity_limit.up = 1e+20

ampl: display cash_limit.down, rating_limit.down, maturity_limit.down;
cash_limit.down = 75
rating_limit.down = 140
maturity_limit.down = 350

ampl: display buy.up, buy.down;
: buy.up buy.down    :=
A    6       3
B    4       2
;
```

AMPL: Sensitivity for the Short Term Financing Model

```
ampl: short_term_financing.mod;
ampl: short_term_financing.dat;
ampl: solve;
ampl: display credit, credit.rc, credit.up, credit.down;
:    credit      credit.rc     credit.up  credit.down    :=
0     0         -0.00321386   0.00321386    -1e+20
1    50.9804     0            0.00318204         0
2     0         -0.00711864   0.00711864    -1e+20
3     0         -0.00315085   0.00315085    -1e+20
4     0          0            0             -1e+20
;
```

AMPL: Sensitivity for the Short Term Financing Model (cont.)

```
ampl: display bonds, bonds.rc, bonds.up, bonds.down;
:      bonds       bonds.rc     bonds.up     bonds.down      :=
0    150            0          0.00399754   -0.00321386
1     49.0196       0          0            -0.00318204
2    203.434        0          0.00706931    0
3      0            0          0             0
4      0            0          0             0
;
```

AMPL: Sensitivity for the Short Term Financing Model (cont.)

```
ampl: display invest, invest.rc, invest.up, invest.down;
:      invest       invest.rc        invest.up          invest.down
-1      0           0                 0                  0
0       0          -0.00399754        0.00399754     -1e+20
1       0          -0.00714           0.00714        -1e+20
2     351.944       0                 0.00393091         -0.0031603
3       0          -0.00391915        0.00391915     -1e+20
4       0          -0.007             0.007          -1e+20
5      92.4969      0              1e+20                   2.76446e-14
;
```

Sensitivity Analysis of the Dedication Model

Let's look at the sensitivity information in the dedication model

```
ampl: model dedication.mod;
ampl: data dedication.dat;
ampl: solve;
ampl: display cash_balance, cash_balance.up, cash_balance.down;
: cash_balance cash_balance.up cash_balance.down    :=
1    0.971429       1e+20          5475.71
2    0.915646      155010          4849.49
3    0.883046      222579          4319.22
4    0.835765      204347          3691.99
5    0.656395      105306          2584.27
6    0.619461      123507          1591.01
7    0.5327        117131           654.206
8    0.524289      154630             0
;
```

How can we interpret these?

Sensitivity Analysis of the Dedication Model

```
ampl: display buy, buy.rc, buy.up, buy.down;
:     buy          buy.rc        buy.up      buy.down     :=
A    62.1361   -1.42109e-14     105          96.4091
B     0          0.830612      1e+20         98.1694
C   125.243    -1.42109e-14     101.843      97.6889
D   151.505     1.42109e-14     101.374      93.2876
E   156.808    -1.42109e-14     102.917      80.7683
F   123.08       0              113.036     100.252
G     0          8.78684       1e+20         91.2132
H   124.157      0              104.989      92.3445
I   104.09       0              111.457     101.139
J    93.4579     0               94.9        37.9011
;
```

Sensitivity Analysis of the Dedication Model

```
ampl: display cash, cash.rc, cash.up, cash.down;
: cash    cash.rc   cash.up   cash.down     :=
0   0     0.0285714   1e+20    0.971429
1   0     0.0557823   1e+20   -0.0557823
2   0     0.0326005   1e+20   -0.0326005
3   0     0.0472812   1e+20   -0.0472812
4   0     0.17937     1e+20   -0.17937
5   0     0.0369341   1e+20   -0.0369341
6   0     0.0867604   1e+20   -0.0867604
7   0     0.0084114   1e+20   -0.0084114
8   0     0.524289    1e+20   -0.524289
;
```

Sensitivity Analysis in PuLP and Pyomo

- Both PuLP and Pyomo also support sensitivity analysis through suffixes.
- Pyomo
  - The option `-solver-suffixes='.*'` should be used.
  - The supported suffixes are `.dual`, `.rc`, and `.slack`.
- PuLP
  - PuLP creates suffixes by default when supported by the solver.
  - The supported suffixed are `.pi` and `.rc`.

Sensitivity Analysis of the Dedication Model with PuLP

```
for t in Periods[1:]:
    prob += (cash[t-1] - cash[t]
             + lpSum(BondData[b, 'Coupon'] * buy[b]
               for b in Bonds if BondData[b, 'Maturity'] >= t)
             + lpSum(BondData[b, 'Principal'] * buy[b]
               for b in Bonds if BondData[b, 'Maturity'] == t)
             == Liabilities[t]), "cash_balance_%s"%t

status = prob.solve()

for t in Periods[1:]:
    print 'Present of $1 liability for period', t,
    print prob.constraints["cash_balance_%s"%t].pi
```

Outline

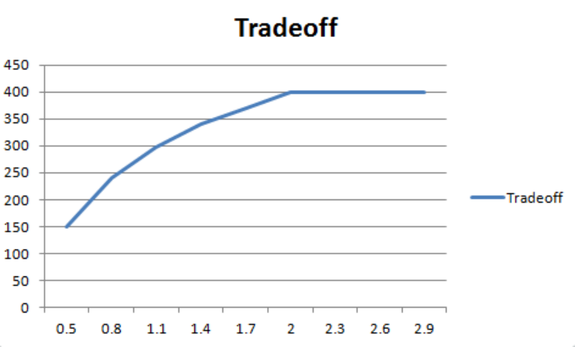
1. Sensitivity Analysis
2. Tradeoff Analysis (Multiobjective Optimization)
3. Nonlinear Modeling
4. Integer Programming
5. Stochastic Programming

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]
```

Outline

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:

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

  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

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

where r is a targeted minimum expected portfolio return.

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

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

Three Markowitz Models (cont.)

$$\text{(M3)} \quad \max_{x \in \mathbb{R}^n} \; \mu^\top x - \lambda x^\top Q x$$
$$\text{s.t.} \quad \sum_{i=1}^{n} x_i = 1,$$

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

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

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

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

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

Definition

Portfolios that satisfy the relation

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

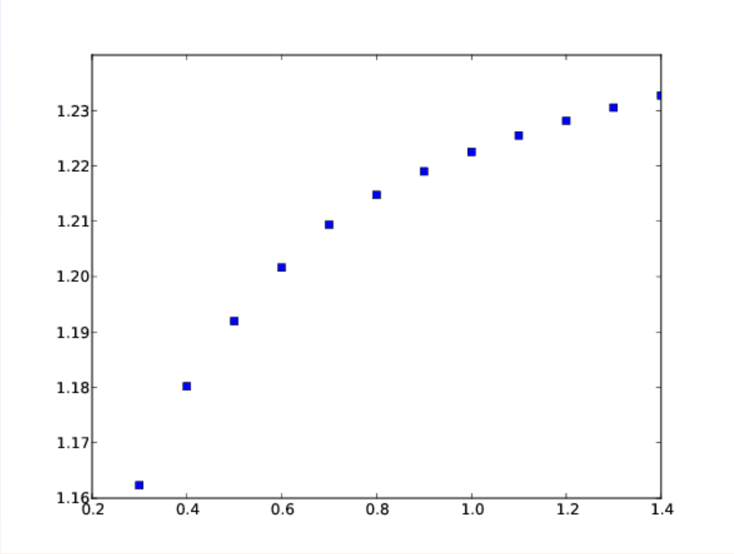
are called *efficient*. The curve $\sigma \mapsto r(\sigma)$, defined for $\sigma \geq \sigma_{\min}$, is called the *efficient frontier*.

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



Outline

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

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

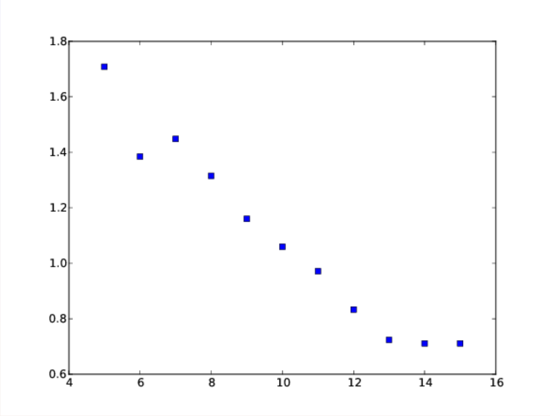
  where z_i is the number of shares of asset i that exist in the market and S^i the value of each share.

- 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.



Outline

Building a Retirement Portfolio

- When I retire in 10 years or so :-), I would like to have a comfortable income.
- I'll need enough savings to generate the income I'll need to support my lavish lifestyle.
- One approach would be to simply formulate a mean-variance portfolio optimization problem, solve it, and then "buy and hold."
- This doesn't explicitly take into account the fact that I can periodically rebalance my portfolio.
- I may make a different investment decision today if I explicitly take into account that I will have *recourse* at a later point in time.
- This is the central idea of stochastic programming.

Modeling Assumptions

- In Y years, I would like to reach a savings goal of G.
- I will rebalance my portfolio every v periods, so that I need to have an investment plan for each of $T = Y/v$ periods (stages).
- We are given a universe $\mathcal{N} = \{1, \ldots, n\}$ of assets to invest in.
- Let $\mu_{it}, i \in \mathcal{N}, t \in \mathcal{T} = \{1, \ldots, T\}$ be the (mean) return of investment i in period t.
- For each dollar by which I exceed my goal of G, I get a reward of q.
- For each dollar I am short of G, I get a penalty of p.
- I have $\$B$ to invest initially.

Variables

- $x_{it}, i \in \mathcal{N}, t \in \mathcal{T}$: Amount of money to invest in asset i at beginning of period t t.
- z : Excess money at the end of horizon.
- w : Shortage in money at the end of the horizon.

A Naive Formulation

minimize

$$qz + pw$$

subject to

$$\begin{aligned}
\sum_{i\in\mathcal{N}} x_{i1} &= B \\
\sum_{i\in\mathcal{N}} x_{it} &= \sum_{i\in\mathcal{N}} (1+\mu_{it}) x_{i,t-1} \qquad \forall t \in \mathcal{T} \\
\sum_{i\in\mathcal{N}} (1+\mu_{iT}) x_{iT} - z + w &= G \\
x_{it} &\geq 0 \qquad \forall i \in \mathcal{N}, t \in \mathcal{T} \\
z, w &\geq 0
\end{aligned}$$

A Better Model

- What are some weaknesses of the model on the previous slide?
- Well, there are many...
- For one, it doesn't take into account the variability in returns (i.e., risk).
- Another is that it doesn't take into account my ability to rebalance my portfolio *after* observing returns from previous periods.
- I can and would change my portfolio after observing the market outcome.
- Let's use our standard notation for a market consisting of n assets with the price of asset i at the end of period t being denoted by the random variable S_t^i.
- Let $R_{it} = S_t^i / S_{t-1}^i$ be the return of asset i in period t.
- As we have done previously, let's take a scenario approach to specifying the distribution of R_{it}.

Scenarios

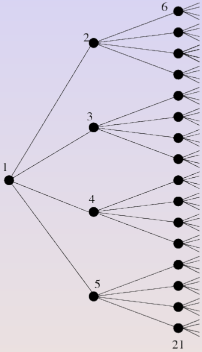
- We let the scenarios consist of all possible sequences of outcomes.
- Generally, we assume that for a particular realization of returns in period t, there will be M possible realizations for returns in period $t+1$.
- We then have M^T possible scenarios indexed by a set S.
- As before, we can then assume that we have a probability space (P^t, Ω^t) for each period t and that Ω^t is partitioned into $|S|$ subsets $\Omega^t_s, s \in S$.
- We then let $p^t_s = P(\Omega^t_s)\ \forall s \in S, t \in \mathcal{T}$.
- For instance, if $M = 4$ and $T = 3$, then we might have...

$t=1$	$t=2$	$t=3$
1	1	1
1	1	2
1	1	3
1	1	4
1	2	1
	⋮	
4	4	4

- $|S| = 64$
- We can specify any probability on this outcome space that we would like.
- The time period outcomes don't need to be equally likely and returns in different time periods need not be mutually independent.

A Scenario Tree

- Essentially, we are approximating the continuous probability distribution of returns using a discrete set of outcomes.
- Conceptually, the sequence of random events (returns) can be arranged into a tree



Making it Stochastic

- Once we have a distribution on the returns, we could add uncertainty into our previous model simply by considering each scenario separately.
- The variables now become
  - $x_{its}, i \in \mathcal{N}, t \in \mathcal{T}$: Amount of money to reinvest in asset i at beginning of period t in scenario s.
  - $z_s, s \in S$: Excess money at the end of horizon in scenario s.
  - $w_s, s \in S$: Shortage in money at the end of the horizon in scenario s.
- Note that the return μ_{its} is now indexed by the scenario s.

A Stochastic Version: First Attempt

minimize

$$???????????????$$

subject to

$$
\begin{aligned}
\sum_{i \in \mathcal{N}} x_{i1} &= B \\
\sum_{i \in \mathcal{N}} x_{its} &= \sum_{i \in \mathcal{N}} (1 + \mu_{its}) x_{i,t-1,s} \qquad \forall t \in \mathcal{T}, \forall s \in S \\
\sum_{i \in \mathcal{N}} \mu_{iTs} x_{iTs} - z_s + w_s &= G \qquad \forall s \in S \\
x_{its} &\geq 0 \qquad \forall i \in \mathcal{N}, t \in \mathcal{T}, \forall s \in S \\
z_s, w_s &\geq 0 \qquad \forall s \in S
\end{aligned}
$$

Easy, Huh?

- We have just converted a multi-stage stochastic program into a deterministic model.
- However, there are some problems with our first attempt.
- What are they?

One Way to Fix It

- What we did to create our *deterministic equivalent* was to create copies of the variables for every scenario at every time period.
- One missing element is that we still have not have a notion of a probability distribution on the scenarios.
- But there's an even bigger problem...
- We need to enforce *nonanticipativity*...
- Let's define E_s^t as the set of scenarios with same outcomes as scenario s up to time t.
- At time t, the copies of all the anticipative decision variables corresponding to scenarios in E_s^t must have the same value.
- Otherwise, we will essentially be making decision at time t using information only available in periods after t.

A Stochastic Version: Explicit Nonanticipativity

minimize

$$\sum_{s \in S} p_s (q z_s - p w_s)$$

subject to

$$\begin{aligned}
\sum_{i \in \mathcal{N}} x_{i1} &= B \\
\sum_{i \in \mathcal{N}} x_{its} &= \sum_{i \in \mathcal{N}} (1 + \mu_{its}) x_{i,t-1,s} \qquad \forall t \in \mathcal{T}, \forall s \in S \\
\sum_{i \in \mathcal{N}} \mu_{iTs} x_{iTs} - z_s + w_s &= G \qquad \forall s \in S \\
x_{its} &= x_{its'} \; \forall i \in \mathcal{N}, \forall t \in T, \forall s \in S, \forall s' \in E_s^t \\
x_{its} &\geq 0 \qquad \forall i \in \mathcal{N}, t \in \mathcal{T}, \forall s \in S \\
z_s, w_s &\geq 0 \qquad \forall s \in S
\end{aligned}$$

Another Way

- We can also enforce nonanticipativity by using the “right” set of variables.
- We have a vector of variables for each node in the scenario tree.
- This vector corresponds to what our decision would be, given the realizations of the random variables we have seen so far.
- Index the nodes $= \{1, 2, \ldots\}$.
- We will need to know the “parent” of any node.
- Let $A(l)$ be the ancestor of node $l \in$ in the scenario tree.
- Let $N(t)$ be the set of all nodes associated with decisions to be made at the beginning of period t.

Another Multistage Formulation

maximize

$$\sum_{l \in N(T)} p_l \left(q z_l + p w_l \right)$$

subject to

$$\begin{array}{rcll} \sum_{i \in \mathcal{N}} x_{i1} & = & B & \\ \sum_{i \in \mathcal{N}} x_{il} & = & \sum_{i \in \mathcal{N}} (1 + \mu_{il}) x_{i,A(l)} & \forall l \in \\ \sum_{i \in \mathcal{N}} \mu_{il} x_{il} - z_l + w_l & = & G & \forall l \in N(T) \\ x_{il} & \geq & 0 & \forall i \in \mathcal{N}, l \in \\ z_l, w_l & \geq & 0 & \forall l \in N(T) \end{array}$$

PuLP Model for Retirement Portfolio (DE-PuLP.py)

```
Investments = ['Stocks', 'Bonds']

NumNodes = 21
NumScen = 64

b = 10000
G = 15000
q = 1 #0.05;
r = 2 #0.10;

Return = {
   0 : {'Stocks' : 1.25, 'Bonds' : 1.05},
    1 : {'Stocks' : 1.10, 'Bonds' : 1.05},
    2 : {'Stocks' : 1.00, 'Bonds' : 1.06},
    3 : {'Stocks' : 0.95, 'Bonds' : 1.08}
    }

NumOutcome = len(Return)
```

PuLP Model for Retirement Portfolio

```
x = LpVariable.dicts('x', [(i, j) for i in range(NumNodes)
                           for j in Investments], 0, None)
y = LpVariable.dicts('y', range(NumScen), 0, None)
w = LpVariable.dicts('w', range(NumScen), 0, None)

A  = dict([(k, (k-1)/NumOutcome) for k in range(1, NumNodes)])
A2 = dict([(s, 5 + s/NumOutcome) for s in range(NumScen)])
O  = dict([(k, (k-1) % NumOutcome)   for k in range(1, NumNodes)])
O2 = dict([(s, s % NumOutcome)   for s in range(NumScen)])

prob += lpSum(float(1)/NumScen * (q * y[s] + r * w[s])
              for s in range(NumScen))
prob += lpSum(x[0,i] for i in Investments) == b,
for k in range(1, NumNodes):
    prob += (lpSum(x[k,i] for i in Investments) ==
    lpSum(Return[O[k]][i] * x[A[k],i]  for i in Investments))
for s in range(NumScen):
    prob += lpSum(Return[O2[s]][i] * x[A2[s],i]
                  for i in Investments) - y[s] + w[s] == G
```