Financial Optimization
ISE 347/447

Lecture 15

Dr. Ted Ralphs

Reading for This Lecture

- C&T Chapter 12

Stock Market Indices

- A *stock market index* is a statistic indicating the composite value of a specified basket of stocks.
- Stock market indices can be weighted in a number of ways.
  - Price: Index is the total/average price of all stocks (i.e., the value of a basket consisting of one share of each stock).
  - Market share: Index is weighted by the total number of shares of each stock outstanding outstanding.
  - Market capitalization: Index is weighted by the total value of all the shares outstanding.
  - Float: Index is weighted by the total value of all publicly traded shares outstanding.
- An *index fund* is a mutual fund whose goal is to provide roughly the same return as that of an associated index (e.g., the S&P 500).
- Index funds have become a popular investment medium for long-term investors.
- There are sound theoretical reasons for investing in an index fund.

Stock Market Indices

- Let us consider an index weighted my market capitalization, such as the NASDAQ Composite.
- The investments in this index form a market (in this case consisting of all stocks listed on the NASDAQ.
- As in Lecture 12, let $\tilde{w}$ be the relative market capitalization of the assets in the market, i.e.,

$$\tilde{w}_i = \frac{z_i S^i}{\sum_{i=0}^{n} z_i S^i},$$

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

The Market Portfolio Revisited

- As we previously noted, under the assumption that the market is efficient, $\tilde{w}$ must be the solution to a Markowitz model consisting of the assets in this market.
- In fact, under these assumptions, $\tilde{w}$ will represent the previously discussed market portfolio.
- Hence, the combination of an index fund with a risk-free asset will always be an efficient allocation.
- Unfortunately, there is ample evidence that these assumptions do not hold in practice.
- Nevertheless, it seems to be the case that fund managers generally underperform the market on average.
- Actively managed funds also incur much higher transaction costs than unmanaged ones.
- It may therefore still make sense to invest in index funds in many cases.

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

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

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

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

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

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

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:

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

Pyomo Model for Constructing Index Fund

```
model.K = Param(mutable = True)
model.assets = Set()
model.T = Set(initialize = range(1994, 2015))
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)

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, rule=selection_rule)
```

Pyomo Model for Constructing Index Fund (cont.)

```
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 mode
model.objective = Objective(sense=maximize, rule=objective_rule)
```

Solution Strategy for Large Instances

- This model consists of a very large number of variables and constraints (250K for the S&P 500).
- Even solving the LP relaxation for such a problem could be difficult.
- To overcome this difficulty, we can use the Lagrangian approach discussed in Lecture 13.
- In this case, we relax the constraints that require each stock to have exactly one representative. Then we get the relaxation

$$L(u) = \max \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij} + \sum_{i=1}^{n} u_i \left(1 - \sum_{j=1}^{n} x_{ij} \right)$$

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

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

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

Evaluating L(u)

We can rewrite the objective function of the Lagrangian relaxation as

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} (\rho_{ij} - u_i) x_{ij} + \sum_{i=1}^{n} u_i$$

If we set

$$C_j = \sum_{i=1}^{n} (\rho_{ij} - u_i)^+,$$

then it is easy to see that

$$\begin{aligned} L(u) = \max & \sum_{j=1}^{n} C_j y_j + \sum_{i=1}^{n} u_i \\ \text{s.t.} & \sum_{j=1}^{n} y_j = q \\ & y_j \in \{0,1\} \qquad \forall\, j = 1, \ldots, n \end{aligned}$$

Evaluating L(u) (cont.)

- The end result is that in an optimal solution to the Lagrangian relaxation, we have
  - The representatives are the q stocks with the largest values of C_j.
  - $x_{ij} = y_j$ if and only if $\rho_{ij} > u_i$.
- We can use this choice of representatives as a heuristic for finding optimal solutions.
  - For a given vector u, choose the q stocks with the largest values of C_j.
  - Assign each remaining stock to the most similar among the representatives.
  - This solution will be feasible.

Solving the Lagrangian Dual

- To get the best possible bound, we would like to solve the Lagrangian dual

$$\min_u L(u)$$

- Let $\{(x^k, y^k)\}_{k=1}^T$ be the set of all 0-1 solutions to the Lagrangian relaxation.
- Then we have that

$$L(u) = \max_{k=1,\ldots,T} \sum_{i=1}^n \sum_{j=1}^n (\rho_{ij} - u_i) x_{ij}^k + \sum_{i=1}^n u_i$$

- Hence, $L(u)$ is a convex function

Subgradient Optimization

- Since L is convex, we can also use a basic *line-search algorithm* to maximize it.
- We start with an initial guess u^0, set $k \leftarrow 0$, and then iterate until a stopping criteria is met.
  1. Determine a subgradient s^k to L at u^k.
  2. For a given step size α^k, set $u^{k+1} \leftarrow u^k + \alpha^k s^k$.
- In our case, finding a subgradient is easy.
- We can just take $s_i^k = 1 - \sum_{j=1}^{n} x_{ij}$
- This algorithm is guaranteed to converge to the optimal solution as long as $\{\alpha^k\}_{k=0}^{\infty} \rightarrow 0$ and $\sum_{k=0}^{\infty} \alpha^k = \infty$
- In practice, one usually uses a geometric progression for the step sizes.

The General Principle

- The principle in this example can be generalized to any (nonconvex) optimization problem.
- Recall the *Lagrangian dual problem* (D) from Lecture 10:

$$\max \Theta(u, v)$$
$$\text{s.t.} \quad u \geq 0$$

where $\Theta(u, v) = \inf_{x \in X} \Phi(x, u, v)$ and

$$\Phi(x, u, v) \equiv f(x) + \sum_{i=1}^{m} u_i g_i(x) + \sum_{i=1}^{l} v_i h_i(x)$$

- We include in the set X the "easy" constraints, so that Θ can be evaluated effectively (this is the *Lagrangian subproblem*).
- We can then use this Lagrangian dual to generate bounds within a branch-and-bound algorithm.

General Subgradient Algorithm for the Lagrangian Dual

- The idea of the subgradient algorithm is to first fix μ, v and solve the Lagrangian subproblem to get x.
- Then update μ, v by moving in an ascent direction for Θ.
- Here is a basic *subgradient algorithm* for solving the Lagrangian dual:

  1. Choose initial Lagrange multipliers $\mu^0 \geq 0$, v^0 and set $t = 0$.
  2. Solve the Lagrangian subproblem (evaluate $\Theta(\mu, v)$) to obtain x^t.
  3. Calculate an ascent direction d for Θ (usually the violation of the constraints at x^t).
  4. Set $(\mu^{t+1}, v^{t+1}) \leftarrow (\mu^t, v^t) + \lambda^t \frac{d}{\|d\|}$ where λ^t is the chosen *step size*.
  5. Set $t \leftarrow t + 1$ and go to step 2.

- This algorithm is guaranteed to converge to the optimal solution as long as $\{\lambda^t\}_{t=0}^{\infty} \to 0$ and $\sum_{t=0}^{\infty} \lambda^t = \infty$
- Sometimes, it's difficult to know when the optimal solution has been reached.

A Linear Programming Formulation

- Note that when X is a finite set, we can write the Lagrangian dual equivalently as

$$\min_{z,u,v}\{z \mid z \geq \Phi(x^k, u, v)\ \forall k = 1, \ldots, T\},$$

  where $\{x^k\}_{i=1}^T$ are the members of set X.

- When Φ is linear, this is a linear program, but with a very large number of constraints.
- We can solve this LP by starting with a subset of the constraints and adding new ones dynamically.
- This is called *dynamic constraint generation*.
- In our example, we have

$$\min_{z,u}\{z + \sum_{i=1}^{n} u_i \mid z \geq \sum_{i=1}^{n}\sum_{j=1}^{n}(\rho_{ij} - u_i)x_{ij}^k\ \forall k = 1, \ldots, T\},$$

  where $\{(x^k, y^k)\}_{i=1}^T$ are the members of X.

Dantzig-Wolfe Decomposition

- When Φ is linear, then the previous algorithm amounts to relaxing a set of linear constraints $A''x \leq b''$.
- The dual of the linear program from the previous slide is an LP with one column for each member of X.

$$\begin{aligned} \max \quad & c^\top x \\ s.t. \quad & \sum_{s \in X} \lambda_s s = x \\ & A''x \leq b'' \\ & \sum_{s \in X} \lambda_s = 1 \\ & \lambda \in \mathbb{R}_+^X \end{aligned}$$

- Solving this LP is equivalent to solving the Lagrangian dual, but this is sometimes preferred for algorithmic reasons.
- The solution method is to dynamically generate the *columns* and is called *dynamic column generation*.