Financial Optimizations
ISE 347/447

Lecture 3

Dr. Ted Ralphs
Reading for This Lecture

- **AMPL Book**: Chapter 1
- **C&T** Sections 3.1 and 3.2
AMPL

- AMPL is one of the most commonly used modeling languages, but many other languages, including GAMS, are similar in concept.
- AMPL has many of the features of a programming language, including loops and conditionals.
- Most available solvers will work with AMPL.
- GMPL and ZIMPL are open source languages that implement subsets of AMPL.
- AMPL will work CPLEX, XPRESS-MP, MOSEK, which are commercial solvers available in the ISE department.
- AMPL can also be used with most of the solvers in COIN-OR, a repository of open source software for operations research.
- You can also submit AMPL models to the NEOS server.
- Student versions can be downloaded from www.ampl.com.
- Finally, you will be able to use AMPL through the Excel plug-in Solver Studio that we will use extensively.
Example: Simple Bond Portfolio Model

• A bond portfolio manager has $100K to allocate to two different bonds.

<table>
<thead>
<tr>
<th>Bond</th>
<th>Yield</th>
<th>Maturity</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>3</td>
<td>A (2)</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>4</td>
<td>Aaa (1)</td>
</tr>
</tbody>
</table>

• The goal is to maximize total return subject to the following limits.
  – The average rating must be at most 1.5 (lower is better).
  – The average maturity must be at most 3.6 years.

• Any cash not invested will be kept in a non-interest bearing account and is assumed to have an implicit rating of 0 (no risk).
**AMPL Concepts**

- In many ways, **AMPL** is like any other programming language.
- **Example**: Bond Portfolio Model

```ampl
ampl: option solver OSAmplClient;
ampl: option OSAmplClient_options "solver clp";
ampl: var X1;
ampl: var X2;
ampl: maximize yield: 4*X1 + 3*X2;
ampl: subject to cash: X1 + X2 <= 100;
ampl: subject to rating: 2*X1 + X2 <= 150;
ampl: subject to maturity: 3*X1 + 4*X2 <= 360;
ampl: subject to X1_limit: X1 >= 0;
ampl: subject to X2_limit: X2 >= 0;
ampl: solve;
...
ampl: display X1;
X1 = 50
ampl: display X2;
X2 = 50
```
**Storing Commands in a File**

- You can type the commands into a file and then load them.
- This makes it easy to modify your model later.
- **Example:**

```ampl
ampl: option solver OSAmplClient;
ampl: option OSAmplClient_options "solver clp";
ampl: model bonds_simple.mod;
ampl: solve;
...
ampl: display X1;
X1 = 50
ampl: display X2;
X2 = 50
```
Generalizing the Model

• Suppose we want to generalize this production model to more than two products.

• AMPL allows the model to be separated from the data.

• Components of a linear optimization problem in AMPL

  – Data

    * **Sets**: lists of products, raw materials, etc.
    * **Parameters**: numerical inputs such as costs, production rates, etc.

  – Model

    * **Variables**: Values in the model that need to be decided upon.
    * **Objective Function**: A function of the variable values to be maximized or minimized.
    * **Constraints**: Functions of the variable values that must lie within given bounds.
Example: General Bond Portfolio Model

set bonds;       # bonds available

param yield {bonds};  # yields
param rating {bonds};  # ratings
param maturity {bonds};  # maturities
param max_rating;      # Maximum average rating allowed
param max_maturity;    # Maximum maturity allowed
param max_cash;        # Maximum available to invest

var buy {bonds} >= 0;    # amount to invest in bond i

maximize total_yield : sum {i in bonds} yield[i] * buy[i];

subject to cash_limit : sum {i in bonds} buy[i] <= max_cash;
subject to rating_limit :
    sum {i in bonds} rating[i]*buy[i] <= max_cash*max_rating;
subject to maturity_limit :
    sum {i in bonds} maturity[i]*buy[i] <= max_cash*max_maturity;
Example: Bond Portfolio Data

set bonds := A B;

param : yield rating maturity :=
  A  4  2   3
  B  3  1   4;

param max_cash := 100;
param max_rating 1.5;
param max_maturity 3.6;
ampl: model bonds.mod;
ampl: data bonds.dat;
ampl: solve;
...
ampl: display buy;
buy [*] :=
A   50
B   50 ;
Modifying the Data

• Suppose we want to increase available production hours by 2000.

• To resolve from scratch, simply modify the data file and reload.

```
ampl: reset data;
ampl: data bonds_alt.dat;
ampl: solve;
...
ampl: display buy;
buy [*] :=
A  30
B  70
;
```
Modifying Individual Data Elements

- Instead of resetting all the data, you can modify one element.

```ampl
ampl: reset data max_cash;
ampl: data;
ampl data: param max_cash := 150;
ampl data: solve;
...
ampl: display buy;
buy [*] :=
A  45
B  105
;```
Extending the Model

- Now suppose we want to add another type of bond.

```plaintext
set bonds := A B C;

param : yield rating maturity :=
    A 4 2 3
    B 3 1 4
    C 5 3 2;

param max_cash := 100;
param max_rating 1.5;
param max_maturity 3.6;
```
Solving the Extended Model

ampl: reset data;
ampl: data bonds_extended.dat;
ampl: solve;
..
ampl: display buy;
buy [*] :=
A  0
B  85
C  15
;
Another obvious source of data is a spreadsheet, such as Excel.

AMPL has commands for accessing data from a spreadsheet directly from the language.

An alternative is to use SolverStudio.

SolverStudio allows the model to be composed within Excel and imports the data from an associated sheet.

Results can be printed to a window or output to the sheet for further analysis.
Further Generalization

• Note that in our AMPL model, we essentially had three “features” of a bond that we wanted to take into account.
  – Maturity
  – Rating
  – Yield

• We constrained the level of two of these and then optimized the third one.

• The constraints for the features all have the same basic form.

• What if we wanted to add another feature?

• We can make the list of features a set and use the concept of a two-dimensional parameter to create a table of bond data.
The Generalized Model

set bonds;
set features;

param bond_data {bonds, features};
param limits{features};
param yield{bonds};

param max_cash;

var buy {bonds} >= 0;

maximize obj : sum {i in bonds} yield[i] * buy[i];

subject to cash_limit : sum {i in bonds} buy[i] <= max_cash;

subject to limit_constraints {f in features}:
sum {i in bonds} bond_data[i, f]*buy[i] <= max_cash*limits[f];
Simple Bond Portfolio Example in Python (PuLP)

```python
from pulp import LpProblem, LpVariable, lpSum, LpMaximize, value

prob = LpProblem("Dedication Model", LpMaximize)

X1 = LpVariable("X1", 0, None)
X2 = LpVariable("X2", 0, None)

prob += 4*X1 + 3*X2
prob += X1 + X2 <= 100
prob += 2*X1 + X2 <= 150
prob += 3*X1 + 4*X2 <= 360

prob.solve()

print 'Optimal total cost is: ', value(prob.objective)

print "X1 :", X1.varValue
print "X2 :", X2.varValue
```
Notes About the Model

• Like the simple AMPL model, we are not using indexing or any sort of abstraction here.

• The syntax is very similar to AMPL.

• To achieve separation of data and model, we use Python’s import mechanism.
Bond Portfolio Example: Abstracting the PuLP Model

from pulp import LpProblem, LpVariable, lpSum, LpMaximize, value
from bonds_data import bonds, max_rating, max_maturity, max_cash
prob = LpProblem("Bond Selection Model", LpMaximize)
buy = LpVariable.dicts('bonds', bonds.keys(), 0, None)
prob += lpSum(bonds[b]['yield'] * buy[b] for b in bonds)
prob += lpSum(buy[b] for b in bonds) <= max_cash, "cash"
prob += (lpSum(bonds[b]['rating'] * buy[b] for b in bonds)
<= max_cash*max_rating, "ratings")
prob += (lpSum(bonds[b]['maturity'] * buy[b] for b in bonds)
<= max_cash*max_maturity, "maturities")
Notes About the Model

• We can use Python’s native `import` mechanism to get the data.
• Note, however, that the data is read and stored *before* the model.
• This means that we don’t need to declare sets and parameters.
• Carriage returns are syntactic (parentheses imply line continuation).

• Constraints
  – Naming of constraints is optional and only necessary for certain kinds of post-solution analysis.
  – Constraints are added to the model using a very intuitive syntax.
  – Objectives are nothing more than expressions that are to be optimized rather than explicitly constrained.

• Indexing
  – Indexing in Python is done using the native dictionary data structure.
  – Note the extensive use of comprehensions, which have a syntax very similar to quantifiers in a mathematical model.
Bond Portfolio Example: Solution in PuLP

```python
prob.solve()

epsilon = .001

print 'Optimal purchases:
for i in bonds:
    if buy[i].varValue > epsilon:
        print 'Bond', i, "":"", buy[i].varValue
```
Bond Portfolio Example: Data Import File

bonds = {'A' : {'yield' : 4,
          'rating' : 2,
          'maturity' : 3},
          
'B' : {'yield' : 3,
          'rating' : 1,
          'maturity' : 4},
      }

max_cash = 100
max_rating = 1.5
max_maturity = 3.6
Notes About the Data Import

- We are storing the data about the bonds in a “dictionary of dictionaries.”
- With this data structure, we don’t need to separately construct the list of bonds.
- We can access the list of bonds as `bonds.keys()`.
- Note, however, that we still end up hard-coding the list of features and we must repeat this list of features for every bond.
- We can avoid this using some advanced Python programming techniques, but SolverStudio makes this easy.
Bond Portfolio Example: PuLP Model in SolverStudio

\[
\text{buy} = \text{LpVariable.dicts('bonds', bonds, 0, None)} \\
\text{for } f \text{ in features:} \\
\hspace{1em} \text{if } \text{sense}[f] == "Max": \\
\hspace{2em} \text{prob} += \text{lpSum}(\text{bond_data}[b, f] \times \text{buy}[b] \text{ for } b \text{ in bonds}) \\
\hspace{1em} \text{elif } \text{sense}[f] == "Max": \\
\hspace{2em} \text{prob} += \text{lpSum}(-\text{bond_data}[b, f] \times \text{buy}[b] \text{ for } b \text{ in bonds}) \\
\hspace{1em} \text{elif } \text{sense}[f] == '>': \\
\hspace{2em} \text{prob} += (\text{lpSum}(\text{bond_data}[b, f] \times \text{buy}[b] \text{ for } b \text{ in bonds}) \\
\hspace{3em} \geq \text{max_cash} \times \text{limits}[f], f) \\
\hspace{1em} \text{else}: \\
\hspace{2em} \text{prob} += (\text{lpSum}(\text{bond_data}[b, f] \times \text{buy}[b] \text{ for } b \text{ in bonds}) \\
\hspace{3em} \leq \text{max_cash} \times \text{limits}[f], f) \\
\text{prob} += \text{lpSum}(\text{buy}[b] \text{ for } b \text{ in bonds}) \leq \text{max_cash}, "cash"
\]

\text{status} = \text{prob.solve()}
Notes About the SolverStudio PuLP Model

- We've explicitly allowed the option of optimizing over one of the features, while constraining the others.

- Later, we'll see how to create tradeoff curves showing the tradeoffs among the constraints imposed on various features.
Portfolio Dedication

Definition 1. Dedication or cash flow matching refers to the funding of known future liabilities through the purchase of a portfolio of risk-free non-callable bonds.

Notes:

• Dedication is used to eliminate interest rate risk.
• Dedicated portfolios do not have to be managed.
• The goal is to construct such portfolio at a minimal price from a set of available bonds.
Example: Portfolio Dedication

• A pension fund faces liabilities totalling $l_j$ for years $j = 1, ..., T$.
• The fund wishes to dedicate these liabilities via a portfolio comprised of $n$ different types of bonds.
• Bond type $i$ costs $c_i$, matures in year $j_i$, and yields a yearly coupon payment of $d_i$ up to maturity.
• The principal paid out at maturity for bond $i$ is $p_i$. 
Example: LP Formulation

We assume that for each year $j$ there is at least one type of bond $i$ with maturity $j_i = j$, and there are none with $j_i > T$.

Let $x_i$ be the number of bonds of type $i$ purchased, and let $z_j$ be the cash on hand at the beginning of year $j$ for $j = 0, \ldots, T$. Then the dedication problem is the following LP,

$$
\begin{align*}
\min_{(x,z)} & \quad z_0 + \sum_i c_i x_i \\
\text{s.t.} & \quad z_{j-1} - z_j + \sum_{\{i: j_i \geq j\}} d_i x_i + \sum_{\{i: j_i = j\}} p_i x_i = \ell_j, \quad (j = 1, \ldots, T-1) \\
& \quad z_{T-1} + \sum_{\{i: j_i = T\}} (p_i + d_i) x_i = \ell_T. \\
& \quad z_j \geq 0, j = 1, \ldots, T \\
& \quad x_i \geq 0, i = 1, \ldots, n
\end{align*}$$

Here is the model for the portfolio dedication example.

set bonds; # bonds available for purchase
param T > 0 integer; # Years in the planning horizon
param liabilities {1..T+1}; # Liabilities by year
param price {bonds}; # The cost of each bond type
param maturity {bonds}; # Bond maturities
param coupon {bonds}; # The coupon payment amounts
param principal {bonds}; # Principal paid at maturity
var buy {bonds} >= 0; # Number of bonds to buy
var cash {0..T} >= 0; # Cash at beginning of year j

minimize total_cost : cash[0] + sum {i in bonds} price[i] * buy[i];

subject to cash_balance {t in 1..T}: cash[t-1] - cash[t] + sum {i in bonds : maturity[i] >= t} coupon[i] * buy[i] + sum {i in bonds : maturity[i] = t} principal[i] * buy[i] = liabilities[t];
## Portfolio Dedication Data

set bonds := A B C D E F G H I J;
param T := 8;
param := liabilities :=
  1 12000  2 18000
  3 20000  4 20000
  5 16000  6 15000
  7 12000  8 10000;
param := price coupon principal maturity :=
  A 102 5 100 1
  B  99 3.5 100 2
  C 101 5 100 2
  D  98 3.5 100 3
  E  98 4 100 4
  F 104 9 100 5
  G 100 6 100 5
  H 101 8 100 6
  I 102 9 100 7
  J  94 7 100 8;
Software for Linear Optimization

• Caveat: What follows includes only linear solvers. We will look at nonlinear solvers a little later.

• Commercial solvers
  – CPLEX ← available in ISE
  – XPRESS-MP ← available in ISE
  – Gurobi ← Free for student use
  – MOSEK
  – LINDO
  – Excel SOLVER

• Open source solvers (free to download and use)
  – CLP
  – DYLP
  – GLPK
  – SOPLEX
  – lp_solve
Computational Infrastructure for Operations Research (COIN-OR)

- COIN-OR is an open source project dedicated to the development of open source software for solving operations research problems.
- COIN-OR distributes a free and open source suite of software that can handle all the classes of problems we’ll discuss.
  - Clp (LP)
  - Cbc (MILP)
  - Ipopt (NLP)
  - SYMPHONY (MILP, BMILP)
  - Bonmin (Convex MINLP)
  - Couenne (Nonconvex MINLP)
  - Optimization Services (Interface)
- COIN also develops standards and interfaces that allow software components to interoperate.
- We will be using COIN software frequently throughout the semester.

http://www.coin-or.org
Using COIN-OR with AMPL

• Install the OSAmplClient.

• Type the following options in AMPL:

    ampl: option solver OSAmplClient;
    ampl: option OSAmplClient_options "solver clp";

• The solver can be any of the above, except for Bonmin (coming soon).

• It is even possible to solve problems remotely and we may try this at some point.
Other Modeling Languages

- **OPL**
  - *OPL Studio* is a modeling IDE available in the ISE department.
  - The model format is similar to *AMPL*.

- **GAMS**
  - Another modeling language like *AMPL*.
  - Also available in ISE.

- **GMPL**
  - Another language very similar to *AMPL*.
  - Works with GLPK, CLP, and SYMPHONY.

- **PuLP/Pyomo**
  - Python-based modeling languages.
  - Similar in concept to *AMPL* but with the full power of Python.