Mixed-integer linear programming in MATLAB

Seth DeLand
Topics

- Refresher on Mixed-integer linear programming (MILP)
- Solving an MILP problem in MATLAB
- MILP algorithms
- Tuning the solver
MILP vs. LP

- **Linear Programming** (linprog)
  \[
  \min_x f^T x \quad \text{s.t.} \quad \begin{cases} 
  Ax \leq b \\
  A_{eq}x = b_{eq} \\
  lb \leq x \leq ub
  \end{cases}
  \]

- **Mixed-integer LP** (intlinprog)
  \[
  \min_x f^T x \quad \text{s.t.} \quad \begin{cases} 
  Ax \leq b \\
  A_{eq}x = b_{eq} \\
  lb \leq x \leq ub
  \end{cases}
  \]

  Some (or all) \( x \) must be integers

---

\[
[xopt, fval, eflag, output] = \text{linprog}(f, A, b, Aeq, beq, lb, ub, opts);
\]

\[
[xopt, fval, eflag, output] = \text{intlinprog}(f, intcon, A, b, Aeq, beq, lb, ub, opts);
\]
**Demo: Cash-flow matching**

**Idea:** Buy bonds to cover pension fund obligations

**Variables:** How many of each bond to buy?

**Constraints:** Payments from bonds must be greater than or equal to pension fund obligations

**Objective:** Minimize the size of the investment you make

<table>
<thead>
<tr>
<th>Time Period</th>
<th>Cash Flow ($)</th>
<th>Payments ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>5 x 10^5</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>10 x 10^5</td>
</tr>
<tr>
<td>7</td>
<td>120</td>
<td>15 x 10^5</td>
</tr>
<tr>
<td>8</td>
<td>140</td>
<td>10 x 10^5</td>
</tr>
</tbody>
</table>
Extending the problem

\[
\min \ f^T x \quad \text{s.t.} \quad \begin{cases} 
Ax & \leq b \\
A_{eq}x & = b_{eq} \\
lb & \leq x \leq ub
\end{cases}
\]

\(x(\text{intcon})\) must be integers

\[
n_1 = 1000y_1 \\
\vdots \\
n_5 = 1000y_5
\]

\[
\begin{bmatrix}
1 & 0 & -1000 & 0 \\
\vdots & \vdots & \vdots & \vdots \\
0 & 1 & 0 & -1000
\end{bmatrix}
\begin{bmatrix}
n_1 \\
\vdots \\
n_5 \\
y_1 \\
y_5
\end{bmatrix} = \begin{bmatrix}
0 \\
\vdots \\
0
\end{bmatrix}
\]

\([-\text{cashFlows}] \begin{bmatrix}
n_1 \\
\vdots \\
n_5
\end{bmatrix} \leq \{-\text{obligations}\}
\]

\([-\text{cashFlows} \ \text{zeros}(8,5)] \begin{bmatrix}
n_1 \\
\vdots \\
n_5 \\
y_1 \\
y_5
\end{bmatrix} \leq \{-\text{obligations}\}\]
Recap

- Better solution from `intlinprog` than rounding the solution from `linprog`

- Added binary variables
  - Allowed us to model “quantities of 1000”
  - Updated sizes of constraint matrices and objective function
Demo: Index replication

- Correlation matrix

\[ C(i,j) : \text{how closely is asset } i \text{ related to asset } j? \]
Index replication

- **Goal**: pick a subset of assets based on correlations
  - If asset A is highly correlated with asset B, then we only need one of them in our replicating portfolio

- Each asset in the replicating portfolio represents one or more assets in the universe

- **Optimization problem**: select 20 assets for the replicating portfolio that best represent the universe
Approach – binary mask

$$C: \begin{bmatrix} 1 & 0.18 & 0.71 & 0.48 & 0.23 \\ 0.18 & 1 & 0.33 & 0.54 & 0.44 \\ 0.71 & 0.33 & 1 & 0.12 & 0.31 \\ 0.48 & 0.54 & 0.12 & 1 & 0.63 \\ 0.23 & 0.44 & 0.31 & 0.63 & 1 \end{bmatrix} \quad x: \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Constraints on x
- Only a single 1 in each row
- Only 2 columns that contain 1’s

Objective
- Maximize $\sum (C \cdot x)$
Approach – binary mask

\[
\text{sum}(C \cdot x) = 3.88
\]

Constraints on \( x \)

- Only a single 1 in each row
- Only 2 columns that contain 1’s

Objective

- Maximize \( \text{sum}(C \cdot x) \)
MILP formulation

- **Variables:**
  \[ x_{i,j} \ (0 - 1) \text{ is stock } i \text{ represented by stock } j ? \]
  \[ y_j \ (0 - 1) \text{ is stock } j \text{ in the replicating portfolio?} \]

- **Constraints:**
  \[ \sum_{i=1}^{n} x_{i,j} = 1 \]
  \[ \sum_{j=1}^{n} y_j = 20 \]
  \[ \sum_{i=1}^{n} x_{i,j} - y_j \leq 0 \]
  Only one “1” per row

- **Objective:**
  \[ \max_x \sum_{i=1}^{n} \sum_{j=1}^{n} C_{i,j} x_{i,j} \]
MILP algorithm

LP Preprocessing

Root LP

Integer Programming Preprocessing

Cut Generation

Heuristics

Branch and Bound

Reduce the problem size
Solve the problem without integer constraints
Tighten LP relaxation
Further tighten LP relaxation
Try to find integer feasible solutions
Systematically search for optimal solution
Tuning options

- **LP Preprocessing**
  - LPPreprocess

- **Root LP**
  - RootLPAlgorithm, RootLPMaxIter

- **Integer Programming Preprocessing**
  - IPPreprocess
  - CutGeneration, CutGenMaxIter

- **Cut Generation**
  - Heuristics, HeuristicsMaxNodes

- **Heuristics**
  - BranchingRule, LPMaxIter, MaxNodes, MaxNumFeasPoints, NodeSelection

- **Branch and Bound**
Recap

- Found 20 assets that best represent universe
- Large optimization problem: >100k binary variables
- `intlinprog` algorithm consists of several steps
  - Change option values to tune solver
Key takeaways

- `intlinprog` function for MILP (R2014a)
- MILP gives better solutions than rounding LP solutions
- Use auxiliary variables to expand scope of problems
- Can solve large problems, but performance is problem dependent