Previous: (1) The Optimisation Process

Next: (3a) Installing Python at Home

(2) Optimisation Concepts

Linear Programing

The simplest type of mathematical programme is a *linear programme*. For your mathematical programme to be a linear programme you need the following conditions to be true:

1. The decision variables must be real variables;
2. The objective constraint must be a linear expression;
3. The constraints must be linear expressions.

Linear expressions are any expression of the form

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \{\leq = \geq\} b$$

where $a_1, a_2, , a_n$ and b are known quantities and $x_1, x_2, , x_n$ are variables. The process of solving a linear programme is called *linear programming*. Linear programming is done via the Revised Simplex Method (also known as the Primal Simplex Method), the Dual Simplex Method or an Interior Point Method. CPLEX allows you to specify which method you use, but we won't go into further detail here.

Integer Programing

Integer programmes are almost identical to linear programmes with one very important exception. Some of the decision variables in integer programmes may need to have only integer values. The variables are known as integer variables. Since most integer programmes contain a mix of real variables and integer variables they are often known as *mixed integer programmes*. While the change from linear programming is a minor one, the effect on the solution process is enormous. Integer programmes can be very difficult problems to solve and there is a lot of current research finding "good" ways to solve integer programmes. Integer programming (the process of solving a (mixed) integer programme) was originally done using the branch-and-bound process. The *branch* part of the process eliminated non-integer values for integer variables in the following way:

- Initially, all variables are left as real variables. The problem is solved using linear programming;
- If one of the integer variables in the linear programming solution has a fractional value, e.g., $x_i = 4.5$, then the linear programme is split in two and the fractional region eliminated. This is done by *branching* on the variable value, e.g., adding the constraint $x_i \leq 4$ to form one linear programme and $x_i \geq 5$ to form the other.
- By finding the optimal solution in each of these new linear programmes and comparing we can find the optimal solution for the original problem.
- If either of the new linear programmes has a fractional value for an integer variable then a new branch is needed.

This branching process results in the formation of a *branch-and-bound tree* (we will discuss the bounding next). Each node in this tree represents a linear programme consisting of the original linear programme and the extra branches added. Eventually all the *leaf nodes* in the tree will

contain solutions where all the integer variables have integer values and no further branching is needed. All these values can be compared and the best one is the solution to the original integer programme.
Note For MIPs of any reasonable size this tree could be huge, in fact it *grows exponentially* as the number of integer variables increases. The bounding process allows sections of the branch-and-bound tree to be removed from consideration before all the leaf nodes have integer solutions. It relies on the following optimization principle:

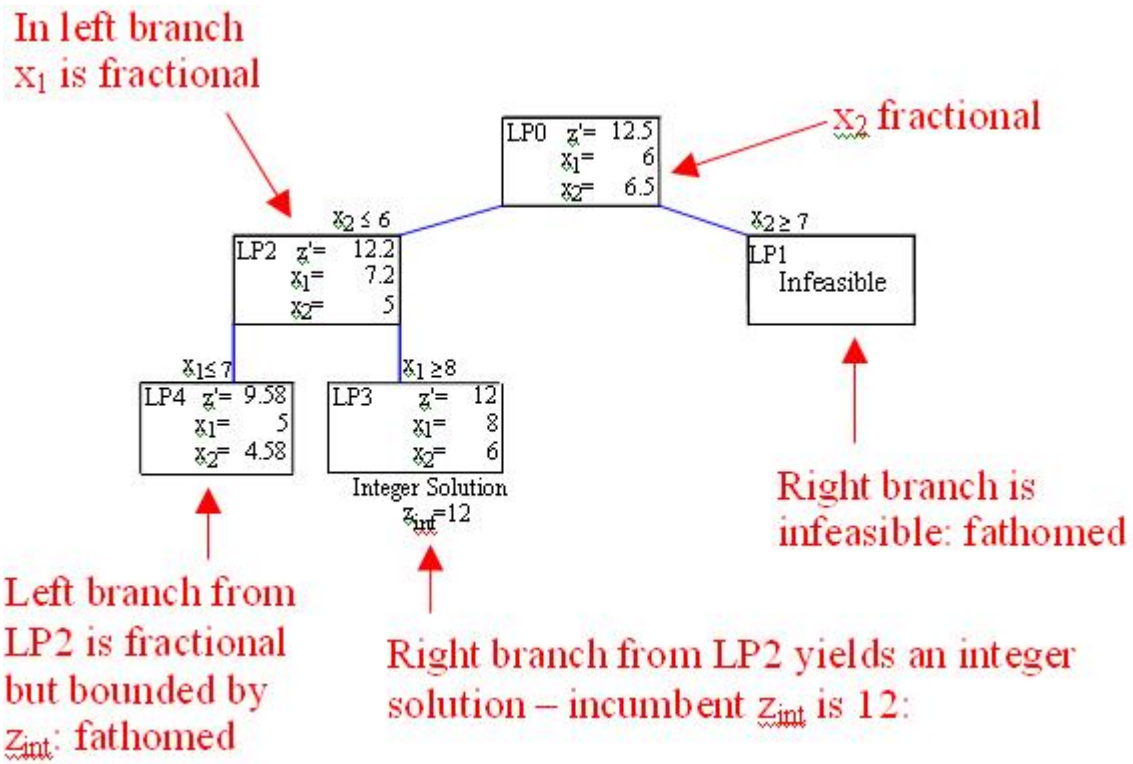
Adding constraints to a mathematical programming will result in a deterioration of the optimal objective value.

This means that adding the branching constraints to the linear programmes at the branch-and-bound tree nodes will mean the resulting nodes will have an optimal objective function value that is equal to or worse than the optimal objective function value of the original linear programme. Thus the objective function values get worse the deeper into the tree you look. Since we are finding the integer solution in the branch-and-bound tree with the best objective value, we can use any integer solutions to bound the tree. The current best *integer* solution is called the *incumbent*. After solving a linear programme at a leaf node of the branch-and-bound tree one of the following conditions holds:

- The linear programme is infeasible (no more branching is possible);
- The linear programme solution is an integer solution with a better objective value than the incumbent. The incumbent is replaced with this new solution;
- The linear programme solution has a worse objective than the incumbent. Any nodes created from this node will also have a worse objective than the incumbent. This node is *bounded* by the incumbent objective;
- The linear programme solution is fractional and has a better objective value than the incumbent. Further branching from this node is necessary to ensure an optimal solution is found.

Only the last condition requires more branching, all the other conditions result in the node becoming *fathomed* and no more branching is required from that node.

Example



The LP Relaxation

The Linear Programming (LP) relaxation is the same as the integer programme, except we "relax" the integer variables to allow them to take fractional values. The integer programme's feasible region lies within the feasible region of the LP relaxation (at points where the integer variables have integer values). Therefore the integer restrictions cause the optimal objective function value to be worse in the integer programme as compared to the LP relaxation. However, if a solution x of the LP relaxation has integer values for the integer variables, then x also solves the integer programme. In some cases, if the solution values for the integer variables are large, then rounding the LP relaxation solution may give a good solution to the integer programme. However, you need to make sure that the rounded solution is not infeasible! For some classes of problem the LP relaxation gives naturally integer solutions:

- An $m \times m$ matrix M is *unimodular* if and only if its determinant $|M|$ is equal to 1 or –1;
- An $m \times n$ matrix M is *totally unimodular* if and only if every $m \times m$ non-singular submatrix of M is unimodular;
- If the constraint matrix A and the right-hand side vector b of a mixed-integer programme are totally unimodular and integer respectively, then the mixed-integer programme is naturally integer and the LP relaxation solution is the optimal solution
    - The transportation problem is an example of one such problem;
    - Most network flow problems are also naturally integer;
    - Some scheduling problems are naturally integer.

When using PuLP, naturally integer variables are defined when the variables are created with the `LpVariable` function. A parameter for this function is either LpContinuous or *LpInteger*.

Master-Slave Constraints

Using zero/one variables, we can control the range of values that other variables take. Suppose that x_{AB} (the amolunt shipped from A to B) is either 0 (we don't ship from A to B) or between 20 and 100 (we ship from A to B with limits specified by the transportation company). We introduce a new 0/1-variable z_{AB} that is 1 if there is a shipment from A to B and 0 otherwise. Then we can use a *master-slave constraints* to let z_{AB} control x_{AB}

$x_{AB} amp;\geq 20 z_{AB}$

$x_{AB} amp;\leq 100 z_{AB}$

Sensitivity Analysis

In STATS/ENGSCI 255 you will have seen the following *sensitivity analysis* output from Excel and/or Storm.

Excel Output

After solving a problem, Excel can generate a sensitivity table as one of its output report sheets.

The sensitivity report for Case Chemicals is shown below. Note that to get all the items shown below, you must tell Excel to 'Assume Linear Model' under Solver Options. Unlike Storm, which gives ranges for values (eg 250 to 1000 for CS-01), Excel gives this information in terms of allowable increases (e.g. 700 on CS-01, i.e. up to 300+700=1000) and decreases (50 on CS-01, i.e. down to 300-50=250).

Changing Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
B2	CS-01	70	0	300	700	50
C2	CS-02	90	0	500	100	350

Constraints

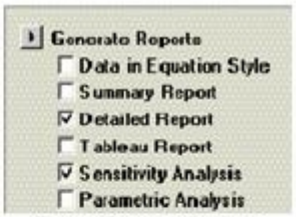
Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
D6	BLENDHRS VALUE	230	33.33333333	230	270	90
D7	PURIHRS VALUE	250	233.3333333	250	45	133
D8	CS02LIM VALUE	90	0	120	1E+30	30

Case Chemicals - Excel Sensitivity Analysis Table

SENSITIVITY ANALYSIS COMPUTER OUTPUT

Storm Output

Once we have found the optimal solution to the Case Chemical problem in Storm, we can choose the Detailed Report and Sensitivity Analysis reports. This gives the output shown below. (Irrelevant sections of the detailed report are not shown.) We explain how to use these tables in the next section.



Storm Report Options

STORM OUTPUT
CASE CHEMICALS
OPTIMAL SOLUTION - DETAILED REPORT

	Constraint	Type	RHS	Slack	Shadow price
1	BLENDHRS	<=	230.0000	0.0000	33.33
2	PURIHRS	<=	250.0000	0.0000	233.33
3	CS2LIM	<=	120.0000	30.0000	0.0000

Objective Function Value = 66000

Case Chemicals - Storm Detailed Report (shadow price information)

STORM OUTPUT
CASE CHEMICALS

SENSITIVITY ANALYSIS OF COST COEFFICIENTS

	Variable	Current Coeff.	Allowable Minimum	Allowable Maximum
1	CS01	300.00	250.00	1000.00
2	CS02	500.00	150.00	600.00

SENSITIVITY ANALYSIS OF RIGHT-HAND SIDE VALUES

	Constraint	Type	Current Value	Allowable Minimum	Allowable Maximum
1	BLENDHRS	<=	230.0000	140.0000	500.0000
2	PURIHRS	<=	250.0000	115.0000	295.0000
3	CS2LIM	<=	120.0000	90.0000	Infinity

Case Chemicals - Storm Sensitivity Report

What do these numbers mean? First, let's look at the Excel sensitivity analysis. For the variables, the *Allowable Increase* and *Allowable Decrease* show how much the objective coefficient of that variable can change without changing the optimal solution (although the objective function will change!). For the constraints the Allowable Increase and Allowable Decrease show how much the right-hand side of the constraint (i.e., the part of the constraint that does **not** involve variables) can increase or decrease without changing which variables are non-zero (although the variable values will change!). The *shadow price* gives the amount the objective function changes for each **unit** change in the right-hand side. If the constraint gets *tighter* then the objective function will *deteriorate*. The following table gives the various combinations for constraints and objective functions:

Constraint Relation	Change in Constraint RHS	Objective Type	Change in Objective Function
$\leq$	$-$ (harder)	min	+ (worse)
$\leq$	+ (easier)	min	$-$ (better)
$\leq$	$-$ (harder)	max	$-$ (worse)
$\leq$	+ (easier)	max	+ (better)
$\geq$	$-$ (easier)	min	$-$ (better)
$\geq$	+ (harder)	min	+ (worse)
$\geq$	$-$ (easier)	max	+ (better)
$\geq$	+ (harder)	max	$-$ (worse)

The Storm sensitivity analysis provides the same information, but gives the Allowable Minimum and the Allowable Maximum instead. **Note** that the Allowable Minimum = Current Value -

Allowable Decrease and the Allowable Maximum = Current Value + Allowable Increase.

Parametric Analysis

In STATS/ENGSCI 255 you will have seen the following *parametric analysis*:

```
Case Chemicals
PARAMETRIC ANALYSIS OF RIGHT-HAND SIDE VALUE - BLENDHRS
COEF = 230.000     LWR LIMIT = -1.000E+37  UPR LIMIT = 1.000E+37
     ------- Range -------        Shadow    ---- Variable ----
          From          To         Price       Leave      Enter
RHS      230.000      500.000       33.333       CS-02  SLACK   1
Obj    66000.000    75000.000

RHS      500.000    1.000E+37        0.000  ----  No change  ----
Obj    75000.000    75000.000

RHS      230.000      140.000       33.333   SLACK   3  SLACK   2
Obj    66000.000    63000.000

RHS      140.000      120.000      150.000       CS-01  SLACK   3
Obj    63000.000    60000.000

RHS      120.000        0.000      500.000       CS-02
Obj    60000.000        0.000

RHS        0.000    -Infinity  ---- Infeasible in this range ----
```

This analysis shows how the objective and shadow price change as a right-hand side value increases (from 230 to 500, then 500 upwards. It also shows how the objective and shadow price change as the right-hand side value decreases (from 230 to 140, 140 to 120, 120 to 0, then 0 downwards). Similar tables are also available in STORM for changes in the cost coefficients. These tables assume that only one quantity (right-hand side, cost) is changing.

 Previous: (1) The Optimisation Process

 Next: (3a) Installing Python at Home