Dhaka Univ. J. Sci. 58(2): 233-237, 2010 (July)

A Computer Technique for Sensitivity Analysis in Linear Programs

Roni Saha and M Babul Hasan

Department of Mathematics, Dhaka University, Dhaka - 1000, Bangladesh

Email: mbabulhasan@yahoo.com

Received on 26. 08. 2009. Accepted for Publication on 10. 03.2010.

Abstract

Sensitivity Analysis is a systematic study of how sensitive solutions are to (small) changes in the data. In our paper, we develop a computer oriented method for analyzing sensitivity. This method shows the changes in optimal solution of a Linear Programming problem for a small change in data values. A numerical example is illustrated to demonstrate our algorithm.

I. Introduction

Sensitivity Analysis (SA) is one of the great successes to emerge from operations research and management science. It is well developed and widely used. Linear Programming (LP) problems in practice are often based on numerical data that represent rough approximations of quantities that are inherently difficult to estimate. Because of this, most LP-based studies include a post optimality investigation of how a change in the data affects the solution. SA is the most helpful tool for this purpose.

Linear Programs

We first briefly discuss the general LP problems. Consider the standard LP problem as follows,

$$\text{Maximize} \quad Z = cx \qquad (1.1)$$

$$\text{subject to} \quad Ax = b \qquad (1.2)$$

$$x \geq 0 \qquad (1.3)$$

Where $A = (a_1, a_2, \ldots, a_m, a_{m+1}, \ldots, a_n)$ is an m × n matrix, $b \in R^m$, $x, c \in R^n$. Let $B = (b_1, b_2, \ldots, b_m)$ be any non-singular sub-matrix of A and x_B be the vector of variables associated with the columns of B.

The general properties and solution procedures of LP problems can be found in Kambo [5], Hasan [2]. We will now concentrate our discussion into SA.

Revised Simplex Method

In real life the LP matrices are thin and sparse (there are usually more columns than rows). According to the characteristic of the LP problems and the way Simplex Method (SM) works, it can easily be shown that a significant amount of redundant information is generated at each step. This necessitates extra computational effort and computer storage. To overcome the drawback of the standard method of pivoting, a computationally more efficient procedure Revised Simplex Method (RSM) is presented. The RSM is an implementation of the standard SM that uses an abbreviated table, reconstructing only that data from the complete simplex tables absolutely necessary to perform the steps of the SM. RSM (more efficient for computing) is now used in all commercially available packages (e.g. IBM MPSX, CDC APEX III).

Sensitivity Analysis

Once the optimal solution to an LP problem has been attained, it may be desirable to study how the current solution changes when the parameters of the problem are changed. The change in parameter of the problem may be discrete or continuous. The study of the effect of discrete changes in parameters on the optimal solution is called SA. In analyzing output, SA is used to explore how changes in the problem data might change the solution to an LP, for example, how a change in production costs or demand projections might affect a production schedule. Because large-scale planning efforts often rely on large amounts of data, much of which represents best-guess estimates, the ability to undertake such SA is critical to the acceptance of the methodology. Indeed, people who are uncertain about data elements are often advised to use SA to resolve the impact of uncertainty. When we use a mathematical model to describe reality we must make approximations. The world is more complicated than the kinds of optimization problems that we are able to solve. Our knowledge of the relevant technology may be imprecise, forcing us to approximate data values. For this reason Sensitivity Analysis is needed to apply in real life models.

The basic idea is to be able to give answers to the changes:

- Changing the Objective Function Coefficient of a Nonbasic Variable.
- Changing the Objective Function Coefficient of a Basic Variable.
- Changing the Right-Hand Side of a Constraint.
- Changing the Column of a Nonbasic Variable.
- Adding a New Activity.

Some Definitions

Objective Function

The linear function $z = \sum_{j=1}^{n} c_j x_j = c_1x_1 + c_2x_2 + \ldots \ldots \ldots + c_nx_n$ which is to be maximized (or minimized) is called objective function of the LP problem.

Constraints

The set of equations or inequalities is called the constraints of the general LP problem. Ax ($\leq$, $=$, $\geq$) b is the set of constraints in the LP.

Feasible Solution

Any solution to an LP which also satisfies the nonnegative restrictions of the problem is called a feasible solution to the LP.

Basic Solution

A basic solution is a solution obtained by setting (n-m) variables equal to zero and solving for the remaining m variables, provided the determinant of the coefficient of these m variables are non-zero. The m variables are called basic variables.

Basic Feasible Solution

A basic feasible solution is a basic solution which also satisfies the nonnegative restrictions $x_j \geq 0$; that is all basic variables are nonnegative.

Optimal Solution

Any feasible solution which optimizes (minimizes or maximizes) the objective function of an LP is called optimal solution to the LP.

Cost/Profit Coefficient

The vector c $(c_1, c_2, \ldots \ldots, c_n)^T$ which is formed by the coefficients of the objective function is called cost/profit coefficient.

Right Hand Side Constant

The vector b $(b_1, b_2, \ldots \ldots, b_n)^T$ which is formed by the right hand side values of the constraints is called right hand side constant.

Basis Matrix

Basis matrix B is the matrix, whose elements are the original columns corresponding to the basic variables.

II. Algorithm

In this section, we present the algorithm of our computer oriented method.

Start

Input type of problem d, number of variables n, number of constraints m, cost coefficients $c[i]$, cost coefficients of basic variables c_{BV}, basis matrix B, columns of variables $a[i]$, right hand side vector b.

Step 1 calculate B^{-1} and $B^{-1}b$.

Step 2 for $i \leq m$, x = solve equation $B^{-1}b = 0$

if $x \neq \phi$, solve inequality $B^{-1}b \geq 0$

Print inequality solution.

Step 4 calculate $c_{BV}B^{-1}$, for $i \leq n$

if $d = 1$, calculate $c_j[i] = c_{BV}B^{-1}a[i] - c[i]$

else, calculate $c_j[i] = c[i] - c_{BV}B^{-1}a[i]$

Step 6 for $i \leq n$, $M = 0$, x = solve equation $c_j[i] = 0$, clear the value of M

if $x \neq \phi$, solve inequality $c_j[i] \geq 0$

Print inequality solution.

Step 7 calculate $c_{BV}B^{-1}b$.

Output $B^{-1}b$, $c_{BV}B^{-1}b$, $c_j[i]$.

Stop.

III. Mathematica Codes

Now, we present our computer codes of Mathematica for SA. We have written the propram in Mathematica 4 for student version, but it will run on any version of Mathematica. In this program we have used some mathematica codes like as,

Solve[equations, vars] attempts to solve equations for vars.

Take[lst, {m, m}] returns a list consisting of the mth object of lst.

InequalitySolve[inequality, vars] attempts to solve inequality for vars.

If[condition, true, false] evaluates condition and executes true if condition is True and executes false if condition is False.

Do[expr,{i, imin, imax}] evaluates expr with the value of i changing from imin to imax in increments of 1.

While[condition, expr] evaluates condition, then expression, repetitively, until condition is False.

For our program we need an LP problem which is solved by any method. Then our method can be applied to analyze the sensitivity of the solution of the solved problem. Our computer oriented program is presented as follows:

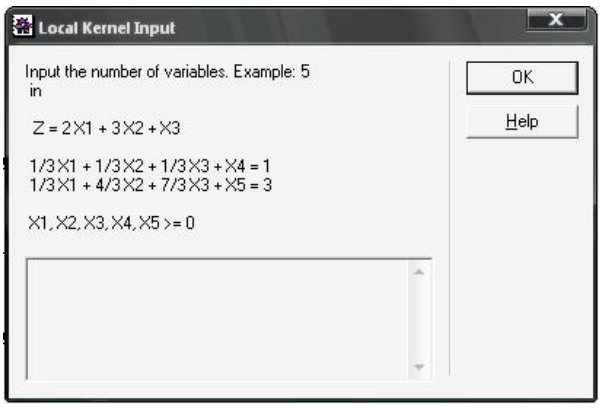
```
<< LinearAlgebra`MatrixManipulation`
<< Algebra`InequalitySolve`
d = Input["d"]; n = Input["n"]; m = Input["m"]; i = 1;
Do[c[i] = Input["c"], {i, 1, n}]; z = Input["z"]; B = Input["B"];
i = 1; Do[a[i] = Input["a"], {i, 1, n}]; b = Input["b"];
IB = Inverse[B]; IBb = IB.Transpose[b]; i = 1;
While[i <= m, x = Solve[Take[IBb, {i, i}] == 0];
  If[x != {{}} && x != {}, Print[N[InequalitySolve[
      (Take[IBb, {i, i}].{1}).{1} >= 0, p]]]; i++];
zB = z.IB; i = 1;
If[d == 1, Do[ca[i] = ((zB.Transpose[a[i]]) - c[i]), {i, 1, n}],
```

```
 Do[ca[i] = (c[i] - (zB.Transpose[a[i]])), {i, 1, n}]];
i = 1; While[i <= n, M = 0; x = Solve[ca[i] == 0];
Clear[M]; If
  [x != {{}} && x != {}, Print[N[InequalitySolve[(ca[i].{1}).{1} >= 0,
      p]]]]; i++]; zBb = zB.Transpose[b];
Print[N[MatrixForm[IBb]] // Simplify]; Print[
N[zBb] // Simplify]; Print[
Array[ca, n] // Simplify // N];
```

We have generalized and designed the previous program for general use. We have used an example in the input box, so that one can easily understand which data values he has to enter in the each input boxes. It makes the program so easy that any one can use this program to analyze the sensitivity of any LP problem. The generalized form of the main program is given below:

```
<< LinearAlgebra`MatrixManipulation`
<< Algebra`InequalitySolve`
d = Input["Input Maximization or Minimization
Problem.\n If Max then input:
 1\n If Min then input:  -1"];
n = Input["Input the number of variables. Example:
5 \n in\n\n  Z = 2 X1 + 3 X2 + X3\n\n 1/3 X1 + 1/3 X2 +
1/3 X3 + X4 =
1\n 1/3 X1 + 4/3 X2 + 7/3 X3 + X5 = 3\n\n X1, X2, X3,
X4, X5 >= 0"];
m = Input["Input the number of constraints.
Example: 2 \n in\n\n  Z = 2 X1
+ 3 X2 + X3\n\n 1/3 X1 + 1/3 X2 + 1/3 X3 + X4 = 1\n
1/3 X1 + 4/3 X2
+ 7/3 X3 + X5 = 3\n\n X1, X2, X3, X4, X5 >= 0"];
i = 1; Do[c[i] = Input["Input the cost coefficent.
Example: 2 \n in\n\n  Z = 2 X1 + 3 X2 + X3\n\n 1/3 X1
+
1/3 X2 + 1/3 X3 + X4 = 1\n 1/3 X1 + 4/3 X2
+ 7/3 X3 + X5 = 3\n\n X1, X2, X3, X4, X5 >= 0"], {i, 1,
n}];
z = Input["Input the cost coefficents of Basic
variables. Example:
{{2,3}} in\n\n  Z = 2 X1 + 3 X2 + X3\n\n 1/3 X1 + 1/3
X2 + 1/3 X3 +
X4 = 1\n 1/3 X1 + 4/3 X2 + 7/3 X3 + X5 = 3\n\n X1,
X2, X3, X4, X5 >= 0"];
B = Input["Input the Matrix B. Example:
{{1/3,1/3},{1/3,4/3}} \n in\n\n
Z = 2 X1 + 3 X2 + X3\n\n 1/3 X1 + 1/3 X2 + 1/3 X3 +
X4 = 1\n 1/3 X1 +
4/3 X2 + 7/3 X3 + X5 = 3\n\n X1, X2, X3, X4, X5 >=
0"];
i = 1; Do[a[i] = Input["Input the column of variable in
constraints.
Example: {{1/3,1/3}} in \n\n Z = 2 X1 + 3 X2 + X3\n\n
1/3 X1 +
1/3 X2 + 1/3 X3 + X4 = 1\n 1/3 X1 + 4/3 X2 + 7/3 X3 +
X5 =
3\n\n X1, X2, X3, X4, X5 >= 0"], {i, 1, n}]
b = Input["Input the right hand side column b of
constraints. Example:
{{1,3}} in\n\n  Z = 2 X1 + 3 X2 + X3\n\n 1/3 X1 + 1/3
X2 +
1/3 X3 + X4 = 1\n 1/3 X1 + 4/3 X2 + 7/3 X3 + X5 =
3\n\n
X1, X2, X3, X4, X5 >= 0"];
Print["\n## Optimal Solution:" ]; Print["The range of
p is: "];
IB = Inverse[B]; IBb = IB.Transpose[b];
i = 1; While[i <= m, x = Solve[Take[IBb, {i, i}] == 0];
  If[x != {{}} && x != {}, Print[N[InequalitySolve[
      (Take[IBb, {i, i}].{1}).{1} >= 0, p]]]]; i++];
zB = z.IB; i = 1;
If[d == 1, Do[ca[i] = ((zB.Transpose[a[i]]) - c[i]), {i, 1,
n}],
  Do[ca[i] = (c[i] - (zB.Transpose[a[i]])), {i, 1, n}]];
i = 1; While[i <= n, M = 0; x = Solve[ca[i] == 0];
Clear[M]; If
  [x != {{}} && x != {}, Print[N[InequalitySolve[(ca[i].{1}).{1} >= 0,
      p]]]]; i++]; zBb = zB.Transpose[b];
Print["The right hand side column of optimal tableau
is: ",
  MatrixForm[IBb] // Simplify // N]
Print["The Optimum value is:  z = ", N[zBb] //
Simplify]
Print["The Zero Row is:  ", Array[ca, n] // Simplify //
N];
```

Now if we run the program an input box will appear like below:



One need to input some data values in different input boxes one after another. They are type of the problem, number of variables, number of constraints, cost coefficients, cost coefficients of basic variables, basis matrix B, column of constraints, and right hand side of constraints.

After inputting all the data values we find the optimal table of the given LP. Now, if we change any value of the problem by the variable p, our program will show the range of p for which the solution will still remain optimal with the changed values in optimal table.

In the following section we will illustrate our technique with a numerical example.

IV. Numerical Illustrations

In this section, we take an LP problem as follows,

$$max \quad z = 10x_1 + 6x_2 + 4x_3$$

$$s.t. \quad x_1 + x_2 + x_3 \le 100 \quad (labor)$$

$$10x_1 + 4x_2 + 5x_3 \le 600 \quad (material)$$

$$2x_1 + 2x_2 + 6x_3 \le 300 \quad (administration)$$

$$x_1, x_2, x_3 \ge 0$$

Solving by usual simplex method we get the optimal solution as, $x_1 = 33.33$, $x_2 = 66.67$, $x_3 = 0$, $x_4 = 0$, $x_5 = 0$, $x_6 = 100$ and $Z = 733.33$. x_4, x_5 and x_6 are slack variables.

Now if we run our program for this particular problem then the output of our program will be as below,

Output 1

```
## Optimal Solution:

The right hand side column of optimal tableau is: (33.3333
                                                   66.6667
                                                   100.    )

The Optimum value is:  z = {{733.333}}

The Zero Row: {{{0.}}, {{0.}}, {{2.66667}}, {{3.33333}}, {{0.666667}}, {{0.}}}
```

Changing the Objective Function Coefficient of Nonbasic Variable

In example, x_3 (the daily production level of product 3) is a nonbasic variable. The unit profit on product 3 in the objective function is $c[3] = 4$. Now if we replaced $c[3]$ in the objective function by p in our program then the output of the program is as like Output 2.

In output 2, only the coefficient of x_3 in the optimal row is changed. All other values of optimal table remain same. The range of p is $p \le 6.66667$ with no lower bound.

So, if we change the coefficient of x_3 in the objective function within the range $-\infty \le p \le 6.66667$, our solution will be still optimal. We can afford the changes in the unit profit value of product 3 from $-\infty$ *to* 6.6667. But it will not change our maximum profit value. If the unit profit of product 3 decreases, it will not affect the optimality of our previous optimal table, but if increases, the solution will remain optimal to the unit profit value 6.66667. If it increases more then 6.66667, then our solution will lose its optimality.

Output 2

```
## Optimal Solution:

 The range of p is:

p ≤ 6.66667

The right hand side column of optimal tableau is: (33.3333
                                                   66.6667
                                                   100.    )

The Optimum value is:  z = {{733.333}}

The Zero Row: {{{0.}}, {{0.}}, {{6.66667 - 1. p}}, {{3.33333}}, {{0.666667}}, {{0.}}}
```

Changing the Objective Function Coefficient of Basic Variable

x_1, the variable for the daily production level of product 1 is a basic variable in this example. The coefficient of x_1 in the objective function is $c[1] = 10$. If we replace $c[1]$ by p in our program then the output of the program will be as like below Output 3. In Output 3, the right hand side is same as before. Only the optimal row and the value of z are changed. The ranges of p in output 3 are like as $p \geq -6$, $p \leq 15$ and $p \geq 6$. So the minimum range will be $6 \leq p \leq 15$. By this we can say that the solution will be optimal if we change the value of $c[1]$ by any value in this range. Hence in order to maximizing the profit we can afford the changes in the profit value of product 1 from 6 to 15.

Output 3

```
## Optimal Solution:
 The range of p is:
p ≥ -6.
p ≤ 15.
p ≥ 6.
The right hand side column of optimal tableau is: (33.3333, 66.6667, 100.)
The Optimum value is:  z = {{400. + 33.3333 p}}
The Zero Row:
 {{{0.}}, {{0.}}, {{0.166667 (6. + p)}}, {{10. - 0.666667 p}}, {{0.166667 (-6. + p)}}, {{0.}}}
```

By the same way, we can show the sensitivity of the optimal solution of this particular factory problem for the other changes of data values.

V. Conclusion

In our paper, we have developed a computer oriented method for analyzing sensitivity of the optimal solution for any changes of data and for any kind of LP problem. And we have showed sensitivity of the optimal solution of a particular factory problem for some changes in data values. Our program is capable of handling any number of constraints and any number of variables. This method can also be extended for solving any kind of LP problems. We concluded that our technique is more efficient for performing sensitivity analysis.

……..

1. Higle J. L. & S. W. Wallace (July-August 2003), "Sensitivity Analysis and Uncertainty in Linear Programming", Interfaces © 2003 Informs, **33**, **4**, 53–60.
2. Hasan M. B., A. F. M. K. Khan and M. A. Islam (2001), "Basic Solution of Linear System of Equations through Computer Algebra", "Ganit: J. Bangladesh Math", Soc. 21, pp. 1-7.
3. Aucamp D. C. and D. I. Steinberg (June 1982), "The Computation of Shadow Prices in Linear Programming", "The Journal of the Operational Research Society", **33**, **6**, 557-565.
4. Winston W. L., "Operation Research, Application and Algorithms, 3rd edition".
5. Kambo, N.S., 1984, "Mathematical programming technique," Affiliated East-West Press Pvt. Ltd., New Delhi.
6. Dantzig G. B., "Linear Programming and extension", Princeton university press, Princeton, N.J (1962).
7. Ravindran, Phillips and Solberg, "Operation Research Principles and Practice."
8. Eugene D., "Schaum's outlines-Mathematica", Mc-Graw-Hill.
9. Taha H. A., "Operations Research: An introduction, 6th edition", Prentice Hill of India, PVT. LTD., New Delhi (1997).
10. Saul I. Gass, "Linear Programming, Methods and Applications, third edition", World Systems Laboratories, Inc. The American University.