A REPORT ON

STUDY AND IMPLEMENTATION OF DIFFERENT OPTIMIZATION ALGORITHMS USING MATLAB

In partial fulfilment of the course

BITS C331

Computer Project

BY

Chirag R. Agarwal

2009B4A7606G

UNDER THE GUIDANCE OF

Dr. ANIL KUMAR

DEPARTMENT OF MATHEMATICS

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
KK BIRLA GOA CAMPUS
GOA – 403726

ACKNOWLEDGEMENTS

I would like to thank Dr. Lucy Gudino, Instructor in charge of the course Computer Project (BITS C331), BITS Pilani KK Birla Goa Campus. I am very much grateful to Dr. Anil Kumar, Faculty of Department of Mathematics, BITS Pilani KK Birla Goa Campus, and our mentor, for giving me an opportunity to work under him and for his continuous help and guidance in my project. Working on this project has helped me look on the topic in a far deeper and relevant sense, and I am grateful to my project mentor for helping me obtaining that insight. Last but not least I wish to avail myself of this opportunity, express a sense of gratitude to my friends and my beloved parents for their support, strength, help and for everything.

TABLE OF CONTENTS

INTRODUCTION

In mathematics and computational science, mathematical optimization (alternatively, optimization or mathematical programming) refers to the selection of a best element from some set of available alternatives.

In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. More generally, optimization includes finding "best available" values of some objective function given a defined domain, including a variety of different types of objective functions and different types of domains.

An optimization problem can be represented in the following way

Given: a function $f: A \longrightarrow R$ from some subset A of an n dimensional real space R^n to R.

> *Sought:* an element x_0 in A such that $f(x_0) \leq f(x)$ for all x in A ("minimization") or such that $f(x_0) \geq f(x)$ for all x in A ("maximization") subject to some constraints.

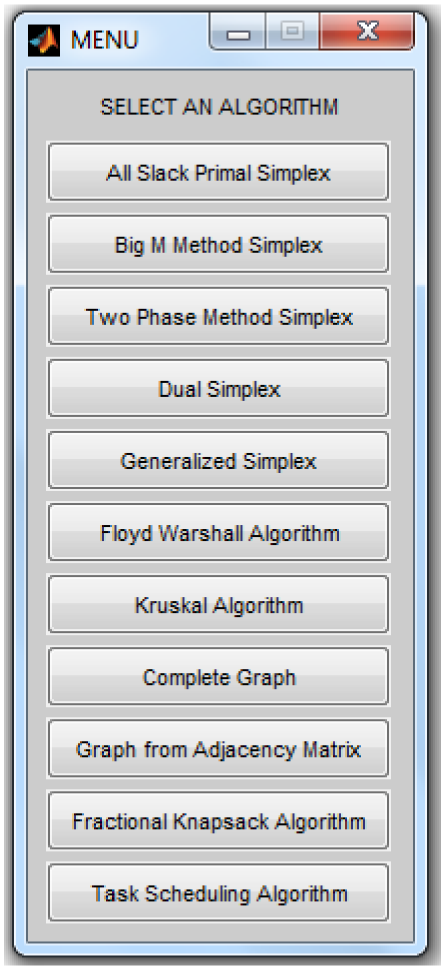
Such a formulation is called an optimization problem or a mathematical programming problem (a term not directly related to computer programming, but still in use for example in linear programming). Many real-world and theoretical problems may be modelled in this general framework. Problems formulated using this technique in the fields of physics and computer vision may refer to the technique as energy minimization, speaking of the value of the function *f* as representing the energy of the system being modelled.

Our motivation for undertaking this project was primarily an interest in various techniques and algorithms related to the vast and ever-growing field of optimization, and thus subsequently undertaking a challenging project in this field. The opportunity to learn about a different field and its implementation on a relevant platform using MATLAB was appealing. The idea of simulating real time design and management applications was alluring. Moreover its applications in fields like Mechanics and Engineering (where it is used for solving problem in rigid body dynamics), Economics (where the utility maximization problem and its dual problem, the expenditure minimization problem are used in microeconomics.), and Operations Research (which uses optimization techniques extensively) made it a current and relevant field to study.

The project incorporates the feature of a GUI i.e. Graphical User Interface, which makes the management, readability and access to data an easy and user-friendly operation. Contrary to various existing programs, this program allows the user to focus more on the data, rather than deal with the intricate ways of the program.

START MENU

The "Start Menu" is provided at the beginning which allows the user to choose from a variety of algorithms, ranging from Simplex type algorithms to Graph plotting and Graph algorithms, to solve whatever problem he has at hand. The menu is as follows:



SIMPLEX ALGORITHM

In mathematical optimization, Dantzig's simplex algorithm (or **simplex method**) is a popular algorithm for linear programming. The journal *Computing in Science and Engineering* listed it as one of the top 10 algorithms of the twentieth century.

The simplex algorithm operates on linear programs in *standard form*, that is linear programming problems of the form,

Minimize $c\,x$

Subject to $Ax = b$, $x \geq 0$

Where $x = (x_1,x_2,\ldots,x_n)^T$ are the variables of the problem, $c = (c_1,c_2,\ldots,c_n)$ are the coefficients of the objective function, A is a $p\times n$ matrix and $b = (b_1,b_2,\ldots,b_p)^T$ are the constants.

ALGORITHM

The simplex algorithm proceeds by performing successive pivot operations which each give an improved basic feasible solution; the choice of pivot element at each step is largely determined by the requirement that this pivot does improve the solution.

PIVOT SECTION:

The geometrical operation of moving from a basic feasible solution to an adjacent basic feasible solution is implemented as a *pivot operation*. First, a nonzero *pivot element* is selected in a non-basic column. The row containing this element is multiplied by its reciprocal to change this element to 1, and then multiples of the row are added to the other rows to change the other entries in the column to 0. The result is that, if the pivot element is in row *r*, then the column becomes the *r*th column of the identity matrix. The variable for this column is now a basic variable, replacing the variable which corresponded to the *r*-th column of the identity matrix before the operation. In effect, the variable corresponding to the pivot column enters the set of basic variables and is called the *entering variable*, and the variable being replaced leaves the set of basic variables and is called the *leaving variable*. The tableau is still in canonical form but with the set of basic variables changed by one element.

ENTERING VARIABLE SELECTION:

If there is more than one column so that the entry in the objective row is positive then the choice of which one to add to the set of basic variables is somewhat arbitrary and several *entering variable choice rules*[1] have been developed.

If all the entries in the objective row are less than or equal to 0 then no choice of entering variable can be made and the solution is in fact optimal. It is easily seen to be optimal since the objective row now corresponds to an equation of the form.

Note that by changing the entering variable choice rule so that it selects a column where the entry in the objective row is negative, the algorithm is changed so that it finds the maximum of the objective function rather than the minimum.

LEAVING VARIABLE SELECTION:

Once the pivot column has been selected, the choice of pivot row is largely determined by the requirement that resulting solution will be feasible. First, only positive entries in the pivot column are considered since this guarantees that the value of the entering variable will be nonnegative. If there are no positive entries in the pivot column then the entering variable can take any nonnegative value with the solution remaining feasible. In this case the objective function is unbounded below and there is no minimum.

Next, the pivot row must be selected so that all the other basic variables remain positive. A calculation shows that this occurs when the resulting value of the entering variable is at a minimum. In other words, if the pivot column is c, then the pivot row r is chosen so that b_r/a_{cr} is the minimum over all r so that $a_{cr} > 0$. This is called the *minimum ratio test*. If there is more than one row for which the minimum is achieved then a *dropping variable choice rule* can be used to make the determination.

DEGENERACY: STALLING AND CYCLING:

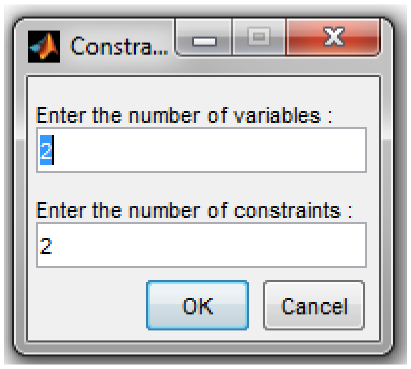
If the values of all basic variables are strictly positive, then a pivot must result in an improvement in the objective value. When this is always the case no set of basic variables occurs twice and the simplex algorithm must terminate after a finite number of steps. Basic feasible solutions where at least one of the *basic* variables is zero are called *degenerate* and may result in pivots for which there is no improvement in the objective value. In this case there is no actual change in the solution but only a change in the set of basic variables. When several such pivots occur in succession, there is no improvement; in large industrial applications, degeneracy is common and such "*stalling*" is notable.

EFFICIENCY:

The simplex method is remarkably efficient in practice and was a great improvement over earlier methods such as Fourier–Motzkin elimination. However, in 1972, Klee and Minty] gave an example showing that the worst-case complexity of simplex method as formulated by Dantzig is exponential time. Since then, for almost every variation on the method, it has been shown that there is a family of linear programs for which it performs badly. It is an open question if there is a variation with polynomial time, or even sub-exponential worst-case complexity. The simplex algorithm has polynomial-time average-case complexity under various probability distributions, with the precise average-case performance of the simplex algorithm depending on the choice of a probability distribution for the random matrices.

MATLAB IMPLEMENTATION: PRIMAL SIMPLEX

This algorithm deals with all the '≤' inequalities and finds the optimal solution for a given system of inequations. At the start, the number of constraints and variables are asked for:


Constra...

Enter the number of variables :

2

Enter the number of constraints :

2

Consider the following example:

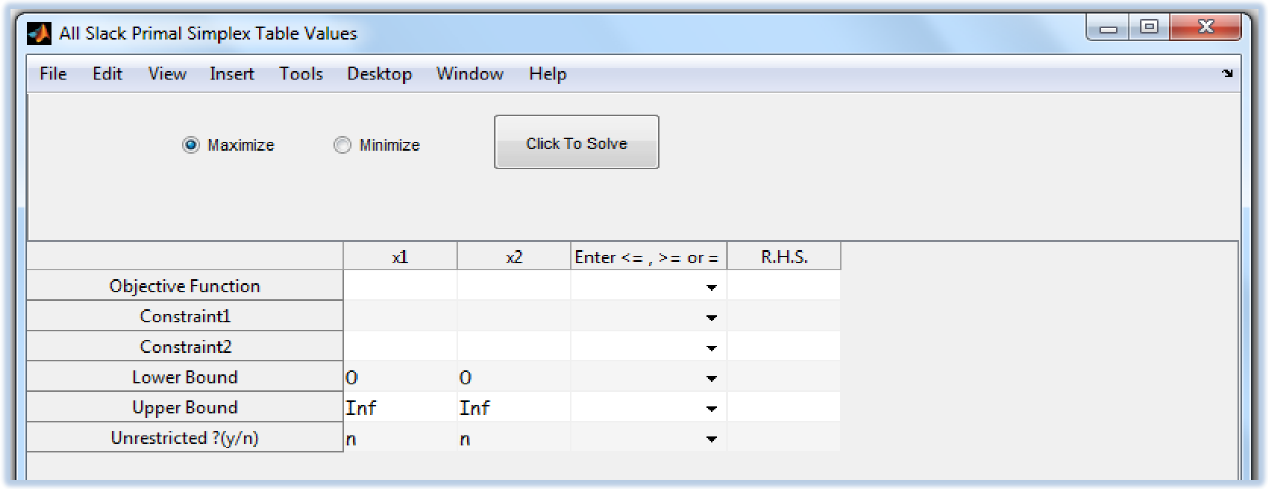
Minimize: $z = 8x_1 - 2x_2$

Subject to: $-4x_1 + 2x_2 \leq 1$

$5x_1 - 4x_2 \leq 3$

$x_i \geq 0 , i=1,2$

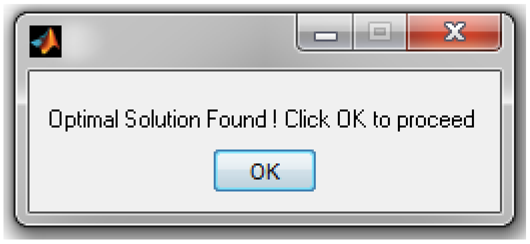
An input GUI is presented to enter the values:


All Slack Primal Simplex Table Values

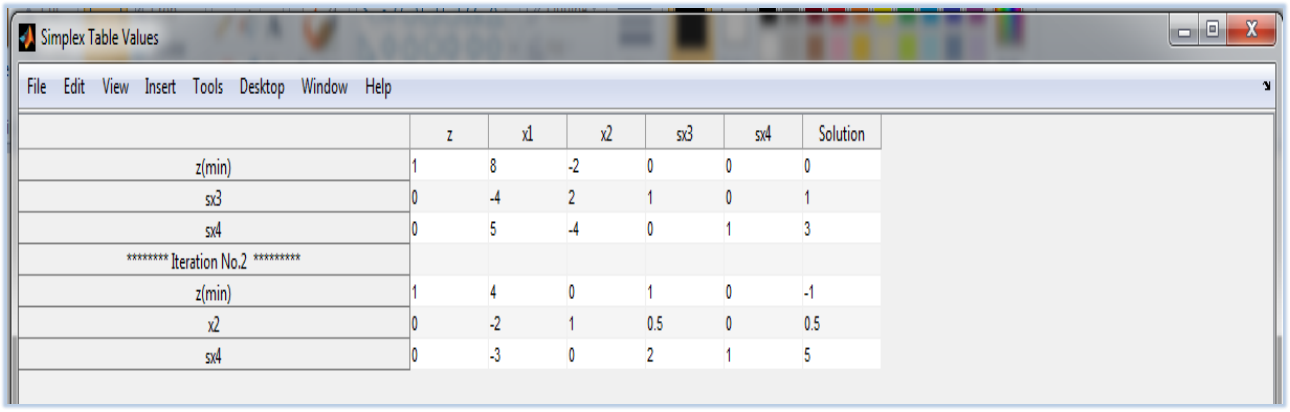
(● Maximize) ○ Minimize

	x1	x2	Enter <= , >= or =	R.H.S.
Objective Function				
Constraint1				
Constraint2				
Lower Bound	0	0		
Upper Bound	Inf	Inf		
Unrestricted ?(y/n)	n	n		

After entering the values, and clicking the "Click To Solve" button, the simplex tableaus are generated. A dialogue box is also made available to notify the user about the solution achieved.


Optimal Solution Found ! Click OK to proceed

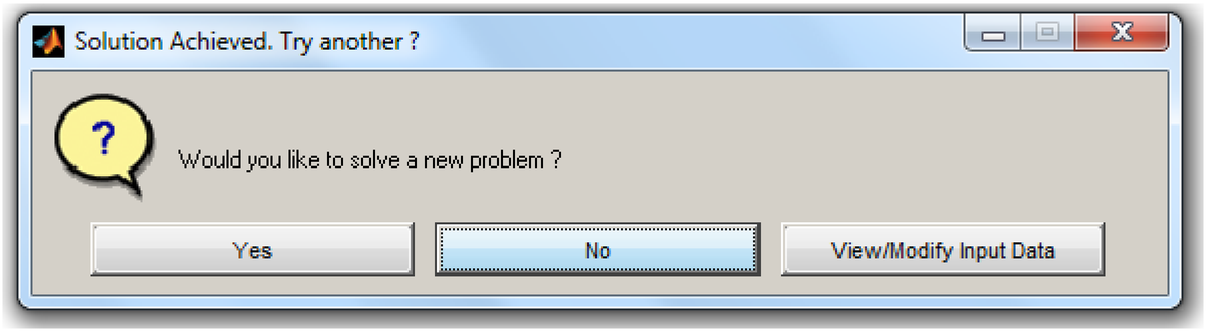
On clicking "Ok", the simplex tableaus are shown as follows:



	z	x1	x2	sx3	sx4	Solution
z(min)	1	8	-2	0	0	0
sx3	0	-4	2	1	0	1
sx4	0	5	-4	0	1	3
******* Iteration No.2 ********						
z(min)	1	4	0	1	0	-1
x2	0	-2	1	0.5	0	0.5
sx4	0	-3	0	2	1	5

The final optimal solution is z (min) = -1, for $x_2 = 0.5$, $x_1 = 0$, as shown in the above table

Also a provisional dialogue box is made to solve another problem, or to modify the given input data, if required by the user:


Solution Achieved. Try another ?

Would you like to solve a new problem ?

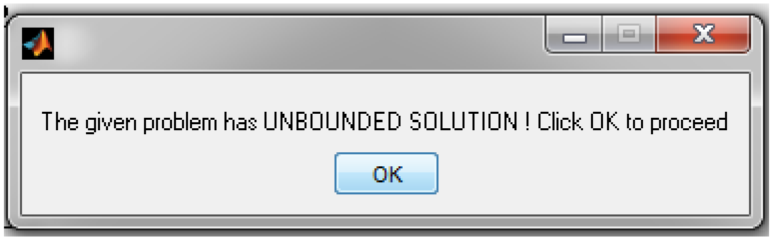
DEALING WITH SPECIAL CASES

UNBOUNDED SOLUTION:

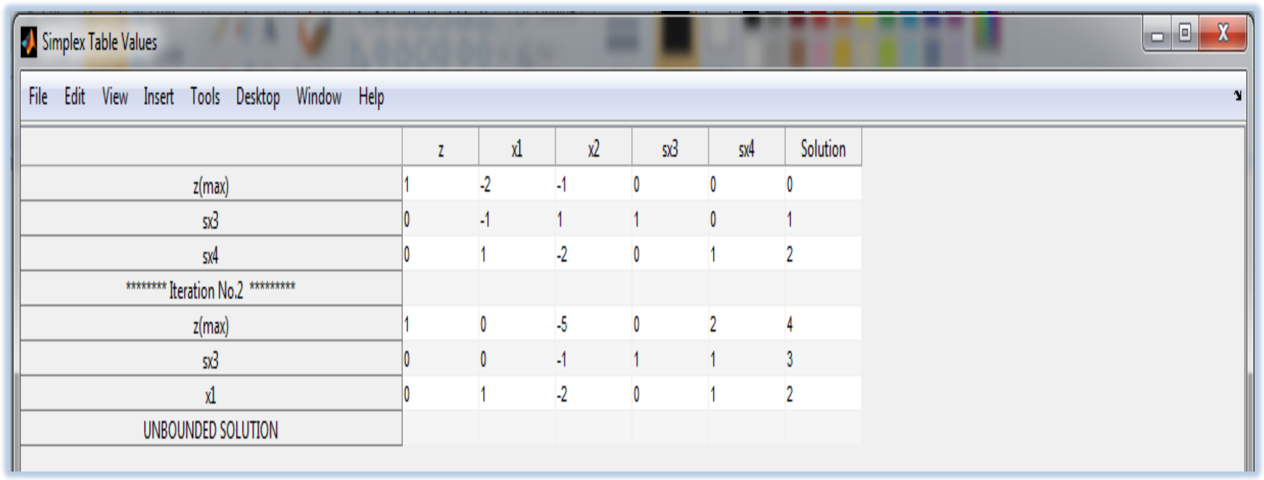
Consider the following example:

$$\begin{aligned} \text{Minimize:} \quad & z = -2x_1 - x_2 \\ \text{Subject to:} \quad & -x_1 + x_2 \leq 1 \\ & x_1 - 2x_2 \leq 2 \\ & x_i \geq 0 \text{ , } i=1,2 \end{aligned}$$

After entering the values in the table shown before, an appropriate dialogue box notifying the user about the existence of UNBOUNDED SOLUTION is shown:


The given problem has UNBOUNDED SOLUTION ! Click OK to proceed

On clicking "Ok", the steps till the Unbounded solution is achieved are shown as follows:



	z	x1	x2	sx3	sx4	Solution
z(max)	1	-2	-1	0	0	0
sx3	0	-1	1	1	0	1
sx4	0	1	-2	0	1	2
******* Iteration No.2 *********						
z(max)	1	0	-5	0	2	4
sx3	0	0	-1	1	1	3
x1	0	1	-2	0	1	2
UNBOUNDED SOLUTION						

<u>ALTERNATE SOLUTION</u>:

Consider the following example:

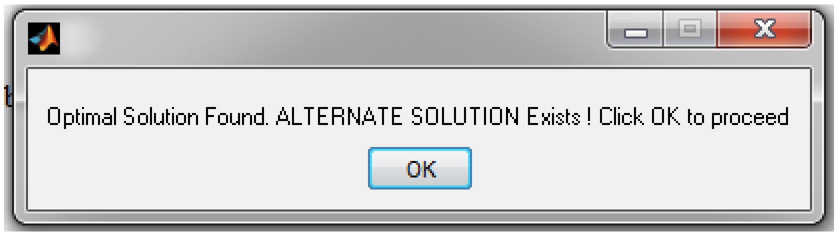
$$\text{Minimize : } z = -x_1 - 0.5x_2$$

$$\text{Subject to : } 2x_1 + x_2 \leq 4$$

$$x_1 + 2x_2 \leq 3$$

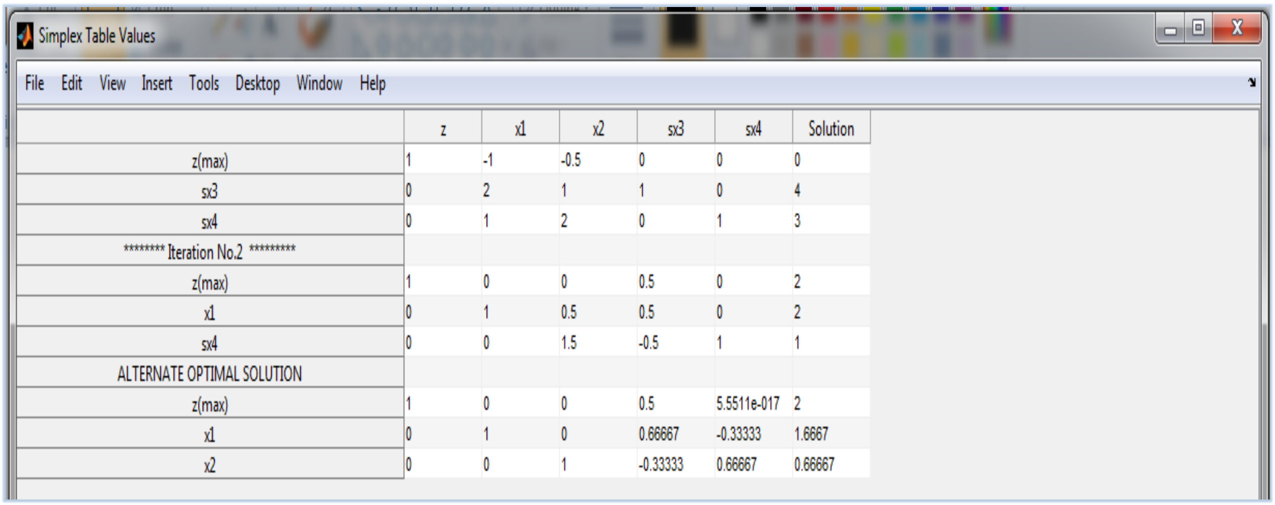
$$x_i \geq 0 \text{ , } i=1,2$$

After entering the values in the table shown before, an appropriate dialogue box notifying the user about the existence of ALTERNATE SOLUTION is shown:



Optimal Solution Found. ALTERNATE SOLUTION Exists ! Click OK to proceed

On clicking "Ok", the steps till the actual solution is achieved are shown. Also the Alternate Solution is shown in the next iteration as follows:



	z	x1	x2	sx3	sx4	Solution
z(max)	1	-1	-0.5	0	0	0
sx3	0	2	1	1	0	4
sx4	0	1	2	0	1	3
******** Iteration No.2 *********						
z(max)	1	0	0	0.5	0	2
x1	0	1	0.5	0.5	0	2
sx4	0	0	1.5	-0.5	1	1
ALTERNATE OPTIMAL SOLUTION						
z(max)	1	0	0	0.5	5.5511e-017	2
x1	0	1	0	0.66667	-0.33333	1.6667
x2	0	0	1	-0.33333	0.66667	0.66667

BOUNDED SIMPLEX:

Consider the following example:

$$\text{Maximize:}\quad z = 3x_1 + x_2 + x_3 + x_4$$

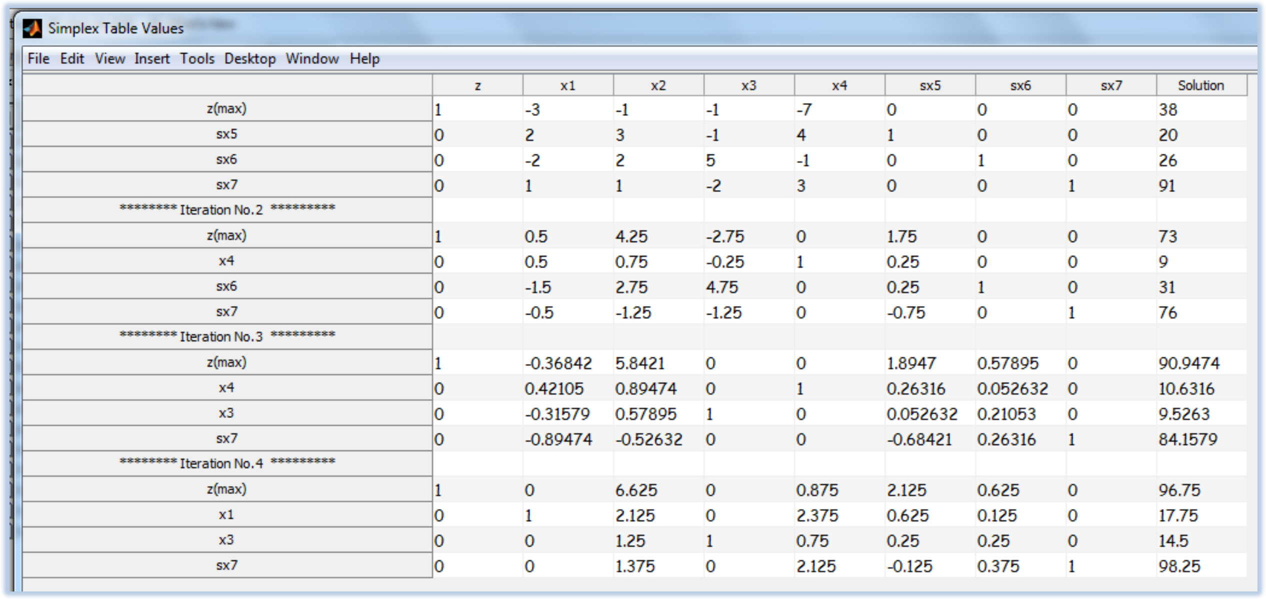
$$\text{Subject to:}\quad 2x_1 + 3x_2 - x_3 + 4x_4 \leq 40$$

$$-2x_1 + 2x_2 + 5x_3 - x_4 \leq 35$$

$$x_1 + x_2 - 2x_3 + 3x_4 \leq 100$$

$$x_1 \geq 2,\ x_2 \geq 1,\ x_3 \geq 3,\ x_4 \geq 4$$

After entering the values, and clicking the "Click To Solve" button, the simplex tableau is generated. A dialogue box is also made available to notify the user about the solution achieved.



	z	x1	x2	x3	x4	sx5	sx6	sx7	Solution
z(max)	1	-3	-1	-1	-7	0	0	0	38
sx5	0	2	3	-1	4	1	0	0	20
sx6	0	-2	2	5	-1	0	1	0	26
sx7	0	1	1	-2	3	0	0	1	91
******** Iteration No.2 *********									
z(max)	1	0.5	4.25	-2.75	0	1.75	0	0	73
x4	0	0.5	0.75	-0.25	1	0.25	0	0	9
sx6	0	-1.5	2.75	4.75	0	0.25	1	0	31
sx7	0	-0.5	-1.25	-1.25	0	-0.75	0	1	76
******** Iteration No.3 *********									
z(max)	1	-0.36842	5.8421	0	0	1.8947	0.57895	0	90.9474
x4	0	0.42105	0.89474	0	1	0.26316	0.052632	0	10.6316
x3	0	-0.31579	0.57895	1	0	0.052632	0.21053	0	9.5263
sx7	0	-0.89474	-0.52632	0	0	-0.68421	0.26316	1	84.1579
******** Iteration No.4 *********									
z(max)	1	0	6.625	0	0.875	2.125	0.625	0	96.75
x1	0	1	2.125	0	2.375	0.625	0.125	0	17.75
x3	0	0	1.25	1	0.75	0.25	0.25	0	14.5
sx7	0	0	1.375	0	2.125	-0.125	0.375	1	98.25

BIG M METHOD

The Big-M method of handling instances with artificial variables is the “commonsense approach”. Essentially, the notion is to make the artificial variables, through their coefficients in the objective function, so costly or unprofitable that any feasible solution to the real problem would be preferred, unless the original instance possessed no feasible solutions at all. But this means that we need to assign, in the objective function, coefficients to the artificial variables that are either very small (maximization problem) or very large (minimization problem); whatever this value, let us call it **Big M**. In fact, this notion is an old trick in optimization in general; we simply associate a penalty value with variables that we do not want to be part of an ultimate solution (unless such an outcome is unavoidable).

Indeed, the penalty is so costly that unless any of the respective variables' inclusion is warranted algorithmically, such variables will never be part of any feasible solution. This method removes artificial variables from the basis. Here, we assign large undesirable (unacceptable penalty) coefficients to artificial variables from the objective function point of view. If the objective function (Z) is to be minimized, then a very large positive price (penalty, M) is assigned to each artificial variable and if Z is to be maximized, then a very large negative price is to be assigned. The penalty will be designated by +M for minimization problem and by –M for a maximization problem and also M>0.

MATLAB IMPLEMENTATION: BIG M METHOD SIMPLEX

Consider the following example:

$$\text{Minimize:} \quad z = 4x_1 + x_2$$

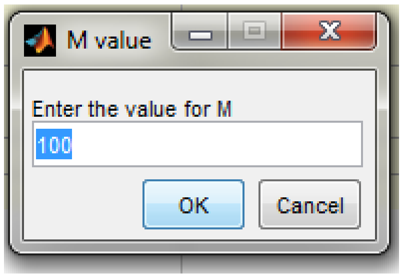
$$\text{Subject to:} \quad 3x_1 + x_2 = 3$$

$$4x_1 + 3x_2 \geq 6$$

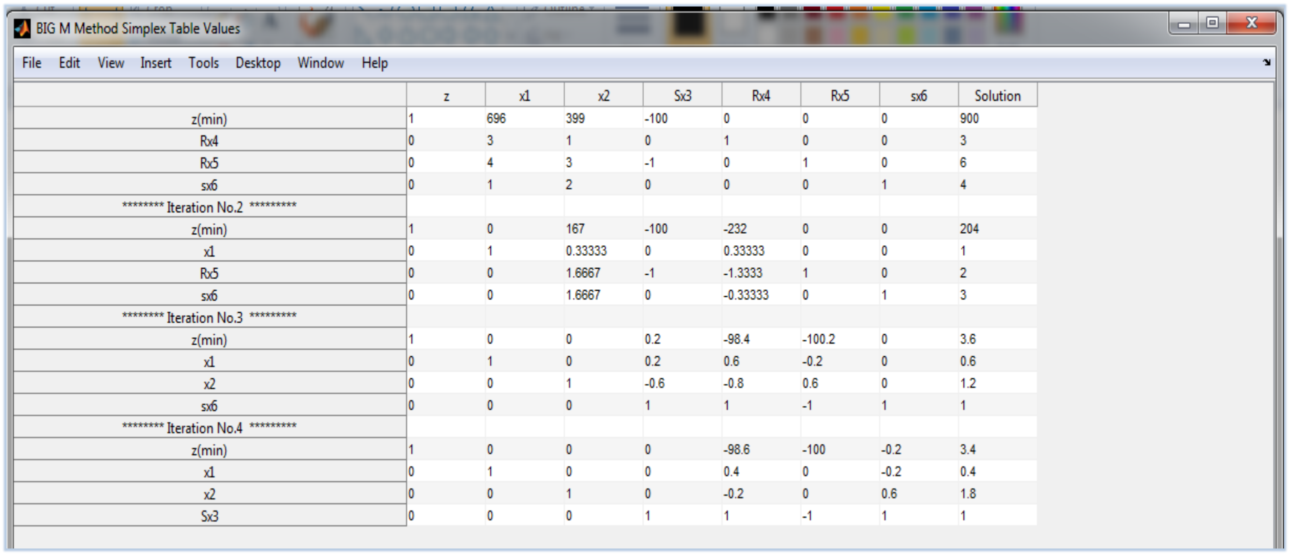
$$x_1 + 2x_2 \leq 4$$

$$x_i \geq 0 \text{ , } i=1,2$$

After entering the values in the table, an additional dialogue box is presented to take the value of "M" from user. The default value is 100.



On clicking "Ok" the final solution is shown as follows:


BIG M Method Simplex Table Values
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	Sx3	Rx4	Rx5	sx6	Solution
z(min)	1	696	399	-100	0	0	0	900
Rx4	0	3	1	0	1	0	0	3
Rx5	0	4	3	-1	0	1	0	6
sx6	0	1	2	0	0	0	1	4
******** Iteration No.2 *********								
z(min)	1	0	167	-100	-232	0	0	204
x1	0	1	0.33333	0	0.33333	0	0	1
Rx5	0	0	1.6667	-1	-1.3333	1	0	2
sx6	0	0	1.6667	0	-0.33333	0	1	3
******** Iteration No.3 *********								
z(min)	1	0	0	0.2	-98.4	-100.2	0	3.6
x1	0	1	0	0.2	0.6	-0.2	0	0.6
x2	0	0	1	-0.6	-0.8	0.6	0	1.2
sx6	0	0	0	1	1	-1	1	1
******** Iteration No.4 *********								
z(min)	1	0	0	0	-98.6	-100	-0.2	3.4
x1	0	1	0	0	0.4	0	-0.2	0.4
x2	0	0	1	0	-0.2	0	0.6	1.8
Sx3	0	0	0	1	1	-1	1	1

DEALING WITH SPECIAL CASES

UNBOUNDED SOLUTION:

Consider the following example:

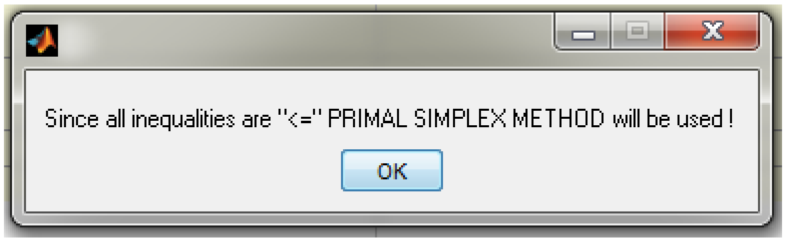
Minimize: $z = -2x_1 - x_2$

Subject to: $-x_1 + x_2 \leq 1$

$x_1 - 2x_2 \leq 2$

$x_i \geq 0$, i=1,2

As all the inequalities are "<=", there is no need to add any Artificial variables. So the given problem can be solved using the Primal Simplex Algorithm. A dialogue box notifying this is shown as follows:



Since all inequalities are "<=" PRIMAL SIMPLEX METHOD will be used !

The final solution is the same as shown previously in the Primal Simplex Algorithm.

ALTERNATE SOLUTION:

Consider the following example:

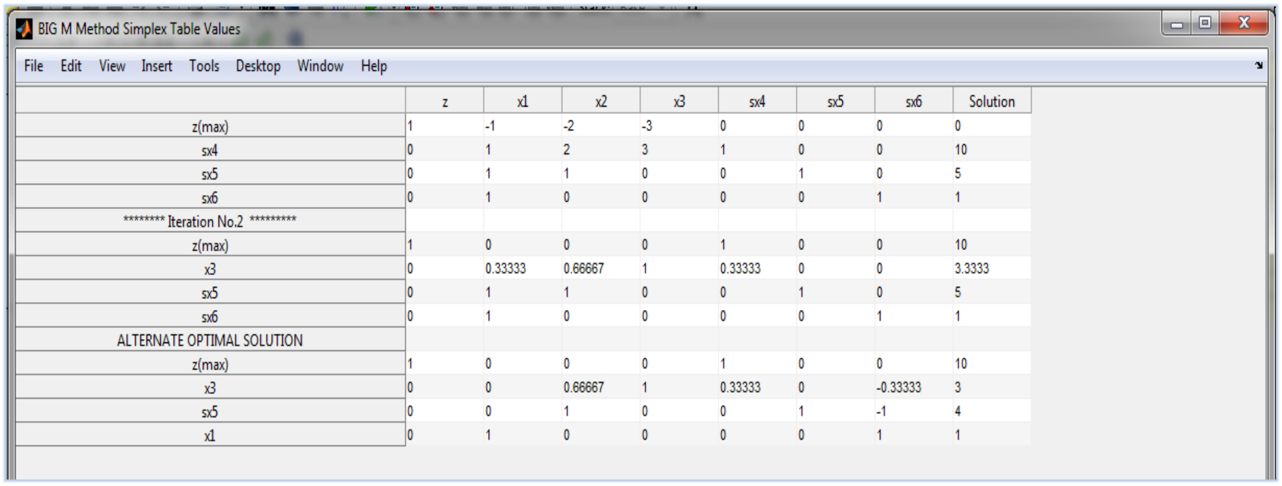
Maximize: $z = x_1 + 2x_2 + 3x_3$

Subject to: $x_1 + 2x_2 + 3x_3 \leq 10$

$x_1 + x_2 \leq 5$

$x_1 \leq 1$; $x_i \geq 0$, i=1,2,3

Upon entering the values in the table, the final solution along with the Alternate Solution is shown as follows:


BIG M Method Simplex Table Values
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	x3	sx4	sx5	sx6	Solution
z(max)	1	-1	-2	-3	0	0	0	0
sx4	0	1	2	3	1	0	0	10
sx5	0	1	1	0	0	1	0	5
sx6	0	1	0	0	0	0	1	1
******* Iteration No.2 ********								
z(max)	1	0	0	0	1	0	0	10
x3	0	0.33333	0.66667	1	0.33333	0	0	3.3333
sx5	0	1	1	0	0	1	0	5
sx6	0	1	0	0	0	0	1	1
ALTERNATE OPTIMAL SOLUTION								
z(max)	1	0	0	0	1	0	0	10
x3	0	0	0.66667	1	0.33333	0	-0.33333	3
sx5	0	0	1	0	0	1	-1	4
x1	0	1	0	0	0	0	1	1

<u>INFEASIBLE SOLUTION</u>:

Consider the following example:

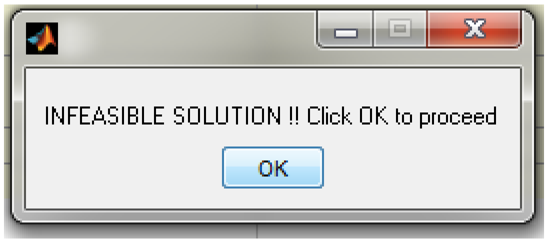
Minimize: $z = 3x_1 + 2x_2$

Subject to: $2x_1 + x_2 \leq 2$

$3x_1 + 4x_2 \leq 12$

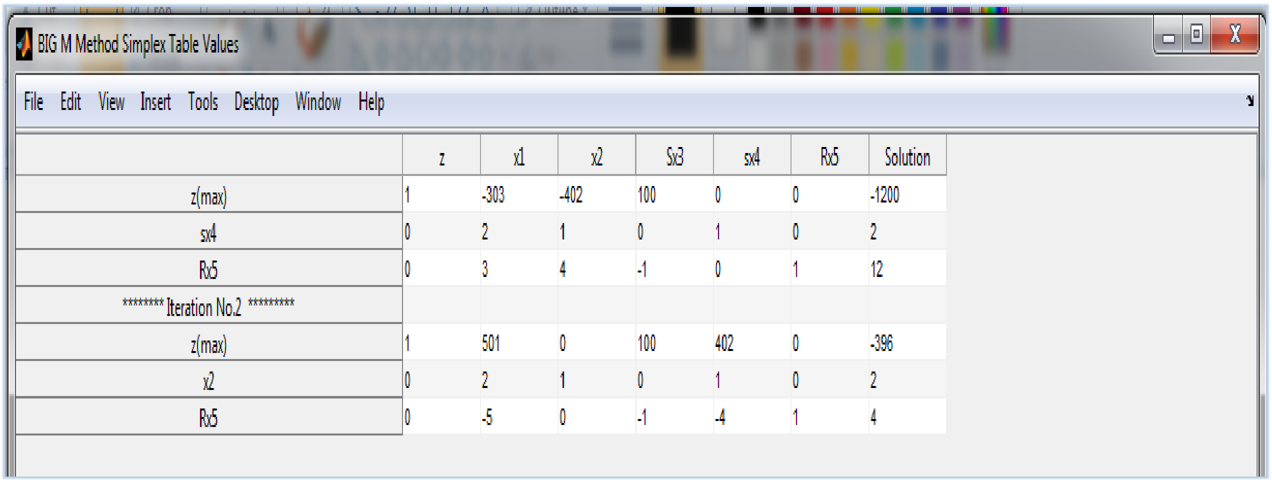
$x_i \geq 0 , i=1,2$

Upon entering the values in the table, an appropriate dialogue box, notifying the user about the existence of Infeasible Solution is shown as follows:



INFEASIBLE SOLUTION !! Click OK to proceed

On clicking “Ok”, the final table till the Infeasibility is achieved, is shown as follows:


BIG M Method Simplex Table Values
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	Sx3	sx4	Rx5	Solution
z(max)	1	-303	-402	100	0	0	-1200
sx4	0	2	1	0	1	0	2
Rx5	0	3	4	-1	0	1	12
******** Iteration No.2 *********							
z(max)	1	501	0	100	402	0	-396
x2	0	2	1	0	1	0	2
Rx5	0	-5	0	-1	-4	1	4

TWO PHASE METHOD

Apart from the BIG M Method, there is another standard method for handling artificial variables within the simplex method, which is called the two phase method.

When a basic feasible solution is not readily available, the two-phase simplex method may be used as an alternative to the big M method. In the two-phase simplex method, we add artificial variables to the same constraints as we did in big M method. Thereafter we find a basic feasible solution to the original LP by solving the Phase I LP. In the Phase I LP, the objective function is to minimize the sum of all artificial variables. At the completion of Phase I, we reintroduce the original LP's objective function and determine the optimal solution to the original LP. Therefore, if are dividing the procedure in various steps:

Step 1

Modify the constraints so that the right-hand side of each constraint is nonnegative. This requires that each constraint with a negative right-hand side be multiplied through by -1.

Step 2

Identify each constraint that is now an = or ≥ constraint. In step 3, we will add the artificial variable to each of these constraints.

Step 3

Convert each inequality constraint to standard form.
For ≤ constraint *i*, we add a slack variable s_i;
For ≥ constraint *i*, we add an excess variable e_i;

Step 4

For now, ignore the original LP's objective function. Instead solve an LP whose objective function is min w'= (sum of all the artificial variables). This is called the **Phase I LP**. The act of solving the phase I LP will force the artificial variables to be zero.

Since each $a_i \geq 0$, solving the Phase I LP will result in one of the following three cases:

Case 1

The optimal value of w' is greater than zero. In this case, the original LP has no feasible solution.

Case 2

The optimal value of w' is equal to zero, and no artificial variables are in the optimal Phase I basis. In this case, we drop all columns in the optimal Phase I tableau that corresponds to the artificial variables. We now combine the original objective function with the constraints from the optimal Phase I tableau. This yields the Phase II LP. The optimal solution to the Phase II LP is the optimal solution to the original LP.

Case 3

The optimal value of w' is equal to zero and at least one artificial variable is in the optimal Phase I basis. In this case, we can find the optimal solution to the original LP if at the end of Phase I we drop from the optimal Phase I tableau all non-basic artificial variables and any variable from the original problem that has a negative coefficient in row 0 of the optimal Phase I tableau.

As with the Big M method, the column for any artificial variable may be dropped from future tableaus as soon as the artificial variable leaves the basis.

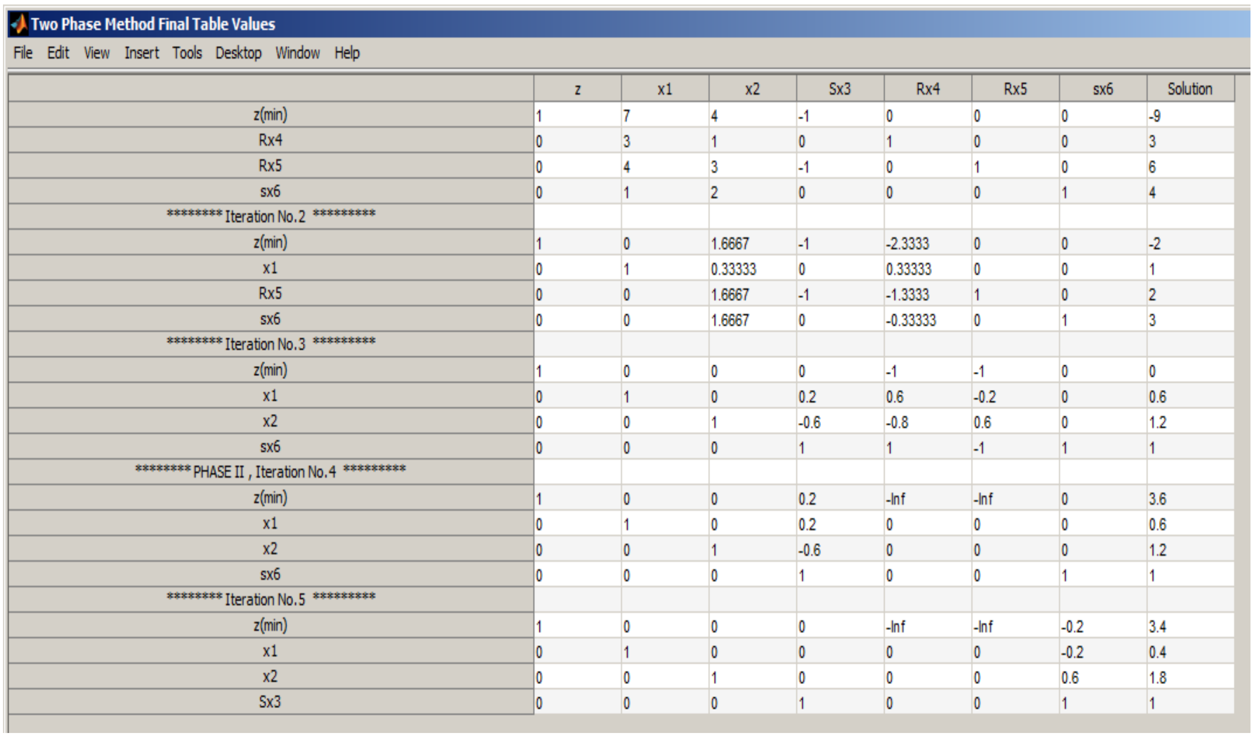
The Big M method and Phase I of the two-phase method make the same sequence of pivots. However, the advantage of the two-phase method is that unlike BIG M method, it does not cause round off errors and other computational difficulties.

MATLAB IMPLEMENTATION: TWO PHASE METHOD

Consider the following example:

$$\begin{aligned} &\text{Minimize:} && z = 4x_1 + x_2 \\ &\text{Subject to:} && 3x_1 + x_2 = 3 \\ &&& 4x_1 + 3x_2 \geq 6 \\ &&& x_1 + 2x_2 \leq 4 \\ &&& x_i \geq 0 \text{ , i=1,2} \end{aligned}$$

After entering the values in the table and clicking on 'Solve' button, we get the final solution table as follows:


Two Phase Method Final Table Values
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	Sx3	Rx4	Rx5	sx6	Solution
z(min)	1	7	4	-1	0	0	0	-9
Rx4	0	3	1	0	1	0	0	3
Rx5	0	4	3	-1	0	1	0	6
sx6	0	1	2	0	0	0	1	4
******** Iteration No.2 *********								
z(min)	1	0	1.6667	-1	-2.3333	0	0	-2
x1	0	1	0.33333	0	0.33333	0	0	1
Rx5	0	0	1.6667	-1	-1.3333	1	0	2
sx6	0	0	1.6667	0	-0.33333	0	1	3
******** Iteration No.3 *********								
z(min)	1	0	0	0	-1	-1	0	0
x1	0	1	0	0.2	0.6	-0.2	0	0.6
x2	0	0	1	-0.6	-0.8	0.6	0	1.2
sx6	0	0	0	1	1	-1	1	1
******** PHASE II , Iteration No.4 *********								
z(min)	1	0	0	0.2	-Inf	-Inf	0	3.6
x1	0	1	0	0.2	0	0	0	0.6
x2	0	0	1	-0.6	0	0	0	1.2
sx6	0	0	0	1	0	0	1	1
******** Iteration No.5 *********								
z(min)	1	0	0	0	-Inf	-Inf	-0.2	3.4
x1	0	1	0	0	0	0	-0.2	0.4
x2	0	0	1	0	0	0	0.6	1.8
Sx3	0	0	0	1	0	0	1	1

As visible, the distinction between the Phase I and Phase II is clearly shown in the table. Note that after Phase I, the z row coefficients of the artificial variables have been assigned very large values such as 'Inf' (in case of maximization problem) and '-Inf' (in case of minimization problem) so as to block them from entering the solution space.

DEALING WITH SPECIAL CASES

INFEASIBLE SOLUTION:

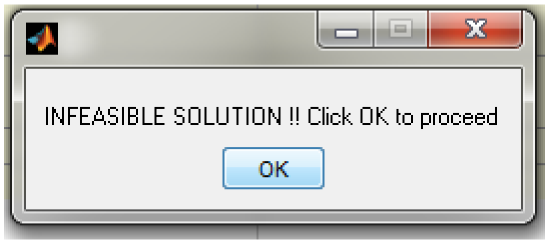
Maximize: $z = 2x_1 + 5x_2$

Subject to: $3x_1 + 2x_2 \geq 6$

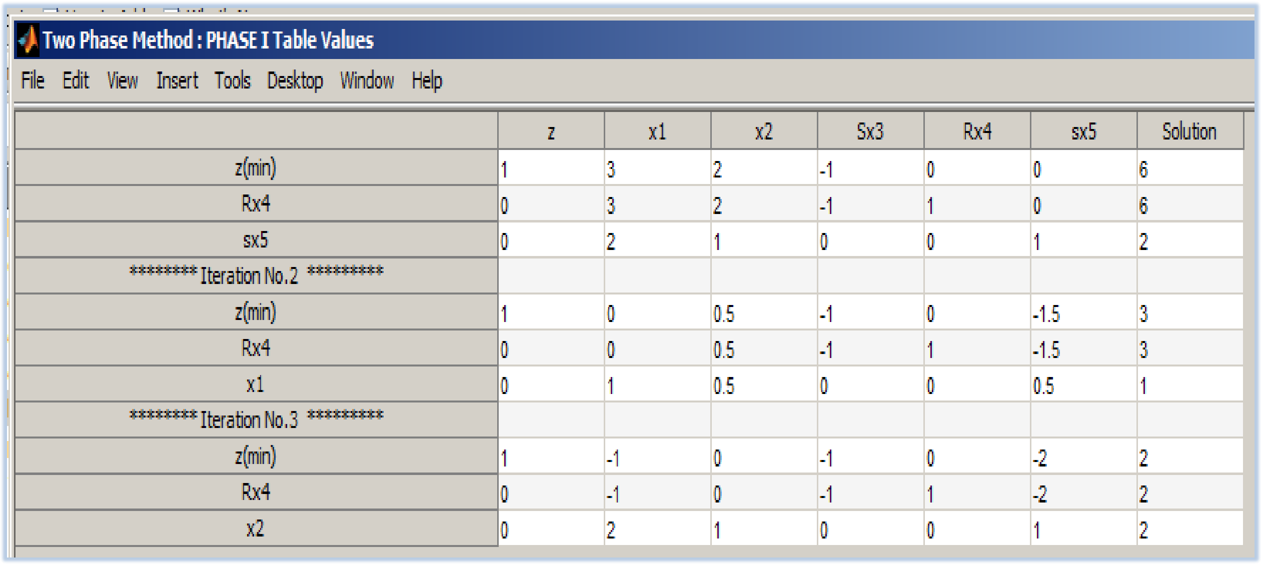
$2x_1 + x_2 \leq 2$

$x_i \geq 0$, i=1,2

After entering the values in the table and clicking on 'Solve' button, an appropriate dialogue box saying that the solution is Infeasible is shown as follows:


INFEASIBLE SOLUTION !! Click OK to proceed

After clicking the 'Ok' button, we get the final solution table as follows:


Two Phase Method : PHASE I Table Values

	z	x1	x2	Sx3	Rx4	sx5	Solution
z(min)	1	3	2	-1	0	0	6
Rx4	0	3	2	-1	1	0	6
sx5	0	2	1	0	0	1	2
********* Iteration No.2 **********							
z(min)	1	0	0.5	-1	0	-1.5	3
Rx4	0	0	0.5	-1	1	-1.5	3
x1	0	1	0.5	0	0	0.5	1
********* Iteration No.3 **********							
z(min)	1	-1	0	-1	0	-2	2
Rx4	0	-1	0	-1	1	-2	2
x2	0	2	1	0	0	1	2

It is clearly visible that because of existence of a non-zero artificial variable as a basic variable in the solution table at termination of Phase I, 'Infeasible Solution' is displayed and the program is terminated.

ARTIFICIAL BASIC VARIABLE IS ZERO AT END OF PHASE I:

Consider the following example:

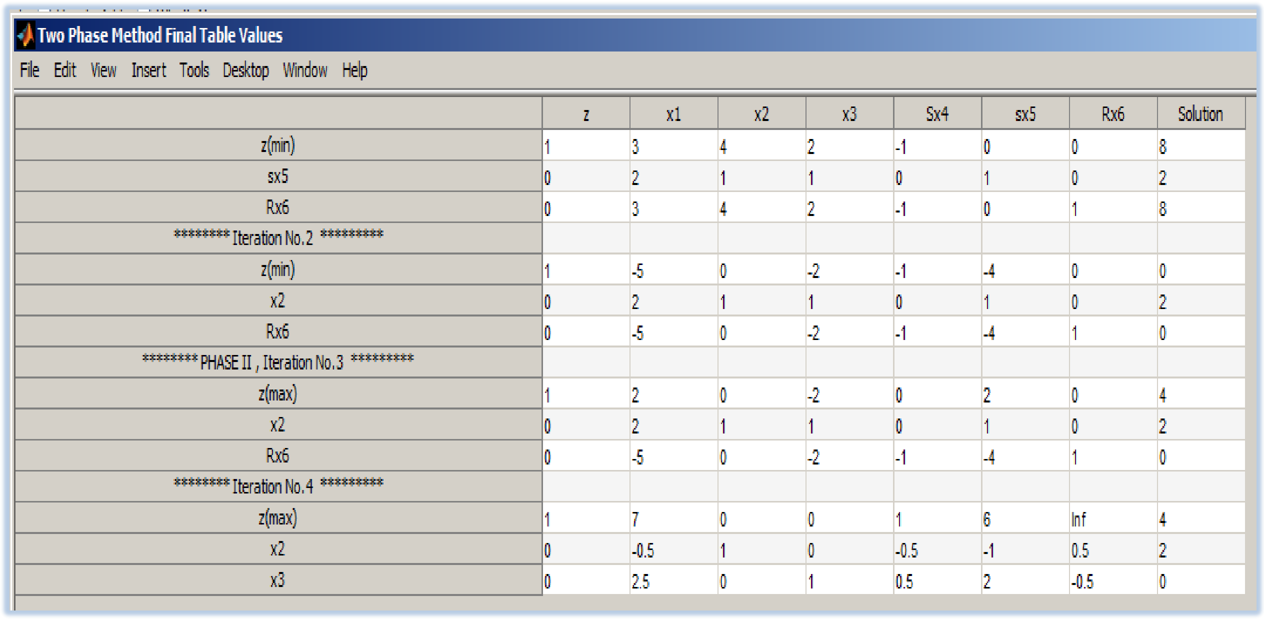
$$\text{Maximize : } \quad z = 2x_1 + 2x_2 + 4x_3$$

$$\text{Subject to : } \quad 2x_1 + x_2 + x_3 \le 2$$

$$3x_1 + 4x_2 + 2x_3 \ge 8$$

$$x_i \ge 0 \text{ , } i=1,2,3$$

After entering the values in the table and clicking on 'Solve' button, we get the final solution table as follows :


Two Phase Method Final Table Values
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	x3	Sx4	sx5	Rx6	Solution
z(min)	1	3	4	2	-1	0	0	8
sx5	0	2	1	1	0	1	0	2
Rx6	0	3	4	2	-1	0	1	8
******** Iteration No.2 *********								
z(min)	1	-5	0	-2	-1	-4	0	0
x2	0	2	1	1	0	1	0	2
Rx6	0	-5	0	-2	-1	-4	1	0
******** PHASE II , Iteration No.3 *********								
z(max)	1	2	0	-2	0	2	0	4
x2	0	2	1	1	0	1	0	2
Rx6	0	-5	0	-2	-1	-4	1	0
******** Iteration No.4 *********								
z(max)	1	7	0	0	1	6	Inf	4
x2	0	-0.5	1	0	-0.5	-1	0.5	2
x3	0	2.5	0	1	0.5	2	-0.5	0

It can be clearly seen that though an artificial variable exists as a basic variable at the end of Phase I, we don't get an Infeasible Solution as its value is zero. In the next iteration, that artificial variable is selected as the leaving variable and thereafter the iterations of the Simplex method proceed as shown.

ALTERNATE UNBOUNDED SOLUTION:

Consider the following example:

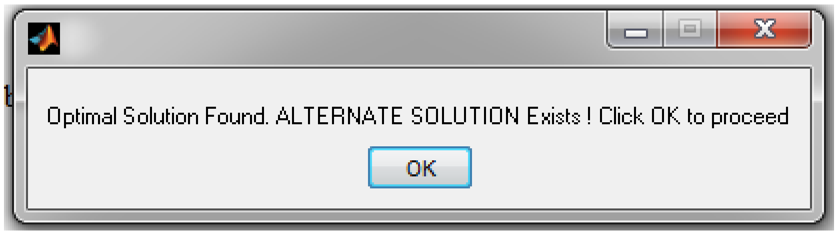
$$\text{Minimize:} \quad z = 3x_1 + 2x_2 + 3x_3$$

$$\text{Subject to:} \quad x_1 + 4x_2 + x_3 \geq 7$$

$$2x_1 + x_2 + x_4 \geq 10$$

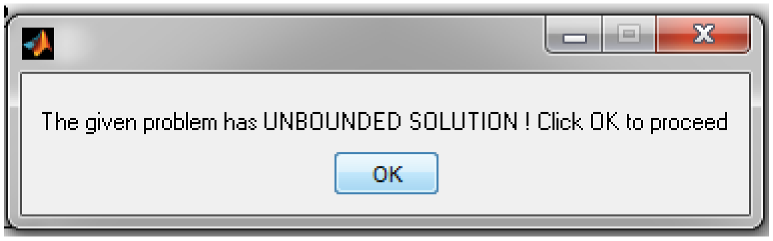
$$x_i \geq 0 \text{ , } i=1,2,3,4$$

After entering the values in the table and clicking on 'Solve' button, we get a dialogue box notifying us about the existence of alternate optimal solution.



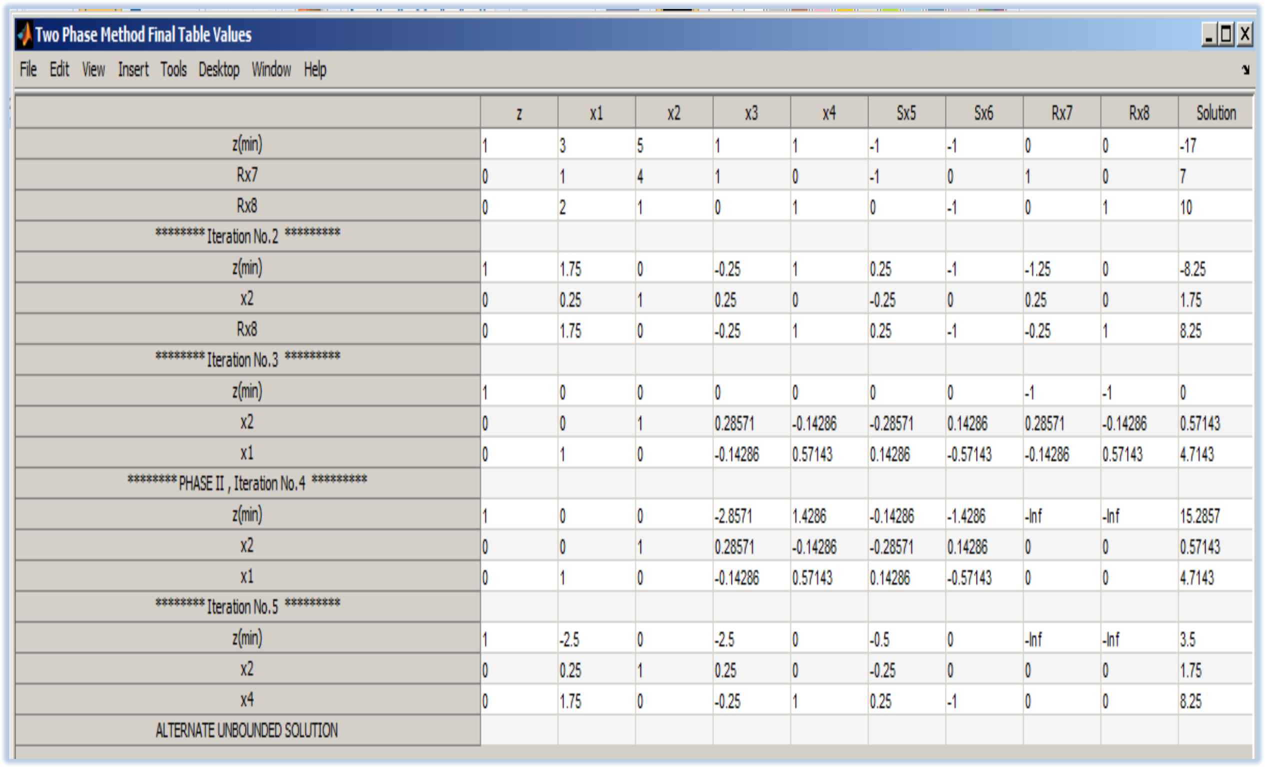
Optimal Solution Found. ALTERNATE SOLUTION Exists ! Click OK to proceed

On clicking 'Ok', if there is an unbounded solution as in this case, an appropriate dialogue box notifying us about it is shown.



The given problem has UNBOUNDED SOLUTION ! Click OK to proceed

Thereafter the final table is displayed as follows:


Two Phase Method Final Table Values
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	x3	x4	Sx5	Sx6	Rx7	Rx8	Solution
z(min)	1	3	5	1	1	-1	-1	0	0	-17
Rx7	0	1	4	1	0	-1	0	1	0	7
Rx8	0	2	1	0	1	0	-1	0	1	10
******** Iteration No.2 *********										
z(min)	1	1.75	0	-0.25	1	0.25	-1	-1.25	0	-8.25
x2	0	0.25	1	0.25	0	-0.25	0	0.25	0	1.75
Rx8	0	1.75	0	-0.25	1	0.25	-1	-0.25	1	8.25
******** Iteration No.3 *********										
z(min)	1	0	0	0	0	0	0	-1	-1	0
x2	0	0	1	0.28571	-0.14286	-0.28571	0.14286	0.28571	-0.14286	0.57143
x1	0	1	0	-0.14286	0.57143	0.14286	-0.57143	-0.14286	0.57143	4.7143
******** PHASE II , Iteration No.4 *********										
z(min)	1	0	0	-2.8571	1.4286	-0.14286	-1.4286	-Inf	-Inf	15.2857
x2	0	0	1	0.28571	-0.14286	-0.28571	0.14286	0	0	0.57143
x1	0	1	0	-0.14286	0.57143	0.14286	-0.57143	0	0	4.7143
******** Iteration No.5 *********										
z(min)	1	-2.5	0	-2.5	0	-0.5	0	-Inf	-Inf	3.5
x2	0	0.25	1	0.25	0	-0.25	0	0	0	1.75
x4	0	1.75	0	-0.25	1	0.25	-1	0	0	8.25
ALTERNATE UNBOUNDED SOLUTION										

BOUNDED VARIABLES:

Consider the following example:

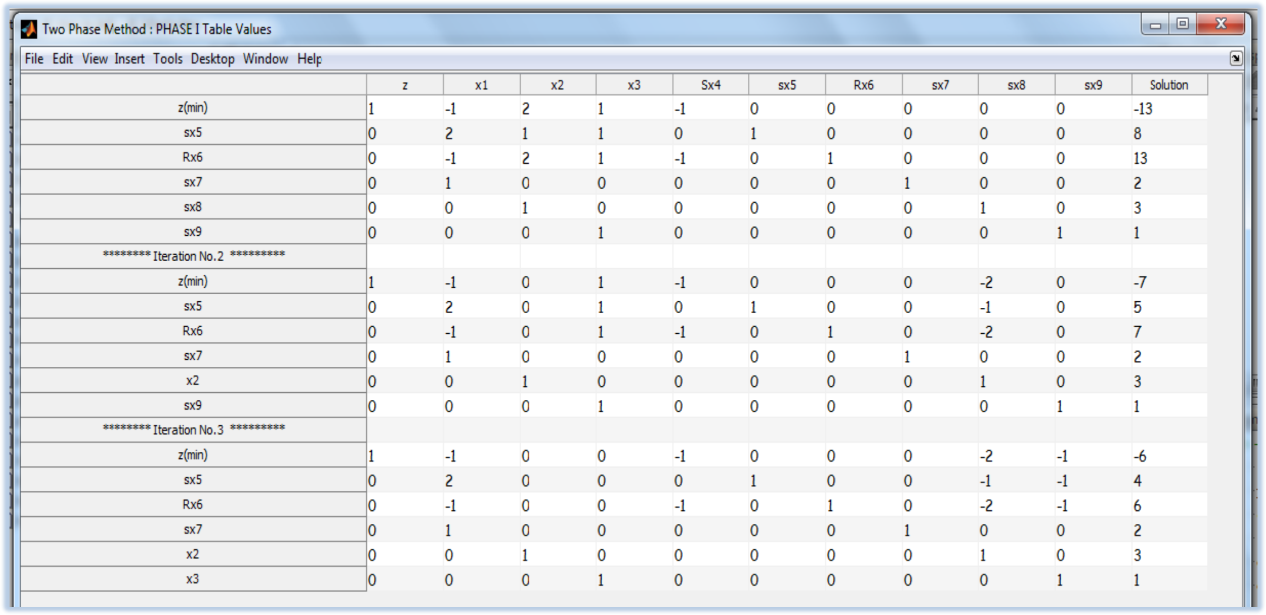
Minimize: $z = -3x_1 - 2x_2 + 2x_3$

Subject to: $2x_1 + x_2 + x_3 \leq 8$

$-x_1 + 2x_2 + x_3 \geq 13$

$0 \leq x_1 \leq 2,\ 0 \leq x_2 \leq 3,\ 0 \leq x_3 \leq 1,$

After entering the values, and clicking the “Click To Solve” button, the simplex tableau is generated.


Two Phase Method : PHASE I Table Values

File Edit View Insert Tools Desktop Window Help

	z	x1	x2	x3	Sx4	sx5	Rx6	sx7	sx8	sx9	Solution
z(min)	1	-1	2	1	-1	0	0	0	0	0	-13
sx5	0	2	1	1	0	1	0	0	0	0	8
Rx6	0	-1	2	1	-1	0	1	0	0	0	13
sx7	0	1	0	0	0	0	0	1	0	0	2
sx8	0	0	1	0	0	0	0	0	1	0	3
sx9	0	0	0	1	0	0	0	0	0	1	1
******** Iteration No.2 *********											
z(min)	1	-1	0	1	-1	0	0	0	-2	0	-7
sx5	0	2	0	1	0	1	0	0	-1	0	5
Rx6	0	-1	0	1	-1	0	1	0	-2	0	7
sx7	0	1	0	0	0	0	0	1	0	0	2
x2	0	0	1	0	0	0	0	0	1	0	3
sx9	0	0	0	1	0	0	0	0	0	1	1
******** Iteration No.3 *********											
z(min)	1	-1	0	0	-1	0	0	0	-2	-1	-6
sx5	0	2	0	0	0	1	0	0	-1	-1	4
Rx6	0	-1	0	0	-1	0	1	0	-2	-1	6
sx7	0	1	0	0	0	0	0	1	0	0	2
x2	0	0	1	0	0	0	0	0	1	0	3
x3	0	0	0	1	0	0	0	0	0	1	1

As one can see, since the artificial variable Rx6 is non-zero at the end of phase 1, the solution is going to be infeasible in nature.

DUAL SIMPLEX ALGORITHM

In the tableau implementation of the primal simplex algorithm, the right-hand-side column is always nonnegative so the basic solution is feasible at every iteration. For purposes of this section, we will say that the basis for the tableau is *primal feasible* if all elements of the right-hand side are nonnegative. Alternatively, when some of the elements are negative, we say that the basis is *primal infeasible*. Up to this point we have always been concerned with primal feasible bases. For the primal simplex algorithm, some elements in row 0 will be negative until the final iteration when the optimality conditions are satisfied. In the event that all elements of row 0 are nonnegative, we say that the associated basis is *dual feasible*. Alternatively, if some of the elements of row 0 are negative, we have a *dual infeasible* basis. As described, the primal simplex method works with primal feasible, but dual infeasible (non-optimal) bases. At the final (optimal) solution, the basis is both primal and dual feasible. Throughout the process we maintain primal feasibility and drive toward dual feasibility. However there is also a variant of the primal approach, known as the dual simplex method, is considered that works in just the opposite fashion. Until the final iteration, each basis examined is primal infeasible (some negative values on the right-hand side) and dual feasible (all elements in row 0 are nonnegative). At the final (optimal) iteration the solution will be both primal and dual feasible. Throughout the process we maintain dual feasibility and drive toward primal feasibility. For a given problem, both the primal and dual simplex algorithms will terminate at the same solution but arrive there from different directions.

The dual simplex algorithm is most suited for problems for which an initial dual feasible solution is easily available. It is particularly useful for re-optimizing a problem after a constraint has been added or some parameters have been changed so that the previously optimal basis is no longer feasible.

MATLAB IMPLEMENTATION: DUAL SIMPLEX

Consider the following example:

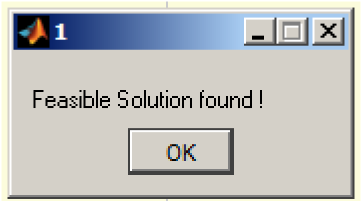
$$\text{Minimize:} \quad z = 3x_1 + x_2$$

$$\text{Subject to:} \quad x_1 + x_2 \geq 1$$

$$2x_1 + 3x_2 \geq 2$$

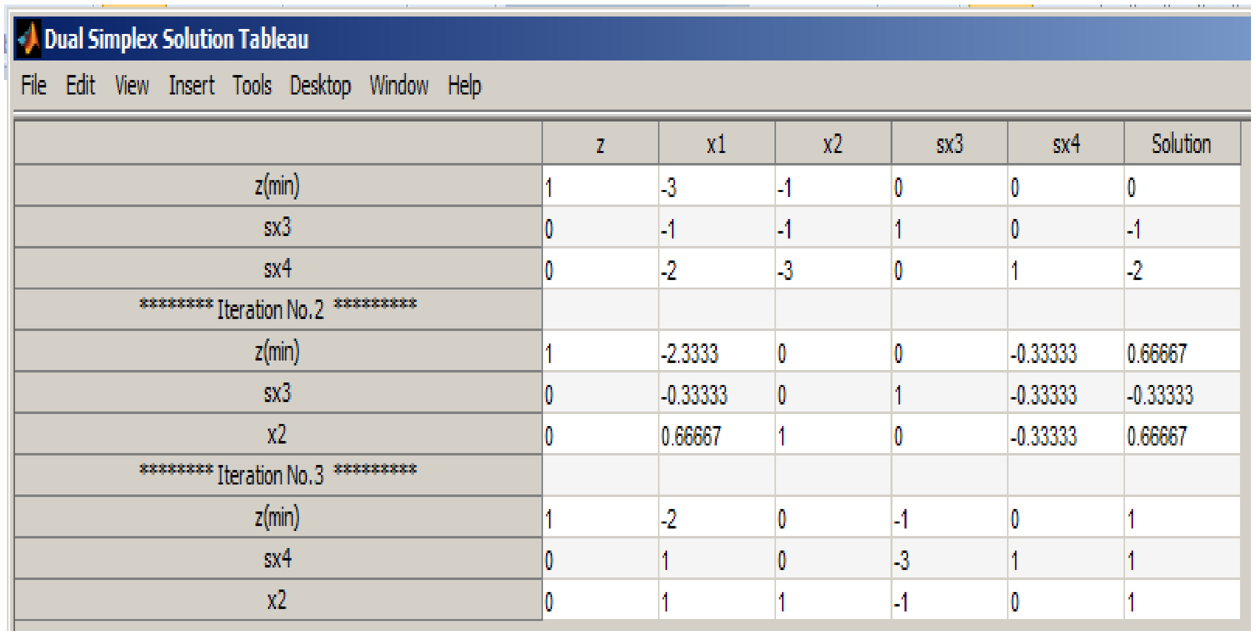
$$x_i \geq 0 \text{ , } i=1,2$$

After entering the values in the table and clicking on 'Solve' button, a dialogue box notifying that a feasible solution has been found is displayed.



Feasible Solution found !

After clicking on 'Ok' button, we get the final solution table as follows:


Dual Simplex Solution Tableau
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	sx3	sx4	Solution
z(min)	1	-3	-1	0	0	0
sx3	0	-1	-1	1	0	-1
sx4	0	-2	-3	0	1	-2
********* Iteration No.2 *********						
z(min)	1	-2.3333	0	0	-0.33333	0.66667
sx3	0	-0.33333	0	1	-0.33333	-0.33333
x2	0	0.66667	1	0	-0.33333	0.66667
********* Iteration No.3 *********						
z(min)	1	-2	0	-1	0	1
sx4	0	1	0	-3	1	1
x2	0	1	1	-1	0	1

IF NO STARTING OPTIMAL SOLUTION:

$$\text{Minimize:} \quad z = -x_1 + x_2$$

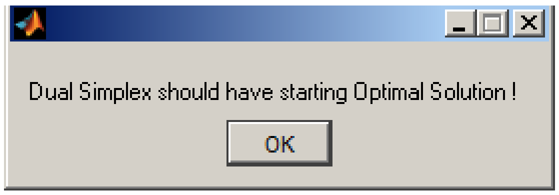
$$\text{Subject to :} \quad x_1 - 4x_2 \geq 5$$

$$2x_1 - 5x_2 \geq 1$$

$$x_1 - 3x_2 \leq 1$$

$$x_i \geq 0 \text{ , } i=1,2$$

After entering the values in the table and clicking on 'Solve' button, an appropriate dialogue box saying that starting optimal condition has to be satisfied is shown and the program is terminated.


Dual Simplex should have starting Optimal Solution !

NO FEASIBLE SOLUTION:

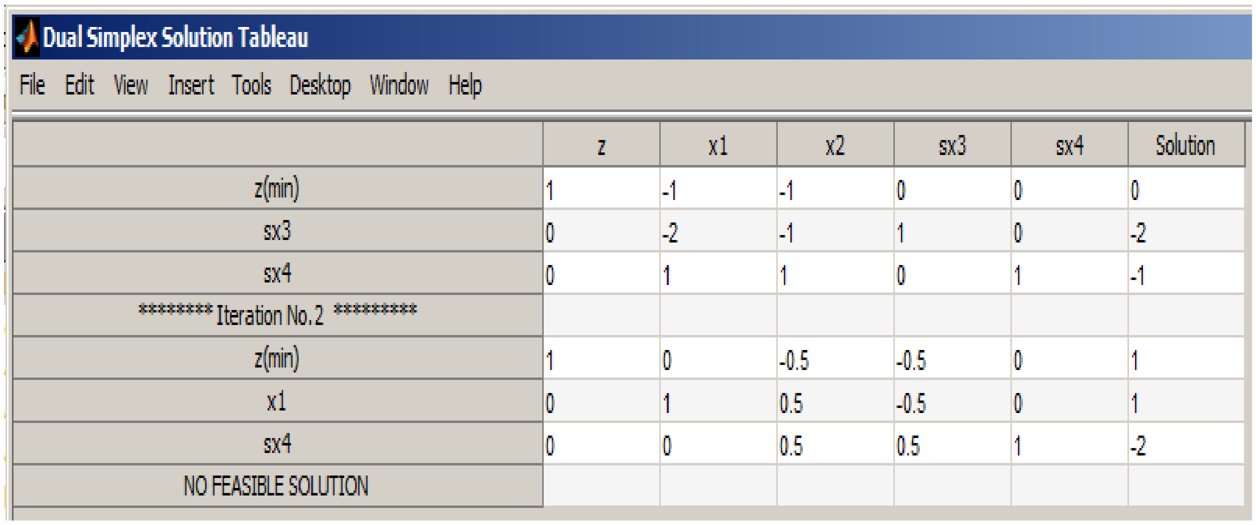
$$\text{Minimize:} \quad z = x_1 + x_2$$

$$\text{Subject to:} \quad 2x_1 + x_2 \geq 2$$

$$-x_1 - x_2 \geq 1$$

$$x_i \geq 0 \text{ , } i=1,2$$

After entering the values in the table and clicking on 'Solve' button, an appropriate dialogue box saying that the solution is Infeasible is shown and the final table is printed as follows:


Dual Simplex Solution Tableau

File Edit View Insert Tools Desktop Window Help

	z	x1	x2	sx3	sx4	Solution
z(min)	1	-1	-1	0	0	0
sx3	0	-2	-1	1	0	-2
sx4	0	1	1	0	1	-1
******** Iteration No.2 *********						
z(min)	1	0	-0.5	-0.5	0	1
x1	0	1	0.5	-0.5	0	1
sx4	0	0	0.5	0.5	1	-2
NO FEASIBLE SOLUTION						

ALTERNATE UNBOUNDED SOLUTION:

Consider the following example:

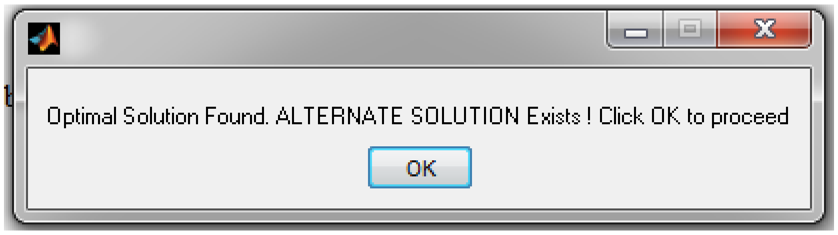
Minimize: $z = 3x_1 + 2x_2 + 3x_3$

Subject to: $x_1 + 4x_2 + x_3 \geq 7$

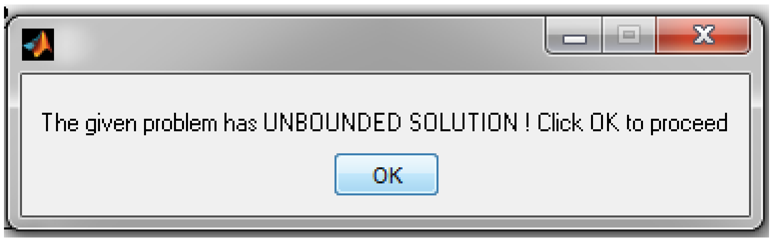
$2x_1 + x_2 + x_4 \geq 10$

$x_i \geq 0$, i=1,2,3,4

After entering the values in the table and clicking on 'Solve' button, we get a dialogue box notifying us about the existence of alternate optimal solution.

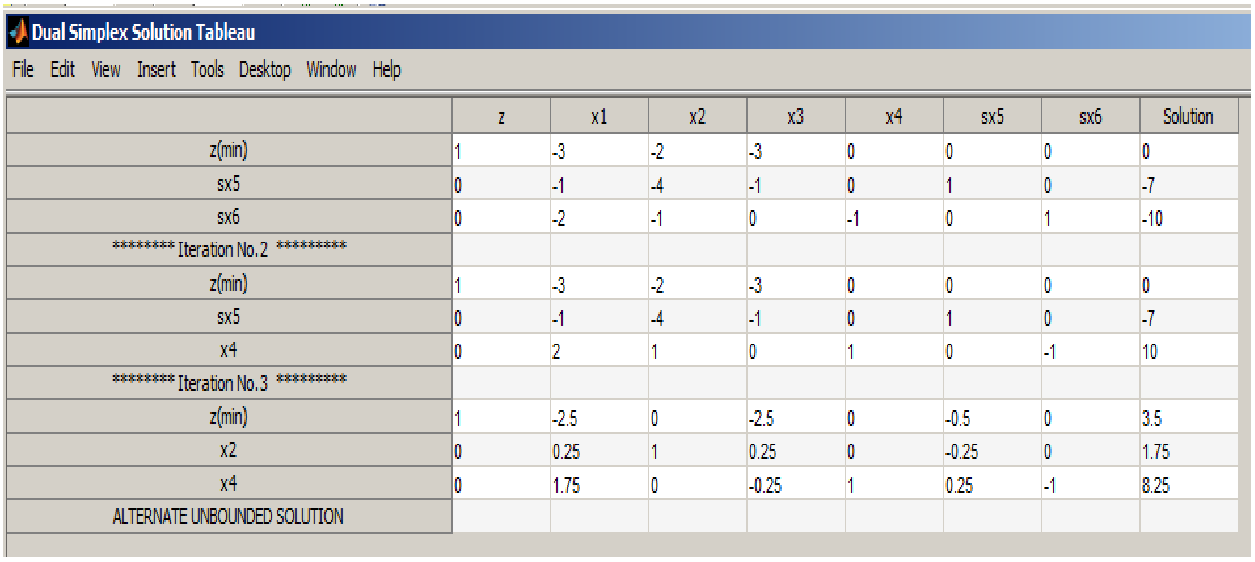

Optimal Solution Found. ALTERNATE SOLUTION Exists ! Click OK to proceed

On clicking 'Ok', if there is an unbounded solution as in this case, an appropriate dialogue box notifying us about it is shown.



The given problem has UNBOUNDED SOLUTION ! Click OK to proceed

Thereafter the final table is displayed as follows:



Dual Simplex Solution Tableau
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	x3	x4	sx5	sx6	Solution
z(min)	1	-3	-2	-3	0	0	0	0
sx5	0	-1	-4	-1	0	1	0	-7
sx6	0	-2	-1	0	-1	0	1	-10
********* Iteration No.2 *********								
z(min)	1	-3	-2	-3	0	0	0	0
sx5	0	-1	-4	-1	0	1	0	-7
x4	0	2	1	0	1	0	-1	10
********* Iteration No.3 *********								
z(min)	1	-2.5	0	-2.5	0	-0.5	0	3.5
x2	0	0.25	1	0.25	0	-0.25	0	1.75
x4	0	1.75	0	-0.25	1	0.25	-1	8.25
ALTERNATE UNBOUNDED SOLUTION								

GENERALIZED SIMPLEX ALGORITHM

The primal simplex algorithm starts with a feasible but non-optimal solution. The dual simplex algorithm starts with an infeasible but optimal solution. However, if an LP model starts both infeasible and non-optimal, then we can tailor the simplex algorithm in two parts. Initially, we can remove the infeasibility of the system, by applying the dual simplex method by making pivot selections regardless of the optimality, to arrive at a feasible and non-optimal solution. Thereafter, we can apply the primal simplex, to the LP, to arrive at the optimal solution. Thus, if feasibility is established, then we can pay attention to optimality by applying the proper optimality condition of the primal simplex method.

The essence of the Generalized Simplex Method is that the Simplex Method is not at all rigid. The literature abounds with variations of the Simplex Method (example, the primal-dual method, the symmetrical method, the criss-cross method and the multiplex method) that give the impression that each procedure is different, when in effect, they all seek a corner point solution, with a slant toward automated computations, and perhaps computational efficiency. One major use of the Generalized Simplex Method is that it does not incorporate the use of any artificial variables of artificial constraints.

MATLAB IMPLEMENTATION: GENERALIZED SIMPLEX

Consider the following example:

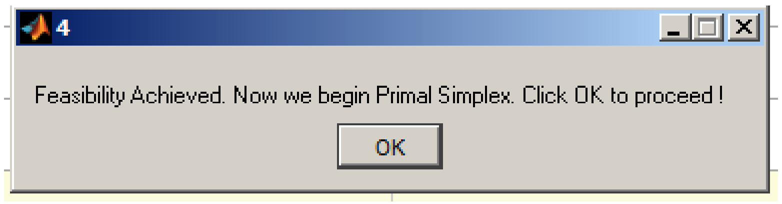
$$\text{Maximize:} \quad z = 3x_1 + 2x_2 + 3x_3$$

$$\text{Subject to:} \quad 2x_1 + x_2 + x_3 \le 2$$

$$3x_1 + 4x_2 + 2x_3 \ge 8$$

$$x_i \ge 0\ ,\ i=1,2,3$$

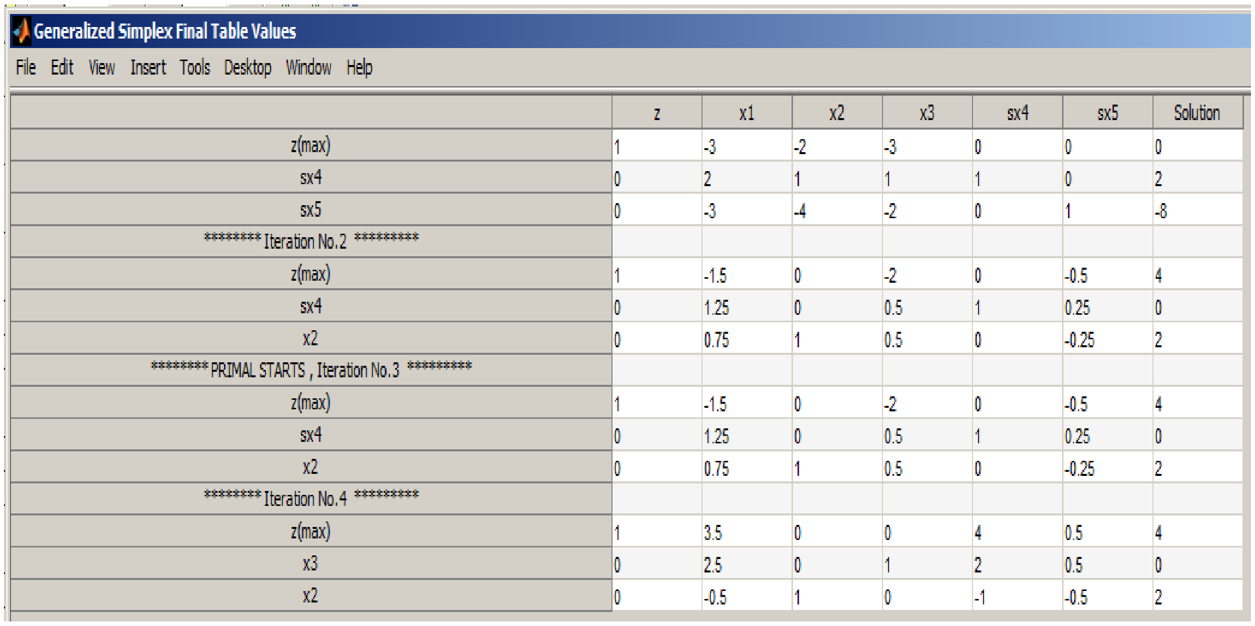
After entering the values in the table and clicking on 'Solve' button, we get a dialogue box notifying that feasibility is achieved and Primal Simplex begins:



4

Feasibility Achieved. Now we begin Primal Simplex. Click OK to proceed !

On clicking 'Ok', the final solution table is shown as follows:



Generalized Simplex Final Table Values

	z	x1	x2	x3	sx4	sx5	Solution
z(max)	1	-3	-2	-3	0	0	0
sx4	0	2	1	1	1	0	2
sx5	0	-3	-4	-2	0	1	-8
******** Iteration No.2 *********							
z(max)	1	-1.5	0	-2	0	-0.5	4
sx4	0	1.25	0	0.5	1	0.25	0
x2	0	0.75	1	0.5	0	-0.25	2
******** PRIMAL STARTS , Iteration No.3 *********							
z(max)	1	-1.5	0	-2	0	-0.5	4
sx4	0	1.25	0	0.5	1	0.25	0
x2	0	0.75	1	0.5	0	-0.25	2
******** Iteration No.4 *********							
z(max)	1	3.5	0	0	4	0.5	4
x3	0	2.5	0	1	2	0.5	0
x2	0	-0.5	1	0	-1	-0.5	2

It is clearly visible that the problem was initially solved using the Dual Simplex Method and later by the Primal Simplex Method. The transition between the two methods is shown clearly in the table.

INFEASIBLE SOLUTION:

Minimize: $z = -x_1 + x_2$

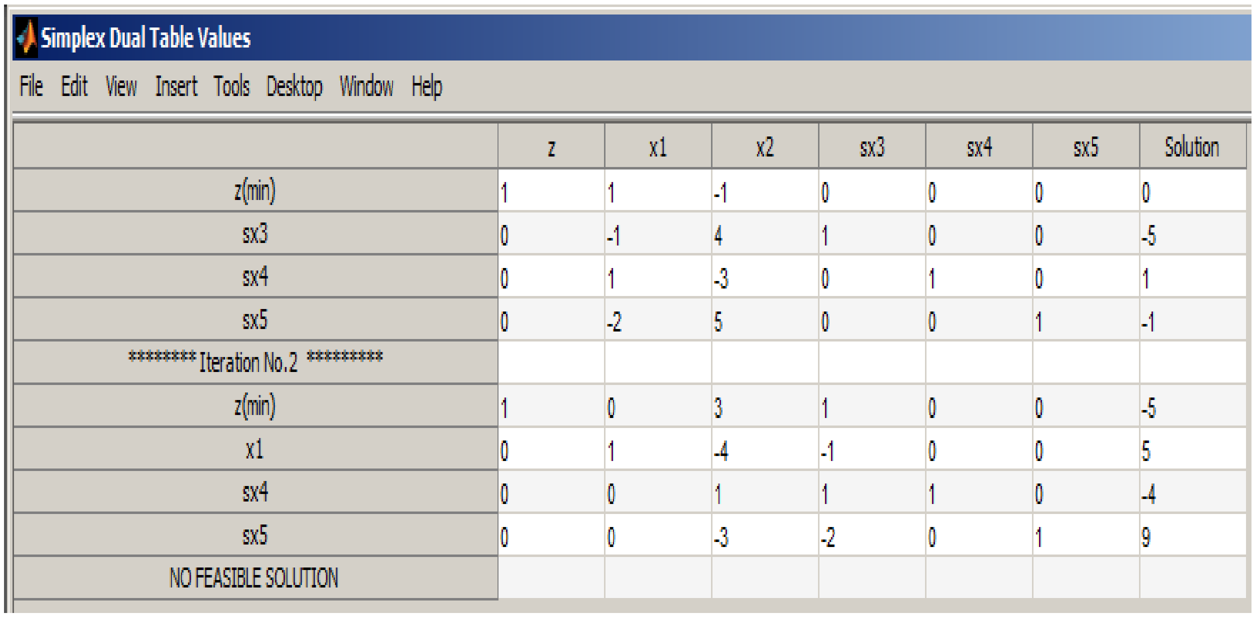
Subject to: $x_1 - 4x_2 \geq 5$

$x_1 - 5x_2 \geq 1$

$x_1 - 3x_2 \leq 1$

$x_i \geq 0$, i=1,2

After entering the values in the table and clicking on 'Solve' button, a relevant dialogue box saying that the solution is Infeasible is shown and the final table is printed as follows:


Simplex Dual Table Values
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	sx3	sx4	sx5	Solution
z(min)	1	1	-1	0	0	0	0
sx3	0	-1	4	1	0	0	-5
sx4	0	1	-3	0	1	0	1
sx5	0	-2	5	0	0	1	-1
******** Iteration No.2 *********							
z(min)	1	0	3	1	0	0	-5
x1	0	1	-4	-1	0	0	5
sx4	0	0	1	1	1	0	-4
sx5	0	0	-3	-2	0	1	9
NO FEASIBLE SOLUTION							

ALTERNATE SOLUTION:

$$\text{Maximize:} \quad z = 6x_1 + 4x_2$$

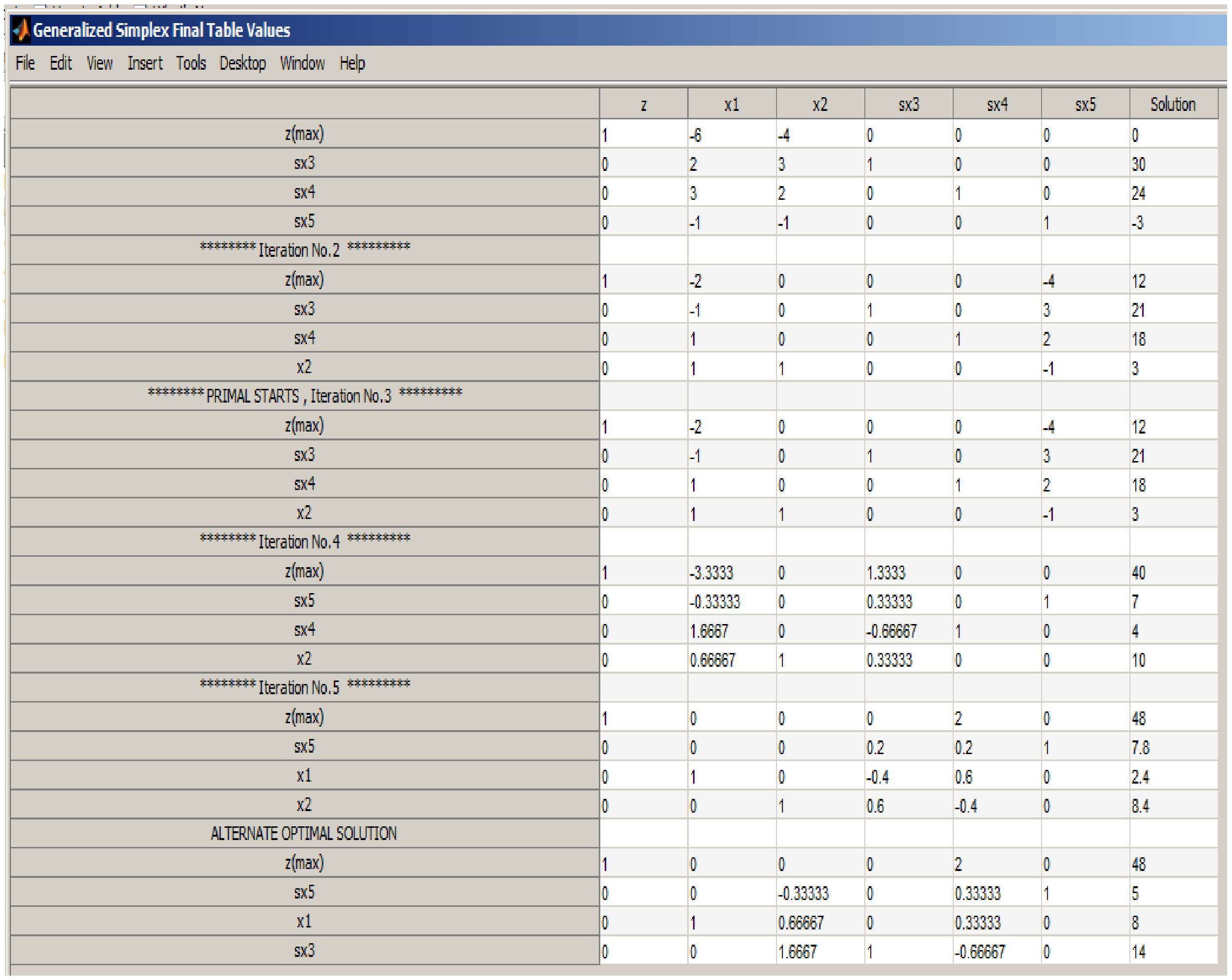
$$\text{Subject to:} \quad 2x_1 + 3x_2 \leq 30$$

$$x_1 + x_2 \geq 3$$

$$3x_1 + 2x_2 \leq 24$$

$$x_i \geq 0 \text{ , } i=1,2$$

After entering the values in the table and clicking on ‘Solve’ button, we get a dialogue box notifying us about the existence of alternate optimal solution and the final solution table with the alternate solution is shown as follows:


Generalized Simplex Final Table Values
File Edit View Insert Tools Desktop Window Help

	z	x1	x2	sx3	sx4	sx5	Solution
z(max)	1	-6	-4	0	0	0	0
sx3	0	2	3	1	0	0	30
sx4	0	3	2	0	1	0	24
sx5	0	-1	-1	0	0	1	-3
******** Iteration No.2 *********							
z(max)	1	-2	0	0	0	-4	12
sx3	0	-1	0	1	0	3	21
sx4	0	1	0	0	1	2	18
x2	0	1	1	0	0	-1	3
******** PRIMAL STARTS , Iteration No.3 *********							
z(max)	1	-2	0	0	0	-4	12
sx3	0	-1	0	1	0	3	21
sx4	0	1	0	0	1	2	18
x2	0	1	1	0	0	-1	3
******** Iteration No.4 *********							
z(max)	1	-3.3333	0	1.3333	0	0	40
sx5	0	-0.33333	0	0.33333	0	1	7
sx4	0	1.6667	0	-0.66667	1	0	4
x2	0	0.66667	1	0.33333	0	0	10
******** Iteration No.5 *********							
z(max)	1	0	0	0	2	0	48
sx5	0	0	0	0.2	0.2	1	7.8
x1	0	1	0	-0.4	0.6	0	2.4
x2	0	0	1	0.6	-0.4	0	8.4
ALTERNATE OPTIMAL SOLUTION							
z(max)	1	0	0	0	2	0	48
sx5	0	0	-0.33333	0	0.33333	1	5
x1	0	1	0.66667	0	0.33333	0	8
sx3	0	0	1.6667	1	-0.66667	0	14

UNBOUNDED SOLUTION:

Consider the following example:

$$\text{Maximize:} \quad z = 107x_1 + x_2 + 2x_3$$

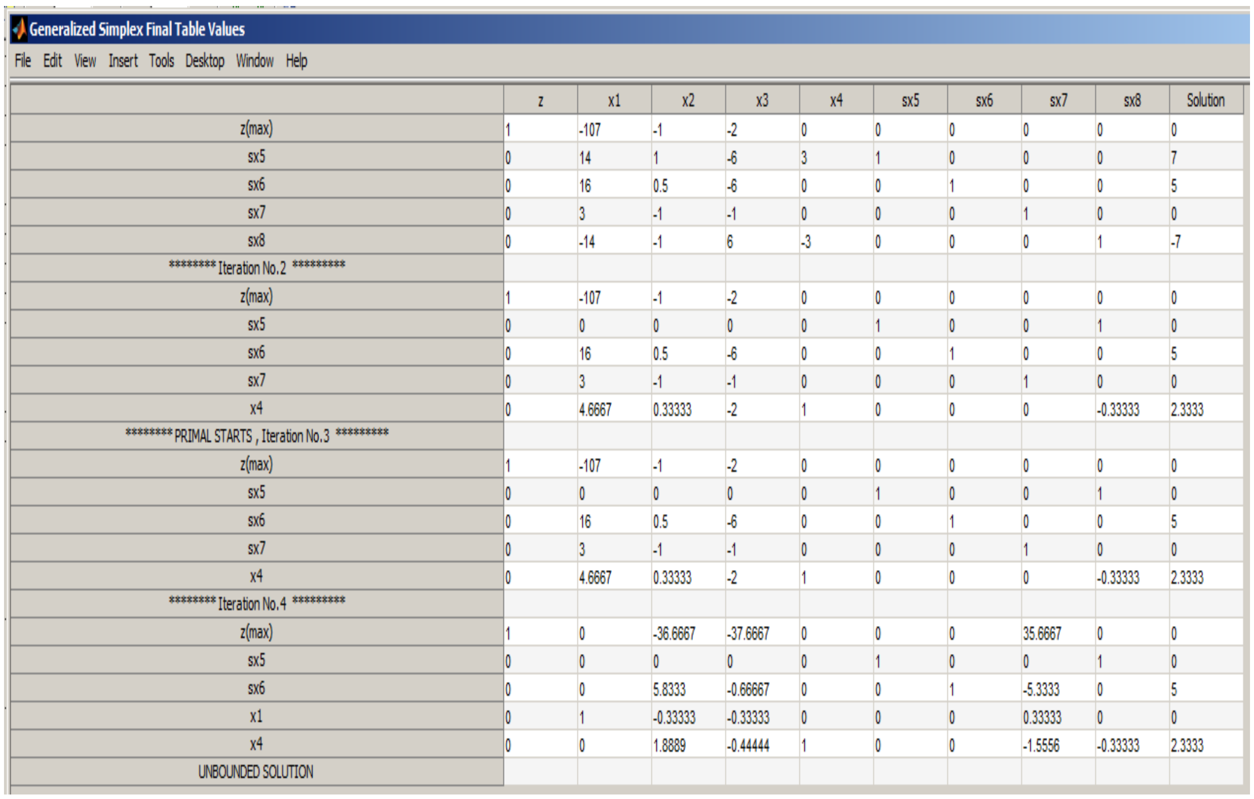
$$\text{Subject to:} \quad 14x_1 + x_2 - 6x_3 + 3x_4 = 7$$

$$3x_1 - x_2 - x_3 \le 0$$

$$16x_1 + 0.5x_2 - 6x_3 \le 5$$

$$x_i \ge 0 , i=1,2,3,4$$

After entering the values in the table and clicking on 'Solve' button, we get a dialogue box notifying us about the existence of unbounded solution and the final solution table is shown as follows:



Generalized Simplex Final Table Values

	z	x1	x2	x3	x4	sx5	sx6	sx7	sx8	Solution
z(max)	1	-107	-1	-2	0	0	0	0	0	0
sx5	0	14	1	-6	3	1	0	0	0	7
sx6	0	16	0.5	-6	0	0	1	0	0	5
sx7	0	3	-1	-1	0	0	0	1	0	0
sx8	0	-14	-1	6	-3	0	0	0	1	-7
******** Iteration No.2 *********										
z(max)	1	-107	-1	-2	0	0	0	0	0	0
sx5	0	0	0	0	0	1	0	0	1	0
sx6	0	16	0.5	-6	0	0	1	0	0	5
sx7	0	3	-1	-1	0	0	0	1	0	0
x4	0	4.6667	0.33333	-2	1	0	0	0	-0.33333	2.3333
******** PRIMAL STARTS , Iteration No.3 *********										
z(max)	1	-107	-1	-2	0	0	0	0	0	0
sx5	0	0	0	0	0	1	0	0	1	0
sx6	0	16	0.5	-6	0	0	1	0	0	5
sx7	0	3	-1	-1	0	0	0	1	0	0
x4	0	4.6667	0.33333	-2	1	0	0	0	-0.33333	2.3333
******** Iteration No.4 *********										
z(max)	1	0	-36.6667	-37.6667	0	0	0	35.6667	0	0
sx5	0	0	0	0	0	1	0	0	1	0
sx6	0	0	5.8333	-0.66667	0	0	1	-5.3333	0	5
x1	0	1	-0.33333	-0.33333	0	0	0	0.33333	0	0
x4	0	0	1.8889	-0.44444	1	0	0	-1.5556	-0.33333	2.3333
UNBOUNDED SOLUTION										

UNRESTRICTED VARIABLE:

Consider the following example:

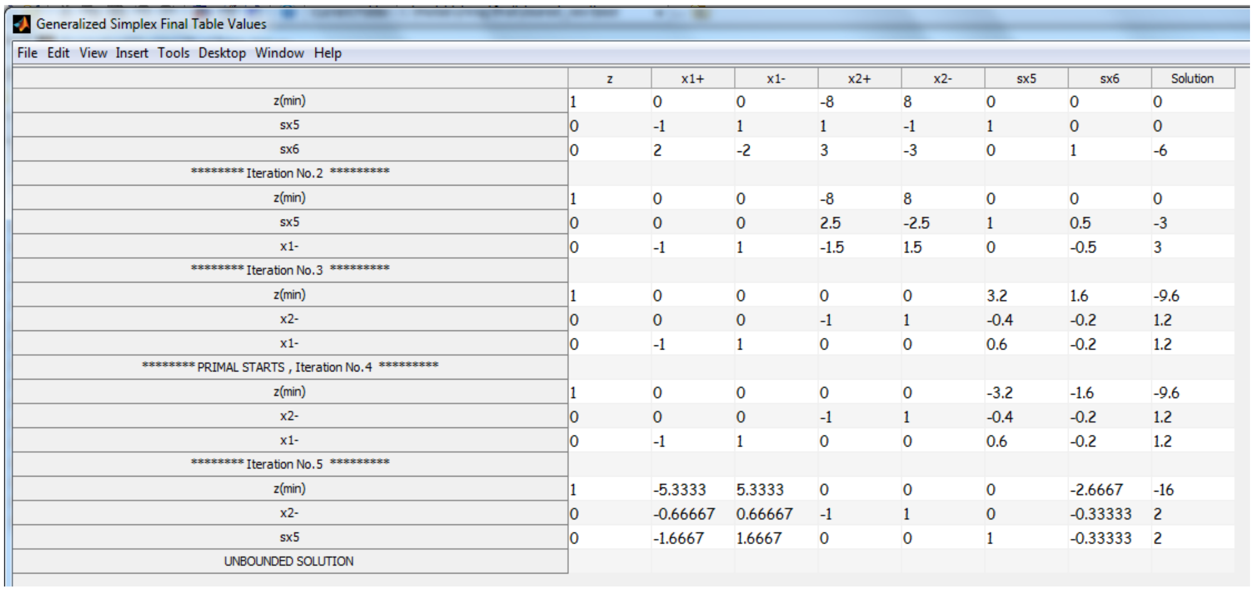
$$\text{Maximize:} \quad z = 8x_2$$

$$\text{Subject to:} \quad x_1 - x_2 \geq 0$$

$$2\,x_1 + 3x_2 \leq -6$$

$$x_1,\ x_2 \ \text{unrestricted}$$

After entering the values, and clicking the "Click To Solve" button, the simplex tableau is generated.


Generalized Simplex Final Table Values
File Edit View Insert Tools Desktop Window Help

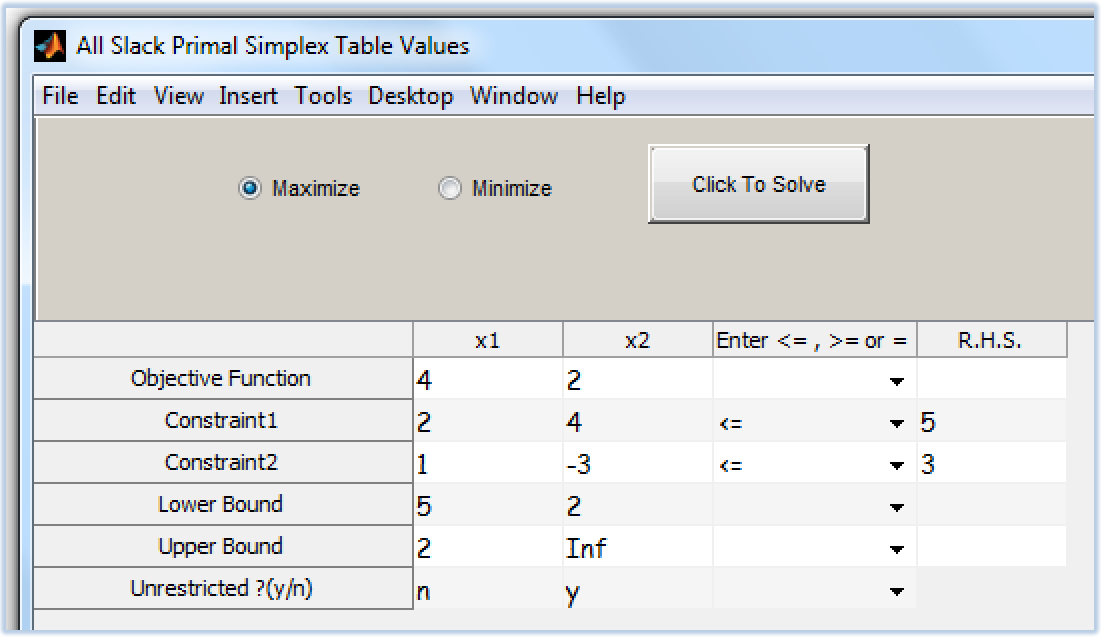
	z	x1+	x1-	x2+	x2-	sx5	sx6	Solution
z(min)	1	0	0	-8	8	0	0	0
sx5	0	-1	1	1	-1	1	0	0
sx6	0	2	-2	3	-3	0	1	-6
******** Iteration No.2 *********								
z(min)	1	0	0	-8	8	0	0	0
sx5	0	0	0	2.5	-2.5	1	0.5	-3
x1-	0	-1	1	-1.5	1.5	0	-0.5	3
******** Iteration No.3 *********								
z(min)	1	0	0	0	0	3.2	1.6	-9.6
x2-	0	0	0	-1	1	-0.4	-0.2	1.2
x1-	0	-1	1	0	0	0.6	-0.2	1.2
******** PRIMAL STARTS , Iteration No.4 *********								
z(min)	1	0	0	0	0	-3.2	-1.6	-9.6
x2-	0	0	0	-1	1	-0.4	-0.2	1.2
x1-	0	-1	1	0	0	0.6	-0.2	1.2
******** Iteration No.5 *********								
z(min)	1	-5.3333	5.3333	0	0	0	-2.6667	-16
x2-	0	-0.66667	0.66667	-1	1	0	-0.33333	2
sx5	0	-1.6667	1.6667	0	0	1	-0.33333	2
UNBOUNDED SOLUTION								

Since x_1 and x_2 are both unrestricted, new variables x_1^+, x_1^-, x_2^+ and x_2^- are introduced in the solution table.

ERROR HANDLING

LOWER BOUND > UPPER BOUND:

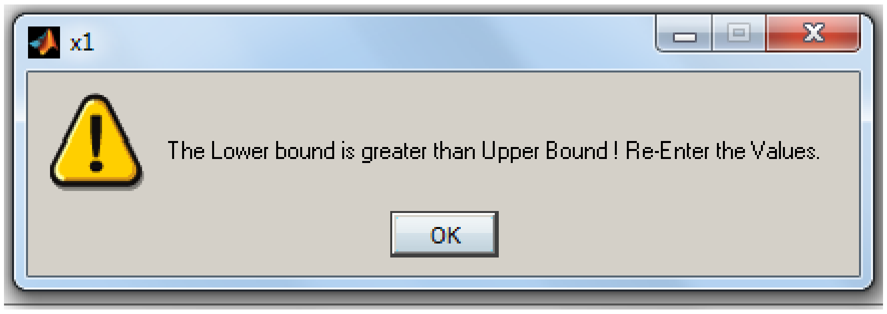
Consider the following table having the input values as shown.


All Slack Primal Simplex Table Values
File Edit View Insert Tools Desktop Window Help
Click To Solve

● Maximize ○ Minimize

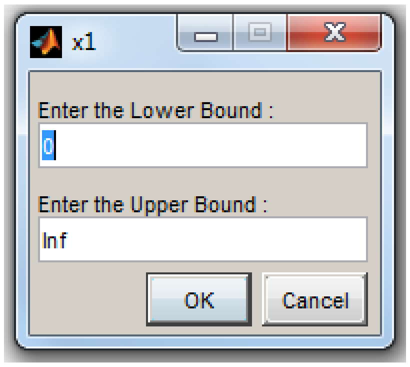
	x1	x2	Enter <= , >= or =	R.H.S.
Objective Function	4	2		
Constraint1	2	4	<=	5
Constraint2	1	-3	<=	3
Lower Bound	5	2		
Upper Bound	2	Inf		
Unrestricted ?(y/n)	n	y		

It is clearly visible that for the variable x_1, the lower bound (5) is greater than the upper bound (2), which is not permissible. To deal with it, an error message box displayed to notify the user about this error, with the variable name displayed as the title of the message box.


x1

The Lower bound is greater than Upper Bound ! Re-Enter the Values.

On clicking 'Ok', an input box is displayed asking the user to enter the new values for the lower and upper bounds. These values are again checked for any errors and if correct, the next step is executed.

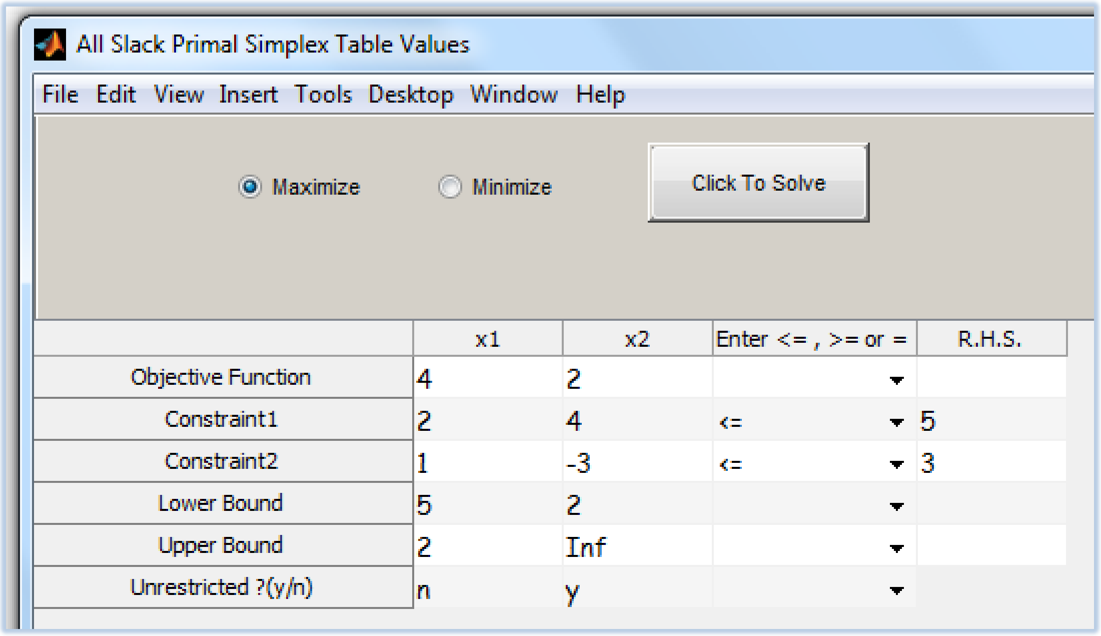


Enter the Lower Bound : 0

Enter the Upper Bound : Inf

VARIABLE WITH BOUNDS AND UNRESTRICTED:

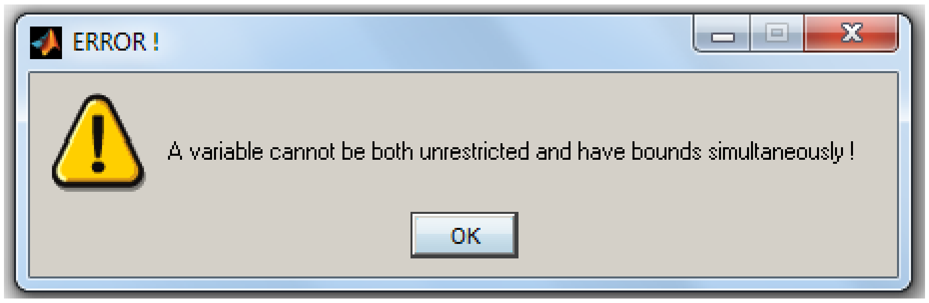
Consider the same example for the input values as shown earlier.


All Slack Primal Simplex Table Values
File Edit View Insert Tools Desktop Window Help
Click To Solve

[✓ Maximize] Minimize

	x1	x2	Enter <= , >= or =	R.H.S.
Objective Function	4	2		
Constraint1	2	4	<=	5
Constraint2	1	-3	<=	3
Lower Bound	5	2		
Upper Bound	2	Inf		
Unrestricted ?(y/n)	n	y		

Now, after the lower bound > upper bound case is solved, a new error message box pops up pointing out that the variable x_2 is unrestricted in spite of having a lower bound (2).

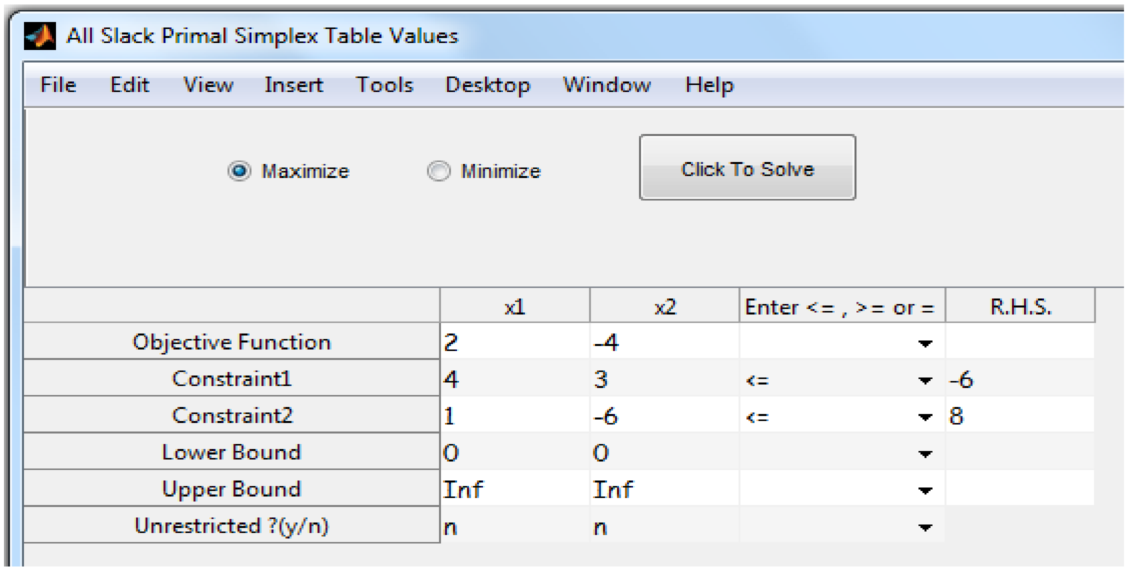


A variable cannot be both unrestricted and have bounds simultaneously !

On clicking 'Ok', the same table is displayed again for the user to correct the values entered for unrestricted and bounds case. Once the user enters the new value, it is again checked for any anomalies and if so, the same steps repeat.

NEGATIVE RHS FOR PRIMAL SIMPLEX:

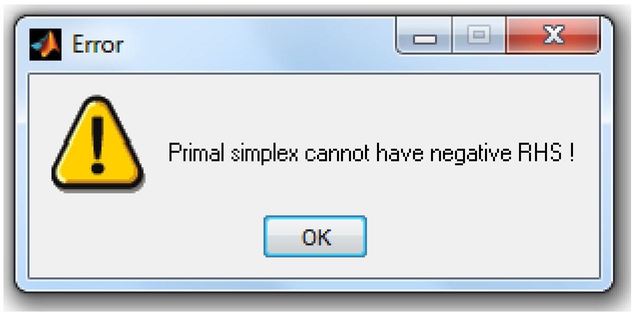
Consider a Primal Simplex problem having one of the constraints such as it has the inequality as '≤' and its RHS is negative.


All Slack Primal Simplex Table Values
File Edit View Insert Tools Desktop Window Help
Click To Solve

(● Maximize) ○ Minimize

	x1	x2	Enter <= , >= or =	R.H.S.
Objective Function	2	-4		
Constraint1	4	3	<=	-6
Constraint2	1	-6	<=	8
Lower Bound	0	0		
Upper Bound	Inf	Inf		
Unrestricted ?(y/n)	n	n		

As the RHS for constraint 1 is negative, we cannot apply Primal Simplex Method. An error message box notifying the same is shown as follows:

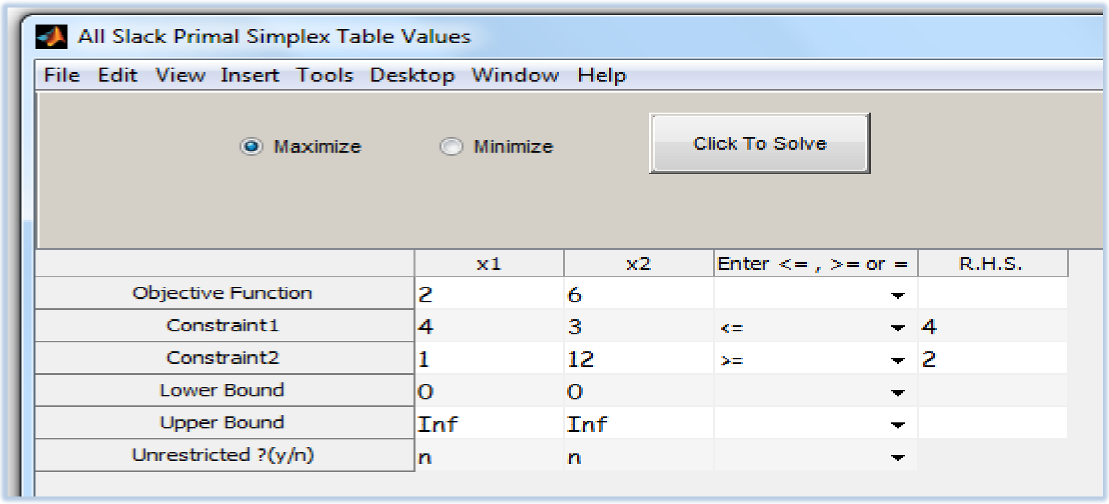


Primal simplex cannot have negative RHS !

On clicking 'Ok', the same table is displayed again for the user to correct the values. However if the RHS is negative and the inequality is '≥', no such error message box is displayed as the inequality is converted to '≤' with the RHS being non-negative by multiplying the entire constraint by -1.

INEQUALITY IS '≥' FOR POSITIVE RHS IN PRIMAL SIMPLEX:

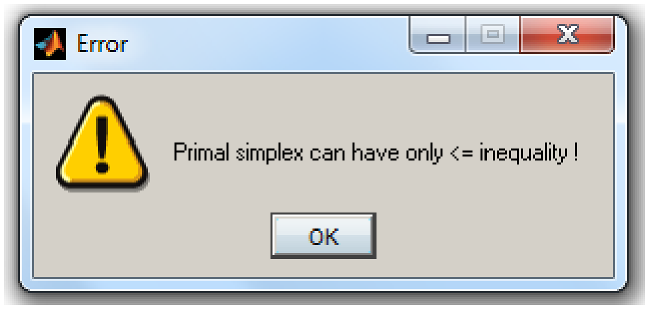
Consider the following input values for a Primal Simplex problem having the inequality for one of the constraints as '≥' with the RHS being non-negative.


All Slack Primal Simplex Table Values
File Edit View Insert Tools Desktop Window Help
Click To Solve

(● Maximize) ○ Minimize

	x1	x2	Enter <= , >= or =	R.H.S.
Objective Function	2	6		
Constraint1	4	3	<=	4
Constraint2	1	12	>=	2
Lower Bound	0	0		
Upper Bound	Inf	Inf		
Unrestricted ?(y/n)	n	n		

To deal with this, an error message box showing this error is displayed.



Primal simplex can have only <= inequality !

On clicking 'Ok', the same starting table is displayed for the user to re-enter the values.

KRUSKAL'S ALGORITHM

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each connected component). Kruskal's algorithm is an example of a greedy algorithm.

DESCRIPTION

- Create a forest *F* (a set of trees), where each vertex in the graph is a separate tree
- Create a set *S* containing all the edges in the graph
- While *S* is nonempty and F is not yet spanning
    1. Remove an edge with minimum weight from *S*
    2. If that edge connects two different trees, then add it to the forest, combining two trees into a single tree
    3. Otherwise discard that edge.

At the termination of the algorithm, the forest has only one component and forms a minimum spanning tree of the graph.

PERFORMANCE

Where E is the number of edges in the graph and V is the number of vertices, Kruskal's algorithm can be shown to run in $O(E \log E)$ time, or equivalently, $O(E \log V)$ time, all with simple data structures. These running times are equivalent because:

- E is at most V^2 and $\log V^2 = 2\log V$ is $O(\log V)$.
- If we ignore isolated vertices, which will each be their own component of the minimum spanning forest, $V \leq E+1$, so $\log V$ is $O(\log E)$.

We can achieve this bound as follows: first sort the edges by weight using a comparison sort in $O(E \log E)$ time; this allows the step "remove an edge with minimum weight from S" to operate in constant time. Next, we use a disjoint-set data structure to keep track of which vertices are in which components. We need to perform $O(E)$ operations, two 'find' operations and possibly one union for each edge. Even a simple disjoint-set data structure such as disjoint-set forests with union by rank can perform $O(E)$ operations in $O(E \log V)$ time. Thus the total time is $O(E \log E) = O(E \log V)$.

Provided that the edges are either already sorted or can be sorted in linear time (for example with counting sort or radix sort), the algorithm can use more sophisticated disjoint-set data structure to finish the execution in even lesser time.

APPLICATIONS

The Kruskal's Algorithm generates a minimum spanning sub-tree for a given connected, weighted graph.

The minimum spanning tree thus generated has many applications such as follows.

1. **Network design**
   – telephone, electrical, hydraulic, TV cable, computer, road

   The standard application is to a problem like phone network design. You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. It should be a spanning tree, since if a network isn't a tree you can always remove some edges and save money.

2. **Approximation algorithms for NP-hard problems**
   – traveling salesperson problem, Steiner tree

   A less obvious application is that the minimum spanning tree can be used to approximately solve the traveling salesman problem. A convenient formal way of defining this problem is to find the shortest path that visits each point at least once.

   Note that if you have a path visiting all points exactly once, it's a special kind of tree. For instance, in the example above, twelve of sixteen spanning trees are actually paths. If you have a path visiting some vertices more than once, you can always drop some edges to get a tree. So in general the MST weight is less than the TSP weight, because it's a minimization over a strictly larger set.

   On the other hand, if you draw a path tracing around the minimum spanning tree, you trace each edge twice and visit all points, so the TSP weight is less than twice the MST weight. Therefore this tour is within a factor of two of optimal.

3. **Indirect applications**

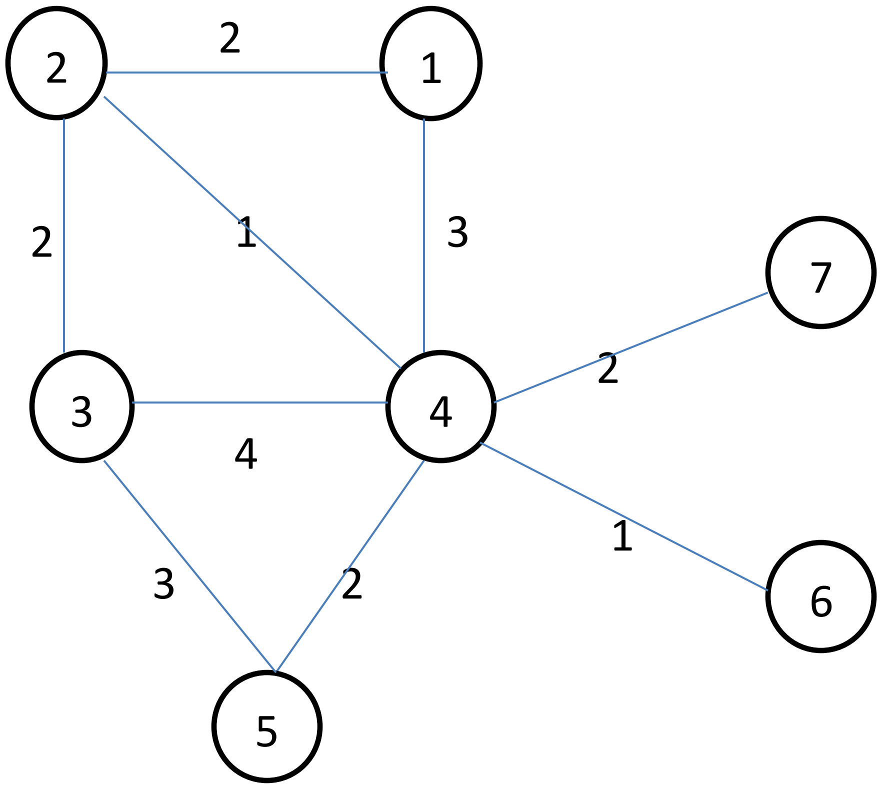
   – Max bottleneck paths
   – LDPC codes for error correction
   – Image registration with Renyi entropy
   – Learning salient features for real-time face verification
   – Reducing data storage in sequencing amino acids in a protein
   – Model locality of particle interactions in turbulent fluid flows
   – Autoconfig protocol for Ethernet bridging to avoid cycles in a network

4. **Cluster analysis**

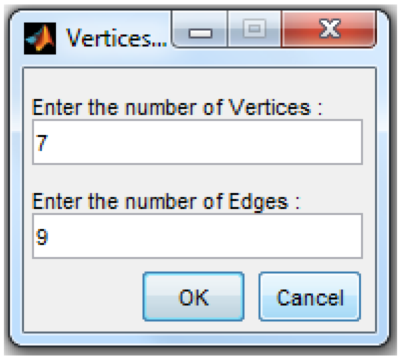
   k-clustering problem can be viewed as finding a Minimum Spanning Tree and deleting the k-1 most expensive edges.

MATLAB IMPLEMENTATION: KRUSKAL ALGORITHM

Consider the following graph:



The program first asks the user to enter the total number of Vertices and Edges in the graph.



Enter the number of Vertices :

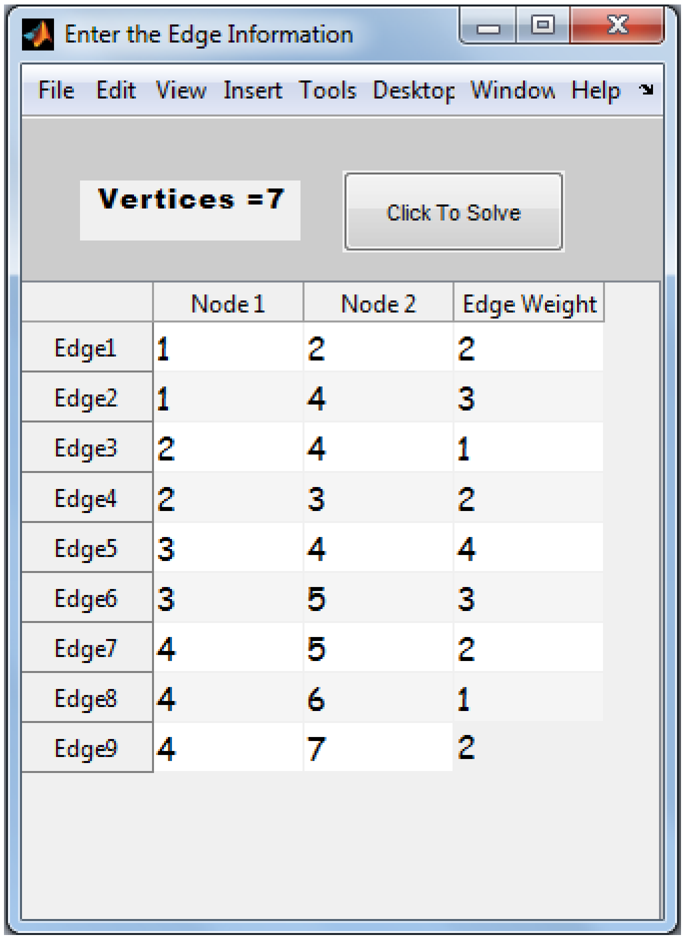
7

Enter the number of Edges :

9

On clicking 'Ok', the input table is presented before the user for him to enter the information about the graph. The information is collected in 3 columns as shown. For an edge between vertices 1 and 2, if the weight of the edge is 5, then the values in the first, second and third column for edge1 will be 1, 2 and 5 respectively.

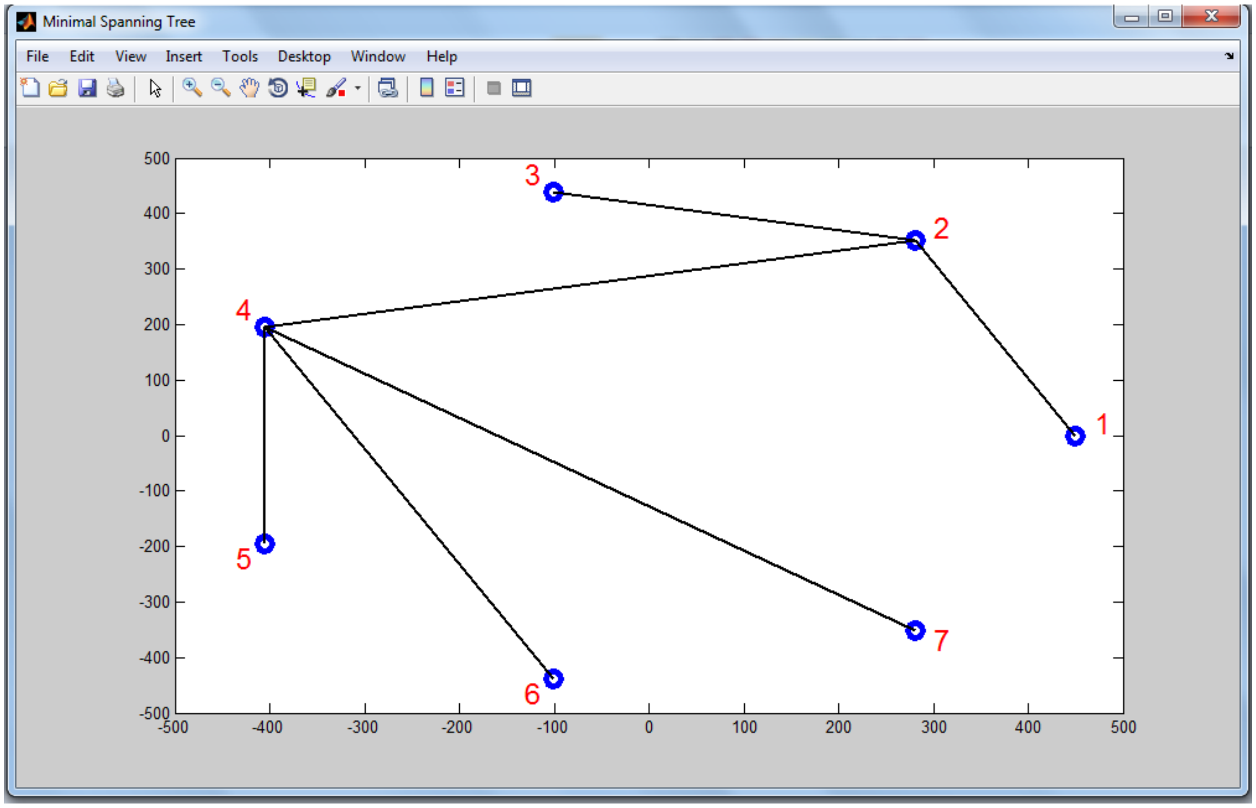
This method of input reduces the tedious process of entering the adjacency matrix of the graph along with the edge weights.


Enter the Edge Information
File Edit View Insert Tools Desktop Window Help
Click To Solve

Vertices =7

	Node 1	Node 2	Edge Weight
Edge1	1	2	2
Edge2	1	4	3
Edge3	2	4	1
Edge4	2	3	2
Edge5	3	4	4
Edge6	3	5	3
Edge7	4	5	2
Edge8	4	6	1
Edge9	4	7	2

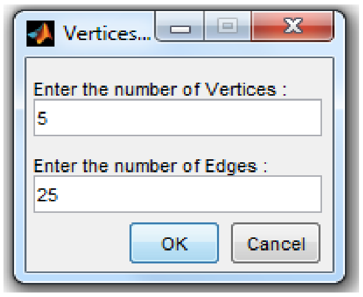
On clicking the 'Click to Solve' button, the final Minimum Spanning Tree, as calculated using the Kruskal Algorithm, is displayed as shown in the figure below.



ERROR HANDLING

NUMBER OF EDGES > MAXIMUM EDGES FOR A GRAPH:

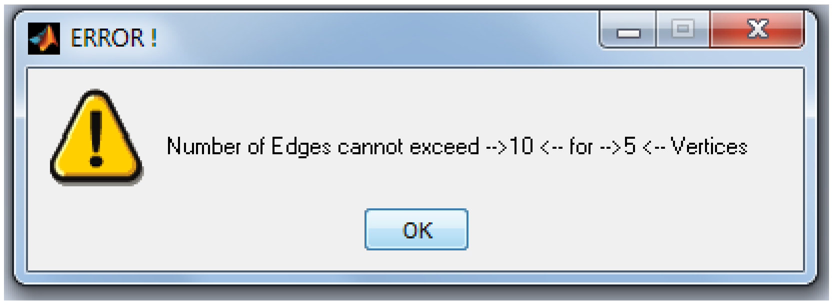
Suppose when entering the total number of vertices and edges for the graph, if for the number of vertices say n, if the total number of edges exceed n*(n-1)/2, then an appropriate message box is displayed.



Enter the number of Vertices :
5

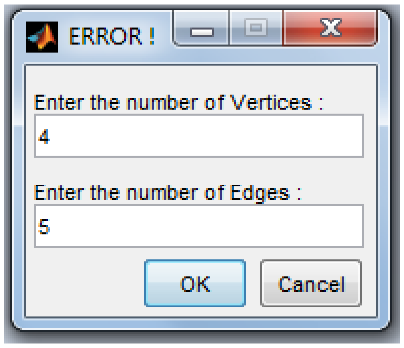
Enter the number of Edges :
25

For instance, for a graph of 5 vertices, the total number of vertices cannot exceed 10. Thus if a value greater than 10 is entered, the following error message is displayed.



Number of Edges cannot exceed -->10 <-- for -->5 <-- Vertices

Then user is then again asked to enter the number of vertices and edges.

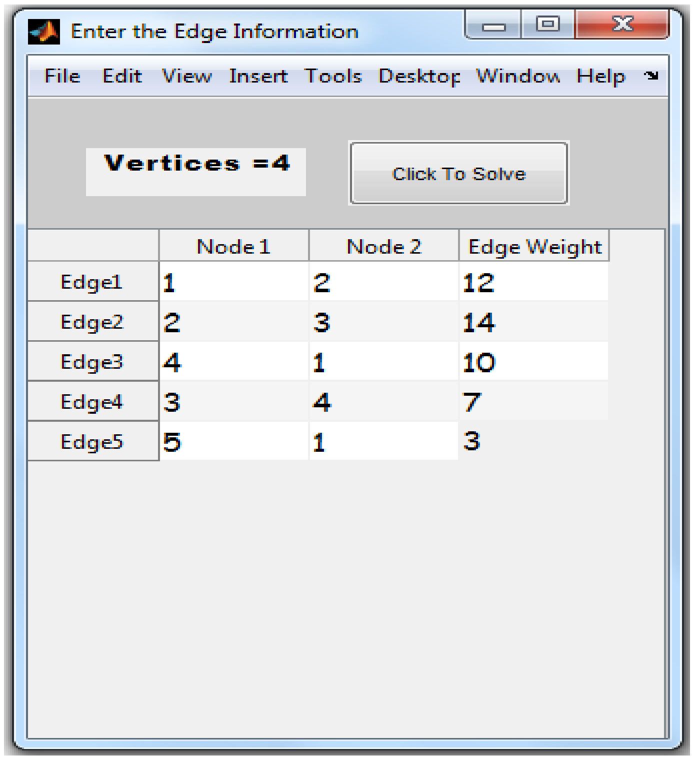


Enter the number of Vertices :
4

Enter the number of Edges :
5

NODE NUMBER EXCEEDS NUMBER OF VERTICES:

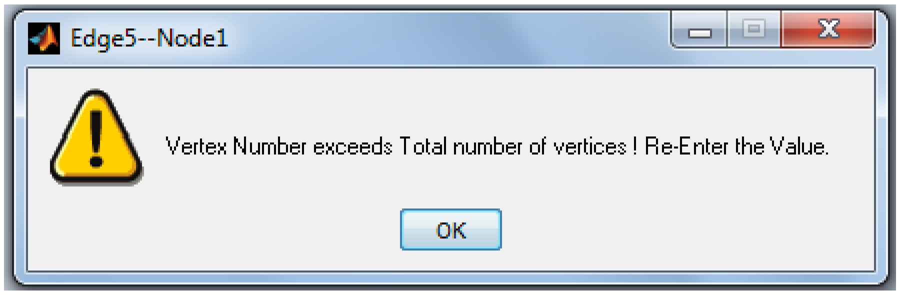
Consider a graph having 4 vertices and 5 edges. While entering the edge information, if the user enters one of the node values greater than the total number of vertices, then an appropriate error message is displayed.


Enter the Edge Information
File Edit View Insert Tools Desktop Window Help
Click To Solve

Vertices =4

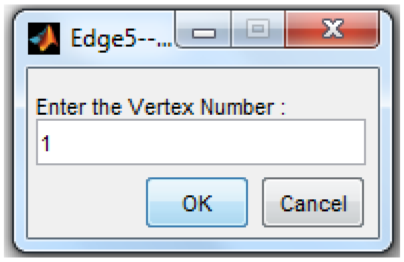
	Node 1	Node 2	Edge Weight
Edge1	1	2	12
Edge2	2	3	14
Edge3	4	1	10
Edge4	3	4	7
Edge5	5	1	3

In the above example, it is clearly visible that for the Edge 5, the node 1 value entered is 5, whereas the total number if vertices is 4. To notify the user about this error, the following error message is displayed.



Vertex Number exceeds Total number of vertices ! Re-Enter the Value.

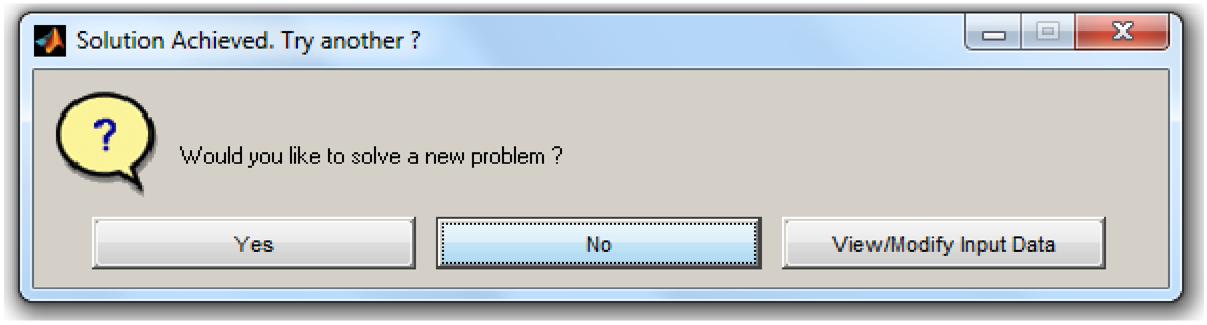
On clicking 'Ok', the following dialogue box is popped to make the user re-enter the value for the respective node.



Enter the Vertex Number :

1

Also a provisional dialogue box is made to solve another problem, or to modify the given input data, if required by the user on closing the output figure window showing the Minimum Spanning Tree:



Would you like to solve a new problem ?

FLOYD WARSHALL ALGORITHM

The Floyd–Warshall algorithm (also known as Floyd's algorithm, Roy–Warshall algorithm, Roy–Floyd algorithm, or the WFI algorithm) is a graph analysis algorithm for finding shortest paths in a weighted graph (with positive or negative edge weights). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between *all* pairs of vertices though it does not return details of the paths themselves. The algorithm is an example of dynamic programming.

The Floyd–Warshall algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with only $\Theta(|V|_3)$ comparisons in a graph. This is remarkable considering that there may be up to $\Omega(|V|_2)$ edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal.

Consider a graph G with vertices V, each numbered 1 through n. Further consider a function shortest_path(i,j) that returns the shortest possible paths from every vertex i to j , with n being the total number of vertices in the graph.

There are two candidates for each of these paths: either the true shortest path is the one which already exists (i.e. edge weight of i and j); or there exists some path that goes from i to $k + 1$, then from $k + 1$ to j that is better. We know that the best path from every vertex i to j is defined by shortest_path(i,j) and it is clear that if there were a better path from i to $k + 1$ to j, then the length of this path would be the concatenation of the shortest path from i to $k + 1$ and the shortest path from $k + 1$ to j .

Therefore, we can define shortest_path(i,j) in terms of the following recursive formula:

```
Shortest Path between i and j = Min { edge_weight (i,j) ,
                                      edge_weight(i,k) + edge_weight(k,j)
                                    }
```

NEGATIVE CYCLES BEHAVIOUR

For numerically meaningful output, the Floyd–Warshall algorithm assumes that there are no negative cycles (in fact, between any pair of vertices which form part of a negative cycle, the shortest path is not well-defined because the path can be arbitrarily negative).

Nevertheless, if there are negative cycles, the Floyd–Warshall algorithm can be used to detect them. The intuition is as follows:

- The Floyd–Warshall algorithm iteratively revises path lengths between all pairs of vertices (i, j), including where $i = j$;
- Initially, the length of the path (i,i) is zero;
- A path $\{(i,k), (k,i)\}$ can only improve upon this if it has length less than zero, i.e. denotes a negative cycle;
- Thus, after the algorithm, (i,i) will be negative if there exists a negative-length path from i back to i.

Hence, to detect negative cycles using the Floyd–Warshall algorithm, one can inspect the diagonal of the path matrix, and the presence of a negative number indicates that the graph contains at least one negative cycle. Obviously, in an undirected graph a negative edge creates a negative cycle (i.e., a closed walk) involving its incident vertices.

ANALYSIS

To find all n^2 of shortestPath(i,j,k) (for all i and j) from those of shortestPath($i,j,k-1$) requires $2n^2$ operations. Since we begin with shortestPath(i,j,0) = edge_weight(i,j) and compute the sequence of n matrices shortestPath(i,j,1), shortestPath(i,j,2), …, shortestPath(i,j,n), the total number of operations used is $n \cdot 2n^2 = 2n^3$. Therefore, the complexity of the algorithm is $\Theta(n^3)$.

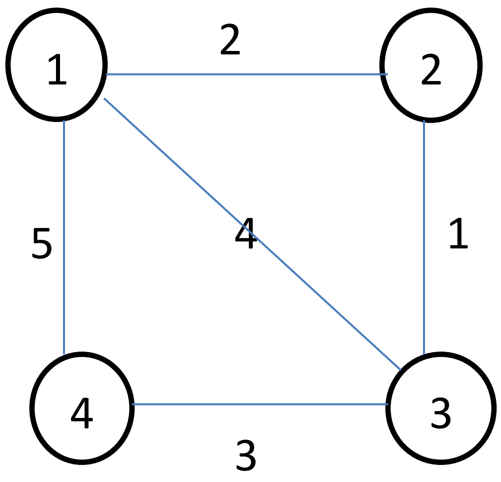
APPLICATIONS AND GENERALIZATIONS

The Floyd–Warshall algorithm can be used to solve the following problems, among others:

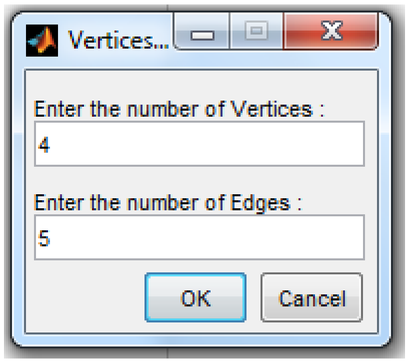
- Shortest paths in directed graphs (Floyd's algorithm)
- Transitive closure of directed graphs (Warshall's algorithm). In Warshall's original formulation of the algorithm, the graph is un- weighted and represented by a Boolean adjacency matrix. Then the addition operation is replaced by logical conjunction (AND) and the minimum operation by logical disjunction (OR)
- Finding a regular expression denoting the regular language accepted by a finite automaton (Kleene's Algorithm)
- Inversion of real matrices (Gauss–Jordan Algorithm)
- Optimal routing. In this application one is interested in finding the path with the maximum flow between two vertices. This means that, rather than taking minima as in the pseudo code above, one instead takes maxima. The edge weights represent fixed constraints on flow. Path weights represent bottlenecks; so the addition operation above is replaced by the minimum operation
- Testing whether an undirected graph is bipartite
- Fast computation of Pathfinder networks
- Maximum Bandwidth Paths in Flow Networks

MATLAB IMPLEMENTATION: FLOYD WARSHALL

Consider the following graph



The program first asks the user to enter the total number of Vertices and Edges in the graph.

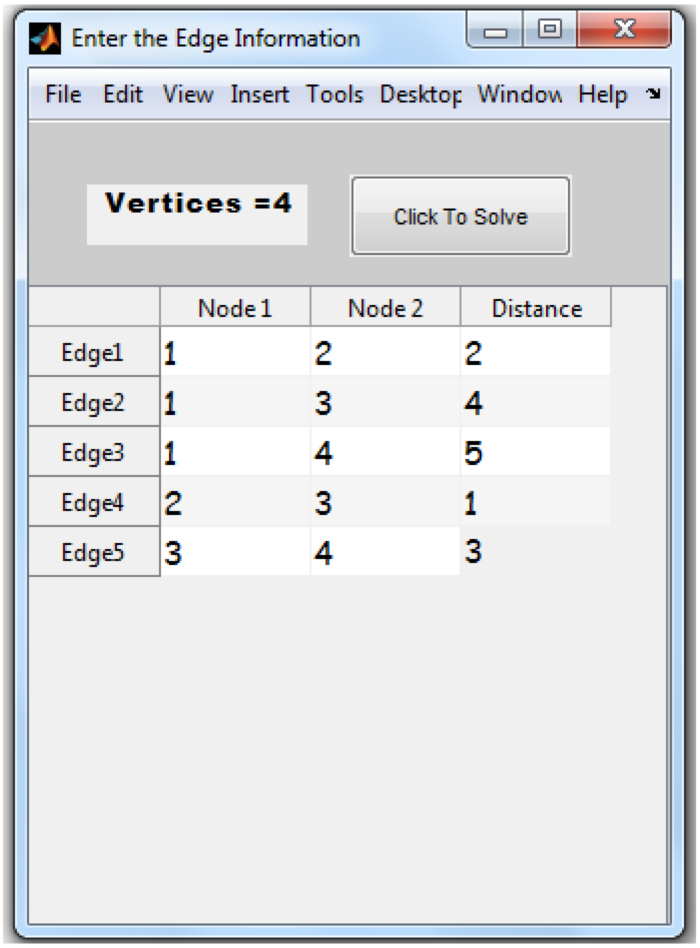


Enter the number of Vertices :
4

Enter the number of Edges :
5

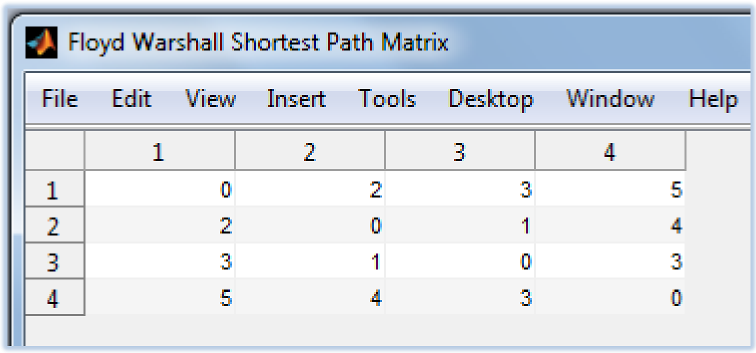
On clicking 'Ok', the input table is presented before the user for him to enter the information about the graph. The information is collected in 3 columns as shown. For an edge between vertices 1 and 2, if the distance between them is 5, then the values in the first, second and third column for edge1 will be 1, 2 and 5 respectively.

This method of input reduces the tedious process of entering the adjacency matrix of the graph along with the distances.


Enter the Edge Information
File Edit View Insert Tools Desktop Window Help
Vertices =4
Click To Solve

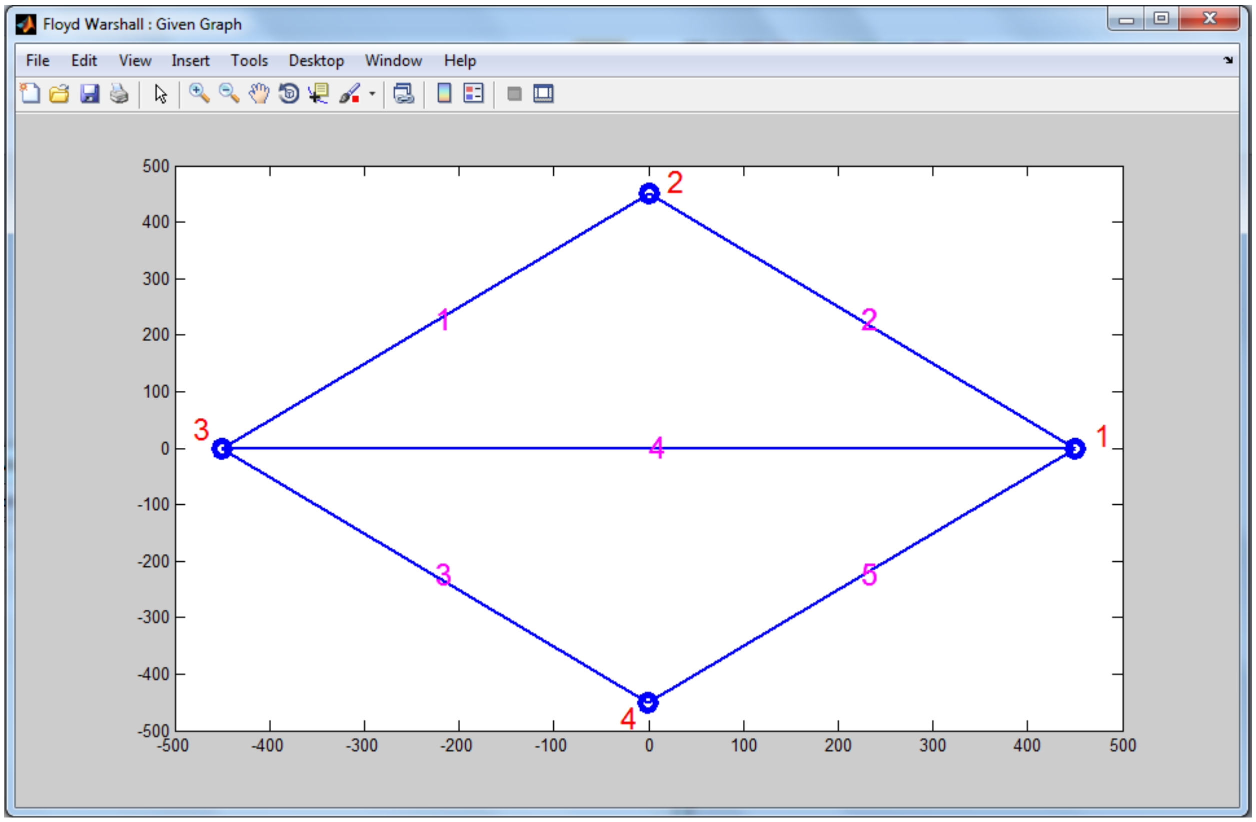
	Node 1	Node 2	Distance
Edge1	1	2	2
Edge2	1	3	4
Edge3	1	4	5
Edge4	2	3	1
Edge5	3	4	3

On clicking the 'Click to Solve' button, the final distance matrix for the given graph is shown. In this symmetric matrix D, the value of D(i,j) is the shortest distance between the vertices i and j. The shortest distance between Node1 and Node3 is updated from 4 to 3 as clearly visible from the given graph and the matrix given below.


Floyd Warshall Shortest Path Matrix
File Edit View Insert Tools Desktop Window Help

	1	2	3	4
1	0	2	3	5
2	2	0	1	4
3	3	1	0	3
4	5	4	3	0

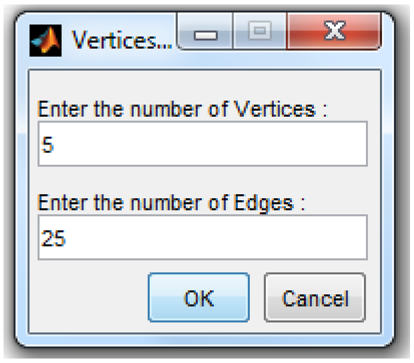
Also, the given graph is displayed along with the distances for any references.



ERROR HANDLING

NUMBER OF EDGES > MAXIMUM EDGES FOR A GRAPH:

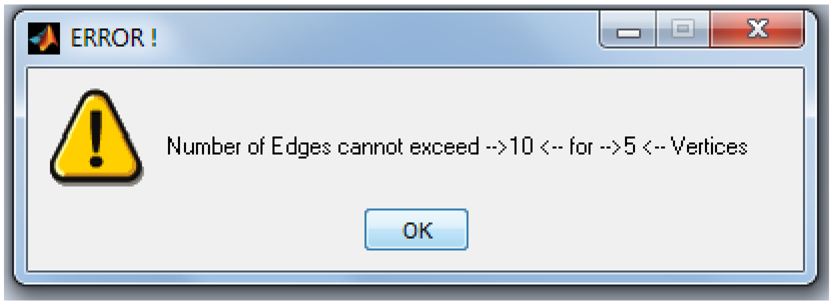
Suppose when entering the total number of vertices and edges for the graph, if for the number of vertices say n, if the total number of edges exceed n*(n-1)/2, then an appropriate message box is displayed.



Enter the number of Vertices :
5

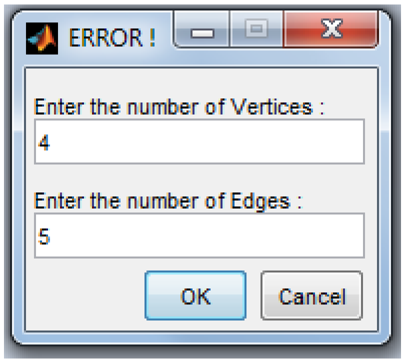
Enter the number of Edges :
25

For instance, for a graph of 5 vertices, the total number of vertices cannot exceed 10. Thus if a value greater than 10 is entered, the following error message is displayed.



Number of Edges cannot exceed -->10 <-- for -->5 <-- Vertices

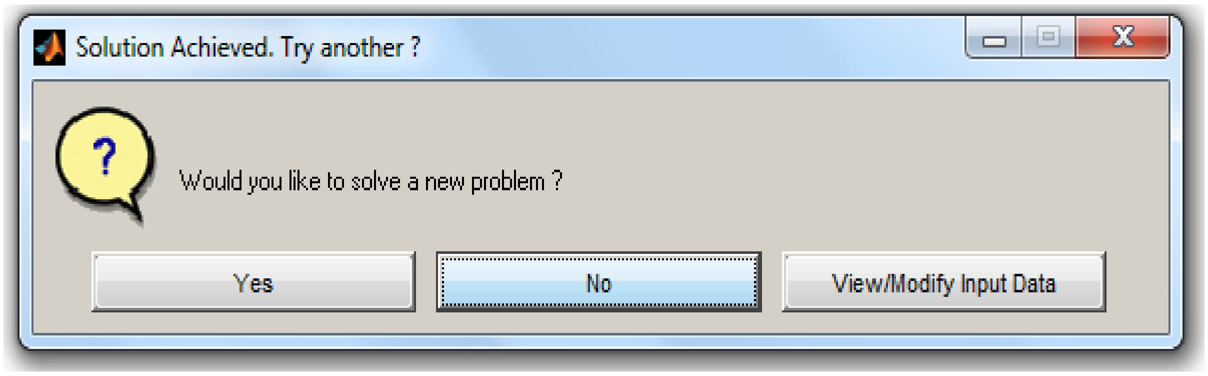
Then user is then again asked to enter the number of vertices and edges.


ERROR !

Enter the number of Vertices :
4

Enter the number of Edges :
5

Also a provisional dialogue box is made to solve another problem, or to modify the given input data, if required by the user on closing the above table:


Solution Achieved. Try another ?

Would you like to solve a new problem ?

FRACTIONAL KNAPSACK ALGORITHM

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the count of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most useful items.

The problem often arises in resource allocation with financial constraints. A similar problem also appears in combinatory, complexity theory, cryptography and applied mathematics.

The decision problem form of the knapsack problem is the question "can a value of at least *V* be achieved without exceeding the weight *W*?"

DEFINITION

In the following, we have *n* kinds of items, 1 through *n*. Each kind of item *i* has a value v_i and a weight w_i. We usually assume that all values and weights are nonnegative. To simplify the representation, we can also assume that the items are listed in increasing order of weight. The maximum weight that we can carry in the bag is *W*.

The most common formulation of the problem is the 0-1 knapsack problem, which restricts the number x_i of copies of each kind of item to zero or one. Mathematically the 0-1-knapsack problem can be formulated as:

Maximize

$$\sum_{i=1}^{n} v_i x_i$$

Subject to

$$\sum_{i=1}^{n} w_i x_i \leqslant W, \qquad x_i \in \{0,1\}$$

The bounded knapsack problem restricts the number x_i of copies of each kind of item to a maximum integer value c_i. Mathematically the bounded knapsack problem can be formulated as:

- Maximize

$$\sum_{i=1}^{n} v_i x_i$$

- Subject to

$$\sum_{i=1}^{n} w_i x_i \leqslant W, \qquad x_i \in \{0, 1, \ldots, c_i\}$$

The unbounded knapsack problem (UKP) places no upper bound on the number of copies of each kind of item.

Of particular interest is the special case of the problem with these properties:

- It is a decision problem,
- It is a 0-1 problem,
- For each kind of item, the weight equals the value: $w_i = v_i$.

Notice that in this special case, the problem is equivalent to this: given a set of nonnegative integers, does any subset of it add up to exactly W? Or, if negative weights are allowed and W is chosen to be zero, the problem is: given a set of integers, does any nonempty subset add up to exactly 0? This special case is called the subset sum problem. In the field of cryptography, the term *knapsack problem* is often used to refer specifically to the subset sum problem.

If multiple knapsacks are allowed, the problem is better thought of as the bin packing problem.

COMPUTATIONAL COMPLEXITY

The knapsack problem is interesting from the perspective of computer science because

- There is a pseudo-polynomial time algorithm using dynamic programming
- There is a fully polynomial-time approximation scheme, which uses the pseudo-polynomial time algorithm as a subroutine
- The problem is NP-complete to solve exactly, thus it is expected that no algorithm can be both correct and fast (polynomial-time) on all cases
- Many cases that arise in practice, and "random instances" from some distributions, can nonetheless be solved exactly.

The subset sum version of the knapsack problem is commonly known as one of Karp's 21 NP-complete problems.

There have been attempts to use subset sum as the basis for public key cryptography systems, such as the Merkle-Hellman knapsack cryptosystem. These attempts typically used some group other than the integers. Merkle-Hellman and several similar algorithms were later broken, because the particular subset sum problems they produced were in fact solvable by polynomial-time algorithms.

One theme in research literature is to identify what the "hard" instances of the knapsack problem look like, or viewed another way, to identify what properties of instances in practice might make them more amenable than their worst-case NP-complete behaviour suggests.

Several algorithms are freely available to solve knapsack problems, based on dynamic programming approach, branch and bound approach or hybridizations of both approaches.

APPLICATIONS

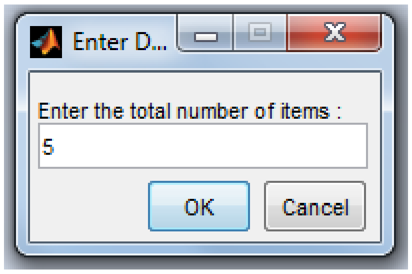
Knapsack problems can be applied to real-world decision-making processes in a wide variety of fields, such as finding the least wasteful cutting of raw materials, selection of capital investments and financial portfolios, selection of assets for asset-backed securitization, and generating keys for the Merkle–Hellman knapsack cryptosystem.

One early application of knapsack algorithms was in the construction and scoring of tests in which the test-takers have a choice as to which questions they answer. On tests with a homogeneous distribution of point values for each question, it is a fairly simple process to provide the test-takers with such a choice.

For example, if an exam contains 12 questions each worth 10 points, the test-taker need only answer 10 questions to achieve a maximum possible score of 100 points. However, on tests with a heterogeneous distribution of point values—that is, when different questions or sections are worth different amounts of points— it is more difficult to provide choices. Feuerman and Weiss proposed a system in which students are given a heterogeneous test with a total of 125 possible points. The students are asked to answer all of the questions to the best of their abilities. Of the possible subsets of problems whose total point values add up to 100, a knapsack algorithm would determine which subset gives each student the highest possible score.

MATLAB IMPLEMENTATION: FRACTIONAL KNAPSACK

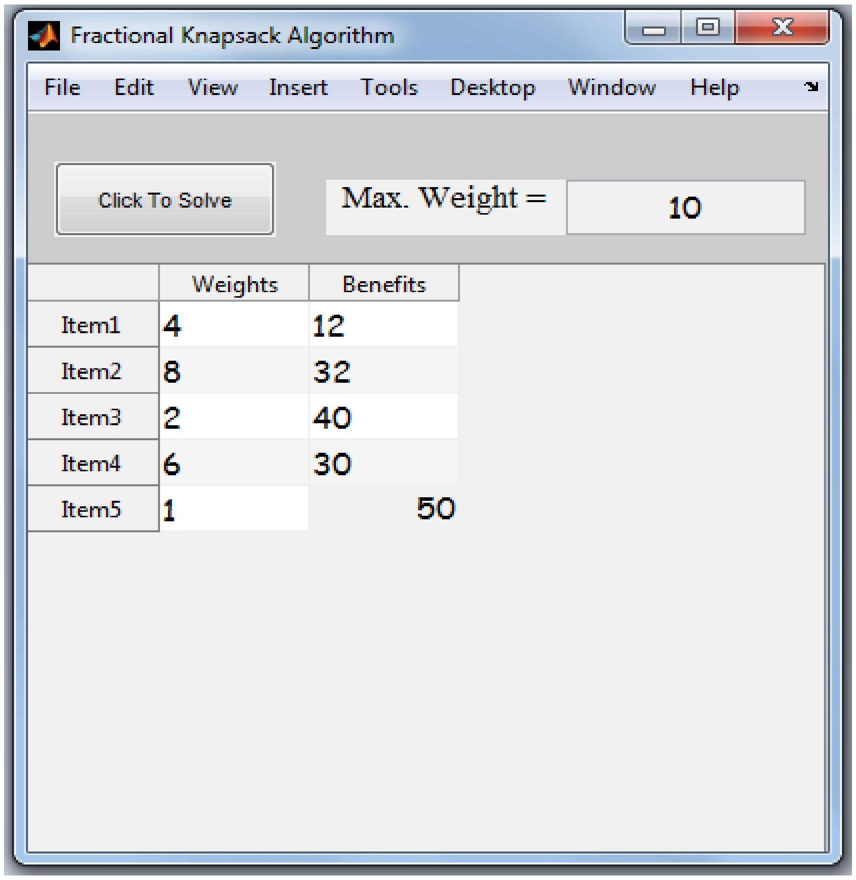
The program first asks the user to enter the total number of items available.



Enter the total number of items :

5

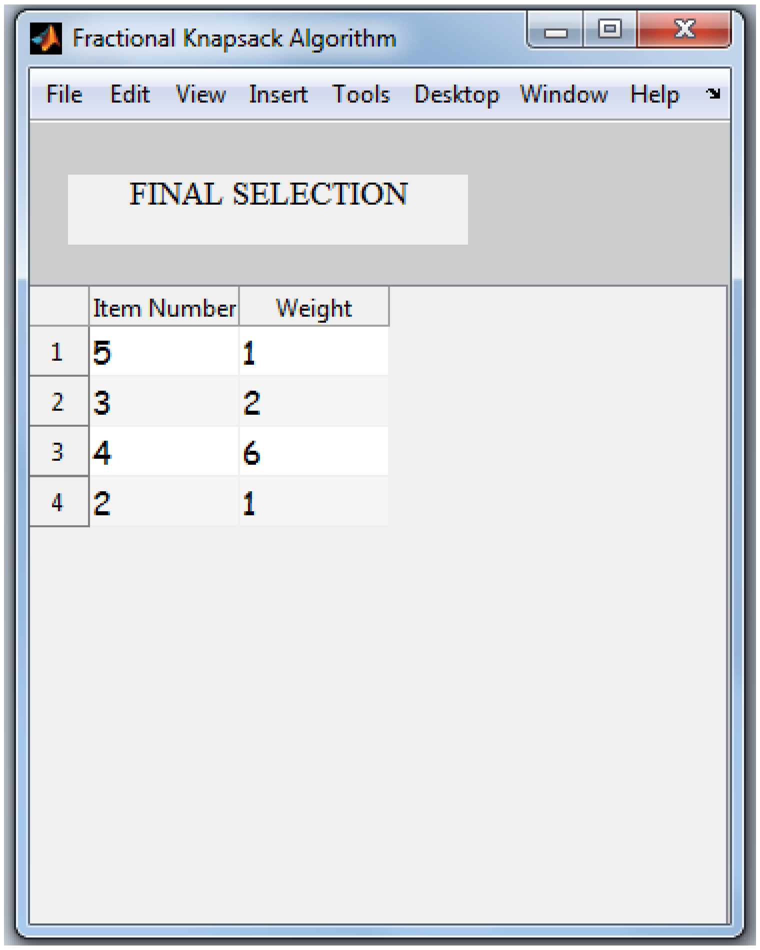
Consider a problem with 5 items to select from. After the 'Ok' button is clicked, an input table is presented to enter the values of benefits and weights of the all the 5 items. Also the value for maximum allowed weight is to be entered at the top as shown below:



Max. Weight = 10

	Weights	Benefits
Item1	4	12
Item2	8	32
Item3	2	40
Item4	6	30
Item5	1	50

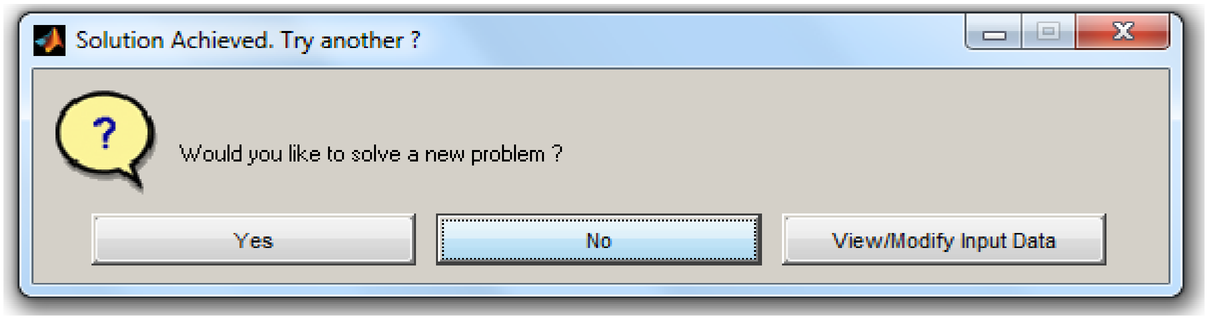
On clicking the 'Click to Solve' button, the final selection of items so as to maximize the benefit, along with their respective weights is shown in a final table.


Fractional Knapsack Algorithm
File Edit View Insert Tools Desktop Window Help
FINAL SELECTION

	Item Number	Weight
1	5	1
2	3	2
3	4	6
4	2	1

This shows that the final selection has 1 unit of item 5, 2 units of item 3 and so on as clearly visible from the above table.

Also a provisional dialogue box is made to solve another problem, or to modify the given input data, if required by the user on closing the above table:



Would you like to solve a new problem ?

TASK SCHEDULING ALGORITHM

Task scheduling is an optimization problem in computer science in which ideal jobs are assigned to resources at particular times. The most basic version is as follows:

We are given **n** tasks {J_1, J_2, ..., J_n} of varying sizes, which need to be scheduled on **m** identical machines, while trying to minimize the makespan. The makespan is the total length of the schedule (that is, when all the jobs have finished processing)

Many tasks in industry and elsewhere require completing a collection of tasks while satisfying temporal and resource constraints. Temporal constraints say that some tasks have to be finished before others can be started; resource constraints say that two tasks requiring the same resource cannot be done simultaneously (e.g., the same machine cannot do two tasks at once). The objective is to create a schedule specifying when each task is to begin and what resources it will use that satisfies all the constraints while taking as little overall time as possible. This is the task scheduling problem. In its general form, it is NP-complete, meaning that there is probably no efficient procedure for exactly finding shortest schedules for arbitrary instances of the problem. Task scheduling is usually done using heuristic algorithms that take advantage of special properties of each specific instance.

PSEUDOCODE

Algorithm taskSchedule(T)

Input: set **T** of tasks w/ start time s_i and finish time f_i

Output: non-conflicting schedule with minimum number of machines

```
m ← 0 {no. of machines}
while T is not empty
    remove task i w/ smallest $s_i$
    if there's a machine j for i then
        schedule i on machine j
    else
        m ← m + 1
        schedule i on machine m
```

PROBLEM VARIATIONS

Many variations of the problem exist, which can be summarized as follows:

- Machines can be related, independent, equal
- Machines can require a certain gap between jobs
- Machines can have sequence-dependent setups
- Objective function can be to minimize the makespan, the Lp norm, etc.
- Jobs may have constraints, for example a job i needs to finish before job j can be started
- Jobs and machines have mutual constraints, for example, certain jobs can be scheduled on some machines only

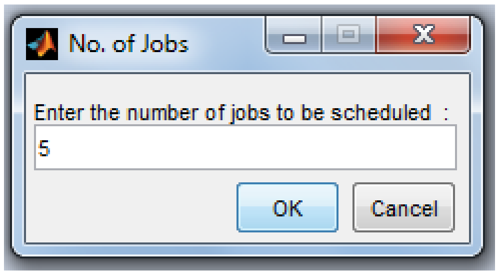
APPLICATIONS

In Manufacturing and Engineering industries, Task scheduling has numerous applications

- Process change-over reduction
- Inventory reduction, leveling
- Reduced scheduling effort
- Increased production efficiency
- Labor load leveling
- Accurate delivery date quotes
- Real time information

MATLAB IMPLEMENTATION: TASK SCHEDULING

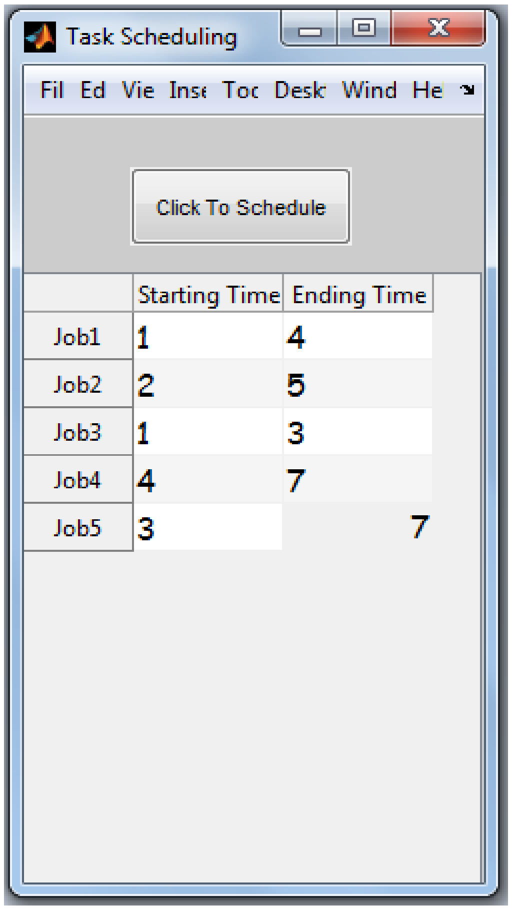
The user is first prompted to enter the total number of jobs to be scheduled.



Enter the number of jobs to be scheduled :

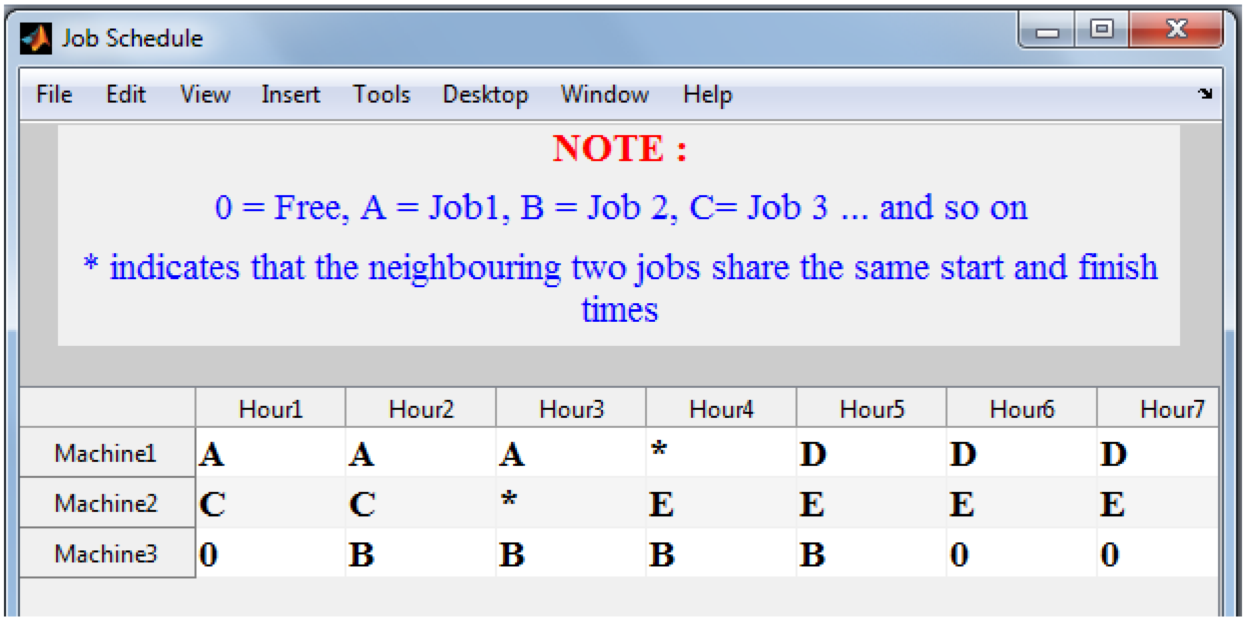
5

On clicking the 'Ok' button, an input table is presented before the user to collect the information about the Starting and Ending times of all the jobs. Consider the following example of 5 jobs having the following start and end times.



	Starting Time	Ending Time
Job1	1	4
Job2	2	5
Job3	1	3
Job4	4	7
Job5	3	7

On clicking the 'Click to Schedule' button, the final scheduled chart is displayed in a tabular format as shown.



NOTE :

0 = Free, A = Job1, B = Job 2, C= Job 3 ... and so on

* indicates that the neighbouring two jobs share the same start and finish times

	Hour1	Hour2	Hour3	Hour4	Hour5	Hour6	Hour7
Machine1	A	A	A	*	D	D	D
Machine2	C	C	*	E	E	E	E
Machine3	0	B	B	B	B	0	0

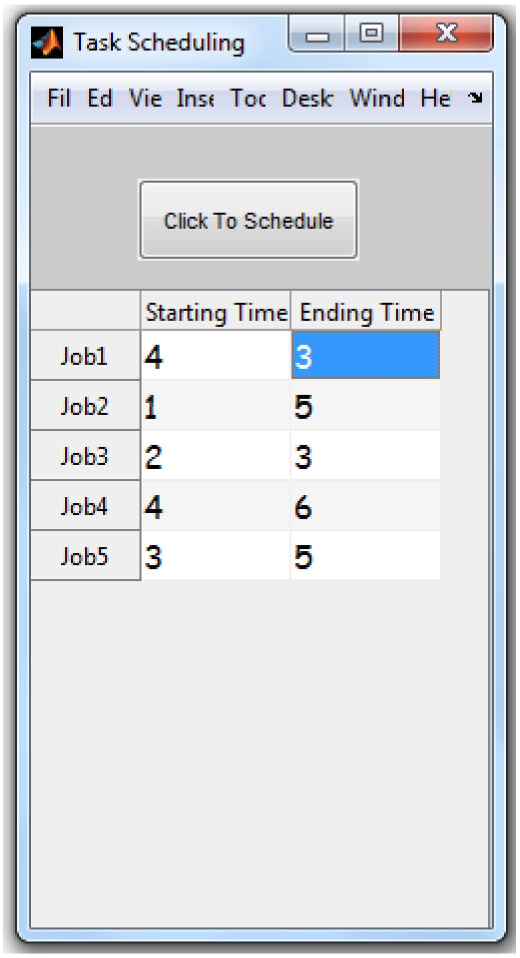
In the above table, Hour1 represent the time from 1o'clock to 2'o'clock, considering the jobs start at 1o'clock. The other necessary information is made available to the user as shown above.

For the above example, minimum 3 machines would have to be used.

ERROR HANDLING

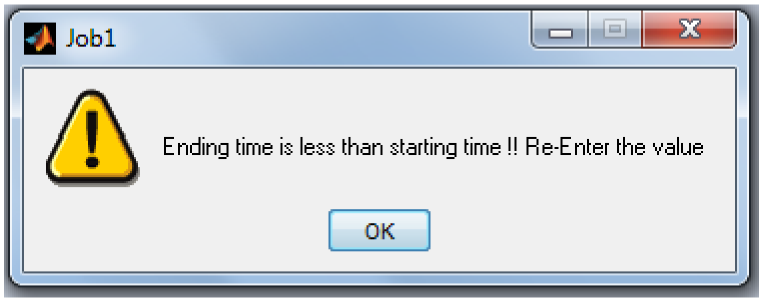
<u>STARTING TIME > ENDING TIME:</u>

Suppose, when entering the values for the starting and ending times, if the user enters the starting time as less than the ending time for a job, an error message is displayed notifying the user about the same.



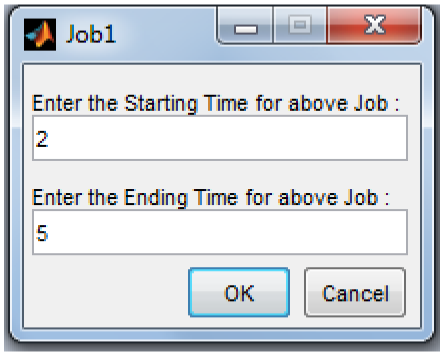
	Starting Time	Ending Time
Job1	4	3
Job2	1	5
Job3	2	3
Job4	4	6
Job5	3	5

As it is clearly visible, the ending time (3) for Job1 is less than the starting time (4). To deal with this, the following error message is displayed.



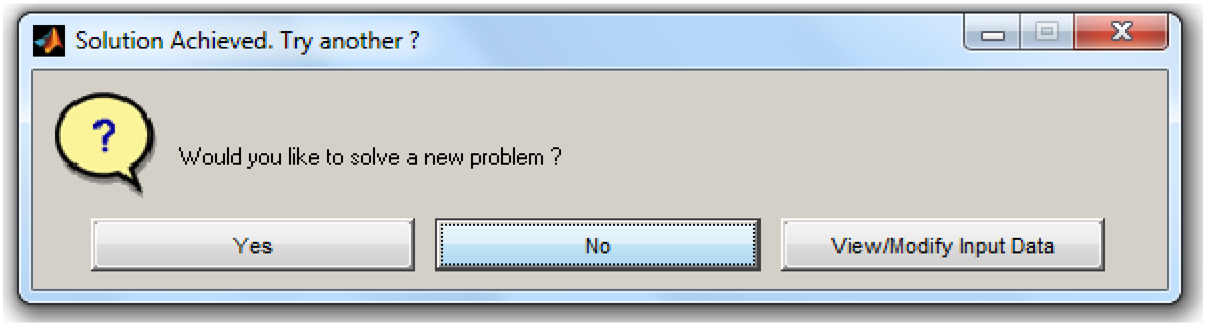
Ending time is less than starting time !! Re-Enter the value

On clicking 'Ok', the following dialogue box is presented to make the user re-enter the values for the start and end times. If the user again enters incorrect values, the same error message is displayed till the user enters proper values.



Enter the Starting Time for above Job :
2
Enter the Ending Time for above Job :
5

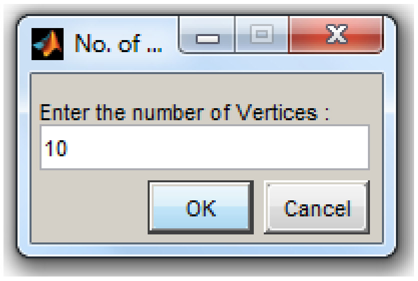
Also a provisional dialogue box is made to solve another problem, or to modify the given input data, if required by the user on closing the output schedule table:



Would you like to solve a new problem ?

COMPLETE GRAPH

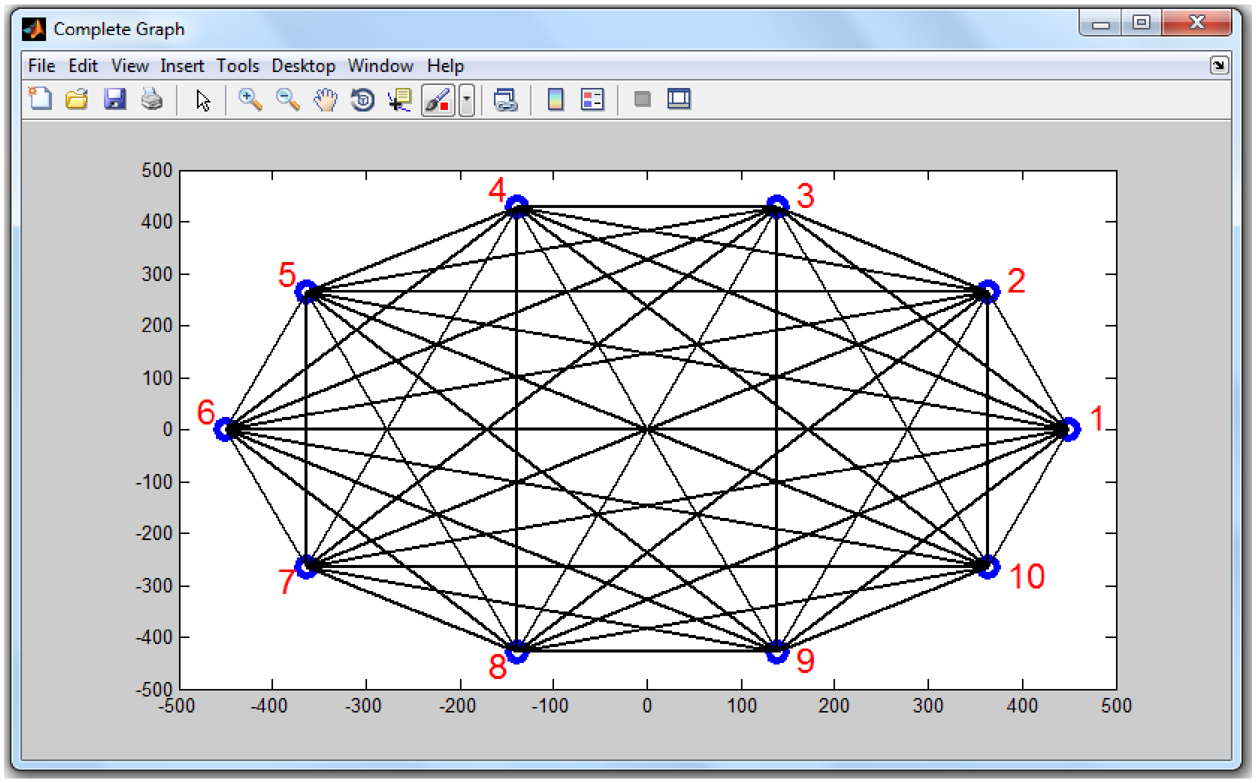
For a given graph of n vertices, it is said to be complete if each vertex is adjacent to every other vertex in the graph. This program inputs the number of vertices of the graph from the user and plots the complete graph.



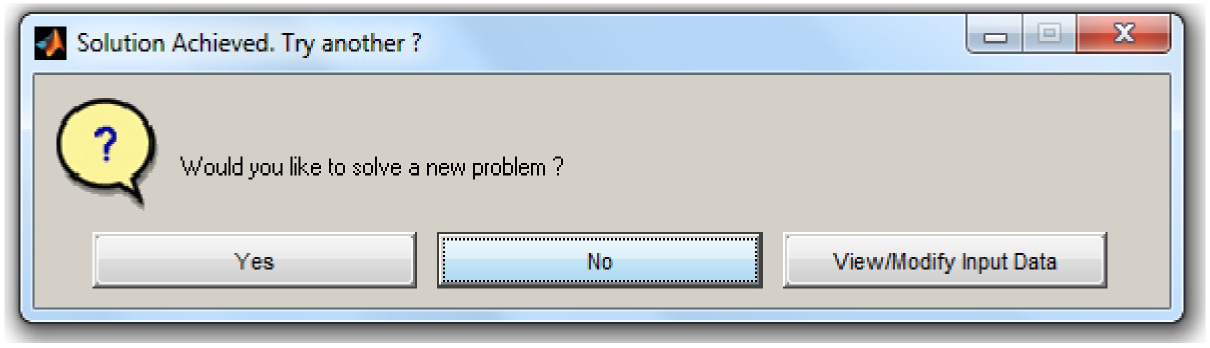
Enter the number of Vertices :

10

Consider a graph having 10 vertices. On clicking ‘Ok’, the complete graph is generated as follows:



Also a provisional dialogue box is made to solve another problem, or to modify the given input data, if required by the user on closing the above figure:

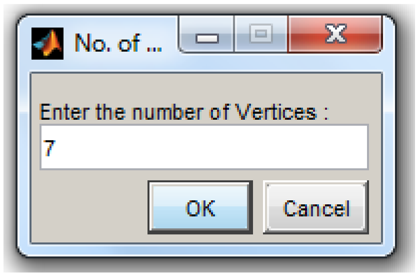


Would you like to solve a new problem ?

GRAPH FROM ADJACENCY MATRIX

An adjacency matrix (A) for a given graph G is a matrix where A(i,j) = A(j,i) = 1 if there exists an edge between the vertices i and j in G. If there exists no edge between the vertices i and j, then A(i,j) = A(j,i) = 0

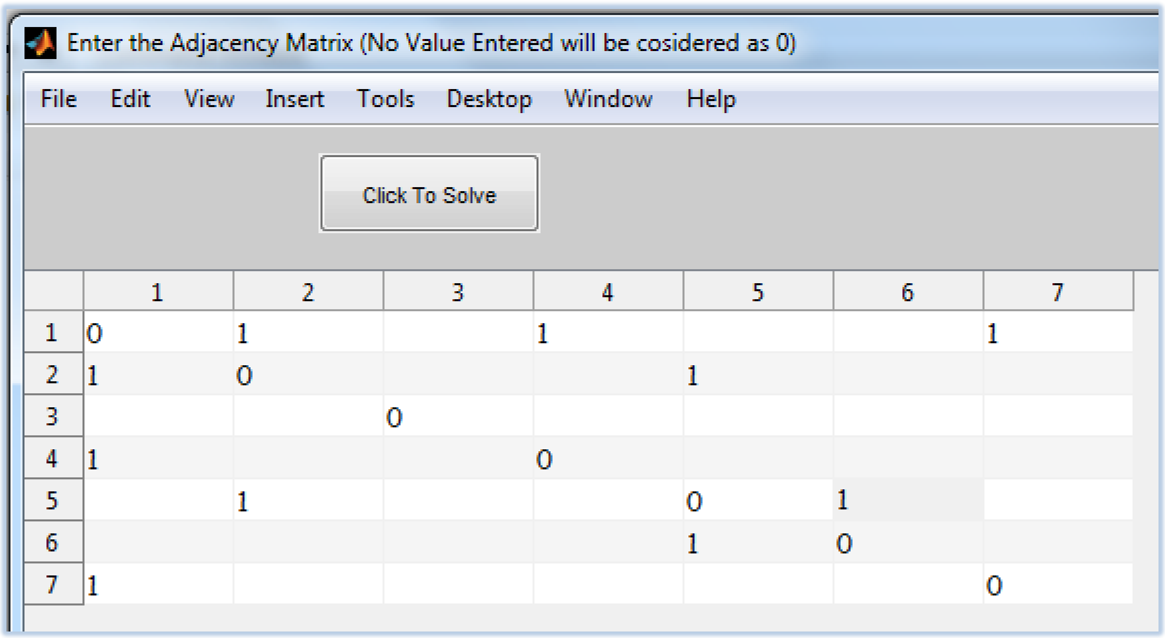
This program takes the adjacency matrix for a graph as input from the user and plots the graph for that matrix.



Enter the number of Vertices :

7

Consider a graph having 7 vertices. On clicking the 'Ok' button, an incomplete adjacency matrix having the diagonal elements as 0 is presented.

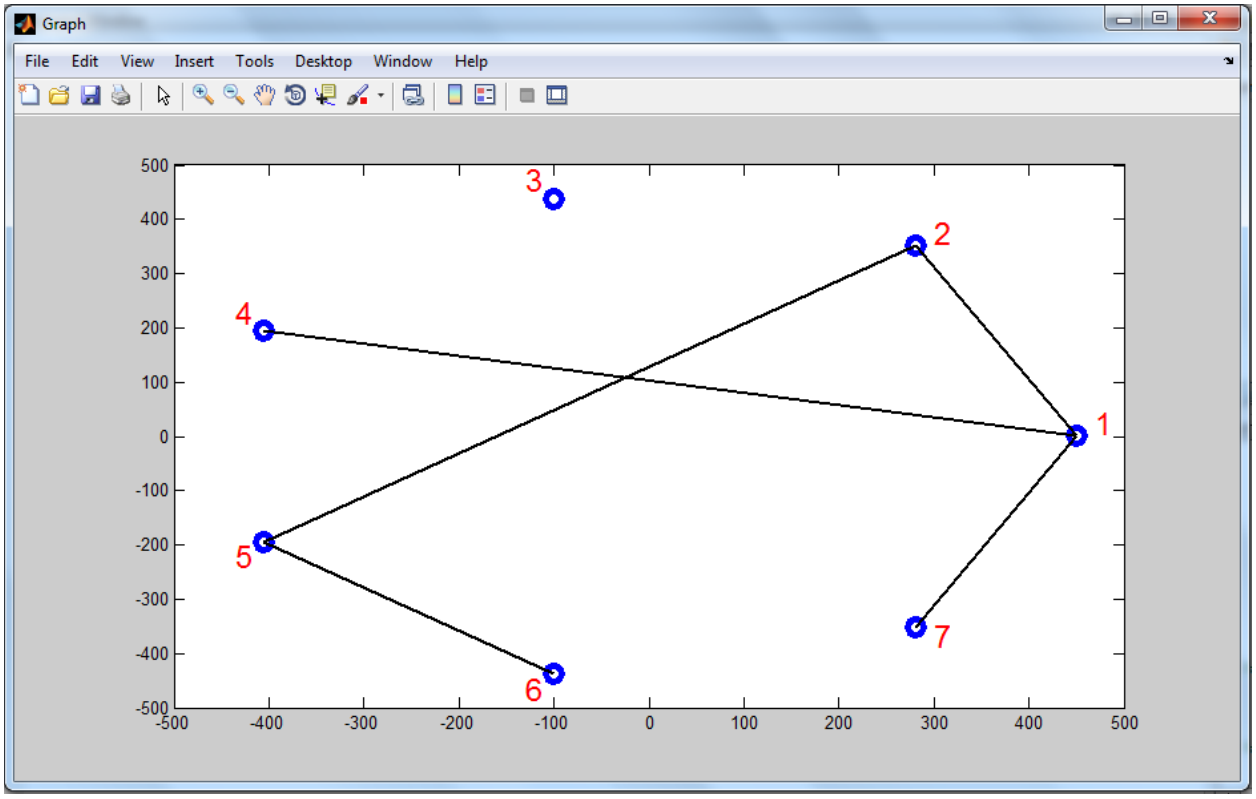


Enter the Adjacency Matrix (No Value Entered will be cosidered as 0)

	1	2	3	4	5	6	7
1	0	1		1			1
2	1	0			1		
3			0				
4	1			0			
5		1			0	1	
6					1	0	
7	1						0

For the sake of reducing the tedious job of entering 0 for all the non-existent edges, a provision has been made to take the default value of a cell as 0 if nothing is entered.

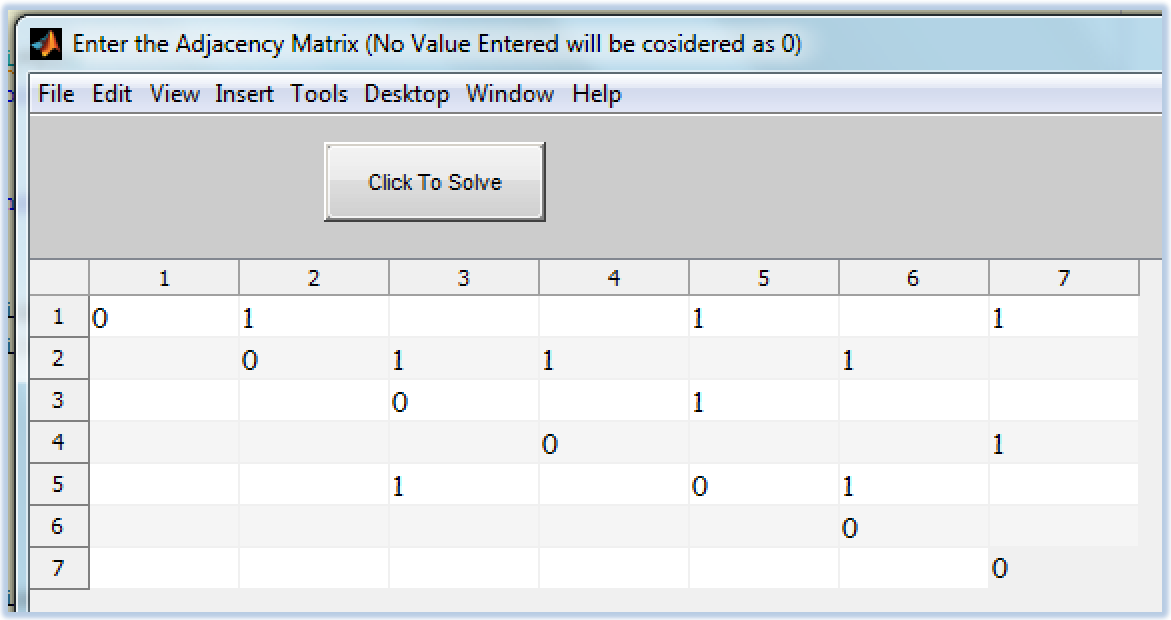
For the above adjacency matrix, the final graph is then shown as follows:



ERROR HANDLING

ASYMMETRIC MATRIX INPUT:

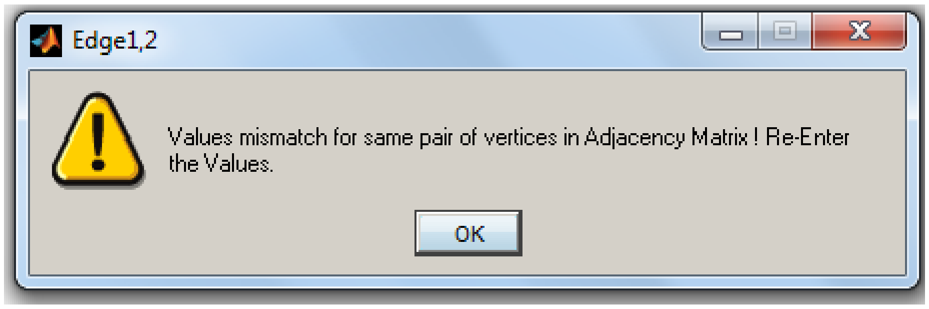
Consider a graph of 7 vertices and the user enters the adjacency matrix as follows:



Enter the Adjacency Matrix (No Value Entered will be cosidered as 0)

	1	2	3	4	5	6	7
1	0	1			1		1
2		0	1	1		1	
3			0		1		
4				0			1
5			1		0	1	
6						0	
7							0

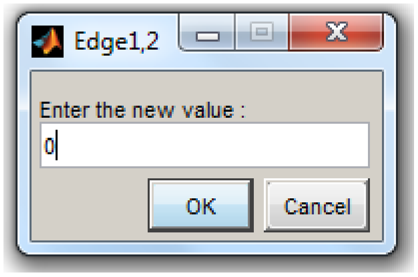
Here, as we can see, the adjacency matrix is not symmetric. An appropriate error message is displayed to notify the user about it.



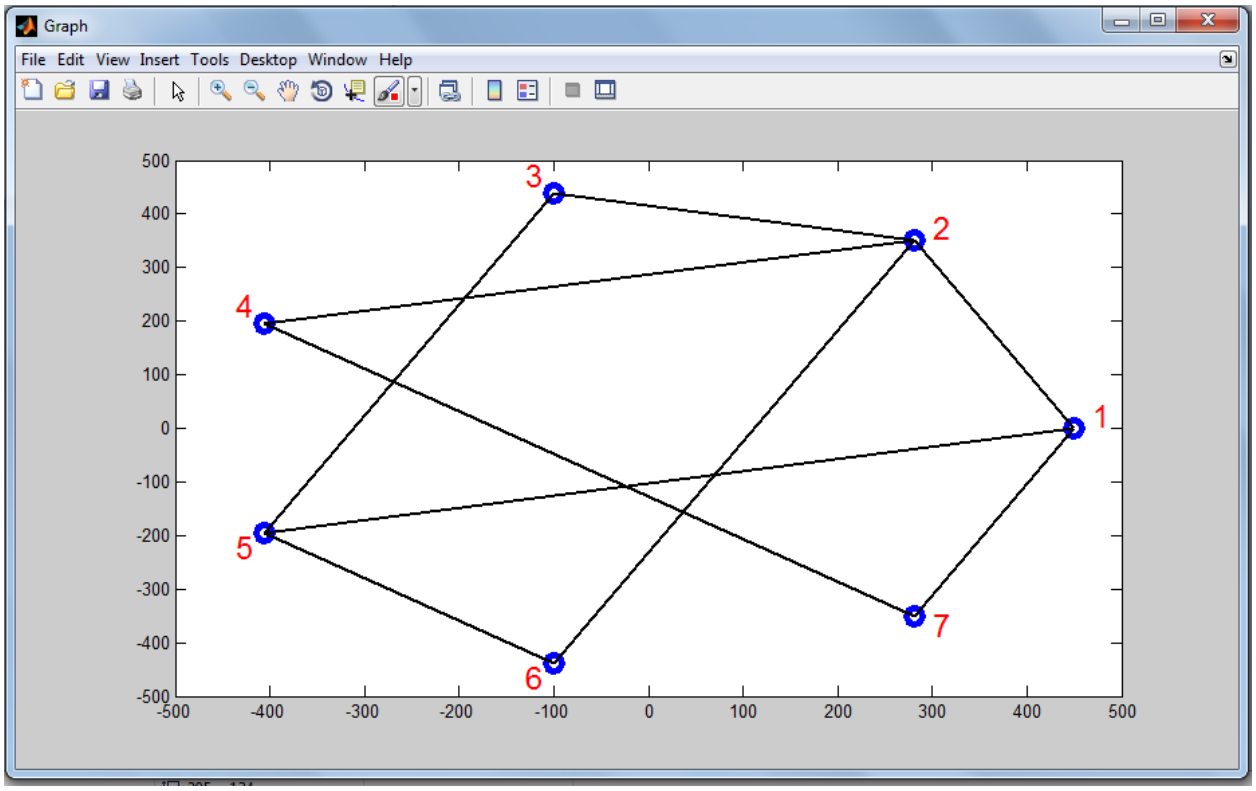
Edge1,2

Values mismatch for same pair of vertices in Adjacency Matrix ! Re-Enter the Values.

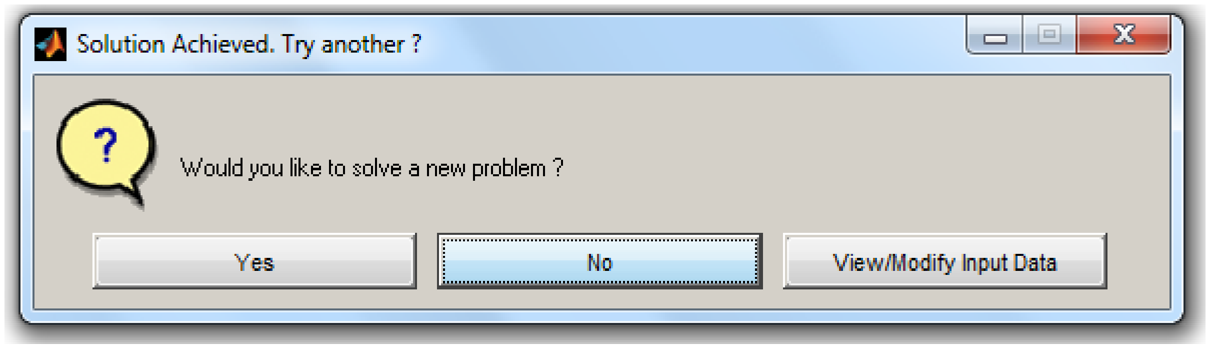
On clicking 'Ok', the new value for the edge 1,2 is asked to the user. This value is then automatically assigned to both A(1,2) and A(2,1), where A is the adjacency matrix.



Similarly, all the anomalies are rectified and then the final graph is drawn as shown below:



Also a provisional dialogue box is made to solve another problem, or to modify the given input data, if required by the user on closing the above figure:


Solution Achieved. Try another ?

Would you like to solve a new problem ?

CONCLUSION

As mentioned previously, Optimization plays an important and influential role in modern day operations research and decision making particularly in areas, where cost cutting and profit-loss are of critical importance. In this regard, the simplex method holds an important place, as it is one of the most widely used algorithms to solve linear optimization problems. However, the Simplex Method computations are particularly tedious and repetitive. To assist a person in this regard, we have developed a user-interactive MATLAB GUI-based application, with instant feedback, which helps to carry out the tedious computations involved. Along with the final solution, the application shows every iteration as one solves it on paper. This feature makes it highly useful for teaching purposes. Graph Theory is used extensively in Computer Science, Biology, Chemistry, Linguistics, Sociology, Networking etc. Some important Graph Algorithms are included which are supported by graphical output, which makes it much more user-interactive and user friendly. The application also includes efficient error handling and runs on any operating system which has MATLAB installed on it.

REFERENCES

1) Hamdy A.Taha , '*Operations Research An Introduction Eighth Edition* ' , 8th Ed., Pearson Education, Inc. , 2007.

2) Brian R. Hunt, Ronald L. Lipsman, Jonathan M. Rosenberg, '*A Guide to Matlab For Beginners And Experienced Users*', Cambridge University Press, 2001.

3) '*Simplex Algorithm'.* Wikimedia Foundation, Inc. http://en.wikipedia.org/wiki/Simplex_algorithm

4) '*Big_M_method'.* Wikimedia Foundation, Inc. http://en.wikipedia.org/wiki/Big_M_method

5) '*Simplex Method'* . The Chinese University of Hong Kong, Department of Mathematics http://www.math.cuhk.edu.hk/~wei/lpch3.pdf

6) *'LP Methods.S1 Dual Simplex Algorithm'* http://www.me.utexas.edu/~jensen/ORMM/supplements/methods/lpmethod/S1_dualsimplex.pdf

7) '*Dual Simplex Algorithm'* *http://www.mcs.csueastbay.edu/~malek/Class/Dualsimplex.html*

8) '*Mathworks*' http://www.mathworks.in

9) '*Kruskal's Algorithm*' http://en.wikipedia.org/wiki/Kruskal's_algorithm

10) '*Floyd Warshall Algorithm*' http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

11) '*Fractional Knapsack Problem*' http://en.wikipedia.org/wiki/Knapsack_problem

12) '*Task Scheduling Algorithm*' http://en.wikipedia.org/wiki/Scheduling_(computing)

13) Guide to GUIs in MATLAB http://blinkdagger.com/category/matlab/