REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Jul 93	THESIS/~~DISSERTATION~~

4. TITLE AND SUBTITLE
A Teaching Tool for Linear Programming

5. FUNDING NUMBERS

6. AUTHOR(S)
Wendy Cook

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT Student Attending: Texas A&M Univ

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/CI/CIA- 93-134

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DEPARTMENT OF THE AIR FORCE
AFIT/CI
2950 P STREET
WRIGHT-PATTERSON AFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release IAW 190-1
Distribution Unlimited
MICHAEL M. BRICKER, SMSgt, USAF
Chief Administration

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

14. SUBJECT TERMS

15. NUMBER OF PAGES
35

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

TABLE OF CONTENTS

Introduction

This paper discusses several main concepts in linear programming, the purpose of which is to supplement the textbook teaching of these concepts to further the students' understanding. These main concepts include bases, extreme points, the simplex algorithm, and sensitivity analysis. Before any of these concepts are discussed, a necessary review of linear algebra is done. The ideas presented here should give the professor additional knowledge about how to successfully teach the students these concepts. Computer programs are developed in MOR/ML in order to aid in these discussions. MOR/ML is a computer programming language which has the syntax necessary to facilitate these concepts. MOR/ML is a "set" structured language which allows for the programming of matrices and sets. This structure makes it possible to formulate the programs dealing with linear programming. The coded programs in MOR/ML and the output for these programs are in Appendix A. The students will be able to understand these concepts by learning and working with the proposed computer programs with the aid of the professor.

Four programs are developed to further the understanding of the concepts of bases and extreme points and how these concepts are related. The emphasis of these programs is to reveal and show the importance of their relationship. The first program listed in Appendix A.1 generates all possible bases. The second program listed in Appendix A.2 generates all possible extreme points. The third program listed in Appendix A.3 generates a feasible extreme point given a feasible basis, and the final program listed in Appendix A.4 generates a feasible basis given a feasible extreme point. These four programs will help the students understand what extreme points and bases are, how they relate to each other, and why

understanding of their relationship is crucial in linear programming.

The discussion of the simplex algorithm is based on a program developed for the purpose of helping the students learn the algorithm. The program is in Appendix A.5. This program and manipulation of it will enable the students to learn the rules and steps of the algorithm. The program will also help students understand how the simplex algorithm is driven by extreme points but uses basic feasible solutions. Students typically have difficulty relating these two concepts. This program is altered in order to show how the students can manipulate the program in order to select the entering and leaving variable. The altered program is in Appendix A.6.

The final set of programs in Appendix A.7 are developed for sensitivity analysis on the original linear program. For all of the programs, the students will find the range for which the change can occur in order to maintain optimality and feasibility of the solution. The specific range is found by manipulating the part of the optimal transformed system which is affected.

The first of these programs performs a sensitivity analysis on the right-hand side (RHS). The second program performs a sensitivity analysis on the basic and non-basic cost coefficients. The last program does a sensitivity analysis on the variable bounds, which are generally greater than or equal to zero.

These programs are developed to be used in parallel with the text in order to aid the students in the understanding of these concepts.

Systems of Equations

Solving systems of linear equations is important when talking about key linear programming concepts such as bases and extreme points. Three systems of linear equations are examined to introduce these two concepts.

The first set of linear equations is of the form **Ax=b** with the x variables unrestricted in sign (uis). For purposes of this discussion, a system of independent linear equations with the same number of equations as unknowns produces a unique solution. An example of a system with two unknowns and two equations with a unique solution is:

$$3x1 + 4x2 = 6$$

$$2x1 + x2 = 4$$

This system of equations gives the solution by simple substitution, $x1 = 2$, $x2 = 0$.

Another system of the same form **Ax=b**, with the x variables unrestricted in sign, is a system with more unknowns than equations. This particular system produces infinite solutions. An example of a system with only one equation with infinite solutions is a line. A line is defined with two unknowns and one equation such as $5x1 + 2x2 = 18$. In order to find the solution to this system, x1 is solved for in terms of x2. For example, $x1 = 18/5 - 2/5x2$. This equation could have also been solved for x2 in terms of x1. Both of these solution spaces produce an infinite number of solutions. A graphical representation is shown in Figure 1.

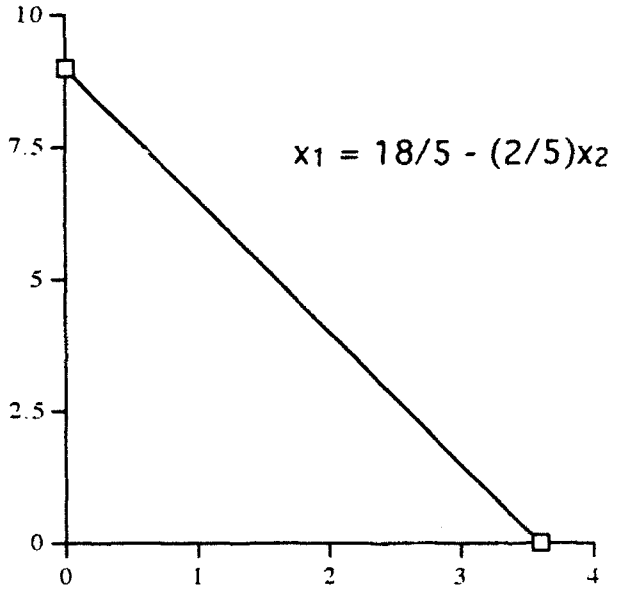

Figure 1.

The next system of linear equations of the form Ax <= b is represented as one equation in Figure 2. For the purpose of this discussion, a system of linear equations is equivalent to a system of constraints in linear programming. The use of one equation versus a system of equations or constraints is used for ease of graphical understanding. This system consisting of one constraint can be transformed into an equality constraint, Ax + s = b. The extension of one constraint to m constraints redefines the feasible solution space. These concepts as applied to one constraint apply to m constraints alike. A slack variable must be added because of the original inequality. The slack variable must be s >=0 because of the nature of the inequality. The concept of slack variables can be explained graphically. For ease of understanding, the following systems of equations each consist of one constraint or line. Figure 3 represents a constraint with the slack variable equal to zero. Figure 4 represents two parallel constraints each

representing a different system where b'<b. The slack variable associated with the system Ax = b' is s'=0, but the slack variable associated with Ax <= b is s which is greater than zero. The value of s is equal to the distance between b and b', so s = b - b'. This figure only represents one value for the slack variable, s. The slack variable associated with the line Ax + s = b in Figure 4 can therefore be thought of as the variable which takes on the value needed to make the original inequality constraint an equality constraint. In order to solve a linear program, the original system of linear inequality constraints is transformed to equality constraints. In order to do this, slack variables are introduced into the inequality constraints.

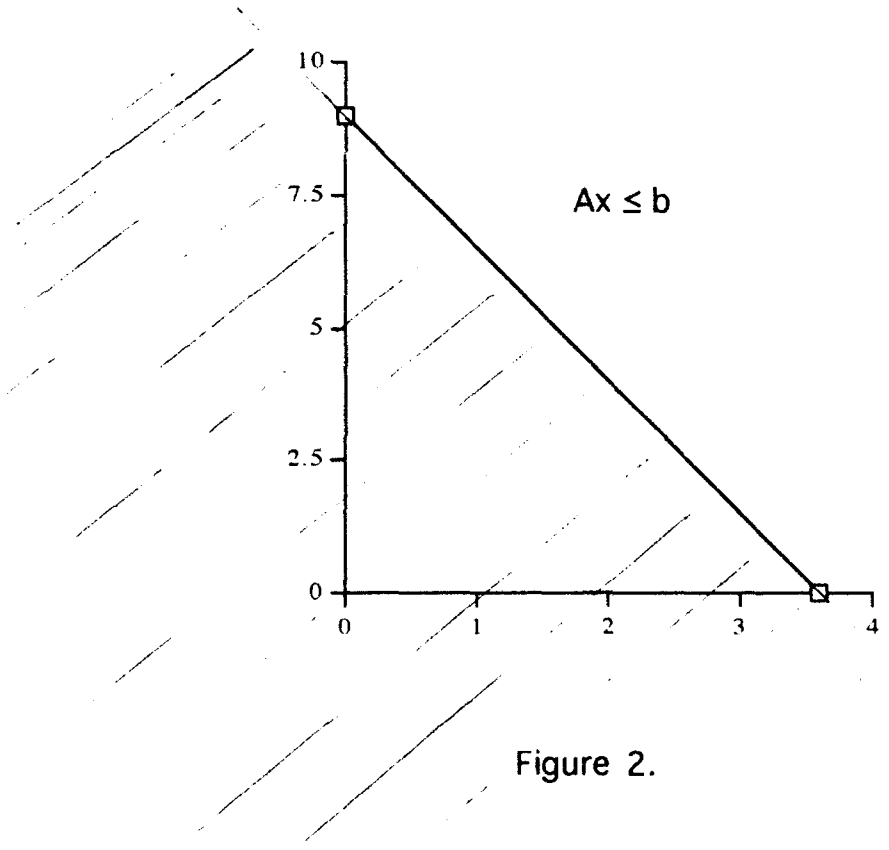

Figure 2.

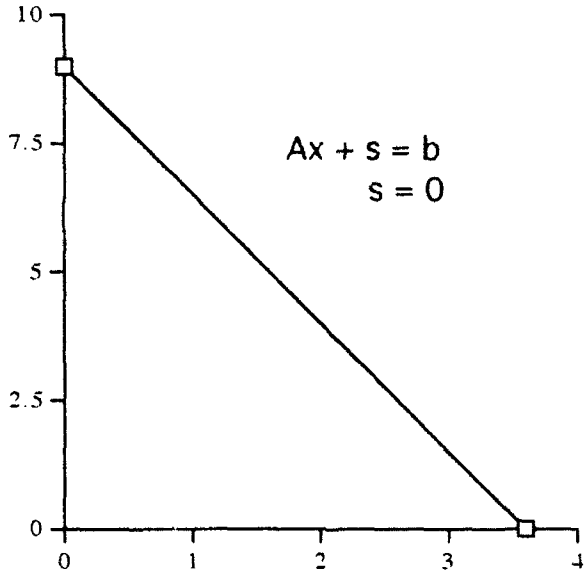

Figure 3.

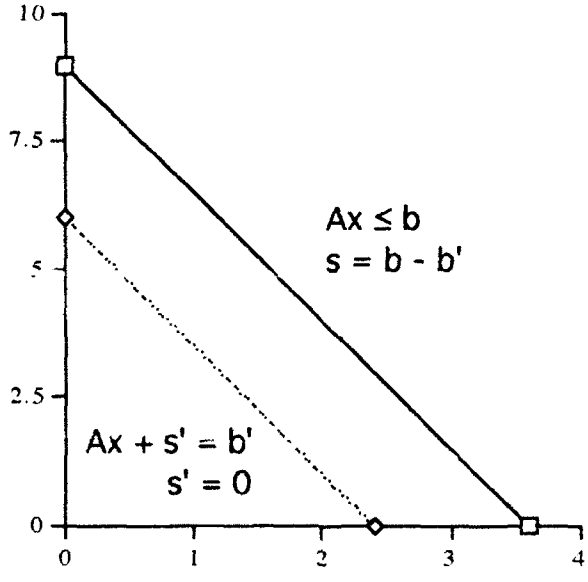

Figure 4.

The final system of linear equations is of the form Ax <= b with both of the x and slack variables greater than or equal to zero. This system consists of one constraint for graphical purposes but is extended

to m constraints for general linear programs. The graphical representation for one constraint is shown in Figure 5.

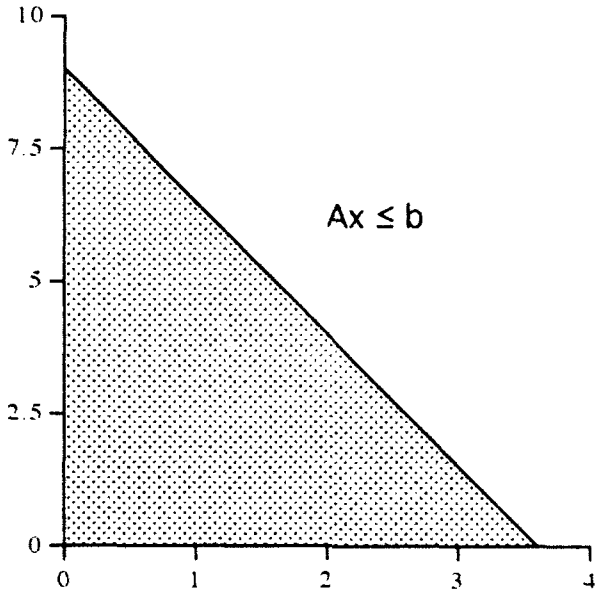

Figure 5.

This particular system introduces the concept of extreme points which are simply the intersection of n independent constraints, where n is the number of x variables. The graphical picture above is two dimensional with axes constraints, x >= 0, and one linear constraint, Ax=b. The intersections of the constraints represent three extreme points. This linear algebra discussion will aid the students' understanding of the following section, bases and extreme points.

Bases and Extreme Points

Bases and extreme points come from a system of equations of a general form, $\mathbf{Ax} = \mathbf{b}$ where $A \in R^m x R^n$, $b \in R^m$, and $x \in R^n$. A basis can be defined as a set of m independent vectors of the system $\{x | \mathbf{Ax} <= \mathbf{b}\}$, which represents a solution space. An extreme point is the intersection of n independent constraints, where n is the number of x variables excluding slack variables. The understanding of the relationship that exists between extreme points and bases is necessary for the complete understanding of the simplex algorithm. For this reason more time should be devoted to the discussion of this relationship.

The common method of introducing students to linear programming and its solution begins with a graphical representation showing all extreme points and the feasible region. The students are them taught the Simplex Method which is an algorithm used to solve linear programs. Hillier & Lieberman begin the discussion of the Simplex Method with a graphical representation of the feasible region (H&L, Figure 4.1). Directly following this graphical representation, Hillier & Lieberman discuss setting up the algorithm and the operations of the algorithm (H&L, Sections 4.2,3). By teaching the simplex algorithm directly following the graphical representation, the students never really understand how the simplex algorithm operates. The concept of traveling from one extreme point to the next until the optimal point is found is discussed from the graphical representation of extreme points, the feasible region, and the gradient of the objective function. Students can understand how the simplex algorithm finds the optimal point in a linear program, but they may not know that the simplex algorithm actually uses bases and not extreme points to get to the optimum. The one-to-one

relationship between extreme points and bases is what allows the explanation of the Simplex Method to be done by a graphical representation of the feasible region. The purpose of this discussion is to fill the "gap" which exists between the graphical representation of the feasible region and the simplex algorithm.

The one-to-one relationship which exists between bases and extreme points depends on whether the solution is non-degenerate. A non-degenerate solution gives a unique extreme point for each basis. On the other hand, a degenerate solution is one in which two or more bases represent the same extreme point. The non-degenerate case is discussed.

When a non-degenerate solution exists, exactly n independent constraints intersect at each extreme point and exactly one basis is associated with this extreme point; therefore, a one-to-one relationship exists between bases and extreme points. In other words, each basis corresponds to one and only one extreme point. In order to actually understand how these two concepts are related, one must generate all possible bases and extreme points.

Generation of bases

The general system of linear equations used for this section is of the form **Ax + s = b** with x,s >= 0. The possible bases are found by taking the combination of n+m variables taken m at a time, where n is the number of x variables and m is the number of constraints or slack variables, s. A point should be noted here. A variable represents a column, so when all possible basis are found, this is in essence the possible combination of columns. The program in Appendix A.1 generates all possible bases. The program requires as input the A matrix which is m x (m+n), the b column vector which is m x 1 transposed in MI_ to a 1 x m row vector, the number

of x variables, and the number of slack variables, s. In order to understand and work with this program, a few MOR/ML commands are introduced.

MOR/ML CONCEPTS

[* *]	Comment statement
{ }	Vector (List)
{{ },{ }}	Matrix (List of lists)
Range[a,b]	Gives a list of whole numbers from a to b
Subsets[a,b]	Gives a list consisting of lists of length b. These lists are the possible combinations of a things taken b at a time
Cardinality[a]	Gives the number of elements in a list
Extract[a,b]	Extracts the bth element from the list, a
GetColumn[a,b]	Gives a list of b columns from the matrix a
Inverse[a]	Gives the inverse of the matrix, a
Append[a,b]	Gives a list with b added to the end of a
Insert[a,b,c]	Inserts into the list, a, the number, b, into the cth position
Drop[a,b]	Gives a list with b elements subtracted from the list, a
Print[" "]	Prints elements enclosed in " "

An exercise left for the students is to adjust the program to give only basic feasible solutions. A feasible basis gives only positive solutions for all of the variables. In order to obtain only basic feasible solutions, eliminate any bases which give negative solutions for one or more variables.

Generation of extreme points

In order to understand the one-to-one relationship between extreme points and bases, all possible extreme points need to be generated from the same system of linear equations that generated all possible feasible bases. In order to generate all possible extreme points, the m+n constraints including non-negativity constraints and the x variables excluding slack variables are required. The procedure is to find all possible combinations of m+n constraints taken n at a time where m+n is the total number of constraints including non-negativity, and n is the number of x variables excluding the slack variables. The solution is then found for each of the n independent constraints giving an extreme point, which is the intersection of these n independent constraints. An MOR/ML computer program listed in Appendix A.2 is written to generate all possible extreme points. This program needs as input the A matrix which is (m+n) x n, the b column vector which is (m+n) x 1 transposed to a 1 x (m+n) row vector for use in ML, the number of x variables, and the number of constraints. An exercise left for the student is to change the program in order to list only the feasible extreme points. An extreme point is infeasible if one or more constraints are violated by the extreme point.

In the case of non-degeneracy, an extreme point is associated with one and only one basis. Once an extreme point is known, the respective basis is also known and vice versa. The constraints which represent an extreme point are active and of the form Ax + s = b where the respective slack variables, s, corresponding to the active constraints are equal to zero. The additional non-negativity constraints for two dimensions for example are x1 >= 0 and x2 >= 0 have no slack variables associated with them. Therefore, the total number of constraints is m+n. When an

extreme point lies on an axis, say the x1 axis, the respective intersecting constraints in two dimensions contain x2 = 0 and one of the m constraints. The basis associated with an extreme point does not contain the slack variables associated with the respective intersecting constraints because these are equal to zero. When an extreme point lies on an axis, say the x1 axis, the basis corresponding to this extreme point does not contain the variable x2 because the constraint x2 >= 0 is tight. Two MOR/ML programs are constructed to further the understanding of this one-to-one relationship between bases and extreme points.

Generation of an extreme point given a basis

The program listed in Appendix A.3 generates an extreme point for a given basis. This program requires several input quantities. These include the A matrix which contains the slack variables so it has dimensions m x (m+n), the b column vector which is m x 1 and is transposed to a 1 x m row vector in MOR/ML, the given basis which is a 1 x m row vector, the number of x variables excluding slack variables, and the number of constraints which is the number of slack variables added. The only additional MOR/ML command required for understanding of this program which has not been introduced is the APPEND command. APPEND[a,b] simply adds on to the list, a, the element b. This program assumes the given basis is feasible and gives the basic feasible solution along with the extreme point associated with the given basis.

Generation of a basis for a given extreme point

The program listed in Appendix A.4 generates a basis for a given extreme point. The required input quantities are the matrix A which is m

x (m+n), the b column vector which is m x 1 and is transposed to a row vector for use in MOR/ML, the number of x variables excluding the slack variables, the number of constraints, and the given extreme point which is 1 x n. This program assumes the given extreme point is feasible so the generated basis and basic feasible solution are also feasible. These two programs will aid the students in understanding the relationship between a basis and an extreme point. For an exercise, the student will determine the feasible bases and extreme points and use the MOR/ML programs to further the understanding of these concepts. The case of degeneracy exists when an extreme point is represented by two or more bases resulting in one or more basic variables equaling zero. Therefore, the one-to-one relationship does not exist as in the case of non-degeneracy. The MOR/ML programs will assist the students in understanding the concept of degeneracy. The students should first generate all possible bases and extreme points and determine which are feasible. Next, the program in Appendix A.3, generation of an extreme point given a feasible basis, should be run. This will give the extreme point associated with each basis. What the students will observe is that the degenerate extreme point is generated by two or more bases.

Simplex Algorithm

The simplex algorithm is presented by a program developed to further the students' understanding of the algorithm. This understanding will come by manipulation of the program in order to learn the simplex rules by observation of the results. The simplex algorithm can be thought of as a function which calls three separate functions, getEnteringVariable, getLeavingVariable, and doPivot. This program is listed in Appendix A.5. This discussion will be based extensively on the program itself. This program will allow the students to change a basis, the collection of basic variables. The students can also select the entering variable, which is the function call getEnteringVariable, and examine the resulting change in the objective function value. The students should observe which of the possible entering variables give a better objective function value. Lastly, the students can also select a leaving variable, which is the function call, getLeavingVariable, arbitrarily and check for infeasibility. In order to understand how the leaving variable should be selected, the students should continue to select a leaving variable until infeasibility occurs, assuming no unbounded solutions and a non-degenerate solution.

Introduction to the simplex algorithm

The simplex algorithm is based on the standard system of linear constraints, $\mathbf{Ax} + \mathbf{s} = \mathbf{b}$, $x,s >= 0$. Where s is the collection of slack variables. This system of linear equations consists of rows and columns, where the rows represent the constraints and the columns represent the variables. A representation of a standard linear program in vector and matrix notation is the following:

optimize $Z = C_B * X_B + C_n * X_n$

subject to $B * X_B + N * X_n = b$

$X_B >= 0, X_n >= 0$

Where C_B is a vector of basic variable cost coefficients. These variables are the original variables--no slacks.

C_n is a vector of non-basic variable coefficients. These variables are the slack variables.

X_B is a vector of basic variables.

X_n is a vector of non-basic variables--slack variables.

B is a matrix of constraint coefficients of the basic variables.

N is a matrix of constraint coefficients of the non-basic variables.

b is the right-hand side vector of the constraints.

The simplex algorithm is an iterative algorithm in which the basis changes at each iteration. The algorithm is continued until an optimal basis is found. Each constraint or row is representative of a basic variable. The basis is changed by simple row operations. Each iteration consists of the completion of three basic function calls from the main function, simplexAlgorithm. The first function call, getEnteringVariable, selects which of the variables not in the current basis, also called non-basic variables, should enter the basis. The second function call, getLeavingVariable, determines which of the variables currently in the basis, also called basic variables, should exit the basis and become non-basic. The third function, doPivot, performs row operations to change the basis. Once the optimal basis is found the algorithm is terminated.

When the algorithm begins, the linear program can be represented as a transformed system in vector and matrix notation. This standard notation also applies to the format of the system at optimality. The three function calls in the program listed in Appendix A.5 are required to complete an iteration and each use selected portions of the transformed system. The transformed system is shown below:

optimize $Z = (C_n - C_B B^{-1} N) * X_n + C_B B^{-1} b$

subject to $I * X_B + B^{-1} N * X_n = B^{-1} b$

$X_B >= 0, X_n >= 0$

Where $(C_n - C_B B^{-1} N)$ is the vector of reduced costs.

$C_B B^{-1} b$ is the objective function value

$B^{-1} b$ is the vector of the optimal right-hand side (RHS) values.

I is the identity matrix

$B^{-1} N$ is the matrix of non-basic variable constraint coefficients.

Selection of the entering variable

The portion of the transformed system considered in the selection of the entering variable is $(C_n - C_B B^{-1} N) * X_n$, the reduced costs. The function call, getEnteringVariable, requires as input the vector of reduced costs above and returns the pivot column number. This number corresponds to the position of the selected entering variable in the reduced costs vector. The function returns a zero if the optimal has been obtained. By selecting which of the non-basic variables should enter the basis, the students should observe different changes in the objective function value. Once all

possible non-basic variables have been selected to enter the basis, the rule for an entering variable should be apparent. A graphical representation will also reveal which variable should enter based on the gradient of the objective function and which extreme point is optimal. The simplex program can be edited in order to select the entering variable and examine the results.

Selection of the exiting variable

The portions of the transformed system considered in the selection of the exiting variable include $B^{-1}N_i$, the i^{th} column of the non-basic entering variable and $B^{-1}b$. The function, getLeavingVariable, requires as input these two vectors and returns the row position of the selected leaving variable. If the problem is unbounded, the function returns a zero. The students can select the leaving basic variable by editing the simplex program. The leaving variable corresponds to a particular constraint. Once the students select the variable to leave, a pivot is performed and the students can observe the results. If the wrong leaving variable is selected and one or more constraints are violated, the solution will be infeasible, assuming non-degeneracy and bounded solutions. A graphical representation will show which variable should leave given an entering variable. The rule for selecting the leaving variable should become apparent.

A graphical representation will reveal how a change of basis in the simplex algorithm relates to moving from one extreme point to another. A pivot is the movement from one extreme point to another. The function call, doPivot, performs the necessary row operations in order to move from one extreme point to another and update the current basis. The

changing of a basis involves inclusion of the selected entering variable into the basis and the elimination of the selected exiting variable from the basis. The function, doPivot, requires as input the column position of the entering variable, the row position of the exiting variable, and the i^{th} column of $B^{-1}N$ corresponding to the entering variable. The students should observe that each change of basis moves from one extreme point to another in the case of non-degeneracy. In order for the students to observe that the simplex algorithm changes a basis at each extreme point, the students are urged to graphically follow the algorithm at each iteration. As the students perform this exercise, they should observe that a unique basis is associated with each individual extreme point. Therefore, the concept that the simplex algorithm is driven by extreme points but uses bases in order to travel from one extreme point to another should be better understood. It should be clear to the students that the graphical and tableau representation of the simplex algorithm are not the same concept, but are directly related. In the process of this exercise the students will also observe that in the case of a degenerate point, a change of basis will remain at the same extreme point. This happens because more than one bases represent the same extreme point.

Sensitivity Analysis

Sensitivity analysis is a vital tool in the analysis of any linear program. MOR/ML programs are developed in order to help students understand what sensitivity analysis is and how simple the concept is to learn. When a large linear program is being solved and interpreted, one may wish to know how much the original program can change without changing the optimal basis or becoming infeasible. The purpose of sensitivity analysis is to indicate how much a particular part of the original program can change before the problem needs to be resolved. Three sections of a linear program are discussed in this section and in the programs.

The right-hand side can easily change once a problem is solved. If the problem is very large it may take days and money to resolve the slightly modified linear program. Instead a range is found on how much a right-hand side element can change and still maintain the feasibility and the current optimal basis. This is the purpose of sensitivity analysis. Another part of a linear program which is discussed is that of the original cost coefficients. A range can also be developed for basic and non-basic cost coefficients. The last item discussed is that of the bounds of basic and non-basic variables. Sensitivity analysis can only indicate a range for which one element can change. When changes on two or more elements occur, this becomes parametric analysis. A range can also be determined for which the two or more elements can change, but once the range is violated, one can only give an upper or lower bound on the objective function value. When two or more elements are changed outside of the optimal range, the problem may or may not remain optimal.

Additional MOR/ML commands

MaxList[a]	Returns the maximum element in the list, a, and the position in which the maximum occurs in the list
MinList[a]	Returns the minimum element in the list, a, and the position in which the maximum occurs in the list
Map[a,b]	Applies the function, a, to the list, b

This discussion uses the same notation as discussed above for the general linear program and transformed/optimal linear program.

Sensitivity analysis on the right-hand side

A range can be determined for which a particular element of the original right-hand side can change to maintain feasibility and the current optimal basis. In order for the current optimal basis to remain feasible, the right-hand side (RHS) must be greater than or equal to zero after the change occurs to a particular element. The current optimal RHS is $B^{-1}b$. To ensure that the RHS will remain feasible after a change occurs, the change must be in some range. In order to determine the range in which the change can occur, the b vector becomes $b + \theta d$. θ represents how much the original element of the RHS changes, and d is a vector of 0's and a 1 in the position of the changing element. The current RHS then becomes $B^{-1}b + \theta B^{-1}d$ which must be greater than or equal to zero. With some algebra, the range on θ becomes $\theta >= -B^{-1}b/B^{-1}d$. Two cases arise here.

1. $B^{-1}d$ is > 0. $\theta >= -B^{-1}b/B^{-1}d$ the maximum of these inequalities is the lower bound for θ. θ can increase without bound.

2. $B^{-1}d$ is < 0. θ < = $-B^{-1}b/B^{-1}d$ the minimum of these inequalities is the upper bound for θ. θ can decrease without bound.

It should be obvious that when $B^{-1}d$ is equal to zero, θ can increase or decrease without bound.

The program listed in Appendix A.7 performs a sensitivity analysis on the RHS and requires as input the current B^{-1}, the current RHS, and the vector d indicating which element of the original RHS is to be changed. The determination of the bounds on θ are the same for a minimum or maximum optimization problem.

Sensitivity analysis on the cost coefficients

The determination of θ for the cost coefficients depends on whether the type of problem is a minimization or maximization problem. For both types of problems, the part of the optimal system which will be affected by the change is the reduced costs. These are $(C_n - C_B B_{-1} N)*X_n$ at optimality. Non-basic and basic original cost coefficients can be changed. Changes made to any of the cost coefficients will not affect the feasibility of the problem, only the optimality of the problem.

The changes to non-basic cost coefficients changes C_n to $C_n + \theta d_n$ in the reduced costs, thus extending the reduced costs to $\{ (C_n + \theta d_n) - C_B B^{-1} N \}$. The range of θ depends on if the problem is a minimization or maximization problem.

MINIMUM/NON-BASIC COST COEFFICIENTS

For a minimization problem, the reduced costs are greater than or equal to zero at optimality. A reduced cost can equal zero, in which case the particular problem would have multiple optimal solutions. The reduced costs must remain > = 0 in order to maintain optimality, thus, $\{ (C_n + \theta d_n) - C_B B^{-1} N \} >= 0$. With some algebra, $\theta >= C_B B^{-1} N - C_n$. In other words, θ can only decrease by the value of the variable's reduced cost. θ can increase without bound.

MAXIMUM/NON-BASIC COST COEFFICIENTS

For a maximization problem, the reduced costs are less than or equal to zero at optimality. Thus, $\{ (C_n + \theta d_n) - C_B B^{-1} N \} <= 0$ for optimality. With some algebra, $\theta <= C_B B^{-1} N - C_n$. θ can only increase by the value of the variable's reduced cost. θ can decrease without bound.

MINIMUM/BASIC COST COEFFICIENTS

The reduced costs are greater than or equal to zero at optimality. The element of the reduced costs which changes is C_B. Thus, the reduced cost are extended to incorporate $C_B + \theta d_B$ to $\{ C_n - (C_B + \theta d_B) B^{-1} N \} >= 0$. With some algebra, this reduces to $\theta <= (C_n - C_B B^{-1} N)/B^{-1} N$ where $B^{-1} N$ is a vector of the coefficients of the non-basic variables in the row of the basic variable. Two cases are considered here.

1. $B^{-1} N > 0$. $\theta <= (C_n - C_B B^{-1} N)/B^{-1} N$ the minimum of these inequalities is the upper bound for θ. θ can decrease without bound.

2. $B^{-1}N < 0$. $\theta >= (C_n - C_B B^{-1}N)/B^{-1}N$ the maximum of these inequalities is the lower bound for θ. θ can increase without bound.

It should be obvious that if $B^{-1}N$ is equal to zero, θ can increase or decrease without bound.

MAXIMUM/BASIC COST COEFFICIENTS

The reduced costs are less than or equal to zero at optimality. The element of the reduced costs which changes is C_B. Thus, the reduced cost are extended to incorporate $C_B + \theta d_B$ to $\{ C_n - (C_B + \theta d_B)B^{-1}N \} <= 0$. With some algebra, this reduces to $\theta >= (C_n - C_B B^{-1}N)/B^{-1}N$ where $B^{-1}N$ is a vector of the coefficients of the non-basic variables in the row of the basic variable. Two cases are considered here.

1. $B^{-1}N < 0$. $\theta <= (C_n - C_B B^{-1}N)/B^{-1}N$ the minimum of these inequalities is the upper bound for θ. θ can decrease without bound.
2. $B^{-1}N > 0$. $\theta >= (C_n - C_B B^{-1}N)/B^{-1}N$ the maximum of these inequalities is the lower bound for θ. θ can increase without bound.

It should be obvious that if $B^{-1}N$ is equal to zero, θ can increase or decrease without bound.

The MOR/ML program listed in Appendix A.7 performs a sensitivity analysis on the non-basic and basic cost coefficients. The non-basic cost

coefficient analysis requires as input the vector of reduced costs and the position in the reduced cost's vector of the variable of interest. The basic cost coefficient analysis requires as input the vector of reduced costs and the vector of coefficients of non-basic variables from the row of the basic variable.

Sensitivity Analysis on the variable bounds

A standard linear program requires that basic and non-basic variables be greater than or equal to zero. The range for which a variable bound can change is independent on the type of optimization problem. A sensitivity analysis can be performed in order to see how much a bound can change on a variable before the problem is no longer optimal and feasible.

BASIC VARIABLE BOUNDS

The part of the optimal program affected by changing the bound on a basic variable is $I*X_B + B^{-1}N*X_n = B^{-1}b$. The bounds on basic variables are $>= 0$ for a linear program. The changing of the bounds from $X_B >= 0$ to some bound other than 0 will affect the feasibility of the optimal solution. The lower bound for a basic variable can decrease without bound. The only bound restricting the basic variable is the upper bound which specifies how high the basic variable can be restricted. In the optimal system the non-basic variables are equal to zero so are unaffected by the bound of the basic variable. In order to see how high the bound on a basic variable can be, the part of the affected optimal system can be reduced to $I*X_B = B^{-1}b$. In order to maintain feasibility and optimality, the upper bound on X_B cannot be greater than $B^{-1}b$, the optimal value of the basic

variable, or the problem becomes infeasible.

NON-BASIC VARIABLE BOUNDS

The part of the optimal system which is affected by the bound of the non-basic variables is $I*X_B + B^{-1}N*X_n = B^{-1}b$ and $X_n >= 0$. When the bound changes on a non-basic variable, the value of the respective basic variable must change accordingly in order to maintain feasibility. In order to find the upper and lower bounds on a non-basic variable, the affected part of the optimal system becomes $X_n = (B^{-1}b - X_B)/B^{-1}N$ where $B^{-1}N$ is the column of the $B^{-1}N$ matrix corresponding to the non-basic variable of interest, and X_B is the lower bound of the corresponding basic variable. In order to maintain feasibility, $(B^{-1}b - X_B)/B^{-1}N$ must be greater than or equal to zero. Therefore, the bound on X_n cannot exceed $(B^{-1}b - X_B)/B^{-1}N$ because $(B^{-1}b - X_B)/B^{-1}N - X_n$ must be >= 0. The bound on the non-basic variable can be decreased to a lower bound and increased to an upper bound.

Upper Bound for a Non-Basic Variable

The upper bound for a non-basic variable cannot exceed the minimum of the elements of $(B^{-1}b - X_B)/B^{-1}N$ if the $B^{-1}N$ elements are positive. The positive coefficients of positive non-basic variables will drive the corresponding basic variable to its lower bound. If the $B^{-1}N$ elements are negative, then the bound on the non-basic variable can increase without bound because the corresponding basic variable will only be forced to be more positive to maintain feasibility and optimality.

Lower Bound for a Non-Basic Variable

The lower bound for a non-basic variable cannot get lower than the maximum of the elements of $(B^{-1}b - X_B)/B^{-1}N$ if the $B^{-1}N$ elements are negative. The negative coefficients of negative non-basic variables will drive the corresponding basic variable to its lower bound. If the $B^{-1}N$ elements are positive, then the bound on the non-basic variable can decrease without bound because the corresponding basic variable will only be forced to be more positive to maintain feasibility and optimality.

The program listed in Appendix A.7 performs a sensitivity analysis on the bounds of basic and non-basic variables. For analysis on a basic variable bound, the program requires as input the current optimal RHS vector and the position of the basic variable of interest in the RHS vector. The analysis for a non-basic variable bound requires as input the current optimal RHS, the $B^{-1}N$ matrix of the optimal system, and the column of the $B^{-1}N$ matrix corresponding the non-basic variable of interest. The $B^{-1}N$ matrix is the constraint coefficients of the non-basic variables in the optimal system.

Conclusion

The concepts introduced in this paper are emphasized by respective computer programs. These programs are built for the purpose of furthering the students' understanding of these concepts.

Appendix A.1

Generation of all possible bases

by

2Lt Wendy Cook

```
0001: [* Generates all possible basis *]
0002: [* A matrix contains slack variables, no non-negativity constraints *]
0003:
0004: A = {{1,1,2,1,0,0},{1,3,1,0,1,0},{-1,1,1,0,0,1}};
0005: b = {9,2,4};
0006: n= 3; m= 3; totvar = n+m;
0007: Ra = Range[1,totvar];
0008: basis = Subsets[Ra,m];
0009: For[i=1,i<=Cardinality[basis],i++,
0010:    set=Extract[basis,i];
0011:    Ab = GetColumn[A,set];
0012:    soln = Inverse[Ab].b;
0013:    If[soln == {}, Print[set,"  is not an independent basis!  "] ];
0014:    Bfs = {};
0015:    For [j=1, j<= totvar, j++,
0016:      Bfs = Append[Bfs,0];
0017:    ];
0018:    For [j=1, j<=m, j++,
0019:      pos = Extract[set,j];
0020:      posval = Extract[soln,j];
0021:      Bfs = Insert[Bfs,posval,pos];
0022:   ];
0023:   Bfs = Drop[Bfs, -m];
0024:   Print["Basis ",i," =   ",set];
0025:   Print["BFS   ",i," =   ",Bfs];
0026: ];
0027:
0028:
0029:
0030:
0031:
0032:
0033:
0034:
0035:
0036:
0037:
0038:
0039:
0040:
0041:
```

MOR/ML Output

```
Basis  1  =   { 1, 2, 3}
BFS    1  =   {0.00,-1.00,5.00, 0, 0, 0}
Basis  2  =   { 1, 2, 4}
BFS    2  =   {-2.50,1.50, 0,10.00, 0, 0}
Basis  3  =   { 1, 2, 5}
BFS    3  =   {2.50,6.50, 0, 0,-20.00, 0}
Basis  4  =   { 1, 2, 6}
BFS    4  =   {12.50,-3.50, 0, 0, 0,20.00}
Basis  5  =   { 1, 3, 4}
BFS    5  =   {-1.00, 0,3.00,4.00, 0, 0}
Basis  6  =   { 1, 3, 5}
BFS    6  =   {0.33, 0,4.33, 0,-2.67, 0}
Basis  7  =   { 1, 3, 6}
BFS    7  =   {-5.00, 0,7.00, 0, 0,-8.00}
Basis  8  =   { 1, 4, 5}
BFS    8  =   {-4.00, 0, 0,13.00,6.00, 0}
Basis  9  =   { 1, 4, 6}
BFS    9  =   {2.00, 0, 0,7.00, 0,6.00}
Basis  10  =   { 1, 5, 6}
BFS    10  =   {9.00, 0, 0, 0,-7.00,13.00}
Basis  11  =   { 2, 3, 4}
BFS    11  =   { 0,-1.00,5.00,0.00, 0, 0}
Basis  12  =   { 2, 3, 5}
BFS    12  =   { 0,-1.00,5.00, 0,0.00, 0}
Basis  13  =   { 2, 3, 6}
BFS    13  =   { 0,-1.00,5.00, 0, 0,0.00}
Basis  14  =   { 2, 4, 5}
BFS    14  =   { 0,4.00, 0,5.00,-10.00, 0}
Basis  15  =   { 2, 4, 6}
BFS    15  =   { 0,0.67, 0,8.33, 0,3.33}
Basis  16  =   { 2, 5, 6}
BFS    16  =   { 0,9.00, 0, 0,-25.00,-5.00}
Basis  17  =   { 3, 4, 5}
BFS    17  =   { 0, 0,4.00,1.00,-2.00, 0}
Basis  18  =   { 3, 4, 6}
BFS    18  =   { 0, 0,2.00,5.00, 0,2.00}
Basis  19  =   { 3, 5, 6}
BFS    19  =   { 0, 0,4.50, 0,-2.50,-0.50}
Basis  20  =   { 4, 5, 6}
BFS    20  =   { 0, 0, 0,9.00,2.00,4.00}
```

Appendix A.2

Generation of all possible extreme points

by

2Lt Wendy Cook

```
001: [* Generate all possible extreme points *]
002: [* A matrix contains no slack variables but includes nonneg constraints *]
003:
004: A = {{1,1,2},{1,3,1},{-1,1,1},{1,0,0},{0,1,0},{0,0,1}};
005: At = Transpose[A];
006: b = {9,2,4,0,0,0};
007: n=3;m=6;
008: R = Range[1,m];
009: cols = Subsets[R,n];
010: For [i=1, i<= Cardinality[cols], i++,
011:     s= cols[[i]];
012:     rows = GetColumn[At,s];
013:     rowt = Transpose[rows];
014:     bt = Extract[b,s];
015:     Print["Constraints  ",s,"  Extreme Point  ",Inverse[rowt].bt]:
016: ];
```

MOR/ML Output

Constraints	{ 1, 2, 3}	Extreme Point	{0.00,-1.00,5.00}
Constraints	{ 1, 2, 4}	Extreme Point	{0.00,-1.00,5.00}
Constraints	{ 1, 2, 5}	Extreme Point	{-5.00,0.00,7.00}
Constraints	{ 1, 2, 6}	Extreme Point	{12.50,-3.50,0.00}
Constraints	{ 1, 3, 4}	Extreme Point	{0.00,-1.00,5.00}
Constraints	{ 1, 3, 5}	Extreme Point	{0.33,0.00,4.33}
Constraints	{ 1, 3, 6}	Extreme Point	{2.50,6.50,0.00}
Constraints	{ 1, 4, 5}	Extreme Point	{0.00,0.00,4.50}
Constraints	{ 1, 4, 6}	Extreme Point	{0.00,9.00,0.00}
Constraints	{ 1, 5, 6}	Extreme Point	{9.00,0.00,0.00}
Constraints	{ 2, 3, 4}	Extreme Point	{0.00,-1.00,5.00}
Constraints	{ 2, 3, 5}	Extreme Point	{-1.00,0.00,3.00}
Constraints	{ 2, 3, 6}	Extreme Point	{-2.50,1.50,0.00}
Constraints	{ 2, 4, 5}	Extreme Point	{0.00,0.00,2.00}
Constraints	{ 2, 4, 6}	Extreme Point	{0.00,0.67,0.00}
Constraints	{ 2, 5, 6}	Extreme Point	{2.00,0.00,0.00}
Constraints	{ 3, 4, 5}	Extreme Point	{0.00,0.00,4.00}
Constraints	{ 3, 4, 6}	Extreme Point	{0.00,4.00,0.00}
Constraints	{ 3, 5, 6}	Extreme Point	{-4.00,0.00,0.00}
Constraints	{ 4, 5, 6}	Extreme Point	{0.00,0.00,0.00}

Appendix A.3

Generation of an extreme point given a basis

by

2Lt Wendy Cook

```
001: [* Given a basic feasible solution, BFS, an extreme point is given *]
002: [* A matrix includes slack variables, no non-negativity constraints *]
003:
004: A = {{1,1,2,1,0,0},{1,3,1,0,1,0},{-1,1,1,0,0,1}};
005: b={9,2,4};
006: Basis = {1,2,4};
007: Ab = GetColumn[A,Basis];
008: Soln = Inverse[Ab].b;
009: If[Soln == {},Print[Basis,"  is not an independent basis!  "] ];
010: n=3;m=3;
011: totvar= n+m;
012: BFS = {};
013: For[i=1,i<= totvar, i++,
014:    BFS = Append[BFS,0];
015: ];
016: For[i=1,i<=m,i++,
017:    pos=Extract[Basis,i];
018:    posval=Extract[Soln,i];
019:    BFS=Insert[BFS,posval,pos];
020: ];
021: BFS = Drop[BFS,-m];
022: Print["BFS= ",BFS];
023: Rn = Range[1,n];
024: Extpoint = Extract[BFS,Rn];
025: Print["Extreme point = ",Extpoint];
026:
027: [* Which constraints are tight *]
028:
029: For[i=n+1, i<= totvar, i++,
030:   slackval = Extract[BFS,i];
031:   If[slackval == 0, Print["Constraint ",i-n," is tight"] ];
032: ];
033:
```

MOR/ML Output

```
BFS= {-2.50,1.50, 0,10.00, 0, 0}
Extreme point = {-2.50,1.50, 0}
Constraint  2  is tight
Constraint  3  is tight
```

Appendix A.4

Generation of a basis for a given extreme point

by

2Lt Wendy Cook

```
0001: [* A basis is generated from a given feasible extreme point *]
0002: [* The A matrix contains slack variables, no non-negativity constraints *]
0003:
0004: A = {{1,1,2,1,0,0},{1,3,1,0,1,0},{-1,1,1,0,0,1}};
0005: b = {9,2,4};
0006: n=3;  [* number of original variables --no slacks *]
0007: m=3;
0008: Rn = Range[1,n];
0009: Extpoint = {-2.5,1.5,0};
0010: For[i=1,i<=m,i++,
0011:    Extpoint = Append[Extpoint,0];
0012: ];
0013: BFS = Extract[Extpoint,Rn];
0014: For[i=1,i<= Cardinality[A],i++,
0015:   const = Extract[A,i];
0016:   Rhs = Extract[b,i];
0017:   constval = const.Extpoint;
0018:   slack = i + n;
0019:   slackval = Rhs - constval;
0020:   BFS = Append[BFS,slackval];
0021: ];
0022: Basis ={};
0023: For[i=1,i<= Cardinality[Extpoint],i++,
0024:    If[Extract[BFS,i]!=0, Basis = Append[Basis,i] ];
0025: ];
0026: Print["BFS =  ",BFS];
0027: Print["Basis = ",Basis];
0028:
0029:
```

MOR/ML Output

```
RFS =  (-2.50,1.50, 0,10.00,0.00,0.00)
asis = { 1, 2, 4}
```

Appendix A.5

Simplex Algorithm in matrix form

by

Dr. Bryan L. Deuermeyer

```
001: [*-------------------------------------------------------+
002:    The Simplex Algorithm in Matrix Form
003: +-------------------------------------------------------*]
004:
005: simplexAlgorithm[c,a,b] := Block[
006: {m,n,Binv,cB,basis,i,z,x,ok,optimal,unbounded},
007: [* Assume each row has a slack variable *]
008: ok = 0;
009: optimal = 1;
010: unbounded = 2;
011:
012: setupLP[] := Block[{i},
013: m = Cardinality[b];
014: n = Cardinality[c];
015: Binv = IdentityMatrix[m];
016: basis = n + Range[m];
017: a = JoinColumns[ a,Binv];
018: c = Join[c,Table[0,{i,m}] ];
019: cB = Extract[c,basis];
020: ];
021:
022: getLeavingVariable[aa,bb]:= Block[{r,p},
023: pdiv[{den,num}] := If[den>0.0,num/den,Infinity];
024: {r,p} = MinList[ Map[pdiv,Transpose[{aa,bb}],1] ];
025: If[r<0.0, Return[0], Return[p] ];
026: ];
027:
028: doPivot[pr,pc,abar] := Block[{p,alfa,t},
029: p = 1.0/abar[[pr]];
030: alfa = -p abar;
031: alfa[[pr]] = p;
032: t = IdentityMatrix[m];
033: t[[pr]] = alfa;
034: Binv = Transpose[t].Binv; [* Binv = E.Binv *]
035: basis[[pr]] = pc;
036: cB[[pr]] = c[[pc]];
037: ];
038:
039: getEnteringVariable[cbar] := Block[{maxZ,pc},
040: {maxZ,pc} = MaxList[cbar];
041: If[ maxZ <= 0.0, Return[0], Return[pc] ];
042: ];
043:
044: simplexStep[w]:= Block[{pc,pr,abar},
045: If[w!=ok, Break[w] ];
046: pc = getEnteringVariable[cB.Binv.a - c];
047: If[ pc == 0, Break[optimal] ];
048: pr = getLeavingVariable[ abar=Binv.GetColumn[a,pc], Binv.b];
049: If[ pr == 0, Break[unbounded] ];
050: doPivot[pr,pc,abar];
051: Return[ok];
052: ];
```

```
0053:
0054: setupLP[];
0055: If[ Nest[simplexStep,ok,100] == 2,
0056: Return["Unbounded"]
0057: ];
0058: bbar = Binv.b;
0059: z = cB.bbar;
0060: x = Table[0.0,{i,n+m}];
0061: For[i=1,i<=m,i++,
0062: x[[ basis[[i]] ]] = bbar[[i]]
0063: ];
0064: Return[{x,z}]
0065: ];
0066:
0067: [*-----------------------------------------------------+
0068:    Now Solve a Problem
0069: +-----------------------------------------------------*]
0070: a= {{1,1,2},{1,3,1},{-1,1,1}};
0071: c= {1,1,-4};
0072: b= {9,2,4};
0073:
0074: simplexAlgorithm[c,a,b]:
0075:
0076:
0077:
0078:
0079:
```

MOR/ML Output

0074: {{0.00,0.00,2.00,5.00,0.00,2.00},-8.00}

Appendix A.6

Altered simplex algorithm in matrix form

by

Dr. Bryan L. Deuermeyer

altered by 2Lt Wendy Cook

The function calls getEnteringVariable and getLeavingVariable are changed in order to allow the students to select the variables.

```
001: [*-------------------------------------------------------+
002:    The Simplex Algorithm in Matrix Form
003: +------------------------ ------------------------------*]
004:
005: simplexAlgorithm[c,a,b] := Block[
006: {m,n,Binv,cB,basis,i,z,x,ok,optimal,unbounded,pr,pc},
007: [* Assume each row has a slack variable *]
008: ok = 0;
009: optimal = 1;
010: unbounded = 2;
011:
012:
013: setupLP[] := Block[{i},
014: m = Cardinality[b];
015: n = Cardinality[c];
016: Binv = IdentityMatrix[m];
017: basis = n + Range[m];
018: a = JoinColumns[ a,Binv];
019: c = Join[c,Table[0,{i,m}] ];
020: cB = Extract[c,basis];
021: ];
022:
023: [* getLeavingVariable[aa,bb]:= Block[{r,p},
024: pdiv[{den,num}] := If[den>0.0,num/den,Infinity];
025: {r,p} = MinList[ Map[pdiv,Transpose[{aa,bb}],1] ];
026: If[r<0.0, Return[0], Return[p] ];
027: ]; *]
028:
029: doPivot[pr,pc,abar] := Block[{p,alfa,t},
030: p = 1.0/abar[[pr]];
031: alfa = -p abar;
032: alfa[[pr]] = p;
033: t = IdentityMatrix[m];
034: t[[pr]] = alfa;
035: Binv = Transpose[t].Binv; [* Binv = E.Binv *]
036: basis[[pr]] = pc;
037: cB[[pr]] = c[[pc]];
038: ];
039:
040: [* getEnteringVariable[cbar] := Block[{maxZ,pc},
041: {maxZ,pc} = MaxList[cbar];
042: If[ maxZ <= 0.0, Return[0], Return[pc] ];
043: ]; *]
044:
045: simplexStep[]:= Block[{pc,pr,abar},
046: [* If[w!=ok, Break[w] ];*]
047: pc = 1; [* getEnteringVariable[cB.Binv.a - c];*]
048: [* If[ pc == 0, Break[optimal] ];*]
049: pr =  1;  [* getLeavingVariable[ abar=Binv.GetColumn[a,pc], Binv.b];*]
050: abar = Binv.GetColumn[a,pc];
051: [*If[ pr == 0, Break[unbounded] ];*]
052: doPivot[pr,pc,abar];
```

```
053: [*Return[ok]; *]
054: ];
055:
056: setupLP[];
057: simplexStep[];
058: [*If[ Nest[simplexStep,ok,100] == 2,*]
059: [*Return["Unbounded"]*]
060: [*];*]
061: bbar = Binv.b;
062: z = cB.bbar;
063: x = Table[0.0,{i,n+m}];
064: For[i=1,i<=m,i++,
065: x[[ basis[[i]] ]] = bbar[[i]]
066: ];
067: Return[{x,z}]
068: ];
069:
070: [*-------------------------------------------------------+
071:    Now Solve a Problem
072: +-------------------------------------------------------*]
073: a = {{1,1,2},{1,3,1},{-1,1,1}};
074: c = {1,1,-4};
075: b = {9,2,4};
076: simplexAlgorithm[c,a,b]:
077:
078:
079:
080:
081:
082:
083:
084:
085:
086:
087:
088:
089:
090:
091:
092:
093:
094:
095:
096:
097:
098:
099:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
```

MOR/ML Output

0076: {{9.00,0.00,0.00,0.00,-7.00,13.00},9.00}

Appendix A.7

Sensitivity Analysis

by

2Lt Wendy Cook

```
                              MOR/ML System 1.42      12/11/92

   File: b:\test.doc    Date: Fri Jun 11 08:07:49 1993

001:
002: [* Right-hand side *]
003:
004: LowBound[Binv,bbar,d]:= Block[{n,p},
005:   dbar = Binv.d;
006:   rdiv[{num,den}]:= If[den>0.0,  - num/den, -Infinity];
007:   {n,p} = MaxList[ Map[rdiv,Transpose[{bbar,dbar}],1] ];
008:   Return[n];
009: ];
010:
011: UpperBound[Binv,bbar,d]:= Block[{n,p},
012:   dbar = Binv.d;
013:   rdiv[{num,den}]:= If[den<0.0, -num/den, Infinity];
014:   {n,p} = MinList[ Map[rdiv,Transpose[{bbar,dbar}],1] ];
015:   Return[n];
016: ];
017:
018: Binv = {{0.5, 0.2, -1}, {-1, 1, 0.5}, {5, -0.3, 2}};
019: bbar = {2,3,1};
020: d = {1,0,0};
021: "This element of the RHS can be decreased by ":LowBound[Binv,bbar,d];
022: "This element of the RHS can be increased by ":UpperBound[Binv,bbar,d];
023:
024: [* Cost Coefficients *]
025:
026: NonBasCoeff[RCosts,RedCostpos]:= Block[{value},
027:   value = Extract[RCosts,RedCostpos];
028:   Return[value];
029: ];
030:
031: LowBasCoeff[RCosts,BinvNRow]:= Block[{r,p},
032:   Low[{frow,srow}]:= If[srow<0.0, frow/srow, -Infinity];
033:   {r,p} = MaxList[ Map[Low,Transpose[{RCosts,BinvNRow}],1] ];
034:   Return[r];
035: ];
036:
037: HighBasCoeff[RCosts,BinvNRow]:= Block[{r,p},
038:   High[{frow,srow}]:= If[srow>0.0, frow/srow, Infinity];
039:   {r,p} = MinList[ Map[High,Transpose[{RCosts,BinvNRow}],1] ];
040:   Return[r];
041: ];
042:
043: RCosts = {1, 2, 3};
044: BinvNRow = {-1, 1, 2};
045: RedCostpos = 1;
046: "This basic coefficient can be decreased by ":
047: LowBasCoeff[RCosts,BinvNRow];
048: "This basic coefficient can be increased by ":
049: HighBasCoeff[RCosts,BinvNRow];
050: "This nonbasic coefficient can be decreased by ":
051: NonBasCoeff[RCosts,RedCostpos];
052:
```

```
053: [* Bounds *]
054:
055: BasVar[bbar,BasicRow]:= Block[{Bound},
056:   Bound = Extract[bbar,BasicRow];
057:   Return[Bound];
058: ];
059:
060: UpNonBasVar[bbar,BinvN,VarCol]:= Block[{r,p,ncol},
061:   ncol = GetColumn[BinvN,VarCol];
062:   Upper[{num,den}]:= If[den>0.0, num/den, Infinity];
063:   {r,p} = MinList[ Map[Upper,Transpose[{bbar,ncol}],1] ];
064:   Return[r];
065: ];
066:
067: LoNonBasVar[bbar,BinvN,VarCol]:= Block[{r,p,ncol},
068:   ncol = GetColumn[BinvN,VarCol];
069:   Lower[{num,den}]:= If[den<0.0, num/den, -Infinity];
070:   {r,p} = MaxList[ Map[Lower,Transpose[{bbar,ncol}],1] ];
071:   Return[r];
072: ];
073:
074: bbar = {10, 10};
075: BasicRow = 2;
076: BinvN = {{-7, -5, -0.667}, {0.8, 0.6, -0.0667}};
077: VarCol = 2;
078: "The lower bound on this basic variable can be increased by ":
079: BasVar[bbar,BasicRow]:
080: "The lower bound on this nonbasic variable can be increased by ":
081: UpNonBasVar[bbar,BinvN,VarCol]:
082: "The lower bound on this nonbasic variable can be decreased by ":
083: LoNonBasVar[bbar,BinvN,VarCol]:
084:
```

MOR/ML Output

```
021:  This element of the RHS can be decreased by
 21: -0.20
022:  This element of the RHS can be increased by
022:  3
 46:  This basic coefficient can be decreased by
 47:  -1
048:  This basic coefficient can be increased by
049: 1.50
 50:  This nonbasic coefficient can be decreased by
051:  1
078:  The lower bound on this basic variable can be increased by
 79:  10
 80:  The lower bound on this nonbasic variable can be increased by
081: 16.67
 82:  The lower bound on this nonbasic variable can be decreased by
 83: -2.00
```

Bibliography

Anton, Howard. Elementary Linear Algebra. 5th ed. New York: John Wiley & Sons, 1973.

Bazaraa, Mokhtar S., John J. Jarvis, and Hanif D. Sherali. Linear Programming and Network Flows. 2nd ed. New York: John Wiley & Sons, 1977.

Hill, Richard O. Jr. Elementary Linear Algebra with Applications. 2nd ed. San Diego: Harcourt Brace Jovanovich, 1991.

Hillier, Frederick S. and Gerald J. Lieberman. Introduction to Operations Research. 5th ed. New York: McGraw-Hill, 1990.

Micro OR/Math Language (MOR/ML). Computer Software. B.L. Deuermeyer, G.L. Curry, and R.M. Feldman, 1991.

Curry, Guy L., Notes for Linear Programming.