National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

Data Perturbation Analyses for Linear Programming

by

Constantinos Karamalis

Masters of Science thesis

Systems Science programme

University of Ottawa

December 1993

ISBN 0-612-00475-9

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

This thesis focuses on several aspects of data perturbation for Linear Programming. Classical questions of degeneracy and post-optimal analysis are given a unified presentation, in view of new interior point methods of linear programming. The performance of these methods is compared to the simplex algorithm; interior point methods are shown to alleviate some difficulties of representation and solution of linear programs. An affine scaling algorithm is implemented in conjunction with a simple rounding heuristic to assess the benefit of interior point trajectories to provide approximate solutions of linear integer programming.

Acknowledgements

First and most of all I wish to thank my thesis supervisor, Professor Jean-Michel Thizy at the University of Ottawa for all his time, and keen interest. His counsel and support was greatly appreciated. I also wish to thank Professor Dan Lane at the University of Ottawa for his encouragement. Thanks go to Paul Monat, Network Manager at the Faculty of Administration, for his help in drawing Figures 1 and 3 of the thesis and to Sylvie Pellerin of the Faculty of Administration, for her insight on so many word processing questions.

I am thankful to my family whose support was essential to finishing my studies, especially the financial one. I also wish to thank my friends who gave me the strength to pull it through, and especially Garrett and Theodore - hope you get your masters soon too - and everyone who believed in me.

This thesis was partially funded by NSERC grant #OGP 0042197.

List of Figures

List of Tables

Table of Contents

Chapter 1 - Introduction

"What happens to the solution of a program if the data varies slightly?", the question at the heart of sensitivity or data perturbation analysis, is a central theme of many numerical methods. Even when restricted to mathematical programming, this general concern has given rise to literally thousands of studies. A very closely connected topic is the influence of computing imprecision on the accuracy of the solution [Fiacco, 1982, 1983]. This thesis focuses on several special facets of the question devoted to linear programming [Deif, 1986]:

- degeneracy: how does a linear programming solver diagnose that data specifications are redundant and treat the effect of changes in these pieces of data?
- what are the possible input data that correspond to a given solution?
- are some linear programming methods less sensitive to degeneracy than others?

The same questions arise for discrete variables, e.g. in mixed integer linear programming: can a new linear programming method revive a conventional heuristic yielding approximate solutions?

1.1 Scope of the thesis

Data perturbation is a standard tool to prevent cycling in linear programming [Charnes, 1952], implemented from the inception of the simplex method [Dantzig, 1963]. Best known for creating such numerical difficulties, degeneracy has also been actively investigated for the computational complexity that it exerts [Chandrasekaran, 1982], and for its direct impact on post-optimal analysis [Gal, 1988]. As good examples of this ongoing attention, two compendia of analyses of degeneracy in optimization will be published in January 1994, which were not available at the time of writing [Gal, 1994].

The simplex algorithm:

The simplex algorithm [Dantzig, 1963] has long been the method of choice to solve linear programs. To find an optimum solution, the simplex algorithm enumerates bases until it finds an optimal one. The existence of optimal bases simplifies the data representation of optimal solutions, and this has contributed to its wide adoption by decision-makers. If the program has m constraints, at most m optimal basic variables are strictly positive, typically much fewer than the n variables present in the problem. To a lesser extent, the simplex algorithm provides also simple dual information, as it uses the complementary slackness property. Most classical post-optimality analyses, focusing on when data perturbations leave the optimum unchanged, hinge on the properties of optimal bases.

Post-optimal analysis:

While early progress was made in improving the speed of the simplex method, additional capabilities such as post-optimal analysis were provided gradually. These additions have become an intrinsic part of the linear programming folklore and are expected by decision-makers (whether or not they are actually used). Thus, every commercial linear programming package must provide post-optimality analysis in order to be considered acceptable, which makes the design of efficient post-optimal analysis an active field of research and development. Much attention has been focused on the interpretation of the results as marginal analysis of resource allocation [Greenberg, 1985, 1986].

Post-optimal analysis has often been presented in an economic context, but many researchers pointed out some shortcomings in its economic interpretation, generally due to the presence of degeneracy [Gal, 1994]. For instance, a dual variable may not be interpreted as a marginal price, but only as a value comprised between a marginal rate of increase and a marginal rate of decrease. Amendments to the classical post-optimal analysis and its presentation have often been proposed, such as interactive requests that the user provide a direction of change from the optimum.

New algorithms for linear programming:

Alternative methods are becoming competitive to solve very large linear programs. The best-known among the new methods are:

- the ellipsoid method [Khachiyan, 1979] of theoretical interest, as it first proved that linear programming could be solved in polynomial time,
- the projective method [Karmarkar, 1984] offering a better theoretical measure of complexity. The projective method has generated widespread interest because it has provided the base for fast solution of very large linear programs on standard uniprocessors, and it is amenable to parallelization; and
- overrelaxation [Mangasarian, 1981], proximal [Rockafellar, 1976, Wright, 1989], interior point methods [Adler, 1994] adapted to new computing environments such as multi-processors, massively parallel architectures and neural networks.

These methods differ from the simplex method that computes vertices of the feasible region. In contrast, the new methods can produce a sequence of points interior to the feasible region. Karmarkar's polynomial-time, interior point algorithm and its ensuing affine scaling implementations [Barnes, 1986, Vanderbei, 1986] have revived interest in many applications of linear programming beyond the reach of the simplex method.

Post-optimality analysis with interior point methods:

Recently developed methods need not use a basis to solve a linear program. Yet, their most common approach to post-optimality analysis has been conservative, following the steps taken by the simplex method, i.e. retrieving a basis in order to report the requested information under the classical format. Basis retrieval may not be easy: although some polynomial algorithms have been devised to extract a basis from an optimal point [Megiddo, 1991], the procedure may be slower than what users may require for sensitivity analysis.

A second, much less investigated approach, is to extend the concept of post-optimality to dispense with bases. This is one of the objectives of the thesis. Additionally, methods bypassing bases may be more robust in the presence of degeneracy.

Data perturbation for mixed integer programming

Data perturbation analysis has yielded fewer results when some decision variables range over discrete domains. Most analyses of mixed integer programming provide approximation schemes with no guarantee of accuracy, or show that the post-optimality problems are NP-complete, i.e., some problem instances are inordinately difficult to solve. As with interior point methods, bases do not provide a sufficient tool of analysis. In this view, the last chapter of this thesis attempts to use interior point methods to solve mixed integer programs.

1.2 Methodology

The thesis first reviews interior point methods such as the projective and the affine-scaling method. Chapter 3 clarifies some lesser known properties of degeneracy in linear programming [Gal, 1986, 1988, 1990], and their impact on the representation of the optimal solution to the decision-maker [Adler, 1989, Jansen, 1993, Mehrotra, 1992]. The computational aspects of these properties are reviewed within the simplex algorithm and the affine-scaling method, both applied to post-optimal analysis.

Virtually all descriptions of the simplex method assume that the constraint matrix has full row rank. In a primal simplex algorithm, rank deficiency can be discovered as a by-product of a Phase I to reach feasibility. Many commercial codes prefer to preprocess the original formulation in the hope of finding redundant constraints. In either case, constraints are dropped and the simplex method will solve a new formulation. Post-optimality is performed on this condensed formulation, while the decision-maker is interested in sensitivity analysis of the original formulation. The thesis aims at extending the definition of degeneracy to general formulations

of linear programs, thereby avoiding to convert post-optimal results in terms of the original formulation.

Primal and dual degeneracy influence post-optimal analyses differently. The relationship and difference between the two phenomena is reviewed. A primal degenerate extreme solution is characterized by the existence of either multiple bases defining it (the well-known case), or by the presence of null variables. Subsequently, difficulties of post-optimal analyses are circumscribed to the subcase of degeneracy created by multiple dual values.

As interior point methods yield iterates in the relative interior of the constraints, a further extension of degeneracy involves non-extreme points. Degeneracy may no longer be defined in terms of bases spanning the right hand-side of the formulation, but rather in terms of the rank of an associated matrix.

With the simplex method, range and parametric analyses find the range of values of the parameters within which the optimal basis remains unchanged. Since interior point methods do not provide bases, one can either maintain the definition and retrieve bases from optimal points, which is theoretically possible, or adapt the scope of post-optimality analysis. The thesis explores whether the following definition is well-founded and operational: range analysis of the coefficients of the right hand-side indicates when the dual variables remain optimal, and range-analysis of objective coefficients reports when the primal variables remain optimal. It also reviews the effect of degeneracy on this alternative definition.

Symbolic systems for linear programming represent solutions by a system of defining constraints. Chapter 5 shows how a uniform representation by constraints obviates many problems of post-optimal analysis.

In Chapter 6, data perturbation analysis is applied to mixed integer programming. Building on the well-known rounding of linear programming solutions, a simple heuristic generates some integer vectors with a random function centered around an incumbent interior solution as it heads

to the optimum. While the experimentation with an interior point method for integer programming is inconclusive, it provides a simple alternative to classical, simplex-based schemes that proceed from a vertex inward.

1.3 Contributions of the thesis

Given the voluminous research on data perturbation analysis for linear programming, many of the results presented in this thesis are probably prefigured by other studies. A main contribution of the thesis is to unify them, and propose a general setting for degeneracy and post-optimal analysis in linear programming. Central or original results are generally entitled in boldface. Specific contributions are summarized below.

Dissociating multiple bases and dual values from degeneracy as three distinct notions (albeit closely related), is a simple but important concept. Disentangling the role of null and unrestricted variables from their non-negative counterparts also sheds light on representation of linear programs and the solution performance of the simplex algorithm as well as interior point methods.

The simplified definition of post-optimal analysis, without reference to bases, is quite attractive for decision-making where bases are not taken in consideration. The affine scaling method is adapted to provide a quick approximation of range analysis. This approach replicates closely the range analysis performed by the simplex method, in which a quick approximation constitutes also the first iteration of a complete optimization for full range analysis.

The thesis illustrates abundantly that degeneracy is essentially a problem of representation of linear programs. A symbolic system based solely on constraint representation is unconventional and somewhat more cumbersome than numerical solvers, but avoids the problems of representation at the focus of this thesis.

While the experimentation with an interior point method for integer programming is limited, it provides a simple alternative to heuristic schemes that proceed from a vertex inward and highlight the potential of interior point methods.

Chapter 2 - Interior Point Methods for Linear Programming

This chapter presents a new method of solving linear programs, the Interior Point method. A presentation of its basic elements is necessary for a full understanding of the applications described in the next chapters. The number of computational steps taken by the simplex algorithm to solve linear programs can increase exponentially with the size of the problem encoding. There have been many attempts to devise linear programming algorithms that would guarantee worst-case polynomial time performance. Khachiyan [1979] devised the first linear programming method to run in polynomial time, the *elliipsoid method* which answered a long-standing theoretical question, but has not yielded any practical algorithms because the matrices that it generates get increasingly dense.

2.1 Interior point methods:

In 1984, N. K. Karmarkar proposed a new polynomial time algorithm for linear programming which, unlike the simplex algorithm, does not enumerate extreme points of the feasible region defined by the linear constraints. The promise of a practical implementation to solve large problems much faster than the simplex method, and to solve problems hitherto out of reach of the best commercial codes, spurred intense research on interior point methods. These methods are divided into two classes: the projective methods (e.g. Karmarkar's algorithm) which force the sequence of generated points to home in on the central trajectory defined by the constraints, and the affine-scaling methods which allow the points to depart further from the central trajectory. The latter have intuitive geometric descriptions, but the actual proof of their convergence is a difficult problem. Large problems are solved by the affine-scaling method because in practice it proves to be better than the projective algorithms. In fact, the two methods are not so different as first thought. Monteiro et al. [1988] showed that the affine-scaling method is actually polynomial if the initial point is close to the centre of the polytope and the step size is small enough. Otherwise, these algorithms are exponential in the worst case. Both approaches have

yielded primal and dual algorithms. Originally, projective and affine-scaling methods presupposed that the problem was not degenerate; a notable exception was an early, but relatively unknown precursor [Dikin, 1967]. The affine-scaling method was later adapted to degenerate problems [Tsuchiya, 1991, 1992].

2.2 Projective Interior Point Methods

Karmarkar [1984] and many subsequent analyses of Interior Point methods consider a particular formulation.

2.2.1 Canonical form:

minimize cx

$Ax = 0$

$1x = 1$

$x \geq 0$,

in which:

- **1** is used to refer to the row vector (1, 1, ... 1, 1) and is assumed here to have the dimension n of x (each coordinate being indexed by j),
- $x^0 = 1/n$ is a feasible point and
- the optimum objective value is $cx^* = 0$.

Several transformations can be used to reduce an LP formulation to its canonical form. The best known is the original transformation of Karmarkar [1984]. First, reformulate the problem as a primal-dual pair, where the objective function expresses the duality gap, vanishing at its minimum. Then, remove all unrestricted variables by substitution and use slack and surplus variables to transform the inequalities to equalities, thus obtaining a problem in standard form:

minimize c'x

subject to

A'x = b,

$x \geq 0$.

The initialization and the main steps of the algorithm to solve the canonical form are described next.

2.2.2 Initialization

First, construct one additional column a = (n+1)b-A'$\mathbf{1}^T$ for an artificial variable x', following a classical initialization scheme [Beale, 1954, Wolfe, 1963]. Denote a large scalar as M and define the new system:

minimize c'x + M x'

subject to

A'x + a x' = b,

$x \geq 0$, $x' \geq 0$.

The previous formulation may be relabeled min c"$\boldsymbol{X}$ s.t. A"$\boldsymbol{X}$ = b, $\boldsymbol{X} \geq 0$, in canonical form, since an interior point $\boldsymbol{X} = \mathbf{1}^T/(n+1)$ has been defined and the optimum is zero. Alternatively, in keeping with the simplex method, x' may be replaced by a vector of artificial variables, which avoids the dense column a.

Next, repeated *projective transformations* must be applied to the feasible set of this problem, which replace the non-negativity constraints by a sphere. In the transformed space, a simple step improves the objective value. A polynomial number of such transformations converges to an optimum solution of the problem. Each iteration is next reviewed in detail. For a given point x, define the following diagonal matrix:

$$X = \begin{bmatrix} x_1 & 0 & 0 & \dots & 0 & \dots & 0 \\ 0 & x_2 & 0 & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & x_m & 0 & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \dots & x_n \end{bmatrix}$$

With this definition, $AX = \begin{bmatrix} a_{11}x_1 & a_{12}x_2 & \dots & a_{1n}x_n \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ a_{m1}x_1 & a_{m2}x_2 & \dots & a_{mn}x_n \end{bmatrix}$.

The projective algorithm creates a sequence of feasible points x^0, x^1, ..., x^k. The diagonal matrix corresponding to x^k will be denoted X_k. Let $B_k = \begin{bmatrix} AX_k \\ 1 \end{bmatrix}$. The index k will usually be omitted.

2.2.3 The main algorithm

To compute x^{k+1}, follow the next procedure:

- First calculate the search direction $\bar{c} = [\, I - B^T(BB^T)^{-1}B \,]\, X_k\, c^T$.

- Find the iterate $x' = \frac{1^T}{n+1} - \alpha\, r\bar{c}/|\bar{c}|$, where the radius of the inscribed sphere is $r = \frac{1}{\sqrt{n(n-1)}}$ and the step size α must be selected in the interval (0,1), as is Karmarkar's proposed value 1/4.
- Finally, the inverse projective transformation of x' yields the iterate in the original space $x^{k+1} = X_k\, x'/1\, X_k\, x'$.

Variants of Karmarkar's algorithm have been implemented by several commercial LP solvers. The affine scaling method, a simplification of Karmarkar's original method, avoids projective transformations but, for step sizes of practical magnitude, forsakes any guarantee of polynomial convergence. Most of the ongoing algorithmic design based on Interior Point methods uses the affine scaling method that performs quite well in practice.

2.3 Affine Scaling Method

Affine scaling versions of Karmarkar's algorithm were proposed by Dikin [1967], and later Barnes [1986] and Vanderbei et al. [1986]. These methods can be viewed as a projected gradient algorithm. An iteration consists of the following steps:

- compute the search direction $\bar{c} = [\, I - B^T(BB^T)^{-1}B\,]\, X_k\, c^T$, where $B = AX_k$,
- determine the maximum feasible step size $\rho = \max_j \bar{c}_j$,
- compute the new iterate $x^{k+1} = x^k - \alpha\rho^{-1}X_k\, \bar{c}$. The scalar α chosen in (0,1) ensures that all coordinates of x^{k+1} remain greater than zero by at least some prespecified small amount.

Stopping Rule

The properties of complementary slackness and dual feasibility are used to determine optimality. Define $r(x)= [\ I- A^T(BB^T)^{-1}BX]\ c^T$ as the reduced price. Set $\delta(x) = -\min_j\{r_j(x)\}$ to measure dual feasibility and $\gamma(x) = \max_j\{r_j(x)\ x_j\}$ to measure the degree of complementary slackness. Vanderbei et al. [1986] show that if the algorithm terminates when $\gamma(x^k) + \delta(x^k)\ M \leq \varepsilon / n$, where M is an upper bound on the objective function value, then x^k is an ε-optimal feasible solution of the problem. Their implementation uses an approximation M of the upper bound; thus the stopping criterion is only approximate. This implementation provides the basis for a heuristic for integer programming described in Chapter 5.

An example is given for a problem, variants of which will be used throughout the thesis:

maximize $x_1 + x_2$

s.t.

$x_1 + x_2 \leq 2$

$x_1,\ x_2 \geq 0$

A graphic representation is given in Figure 1, in which the arrow indicates the gradient of the objective function. To cast the problem in the form shown in Section 2.2.2, add the slack variable x_3, and get:

minimize $-x_1 - x_2$

s.t.

$x_1 + x_2 + x_3 = 1$

$x_1,\ x_2,\ x_3, \geq 0$

An artificial variable x_4 is given a large coefficient. The canonical representation is therefore:

minimize $-x_1 - x_2 + M x_4$

s.t.

$$x_1 + x_2 + x_3 = 1$$

$$x_1,\ x_2,\ x_3,\ x_4 \geq 0$$

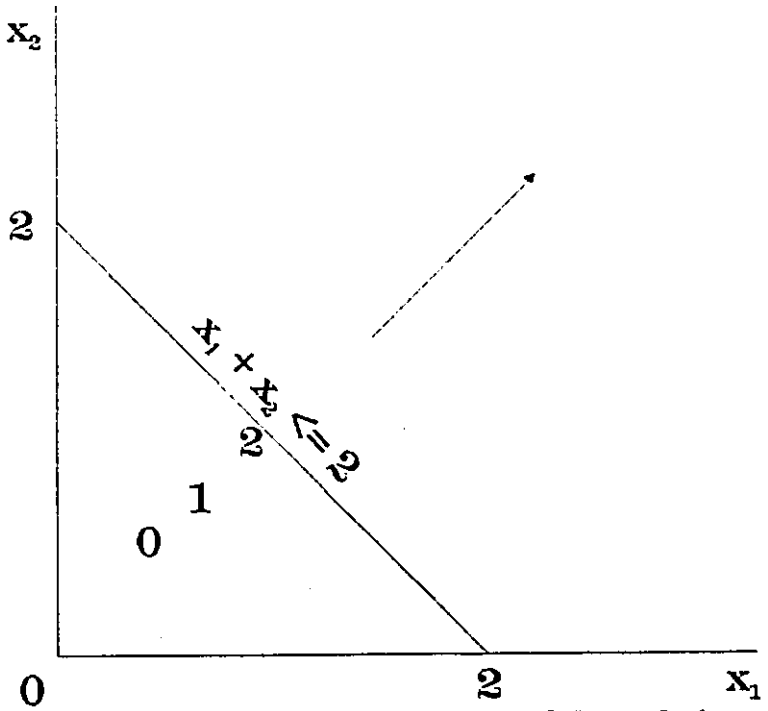

Figure 1 - Feasible region of Example 1

Each stage of the calculation is displayed in Appendix A. For efficiency, the algorithm implemented in Chapter 6 does not calculate an explicit inverse of BB^T, but finds a solution u of the system: $(BB^T)^{-1} u = BXc^T$. For this illustration, the initial point is chosen as $x^o = (1/2, 1/2, 1/2, 1/2)$. The iterate points are represented in Figure 1 by the numbers 0, 1 and 2 (the optimum). Note that the optimum is not an extreme point, an observation exploited in Chapters 3 and 4. Affine scaling methods were later applied to primal-dual formulations of linear programs with great success [Kojima, 1987, Monteiro, 1988, McShane, 1988], as described in the following section.

2.4 Primal-Dual Method:

Megiddo and Shub [1986] combine the primal and dual problems, using logarithmic barrier functions to solve the resulting set of constraints. Kojima et al. [Kojima, 1987] embed these properties in a primal-dual interior point algorithm. An implementation by McShane et al. [1988] is described next. Consider the primal problem:

minimize c x

$A\ x = b$

$x \geq 0$

Its dual is:

minimize y b

$y\ A + z = c$,

$z \geq 0$

Enforcing the non-negativity requirement with a logarithmic barrier function yields the program:

maximize $f(x, \mu) = cx - \mu \sum_{i=1}^{n} \ln x_i$

$A\ x = b$

$x \geq 0$

For a given point z, define the following diagonal matrix:

$$Z = \begin{bmatrix} z_1 & 0 & 0 & \dots & 0 & \dots & 0 \\ 0 & z_2 & 0 & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & z_m & 0 & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \dots & z_n \end{bmatrix}$$

The optimum of the preceding program for a given value of μ can be found by solving the primal-dual system:

$1XZ = \mu 1$

$A\ x = b$

$y\ A + z = c$

The first two equations ensure primal feasibility; the first and third ensure dual feasibility. As $\mu \rightarrow 0$, the local optimum converges to the optimum of the original problem. This system of equations can be solved by Newton's method. Corrections of the current estimate x, y and z, are calculated as Δx, Δy, and Δz, respectively, by the following formulae:

$$\Delta x(\mu) = [Z^{-1} - Z^{-1} X A^T (A X^{-1} X A^T)^{-1} A Z^{-1}] v(\mu)$$

$$\Delta y^T(\mu) = - (A Z^{-1} X A^T)^{-1} A Z^{-1} v(\mu)$$

$$\Delta z^T(\mu) = A^T (A Z^{-1} X A^T)^{-1} A Z^{-1} v(\mu)$$

or:

$$\Delta y^T(\mu) = - (A Z^{-1} X A^T)^{-1} A Z^{-1} v(\mu)$$

$$\Delta z(\mu) = \Delta y(\mu) A$$

$$\Delta x(\mu) = Z^{-1} v(\mu) - Z^{-1} X \Delta z^T(\mu)$$

To update x, y, z, subtract some multiple α of the values Δ, using a scalar α such that x and z remain positive. To make μ converge towards zero, it is possible to use $\mu^{n+1} = (by - cx)/n^2$, a scaled duality gap. As the solution approaches optimality, the primal cost cx approaches b·y, so $\mu \rightarrow 0$.

Chapter 3 - Degeneracy

In this chapter, properties of degenerate systems of constraints are reviewed and extended, to be applied to completely general formulations of linear programs, which will set foundations for post-optimal analysis. We will consider the systems of constraints $Ax \rho b$ (where A has n columns, and ρ denotes $\geq$, = or $\leq$) and assume neither that all variables are non-negative, nor that the rows of coefficients of tight constraints are linearly independent. Assume without loss of generality that the variables have been sorted so that $x = [x^+, x^\pm]$, where $x^+ \geq 0$. Let A^+ (resp. $A^\pm$) include the columns of A corresponding to x^+ (resp $x^\pm$). Correspondingly, excerpt from X the blocks X^+ and $X^\pm$.

3.1 Definition of Degeneracy

The largest part of the literature on degeneracy of linear constraints has been spurred by the numerical difficulties encountered at a degenerate iteration of the simplex method. Although degeneracy is usually presented in this framework, our approach will use a more general definition while retaining its pointwise characterization.

Definition 1 **Degenerate solution** [Gill, 1981]

A solution of a system $Ax \rho b$ is degenerate if and only if the rows of coefficients of tight constraints are linearly dependent.

The tight constraints are often called defining hyperplanes, and the system is said not to have full row rank, which will also be abbreviated as "linearly dependent system of constraints". By abuse of language, solutions of degenerate systems of constraints are called degenerate solutions or points, a convention so widespread that it is adopted in the chapter, with proper qualification.

Strictly speaking, a degenerate solution need not be feasible; the issue of feasibility is ignored until Section 5. Definition 2 will later be expanded to include degenerate faces.

Definition 2 **Degenerate problem**

A system Ax ρ b is degenerate if and only if a solution is degenerate.

We will sometimes call the system primal degenerate. Degeneracy, tied to an *algebraic* representation Ax ρ b of a polyhedron, remains invariant through simple algebraic operations such as some projections or related transformations, e.g. forming the standard formulation $Ax = b$, $x^{+} \geq 0$ (by adding a non-negative slack variable and replacing one inequality by its corresponding equality). Its geometric properties, such as invariance through projection, are limited; for example, degeneracy arises by duplicating a constraint of an otherwise non-degenerate system of inequalities. Yet, some geometric elements other than points can conveniently be called degenerate, as shown below.

Definition 3 Face

A constraint $az \leq b$ *defines a s-dimensional face of a t-dimensional polytope P if it is satisfied by all* $z \varepsilon P$ *and satisfied with equality by s affinely independent vectors* $z \varepsilon P$, $1 \leq s \leq t$. *Strictly speaking, the face is the set* $P \bigcap \{z \mid az = b\}$.

Definition 4 Degenerate face

The face of the polyhedron represented by the system Ax ρ b is called degenerate if and only if it is defined by a subsystem of linearly dependent hyperplanes.

Note that if a subsystem of tight constraints is linearly dependent, so is the system of tight constraints. For a geometric interpretation, consider subfaces of a face, i.e. subsets that form faces of lower dimension.

Property 1

If a face is degenerate, then all of its subfaces are degenerate.

Proof: All the linearly dependent hyperplanes of the face are part of the system of tight constraints that defines a subface. Thus, its system does not have full row rank. ■

A *subface* can be empty (or 0-dimensional) and actually need not be a feasible set. An extreme point is a one-dimensional face of a polyhedron. By Definition 2, if the system has a degenerate solution, then the problem is degenerate. Whereas solutions are generally associated to points, they can be directly associated to a face. In this sense, the following Definition 5 generalizes Definition 2 because it defines a degenerate subsystem of any rank in terms of a corresponding degenerate system. In particular, problem degeneracy can be defined by a system of constraints of rank less than n, i.e. by a degenerate face instead of a degenerate point.

Definition 5

A system is degenerate if and only if it is solved by every solution of a linearly dependent subsystem Ax ρ *b of tight constraints.*

As a consequence of Definition 2 and Definition 5, a degenerate face can equivalently be defined as a face containing all degenerate points. Definition 8 below introduces an alternative definition of a degenerate face as a face containing a degenerate interior point. Some preliminary definitions are necessary.

Definition 6 True inequalities, implicit equalities, null variables

An inequality $ax < b$ is called a true inequality if $ax < b$ for some feasible point x otherwise, it is an implicit equality. If a non-negativity constraint holds as an implicit equality, the corresponding variable is called null.

A true inequality for a system defining a face will be said to hold strictly.

Definition 7 Relative interior

The relative interior of a face is its subset defined by the inequalities holding strictly.

A point in the relative interior of a face will simply be called degenerate interior point.

Definition 8 Degenerate interior point

The face of the polyhedron is degenerate if it contains a degenerate interior point.

Justification: The only tight constraints satisfied by an interior point are those defining the entire face (all others holding strictly). Therefore the face is defined by a degenerate system.

For a face with a relative interior, the converse holds, as noted immediately after Definition 5.

3.2 Degeneracy in the Standard Representation of Linear Programming

The standard representation $Ax = b$, $x \geq 0$ of a linear program deserves special attention among the systems $Ax \rho b$ because of the central role of the simplex algorithm. We delay the characterization of degeneracy in the simplex algorithm since the latter relies on extreme points

which will be analyzed in the next section. However, some interior point methods [Dikin, 1967, Karmarkar, 1984] work also with the standard formulation of a linear program and require primal or dual non-degeneracy at every point, although new results [Tsuchiya, 1991, 1992] dispense with such assumptions. In the standard formulation of a linear program, m will denote the number of equations.

Property 2: **Degeneracy in the interior point method**

x is a degenerate solution of the system Ax = b, $x \geq 0$ if and only if r(AX)<m.

Proof: Let Ax = b, x" = 0 represent the system of tight constraints, where x" is a subvector of x, the complement of which is x' > 0. Partition A as [A'|A"] accordingly. The rows of the system are linearly independent if and only if r([A'|0]) = m. But r([A'|0]) = r(AX). ■

In [Vanderbei, 1988], degeneracy is characterized directly by the rank of AX^2A^T, a matrix which appears in all interior point algorithms described in the previous chapter. Note that $r(AX^2A^T) = r(AX)$.

3.3 Degenerate extreme points

The next two sections focus on extreme points and bases of the simplex algorithm within which degeneracy was first analyzed. In fact, it can be shown that degenerate systems can be detected, using only extreme points. Definitions 1 and 2 of Section 3.1 are first adapted to extreme points defined by the standard formulation of a linear program. The ensuing definition is encountered in most introductory texts on linear programming; it is then extended to any linear programming formulation. Thus, degeneracy analysis can focus on a formulation originally designed by modelers instead of forcing them to reason on one privileged representation. We first recall:

Definition 9 **Extreme point**

A solution of a subsystem of tight constraints A'x = b' such that r(A')=n, corresponds to an extreme point.

Property 3:

An extreme point of a system Ax ρ b is also an extreme point of its standard representation A'x' = b, x ≥0.

Proof: Add slack variables, which adjoins an identity matrix to A. When the slack variables are zero, the corresponding nonnegativity constraints are added to the system of tight constraints. Otherwise, the unit coefficients of the slack variables in the slack constraints further expand the rank of the system of equalities. Altogether, the rank of the system has therefore been increased by the number of slack variables. ■

First, Definition 1 is specialized to the classical Definition 10 for extreme points of standard linear programs, (see for instance [Bazaraa, 1977, p. 92]).

Definition 10 **Classical definition of degenerate standard linear programs**

In the system Ax = b x≥0, where r(A)=m, an extreme solution is degenerate if and only if less than m of its components are positive.

Justification: An extreme solution is characterized by a system of tight constraints of rank n, which by definition of degeneracy, is less than the total number of tight constraints. Thus, there exist at least n+1 tight constraints, among which at least n-m+1 are non-negativity constraints. ■

Definition 10, the most frequently encountered in the literature of linear programming, requires linearly independent constraints Ax = b. However, the restriction can easily be relaxed, provided that Property 4 below is adjoined to it.

Property 4 Inclusion of linearly dependent constraints

If the system Ax = b, x≥0 has linearly dependent equalities, it is degenerate.

The proof is trivial. ■

Note in this case that all solutions are degenerate. The converse of Property 4 is obviously false. On the other hand, if some variables are not restricted to be non-negative, Definition 3 must be amended to become Definition 11 below. The following example shows why the modification is necessary.

Example: $x_1 + x_2 = 1$

$x_1 = 1.$

Whereas the constraints are linearly independent, x = (1,0) has 1 positive component (1 < m=2).

Definition 11 **Degenerate standard linear programs with unrestricted variables**

In the system Ax = b, $x^+ \geq 0$ where A has m equalities, an extreme solution is degenerate if and only if the constraints are linearly dependent or more than n - m of its restricted components x^+ are zero.

Justification: The justification of Definition 10 can be slightly modified by considering Property 4 and disregarding variables of unrestricted sign.

Definition 11 implies that the magnitude of an unrestricted variable is unrelated to degeneracy. Section 3.6 and Chapter 4 show that Definition 11 circumscribes precisely the cases leading to algorithmic difficulties such as cycling and truncated range analysis.

Property 2, the other definition of degeneracy based on AX, must be amended in presence of unrestricted variables, and becomes asymmetric.

Property 5:

x is a degenerate solution of the system Ax = b, $x^+ \geq 0$ if and only if $r([A^\pm|AX^+]) < m$.

Proof: Let Ax = b, $x^+_o = 0$ represent the system of tight constraints, where x^+_o is a subvector of x^+, the complement of which is $x^+_+ > 0$. Partition A as $[A^\pm|A^+_+|A^+_o]$ accordingly. The system has full row rank if and only if $r([A^\pm|A^+_+|0]) = m$. But $r([A^\pm|A^+_+|0]) = r([A^\pm|AX^+])$. ■

The preceding characterization of degeneracy by the rank of $[A^\pm|AX^+]$ is not consistent with the algorithmic computation of AX^2A^T for interior point methods; therefore even a nondegenerate point could cause numerical difficulty. On-going research attempts to reconcile the two notions, e.g. by bypassing the numerical difficulties created by very small unrestricted variables, and thereby to relegate numerical difficulties to degeneracy only [Andersen, 1993, Fourer, 1992, Tomlin, 1993, Vanderbei, 1988, 1993].

For a system Ax = b, $x^+ \geq 0$, a degenerate point belongs to a degenerate face containing all degenerate points, in particular extreme points. This shows that if a system of constraints has a degenerate point, it has a degenerate extreme point. Thus, even restrictive Definition 11 detecting pointwise degeneracy can be combined with Definition 2 to diagnose a degenerate system.

3.4 Degenerate Bases

In the simplex method, extreme points have traditionally been associated with bases. This relationship is also exploited in post-optimal analysis, and needs to be further refined if the program does not have full row rank.

3.4.1 Relationship between extreme points and bases:

To each extreme point of the standard representation of a linear program: $Ax = b$, $x \geq 0$, such that $r(A)=m$, one can associate a basis, setting all non-basic variables at zero. For example, the following proposition appears without proof [Bazaraa, 1977, Theorem 3.4, p. 92]:

Property 6: **Classical characterization of a degenerate point by bases**

For every extreme point (basic feasible solution) there corresponds a basis (not necessarily unique), and, conversely, for every basis there corresponds a (unique) extreme point. Moreover, if an extreme point has more than one basis representing it, then it is degenerate. Conversely, a degenerate extreme point has more than one basis representing it if and only if the system Ax = b itself does not imply that the degenerate basic variables corresponding to an associated basis are identically zero.

The relationship between degeneracy and multiple bases or null variables will be analyzed in Section 3.4.2. In the remainder of this section, the first part of the preceding characterization is adapted to general formulations $Ax = b$ where $r(A)<m$ and the sign of some variables is unrestricted. In the latter case, not all bases yield extreme points, as shown by the non-extreme point (0, 1, 2) corresponding to the basis (x_2, x_3) for the system of constraints:

$$x_1 - x_2 = -1$$
$$x_1 + x_3 = 2$$
$$x_3 \geq 0.$$

First, bases yielding extreme points of systems of linearly independent equalities are characterized

Property 7 **Degenerate linear programs and partial bases**

Every extreme solution of the system $Ax=b$, $x^+ \geq 0$, corresponds to one or several submatrices of rank r(A) containing all the unrestricted and the positive variables.

Proof: Every such submatrix plus n-r(A) non-negativity constraints form the full column rank system of tight constraints. Note that each variable outside the submatrix must have a non-negativity constraint. ■

In the sequel, it will be assumed that any basis representing a point will contain all its unrestricted components.

3.4.2 Degeneracy by multiple bases or implicit equalities (null variables)

Degenerate points are commonly identified with the existence of multiple bases (and with cycling of the simplex algorithm, as seen in Section 3.6). This characterization is refined in this next property.

Property 8:

An extreme solution of the system $Ax=b$, $x^+ \geq 0$, is degenerate if and only if it is either spanned over different columns by non-singular submatrices of rank r(A) including all unrestricted variables or all the non-negative basic variables that are zero in the associated submatrix are actually null, and these two cases are exclusive.

Proof: By Definition 11 and Property 7, if an extreme solution is degenerate, at least one non-negative variable is zero in the regular submatrix which can be used to produce the

equivalent system shown as a tableau of the simplex algorithm (obtained from the original formulation by row manipulation which does not change the rank of any subsystem):

I' 0 A'	0
0 I" A"	b"
A_1 A_2 A_3	b

Call $b' = \begin{pmatrix} 0 \\ b'' \\ b \end{pmatrix}$, where no element of b" corresponding to a non-negative variable is zero. If we can pivot on a non-zero coefficient α of A', then the new right hand-side becomes $b'-\alpha 0$ and in fact represents the same solution with a new submatrix. Note that the pivot may be negative.

Otherwise, A' = 0, i.e., the first k constraints of the system are $x_1=x_2=...=x_k=0$ for the k rows of A'. Note that the basis is then uniquely determined, since the only way to obtain a new regular submatrix with same right-hand side would be to exclude an unrestricted variable, which is not allowed.

Reciprocally, suppose that there are several submatrices such as B_1 and B_2 representing the same solution. All positive or unrestricted coordinates correspond to columns of both submatrices. Therefore the variables that are not common must be zero. ■

The following example [Bazaraa, 1977, page 90] characterizes a degenerate point with unique basis.

$$x_1 + x_2 + x_3 = 1$$
$$-x_1 + x_2 + x_3 = 1$$
$$x_1,\ x_2,\ x_3 \geq 0$$

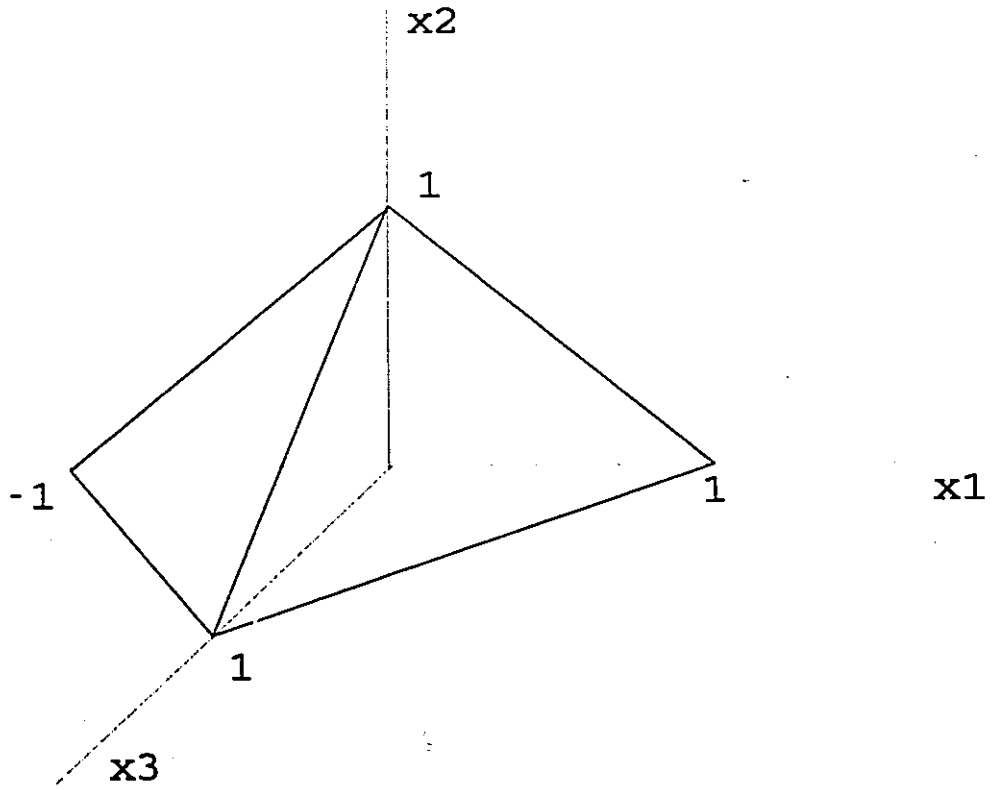

Figure 2 - The feasible region of example UB

The basic feasible solution x = (0,1,0) defines a degenerate extreme point. Among the four hyperplanes binding at x, there are three ways of choosing three linearly independent hyperplanes defining x. However, the basis, formed by x_1 and x_2 is unique. The simplex tableau is:

1 0 0	0
0 1 1	1

If only one basis defines a degenerate point, one variable is null or equivalently its non-negativity constraint is an implicit equality. Thus, if a system of constraints has only true inequalities, there is a one-to-one correspondence between a degenerate point and multiple bases. However, detecting true inequalities is not trivial (e.g. requires phase I of the simplex method).

3.5 The Phase I method and null variables

The preceding properties show that for full row rank systems $Ax = b$, $x \geq 0$, degeneracy corresponds to multiple bases. Although the aim of the thesis is to characterize degeneracy for more general systems, it was felt appropriate to devote a section to review the traditional method of simplifying standard systems $Ax = b$, $x \geq 0$ with $r=r(A) < m$ by dropping m-r constraints and performing standard simplex pivots on the r constraints left. Obviously, the m-r constraints must be selected judiciously, a non-trivial procedure solved for example by the simplex phase I method with up to m artificial variables. Null variables and their corresponding implicit equalities are revealed at the end of Phase I as basic artificial variables with a value 0. These can either be eliminated before proceeding to Phase II or be left in place, forming a degenerate solution with m-r basic null artificial variables, as shown in the last three tableaus of the following program. Details of the latter procedure are contained in [Bazaraa, 1977, p.149].

Example

$$
\begin{aligned}
x_1 + x_2 &= 1 \\
x_3 &= 1 \\
x_1 + x_2 + x_3 &= 2 \\
x_1,\ x_2,\ x_3 &\geq 0
\end{aligned}
$$

Add artificial variables x_4, x_5, x_6. Phase I will minimize $x_4 + x_5 + x_6$. Form an initial tableau:

1	1	0	1	0	0	1
0	0	1	0	1	0	1
1	1	1	0	0	1	2
-1	-1	-1	0	0	0	0

Bringing the first variable yields the updated tableau:

1	1	0	1	0	0	1
0	0	1	0	1	0	1
0	0	1	-1	0	1	1
0	0	-1	1	0	0	1

The third variable becomes basic. Thus, at the end of Phase I, the tableau is:

1	1	0	1	0	0	1
0	0	1	0	1	0	1
0	0	0	-1	-1	1	0
0	0	0	1	0	0	2

Degeneracy is clearly detected as one artificial variable cannot be simply removed as if it was nonbasic. If the columns of other artificial variables are removed, the optimal tableau becomes:

1	1	0	0	1
0	0	1	0	1
0	0	0	1	0
0	0	0	0	2

The artificial variable is null. We can remove the redundant row and get the following tableau:

1	0	0	0
0	1	1	1
0	0	0	1

3.6 Dual degeneracy:

Next, we focus on optimization by associating an objective function to a system of constraints. In the simplex algorithm, each reduced cost measures the influence of its corresponding variable on the objective function. A zero reduced cost indicates that a shift in the corresponding variable will maintain the same objective value. The expression dual multipliers is used to encompass dual variables and reduced costs. Here, we first develop a classical definition of dual degeneracy, based on the standard representation of a linear program.

3.6.1 Classical definition of dual degeneracy

Definition 12: **Degenerate multipliers**

Consider the standard linear program: max cx s.t. Ax = b, x≥0, where r(A)=m.

A set of dual multipliers is degenerate if and only if more than m of its reduced costs are zero.

Definition 13: **Dual degenerate problem**

A problem is dual degenerate if and only if a set of dual multipliers is degenerate.

Bases or points that share the same objective value are generally called *alternative* or *alternate*. In this section, the name alternate will be restricted to those bases with identical dual multipliers, excluding multiple bases for a unique solution with different multipliers. However, some multiple bases can also be alternate. Alternate points or bases are commonly associated to dual degeneracy. An exact relationship is defined next.

3.6.2 Dual degeneracy and alternate points or bases in the simplex algorithm:

Consider a column of the simplex algorithm with one zero reduced cost; the following cases may arise:

a) All of the coefficients of the column are zero. In this column, no simplex pivot can yield any alternate solution. If every column with zero reduced cost displays this pattern, any pivot in another column will modify the dual multipliers, therefore there is no alternate solution.

b) One coefficient is non-zero. Then a pivot in a row with a non-zero right hand-side coefficient b_i yields an alternate basis.

Note that feasibility has not been considered in the pivot, therefore some alternate solutions may be infeasible. The partial relationship between dual degeneracy and alternate solutions reveals a great similarity between primal and dual degeneracy. A direct analysis of dual extreme, null multipliers, etc. would display further relationships. In a primal algorithm such as the simplex algorithm, the sign of the dual multipliers is not restricted, leaving fewer special cases (e.g. unrestricted variables) than primal degeneracy has. Thus, within the focus of the thesis on primal

formulations, a direct analysis is omitted, and emphasis is immediately placed on the primal-dual analogy. Later, differences will be noted.

3.6.3 Relationship between dual and primal degeneracy

Since degeneracy has been associated with systems of constraints, it is natural to interpret dual degeneracy as (primal) degeneracy of a dual system of constraints involving as variables the dual multipliers. Row and column rank deficiency are in obvious dual relationship, recognized by the simple names primal and dual degeneracy. The relationship is further analyzed in the context of the simplex algorithm operating on the standard form of linear programs. This analysis will yield dual interpretations of alternate bases, true inequalities and implicit equalities, and characterizations of degenerate dual bases. First, Definition 12 is related to Definition 1 for the dual problem. A short review of the dual simplex algorithm, similar to [Lasdon, 1970], crystallizes the relationship between primal and dual viewpoints by showing that the dual simplex algorithm for a primal problem is a primal simplex algorithm for the dual problem. Consider a primal problem in standard form:

$$\begin{aligned} &\text{minimize } cx \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

with a basis B partitioning the problem as $c = (c_B\ c_N)$, $A = (B\ N)$. Consider its dual:

$$\begin{aligned} &\text{maximize } ub \\ &uA + \upsilon = c \\ &\upsilon \geq 0. \end{aligned}$$

The problem can be written:

maximize ub

$u\mathcal{A} = c$,

where $\mathcal{A}^T = \begin{pmatrix} A \\ I \end{pmatrix}$, $b = \begin{pmatrix} b \\ 0 \end{pmatrix}$, $u = (u, \upsilon)$, and $\upsilon \geq 0$. A basis is given by $\mathcal{B} = \begin{pmatrix} B^T & O \\ N^T & I_N \end{pmatrix}$; thus,

$\mathcal{A}^T = \begin{pmatrix} B^T & I_B & O \\ N^T & O & I_N \end{pmatrix}$. A current tableau of the simplex algorithm applied to the dual problem can be obtained by premultiplying $(\mathcal{A}^T, c^T)$ by the basis inverse $\mathcal{B}^{-1} = \begin{pmatrix} B^T & O \\ N^T & I_N \end{pmatrix}^{-1} = \begin{pmatrix} B^{-T} & O \\ -N^T B^{-T} & I_N \end{pmatrix}$.

The current tableau is:

$$\mathcal{B}^{-1}\mathcal{A}^T = \begin{pmatrix} B^{-T} & 0 \\ -NB^{-T} & I_N \end{pmatrix}\begin{pmatrix} B & I_B & 0 \\ N & 0 & I_N \end{pmatrix} = \begin{pmatrix} I & B^{-T} & 0 \\ 0 & -N^T B^{-T} & I_N \end{pmatrix}$$

and the updated right-hand side:

$$\mathcal{B}^{-1}c^T = \begin{pmatrix} B^{-T} & O \\ -N^T B^{-T} & I_N \end{pmatrix} c^T = \begin{pmatrix} B^{-T} c_B^T \\ c_N^T - N^T B^{-T} c_B^T \end{pmatrix}$$

which contains the dual multipliers (reduced costs) mentioned in Definition 12. Therefore, dual degeneracy in the simplex algorithm can be directly interpreted as (primal) degeneracy in the dual simplex algorithm. In particular, alternate primal bases correspond to multiple dual bases associated with one set of multipliers, the one solution of the dual problem and reciprocally.

On the other hand, a degenerate dual basis is not alternate if it contains null dual multipliers. The latter correspond to a constraint that is never tight, or strongly redundant. Similar, an implicit dual equality - i.e. a zero reduced cost - corresponds to the nonnegativity restriction of a variable that is always positive (strongly redundant nonnegativity). Thus, a strongly redundant inequality may render a dual basis degenerate. As in Phase I of the simplex algorithm, it may be interesting to remove strongly redundant inequalities from a formulation: in such a case, degenerate bases will correspond to alternate (optimal) points, which could coincide.

Dual degeneracy and rank of the matrix AX

For the sake of completeness, an example shows that there is no relationship between the alternate definition of primal degeneracy, i.e., the row rank deficiency of AX and dual degeneracy. A problem can be dual degenerate while all its matrices AX have full row rank, as in the following example:

$$
\begin{aligned}
&\text{maximize } x_1 + x_2 \\
&\text{st} \\
&x_1 + x_2 + x_3 = 6 \\
&\quad x_2 + x_4 = 3 \\
&x_1 + 2x_2 + x_5 = 8 \\
&x_1, x_2, x_3, x_4, x_5 \geq 0
\end{aligned}
$$

First, the row rank of AX is displayed for every extreme point. A non-degenerate solution is (6,0,0,3,2) with a full row rank AX

$$
AX = \begin{bmatrix} 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 6 & 0 & 0 & 0 & 2 \end{bmatrix}
$$

For the point (0,0,6,3,7)

$$AX = \begin{bmatrix} 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 7 \end{bmatrix}$$

For the point (5,1,0,3,0) :

$$AX = \begin{bmatrix} 5 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 5 & 2 & 0 & 0 & 0 \end{bmatrix}$$

For the point (0,3,3,0,1):

$$AX = \begin{bmatrix} 0 & 3 & 3 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 1 \end{bmatrix}$$

And for (6,0,0,3,1):

$$AX = \begin{bmatrix} 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and therefore it has full row rank for any strictly convex combination of these.

The dual problem is:

$$\begin{aligned}
&\text{minimize } 6u + 3v + 8w \\
&\text{st} \\
&u \quad + w \leq 1 \\
&u + v + 2w \leq 1 \\
&u, v, w \geq 0,
\end{aligned}$$

and it is degenerate with solution: (1,0,0,0,0).

Dual degeneracy can also be characterized by the column deficiency of UA, where U is the diagonal matrix formed by the dual multipliers, which is in no immediate relationship with AX.

3.6.4 Independence of primal and dual degeneracy:

The previous relationships arise from a dual perspective on the same formulation, i.e. the dual properties of a primal problem are interpreted as the primal properties of the dual problem. On the other hand, there is virtually no relationship between primal and dual properties of a given formulation. First, primal and dual degeneracy occur independently as shown in the following example where (1,0,0) is a non-degenerate solution, with degenerate dual multipliers (1,0,0). Actually, the primal problem is not degenerate.

$$\begin{aligned}
&\text{maximize } x_1 + x_2 + x_3 \\
&x_1 + x_2 + x_3 \leq 1 \\
&-x_1 + x_2 + x_3 \leq 1 \\
&x_1,\ x_2,\ x_3 \geq 0.
\end{aligned}$$

Also, each basis associated with a degenerate point may correspond to different dual multipliers. In this sense, primal multiple bases may not correspond to dual multiple bases. The primal multiple bases (1,2), (1,3), (1,4) of the program:

maximize $x_1 + 2x_2$

s.t.

$x_1 + x_2 \leq 1$

$2x_1 + x_2 \leq 2$

$x_1,\ x_2 \geq 0$

has a dual formulation with no degenerate solution:

minimize u_1+2u_2

s.t.

$u_1+2u_2 \geq 1$

$u_1+u_2 \geq 2$

$u_1,u_2 \geq 0$

3.6.5 Degeneracy and multiple dual variables:

It is also often thought that degenerate points feature multiple dual multipliers. Intuitively, multiple dual multipliers should arise with multiple bases. However, degenerate points may correspond to a unique basis. Also, multiple bases may yield only one dual multiplier as in the simple counterexample given by the degenerate point (1,0,0) of the linear program:

maximize 0x

$x \leq 1$

$2x \leq 2$

with unique dual multiplier (0,0) as solution of the dual program:

minimize $u_1 + 2u_2$

$u_1 + 2u_2 = 0$

$u_1,\ u_2 \geq 0$

Finally, some of the multiple bases may correspond to infeasible alternate dual solutions, sometimes ignored. This omission is more frequent in a dual setting, where some alternate bases may correspond to infeasible primal solutions; for example, the set of degenerate dual multipliers (1, 1, 0, 1) (1, 1, 0, 0) corresponds to a unique primal feasible basis (x_1) and extreme point (1, 0, 0) of the linear program:

maximize x_1-x_2

st

x_1-x_2≤1

x_1,x_2≥0

3.7 Degeneracy and cycling in the simplex algorithm

Cycling, which occurs when one solution is repeated at several iterations of the algorithm, is generally ascribed to degeneracy. This subsection shows that cycling is in fact related to multiple or alternate bases only. Since at an iteration of the simplex algorithm, the change of objective value is Δx_j (c_j -z_j), cycling can occur either when Δx_j or c_j - z_j vanish, which correspond roughly to primal and dual degeneracy.

Cycling under primal degeneracy.

We first examine cycling when a non-basic variable replaces a zero basic variable i.e., Δx_j is zero. The unit matrix representing the basis is assumed to be at the left of the following schematic tableau, where j is the index of the entering variable, r designates the row and the column of the leaving variable and i denotes any row different from r.

0	a_{ij}	b_i
1	a_{rj}	0
0	c_j	

The right hand-side is not changed by a pivot.

$-a_{ij}/a_{rj}$	0	b_i
$1/a_{rj}$	1	0
$-c_{ij}/a_{rj}$	0	

Cycling is avoided in two cases, neither of which features multiples bases:

- First, if the zero basic variable is unrestricted, it may not leave the basis. In the simplex algorithm, the minimum ratio rule is designed to limit the increase of the entering variable in order to keep the corresponding basic variable nonnegative, which unconstrained variables need not satisfy. Thus, it can be omitted for this unrestricted basic variable which will change sign, but remain basic. Note from the previous sections that a zero unconstrained variable does not create degeneracy.

- Second, a non-negative basic variable at level zero will not be chosen to leave the basis if all the coefficients a_{rj} of its corresponding row r are zero as shown below. In this case, the basis is unique.

0	a_{ij}	b_i
1	0	0
0	c_j	

Consequently, in the simplex method, cycling due to primal degeneracy can only occur if a current solution can be represented by multiple bases. Reciprocally, multiple bases can be used to cause cycling. Consider again the current tableau, repeated below: as a zero variable leaves the basis, the entering variable becomes basic at level zero as shown in the following schematic tableau.

0	a_{ij}	b_i
1	a_{rj}	0
0	c_j	

Note that the sign of the reduced cost of the variable r is the opposite of c_j and under the usual rules of the simplex algorithm, the variable r may not enter the basis again. [Marshall, 1969] has shown that at least 6 variables are involved in cycling. However, if it entered the basis, it would yield a feasible solution, leading to a series of cycling pivots. Some modified rules, such as at the end of Phase I [Bazaraa, 1977], do allow the variable to enter the basis.

$-a_{ij}/a_{rj}$	0	$b_i - a_{ij}b_r/a_{rj}$
$1/a_{rj}$	1	b_r/a_{rj}
$-c_j/a_{rj}$	0	

Cycling under dual degeneracy

When the reduced cost $c_j - z_j$ vanishes, dual degeneracy may not cause cycling if all elements of the column with 0 reduced cost are zero, preventing any pivot, as the second column of the schematic tableau:

0	0	b_i
1	0	b_r
0	0	

In this case, there is no alternate dual basis. On the other hand, any set of alternate dual bases can be used to create cycles as shown next. Consider the schematic tableau displaying the entering column:

0	a_{ij}	b_i
1	a_{rj}	b_r
0	0	

Suppose that the variable j corresponding to a column with 0 reduced cost is entering the basis. Selecting the leaving variable r satisfying $a_{rj} > 0$ and $b_r/a_{rj} \leq b_i/a_{ij}$ for all indices i such that $a_{ij} > 0$, yields the tableau:

$-a_{ij}/a_{rj}$	0	$b_i - a_{ij}b_r/a_{rj}$
$1/a_{rj}$	1	b_r/a_{rj}
0	0	

For a second pivot, select the variable r to reenter the basis, since its reduced cost is zero. Now, the variable j can be chosen as the leaving variable because $\frac{1}{a_{rj}} > 0$ and the minimum ratio satisfies:

$$\frac{b_r/a_{rj}}{1/a_{rj}} \leq \frac{b_i - a_{ij}/a_{rj}\, b_r}{-a_{ij}/a_{rj}} = -b_i \frac{a_{rj}}{a_{ij}} + b_r.$$

for all indices such that $-a_{ij}/a_{rj} \geq 0$, which holds because $b_r \leq -b_i \frac{a_{rj}}{a_{ij}} + b_r$, itself deriving from the condition $0 \leq b_i$.

In summary, cycling can only occur in the simplex method if a current solution can be represented by multiple bases.

Chapter 4 - Degeneracy and post-optimal analysis

Post-optimality analysis, the best-known implementation of data perturbation for linear programming, consists of three parts: sensitivity, range and parametric analyses. This chapter focuses mostly on range analysis, addressing the question: for what change of (rim) coefficient does the solution remain optimal? The adverse effect of degeneracy on range analysis with the simplex method is reviewed and remedies are proposed using both the simplex and affine scaling algorithms. The analysis is related to [Mehrotra, 1992].

4.1 Range estimation with the optimum tableau of simplex algorithm:

Consider a linear program in standard form:

minimize cx
subject to
$Ax = b$,
$x \geq 0$

The optimal solution x* can be partitioned as $x_B^* = B^{-1}b$, $x_N^* = 0$, where B indexes the basic coordinates. If the right hand-side is changed by $t\Delta b$, the potential optimum $x_B = B^{-1}(b + t\Delta b) = x^*_B + tB^{-1}\Delta b$, remains feasible if and only if $x^*_B + tB^{-1}\Delta b \geq 0$. Let $\bar{x}_i$ denote the i-th coefficient of $B^{-1}\Delta b$. The preceding feasibility condition can easily be translated as a range on the values of:

RES: a right hand-side estimate by the simplex method

$$\max_{\bar{x}_i>0} -\frac{x^*_{B_i}}{\bar{x}_i} \leq t \leq \min_{\bar{x}_i<0} -\frac{x^*_{B_i}}{\bar{x}_i}$$

In particular, when Δb is the j-th unit vector (to implement a change of one coefficient of the right hand-side), the coefficients of the j-th column of B^{-1} are often represented explicitly in a full tableau of the simplex algorithm.

To illustrate the estimation of RES, two examples are given, the second of which displaying alternative optima.

maximize $x_1 + x_2$

s.t.

$x_1 + 2x_2 \leq 3$

$2x_1 + x_2 \leq 3$

$x_1,\ x_2 \geq 0$

At the optimum $x^* = (1,1)$, the range of b_t is determined by RES according to the following calculations: $\Delta b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $B^{-1} = \frac{1}{3}\begin{pmatrix} -1 & 2 \\ 2 & -1 \end{pmatrix}$, $B^{-1}b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $B^{-1}\Delta b = \frac{1}{3}\begin{pmatrix} -1 \\ 2 \end{pmatrix}$. Thus, $-3/2 \leq t \leq 3$, a range which can also be found by direct (geometric) arguments. Obviously, the optimum x will change with t, but the optimum set of dual multipliers (1/3, 1/3) will remain the same.

The estimate RES depends on the basis B that yields the optimum dual multipliers. With degeneracy, alternative bases may not yield the full ranges for which the optimal solution remains the same, as shown next.

Example AB: Alternative bases.

maximize $-x_1 - 2x_2$

s.t.

$-x_1 - x_2 \leq 0$

$-x_1 - 2x_2 \leq -1$

$x_1,\ x_2 \geq 0$

An optimum of the problem is $x_1 = x_4 = 0$, $x_2 = x_3 = 0.5$, for a basis (x_2, x_3), where x_3 and x_4 are the slack variables. The range of b_1 is determined by RES by the following calculations:

$$\Delta b = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad B^{-1} = \begin{pmatrix} 0 & -.5 \\ 1 & -.5 \end{pmatrix} \quad B^{-1}b = \begin{pmatrix} .5 \\ .5 \end{pmatrix} \quad B^{-1}\Delta b = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$ yielding the lower bound -.5, whereas

a direct analysis yields a lower bound -1. Thus, with alternative bases, range analysis can only answer the question: for what change of right hand-side does the basis remain optimal?
This failure to detect full range is often ascribed to degeneracy, but in fact arises from the presence of alternative bases. A more detailed discussion from a dual viewpoint is contained in the next section.

4.2 Range estimation of the objective coefficients with the simplex algorithm

Estimating the range of the coefficients of the objective function is in dual relationship with the estimation RES of the preceding section. It is treated independently here for two reasons. First, some lack of symmetry introduced by the standard form of the simplex tableau yields qualitatively different characterizations of ranges. Second, the range analysis of the coefficients of the objective function is easier to interpret geometrically and clearly shows the limitations of the procedures CES proposed below (with the simplex algorithm) and CEI proposed in Section 4.3 (with interior point methods). The problem is first stated in a general framework. Consider the linear program in standard form:

minimize cx
subject to
$Ax = b$,
$x \geq 0$

and denote by r the vector of reduced costs.

Property 9: **Complementary slackness for standard programs.**

A feasible solution x of a standard linear program is optimal if and only if:

$r \geq 0$ (NONNEG)

and,

for any $j \in J$, $r_j = 0$ *whenever* $x_j > 0$. (COMPLEM)

■

Using these conditions, it is possible to deduce some estimates on the range of objective coefficients: given an optimal feasible solution x, find some values c' for which x remains an optimal feasible solution of the modified program:

P': minimize c'x

subject to

A'x = b',

$x \geq 0$.

We must find a dual vector u' such that r'=c'-u'A satisfies the sufficient conditions (NONNEG) and (COMPLEM). A brief review of the classical range estimation with the simplex algorithm is given.

The partition (B, N) is now applied to the objective coefficients. With a change of objective coefficients $t\Delta c$, let $\bar{r}_j = (\Delta c_N - \Delta c_B B^{-1})_j$; the reduced costs become $(c_N + t\Delta c_N) - (c_B + t\Delta c_B)B^{-1} = r + t\bar{r}$. The solution remains optimal if $r + t\bar{r} \geq 0$. The preceding optimality condition can easily be translated as a range on the values of t.

CES: A cost range estimate by the simplex method

$$\max_{\bar{r}_j > 0} -\frac{r_j}{\bar{r}_j} \leq t \leq \min_{\bar{r}_j < 0} -\frac{r_j}{\bar{r}_j}$$

As in the estimate RES, B^{-1} is often represented explicitly in a full simplex tableau and CES can be readily implemented.

The estimate CES depends on the basis B. With degeneracy, different bases may not yield the full ranges for which the primal solution remains optimal. Two examples will be examined:

Example MB: Multiple basis.

maximize x_2

s.t.

$x_1 + x_2 \leq 1$

$x_1 + 2x_2 \leq 2$

$x_1, x_2 \geq 0$

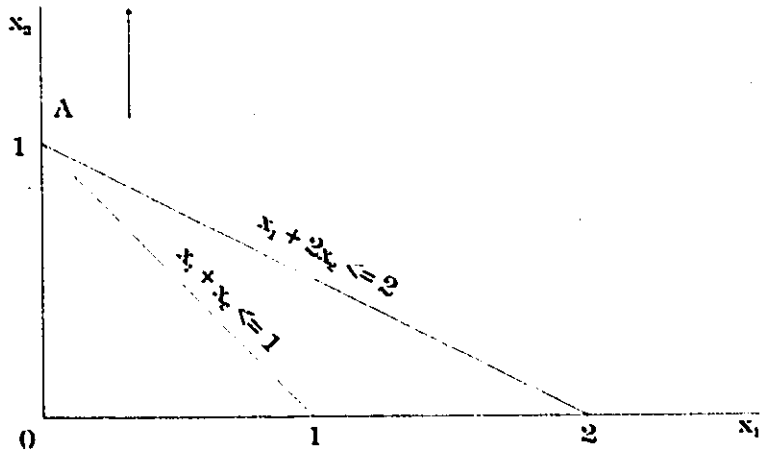

Figure 3 - The feasible region of example MB.

The feasible region is given in Figure 3, where the arrow denotes the gradient of the objective function. The optimum is represented by the point A, for $x_1=0$, $x_2=1$. Using bases, the preceding method of range analysis of the simplex algorithm diagnoses the following ranges of objective coefficients:

Basis (x_2,x_3):

c_1 can increase by .5 and decrease by infinity,

c_2 can increase by infinity and decrease by one.

Basis (x_2,x_4):

c_1 can increase by one and decrease by infinity,

c_2 can increase by infinity and decrease by one.

Direct (geometric) arguments show that the basis (x_2,x_4) provides full ranges for the solution to remain optimal. The analysis that uses the basis (x_2,x_3), provides a range of increase of c_1 smaller than one, because it fails to detect that the point remains optimal through basis changes.

Three bases can be associated with the optimum x. The basis (x_1,x_2) does yield the optimum primal, but is not dual optimal, therefore would not yield the final tableau of a primal simplex algorithm, neither be used for the estimation CES. The preceding example shows that if a program has multiple optimal bases, the estimation CES can only answer the question: for what change of objective coefficient does the basis remain optimal?

Degenerate solutions can be defined by a unique basis, and correspondingly, simplex-based range analysis yield proper ranges, as shown in the following example:

Example UB: Unique basis.

maximize $c_1x_1 + c_2x_2 + c_3x_3$

s.t.

$x_1 + x_2 + x_3 = 1$

$-x_1 + x_2 + x_3 = 1$

$x_1, x_2, x_3 \geq 0$

Consider values of c_1, c_2, c_3 yielding an optimum of $x_1 = x_3 = 0$, $x_2 = 1$ (e.g. $c_1 = c_3 = 0$, $c_2 = 1$). The range estimation CES finds the full ranges of rim values, namely:

c_1 can get any values from minus infinity to infinity

c_2 can get any values from c_3 to infinity and

c_3 can get any values from minus infinity to c_2

b_1 can be increased by infinity and decreased by zero

b_2 can be increased by zero and decreased by two.

In conclusion, if the optimal basis is unique, the evaluation CES also answers the original question: for what change of objective coefficients does the solution remain optimal.

Returning briefly to the determination of right hand-side ranges, Example MB will be useful to reemphasize the distinction between primal and dual degeneracy seen in Section 3.6, and its influence on range analysis.

Example MB: Multiple basis.

The estimation CES can be applied to the dual problem AB of Section 4.1 to provide ranges of right hand-side coefficients for dual multipliers to remain optimal. Note that for each dual multiplier, the basis is unique and consequently the estimates provided are proper. Example AB of Section 4.1 is the dual formulation of Example MB.

Basis (x_2,x_3), dual (0,.5):

b_1 can increase by infinity and decrease by zero,

b_2 can increase by zero and decrease by two.

Basis (x_2,x_4), dual (1,0):

b_1 can increase by zero and decrease by one,

b_2 can increase by infinity and decrease by zero.

The same results can be found by direct (geometric) arguments, i.e. considering the objective function of the dual problem. Thus, the multiplicity of primal bases is unrelated to that of dual bases i.e. the bases of the dual problem.

4.3 Range estimation with interior point methods

Post-optimal analysis can also be performed after an optimal solution has been found by interior point methods [Adler, 1989, Mehrotra, 1992]. While all linear programming textbooks present range analysis by the simplex method, the estimates, derived from complementary slackness, can also be used for range analysis with interior point methods. As in the preceding section, for ease of derivation, the analysis focuses on objective coefficients, with the understanding that the

results can be fully extended to the right hand-side. We temporarily ignore primal degeneracy. Recall the notation B=AX. The search direction can be calculated as $\bar{c} = [\ I - B^T(BB^T)^{-1}B\ ]\ Xc^T$. Suppose a variation Δc is proposed. Calculate $\bar{c}' = [I - B^T(BB^T)^{-1}B]\ X\ \Delta c^T$. An algorithm is given below which, for a given direction of variation Δc, estimates a range for t within which x^* remains an optimal solution of the preceding program.

CEI: An interior point estimate for cost range analysis

Let U be a matrix satisfying $AX^2\ A^T\ U = AX$. Define:

$r = (I - A^T\ U\ X)\ c^T$

and

$\bar{r} = (I - A^T\ UX)\ \Delta c^T$

If $\bar{r}\,x \neq 0$. STOP: the allowable range of variation is the direction Δc is 0. Otherwise, the range is:

$$\max_{\bar{r}_j>0} -\frac{r_j}{\bar{r}_j} \leq t \leq \min_{\bar{r}_j<0} -\frac{r_j}{\bar{r}_j}$$

Note that the test $\bar{r}\,x \neq 0$ enforces sufficient condition (COMPLEM), and the above inequalities enforce sufficient condition (NONNEG).

There are 2 important cases where the algorithm CEI provides a full range of variation. First, if $\bar{r}\,x \neq 0$, then for any variation $c' = c + t\Delta c$, $(r + t\bar{r})\,x \neq 0$. Therefore x cannot be optimal.

The other case is when x is not degenerate, as shown in the next property.

Property 10: CEI without primal degeneracy

If the optimal point x^* is not primal degenerate, x is optimal for the modified problem P' if and only if the variation stays within the range determined by CEI.

Proof: Property 9 and the algorithm CEI provide the 'if' part constructively. For the converse, suppose that x* is optimal for P'. Let r' = (I - $A^T UX$) c'. By primal non-degeneracy, r' $\geq$ 0 is the unique set of optimal reduced costs. But if c' was outside the range given by CEI, one component of r' would be negative, which is a contradiction. ■

The expression $(BB^T)^{-1}$ is undefined under primal degeneracy. However, the system $BB^T U = B$ still has a solution and the calculation CEI can easily be extended, as shown next.

Property 11: CEI under primal degeneracy

If r(A) > 0, then there exists a matrix U such that $BB^T U = B$

Proof: If B has full row rank, so does BB^T, therefore there exists a solution U. Otherwise, denote by I a maximal index subset of independent rows of B and $\bar{I}$ its complement. Suppose without loss of generality that I comprises the first |I| rows.

Let $B = \begin{bmatrix} B_I \\ B_{\bar{I}} \end{bmatrix}$; $B_I B_I^T$ has full rank, therefore there exists a matrix U_I such that: $B_I B_I^T U_I = B_I$.

Since the rows of $B_{\bar{I}}$ are linear combinations of the rows of B_I, there exists a matrix N such that

$B_{\bar{I}} = NB_I$, therefore $B_{\bar{I}} B_I^T U_I = NB_I$. A solution is $U = \begin{bmatrix} U_I \\ \mathbf{0} \end{bmatrix}$, since $BB^T \begin{bmatrix} U \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} B_I B_I^T U_I \\ B_{\bar{I}} B_I^T U_I \end{bmatrix} = \begin{bmatrix} B_I \\ B_{\bar{I}} \end{bmatrix}$

which proves the property. ■

However, the preceding proposition does not indicate how to choose a matrix U that yields an optimal reduced cost r. Such a matrix would be provided by an affine scaling algorithm, since the termination test requires r $\geq$ 0. In case of primal degeneracy, pure affine scaling algorithms

often require many iterations to find such a vector and provide an optimal reduced cost. The matrix U may not yield the largest range of variation of the objective coefficient.

As an illustration, the example MB of the previous section, displaying degeneracy, is treated by CEI, which does not need yield the full range of variation of the coefficients of the objective function.

Example MB: Multiple basis.

maximize x_2

s.t.

$x_1 + x_2 \leq 1$

$x_1 + 2x_2 \leq 2$

$x_1, x_2 \geq 0$

It can be actually shown that the estimation CEI yields the full range of variation of the coefficients of the objective function for no trajectory inside the feasible region leading to the optimum. Each trajectory inside the feasible region is parametrized by h, whereas ε measures the closeness of a feasible point x of the optimal point. Thus, let $x = [\varepsilon h, 1-\varepsilon, \varepsilon(1-h), \varepsilon(2-h)]$, where ε is arbitrarily small and $0 \leq h \leq 1$ (the analysis holds for h outside of the preceding range, and h can be considered very large for completeness). Calculating r and r' by the formulae of CEI yields:

$\lim_{\varepsilon \to 0} r = [(2-h)^2+2(1-h)^2, 0, 4(1-h), h^2+2(1-h)^2] / [h^2+(2-h)^2+4(1-h)^2]$

and

$\lim_{\varepsilon \to 0} \bar{r} = [(2-h)^2+4(1-h)^2, 0, -2h^2, h^2] / [h^2+(2-h)^2+4(1-h)^2]$

For an infinitesimally small neighborhood, a sufficient condition for $r + t\bar{r}$ to be non-negative is $\max \{-1, -2(1-h)^2 / h^2, -[2(1-h)^2 + (2-h)^2] / [4(1-h)^2+(2-h)^2]\} \leq t \leq 2(1-h) / h^2$, and there is no single value h that yields the full range of allowable variation of the first coefficient of the

objective function (i.e. any value no greater than -1). Below is a small sample of values for the ranges of variations.

h	t_L	t_U
0	-3/4	∞
.4226	-.8255	6.464
.5	-11/13	8
1	-1	0

In conclusion, the range estimate depends on the direction from which the optimum is approached, which may vary with the initial point. No direction yields a range which dominates others strictly, and a central trajectory [Karmarkar, 1984], displayed in the table for h=.4226, does not yield the largest range. The next section will show that the range estimates can be refined by a linear program; actually, the vector r' provides a starting direction for determining the full range of variation of the coefficients.

Finally, for the example displaying a degenerate solution with unique basis, the estimate CEI does not provide a full range.

Example UB': Unique basis.

maximize x_2

s.t.

$x_1 + x_2 + x_3 = 1$

$-x_1 + x_2 + x_3 = 1$

$x_1, x_2, x_3 \geq 0$

The calculations are

$$B = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$BB^T = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ which is singular. However, a solution of the system $BB^TU = B$ is

$U = \begin{pmatrix} 0 & a & 0 \\ 0 & 1-a & 0 \end{pmatrix}$ for any scalar a. With this value, $A^TUX = \begin{bmatrix} 0 & 2a-1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ and

$I-A^TUX = \begin{bmatrix} 1 & 1-2a & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix}$. Finally, for $c^T = (0\ -1\ \ 0)$ and $\Delta c^T = (0\ -1\ \ 0)$, $r = \bar{r} = (2a - 1,\ 0,\ 1)$.

Notice that $a > 1/2$ for r to be nonnegative (and the affine scaling algorithm to detect optimality). With these calculations, the range of c_2 would be unchanged, but for $\Delta c^T = (-1\ 0\ 0)$, $\bar{r} = (-1\ 0\ 0)$ and the upper bound range of c_1 would be only $2a-1$.

By dual correspondence with the estimation CEI, a range estimation for the coefficients of the right hand-side could be based on a dual affine scaling direction: $R^{-2}A^T(AR^{-2}A^T)^{-1}\ b$, where R is the diagonal matrix corresponding to the reduced costs of the problem [Adler, 1989a]. Primal-dual affine algorithms aim to obtain the relationship $RX = a\mathbf{1}$ at the optimum, with which the direction could also be characterized by

$$X^2A^T(AX^2A^T)^{-1}\ b \qquad \text{(DUAL\_DIR)}$$

a property used at the end of this chapter.

4.4 A linear program for full range analysis

Many proposals for range analysis in the presence of multiple bases have been offered. In the worst case, an auxiliary linear program, shown below, must be solved. Here, the focus is again on the range analysis of the right hand-side, with an implicit dual correspondence with the range analysis of the objective coefficients.

4.4.1 Range analysis of the right hand-side

Denote by A° the submatrix of A corresponding to zero reduced costs of the program P. The range of variation of a right hand-side coefficient is found as the solution of the pair of programs (for $\delta = \pm 1$):

R: δmaximize δt

$$A^\circ x - \Delta b t = b$$

$$x \geq 0$$

Example AB: Alternative bases.

Example:

δ maximize δt

s.t.

$$x_1 + x_2 - t \geq 0$$

$$x_1 + 2x_2 = 1$$

$$x_1, x_2 \geq 0$$

4.4.2 RES, a first iteration for R:

Next, a relationship is displayed between the preceding program R and the range evaluation RES presented in Section 4.1 . First, note that the optimal solution x^* of the original program P yields a feasible solution (x^{o*}, t=0) of the new program R. Suppose temporarily that A° contains the basis B of the optimal solution x^*, which also yields a basis for the new program R. Since t is not restricted to be non-negative, and $\delta = \pm 1$, t is a candidate to enter the basis. If δ=1, the minimum ratio is $\min_{\bar{b}_i<0} \frac{x_{Bi}}{-\bar{b}_i}$, occurring for $i=i_o$ and the objective function will increase by

$x_{B_{i_1}} / -\bar{b}_{i_1}$, precisely the estimate RES. If $\delta=-1$, the minimum ratio is $\min_{\bar{b}_i>0} \frac{x_{Bi}}{-\bar{b}_i}$, occurring for $i=i_1$,

and the objective function will decrease by $x_{B_{i_1}} / -\bar{b}_{i_1}$.

Therefore, if B is contained in A°, the range evaluation RES encapsulates the reoptimization of the pair of programs R by the simplex algorithm. If, on the other hand, the optimal solution x^{o^*} was primal degenerate, some basic variables would be zero, and A° would not contain B. Enlarging A° to contain B will provide an initial basis, but, after pivot, the solution may remain $(x^{o^*}, t=0)$. The classical range evaluation RES is equivalent to augmenting A° in order to restore the base B, which is not entirely consistent with the program R, as it introduces degeneracy. In this case, RES constitutes only the first iteration of the program R which in the worst case may require a very large number of iterations before completion.

The optima are $-\infty$ and 1, corresponding to the full range described in Section 4.2.

4.4.3 Range analysis of the coefficients of the objective function with the simplex algorithm.

By duality, corresponding programs can be designed for a range analysis of the coefficients of the objective function.

C: maximize cx

$$A^\bullet x = 0$$

$$\Delta cx = \delta$$

$$x \geq 0$$

where $A^\bullet$ is formed by the columns j of A such that $x_j = 0$.

4.5 Range calculation by the affine scaling method:

In this section, the affine scaling algorithm is applied to the problem R. As was shown for the simplex algorithm, the approximation REI is shown to constitute also the first iteration of a complete optimization for full range analysis. For ease of derivation, the analysis focuses on the right hand-side; the results can be extended to objective coefficients. Problem R has an objective vector $c'^T = (0, -1)^T$ and a constraint matrix: $A'=(A_o|\Delta)$, where Δ will be used as short-hand notation for $-\Delta b$.

Note again that the optimal solution x^* of the original program yields a feasible solution $(x^{o*}, t=0)$ of the program R. The additional variable t is unrestricted, and will be treated here as having asymptotically large lower bound -M, yielding $X' = \begin{pmatrix} X & 0 \\ 0 & M \end{pmatrix}$. Denote $B' = A'X'$. The affine scaling direction can be calculated as: $\bar{c} = [\, I - B'^T(B'B'^T)^{-1}B' \,] X' c'^T$. Let $\Delta = \Delta M$, $H = (BB^T)^{-1}$, $k_0 = (\Delta^T H\Delta)^{-1}$ and $k = (1 + M^{-2}k_0^{-1})^{-1}$. A simple rank-one update yields $(B'B'^T)^{-1} = H - kH\Delta\Delta^T H^T$, hence $\bar{c} = X'^2 \left\{ \begin{pmatrix} 0 \\ -1 \end{pmatrix} - A'^T [H - kH\Delta\Delta^T H^T] M\Delta \right\}$, or simply:

$$\bar{c} = X'^2 \left\{ \begin{pmatrix} 0 \\ -1 \end{pmatrix} + M\, A'^T H\Delta\, (1 - k\Delta^T H^T \Delta) \right\}, \text{ which can be further simplified to:}$$

$$\bar{c} = X'^2 \left(\begin{pmatrix} 0 \\ -1 \end{pmatrix} + kM\, A'^T H\Delta \right) = kX'^2 \begin{bmatrix} MA^{oT} H\Delta \\ -1 \end{bmatrix}.$$

As M increases, the direction $\bar{c}$ tends to $k_o\begin{bmatrix} X^2 A^{oT} H\Delta \\ -1 \end{bmatrix}$, in accordance with the dual affine direction (DUAL_DIR) presented at the end of Section 4.3. To maintain feasibility of the dual affine scaling, the step size must maintain the reduced costs nonnegative, precisely the criterion CEI. Hence, CEI can be viewed as the first iteration of the affine scaling algorithm to solve the program R.

In conclusion, there are close relationships between the evaluation of range by ratio and programs to detect full ranges, both within the simplex and the affine scaling methods. No example was chosen featuring r(A) < m. The classical approach is to eliminate redundant constraints, and perform the analysis on a full rank system. Chapter 3 showed that subbases can be directly used; similarly, the interior point method can be extended to degenerate and rank deficient problems

Chapter 5 - Data perturbation with Constraint Programming Languages

The previous chapter focused on the modeling and computational problems created by degeneracy or multiple solutions. Both types of problems stem from the representation of linear programs: in the case of degeneracy, there are several algebraic representations of a feasible set; for multiple solution, there is no unique numerical representation of the optimum set.

This chapter analyzes some features of a solutions system offering a symbolic representation of linear programming. First note that the input of conventional linear programs (implicit representation by a family of constraints) differs fundamentally from their output (explicit numerical representation of an optimal point), thereby hindering much interactive problem solving. Because of their implicit, or symbolic input, mathematical programming systems are related to symbolic modeling [Wolfram, 1991, Complete Logic Systems, 1987, Konopasek, 1984, Steele, 1980, Sutherland, 1963]. However, users typically expect an explicit output, and solution analysis programs such as Analyze [Greenberg, 1983, 1986, 1987, 1993] attempt to reconcile both representation modes, for instance by examining the output in light of the input. Some languages [Jaffar, 1990, Durand, 1994] offer an uniform mode of representation based on constraints which form the mainstay of linear programs. For example, optimal regions can be reported implicitly through the system of constraints that defines them. This system can further be amended or analyzed in post-optimal analysis. A representation based solely on constraints is unfamiliar to most mathematical programmers, but avoids the difficulties encountered in the previous chapters.

5.1 Constraint Programming Languages

A number of programming languages based on constraints have been implemented [Borning, 1981, Gosling, 1983, Leler, 1988]. Many of these languages deal with arithmetic constraints. These languages are often called *declarative* because constraints are nondirectional, unlike

numerical calculations. Thus, constraint programming languages share much in common with declarative logic programs. Several constraint programming languages use the syntax of Prolog, the most popular logic programming language [Colmerauer, 1982].

5.2 CLP($\Re$)

The programming language CLP($\Re$) was designed to be an instance of the Constraint Logic Programming Scheme, a family of rule-based constraint programming languages. Unlike Prolog, which allows some arithmetic as an add-on feature not integrated in the original design of the language, CLP($\Re$) defines the semantics of arithmetic constraints directly in the domain of real numbers $\Re$. Certain privileged predicates, such as equality, and various forms of inequalities ($\leq, \neq, \geq$) define *constraints*, and may comprise arithmetic expressions over real numbers as the following examples show. Thus, the domain of computation $\Re$ of CLP($\Re$) is the algebraic structure consisting of uninterpreted functors over real numbers. An important property of CLP($\Re$) is that the constraints are treated uniformly in the sense that they are used to specify the input parameters to a program, they are the only primitives used in the execution of a program, and they are used to describe the output of a program.

A first example illustrates the lack of directionality of CLP($\Re$) i.e., its declarative aspect.

Example 1 Consider the exchange of Canadian $ (Y) for US $ (X). The exchange is expressed by the predicate:

```
exchange_rate(X,Y) :- X=0.83*Y.
```

and thus, the goal,

```
?- exchange_rate(X,2)
```

queries the US$ value of Can $2. Such a program can be written in most versions of Prolog. However, the goal,

```
?- exchange_rate(1.76,Y).
```

succeeds in the CLP($\Re$) whereas it would fail in Prolog. Many extensions of Prolog propose to handle equations, using *local propagation*, but are rarely able to solve systems of simultaneous equations. By contrast, CLP($\Re$) is able to respond correctly when queried about the following simultaneous equations:

Example 2.

```
?- 4X+2Y=12, 2X+4Y=12.
```

Implementation of a CLP language, and of CLP($\Re$) in particular, raises new problems in the design of a constraint-solver. For example, the constraint solver must have a good 'average' behaviour and must be *incremental* in the sense that adding new constraints must not entail the re-solving of old constraints. Constraints are filtered through an inference engine, an engine/solver interface, an equation solver and an inequality solver. This sequence of modules reflects a classification and prioritization of the classes of constraints. Modules solving higher priority constraints are isolated from the complexities of modules solving lower priority constraints. This multiple-phase solving of constraints, together with a set of associated algorithms, gives rise to a practical system.

The following paragraphs sketch the solution mechanisms used by CLP($\Re$). The interpreter, illustrated in Figure 3.1, contains a Prolog-like *engine*, a constraint *solver*, and a module that provides an *interface* between the two.

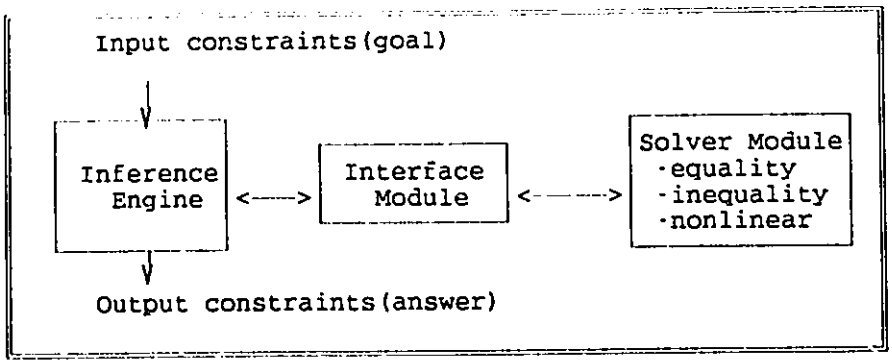

Figure 3.1: A diagram of the components of the CLP($\Re$) interpreter

The engine of the system is an inference mechanism that recognizes the constraints, handing to the interface those that it cannot solve. The major step performed by the engine is the creation of bindings of certain variables, plus the usual Prolog engine functions (unification, backtracking).

The interface evaluates complex arithmetic expressions and transforms constraints into a small number of standard forms. In this transformation, the input constraints are dispatched to different modules of the solver. The interface assists the engine as illustrated in the following example.

The constraint solver solves constraints that cannot be handled by the engine or the interface. This solver is the main novelty in the implementation of the CLP($\Re$) system. It is divided into three sub-modules, an equation solver, an inequality solver, and a nonlinear constraint solver. The nonlinear constraint solver provides a delay mechanism for nonlinear constraints by storing nonlinear equations, and sends equations to the linear equation solver whenever they become linear. Given a set of constraints, the solver first determines the solvability of that set and, if solvable, computes the solution. CLP($\Re$) uses a modified simplex method [Shambin, 1974] to solve sets of linear constraints.

A variant of Example 2 is now used to illustrate the role of the engine and of the interface in preparing the work for the solver:

```
first(T,U) :- 4T+2U=12.
second(V,W) :- 2V+4W=12.

?- first(X,Y), second(X,Y).
```

The engine produces the following sequence of equations:

X=T
Y=U
4T+2U=12
X=V
Y=W
2V+4W=12

The interface simplifies this set and only sends to the solver the following two equations:

4X+2Y=12
2X+4Y=12

The solver then returns the solution:

```
X=2, Y=2.
```

CLP($\Re$) also has the potential to produce implicit output. As constraints are defined syntactically, not only do they operate as tests, but they can define results. For example the following goal:

```
?- X1>=1, X1>=4, X1=<5.
```

returns the answer below:

```
X1>=4, X1=<5.
```

The ability to return implicit answers is a powerful feature of CLP($\Re$). Answering a goal is defined in terms of satisfiability of the corresponding set of constraints and not in terms of finding a value. At each step of the process, the interpreter has to deal with only a single question: are the constraints solvable?

5.3 Linear programming with CLP($\Re$)

CLP($\Re$) can solve linear programs as satisfaction problems [Brown 1989, Muli, 1992]. The simplest method is to express them as primal-dual pairs of systems of constraints. The symbolic answers will in turn address the problems caused by degeneracy. Some examples of post-optimal analyses are given next. Consider the variant of linear program of Chapter 4:

Example MB': Multiple basis

maximize $c_1x_1+c_2x_2$.

s.t.

$x_1 + x_2 \leq 1$

$x_1 + 2x_2 \leq 2$

$x_1, x_2 \geq 0$

Its dual is:

minimize u+2v,

$u+v \geq c_1$,

$u+2v \geq c_2$,

$u \geq 0$,

$v \geq 0$,

and the weak theorem of duality requires:

$u+2v = c_1x+c_2x_2.$

The program can therefore be solved as the primal-dual system:

$x_1 + x_2 \leq 1$

$x_1 +2x_2 \leq 2$

$x_1, x_2 \geq 0$

$u+v \geq c_1,$

$u+2v \geq c_2,$

$u \geq 0,$

$v \geq 0,$

$u+2v = c_1x_1+c_2x_2.$

The primal-dual system can be represented by CLP($\Re$) as:

```
lp(C1,C2,X,Y,U,V):- X+Y <= 1,          /*                */
                    X+2*Y <= 2,        /*    Primal      */
                    X >= 0,            /*  constraints   */
                    Y >= 0,            /*                */

                    U+V >= C1,         /*                */
                    U+2*V >= C2,       /*    Dual        */
                    U >= 0,            /*  constraints   */
                    V >= 0,            /*                */
                    U+2*V = C1*X+C2*Y. /*  Primal=Dual   */
```

Consider now the linear program for given values of objective coefficients $c_1 = 0$, $c_2 = 1$:

maximize x_2

s.t.

$x_1 + x_2 \leq 1$

$x_1 + 2x_2 \leq 2$

$x_1, x_2 \geq 0.$

This program has an optimal dual face:

$x = 0$

$y = 1$

$u + 2v = 1$

$0 \leq v \leq 0.5$

diagnosed explicitly by CLP($\Re$) as:

```
1 ?- lp(0,1,X,Y,U,V).

U = -2*V + 1
Y = 1
X = 0
0 <= V
V <= 0.5

*** Yes
```

The optimum of this program remains x = 0, y = 1 for any value of the objective coefficients:

$c_1 \leq u + v$

$c_2 = u + 2*v$

which can be found by the CLP($\Re$) query:

```
2 ?- lp(C1,C2,0,1,U,V).

C2 = U + 2*V
C1 <= U + V
0 <= V
0 <= U

*** Yes
```

As shown in the previous chapter, any value of the objective coefficient c_1 can increase by one and decrease by infinity, and c_2 can increase by infinity and decrease by one. This can be obtained by two separate range analyses, a first one for the coefficient c_1 performed by fixing the coefficient c_2 at 1, as:

```
3 ?- lp(C1,1,0,1,U,V).

U = -2*V + 1
C1 + V <= 1
0 <= V
V <= 0.5

*** Yes
```

and the other range analysis for the coefficient c_2, performed by fixing the coefficient c_1 at 0, as:

```
4 ?- lp(0,C2,0,1,U,V).

C2 = U + 2*V
0 <= V
0 <= U
0 <= U + V

*** Yes
```

Although the preceding goals included the dual variables for a representation of dual faces, a more concise predicate yields a projection onto the primal variables only:

```
lp(C1,C2,X,Y): - lp(C1,C2,X,Y,U,V)
```

When queried, this predicate produces the results in the format expected of range analysis. The preceding queries are now repeated in their projected form. First is a combined range analysis for the objective coefficients:

```
2 ?- lp(C1,C2,0,1).

C1 <= C2
0 <= C2

*** Yes
```

As Bradley [1977] points out, range analysis of each coefficient is the intersection of the preceding surface with the axes. This is best shown as:

```
3 ?- lp(C1,C2,0,1),C1=0.

C1 = 0
0 <= C2

*** Yes
```

The preceding query could be simplified as:

```
2 ?- lp(0,C2,0,1).

0 <= C2

*** Yes
```

Range analysis of c_1 can be obtained by the query:

```
4 ?- lp(C1,1,0,1).

C1 <= 1

*** Yes
```

This chapter shows that many of the difficulties encountered by post optimal analyses stem from the numerical representation of the solutions adapted by traditional mathematical programming systems. A representation by constraints alleviates these problems. As [Lassez, 1990] points out, defining a primal solution by constraints is a *dual* representation, but not of the traditional type. Appealing as this symbolic representation is, CLP($\Re$) is only an approximate implementation of a constraint programming scheme, relying on numerical approximations. It is hampered by two factors:

- speed: symbolic languages, as CLP($\Re$), are slow and practically not competitive with numerical linear programming solvers,
- precision: proper selection of constraints to represent the output hinges on a sophisticated, but slow interface and some judicious tolerances chosen for the symbolic solver, which will degrade with the size of the problem and the relative magnitude of its data. In a sense, the choice of computational rules embedded in the solver removes many issues of data representation and perturbation from users' decision.

Chapter 6 - An Interior Point Heuristic for Integer Programming

Data perturbation analysis is hindered by discrete variables as occur in mixed integer programs. Early proposals for post-optimal analysis of integer programs include [Marsten, 1975, Piper, 1975]. Since, the problem has been shown to be NP-complete [Blair, 1986] which has refocused interest on heuristic sensitivity analysis. From a different viewpoint [Moores, 1988] pointed out that many integer programs are highly degenerate, and yet relatively few studies analyze degeneracy in integer programming. For example [Balas, 1972] shows that degeneracy hampers the use of all-integer pivots in matching problems.

Given the difficulty of a direct perturbation analysis, an alternative is to perturb the optimum and analyze the admissible data that keep it optimal. This approach, when applied to systems of equations, has been termed backward error analysis [Givens, 1954, Wilkinson, 1963], with a goal of showing that numerical approximations were to be viewed as applying to the input data, rather than the output [Hotelling, 1943]. Backward analysis is related to inverse optimization [Bitran, 1981]. This approach is attractive to integer programming, where a near optimum can be obtained heuristically.

This chapter presents an implementation of the most intuitive heuristic for integer programming: rounding off fractional linear programming solutions, which has been shown to be effective on some known classes of integer programs [Chandrasekaran, 1984]. A simple enhancement is to solve a linear program and then generate some integer vectors with a random function centered around the value of the linear program optimum. Most vectors in a given sample are infeasible, as measured by the sum of infeasibilites, but the minimum sum of infeasibilities in the sample decreases uniformly with the sample size. The objective of the experiments is simply to assess the potential of interior point trajectories for rounding, not to design a sophisticated heuristic which would compete with a large family of efficient searches [Fox, 1990, Toyoda, 1975, Vassilev, 1991].

6.1 A simplex-based heuristic for integer programming

Successful rounding of the linear programming solutions is obviously hampered by their proximity to the boundary of the feasible region. Since the simplex algorithm yields extreme solutions, it may be helpful to take a positive step into the interior of the feasible region [Hillier, 1969, Jeroslow, 1975]. Instead, in a preliminary study not reported here, the constraints of the original problem were simply tightened judiciously before solving the linear program. For example, the linear relaxation of the original problem can be solved and the slack variables used to adjust the right-hand sides. Only then is the second linear program (with adjusted constraints) used to generate integer solutions. In fact, at least 5 successively modified linear programs were needed before achieving some measure of success; a unique but larger modification was unsuccessful. Thus, adjustment is slow and an interior point algorithm is an attractive alternative to the simplex algorithm, since its iterates stay strictly inside the feasible polytope [Karmarkar, 1988, Vanelli, 1991].

6.2 A direct primal IM approach to integer programming:

With this aim, an affine scaling algorithm can be used for the linear relaxation of the problem:

maximize cx
$Ax = b$
$x \geq 0$ and integer

As the iterate progresses toward optimality in the interior of the feasible region, its coordinates are rounded to provide candidate integer solutions.

Each coefficient of the constraint matrix A is generated from a uniform distribution U[-16,16] A feasible solution x° is generated with density .8+ U[0, .2] and each coordinate is generated from [0,16]. The right-hand side is then calculated as Ax° + U[0,20] to guarantee that at least one feasible solution be somewhat removed from the constraints. Each coefficient of the objective function is generated from U[-100,100].

In a first experiment, to assess the potential of rounding, for each interior point x' of the affine trajectory leading to the optimum, all integer points are generated by rounding, i.e. form the set $\{ (\lfloor x_j \rfloor + \delta)_{j=1,\ldots,n}$ for $\delta = 0$ or $1 \}$. Nine problem of 5 constraints and 5 variables were solved. The tables of Appendix B, reporting the computational results, indicate in each row (corresponding to one iteration of the affine scaling algorithm):

- how many rounded vectors x' are feasible, i.e., satisfy each constraint of the program $Ax' \leq b$,
- the value of the best objective function cx' at each iteration relative to the optimum ('% optimal'),
- the average objective value of every feasible point generated at a given iteration.

The last two values are then displayed relative to the objective function of the current fractional point on the trajectory ('% current').

A sample table is reproduced here for discussion. On the average, the number of integer feasible solutions decreases as the iterates near the optimum, but the relative quality of their objective value increases markedly with respect to the linear programming optimum (the integer programs were not solved, and their optimal value is unknown). In the first iterations, the quality of the integer solution is quite poor, as shown by 100% deviation from the linear programming optimum. Also, the average objective value of the entire sample is not too far from the best integer point. Therefore, a heuristic would not be required to find the best neighbor of the fractional point.

From the last two columns, we note also that the objective values of the integer points generated tend to get closer to the objective value of the *current solution* (as opposed to the optimal one). The objective value of the integer solution could be higher than that of the current point, hence some negative percentage deviations are contained in the tables.

In some cases, integer solutions are generated at any point of the trajectory leading to the optimum. In other cases, successful rounding ceases when the point gets close to optimality.

We can conclude that the heuristic has a better quality as the fractional point gets closer to the optimum, but rounding becomes erratic. For the problem illustrated in Table 1, two graphs represent the progress of the heuristic along the trajectory. It is clear that the potential of the

heuristic to generate integer points decreases as the linear program reaches optimality. On the other hand, when provided, the integer solutions have a much better objective function. A heuristic should therefore carefully measure the closeness of the current point to the optimum to spend most effort on rounding.

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
15	1.00	1.02	0.98	0.86
13	1.25	1.30	0.35	0.21
40	1.15	1.25	0.56	0.27
30	0.89	0.99	5.17	1.50
16	0.76	0.84	-0.68	-0.09
24	0.19	0.29	-0.08	0.05
14	0.19	0.28	-0.08	0.05
13	0.12	0.24	-0.16	-0.01
3	0.12	0.19	-0.16	-0.08
7	0.09	0.19	-0.21	-0.08
7	0.09	0.19	-0.16	-0.03
4	0.02	0.07	0.02	0.07
2	0.02	0.04	0.02	0.04
2	0.02	0.04	0.02	0.04

Table 1: A complete search of all integer solutions near the fractional one.

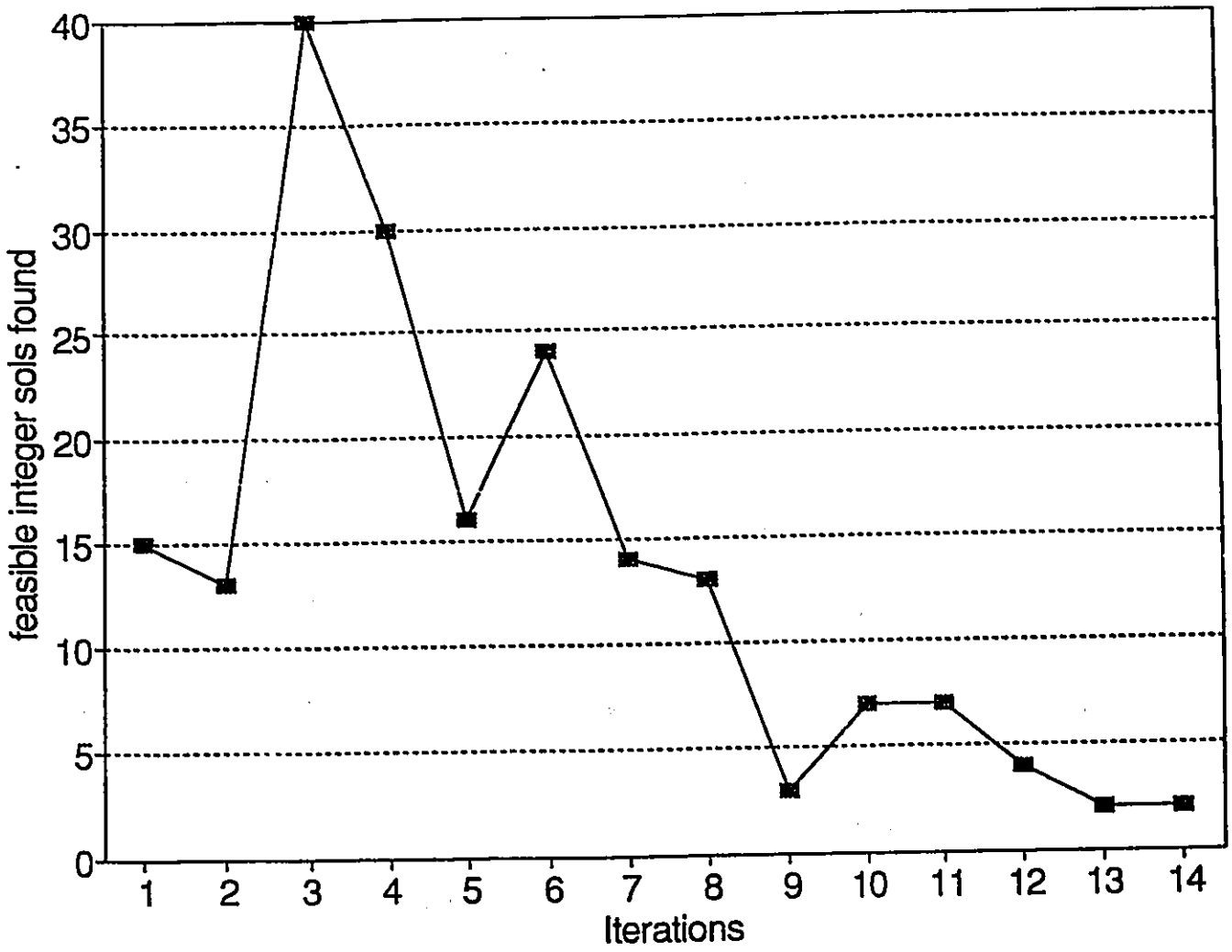

Figure 4 - Number of integer points neighboring an iterate of the affine scaling algorithm

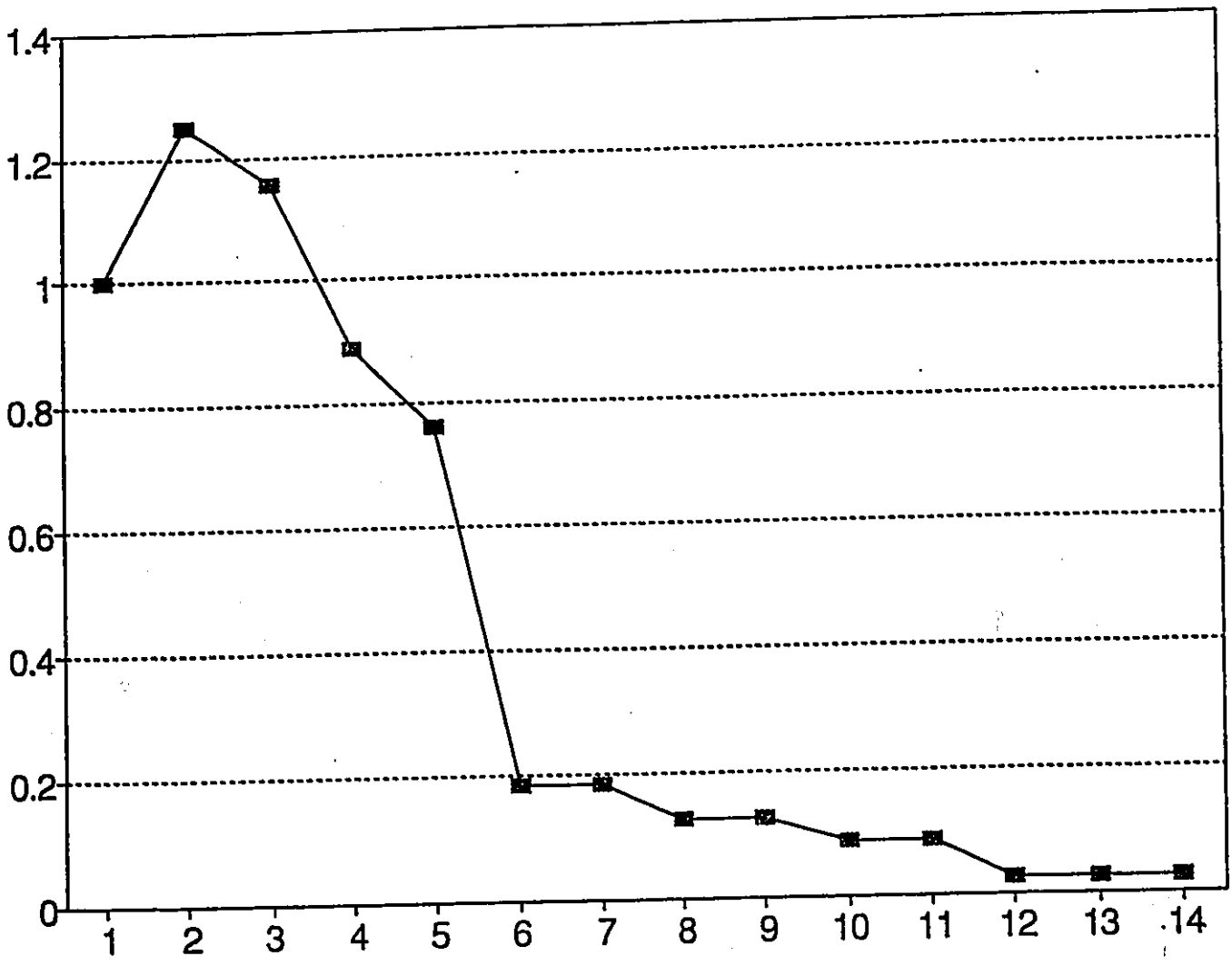

Figure 5 - Relative objective value of the integer point vs. optimal linear relaxation

6.2.3 *Random generation of integer points*

In a second experiment, the fractional iterate is rounded randomly. Several random trials are performed at each iteration. For a fractional iterate $(x_j)_{j=1,\ldots,n}$, the sample generated will be $\{(x'_j = \lceil U[x_j, 1+x_j] - \lfloor x_j \rfloor \rceil + \lfloor x_j \rfloor)_{j=1,\ldots,n}\}_t$. Therefore $E_t(x'_j)=x_j$ for j=1,...,n.

Example: Suppose that the current point is (3.57, 5.43, 6.38, 2.72). Below is a sample of generated integer vectors:

(3 5 6 3)
(3 5 7 3)
(3 6 6 3)
(3 6 7 2)
(3 6 7 3)
(4 5 6 3)
(4 5 7 2)
(4 6 6 3)
(4 6 7 2)

Notice that the sample average of each coordinate is roughly the value of the fractional point.

The random generation is applied to the same problems as generated previously. Tables in Appendix B display the same performance measures as in the first experiment. Samples of 800 vectors were generated. Although the size of these samples seems to be large, they can be generated quite quickly, and the calculation of their objective value and feasibility is straightforward. On the other hand, much replication is expected in each sample, especially as the distribution chosen favors integer coordinates close to those of the fractional point. The following table, corresponding to Figure 4, shows that random generation is feasible. Because of the replication, some iterations yield more feasible solutions than the complete enumeration of the previous

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	–	–	–	–
2	1.32	1.32	0.28	0.28
8	1.34	1.34	0.16	0.16
5	0.99	0.99	1.17	1.17
0	–	–	–	–
25	0.23	0.23	0.00	0.00
25	0.23	0.23	0.00	0.00
25	0.23	0.23	0.00	0.00
25	0.23	0.23	0.00	0.00
25	0.23	0.23	0.00	0.00
20	0.16	0.20	0.05	0.08
9	0.00	0.00	0.04	0.04
7	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00

Table 2: A random search of integer solutions for the same problem as above.

experiment. For the problem investigated previously, random generation was quite successful. displays the same performance measures. For the particular problem investigated

A last series of experiments was conducted for problems of 10 variables and ten constraints. The general pattern of rounding is similar to the one described above, with the following differences:

- systematic exploration of all integer points around a fractional iterate is quite time consuming, and the experiments were limited to a small number of problems. Results are reported on page 122 seq. of Appendix B.
- rounding seems less successful for large problems than the preceding set; correspondingly, random generation, not reported here, was not promising.

Conclusion - Directions for future research

This study addressed the difficulties of data representation of linear programming, partly by illustration and partly by proof. Many aspects of degeneracy and post-optimal analysis were deliberately left out, and need to receive further attention. Some specific omissions are listed below.

Degeneracy in Linear Programming

Although the role of degeneracy on post-optimal analysis is clearly shown, Chapter 4 does not extend the estimations RES, CES and CEI to linearly dependent systems ($r(A) < m$).

In the analysis of degeneracy, little attention has been paid to (strong) complementary slackness. Some interior point algorithms guarantee the optimal solution display strong complementary slackness, with some properties much closer to simplex based solutions.

Post-optimal analysis

The procedure for post-optimal analysis is very similar to 'warm starts' for program reoptimization, as are used in many applications, such as column generation or decomposition methods. The estimation of a direction of range evaluation should also provide a good direction for reoptimization [Browne, 1993].

Integer Programming Heuristics

The analysis of Chapter 6 addresses only the *potential* of a generation of integer points by an interior point method and is fairly conclusive from that viewpoint. It leaves aside its efficient implementation. Many questions are forcibly ignored and should be addressed by future studies:

- *choice of a good stopping criterion*: In the primal method, an optimal reduced cost may not be approached until the incumbent is optimum, and although former iterates may be near-optimal, the algorithm may not have detected it. Direct measures of feasibility based on the constraints may fail to reflect the degree of optimality of a given incumbent point.

- *influence of the initial point*: the position of the primal iterates depends on the initial point. A heuristic could be designed to generate a good initial solution.
- benefit of a *centered solution*: intuitively, a centered trajectory should stay away from the constraints that typically inhibit rounding. A centered trajectory is obtained by applying the affine scaling algorithm to the modified problem shown below:

maximize $cx - \mu \sum_{i=1}^{n} \ln x_i$

s.t.

$A x = b$

$x \geq 0$

in which a parameter μ is either chosen experimentally, or of the same order as the duality gap. On balance, it must be pointed out that the definition of center is algebraic and does not forcibly lead to a geometric center.

- extending the idea of a barrier, or merit function would it be advantageous to amend the objective function in order to create zones of attraction around integer feasible points [Karmarkar, 1988].

Asymptotic integer programming:

The rounding of fractional solutions is obviously more successful if the point is well within the feasible region. Although intuitive, this capability has both practical and theoretical applications. There is agreement that rounding is acceptable when the magnitude of the solution is large, with some theoretical justification found in asymptotic integer programming [Gomory, 1965]. At the opposite, rounding is less likely to succeed for 0-1 programming, and some justification was given [Balas, 1973]. Given the wide acceptability of rounding, experiments as those of Chapter 6 could revive practitioners' interest in fast and simple heuristics for integer programming.

Bibliography

Adler, I. and R.D.C. Monteiro, *A Geometric View of Parametric Linear Programming*, Tech. Report, February, 1989.

Adler, I., N. Karmarkar, M.G.C. Resende, and G. Veiga, "An implementation of Karmarkar's algorithm for linear programming" *Mathematical Programming*, 44, pp.297-335, 1989.

Adler, I. and S. Verma "Linear and Convex Optimization by Neural Networks," *ORSA Computer Science Technical Section Conference*, Williamsburg, Virginia, January 6, 1994.

Andersen, E.D., private communication, technical report forthcoming, Faculty of Social Sciences, Odense University, 1993.

Balas, E and M. Padberg, "On the Set Covering Problem", *Operations Research* **20**, pp. 1153-1161, 1972.

Balas, E, "A note on the Group Theoretic Approach to Integer Programming and the 0-1 Case", Operations Research, 21 (1), 1973.

Barnes, E.R., "A Variation on Karmarkar's Algorithm for Solving Linear Programming Problems", *Mathematical Programming*, Volume 36, pp. 174-182, 1986.

Bazaraa, M.S., J.J. Jarvis and H.D. Sherali *Linear Programming and Network Flows*, Second Edition, John Wiley and Sons, 1977.

Beale, E.M.L, "An Alternative Method for Linear Programming", Proceedings of the Cambridge Philosophical Society, 50(4), 1954.

Bitran, G.R., V. Chandru, D.E. Sempolinski and J.F. Shapiro, "Inverse Optimization: An Application to the Capacitated Plant Location Problem", 27, 10, *Management Science*, 1981.

Blair, C.E. and R.G. Jeroslow (1986), "Computational Complexity of Some Problems in Parametric Discrete Programming", *Mathematics of Operations Research*, 11(2), pp.241-260.

Borning, A., "The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory", *ACM Transaction on Programming Languages and Systems*, 3(4), pp.353-387, 1981.

Bradley, G., A. Hax and T. Magnanti, *Applied Mathematical Programming*, Addison-Wesley Publishing Co., 1977..

Brown, R.G., J.W. Chinneck and G.M. Karam, *Optimization With Constraint Programming Systems*, Proceedings of "Impact of Recent Computer Advances on Operations Research", Williamsburg, January 1989. R. Sharda et al. (eds.), Publications in Operations Research Series, No. 9, Elsevier Science Publishers, 1989.

Browne, C. B., "Using Interior Point Methods to Solve the Multicommodity Network Flow Problem", Masters of Science thesis, University of Ottawa, 1993.

Chandrasekaran, R., S.N. Kabadi, and K.G. Murty, "Some NP-Complete Problems in Linear Programming", *Operations Research Letters*, 1, (3), pp. 101-104, 1982.

Chandrasekaran, R., "Integer Programming Problems for which a Simple Rounding Type Algorithm Works", Progress in Combinatorial Optimization, *Academic Press Canada*, pp. 101-106, 1984.

Charnes, A., "Optimality and Degeneracy in Linear Programming", Econometrica, 20 (2), pp. 160-170, 1952.

Colmerauer, A., *Prolog and Infinite Trees*, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseilles II, 1982.

Complete Logic Systems, *Trilogy User's Guide*, , North Vancouver, British Columbia, Canada, 1987.

Dantzig, G.B., *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J., 1963.

Deif, A.S., "Sensitivity analysis in linear systems", Springer-Verlag, 1986.

Dikin, I.I., "Iterative Solution of Problems of Linear and Quadratic Programming", *Soviet Math Docl.*, 8, 3, 1967.

Durand, J., M. Epstein, E. Freeman and J. Xu, "On Solving Optimization Problems with the CLP Language clp(X)", *ORSA Computer Science Technical Section Conference*, Williamsburg, Virginia, January 6, 1994.

Fiacco, A.V., ed., "Mathematical programming with data perturbations" in *Lecture notes in pure and applied mathematics*, v. 73, Dekker, 1982.

Fiacco, A.V., "Introduction to sensitivity and stability analysis in nonlinear programming", *Mathematics in science and engineering*; v. 165 Academic Press, 1983.

Fiacco, A.V., ed., "Mathematical programming with data perturbations II" in *Lecture notes in pure and applied mathematics*, v. 85, Dekker, 1983.

Fourer, R. and S. Mehrotra, "Solving Symmetric Indefinite Systems in an Interior-Point Method for Linear Programming." Technical report 92-01, Dept. of Industrial Engineering and Management Sciences, Northwestern University, 1992.

Fox & Nachtscheim, "An Analysis of six greedy selection rules for a class of 0-1 IP models", Naval Res. Log. 37, 1990, pp. 299-307.

Gal, T., "Shadow Prices and Sensitivity Analysis in Linear Programming under Degeneracy", *OR Spektrum*, 8, 1986.

Gal, T., "Degeneracy problems in mathematical programming and degeneracy graphs", *ORION* 6, 1, pp. 3-36, 1990.

Gal, T. (ed.) "Degeneracy in Optimization Problems', *Annals of Operations Research*, 1994.

Gal, T. and H.J.P. Kruse, "Survey of solved and open problems in the degeneracy phenomenon", *Mathematical Programming (B)*, 42, 1, pp. 125-133, 1988.

Gill, P.E., W. Murray and M.H. Wright, "Practical Optimization", *Academic Press*, 1981.

Givens, W. "Numerical computations of the characteristic values of a real symmetric matrix," Oak Ridge National Laboratory, ORNL-1574, 1954.

Gomory, R.E., "On the Relation between Integer and Noninteger Solutions to Linear Programs", Proc. of the National Academy of Sciences, 53, pp. 260-265, 1965.

Gosling, J., "Algebraic Constraints", Technical Report CS-83-12, Carnegie-Mellon University, Pittsburgh, Pa., 1983.

Greenberg, H.J., "A functional Description of ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models", *ACM Transactions on Mathematical Software*, Naval Research and Logistics Quarterly, 9 (1), pp.18-56, 1983.

Greenberg, H.J. and F.H. Murphy, "Computing Market Equilibria with Price Regulations Using Mathematical Programming", *Operations Research*, 935, 1985.

Greenberg, H.J., "An Analysis of Degeneracy", *Naval Research and Logistics Quarterly*, 33, pp. 635-655, 1986.

Greenberg, H.J., "ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models", *Operations Research Letters*, 6 (5), pp.249-255, 1987.

Greenberg, H.J., "How to Analyze the Results of Linear Programs: Part 1: Preliminaries", *Interfaces* 23 (4), pp.56-67, 1993.

Greenberg, H.J., "How to Analyze the Results of Linear Programs: Part 2: Price Interpretation" *Interfaces* 23 (5), pp.97-114, 1993.

Hillier, F.S., "Efficient Heuristic Procedures for Integer Linear Programming with an Interior", *Operations Research* 17, pp. 600-637, 1969.

Hotelling, H. "Some new methods in matrix calculations," *Annals of Mathematical Statistics 14*, pp. 1-34, 1943.

Jaffar, J., S. Michaylov, P.J. Stuckey and R.H.C. Yap, "The CLP(ℜ) Language and System", Technical Report 16292 (#72336), IBM Research Division, Yorktown Heights, 1990.15.

Jansen, B., C. Roos and T. Terlaky, *Optimal Bases versus Optimal Partitions for Postoptimal Analysis in Linear Programming*, Technical Report 92-21, Fac. of Techn. Math./Computer Science, Delft University of Technology, revised in May, 1993.

Jeroslow, R.G. and T.H.C. Smith, "Experimental Results on Hillier's Linear Search", *Mathematical Programming*, 9, pp. 371-376, 1975.

Karmarkar, N., "A New Polynomial-Time Algorithm for Linear Programming", *Combinatorica*, pp. 373-395, 1984.

Karmarkar, N., K.G. Ramakrishnan and M.G.C. Resende, "An Interior Point Algorithm for Zero-One Integer Programming", ORSA/TIMS Joint National Meeting, Denver, October 1988.

Khachiyan, L.G., A Polynomial Algorithm in Linear Programming, *Soviet Mathematics Doklady*, 20 (1), p.191-194, 1979.

Kojima, M., S. Mizuno and A. Yoshise, *A Primal-Dual Interior Point Algorithm for Linear Programming*, Technical Report, Tokyo Institute of Technology, February, 1987.

Konopasek, M. and S. Jayaraman, *The TK!Solver Book*, Osborne/McGraw-Hill, Berkeley, Ca. 1984.

Lasdon, L.S., *Optimization Theory for Large Systems*, Macmillan, New York, 1970.

Lassez, J.-L., *Querying Constraints*, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, 1990.

Leler, W., *Constraint programming Languages: Their Specification and Generation*, Addison-Wesley, Reading, Mass., 1988.

Mangasarian, O. L., "Iterative solution of linear programs", *SIAM Journal on Numerical Analysis*, 18, 606-614, 1981.

Marshall, K.T. and J.W. Suurballe, "A note on Cycling in the Simplex Method", Naval Research Logistics Quarterly, 16, pp. 121-137, 1969.

Marsten, R.E. and T.L. Morin, "Parametric Integer Programming: The Right-Hand-Side Case", Annals of Discrete Mathematics, 1, pp. 375-390, 1975.

McShane, K.A., C.L. Monma, and D. Shannon, "An Implementation of a Primal-Dual Interior Point Method for Linear Programming", School of Operatins Research and Industrial Engineering, Cornell University, 1988.

Megiddo, N. and M. Shub, "Boundary Behavior of Interior Point Algorithms in Linear Programming", *Technical Report RJ 5319*, IBM Almaden Research Center, September 1986.

Megiddo, N., "On Finding Primal- and Dual-Optimal bases", *ORSA Journal on Computing*, Vol. 3, No. 1, 1991.

Mehrotra S. and R.D.C. Monteiro, *Parametric and Range Analysis for Interior Point Methods*, Technical Report, Department of Industrial Engineering, University of Arizona, April 1992.

Monteiro, R.C., I. Adler, and M.G.C. Resende, *A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming and its power series extension.* Technical Report, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720, 1988.

Moores, B., "A note on degeneracy in integer linear programming problems", *Journal of the Operational Research Society*, 39 (12), pp. 1175-1178, 1988.

Muli H K., "Optimization Methods in Logic Programming Applied to Expert Systems for Capital Budgeting", Masters of Science thesis, University of Ottawa, 1992.

Piper, C.J. and Zoltners, A.A., "Some easy postoptimality analysis for zero-one programming", Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pa., 1975, (published in *Management Science*).

Rockafellar, R.T., Monotone "Operators and the Proximal Point Algorithm", *SIAM Journal on Control and Optimization*, 14, 5, pp. 877-898, 1976.

Shambin, J.E., and G.T. Stevens Jr. *Operations Research: A Fundamental Approach*, McGraw-Hill, New York: 1974.

Steele, G.J., "The Definition and Implementation of a Computer Programming Language Based on Constraints", Technical Report AI-TR.595, MIT, Cambridge, Mass., 1980.

Sutherland, I.E. "Sketchpad: A Man-Machine Graphical Communication Interface", Ph.D. Thesis, MIT, Cambridge, Mass, 1963.

Tomlin J., private communication, IBM Almaden Research Laboratory, 1993.

Toyoda, Y., "A simplified algorithm for obtaining approximate solutions to zero-one programming problems", Management Sci., 21, pp.1417-1427, 1975.

Tsuchiya T., "Global convergence of the affine scaling methods for degenerate linear programming problems", *Mathematical Programming*, 52, pp.377-404, 1991.

Tsuchiya T., "Global convergence property of the affine scaling methods for primal degenerate linear programming problems", *Mathematics of Operations Research*, 17, (3), pp.527-558, 1992.

Vanderbei, R.J., M.S. Meketon and B.A. Freedman, "A Modification of Karmarkar's Linear Programming Algorithm", *Combinatorica*, 395, 1986.

Vanderbei, R.J., *A Simplified Proof of Dikin's Convergence Result for the Affine-Scalling Algorithm*, Technical Report AT&T Bell Laboratories, 1988.

Vanderbei, R.J., "ALPO: Another Linear Program Optimizer" *ORSA Journal on Computing*, 5 (2), 1993.

Vanderbei, R.J. and T.J. Carpenter, "Symmetric Indefinite Systems for Interior-Point Methods." Technical report SOR-91-7, Dept. of Civil Engineering and Operations Research, Princeton Univ.

Vannelli, A., "An adaptation of the interior point method for solving the global routing problem", IEEE trans on computer-aided design, 10 (2), pp.193-203, 1991

Vassilev V. and K. Genova, "An algorithm of internal feasible directions for linear intger programming", *European Journal of Operational Research*, 52 (2), pp. 203-214, 1991.

Wilkinson, J.H., "Rounding Errors in Algebraic Processes", London: Her Majesty's Stationnery Office, 1963.

Wolfe, P. and L. Cutler, "Experiments in Linear Programming", Recent Advances in Mathematical Programming, R.L. Graves and P. Wolfe (Eds.), McGraw-Hill, New York, pp. 177-200, 1963.

Wolfram, S., "Mathematica : a system for doing mathematics by computer.", Addison-Wesley Pub. Co., Redwood City, Calif., 1991.

Wright, S.J., "Implementing Proximal Point Methods for Linear Programming", Mathematics and Computer Science Division, Argonne National Laboratory, preprint MCS-P45-0189, January 1989.

Appendix A - Detailed calculations of an affine scaling solution

The iterations of an affine scaling algorithm to solve the following problem are shown below.

maximize $x_1 + x_2$

s.t.

$x_1 + x_2 \leq 2$

$x_1,\ x_2 \geq 0$

To cast the problem in the form shown in Section 2.2.2, add the slack variable x_3, and get:

minimize $-x_1 - x_2$

s.t.

$x_1 + x_2 + x_3 = 1$

$x_1,\ x_2, x_3, \geq 0$

An artificial variable x_4 is given a large coefficient. The canonical representation is therefore:

minimize $-x_1 - x_2 + M\ x_4$

s.t.

$x_1 + x_2 + x_3 = 1$

$x_1,\ x_2, x_3, x_4\ \geq 0$

1st iteration

$B = (1/2 \quad 1/2 \quad 1/2 \quad 1/2)$

$$BB^T = (1/2 \quad 1/2 \quad 1/2 \quad 1/2) \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} = 1$$

$(BB^T)^{-1} = 1$

$(BB^T)^{-1}B = (1/2 \quad 1/2 \quad 1/2 \quad 1/2)$

$$B^T(BB^T)^{-1}B = \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} (\ 1/2 \quad 1/2 \quad 1/2 \quad 1/2 \ ) = \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix}$$

$$I - B^T(BB^T)^{-1}B = \begin{bmatrix} 3/4 & -1/4 & -1/4 & -1/4 \\ -1/4 & 3/4 & -1/4 & -1/4 \\ -1/4 & -1/4 & 3/4 & -1/4 \\ -1/4 & -1/4 & -1/4 & 3/4 \end{bmatrix}$$

$$(I - B^T(BB^T)^{-1}B)\,X = \begin{bmatrix} 3/4 & -1/4 & -1/4 & -1/4 \\ -1/4 & 3/4 & -1/4 & -1/4 \\ -1/4 & -1/4 & 3/4 & -1/4 \\ -1/4 & -1/4 & -1/4 & 3/4 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 \end{bmatrix} = \begin{bmatrix} 3/8 & -1/8 & -1/8 & -1/8 \\ -1/8 & 3/8 & -1/8 & -1/8 \\ -1/8 & -1/8 & 3/8 & -1/8 \\ -1/8 & -1/8 & -1/8 & 3/8 \end{bmatrix}$$

$$(I - B^T(BB^T)^{-1}B)\,X\,c^T = \begin{bmatrix} 3/8 & -1/8 & -1/8 & -1/8 \\ -1/8 & 3/8 & -1/8 & -1/8 \\ -1/8 & -1/8 & 3/8 & -1/8 \\ -1/8 & -1/8 & -1/8 & 3/8 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 0 \\ M \end{bmatrix} = \begin{bmatrix} -\frac{1}{4} - \frac{M}{8} \\ -\frac{1}{4} - \frac{M}{8} \\ \frac{1}{4} - \frac{M}{8} \\ \frac{1}{4} + 3\frac{M}{8} \end{bmatrix}$$

$$X\,(I - B^T(BB^T)^{-1}B)\,X\,c^T = \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} -\frac{1}{4} - \frac{M}{8} \\ -\frac{1}{4} - \frac{M}{8} \\ \frac{1}{4} - \frac{M}{8} \\ \frac{1}{4} + 3\frac{M}{8} \end{bmatrix} = \begin{bmatrix} -\frac{1}{8} - \frac{M}{16} \\ -\frac{1}{8} - \frac{M}{16} \\ \frac{1}{8} - \frac{M}{16} \\ \frac{1}{8} + 3\frac{M}{16} \end{bmatrix}$$

To maintain feasibility $\rho = 3M/8$ and the step is $\rho^{-1}X_k\bar{c} = \begin{bmatrix} -1/6 \\ -1/6 \\ -1/6 \\ +1/2 \end{bmatrix}$

$$\text{Thus, } x^1 = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} - \begin{bmatrix} -1/6 \\ -1/6 \\ -1/6 \\ +1/2 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 2/3 \\ 2/3 \\ 0 \end{bmatrix}$$

2nd iteration

$B = (2/3 \quad 2/3 \quad 2/3 \quad 0)$

$$BB^T = (2/3 \quad 2/3 \quad 2/3 \quad 0) \begin{bmatrix} 2/3 \\ 2/3 \\ 2/3 \\ 0 \end{bmatrix} = 4/3$$

$(BB^T)^{-1} = 3/4$

$(BB^T)^{-1}B = 3/4 \quad (2/3 \quad 2/3 \quad 2/3 \quad 0) = (1/2 \quad 1/2 \quad 1/2 \quad 0)$

$$B^T(BB^T)^{-1}B = \begin{bmatrix} 2/1 \\ 2/3 \\ 2/3 \\ 0 \end{bmatrix} (1/2 \quad 1/2 \quad 1/2 \quad 0) = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$I - B^T(BB^T)^{-1}B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2/3 & -1/3 & -1/3 & 0 \\ -1/3 & 2/3 & -1/3 & 0 \\ -1/3 & -1/3 & 2/3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[I - B^T(BB^T)^{-1}B]\ X = \begin{bmatrix} 2/3 & -1/3 & -1/3 & 0 \\ -1/3 & 2/3 & -1/3 & 0 \\ -1/3 & -1/3 & 2/3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2/3 & 0 & 0 & 0 \\ 0 & 2/3 & 0 & 0 \\ 0 & 0 & 2/3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 4/9 & -2/9 & -2/9 & 0 \\ -2/9 & 4/9 & -2/9 & 0 \\ -2/9 & -2/9 & 4/9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[I - B^T(BB^T)^{-1}B]\ X\ c^T = \begin{bmatrix} 4/9 & -2/9 & -2/9 & 0 \\ -2/9 & 4/9 & -2/9 & 0 \\ -2/9 & -2/9 & 4/9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 0 \\ M \end{bmatrix} = \begin{bmatrix} -2/9 \\ -2/9 \\ 4/9 \\ 0 \end{bmatrix}$$

$$X\,[I - B^T(BB^T)^{-1}B]\,X\,c^T = \begin{bmatrix} 2/3 & 0 & 0 & 0 \\ 0 & 2/3 & 0 & 0 \\ 0 & 0 & 2/3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -2/9 \\ -2/9 \\ 4/9 \\ 0 \end{bmatrix} = \begin{bmatrix} -4/27 \\ -4/27 \\ 8/27 \\ 0 \end{bmatrix}$$

$$\rho = \frac{8/27}{2/3} = \frac{4}{9}$$

$$\rho^{-1} = \frac{9}{4}$$

$$\rho^{-1} X_K \bar{C} = \begin{bmatrix} 3/2 & 0 & 0 & 0 \\ 0 & 3/2 & 0 & 0 \\ 0 & 0 & 3/2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -2/9 \\ -2/9 \\ 4/9 \\ 0 \end{bmatrix} = \begin{bmatrix} -1/3 \\ -1/3 \\ 2/3 \\ 0 \end{bmatrix}$$

$$X^{K+1} = X^K - \rho^{-1} X_K \bar{C} = \begin{bmatrix} 2/3 \\ 2/3 \\ 2/3 \\ 0 \end{bmatrix} - \begin{bmatrix} -1/3 \\ -1/3 \\ 2/3 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

3rd iteration

$B = (1\ 1\ 0\ 0)$

$$BB^T = (1\ 1\ 0\ 0)\begin{bmatrix}1\\1\\0\\0\end{bmatrix} = 2$$

$$(BB^T)^{-1} = 1/2$$

$$(BB^T)^{-1}B = 1/2\ \ (1\ 1\ 0\ 0) = (1/2\ \ 1/2\ \ 0\ \ 0)$$

$$B^T(BB^T)^{-1}B = \begin{bmatrix}1\\1\\0\\0\end{bmatrix}(1/2\ \ 1/2\ \ 1/2\ \ 0) = \begin{bmatrix}1/2 & 1/2 & 0 & 0\\1/2 & 1/2 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{bmatrix}$$

$$I - B^T(BB^T)^{-1}B = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix} - \begin{bmatrix}1/2 & 1/2 & 0 & 0\\1/2 & 1/2 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}1/2 & -1/2 & 0 & 0\\-1/2 & 1/2 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 0\end{bmatrix}$$

$$[I - B^T(BB^T)^{-1}B]\,X = \begin{bmatrix} 1/2 & -1/2 & 0 & 0 \\ -1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1/2 & -1/2 & 0 & 0 \\ -1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[I - B^T(BB^T)^{-1}B]\,X\,c^T = \begin{bmatrix} 1/2 & -1/2 & 0 & 0 \\ -1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 0 \\ M \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$X\,[I - B^T(BB^T)^{-1}B]\,X\,c^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\rho = \frac{0}{1} = 0$$

Appendix B - Detailed computational results for the integer heuristic

The following tables represent the complete generation of integer points around the fractional linear programming incumbent solutions, for problems with 5 constraint and 5 variables.

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
0	_	_	_	_
14	_	_	-0.12	-0.03
4	_	_	-0.05	-0.01
4	_	_	-0.04	-0.01
9	_	_	-0.04	0.04
15	0.57	0.57	-0.01	0.00
0	_	_	_	_
2	0.56	0.56	-0.01	0.00
5	0.47	0.47	0.00	0.00
5	0.47	0.47	0.00	0.00
5	0.47	0.47	0.00	0.00
4	0.47	0.47	0.00	0.00
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 1a: A complete search of all integer solutions in Problem 1

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
0	_	_	_	_
12	0.91	0.92	-0.41	-0.26
12	0.91	0.92	-0.39	-0.23
12	0.91	0.92	-0.35	-0.20
59	0.68	0.70	-0.04	0.03
8	0.53	0.54	0.00	0.02
8	0.53	0.54	0.00	0.02
8	0.53	0.54	0.00	0.02
8	0.53	0.54	0.00	0.02
8	0.53	0.54	0.00	0.02
8	0.53	0.54	0.00	0.02
2	0.50	0.51	0.02	0.04
1	0.50	0.50	0.02	0.02
2	0.49	0.49	0.00	0.01
2	0.49	0.49	0.00	0.01
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09

1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
60	0.91	1.01	21.74	-1.69
9	0.91	0.98	-30.34	-4.20
14	0.88	0.95	-1.34	-0.04
10	0.88	0.95	-0.55	0.30
10	0.88	0.95	-0.53	0.30
10	0.88	0.95	-0.51	0.31
13	0.60	0.68	-0.03	0.19
6	0.60	0.69	-0.02	0.20
6	0.60	0.69	-0.02	0.20
6	0.60	0.69	-0.02	0.20
6	0.60	0.69	-0.02	0.20
6	0.60	0.69	-0.02	0.20
3	0.26	0.30	-0.03	0.02
3	0.26	0.30	-0.03	0.02
4	0.19	0.23	0.03	0.08
2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06

2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06
4	0.09	0.14	0.03	0.09
1	0.09	0.09	0.09	0.09
1	0.09	0.09	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09

1	0.66	0.66	0.09	0.09
1	0.66	0.66	0.09	0.09

Table 2a: A complete search of all integer solutions in Problem 2

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
0	_	_	_	_
1	0.17	0.17	0.15	0.15
1	0.17	0.17	0.15	0.15
1	0.17	0.17	0.15	0.15
1	0.10	0.10	0.09	0.09
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 3a: A complete search of all integer solutions in Problem 3

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
0	_	_	_	_
0	_	_	_	_
9	0.71	0.74	0.01	0.09
9	0.70	0.73	0.01	0.11
9	0.70	0.73	0.02	0.11
8	0.70	0.73	0.02	0.11
16	0.69	0.73	0.04	0.15
3	0.54	0.56	0.00	0.02
3	0.54	0.56	0.00	0.02
3	0.54	0.56	0.00	0.02
3	0.54	0.56	0.00	0.02
2	0.54	0.55	0.01	0.02
3	0.00	0.01	0.00	0.01
3	0.00	0.01	0.00	0.01
1	0.00	0.00	0.00	0.00
1				

Table 4a: A complete search of all integer solutions in Problem 4

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
0	_	_	_	_
0	_	_	_	_
2	0.96	0.96	1.08	1.08
0	_	_	_	_
2	0.96	1.32	1.05	0.52
2	0.96	1.32	1.05	0.52
2	0.96	1.32	1.05	0.52
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
4	0.47	0.98	0.47	0.98
1	0.47	0.47	0.47	0.47
1	0.47	0.47	0.47	0.47

Table 5a: A complete search of all integer solutions in Problem 5

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
3	0.45	0.48	0.02	0.07
3	0.45	0.47	0.08	0.11
3	0.45	0.47	0.10	0.12
3	0.45	0.47	0.10	0.12
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
3	0.45	0.47	0.13	0.15
6	0.35	0.43	0.00	0.12
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 6a: A complete search of all integer solutions in Problem 6

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
2	0.24	0.26	0.00	0.02
3	0.24	0.27	0.00	0.04
3	0.24	0.27	0.00	0.04
3	0.15	0.18	0.00	0.03
3	0.15	0.18	0.00	0.04
3	0.13	0.16	0.00	0.04
3	0.13	0.16	0.00	0.04
3	0.13	0.16	0.00	0.04
2	0.13	0.17	0.00	0.05
2	0.13	0.17	0.00	0.05
2	0.13	0.17	0.00	0.05
2	0.13	0.17	0.00	0.05
2	0.13	0.17	0.00	0.05
2	0.13	0.17	0.01	0.05
6	0.04	0.11	0.04	0.11
6	0.04	0.11	0.04	0.11

Table 7a: A complete search of all integer solutions in Problem 7

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
60	0.91	1.01	21.74	-1.69
9	0.91	0.98	-30.34	-4.20
14	0.88	0.95	-1.34	-0.04
10	0.88	0.95	-0.55	0.30
10	0.88	0.95	-0.53	0.30
10	0.88	0.95	-0.51	0.31
13	0.60	0.68	-0.03	0.19
6	0.60	0.69	-0.02	0.20
6	0.60	0.69	-0.02	0.20
6	0.60	0.69	-0.02	0.20
6	0.60	0.69	-0.02	0.20
6	0.60	0.69	-0.02	0.20
3	0.26	0.30	-0.03	0.02
3	0.26	0.30	-0.03	0.02
4	0.19	0.23	0.03	0.08
2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06
2	0.19	0.21	0.03	0.06
4	0.09	0.14	0.03	0.09
1	0.09	0.09	0.09	0.09
1	0.09	0.09	0.09	0.09
1	0.09	0.09	0.09	0.09

Table 8a: A complete search of all integer solutions in Problem 8

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% current Best solution	% current Average solution
15	1.00	1.02	0.98	0.86
13	1.25	1.30	0.35	0.21
40	1.15	1.25	0.56	0.27
30	0.89	0.99	5.17	1.50
16	0.76	0.84	-0.68	-0.09
24	0.19	0.29	-0.08	0.05
14	0.19	0.28	-0.08	0.05
13	0.12	0.24	-0.16	-0.01
3	0.12	0.19	-0.16	-0.08
7	0.09	0.19	-0.21	-0.08
7	0.09	0.19	-0.16	-0.03
4	0.02	0.07	0.02	0.07
2	0.02	0.04	0.02	0.04
2	0.02	0.04	0.02	0.04

Table 9a: A complete search of all integer solutions in Problem 9

The following tables represent the generation of integer points with a probability of 80%. Each table corresponds to a table presented before.

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
16	0.97	0.97	-0.01	0.06
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
20	0.56	0.56	0.00	0.00
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 1b: A random search of integer solutions in Problem 1

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
6	0.69	0.69	0.00	0.00
3	0.54	0.54	0.01	0.01
8	0.54	0.54	0.01	0.01
4	0.54	0.54	0.01	0.01
3	0.54	0.54	0.01	0.01
6	0.54	0.54	-0.01	-0.01
9	0.54	0.54	0.00	0.00
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 2b: A random search of integer solutions in Problem 2

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 3b: A random search of integer solutions in Problem 3

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
0	_	_	_	_
3	0.71	0.71	0.03	0.03
10	0.70	0.71	0.10	0.13
9	0.70	0.71	-0.04	-0.02
14	0.70	0.71	0.00	0.02
4	0.69	0.69	0.06	0.06
15	0.54	0.55	0.02	0.03
13	0.54	0.55	-0.01	0.01
15	0.54	0.55	0.02	0.03
7	0.54	0.55	0.02	0.02
11	0.54	0.54	0.00	0.00
21	0.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00

Table 4b: A random search of integer solutions in Problem 4

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
2	0.26	0.26	0.26	0.26
0	_	_	_	_
1	0.26	0.26	0.61	0.61

Table 5b: A random search of integer solutions in Problem 5

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 6b: A random search of integer solutions in Problem 6

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
0	_	_	_	_
15	0.27	0.27	0.00	0.00
17	0.27	0.27	0.03	0.03
15	0.24	0.27	0.06	0.10
3	0.18	0.18	-0.07	-0.07
6	0.15	0.18	0.06	0.08
7	0.13	0.17	0.00	0.05
6	0.13	0.14	0.00	0.02
6	0.13	0.14	-0.10	-0.08
2	0.13	0.13	0.03	0.03
5	0.13	0.13	0.03	0.03
9	0.13	0.14	0.00	0.02
6	0.13	0.17	0.00	0.05
5	0.13	0.17	0.03	0.08
11	0.13	0.13	0.03	0.04
1	0.10	0.10	0.10	0.10
1	0.07	0.07	0.08	0.08
6	0.04	0.11	0.04	0.11

Table 7b: A random search of integer solutions in Problem 7

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
47	0.97	1.00	-2.29	1.55
0	_	_	_	_
11	0.95	0.95	0.17	0.17
5	0.95	0.95	-0.61	-0.61
3	0.95	0.95	0.57	0.57
1	0.95	0.95	0.63	0.63
4	0.95	0.95	0.31	0.31
7	0.61	0.61	0.00	0.00
7	0.61	0.62	-0.04	-0.02
9	0.61	0.61	-0.01	-0.01
9	0.61	0.61	-0.04	-0.04
5	0.61	0.62	-0.14	-0.12
9	0.61	0.61	0.00	0.00
3	0.29	0.29	-0.02	-0.02
10	0.29	0.29	0.00	0.00
15	0.21	0.21	0.00	0.00
10	0.21	0.21	0.05	0.05
16	0.21	0.21	0.05	0.05
10	0.21	0.21	0.05	0.05
13	0.21	0.21	0.05	0.05
7	0.21	0.21	0.00	0.00
14	0.21	0.21	0.00	0.00
1	0.12	0.12	0.04	0.04
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 8b: A random search of integer solutions in Problem 8

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
2	1.32	1.32	0.28	0.28
8	1.34	1.34	0.16	0.16
5	0.99	0.99	1.17	1.17
0	_	_	_	_
25	0.23	0.23	0.00	0.00
25	0.23	0.23	0.00	0.00
25	0.23	0.23	0.00	0.00
25	0.23	0.23	0.00	0.00
25	0.23	0.23	0.00	0.00
20	0.16	0.20	0.05	0.08
9	0.00	0.00	0.04	0.04
7	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00

Table 9b: A random search of integer solutions in Problem 9

The following tables show similar results for problems with 10 constraints and 10 variables.

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
20	0.25	0.32	0.09	0.19
20	0.25	0.32	0.10	0.19
20	0.25	0.32	0.10	0.19
20	0.25	0.32	0.11	0.20
13	0.13	0.19	0.12	0.18
13	0.13	0.19	0.13	0.18
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 10a: A complete search of integer solutions in Problem 10

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
20	0.25	0.32	0.09	0.19
20	0.25	0.32	0.10	0.19
20	0.25	0.32	0.10	0.19
20	0.25	0.32	0.11	0.20
13	0.13	0.19	0.12	0.18
13	0.13	0.19	0.13	0.18
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
5	0.13	0.16	0.13	0.16
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 11a: A complete search of integer solutions in Problem 11

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
0	_	_	_	_
16	0.74	0.74	0.00	0.00
8	0.74	0.74	-0.08	-0.08
1	0.74	0.74	-0.06	-0.06
0	_	_	_	_
0	_	_	_	_
3	0.67	0.68	0.02	0.05
1	0.67	0.67	-0.09	-0.09
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 10b: A random search of integer solutions in Problem 10

Number of feasible solutions	% optimal Best solution	% optimal Average solution	% optimal Best solution	% optimal Average solution
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
1	0.31	0.31	0.22	0.22
1	0.24	0.24	0.15	0.15
9	0.11	0.11	0.11	0.11
3	0.11	0.11	0.11	0.11
3	0.11	0.11	0.17	0.17
4	0.11	0.11	0.12	0.12
0	_	_	_	_
3	0.11	0.11	0.17	0.17
1	0.11	0.11	0.15	0.15
8	0.11	0.11	0.13	0.13
1	0.11	0.11	0.13	0.13
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_
0	_	_	_	_

Table 11b: A random search of integer solutions in Problem 11

Appendix C - Computer programs

The problem generator (Main1.for) and its accompanying routines (abort.for, randomiz.for and int2str.for):

```
      Program NipGen
C
C  Program generator to be used mainly with
C  NBAFF. Creates mathematical problems in
C  the MPS and NBF (for the NBAFF) format.
C
      External abort, rmarin, ranmar, int2str
      CHARACTER*12 FILE0, file1, const, const2
      character*2 int2str
      CHARACTER type*7, byte
      INTEGER A,B,C,n,m,bcoef,IJ,KL,len,IJ2
      INTEGER ISOPT, ISCOMP, NZOV, ISBND, ITRMAX, IPRINT, IRMAX
      INTEGER I, IPROB, J, MRAN
      REAL AA(100)
      DIMENSION A(40,100), B(100), C(40)
      REAL temp(100), CHCK
      n=0
      m=0
C
C  User input to define the program parameters:
C  variables - constraints
C  filenames (with the extentions)
C  seeds for the random generator (two are needed)
C     seed 1 takes values from 0 - 31328
C     seed 2 takes values from 0 - 30081
C
      PRINT *, "Please input the number of variables (columns) n:"
      Read (5,'(I2)') n
      Print *, "and the number of constraints (rows) m:"
      Read (5,'(I2)') m
      Print *, "The filename and filetype for the MPS:"
      Read (5,'(A12)') file0
      Print *, "The filename and filetype for the NBAFF:"
      Read (5,'(A12)') file1
      Print *, "The random seed (0-31328  ie: 1802):"
      Read (5,'(I4)') IJ2
      Print *, "The other random seed (0-30081  ie: 9373):"
      Read (5,'(I4)') IJ3
```

```
      Print *, "The M:"
      Read (5,'(F4.0)') costa
C
C  Data initialization
C
      mran=0
      ij=ij2
      kl=ij3
      iscomp=0
      nzov=n
      isbnd=-1
      itrmax=9999
      iprint=1
      isign=1
      irmax=100
      isopt=0
c
C  Do the random routine initialization
C
      call rmarin(ij,kl)
C
C  Generate 20000 random numbers
C  to ensure we are well inside the routine
C
      len = 100
      do 11 i = 1, 200
        call RANMAR(temp, len)
 11   continue

      len = 6
      call RANMAR(temp, len)
C
C  Open files and check for dupliacate names
C
   20 format (3f12.4)
      type='new'
   33 open( 12, FILE=file0, status=type, access='sequential',
     +      form='formatted', err=60 )
      type='new'
      goto 34
   60 print *, "Warning: '", file0 ,
     + "' already exists"
      print *, "Overwrite (y/n): "
      Read (5,'(A1)') byte
      if (byte .eq. 'y' .or. byte .eq. 'Y' ) then
```

```
        type='unknown'
        goto 33
      else
        call abort( 'output file already exists' )
      endif
 34   open( 21, FILE=file1, status=type, access='sequential',
     +     form='formatted', err=70 )
      goto 71
 70   print *, "Warning: '", file1,
     +         "'already exists"
      print *, "Overwrite (y/n): "
      byte=' '
      read (5,'(A1)') byte
      if (byte .eq. 'y' .or. byte .eq. 'Y' ) then
        type='unknown'
        goto 34
      else
        call abort ( 'output file already exists' )
      endif
C
C  Generate problem of size m * n
C  (variable IPROB can be used to create multiple problems)
C
 71   len=1
      DO 15 IPROB = 1,1
C
C  Generate A and B matrices
C

   5  DO 4 i = 1,n
C
C  bcoef is essentially a temporary value
C  for the x. There is a 20% chance that this
C  is 0. Otherwise it is an integer between 0
C  and 16.
C
        bcoef=0
        call ranmar(temp,len)
        chck=temp(1)*100
        IF ( chck .GT. 20) THEN
          call ranmar(temp,len)
          bcoef= INT(temp(1)*4.096*4.096)
        ENDIF
        len=2
        A(i,i)= costa
```

```
          DO 104 j = 1,m
            if (i .ne. j) A(i,j)=0.0
            call ranmar(temp,len)
        A(i,j)=A(i,j)+INT(4.096*4.096*temp(1))*(-1)**INT(TEMP(2)*100)
104             b(j)=bcoef*A(i,j)+b(j)
4       CONTINUE
        Do 105 j=1,m
          call ranmar(temp,len)
          b(j)=b(j)+INT(temp(1)*10+temp(2)*10)
105     Continue
C
C  Generate C vector
C
      DO 6 i = 1,n
         call ranmar(temp,len)
         c(i)=INT(100*temp(1))*(-1)**INT(temp(2)*100)
 6    continue
      mran = mran + 1
C
C  Write the two files on the disk
C  starting with the MPS one.
C
 229  WRITE( 12,221 )
 221  FORMAT ('NAME ( MAX )',/, 'ROWS',/,' N   1')
      DO 24 j=1,m
       const='L CONS'//int2str(j)
 24    write (12,31) const
 31   Format (2X,A12)
      WRITE( 12,27 )
 27   FORMAT ('COLUMNS')
      WRITE( 12,29)
 29   FORMAT(4X,'INTEGER1',2X,'''MARKER''',17X,'''INTORG''')
      DO 23 i = 1,n
       IF (c(i) .GT. 0 .OR. c(i) .LT. 0) THEN
         const='X'//int2str(i)
         WRITE( 12,12 ) const,c(i)
 12      FORMAT(4X,A10,' 1',I13)
       ENDIF
      DO 9 j=1,m
       IF (a(i,j) .GT. 0 .or. a(i,j) .lt. 0) THEN
         const='X'//int2str(i)
         const2=' CONS'//int2str(j)
         WRITE( 12,10 ) const,const2,a(i,j)
 10      FORMAT(4X,A10,A8,I7)
 9     ENDIF
```

```
23    CONTINUE
      WRITE( 12,28 )
 28   FORMAT (4X,'INTEGER2',2X,'''MARKER''',17X,'''INTEND''')
      WRITE(12,13)
13    FORMAT('RHS')
      DO 14 j=1,m
        const='CONS'//int2str(j)
        WRITE(12,113) const,b(j)
113     FORMAT(4X,'RHS',8X,A7,I7)
 14   CONTINUE
      WRITE (12,114)
114   FORMAT('ENDATA')
      CLOSE (12)
      write(21,300) n,m,isbnd,itrmax,iprint,isign,irmax,iscomp,nzov,
     1  isopt
 300  format(1015)
      do 305 j=1,m
 305    aa(j)=float(b(j))
      write(21,310) ( aa(j),j=1,m)
      do 320 j=1,m
        do 330 i=1,n
 330      aa(i)=float(a(i,j))
 320    write(21,310) (aa(i), i=1,n)
     do 340 i=1,n
 340    aa(i)=float(c(i))
     write(21,310) (aa(i), i=1,n)
 310  format(15F5.0)
     write(21,350) n
 350  format(I3//)
     close (21)
 15   continue
     END
```

(MAIN1.FOR)

```
      subroutine abort( msg )
C
C  Subroutine to abort to DOS
C  and print an error message
C
      character msg*(*)
      print *, 'Error: ', msg
      stop 4
      end
```

(ABORT.FOR)

```
      subroutine RMARIN(IJ,KL)
C This is the initialization routine for the random number generator RANMAR()
C NOTE: The seed variables can have values between: 0 <= IJ <=31328
C           0 <= KL <=30081
C The random number sequences created by these two seeds are of sufficient
C length to complete an entire calculation with. For example, if several
C different groups are working on different parts of the same calculation,
C each group could be assigned its own IJ seed. This would leave each group
C with 30000 choices for the second seed. That is to say, this random
C number generator can create 900 million different subsequences -- with
C each subsequence having a length of approximately 10^30.
C
C Use IJ = 1802 & KL = 9373 to test the random number generator. The
C subroutine RANMAR should be used to generate 20000 random numbers.
C Then display the next six random numbers generated multiplied by 4096*4096
C If the random number generator is working properly, the random numbers
C should be:
C           6533892.0  14220222.0  7275067.0
C           6172232.0  8354498.0   10633180.0

      real U(97), C, CD, CM , S, T
      integer I97, J97, I, J, K, L, II, JJ, M
      logical*1 TEST
C     data TEST /.FALSE./
      common /raset1/ U, C, CD, CM, I97, J97, TEST

      if( IJ .lt. 0  .or.  IJ .gt. 31328  .or.
     *    KL .lt. 0  .or.  KL .gt. 30081 ) then
          print '(A)', ' The first random number seed must have a value
     *between 0 and 31328'
          print '(A)',' The second seed must have a value between 0 and
     *30081'
          stop
      endif

      i = mod(IJ/177, 177) + 2
      j = mod(IJ    , 177) + 2
      k = mod(KL/169, 178) + 1
      l = mod(KL,     169)

      do 2 ii = 1, 97
        s = 0.0
        t = 0.5
        do 3 jj = 1, 24
          m = mod(mod(i*j, 179)*k, 179)
```

```
            i = j
            j = k
            k = m
            l = mod(53*l+1, 169)
            if (mod(l*m, 64) .ge. 32) then
               s = s + t
            endif
            t = 0.5 * t
3         continue
          U(ii) = s
2       continue

        C = 362436.0 / 16777216.0
        CD = 7654321.0 / 16777216.0
        CM = 16777213.0 /16777216.0
        I97 = 97
        J97 = 33
        TEST = .TRUE.
        return
        end

        subroutine ranmar(RVEC, LEN)
C This is the random number generator proposed by George Marsaglia in
C Florida State University Report: FSU-SCRI-87-50
C It was slightly modified by F. James to produce an array of pseudorandom
C numbers.
        REAL RVEC(*)
        real U(97), C, CD, CM, UNI
        integer I97, J97
        logical*1 TEST
        common /raset1/ U, C, CD, CM, I97, J97, TEST
        integer ivec

C       if( .NOT. TEST ) then
C          print '(A)',' Call the init routine (RMARIN) before calling RAN
C     *MAR'
C          stop
C       endif

        do 100 ivec = 1, LEN
          uni = U(I97) - U(J97)
          if( uni .lt. 0.0 ) uni = uni + 1.0
          U(I97) = uni
          I97 = I97 - 1
          if(I97 .eq. 0) I97 = 97
```

```
      J97 = J97 - 1
      if(J97 .eq. 0) J97 = 97
      C = C - CD
      if( C .lt. 0.0 ) C = C + CM
      uni = uni - C
      if( uni .lt. 0.0 ) uni = uni + 1.0
      RVEC(ivec) = uni
100   continue
      return
      end
```

(RANDOMIZ.FOR)

```
c -- Convert integer to string.
c   Argument should be < 100.
      character function int2str*2(int_number)
      integer int_number
      character*2 result, result1
      result1=char(48+mod(int_number,10))
      if (int_number.gt.9) then
        result=char(48+int(int_number/10)) // result1
        result1=result
      endif
      int2str=result1
      return
      end
```

(INT2STR.FOR)

The problem solver (Main2.for) and its accompanying routines (canon.for, chekup.for, finsol.for, firsts.for, input.for, lpcode.for, numsols.for, output.for, printsls.for, probsols.for, prob2.for, prob2a.for, rprice.for, subs.for and updatx.for)

```
      PROGRAM LINPRO
C
C-----------------------------------------------------------
C   MAIN PROGRAM LINPRO
C
C      LINear  PROgramming code by using modified Karmarkar's LP
C      Algorithm
C      Ref: R.J. Vanderbei, M.C. Meketon and B.A. Freedman
C           Algorithmica (1986) 1:395-407
C
C   AUTHOR:
C           Wun-Hwa Chen
C           School of Management
C           SUNY at Buffalo
C
C   VERSION DATE: January 1989
C-----------------------------------------------------------
C
C  Modified by: C. Karamalis
C               Systems Science
C               Univ. of Ottawa
C  to solve linear problems (integer and non).
C
C Version date: December 1993
C
C=============================================================


      IMPLICIT REAL*8(A-H, O-Z)
      External rmarin, ranmar
C     IMPLICIT REAL*8(A-H, O-Z)
      PARAMETER (LDA=50,LC=80,LN=LDA+LC+1)
      DIMENSION A(LDA,LN),B(LDA),C(LN),BOUND(LN),
     1  X(LN),DX(LN),R(LN),W(LDA),WRK1(LN),WRK2(LN,LDA),WRK3(LDA),
     2  SOL(LC),SLACK(LDA),DUAL(LDA),S(LDA)
      Real temp(100)
      Integer len, i, ij, kl
      COMMON /MU/ XMU
```

```
      READ (5,*) XMU
       ij=1802
       kl=9373
       call rmarin(ij,kl)
       lcn=100
       do 11 i=1,200
11        call ranmar(temp,lcn)
       lcn=6
      CALL INPUT(LDA,A,B,C,S,BOUND,NCONS,NVAR,NUMEQU)
      CALL CANON(LDA,A,B,C,S,NCONS,NVAR,NUMEQU,M,N)
      CALL LPCODE(LDA,LN,A,B,C,M,N,X,DX,R,W,WRK1,WRK2,WRK3,
     x  BOUND,ISTATE,OBJ,ITERS)
      CALL FINSOL(NVAR,NCONS,M,N,X,C,ISTATE,OBJ,SOL,W,SLACK)
      CALL OUTPUT(LDA,NCONS,NVAR,A,B,C,SOL,W,SLACK,ISTATE,OBJ,ITERS,
     1 bound)
      STOP
      END
```

(MAIN2.FOR)

```
C
      SUBROUTINE CANON(LDA,A,B,C,S,NCONS,NVAR,NUMEQU,M,N)
C
C-----------------------------------------------------------------
C  Purpose: This routine transforms the LP problem to cannonical form.
C
C  Inputs :
C  Outputs:
C-----------------------------------------------------------------
C
      IMPLICIT REAL*8(A-H, O-Z)
      DIMENSION A(LDA,1),B(1),C(1),S(1)
C
      INTEGER I
       PRINT *, 'The data are'
       print *, (C(i), i=1,2)
       print *, (A(1,i), i=1,2), '<=' , b(1)
       print *, (A(2,i), i=1,2), '<=' , b(2)
      N = NVAR + NCONS - NUMEQU
      M = NCONS
      N1 = NVAR + 1
      print *, 'N, M, N1 =' , n, m, n1
      DO 10 J=N1,N
         C(J) = 0.
         DO 20 I=1,M
            A(I,J) = 0.
   20    CONTINUE
   10 CONTINUE
      J = NVAR
      DO 30 I = 1, M
         IF( S(I) .EQ. 0. ) GOTO 30
         J = J + 1
         A(I,J) = 1.
   30 CONTINUE
      WRITE(6,*) ' LP IN CANONIAL FORM:  '
      WRITE(6,*) '  M:  ', M, '   N: ', N
      WRITE(6,*) ' A MATRIX: '
      DO 104 I=1,M
  104  WRITE(6,101) (A(I,J),J=1,N)
      WRITE(6,*) '  B VECTOR: '
      WRITE(6,101) (B(I),I=1,M)
      WRITE(6,*) '  C VECTOR: '
      WRITE(6,101) (C(I),I=1,N)
  101  FORMAT(5(1X,F10.2,1X))
      RETURN
```

```
      END
C
                  (CANON.FOR)
```

```
C
      SUBROUTINE CHEKUP(X,DX,N,R,CRIT,GAMMA,IPIVOT,ISTATE,BOUND)
C
C-----------------------------------------------------------------
C  Purpose: This routine checks the stopping rule.
C
C           Stopping Rule: Given epsilon,
C
C              gamma + delta * M <= ( epsilon / N )
C           where:
C              gamma = max_i x_i r_i,
C              delta = - min_i r_i, and
C              M = average (x)
C
C  Inputs :
C  Outputs:
C-----------------------------------------------------------------
C
C
      IMPLICIT REAL*8(A-H, O-Z)
      INTEGER I
      DIMENSION R(1),X(1),DX(1),BOUND(1)

C
      ISTATE = 0
      GAMMA=-1.0D10
      IPIVOT=1
      DELTA=1.0D10
      SMALL = 1.0D-12
      BIGM=0.
      Beta=0.
      DO 20 I=1,N
         CR=R(I)
         CDX=DX(I)
         beta=beta+r(i)*dx(i)
         IF(BOUND(I).LE.0.) GOTO 15
         if (BOUND(I) .NE. X(i)) THEN
            TEMP1 = -1 * CR / ( BOUND(I) - X(I) )
         ELSE
            TEMP1=-.9E+32
         ENDIF
         TEMP2 = CR / X(I)
         CR = DMAX1(TEMP1*CDX,TEMP2*CDX)
```

```
15      IF( CR .GT. GAMMA ) THEN
          GAMMA= CR
          IPIVOT=I
        ENDIF
        IF(I.EQ.N .AND. DX(I) .EQ. 0.0) GO TO 20
          CR = DMIN1(TEMP1,TEMP2)
          IF( CR .LT. DELTA ) DELTA = CR
        BIGM = BIGM + X(I)
20    CONTINUE
      IF( GAMMA .GT. SMALL ) THEN
        BIGM = BIGM / FLOAT(N)
        DELTA = -1. * DELTA
        IF( (GAMMA+DELTA*BIGM) .LE. CRIT) ISTATE = 4
      ENDIF

      RETURN
      END
C
```

(CHEKUP.FOR)

```
C
C
      SUBROUTINE FINSOL(NVAR,NCONS,M,N,X,C,ISTATE,OBJ,SOL,W,SLACK)
      IMPLICIT REAL*8(A-H, O-Z)
      INTEGER I
      DIMENSION C(1),X(1),W(1),SOL(1),SLACK(1)
C
C-----------------------------------------------------------------
C  Purpose: This routine generate the optimal solution from the last
C           iteration x vector
C
C  Inputs :
C  Outputs:
C-----------------------------------------------------------------

      IF (ISTATE .LE. 5) THEN
         OBJ = 0.
         DO 10 I = 1, NVAR
            SOL(I) = X(I)
            OBJ = OBJ + C(I) * SOL(I)
 10      CONTINUE
         DO 50 I = 1, M
            SLACK(I) = X(I+NVAR)
 50      CONTINUE
      ENDIF
      RETURN
      END
                  (FINSOL.FOR)
```

```
C
C
      SUBROUTINE FIRSTS (LDA,A,B,C,M,N,X,BOUND)
C
C-----------------------------------------------------------------
C  Purpose: This routine setup the A,c to generate an initial
C           interior solution x^0 = 1.
C
C           A' = [ A | b - A1 ], c'= [ c | M ],  x = 1
C           i.e.,
C              a_{i,n+1} = b_i - sum_j a_{ij} forall i
C              c_{n+1} = infinity
C              x_j = 1  forall j
C
C  Inputs :
C  Outputs:
C-----------------------------------------------------------------
C
      IMPLICIT REAL*8(A-H, O-Z)
      INTEGER N1, I, J
      REAL SUMA
      DIMENSION A(LDA,1),B(1),C(1),X(1),BOUND(1)
C
C  Introduce an artificial variables with:
C           a_n = b-A1
C           c_n = M
C


      N1 = N + 1
      DO 30 I=1,M
      SUMA=0.
      DO 40 J=1,N
   40 SUMA=SUMA+A(I,J)
      A(I,N1)=2.*B(I)-SUMA
   30 CONTINUE
      N = N1
      C(N)=1.D10
c
c  Assign the initial interior solution
c           X = 1
c
      DO 50 J=1,N
```

```
50  X(J)=0.5
    RETURN
    END
```

(FIRSTS.FOR)

```
      SUBROUTINE INPUT (LDA,A,B,C,S,BOUND,M,N,NUMEQU)
C
C-----------------------------------------------------------
C  Purpose: This routine read in the LP problem in LINPRO format.

C  Inputs :
C  Outputs:
C-----------------------------------------------------------

      IMPLICIT REAL*8(A-H, O-Z)
      DIMENSION A(LDA,1),B(1),C(1),BOUND(1),S(1)
      CHARACTER FNAME*12, byte, type*8
C
C  INPUT THE NECESSARY DATA
C
 1007 WRITE(6,1000)
 1000 FORMAT(' INPUT FILE NAME ?')
      READ(5,'(A12)') FNAME
      OPEN(1,FILE=FNAME,FORM='FORMATTED',STATUS='OLD',ACCESS=
     x'SEQUENTIAL',RECL=80, err=1005)
      goto 1006
 1005 PRINT *, 'Warning! ', FNAME, 'does not exist!'
      Print *, 'Input another name? (y/n)'
      Read (5,'(A1)') byte
      if ( byte .eq. 'y' .or. byte .eq. 'Y') then
         type='unknown'
         goto 1007
      else
         call abort('Input file is missing!')
      endif
 1006 WRITE(6,1001)
 1001 FORMAT('OUTPUT FILE NAME ?')
      READ(5,'(A12)') FNAME
      type='new'
 1002 OPEN(6,FILE=FNAME,status=type,FORM='FORMATTED',
     xACCESS='SEQUENTIAL',RECL=132,err=1003)
      goto 1004
 1003 PRINT *, 'Warning! ', FNAME, 'already exists!'
      Print *, 'Overwrite? (y/n)'
      Read (5,'(A1)') byte
      if ( byte .eq. 'y' .or. byte .eq. 'Y') then
         type='unknown'
         goto 1002
```

```
      else
         call abort('Output file already exists.')
      endif
C
C
C
 1004 MAXM=80
      MAXN=200
C
      READ(1,100) N,M,ISBND,ITRMAX,IPRINT,ISIGN,IRMAX,ISCOMP
  100 FORMAT(10I5)
      IF(N .EQ. 0) RETURN
      IF( ( N .LE. MAXN) .AND. ( M .LE. MAXM)) GO TO 2
      WRITE(6,1) N,MAXN,M,MAXM
    1 FORMAT( '1' /// 20X,'INPUT ERROR  N',I6,'  LIMIT',I6,'  M',I6,
     1  '  LIMIT',I6)
      STOP
    2 READ(1,101) (B(I),I=1,M)
  101 FORMAT(15F5.0)
      IF(ISBND .EQ. 0) GO TO 10
      IF(ISBND .NE. -1) GO TO 9
      ISBND=1
      DO 8 I=1,N+M
    8 BOUND(I)=50000.0
      GO TO 15
    9 READ(1,101) (BOUND(I),I=1,N)
      GO TO 15
   10 DO 12 I=1,N
   12 BOUND(I)=-1.0
   15 IF(ISIGN .EQ. 2) GO TO 18
      SIGN=ISIGN
      DO 17 I=1,M
   17 S(I)=SIGN
      GO TO 19
   18 READ(1,101) (S(I),I=1,M)
   19 IF(ISCOMP .GT. 0) GO TO 40
      DO 20 I=1,M
   20 READ(1,101) (A(I,J),J=1,N)
      GO TO 24
   40 DO 45 I=1,M
      DO 45 J=1,N
   45 A(I,J)=0.0
   21 READ(1,102) I,J,AIJ
  102 FORMAT(I5,I5,F10.0)
      IF(I .GT. 10000) GO TO 24
```

```
      IF((I .LE. M).AND.(J .LE. N)) GO TO 23
      WRITE(6,22) I,M,J,N
   22 FORMAT(// 9X,'INPUT ERROR',9X,'I',I5,'  M',I5,'  J',I5,'  N',I5)
      STOP
   23 A(I,J)=AIJ
      GO TO 21
   24 READ(1,101) (C(I),I=1,N)
      NUMEQU=0
      IF(ISIGN .EQ. 1) GO TO 31
      DO 30 I=1,M
      SI=S(I)
      IF(SI .NE. 0.0) GO TO 25
      NUMEQU=NUMEQU+1
      GO TO 30
   25 IF(SI .EQ. 1.0) GO TO 30
      B(I)=-B(I)
      S(I)=1.0
      DO 26 J=1,N
   26 A(I,J)=-A(I,J)
   30 CONTINUE
   31 CONTINUE
      DO 998 I=1,N
  998 C(I) = - C(I)
      RETURN
      END
```

(INPUT.FOR)

```
C
C*********************************************************************
C  The main modified Karmarkar's algorithm
C*********************************************************************
C
      SUBROUTINE LPCODE(LDA,LN,A,B,C,M,N,X,DX,R,W,WRK1,WRK2,WRK3,
     x  BOUND,ISTATE,OBJ,ITERS)
      IMPLICIT REAL*8(A-H, O-Z)
C       DOUBLE PRECISION BIG, SMALL, EPSIL, CRIT, GAMMA
      INTEGER IPIVOT
      DIMENSION A(LDA,1),B(1),C(1),BOUND(1),X(1),R(1),W(1),DX(1)
      DIMENSION WRK1(1),WRK2(LN,1),WRK3(1)
C
C-----------------------------------------------------------------
C
C           This subroutine will take a canonial form of LP and solve i
C           by the modified Karmarkar's algorithm
C
C  ON ENTRY
C  ON RETURN
C-----------------------------------------------------------------
      LOGICAL FEAS
C
C  Initialize parameters
C
      BIG=1.0D11
      SMALL=1.0D-11
      ITERS=0
      EPSIL=1.0D-3
      FEAS = .FALSE.
C     GAMMA AND IPIVOT were added for conformity
      GAMMA = 0.
      IPIVOT = 0
C     end of additions.
C
C  LP Solver
C
      CALL FIRSTS(LDA,A,B,C,M,N,X,BOUND)
      CRIT = EPSIL / FLOAT(N)
   10 ITERS=ITERS+1

      CALL RPRICE(LDA,LN,A,C,M,N,X,DX,R,W,WRK1,WRK2,WRK3,BOUND)
```

```
      CALL CHEKUP(X,DX,N,R,CRIT,GAMMA,IPIVOT,ISTATE,BOUND)
      IF(ISTATE.NE.0) RETURN
      CALL UPDATX(LDA,N,M,GAMMA,IPIVOT,FEAS,X,DX,R,A,B,C,BOUND)
      GO TO 10
  101 FORMAT(///,1X,20('*'),4X,'ITERATION NO:  ',I5,4X,20('*'),/)
      END
```

(LPCODE.FOR)

```
      subroutine numsols(lda,n,m,x,a,b,c,nums,bound)

      IMPLICIT REAL*8(A-H, O-Z)
C      REAL X2(100), X0, OBJI, SUMINF, SUM
      INTEGER IX(100), I, II, J, I5
      DIMENSION A(LDA,1),C(1),B(1)
      DIMENSION X(1),bound(1)
      REAL temp(100), x2(100)
      integer len

      CHARACTER*10 INTFIL(1)
      COMMON /XO/ X0(100)
      len=2
      print *, 'Nums:---------->', nums
      do 125 i=1,n-m
          print *,'Writing x(i)', x(i)
          Write (intfil,812) x(i)
  812     Format(F10.4)
          print *,'Reading', x(i)
          Read (intfil,813) x2(i)
          print *,'x2(i)=', x2(i)
  813     format(f10.4)
  125   continue
        do 124 i1=1,nums
          do 123 i=1,n-m
            if (x2(i) .eq. IFIX(x2(i))) then
              ix(i)=IFIX(x2(i))
            else
              call ranmar(temp,len)
          ix(i)=IFIX(x2(i)+temp(1)*(x2(i)/10)*((-1)**IFIX(temp(2)*10)))
            If (ix(i) .gt. bound(i)) ix(i)=IFIX(bound(i))
            endif
  123     continue
          obji=0.
  831     DO 832 j=1,n-m
  832       obji=obji+c(j)*ix(j)
          suminf=0.
          do 835 i=1,m
            sum=0.
            DO 836 j=1,n-m
  836         sum=sum+a(i,j)*ix(j)
            SUM=sum-B(I)
            if(sum.LT.0.) SUM=0.
```

```
835       SUMINF=SUMINF+SUM
          WRITE(5,819) (ix(i5), i5=1,n-m)
819       FORMAT(' Integer in inter',10(I5,1X))
          if(SUMINF.GT.0.) WRITE(5,840) SUMINF,OBJI
          if(SUMINF.GT.0.) WRITE(6,840) SUMINF,OBJI
840       FORMAT(' NFEAS, SUM OF INFEASIBILITIES:',2f10.0)
          if(SUMINF.LE.0.) WRITE(5,845) OBJI
830       if(SUMINF.LE.0.) WRITE(6,845) OBJI
845       FORMAT(' ********** FEASIBLE, INTEGER OBJECTIVE:',f10.0)
124       continue
          Return
          END
```

(NUMSOLS.FOR)

```
      SUBROUTINE OUTPUT(LDA,M,N,A,B,C,X,DUAL,SLACK,ISTATE,OBJ,ITERS,
     1 bound)
C
C-----------------------------------------------------------------
C  Purpose: This routine print the optimal solution found.
C
C  Inputs :
C  Outputs:
C-----------------------------------------------------------------
C
      IMPLICIT REAL*8(A-H, O-Z)
C       REAL X0*8
      DIMENSION A(LDA,1),B(1),C(1),X(1),DUAL(1),SLACK(1),bound(1)
      INTEGER IX(100)
      COMMON /XO/ X0(100)
C
  100 FORMAT(/ 1X,'N',I4,'  NCOL',I4,'  M',I4,'  MNOW',I4,
     1  '  ISFEAS',I4,' IRCNT',I4)
  101 FORMAT( // 1X,'OBJ',F15.5,'  ISTATE',I4,'  ITERATIONS',
     1 I4,'  DETERMINANT',F18.5,'  INFEAS',I4,'  NINTO',
     2 I4,'  NOUTOF',I4)
  102 FORMAT( / 1X,'X VECTOR')
  104 FORMAT( / 1X,'SLACK VECTOR')
  105 FORMAT( / 1X,'COST VECTOR')
  106 FORMAT( / 1X,'BOUND VECTOR')
  107 FORMAT( / 1X,'B VECTOR')
  108 FORMAT( / 1X,'SIGN VECTOR')
  109 FORMAT( / 1X,'A MATRIX')
  110 FORMAT( /// 1X,'TABLEAU')
  111 FORMAT( / 1X,'DUAL VECTOR')
  200 FORMAT(10X,10F12.5)
  201 FORMAT(10X,10I12)
  202 FORMAT(100(/ 10X,10I12))
  203 FORMAT(/ 3X,I3,4X,10F12.6,100(/ 10X,10F12.6))
  301 FORMAT( '1' // ' INCOMPLETE SOLUTION')
  302 FORMAT( '1' // ' OPTIMAL SOLUTION')
  303 FORMAT( '1' // ' PROBLEM INFEASIBLE')
  304 FORMAT( '1' // ' UNBOUNDED SOLUTION')
C
C
      IF(ISTATE .LT. 7) GO TO 5
      WRITE(6,301)
      GO TO 10
```

```
  5  IF(ISTATE .EQ. 4) WRITE(6,302)
     IF(ISTATE .EQ. 5) WRITE(6,303)
     IF(ISTATE .EQ. 6) WRITE(6,304)

CC      COmments can be removed to get more data on the file
   10  Print *," "
CC   10  WRITE(6,101) OBJ,ISTATE,ITERS,DETERM,INFEAS,NINTO,NOUTOF
CC      WRITE(6,100) N,NCOL,M,MNOW,ISFEAS,IRCNT
CC      WRITE(6,102)
CC      WRITE(6,200) (X(I),I=1,N)
CC      WRITE(6,104)
CC      WRITE(6,200) (SLACK(I),I=1,M)
CC      WRITE(6,111)
CC      WRITE(6,200) (DUAL(I),I=1,M)
CC      WRITE(6,105)
CC      WRITE(6,200) (C(I),I=1,N)
CW     WRITE(6,106)
CW     WRITE(6,200) (BOUND(I),I=1,N)
CC      WRITE(6,107)
CC      WRITE(6,200) (B(I),I=1,M)
CW     WRITE(6,108)
CW     WRITE(6,200) (S(I),I=1,M)
CW     IF(IPRINT .EQ. 0) RETURN
CW     WRITE(6,109)
CW     DO 25 I=1,M
CW25   WRITE(6,200) (A(I,J),J=1,N)
CW     IF(IPRINT .EQ. 1) RETURN
C
CW30   IIT=0
CW     WRITE(6,110)
CW     WRITE(6,202) IIT,(LABCOL(J),J=1,NCOL)
CW     WRITE(6,203) IIT,OBJ,(TOP(J),J=1,NCOL)
CW     IF(IPRINT .EQ. 2) RETURN
CW     DO 40 I=1,MNOW
CW40   WRITE(6,203) LABROW(I),SOL(I),(TAB(I,J),J=1,NCOL)


      WRITE(6,*) '*********** FINAL NODE *****************'
      ione=1
      call printsls(lda,n+m,m,x,a,b,c,bound)
       return
       end
```

(OUTPUT.FOR)

```
      Subroutine PrintSls(lda,n,m,x,a,b,c,bound)
      IMPLICIT REAL*8(A-H, O-Z)
C           REAL X2(100), X0, OBJI, SUMINF, SUM
      INTEGER IX(200), I, i1, J, I5
      DIMENSION A(LDA,1),C(1),B(1)
      DIMENSION X(1),bound(1)
      REAL temp(100), x2(100)
      integer len

      CHARACTER*10 INTFIL(1)
      COMMON /XO/ X0(100)

      idone=0
      print *, 'Enter the number of random trials in the format:'
      print *, 'Integer: >=1: Select a number of trials'
      print *, 'Real: 0<=x<1: Probability of selecting a solution'
      print *, 'Or  type  -1: Search all possible solutions x-1,x,x+1.'
      print *, 'Or type   -2: Search all x,x+0.d'
      print *, 'Or type   -3: Search depending on the decimal value'
      print *, 'Input the number: '
      Read *, selection
      if (selection .lt. -2) then
         selection=1.0
         call prob2a(lda,n,m,x,a,b,c,selection)
      else
         if (selection .lt. -1) then
            selection=1.0
            call prob2(lda,n,m,x,a,b,c,selection)
         else
            If (selection .lt. 0) then
               selection=1.0
               call probsols(lda,n,m,x,a,b,c,selection)
            else
               if (selection .eq. 0) then
                  return
               else
                  if (selection .lt. 1) then
                     call probsols(lda,n,m,x,a,b,c,selection)
                  else
                     nscl=IFIX(selection)
                     call numsols(lda,n,m,x,a,b,c,nscl,bound)
                  endif
```

```
            endif
         endif
      endif
endif
RETURN
END
```

(PRINTSLS.FOR)

```
      Subroutine ProbSols(LDA,N,M,X,A,B,C,prob)

      IMPLICIT REAL*8(A-H, O-Z)
      DIMENSION A(LDA,1),C(1),B(1)
      DIMENSION R(1),X(1),DX(1)
      DIMENSION IX(100), X2(100)
      Real temp(100), minint
      integer len,sumint
      CHARACTER*10 INTFIL(1)
      COMMON /XO/ X0(100)
      idone=0
      len=1
      aveint=0.0
      sumint=0
      prob=1.0
        if (prob .eq. 0.) goto 820
        do 815 i=1,n-m
        Write (intfil,812) x(i)
812     Format(F10.4)
        Read (intfil,813) x2(i)
813     format(f10.4)
        if (x2(i) .ne. IFIX(x2(i))) then
          ix(i)=IFIX(x2(i))-1
          if (ix(i) .lt. 0) ix(i)=ix(i)+1
        else
          ix(i)=IFIX(x2(i))
815     endif
       minint=99999.9
816    IF (idone .lt. 1) then
         DO 838 i1=0,2
           rand=0.0
           IF (prob .LT. 1.0) THEN
             call ranmar(temp,len)
             rand=temp(1)
           ENDIF
           if (rand .ge. prob) goto 838
           if (x2(1) .eq. IFIX(x2(1))) then
             if (i1 .eq. 0) then
                objj=0.
                goto 831
             else
```

```
              goto 817
            endif
          endif
          ix(1)=ix(1)+i1
          if (ix(1) .gt. IFIX(x2(1))+1) goto 817
          obji=0.
831       DO 832 j=1,n-m
832         obji=obji+c(j)*ix(j)

        suminf=0.
        do 835 i=1,m
          sum=0.
          DO 836 j=1,n-m
836         sum=sum+a(i,j)*ix(j)
          SUM=sum-B(I)
          if(sum.LT.0.) SUM=0.
 835      SUMINF=SUMINF+SUM
 840  FORMAT(' NFEAS, SUM OF INFEASIBILITIES:',2f10.0)
      if(SUMINF.LE.0.05) WRITE(5,846) (ix(i5),ix(i5+1), i5=1,n-m,2)
 830    IF(SUMINF.LE.0.05) THEN
          if (obji .lt. minint) minint=obji
          aveint=aveint+obji
          sumint=sumint+1
        ENDIF
 845  FORMAT(' ****************** FEASIBLE, INTEGER OBJECTIVE:',f10.0)
 846  FORMAT(2I4)
 817  if (x2(1) .ne. IFIX(x2(1))) then
         ix(1)=IFIX(x2(1))-1
         if (ix(1) .lt. 0) ix(1)=ix(1)+1
      endif
 838  continue
      i2=2
      if (x2(i2) .ne. IFIX(x2(i2))) then
          ix(i2)=ix(i2)+1
      endif
 833  itemp=IFIX(x2(i2))
      if (ix(i2) .gt. IFIX(x2(i2))+1 .or. x2(i2) .eq. itemp) then
        IF (x2(i2) .eq. itemp) then
          ix(i2)=IFIX(x2(i2))
        Else
          ix(i2)=IFIX(x2(i2))-1
        Endif
        If (ix(i2) .lt. 0) ix(i2)=ix(i2)+1
        i2=i2+1
        if (x2(i2) .ne. IFIX(x2(i2))) ix(i2)=ix(i2)+1
```

```
      if (i2 .le. n-m) goto 833
    Endif
    if (i2 .gt. n-m) idone=1
    if (idone .NE. 1) goto 816
    Endif
    If (sumint .lt. 1) then
       aveint=minint
    else
       aveint=aveint/sumint
    endif
    obji= 0
    do 999 j= 1, n-m
999     obji=obji+c(j)*x(j)
    Write(6,849) obji, sumint, minint, aveint
849  FORMAT(' O', F9.2,' n ',i4,' m ', f10.2,' A ',F10.2)
847  FORMAT(' Minimum integer for this step:',f10.3)
848  FORMAT(' Average integer for this step:',f10.3)
820  RETURN
    END
```

(PROBSOLS.FOR)

```
      Subroutine Prob2(LDA,N,M,X,A,B,C,prob)
C
C     Mainly for the grid search
C
      IMPLICIT REAL*8(A-H, O-Z)
      DIMENSION A(LDA,1),C(1),B(1)
      DIMENSION R(1),X(1),DX(1)
      DIMENSION X2(100)
      Real temp(100)
      integer len
      CHARACTER*10 INTFIL(1)
      COMMON /XO/ X0(100)
      sumint=0.0
      os=0.0
      idone=0
      rand=0.0
      len=1
      Print *,'Please input the grid length '
      READ *,delta
      delta=delta/10.0
      prob=1.0
      print *, 'Probability is: (SELECTION)', prob
      print *, 'n m:' ,n,m
      if (prob .eq. 0.) goto 820
      do 815 i=1,n-m
         x2(i)=(IFIX(x(i)/delta))*delta
         os=os+c(i)*x(i)
  815 continue
  816 IF (idone .lt. 1) then
        DO 838 i1=0,1
          if (prob .lt. 1.0) then
            call ranmar(temp,len)
            rand=temp(1)
          endif
          if (rand .ge. prob) goto 838
          x2(1)=x2(1)+i1*delta
C          x2(1)=IFIX(x2(1)/delta)*delta
          if (x2(1) .gt. ((IFIX(x(1)/delta)+1.1)*delta)) goto 817
          obji=0.
  831     DO 832 j=1,n-m
  832       obji=obji+c(j)*x2(j)
        suminf=0.
        do 835 i=1,m
```

```
          sum=0.
          DO 836 j=1,n-m
836          sum=sum+a(i,j)*x2(j)
          SUM=sum-B(I)
          if(sum.LT.0.) SUM=0.
835        SUMINF=SUMINF+SUM
        if(SUMINF.LE.0.05) WRITE(5,845) OBJI
830   IF(SUMINF.LE.0.05) sumint=sumint+1.0
845   FORMAT(' ****************** FEASIBLE, INTEGER OBJECTIVE:',f10.0)
846   FORMAT(30F10.4)
817   x2(1)=IFIX(x(1)/delta)*delta
838   continue
      i2=2
      x2(i2)=x2(i2)+delta
833   if (x2(i2) .gt. (IFIX(x(i2)/delta)+1.1)*delta) then
        x2(i2)=IFIX(x(i2)/delta)*delta
        i2=i2+1
        x2(i2)=x2(i2)+delta
        if (i2 .le. n-m) goto 833
      Endif
      if (i2 .gt. n-m) idone=1
      if (idone .NE. 1) goto 816
      Endif
820   WRITE(6,847) os, (x(i), i=1,n), sumint
847   FORMAT(1X,F10.4,1X,25F7.3,'=Number of int feas sols:',F6.0)
      RETURN
      END
```

(PROB2.FOR)

```
      Subroutine Prob2A(LDA,N,M,X,A,B,C,prob)
C
C     Mainly for the search depending on the decimal value
C
      IMPLICIT REAL*8(A-H, O-Z)
      DIMENSION A(LDA,1),C(1),B(1)
      DIMENSION R(1),X(1),DX(1)
      DIMENSION X2(100), X3(100)
      Real temp(100), minint
      integer len, sumint
      CHARACTER*10 INTFIL(1)
      COMMON /XO/ X0(100)
      sumint=0
      aveint=0.0
      os=0.0
      idone=0
      len=1
      Print *,'Please input the prob: '
      read *,prob
      num=prob*(2.0**(n-m))
      print *,num
      delta=10.0
      delta=delta/10.0
      if (prob .eq. 0.) return
      do 815 i=1,n-m
         os=os+c(i)*x(i)
  815 continue
      minint=99999.9
        DO 838 i1=1,num
          obji=0.
  831     DO 832 j=1,n-m
           x3(j)=x(j)-IFIX(x(j))
          call ranmar(temp,len)
          rand=temp(1)
          if (rand .ge. (1-x3(j))) then
            x2(j)=IFIX(x(j))+1
          else
            x2(j)=IFIX(x(j))
          endif
  832       obji=obji+c(j)*x2(j)
        suminf=0.
        do 835 i=1,m
          sum=0.
```

```
        DO 836 j=1,n-m
836        sum=sum+a(i,j)*x2(j)
        SUM=sum-B(I)
        if(sum.LT.0.) SUM=0.
835      SUMINF=SUMINF+SUM
       IF(SUMINF.LE.0.05) THEN
         WRITE(5,845) OBJI
         IF (objj .lt. minint) minint=objj
         aveint=aveint+objj
         sumint=sumint+1
       ENDIF
830  IF(SUMINF.LE.0.05) WRITE (6,845) objj
845  FORMAT(' ********** FEASIBLE, INTEGER OBJECTIVE:',f10.0)
838  continue
878  FORMAT(1X,'Mean of values produced:', 30F10.4)
     if (sumint .lt. 1) then
       aveint=minint
     else
       aveint=aveint/sumint
     ENDIF
     WRITE(6,849) objj, sumint, minint, aveint
849  FORMAT(' O', F9.2,' n ',i4,' m ',f10.2,' A ', f10.2)
     RETURN
     END
```

(PROB2A.FOR)

```
C
      SUBROUTINE RPRICE(LDA,LN,A,C,M,N,X,DX,R,W,Y,BT,QRAUX,BOUND)
C
C-----------------------------------------------------------------
C  Purpose: This routine calculate the reduced price vector c_p by usin
C           the orthogonal decomposition (QR decomposition) and
C           Householder transformation to solve the least square proble
C           which is formed by the following matrix operations:
C                              T        T
C              Y = D c,   X = B  = ( A D )
C
C           Namely,
C
C            Y = [ c1 x1, c2 x2, ..., cn xn ]
C                  _                      _
C                 |  a11 x1, . . . am1 x1  |
C                 |  a12 x2,       am2 x2  |
C            X =  |    .     .             |
C                 |    .       .           |
C                 |    .         .         |
C                 |_ a1n xn,       amn xn _|
C  Inputs :
C  Outputs:
C-----------------------------------------------------------------
C
      IMPLICIT REAL*8(A-H, O-Z)
      DIMENSION A(LDA,1),C(1),X(1),DX(1),Y(1),BT(LN,1),R(1),W(1),
     * BOUND(1),QRAUX(1)
      COMMON /MU/ XMU
C
      DO 10 I=1,N
      DX(I)=X(I)
      IF(BOUND(I).LE.0.) GO TO 10
      DX(I)=DMIN1(X(I),BOUND(I)-X(I))
   10 CONTINUE
      Print *, XMU
C
C  Setup the X,Y matrix for the least square problem
C
      DO 20 I=1,N

      Y(I)=(C(I)-XMU/X(I))*DX(I)
```

```
C       Y(I)=C(I)*DX(I)
 20  CONTINUE
     DO 40 J=1,N
     TEMP=DX(J)
     DO 30 I=1,M
     BT(J,I)=A(I,J)*TEMP
 30  CONTINUE
 40  CONTINUE
     CALL DQRDC(BT,LN,N,M,QRAUX,K)
     CALL DQRSL(BT,LN,Y,N,K,QRAUX,Y,W,R,INFO)

CC   REMOVE THE CCs TO GET BACK TO ORIGINAL OUTPUT

CC      WRITE(6,102)
CC      WRITE(6,101) (W(I),I=1,M)
CC     WRITE(6,103)
 101  FORMAT(1X,5(D12.4,1X))
 102  FORMAT(/,' DUAL VECTOR : W(I)',/)
 103  FORMAT(/,' REDUCED PRICE VECTOR : R(I)',/)
     RETURN
     END
C
```

(RPRICE.FOR)

```
C
C-----------------------------------------------------------
C***********************************************************************
C
C       SUBROUTINES TO CALCULATE THE REDUCED PRICE VECTOR:
C   QR DECOMPOSITION WITH HOUSEHOLDER : DQRDC, DQRSL, (BLAS SET)
C
C***********************************************************************
C-----------------------------------------------------------
C
C
C***********************************************************************
C   TO CALCULATE QR DECOMPOSITION BY HOUSEHOLDER TRANSFORMATION
C***********************************************************************
C
      SUBROUTINE DQRDC(X,LDX,N,P,QRAUX,LUP)
      INTEGER LDX,N,P
      DOUBLE PRECISION X(LDX,1),QRAUX(1)
C
C     DQRDC USES HOUSEHOLDER TRANSFORMATIONS TO COMPUTE THE QR
C     FACTORIZATION OF AN N BY P MATRIX X.  COLUMN PIVOTING
C     BASED ON THE 2-NORMS OF THE REDUCED COLUMNS MAY BE
C     PERFORMED AT THE USERS OPTION.
C
C     ON ENTRY
C
C        X      DOUBLE PRECISION(LDX,P), WHERE LDX .GE. N.
C               X CONTAINS THE MATRIX WHOSE DECOMPOSITION IS TO BE
C               COMPUTED.
C
C        LDX    INTEGER.
C               LDX IS THE LEADING DIMENSION OF THE ARRAY X.
C
C        N      INTEGER.
C               N IS THE NUMBER OF ROWS OF THE MATRIX X.
C
C        P      INTEGER.
C               P IS THE NUMBER OF COLUMNS OF THE MATRIX X.
C
C     ON RETURN
C
C        X      X CONTAINS IN ITS UPPER TRIANGLE THE UPPER
C               TRIANGULAR MATRIX R OF THE QR FACTORIZATION.
```

```
C           BELOW ITS DIAGONAL X CONTAINS INFORMATION FROM
C           WHICH THE ORTHOGONAL PART OF THE DECOMPOSITION
C           CAN BE RECOVERED.  NOTE THAT IF PIVOTING HAS
C           BEEN REQUESTED, THE DECOMPOSITION IS NOT THAT
C           OF THE ORIGINAL MATRIX X BUT THAT OF X
C           WITH ITS COLUMNS PERMUTED AS DESCRIBED BY JPVT.
C
C     QRAUX   DOUBLE PRECISION(P).
C           QRAUX CONTAINS FURTHER INFORMATION REQUIRED TO RECOVER
C           THE ORTHOGONAL PART OF THE DECOMPOSITION.
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     G.W. STEWART, UNIVERSITY OF MARYLAND, ARGONNE NATIONAL LAB.
C
C     DQRDC USES THE FOLLOWING FUNCTIONS AND SUBPROGRAMS.
C
C     BLAS DAXPY,DDOT,DSCAL,DNRM2
C     FORTRAN DABS,DMAX1,MIN0,DSQRT
C
C     INTERNAL VARIABLES
C
      INTEGER J,JP,L,LP1,LUP,MAXJ
      DOUBLE PRECISION MAXNRM,DNRM2,TT
      DOUBLE PRECISION DDOT,NRMXL,T
C
C
C     PERFORM THE HOUSEHOLDER REDUCTION OF X.
C
      LUP = MIN0(N,P)
      DO 200 L = 1, LUP
         QRAUX(L) = 0.0D0
         IF (L .EQ. N) GO TO 190
C
C           COMPUTE THE HOUSEHOLDER TRANSFORMATION FOR COLUMN L.
C
            NRMXL = DNRM2(N-L+1,X(L,L),1)
            IF (NRMXL .EQ. 0.0D0) GO TO 180
               IF (X(L,L) .NE. 0.0D0) NRMXL = DSIGN(NRMXL,X(L,L))
               CALL DSCAL(N-L+1,1.0D0/NRMXL,X(L,L),1)
               X(L,L) = 1.0D0 + X(L,L)
C
C              APPLY THE TRANSFORMATION TO THE REMAINING COLUMNS,
C              UPDATING THE NORMS.
C
               LP1 = L + 1
```

```
            IF (P .LT. LP1) GO TO 170
            DO 160 J = LP1, P
               T = -DDOT(N-L+1,X(L,L),1,X(L,J),1)/X(L,L)
               CALL DAXPY(N-L+1,T,X(L,L),1,X(L,J),1)
  160       CONTINUE
  170       CONTINUE
C
C           SAVE THE TRANSFORMATION.
C
            QRAUX(L) = X(L,L)
            X(L,L) = -NRMXL
  180     CONTINUE
  190   CONTINUE
  200 CONTINUE
      RETURN
      END
C
C
C*********************************************************************
C  SUBROUTINE TO SOLVE THE LEAST SQUARE PROBLEM
C*********************************************************************
C
      SUBROUTINE DQRSL(X,LDX,Y,N,K,QRAUX,QTY,B,RSD,INFO)
      INTEGER LDX,N,K,INFO
      DOUBLE PRECISION X(LDX,1),QRAUX(1),Y(1),QTY(1),B(1),RSD(1)
C
C     DQRSL APPLIES THE OUTPUT OF DQRDC TO COMPUTE COORDINATE
C     TRANSFORMATIONS, PROJECTIONS, AND LEAST SQUARES SOLUTIONS.
C     FOR K .LE. MIN(N,P), LET XK BE THE MATRIX
C
C           XK = (X(JPVT(1)),X(JPVT(2)), ... ,X(JPVT(K)))
C
C     FORMED FROM COLUMNNS JPVT(1), ... ,JPVT(K) OF THE ORIGINAL
C     N X P MATRIX X THAT WAS INPUT TO DQRDC (IF NO PIVOTING WAS
C     DONE, XK CONSISTS OF THE FIRST K COLUMNS OF X IN THEIR
C     ORIGINAL ORDER).  DQRDC PRODUCES A FACTORED ORTHOGONAL MATRIX
Q
C     AND AN UPPER TRIANGULAR MATRIX R SUCH THAT
C
C              XK = Q * (R)
C                       (0)
C
C     THIS INFORMATION IS CONTAINED IN CODED FORM IN THE ARRAYS
C     X AND QRAUX.
C
```

```
C     ON ENTRY
C
C        X      DOUBLE PRECISION(LDX,P).
C               X CONTAINS THE OUTPUT OF DQRDC.
C
C        LDX    INTEGER.
C               LDX IS THE LEADING DIMENSION OF THE ARRAY X.
C
C        N      INTEGER.
C               N IS THE NUMBER OF ROWS OF THE MATRIX XK.  IT MUST
C               HAVE THE SAME VALUE AS N IN DQRDC.
C
C        K      INTEGER.
C               K IS THE NUMBER OF COLUMNS OF THE MATRIX XK.  K
C               MUST NOT BE GREATER THAN MIN(N,P), WHERE P IS THE
C               SAME AS IN THE CALLING SEQUENCE TO DQRDC.
C
C        QRAUX  DOUBLE PRECISION(P).
C               QRAUX CONTAINS THE AUXILIARY OUTPUT FROM DQRDC.
C
C        Y      DOUBLE PRECISION(N)
C               Y CONTAINS AN N-VECTOR THAT IS TO BE MANIPULATED
C               BY DQRSL.
C
C     ON RETURN
C
C        QTY    DOUBLE PRECISION(N).
C               QTY CONTAINS TRANS(Q)*Y, IF ITS COMPUTATION HAS
C               BEEN REQUESTED.  HERE TRANS(Q) IS THE
C               TRANSPOSE OF THE MATRIX Q.
C
C        B      DOUBLE PRECISION(K)
C               B CONTAINS THE SOLUTION OF THE LEAST SQUARES PROBLEM
C
C                  MINIMIZE NORM2(Y - XK*B),
C
C               IF ITS COMPUTATION HAS BEEN REQUESTED.  (NOTE THAT
C               IF PIVOTING WAS REQUESTED IN DQRDC, THE J-TH
C               COMPONENT OF B WILL BE ASSOCIATED WITH COLUMN JPVT(J)
C               OF THE ORIGINAL MATRIX X THAT WAS INPUT INTO DQRDC.)
C
C        RSD    DOUBLE PRECISION(N).
C               RSD CONTAINS THE LEAST SQUARES RESIDUAL Y - XK*B,
C               IF ITS COMPUTATION HAS BEEN REQUESTED.  RSD IS
C               ALSO THE ORTHOGONAL PROJECTION OF Y ONTO THE
```

```
C           ORTHOGONAL COMPLEMENT OF THE COLUMN SPACE OF XK.
C
C       INFO  INTEGER.
C             INFO IS ZERO UNLESS THE COMPUTATION OF B HAS
C             BEEN REQUESTED AND R IS EXACTLY SINGULAR.  IN
C             THIS CASE, INFO IS THE INDEX OF THE FIRST ZERO
C             DIAGONAL ELEMENT OF R AND B IS LEFT UNALTERED.
C
C
C     CALL DQRSL(X,LDX,N,K,QRAUX,Y,DUM,Y,B,Y,DUM,110,INFO)
C
C     WILL RESULT IN THE COMPUTATION OF B AND RSD, WITH RSD
C     OVERWRITING Y.
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     G.W. STEWART, UNIVERSITY OF MARYLAND, ARGONNE NATIONAL LAB.
C
C     DQRSL USES THE FOLLOWING FUNCTIONS AND SUBPROGRAMS.
C
C     BLAS DAXPY,DCOPY,DDOT
C     FORTRAN DABS,MIN0,MOD
C
C     INTERNAL VARIABLES
C
      INTEGER I,J,JJ,JU,KP1
      DOUBLE PRECISION DDOT,T,TEMP
      LOGICAL CB
C
C
C     SET INFO FLAG.
C
      CB = .TRUE.
      INFO = 0
      JU = MIN0(K,N-1)
C
C     SPECIAL ACTION WHEN N=1.
C
      IF (JU .NE. 0) GO TO 40
         IF (.NOT.CB) GO TO 30
            IF (X(1,1) .EQ. 0.0D0) THEN
               INFO = 1
            ELSE
               B(1) = Y(1)/X(1,1)
            ENDIF
   30    CONTINUE
```

```
      RSD(1) = 0.0D0
    GO TO 250
 40 CONTINUE
C
C       SET UP TO COMPUTE QTY.
C
        CALL DCOPY(N,Y,1,QTY,1)
C
C         COMPUTE TRANS(Q)*Y.
C
          DO 90 J = 1, JU
            IF (QRAUX(J) .EQ. 0.0D0) GO TO 80
               TEMP = X(J,J)
               X(J,J) = QRAUX(J)
               T = -DDOT(N-J+1,X(J,J),1,QTY(J),1)/X(J,J)
               CALL DAXPY(N-J+1,T,X(J,J),1,QTY(J),1)
               X(J,J) = TEMP
 80         CONTINUE
 90       CONTINUE
100     CONTINUE
C
C       SET UP TO COMPUTE B.
C
        IF (.NOT.CB) GO TO 200
          CALL DCOPY(K,QTY,1,B,1)
C
C         COMPUTE B.
C
          DO 170 JJ = 1, K
            J = K - JJ + 1
            IF (X(J,J) .NE. 0.0D0) GO TO 150
               INFO = J
C           .....EXIT
               GO TO 180
150         CONTINUE
            B(J) = B(J)/X(J,J)
C           IF (B(J) .LT. 0.1D-10 .AND. B(J) .GT. -0.1D-10) B(J)=0.0
C ABOVE LINE WAS ADDED ********************************************
            IF (J .EQ. 1) GO TO 160
               T = -B(J)
               CALL DAXPY(J-1,T,X(1,J),1,B,1)
160         CONTINUE
170       CONTINUE
180       CONTINUE
200     CONTINUE
```

```
C
C       SET UP TO COMPUTE RSD.
C
        KP1 = K + 1
        IF (K .LT. N) CALL DCOPY(N-K,QTY(KP1),1,RSD(KP1),1)
        DO 210 I = 1, K
          RSD(I) = 0.0D0
  210   CONTINUE
C
C         COMPUTE RSD.
C
        DO 230 JJ = 1, JU
          J = JU - JJ + 1
          IF (QRAUX(J) .EQ. 0.0D0) GO TO 230
            TEMP = X(J,J)
            X(J,J) = QRAUX(J)
            T = -DDOT(N-J+1,X(J,J),1,RSD(J),1)/X(J,J)
            CALL DAXPY(N-J+1,T,X(J,J),1,RSD(J),1)
            X(J,J) = TEMP
  230   CONTINUE
  250 CONTINUE
      RETURN
      END
C
C*********************************************************************
C           BLAS FUNCTIONS AND SUBROUTINES
C*********************************************************************
C
      SUBROUTINE  DSCAL(N,DA,DX,INCX)
C
C     SCALES A VECTOR BY A CONSTANT.
C     USES UNROLLED LOOPS FOR INCREMENT EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION DA,DX(1)
      INTEGER I,INCX,M,MP1,N,NINCX
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GO TO 20
C
C       CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        DX(I) = DA*DX(I)
```

```
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DX(I) = DA*DX(I)
   30 CONTINUE
      IF( N .LT. 5 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
        DX(I) = DA*DX(I)
        DX(I + 1) = DA*DX(I + 1)
        DX(I + 2) = DA*DX(I + 2)
        DX(I + 3) = DA*DX(I + 3)
        DX(I + 4) = DA*DX(I + 4)
   50 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE DAXPY(N,DA,DX,INCX,DY,INCY)
C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION DX(1),DY(1),DA
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF (DA .EQ. 0.0D0) RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
```

```
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DY(IY) = DY(IY) + DA*DX(IX)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,4)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DY(I) = DY(I) + DA*DX(I)
   30 CONTINUE
      IF( N .LT. 4 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,4
        DY(I) = DY(I) + DA*DX(I)
        DY(I + 1) = DY(I + 1) + DA*DX(I + 1)
        DY(I + 2) = DY(I + 2) + DA*DX(I + 2)
        DY(I + 3) = DY(I + 3) + DA*DX(I + 3)
   50 CONTINUE
      RETURN
      END
C
C
      DOUBLE PRECISION FUNCTION DDOT(N,DX,INCX,DY,INCY)
C
C     FORMS THE DOT PRODUCT OF TWO VECTORS.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION DX(1),DY(1),DTEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      DDOT = 0.0D0
      DTEMP = 0.0D0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
```

```
C        NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DTEMP = DTEMP + DX(IX)*DY(IY)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      DDOT = DTEMP
      RETURN
C
C       CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C       CLEAN-UP LOOP
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DTEMP = DTEMP + DX(I)*DY(I)
   30 CONTINUE
      IF( N .LT. 5 ) GO TO 60
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
        DTEMP = DTEMP + DX(I)*DY(I) + DX(I + 1)*DY(I + 1) +
     1  DX(I + 2)*DY(I + 2) + DX(I + 3)*DY(I + 3) + DX(I + 4)*DY(I + 4)
   50 CONTINUE
   60 DDOT = DTEMP
      RETURN
      END
C
C
      DOUBLE PRECISION FUNCTION DNRM2 ( N, DX, INCX)
      INTEGER          NEXT
      DOUBLE PRECISION   DX(1), CUTLO, CUTHI, HITEST, SUM, XMAX,ZERO,ON
      DATA   ZERO, ONE /0.0D0, 1.0D0/
C
C     EUCLIDEAN NORM OF THE N-VECTOR STORED IN DX() WITH STORAGE
C     INCREMENT INCX .
C     IF    N .LE. 0 RETURN WITH RESULT = 0.
C     IF N .GE. 1 THEN INCX MUST BE .GE. 1
C
```

```
C         C.L.LAWSON, 1978 JAN 08
C
C    FOUR PHASE METHOD     USING TWO BUILT-IN CONSTANTS THAT ARE
C    HOPEFULLY APPLICABLE TO ALL MACHINES.
C       CUTLO = MAXIMUM OF  DSQRT(U/EPS)  OVER ALL KNOWN MACHINES.
C       CUTHI = MINIMUM OF  DSQRT(V)      OVER ALL KNOWN MACHINES.
C    WHERE
C       EPS = SMALLEST NO. SUCH THAT EPS + 1. .GT. 1.
C       U   = SMALLEST POSITIVE NO.   (UNDERFLOW LIMIT)
C       V   = LARGEST  NO.            (OVERFLOW  LIMIT)
C
C    BRIEF OUTLINE OF ALGORITHM..
C
C    PHASE 1    SCANS ZERO COMPONENTS.
C    MOVE TO PHASE 2 WHEN A COMPONENT IS NONZERO AND .LE. CUTLO
C    MOVE TO PHASE 3 WHEN A COMPONENT IS .GT. CUTLO
C    MOVE TO PHASE 4 WHEN A COMPONENT IS .GE. CUTHI/M
C    WHERE M = N FOR X() REAL  AND M = 2*N FOR COMPLEX.
C
C    VALUES FOR CUTLO AND CUTHI..
C   FROM THE ENVIRONMENTAL PARAMETERS LISTED IN THE IMSL CONVERTER
C    DOCUMENT THE LIMITING VALUES ARE AS FOLLOWS..
C    CUTLO, S.P.   U/EPS = 2**(-102) FOR  HONEYWELL.  CLOSE SECONDS AR
C                  UNIVAC AND DEC AT 2**(-103)
C                  THUS CUTLO = 2**(-51) = 4.44089E-16
C    CUTHI, S.P.   V = 2**127 FOR UNIVAC, HONEYWELL, AND DEC.
C                  THUS CUTHI = 2**(63.5) = 1.30438E19
C    CUTLO, D.P.   U/EPS = 2**(-67) FOR HONEYWELL AND DEC.
C                  THUS CUTLO = 2**(-33.5) = 8.23181D-11
C    CUTHI, D.P.   SAME AS S.P.  CUTHI = 1.30438D19
C    DATA CUTLO, CUTHI / 8.232D-11,  1.304D19 /
C    DATA CUTLO, CUTHI / 4.441E-16,  1.304E19 /
     DATA CUTLO, CUTHI / 8.232D-11,  1.304D19 /
C
     IF(N .GT. 0) GO TO 10
        DNRM2  = ZERO
        GO TO 300
C
  10 ASSIGN 30 TO NEXT
     SUM = ZERO
     NN = N * INCX
C                                    BEGIN MAIN LOOP
     I = 1
  20    GO TO NEXT,(30, 50, 70, 110)
  30 IF( DABS(DX(I)) .GT. CUTLO) GO TO 85
```

```
      ASSIGN 50 TO NEXT
      XMAX = ZERO
C
C                     PHASE 1.  SUM IS ZERO
C
   50 IF( DX(I) .EQ. ZERO) GO TO 200
      IF( DABS(DX(I)) .GT. CUTLO) GO TO 85
C
C                           PREPARE FOR PHASE 2.
      ASSIGN 70 TO NEXT
      GO TO 105
C
C                           PREPARE FOR PHASE 4.
C
  100 I = J
      ASSIGN 110 TO NEXT
      SUM = (SUM / DX(I)) / DX(I)
  105 XMAX = DABS(DX(I))
      GO TO 115
C
C                 PHASE 2.  SUM IS SMALL.
C                       SCALE TO AVOID DESTRUCTIVE UNDERFLOW.
C
   70 IF( DABS(DX(I)) .GT. CUTLO ) GO TO 75
C
C                   COMMON CODE FOR PHASES 2 AND 4.
C                   IN PHASE 4 SUM IS LARGE.  SCALE TO AVOID OVERFLOW
C
  110 IF( DABS(DX(I)) .LE. XMAX ) GO TO 115
          SUM = ONE + SUM * (XMAX / DX(I))**2
          XMAX = DABS(DX(I))
          GO TO 200
C
  115 SUM = SUM + (DX(I)/XMAX)**2
      GO TO 200
C
C
C                 PREPARE FOR PHASE 3.
C
   75 SUM = (SUM * XMAX) * XMAX
C
C
C     FOR REAL OR D.P. SET HITEST = CUTHI/N
C     FOR COMPLEX      SET HITEST = CUTHI/(2*N)
C
```

```
  85 HITEST = CUTHI/FLOAT( N )
C
C                 PHASE 3.  SUM IS MID-RANGE.  NO SCALING.
C
      DO 95 J =I,NN,INCX
      IF(DABS(DX(J)) .GE. HITEST) GO TO 100
  95    SUM = SUM + DX(J)**2
      DNRM2 = DSQRT( SUM )
      GO TO 300
C
 200 CONTINUE
      I = I + INCX
      IF ( I .LE. NN ) GO TO 20
C
C             END OF MAIN LOOP.
C
C             COMPUTE SQUARE ROOT AND ADJUST FOR SCALING.
C
      DNRM2 = XMAX * DSQRT(SUM)
 300 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE  DCOPY(N,DX,INCX,DY,INCY)
C
C     COPIES A VECTOR, X, TO A VECTOR, Y.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION DX(1),DY(1)
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DY(IY) = DX(IX)
```

```
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C       CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C       CLEAN-UP LOOP
C
   20 M = MOD(N,7)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
         DY(I) = DX(I)
   30 CONTINUE
      IF( N .LT. 7 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,7
         DY(I) = DX(I)
         DY(I + 1) = DX(I + 1)
         DY(I + 2) = DX(I + 2)
         DY(I + 3) = DX(I + 3)
         DY(I + 4) = DX(I + 4)
         DY(I + 5) = DX(I + 5)
         DY(I + 6) = DX(I + 6)
   50 CONTINUE
      RETURN
      END
C
C ---------- END OF BLAS FUNCTIONS AND PROCEDURES ----------
C
```

(SUBS.FOR)

```
C
      SUBROUTINE UPDATX(LDA,N,M,GAMMA,IPIVOT,FEAS,X,DX,R,A,B,C,BOUND)
C
C-----------------------------------------------------------------
C  Purpose: This routine update the x vector.
C
C            k+1         k
C           x     =  T( x    ), where:
C                                    2          T
C           T(x) = x - (alpha / gamma) * D  * ( c - A w)
C
C  Inputs :
C  Outputs:
C-----------------------------------------------------------------
C
C
      IMPLICIT REAL*8(A-H, O-Z)
      DIMENSION A(LDA,1),C(1),B(1)
      DIMENSION R(1),X(1),DX(1),BOUND(1)
      DIMENSION IX(100), X2(100)
      CHARACTER*10 INTFIL(1)


      LOGICAL FEAS
      COMMON /XO/ X0(100)
C
      ALPHA=0.999
      cx=0.0
      IF(FEAS) GOTO 5
      IF(IPIVOT.EQ.N) THEN
         N = N - 1
         ALPHA = 1.
         FEAS = .TRUE.
      ENDIF
C     IF(IPIVOT.EQ.N) ALPHA=1.0
  5   STEP = - ALPHA / GAMMA
      DO 10 I=1,N
      X0(I)=X(I)
      X(I)=X(I)+STEP*DX(I)*R(I)
      cx=cx+x(i)*c(i)
  10  CONTINUE
```

```
      ione=1
C
      Print *, 'OBJ. VAL. : ', cx
      WRITE(6,123) cx,(x(i),i=1,n)
  123  FORMAT(F12.5,1X,30F5.2)
      CALL PRINTSLS(LDA,N,M,X,A,B,C,bound)
      PRINT *,'                '
      Return
      End
```

(UPDATX.FOR)