THE CONJUGATE GRADIENT METHOD AND MANAGEMENT SYSTEMS

by

CHRISTOPHER REBELO NUNES

B.E.(Elec.)(Hons.), B.E.(Mech.)(Hons.), University of Bombay
Bombay, India, 1966, 1967

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1969

Approved by

Major Professor

TABLE OF CONTENTS

1. INTRODUCTION

Linear programming has long been used to solve linear problems of management systems, but no effective algorithms have been developed for solving a general nonlinear programming problem. Several methods like the calculus of variation, dynamic programming and maximum principle have been recently developed. However, none of these techniques are very useful for solving complex management systems. The dynamic programming technique [12] faces dimensionality difficulty, whereby it cannot be used to solve problems with a large number of state variables. This is a severe limitation. Although several techniques are available to overcome this difficulty, they essentially trade computer memory for computer time. None of these techniques are as efficient as the original dynamic programming algorithm.

It is desirable to distinguish between two classes of nonlinear programming problems. The first class consists of nonlinear objective function and linear constraints while the second has a linear or nonlinear objective function and nonlinear constraints. The former is the less difficult of the two. In developing techniques for solving this class of problems, much can be gained from a knowledge of the established methods for unconstrained optimization [17]. Along these lines two possible approaches can be followed. One is to transform the constrained problem into unconstrained maximization problems, followed by the use of known techniques to solve each maximization problem. This type of approach is illustrated by Rosenbrock [16]. The other theoretically more satisfying approach, is to modify and extend methods of unconstrained maximization so that they can handle inequality constraints directly. The best example of this is Rosen's gradient projection method [15], which is an extension of the steepest ascent technique.

Unfortunately it suffers the same drawbacks i.e. slow rate of convergence for a nonlinear function. The variable metric method developed by Davidon [4] and later simplified by Fletcher and Powell [5] is a most powerful quadratically convergent technique for unconstrained optimization. Extension of the work of Davidon and Fletcher and Powell to the case of maximization under linear constraints appeared to be a promising path of attack in the solution of management problems.

The conjugate gradient method of Goldfarb [7,8,9] shows a significant improvement over existing nonlinear programming techniques in the rate of convergence. The purpose of this report is to apply the above method to industrial management problems characterized by linear constraints. This method is in many ways analogous to Rosen's gradient projection technique.

The geometrical interpretation and the outline of the algorithm is given. Two management problems are solved using this algorithm, and their optimum results are tabulated. The computational aspects with the aid of flow charts and computer programs are described in detail.

2. THE CONJUGATE GRADIENT METHOD

This method is based upon Davidon's variable metric method for unconstrained minimization. The conjugate gradient technique starts with an initial point in the m-dimensional variable space. If the given initial point is not feasible, a feasible one (satisfying all constraints) is obtained using Rosen's technique [15] for locating a feasible point. Starting with this feasible point a stepwise procedure is followed which gives a new feasible point at each step with an increased value of the objective function for a maximization problem. As the method proceeds, information about the local curvature of the objective function is incorporated into a matrix $\underline{H}_q^i$.

The length of the step is taken as the maximum possible without leaving the feasible region i.e. violating the constraint. Directions of search are determined by premultiplying the vector gradient $\underline{g}$ by the matrix $\underline{H}_q^i$. This variable metric $\underline{H}_q^i$ is modified when a constraint is added to or removed from the basis. To locate the maximum a cubic interpolation scheme of Davidon [4] given in the Appendix B is used.

2.1 Formulation of the problem

The conjugate gradient algorithm given below is used to obtain a local maximum of a nonlinear objective function

$$f(\underline{x}) = f(x_1, x_2, \ldots\ldots, x_m) \tag{1}$$

of m variables x_j, j=1,2, ..., m subject to linear equality and inequality constraints of the type

$$\sum_{j=1}^{m} a_{ij} x_j - b_i = 0, \quad i = 1,2, \ldots\ldots, e$$

$$\sum_{j=1}^{m} a_{ij} x_j - b_i \geq 0, \quad i=e+1,e+2,.....,k$$

This formulation includes as a special case non-negativity constraints (bounds) common in both linear and nonlinear programming problems.

On normalizing the above constraints we have

$$\sum_{j=1}^{m} n_{ij} x_j - b_i = 0, \quad i=1,2,.....,e \tag{2}$$

$$\sum_{j=1}^{m} n_{ij} x_j - b_i \geq 0, \quad i=e+1,e+2,.....,k \tag{3}$$

where the n_{ij} have been normalized such that

$$\sum_{j=1}^{m} (n_{ij})^2 = 1, \quad i=1,2,.....,k \tag{14}$$

It is obvious that bounds need not be normalized.

Geometrically x_i, i=1,2,.....,m represents a point in an m-dimensional Euclidean space E^m. Such a point can be represented by an m-dimensional column vector

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

The transpose of the vector $\underline{x}$ is denoted by $\underline{x}^T$. Subscripts on vectors

usually indicate components while superscripts in general differentiate between points.

Let $\underline{n}_i$ denote the column vector

$$\underline{n}_i = \begin{bmatrix} n_{i1} \\ n_{i2} \\ \vdots \\ n_{im} \end{bmatrix}$$

Then equations (2) through (4) can be written as

$$\underline{n}_i^T \underline{x} - b_i' = 0, \quad i=1,2,.....,e \tag{5}$$

$$\underline{n}_i^T \underline{x} - b_i' \geq 0, \quad i=e+1,e+2,....,k \tag{6}$$

where the $\underline{n}_i$ are unit normals, such that

$$\underline{n}_i^T \underline{n}_i = 1, \quad i=1,2,.....,k \tag{7}$$

The constraints (5) and (6) represent e hyperplanes and (k-e) closed half spaces, whose intersection in E^m is a convex polyhedral region R often called the "feasible" region. This region is assumed to be bounded and for this to be true it is necessary that $k \geq m+1$. Any point x such that $x \in R$ is called a "feasible" point. The hyperplane corresponding to a strict equality for a particular i in equation (6) is called the "defining" hyperplane for the associated half space. The boundary B of R is the intersection of R with the union of the (k-e) defining hyperplanes.

A set of hyperplanes is linearly independent if the set of unit normals to these hyperplanes, is linearly independent. If q are the number of linearly independent defining hyperplanes, then the intersection of these hyperplanes forms an (m-q) dimensional subspace M_q of E^m. If movement is restricted to M_q, then these hyperplanes are referred to as the "constraint basis". $\tilde{M}_q$ is a q-dimensional subspace spanned by the q unit normals to these q hyperplanes, hence $\tilde{M}_q$ is the orthogonal component to M_q i.e.

$$M_q \cap \tilde{M}_q = 0$$

and

$$M_q \cup \tilde{M}_q = E^m$$

2.2 Geometrical interpretation

It is convenient to use geometrical concepts to describe the maximization procedure. As previously mentioned $\underline{x}^i$ is a point in the m-dimensional Euclidean space E^m. The gradient at this point is given by

$$g(\underline{x}^i) = \left\{\frac{\partial f(\underline{x}^i)}{\partial x_j}\right\}$$

The components of this gradient can be considered as the coordinates of a point in another m-dimensional Euclidean space. If $f(\underline{x})$ is differentiable at all points $\underline{x}^i$, then each point $\underline{x}^i$ in the position space has associated with it a point $g(\underline{x}^i)$ in the gradient space. In addition, if $f(\underline{x})$ is twice differentiable, then in the neighborhood of any point $\underline{x}^i$ in E^m the second partial derivatives of $f(\underline{x})$ specify a linear mapping of changes in position, $\underline{dx}$, onto changes in gradient $\underline{dg}$, according to the relation

$$\underline{dg} = G(\underline{x})\ \underline{dx}$$

where the Hessian matrix $G(\underline{x})$ of second partial derivatives is given by

$$G(\underline{x}) = \left\{\frac{\partial^2 f(\underline{x})}{\partial x_i \partial x_j}\right\}$$

The vectors $\underline{dx}$ and $\underline{dg}$ will be directed likewise if $\underline{dx}$ is an eigenvector of $G(\underline{x})$. If the ratios among the corresponding eigenvalues of the matrix $G(\underline{x})$ are large, then for most $\underline{dx}$ there will be a considerable difference in the directions of $\underline{dx}$ and $\underline{dg}$.

Consider the problem of locating the point at which a strictly concave quadratic function

$$f(\underline{x}) = f_0 + \underline{a}^T \underline{x} + \frac{1}{2} \underline{x}^T \underline{G} \underline{x} \tag{8}$$

is maximized. Then the Hessian matrix is equal to the constant negative definite matrix $\underline{G}$ over the entire space E^m and we have

$$\underline{g}^{i+1} - \underline{g}^i = \underline{G}(\underline{x}^{i+1} - \underline{x}^i) \tag{9}$$

where $\underline{g}^i = g(\underline{x}^i)$

Therefore $$\underline{x}^{i+1} - \underline{x}^i = \underline{G}^{-1} (\underline{g}^{i+1} - \underline{g}^i) \tag{10}$$

where $\underline{G}^{-1}$ is the inverse of $\underline{G}$.

In the unconstrained case, where the feasible region R is all of E^m, a necessary and sufficient condition for a global maximum is that $g(\underline{x}) = 0$. The point at which this maximum occurs can then be found in one step by computing

$$\underline{x}^{i+1} = \underline{x}^i - \underline{G}^{-1} \underline{g}^i \tag{11}$$

This, of course, is just Newton's method (referred to as the second-order gradient method), for solving the set of m equations in m unknowns,

$g_j\ (x_1, x_2, \ldots\ldots, x_m) = 0$, $j = 1,2, \ldots\ldots, m$, resulting from the necessary and sufficient conditions for a global maximum of a strictly concave function.

It is obvious from the above discussion that the method of steepest ascent (i.e. moving in the gradient direction), and the iterative procedure based upon moving in a direction $-\underline{G}^{-1}\ \underline{g}^{i}$ can lead to very different paths to a solution.

In the above discussion it was assumed that the objective function $f(\underline{x})$ was quadratic in $\underline{x}$. If this were true or nearly so, then Newton's method might be the optimal maximization scheme to use. However, in most of the management problems $f(\underline{x})$ is not a quadratic function and this assumption is reasonably accurate only in a small neighborhood of a point. Therefore, evaluation of the Hessian matrix and inversion of it at each step might not, therefore, be the best policy to adopt as this can be computationally quite time consuming. In addition a serious drawback of second-order gradient methods is that for non-concave functions, convergence cannot be assured for starting points not close to the maximum.

Davidon's variable metric method [4] and its reformulation by Fletcher and Powell [5], takes a more satisfactory approach for locating the maximum of an unconstrained nonlinear function. Their method is to initially choose an approximation to $\underline{G}^{-1}$, and by using the actual relationship that exists between changes in $\underline{g}$ and in $\underline{x}$ improve this approximation with each subsequent iteration. This eventually results in a technique that incorporates the advantages of both Newton's and the steepest ascent methods, while avoiding their major limitations. As in the former method, Davidon's technique is stable and generates directions of search that are always

uphill, and does not require the computation of second partial derivatives. As in the latter method, it is quadratically convergent.

Consider now the problem of locating the point at which $f(\underline{x})$ assumes its maximum in M_q starting from a point $\underline{x}^i$. A necessary condition for $f^*(\underline{x})$ to be the global maximum of $f(\underline{x})$ for $\underline{x}$ in M_q is that the gradient $g^*(\underline{x})$ should lie in the subspace orthogonal to M_q i.e. lie in $\tilde{M}_q$. Therefore,

$$g^*(\underline{x}) = \underline{N}_q \underline{\alpha}$$

where $\underline{N}_q = [\underline{n}_1, \underline{n}_2, \ldots\ldots\ldots, \underline{n}_q]$ is an (m x q) matrix and $\underline{\alpha}$ is a q-dimensional column vector. The columns of $\underline{N}_q$ are the unit normals to the hyperplanes under consideration and lie in $\tilde{M}_q$. Substituting $\underline{g}^{i+1} = \underline{N}_q \underline{\alpha}$ into equation (10) we have

$$\underline{x}^{i+1} - \underline{x}^i = \underline{G}^{-1} (\underline{N}_q \underline{\alpha} - \underline{g}^i) \tag{12}$$

$(\underline{x}^{i+1} - \underline{x}^i)$ lies in subspace M_q. Therefore,

$$\underline{N}_q{}^T (\underline{x}^{i+1} - \underline{x}^i) = 0 = \underline{N}_q{}^T \underline{G}^{-1} \underline{N}_q \underline{\alpha} - \underline{N}_q{}^T \underline{G}^{-1} \underline{g}^i$$

Since the rank of $\underline{N}_q$ is $q \leq m$ and that of $\underline{G}^{-1}$ is $m \geq q$ the inverse of $(\underline{N}_q{}^T \underline{G}^{-1} \underline{N}_q)$ exists [3] and $\underline{\alpha}$ is given by

$$\underline{\alpha} = (\underline{N}_q{}^T \underline{G}^{-1} \underline{N}_q)^{-1} \underline{N}_q{}^T \underline{G}^{-1} \underline{g}^i$$

Equation (12) can then be written as

$$\underline{x}^{i+1} - \underline{x}^i = \underline{G}^{-1} (\underline{N}_q{}^T (\underline{N}_q{}^T \underline{G}^{-1} \underline{N}_q)^{-1} \underline{N}_q{}^T \underline{G}^{-1} \underline{g}^i - \underline{g}^i)$$

Therefore,

$$\underline{x}^{i+1} = \underline{x}^{i} + (\underline{G}^{-1}\,\underline{N}_q\,(\underline{N}_q^{T}\,\underline{G}^{-1}\,\underline{N}_q)^{-1}\,\underline{N}_q^{T}\,\underline{G}^{-1} - \underline{G}^{-1})\underline{g}^{i} \tag{13}$$

Equation (13) can be expressed as

$$\underline{x}^{i+1} = \underline{x}^{i} - \hat{\underline{P}}_q\,\underline{G}^{-1}\underline{g}^{i} \tag{14}$$

where $\hat{\underline{P}}_q = \underline{I} - \underline{G}^{-1}\,\underline{N}_q(\underline{N}_q^{T}\,\underline{G}^{-1}\,\underline{N}_q)^{-1}\,\underline{N}_q^{T}$

is a non-euclidean projection operator that projects E^m onto subspace M_q.

Equation (14) shows that to locate a maximum one should move in a direction $-\hat{\underline{P}}_q\,\underline{G}^{-1}\,\underline{g}^{i}$. As in the unconstrained case, a better approach for the nonlinear problem is one based upon Davidon's method. In the conjugate gradient algorithm described in the next section a trial value for the operator $-\hat{\underline{P}}_q\,\underline{G}^{-1}$ at a point in M_q is initially specified. This is the positive definite variable metric $\underline{H}_q$.

3. OUTLINE OF THE CONJUGATE GRADIENT ALGORITHM

This algorithm is essentially an extension of Fletcher and Powell's version of Davidon's variable metric method to the maximization of an objective function subject to linear constraints. It closely follows Rosen's gradient projection method; the main difference is that the directions of search are determined by premultiplying the gradient vector by the matrix $\underline{H}_q$ rather than by the orthogonal projection matrix $\underline{P}_q$. The algorithm is based upon the manner in which the matrix $\underline{H}_q$ is updated. In the course of the statement of this algorithm, eqs. (2) and (4) are given for modifying $\underline{H}_q$ when a hyperplane is either dropped from or added to the constraint basis, while equation (5), due to Davidon [4] is used to improve the approximation of $\underline{H}_q$ to $-\hat{\underline{P}}_q\, G^{-1}$. Recursion relations for obtaining the inverse matrix $(\underline{N}_q^T\, \underline{N}_q)^{-1}$ on changing the constraint basis, are given in the Appendix A.

Let $\underline{x}^i$ denote the current point and let $f(\underline{x}^i)$ and $g(\underline{x}^i)$ the corresponding value of the function and the gradient. The matrix operator $\underline{H}_q$ is therefore $\underline{H}_q{}^i$. The basic conjugate gradient algorithm put forward by Goldfarb [7] can be stated as follows:

(0) Initially some feasible point $\underline{x}^0$ is obtained and $\underline{H}_0{}^0$ is chosen to be any positive definite matrix, usually $\underline{I}$, the identity matrix. If the point $\underline{x}^0$ lies in the intersection M_q of q linearly independent hyperplanes, then these constraints are added to the constraint basis and $\underline{H}_q{}^0$ is computed by recursively using eq. (4) (given in step (iv) of this algorithm) q-times once for each hyperplane. The e hyperplanes that correspond to the equality constraints (eq. (5) in section 2.1), are added first. Compute the gradient $\underline{g}^0 = g(\underline{x}^0)$.

(i) Compute $\underline{\alpha}$ as follows

$$\underline{\alpha} = (\underline{N}_q^T \underline{N}_q)^{-1} \underline{N}_q^T \underline{g}^i \tag{1}$$

If $||\underline{g}^i|| = 0$ and $\alpha_j \leq 0$, $j = e+1, e+2, \ldots\ldots, q$ then

$\underline{x}^i$ is a global maximum. If this fails, go to (ii).

(ii) Compute $||\underline{H}_q^i \underline{g}^i||$ then either $2||\underline{H}_q^i \underline{g}^i|| > \max (0, \alpha_q d_{qq}^{-\frac{1}{2}})$ or $2||\underline{H}_q^i \underline{g}^i|| \leq \alpha_q d_{qq}^{-\frac{1}{2}}$, where d_{qq} is the qth diagonal element of the inverse matrix $(\underline{N}_q^T \underline{N}_q)^{-1}$ and where it is assumed that $\alpha_q d_{qq}^{-\frac{1}{2}} \geq \alpha_i d_{ii}^{-\frac{1}{2}}$, $i = e+1, e+2, \ldots\ldots, q-1$

If the former applies proceed to (iii). If the latter holds good, drop the qth hyperplane from the constraint basis and compute $\underline{H}_{q-1}^i$ from

$$\underline{H}_{q-1}^i = \underline{H}_q^i + \frac{\underline{P}_{q-1} \underline{n}_q \underline{n}_q^T \underline{P}_{q-1}}{||\underline{P}_{q-1} \underline{n}_q||^2} \tag{2}$$

where $$\underline{P}_{q-1} = \underline{I} - \underline{N}_{q-1} (\underline{N}_q^T \underline{N}_{q-1})^{-1} \underline{N}_{q-1}^T$$

Let $q = q-1$ and return to (i).

(iii) Let $\underline{s}^i = \underline{H}_q^i \underline{g}^i$ and compute λ^i from the following formulas:

$$\lambda_j = -(\underline{n}_j^T \underline{x}^i - b_j)/ \underline{n}_j^T \underline{s}^i, \quad j = q+1, q+2, \ldots\ldots, k$$

$$\lambda^i = \min (\lambda_j > 0) \tag{3}$$

Using the scheme given in the Appendix B, interpolate cubically to

obtain γ^i, $0 \leq \gamma^i \leq \lambda^i$ such that $f(\underline{x}^i + \gamma^i \underline{s}^i)$ is maximized. Set $\underline{x}^{i+1} = \underline{x}^i + \gamma^i \underline{s}^i$ and compute $\underline{g}^{i+1} = g(\underline{x}^{i+1})$

(iv) If $\gamma^i = \lambda^i$, add to the constraint basis the hyperplane corresponding to the min (λ_j) in (iii). Obtain $\underset{\sim}{H}_{q+1}^{i+1}$ from the relation

$$\underline{H}_{q+1}^{i+1} = \underline{H}_q^i - \frac{\underline{H}_q^i \underline{n}_j^T \underline{n}_j \underline{H}_q^i}{\underline{n}_j^T \underline{H}_q^i \underline{n}_j} \tag{4}$$

Set $q = q+1$ and $i = i+1$ and return to (i).

(v) Otherwise set $\underline{y}^i = \underline{g}^{i+1} - \underline{g}^i$ and $\underline{\sigma}^i = \gamma^i \underline{s}^i$ and update $\underline{H}_q^i$ as follows:

$$\underline{H}_q^{i+1} = \underline{H}_q^i + \underline{A}^i + \underline{B}^i \tag{5}$$

where $$\underline{A}^i = - \frac{\underline{\sigma}^i (\underline{\sigma}^i)^T}{(\underline{\sigma}^i)^T \underline{y}^i}$$

and $$\underline{B}^i = - \frac{\underline{H}_q^i \underline{y}^i (\underline{y}^i)^T \underline{H}_q^i}{(\underline{y}^i)^T \underline{H}_q^i \underline{y}^i}$$

The constraint basis, and consequently M_q remains unchanged. Set i = i+1 and return to (i).

The above equation (5) serves two purposes. The term $\underline{A}^i$ improves the approximation of $\underline{H}_q^i$ to $-\hat{\underline{P}}_q \underline{G}^{-1}$ and the term $\underline{B}^i$ generates mutually conjugate direction of search.

4. FUNCTIONAL DESCRIPTION OF THE COMPUTER LOGIC

In formulating a problem to be run on the IBM 360/50 the following quantities had to be defined for the successful operation of the conjugate gradient technique:

a) A set of variables, x_j, j = 1,2, ..., m

b) Linear constraints, C_i, i = 1,2, ..., k

c) The objective function, $f(\underline{x})$, to be maximized and its gradient $g(\underline{x})$.

d) A initial point $\underline{x}^0$.

Description of the constraints

The linear constraints may consist of inequalities or equalities or both. An inequality constraint, C_i, must be written in the form $\sum_{j=1}^{j=m} a_{ij} x_j \geq b_i$. The constraints form a convex region which is usually bounded. The maximum of a concave nonlinear function will be found even if the region is unbounded. However, the use of lower and upper bounds on some or all of the variables will often speed up the convergence to the maximum. For the technique to operate, at least one constraint is essential.

Constraints on only one variable are called bounds. Any constraint which can be expressed as $\pm\ x_j \geq b_i$ is considered as a bound. The input consists first of bounds, if any, which are numbered as $C_1, C_2, \ldots\ldots, C_b$, say, and then the constraints other than bounds, numbered as $C_{b+1}, C_{b+2}, \ldots\ldots, C_k$. Non-negativity constraints are not assumed by the program, and have to be expressed as bounds. A bound requires less computation time and should be used in place of a constraint, wherever possible.

The constraints C_i are considered as equality if they can be expressed

in the form $\sum_{j=1}^{m} a_{ij} x_j - b_i = 0$. The objective function $f(\underline{x})$ to be maximized and its gradient $g(\underline{x})$ are computed by the subroutine FUNCT. The program will always find a global maximum if $f(\underline{x})$ is a concave function (a linear function is a special case of a concave function) and has continuous gradient. However, solutions may be obtained even if the above two conditions are not satisfied. Generally, if $f(\underline{x})$ is not concave, a local maximum will be found which may depend on the initial point chosen. Under such circumstances several widely seperated starting points should be tried. If different results are obtained, the best that can be done is to take the maximum value of this set of local maxima. A minimization problem can be solved by this algorithm as maximizing the negative of the original function, namely maximizing $-f(\underline{x})$.

The initial point $\underline{x}^0$ need not be feasible, as the program will first obtain a feasible point, if one exists. However, this procedure can be time consuming. The best estimate of the feasible point is preferred. The closer the starting point is to the optimum, the lesser is the time consumed for obtaining the correct solution.

As input, m number of variables, k total number of constraints including bounds, b number of bounds, and e number of equality constraints, are read first. If $b \neq 0$, the b subscripts for the bounds are read next. these are $\pm$ j's, representing $\pm x_j \geq b_i$. If $(k-b) \neq 0$, the constraints C_i, i = b+1,b+2,.....,k, are read next i.e. a_{ij} and b_i, starting first from equality constraints. The initial point $\underline{x}^0$ is chosen as described above. As in the gradient projection method, scaling of the constraints is also essential in this technique. The scaling divisor for a particular equality or inequality constraint is the square root of the

sum of the coefficients of $\underline{x}$ squared. This is essentially normalization of the constraints whereby the coefficients of $\underline{x}$ become n_{ij} and the right hand side of the constraints become b_i'.

Tolerances

The gradient tolerance, ε_1, is used to determine when the norm of the gradient is zero, and the problem has reached a maximum. The value of ε_1 is harder to determine for a nonlinear function. In general, the smaller the value of ε_1, the better the "maximum" will be. But, this will be at the cost of computation time. The value of ε_1 is chosen to be .001.

The constraint tolerance, ε_2, determines when a point is on a constraint, and is essentially an acceptable error in satisfying the constraints. ε_2 is taken = .0001.

The linear dependence tolerance, ε_3, determines when a constraint is linearly dependent and therefore cannot be added to the basis. The value of ε_3 is chosen as .005.

Linearly dependent constraints

The normal distance from the point $\underline{x}$ to a constraint C_i is called lambda $\lambda_i(\underline{x}) = \sum_{j=1}^{m} n_{ij}\, x_j - b_i'$. These lambdas are tested to determine feasibility and which constraints are satisfied as equalities. The constraint vectors form the columns of the constraint matrix $\underline{N}_k$;

$\underline{N}_k = [\underline{n}_1, \underline{n}_2, \ldots\ldots\ldots, \underline{n}_k]$.

The basis is the set of constraints which are active, or limiting, at a particular point during the solution of a problem. The initial basis consists of equalities, if any, which are, by definition, always limiting. In the course of the solution other constraints which are found to be

limiting are added to the basis, and some of these may be dropped from the basis later if they are no longer limiting. The algorithm drops or adds one constraint at a time. The number of constraints in the basis may be less than or equal to m. Those columns of the constraint matrix which are in the basis are represented by $\underline{N}_q$. Corresponding to the basis we have an inverse matrix, $(\underline{N}_q^T \underline{N}_q)^{-1}$.

The concept of linear dependence is very crucial to the overall accuracy of the solution. Since $(\underline{N}_q^T \underline{N}_q)$ must be non-singular, constraints which are linearly dependent on the constraints in the basis must not be added. Since most of the computation is based on the inverse of this matrix, it is also necessary to exclude those constraints which are almost linearly dependent in order to retain a reasonable amount of accuracy. The procedure given in Appendix A, in addition to computing the inverse matrix, results in a convenient test for linear dependence, since a constraint is linearly dependent if its projection is zero. This test is actually made against a non zero tolerance, ε_3. If $||\underline{P}\,\underline{n}_i|| \leq \varepsilon_3$, then C_i is linearly dependent and is not added to the basis.

As previously mentioned any point satisfying the equality and inequality constraints is a feasible point. When a constraint is added to the basis, it is considered as an equality when testing feasibility. Since $\underline{x}$ should remain on the constraints in the basis, this gives a good criterion for detecting round-off errors and preventing $\underline{x}$ from going astray. A linearly dependent constraint should theoretically be satisfied along with the constraints in the basis, so, to be feasible, $\underline{x}$ must also be on the linearly dependent constraint. If $\underline{x}$ is not feasible with respect to a particular constraint, then that constraint is said to be violated. The test for feasibility at any $\underline{x}$ is: If $\lambda_i(\underline{x}) \geq -\varepsilon_2$, all C_i, and

$|\lambda_i(\underline{x})| \leq \varepsilon_2$, all C_i in the basis, $\underline{x}$ is feasible. The $\underline{x}$-correction procedure given in Section 5.2 corrects $\underline{x}$ to the constraints in the basis, whereby it finds an $\underline{x}$ for which $|\lambda_i(\underline{x})| \leq \varepsilon_2$ for all C_i in the basis.

The gradient, $g(\underline{x})$, gives the direction of increasing $f(\underline{x})$. Steps are taken along $\underline{H}_q\ \underline{g}$ to increase f(x) while staying feasible. If $||\underline{Pg}|| \leq \varepsilon_1$, and $r_i/\sqrt{d_{ii}} \leq \varepsilon_1$, where i is an inequality constraint in the basis, then this is the maximum. The quantity r_i is as defined in Appendix A and d_{ii} is the ith element of the inverse matrix. It is possible to take a step which requires no changes in the basis. Such a step is called an interior step. That is, if $\underline{x}^1$ lies on the same intersection as $\underline{x}^0$, then the step to $\underline{x}^1$ was interior. When a step is interior, a test is made for a constraint to be dropped even if the gradient is not zero. This speeds up the convergence of a nonlinear problem, since a zero projected gradient does not have to be found with a basis that is not optimal.

Forming a Basis and Re-inverting

If the problem contains equalities, the initial basis consists of these equalities. The increase of this initial basis is computed when and only when new constraint data is read. When computing this initial inverse, the equalities are added in order of maximum $||\underline{P}\ \underline{n}||$. After the first equality is added, the remaining equalities are projected, and the one with the maximum $||\underline{P}\ \underline{n}||$ (most linearly independent) is added next. This process is repeated until a linearly dependent equality is found or all equalities have been added to the basis. Equalities which are linearly dependent, if any, are not added to the basis, but will still be classified as equalities rather than linearly dependent constraints.

Changes are made in the basis to find a feasible point as well as the

maximum. Briefly, the normal criteria for adding a constraint to the basis are: 1) if $\underline{x}$ is not feasible, add the constraint with the most negative lambda, and 2) if the gradient is non-zero, add the constraint with the most negative $\underline{z}^T \underline{n}_i$ where $|\lambda_i| \leq \epsilon_2$, if any. $\underline{z}$ is the unit vector in direction of step i.e. $\underline{z} = \underline{H}_q \underline{g} / ||\underline{H}_q \underline{g}||$. Constraints may be dropped from the basis for any of the following reasons: 1) $\underline{x}$ is not feasible and the constraint with the most negative lambda is linearly dependent; 2) the step was interior, and 3) the gradient is zero.

MXRN, maximum number of re-inversions. The number of re-inversions is limited to prevent the problem from re-inverting too often thereby consuming extra time. The program will not re-invert twice in a row to the same basis, but it may re-invert after only one basis change or possibly repeat a series of re-inversions.

LIMIT, maximum number of steps. The number of steps required to reach the maximum is difficult to pre-determine since it depends on such factors as size, type of function and number of constraints in the basis. The choice of maximum number of steps should be such that results are not repeated unnecessarily after the maximum is attained.

Subroutine CLASS classifies the constraints into categories v and w. In the program

u = linearly dependent constraints $||\underline{P}\, \underline{n}_i|| \leq \epsilon_3$

v = constraints not in the basis with $\lambda = 0$

w = constraint not in the basis with $\lambda > 0$

q = number of constraints in the basis

q^* = constraints added to the initial basis

Subroutine REINV essentially computes the inverse matrix, whereas subroutine COMP1 and COMP2 do the matrix computations required by the conjugate

gradient algorithm.

In the light of the above description, the computational procedure followed for the method under consideration is briefly described.

The main program is divided into three parts: input, starting procedure and the iteration procedure. The first section reads the input and obtains the initial conditions for the problem. When the constraints are read, they are normalized and stored. If equality constraints are present, then these equalities form the initial basis, and the corresponding inverse matrix is computed and stored.

The next section is the starting procedure, in which the program tests for feasibility and, if necessary, obtains a feasible $\underline{x}$. If $\underline{x}$ is not feasible, the procedure for finding a feasible $\underline{x}$ is as follows. If one or more constraints in the basis are violated, i.e. $|\lambda_i| > \varepsilon_2$, one $\underline{x}$ correction is computed. All constraints in the current constraint basis should then be satisfied. If any constraint in the basis is still violated, this is considered as an $\underline{x}$-correction failure, and this necessitates a re-inversion to proceed further. If constraints which are not in the basis are violated i.e. $\lambda_i(\underline{x}) < -\varepsilon_2$, the program computes the most negative lambda and tries to add the corresponding constraint to the basis. If this constraint is linearly independent, only then it is added, and the procedure is repeated starting from the $\underline{x}$-correction. On the other hand if the constraint is linearly dependent, the program tests against ε_3 for a constraint to be dropped. If it finds no constraint to be dropped on the initial iteration, there is no feasible $\underline{x}$. On the later iterations, this is considered to be an $\underline{x}$-correction failure. In either case, a re-inversion is essential to proceed. However, in the case of no feasible $\underline{x}$ a re-inversion will not necessarily obtain a feasible $\underline{x}$ if one does not exist. If there

is a constraint to be dropped then it is dropped from the basis. However, the program tries to add the constraint with the most negative lambda. If this constraint is still linearly dependent, it is considered to be an $\underline{x}$-correction failure and this necessitates a re-inversion to proceed. Depending upon the number of constraints in the basis the matrix $\underline{H}_q$ is computed according to the relation (4) in section 3.

On obtaining a feasible $\underline{x}$ at the beginning of each iteration, subroutine FUNCT is called upon to compute $f(\underline{x})$ and $g(\underline{x})$. This entitles the program to enter the iteration procedure. An iteration includes the projection of the gradient, testing for the maximum, changes in the basis, computation of the matrix $\underline{H}_q$ and the calculation of $\underline{z}$. The subroutine CLASS classifies the non-basis constraints into v and w. If $||\underline{g}|| \leq \varepsilon$ and $\alpha_j \leq 0$, j=e+1,e+2, ...,q, then the point is the maximum required.

If $||\underline{g}|| > \varepsilon_1$ and q=0 then $\underline{z}$ is computed. If q > 0 and $||\underline{Pg}|| \leq \varepsilon_1$ then the program makes the necessary test as given in the flow chart (Appendix C) for dropping a constraint from the basis. If there is no constraint to drop, then the maximum test is satisfied, and the current point is the optimum. On dropping the constraint, the matrix $\underline{H}_q$ is computed using the proper relation.

If $||\underline{Pg}|| > \varepsilon_1$ and (m-q) > 0, $\underline{z}$ is computed. If v > 0 and $(\min \underline{z}^T \underline{n}_i) < 0$, for i in v, then the corresponding constraint if linearly independent is added to the basis and $\underline{H}_q$ is computed. If the constraint is linearly dependent the program repeats for $(\min \underline{z}^T \underline{n}_i) < 0$, i in v. When v=0, the program tests for basis change at this iteration. If q^* or $\eta = 0$ the program tests against $2||\underline{Hg}||$ for a constraint to be dropped (step (ii) of the conjugate gradient algorithm). On dropping a constraint, the program

returns to a location where the gradient is projected and tested as given in the flow chart. If the constraint cannot be dropped, then step (iii) of the algorithm is followed, wherein the cubic interpolation scheme is followed to obtain a higher value of $\underline{x}$. If $\gamma^i = \lambda^i$ as in step (iv), then the corresponding constraint of step (iii) is added to the basis and $\underline{H}_q$ is computed. If step (iv) fails, proceed to step (iv). If step (iv) fails, then step (v) is executed. At the beginning of step (iii), the iteration counter is changed. The program stops if the number of iterations exceeds the limit given.

5. APPLICATION OF THE CONJUGATE GRADIENT METHOD TO MANAGEMENT PROBLEMS

Faced with the increasing complexities of engineering, production and sales in a highly developed and often strongly competitive economy, management must deal with a multitude of interlocking and often far-reaching tasks involving many decision-making and control problems. In many industries the control of inventory is intimately related to the scheduling of production, and in this section of the report two different production scheduling and inventory models are considered.

The first one is a simple production planning model involving a single state variable. The second is a more complex model consisting of two state variables. Solutions were obtained by the application of the conjugate gradient technique.

5.1 Test Problem No. 1

An inventory model with one state variable

Consider the case of a manufacturing enterprise whose sales rate is known with certainty. The rate of change of inventory level I(t) is given by

$$\frac{dI(t)}{dt} = P(t) - Q(t) \tag{1}$$

where $P(t)$ = production rate at time t

and $Q(t)$ = sales rate at time t.

The problem is to minimize the cost function given by

$$C_T = \int_0^T [C_I (I_m - I(t))^2 + C_P \exp (P_m - P(t))^2] \, dt \tag{2}$$

where C_T is the total cost of production and inventory. C_I is the cost of carrying inventory and C_P is the minimum production cost which occurs when the production rate equals P_m. The quantity P_m can be considered as the production capacity of the manufacturing plant. Since the plant is designed for a capacity P_m, an increase in production may require additional equipment and manual labor which can be very expensive. On the other hand, a decrease in production below P_m will be equally expensive due to maintenance of unused equipment and idle labor which cannot be decreased due to contract agreements. The quantity I_m can be considered as the capacity for storage of inventory. In actual practice, the minimum storage cost is obtained when the storage capacity is completely utilized. In addition, the cost function given by equation (2) has the smoothing capability which is frequently desirable for many manufacturing processes. In this case, I_m and P_m can be considered as desirable inventory and production levels. Let us further assume that the sales forecast is known and is given by the linear relation

$$Q(t) = a + bt \tag{3}$$

and the initial inventory is

$$I(0) = c \tag{4}$$

where a, b and c are known constants.

Substituting for Q(t) in equation (1) we have

$$\frac{dI(t)}{dt} = P(t) - a - bt \tag{5}$$

In order to solve this model by the conjugate gradient algorithm, equations (2) and (5) are approximated by difference equations.

Thus, equation (5) is reduced to

$$I(t + \Delta t) = I(t) + (P(t) - a - bt)\Delta t \tag{6}$$

To obtain the cost, equation (2) has to be evaluated over the limits $n\Delta t$ to $(n+1)\Delta t$. If Δt is small, this integral can be approximated by

$$\int_{n\Delta t}^{(n+1)\Delta t} [C_I (I_m - I(t))^2 + C_P \exp(P_m - P(t))^2] \, dt$$

$$= [C_I (I_m - I(t))^2 + C_P \exp(P_m - P(t))^2] \Delta t \tag{7}$$

The following numerical values are assumed

a	= 2	C_I	= 0.1	P_m	= 5
b	= 1	C_P	= 0.001	t_0	= 0
c	= 5	I_m	= 10	T	= 1

The range of the control variable P is taken to be between 0 and 7. These are essentially bounds on the amount of production in the method employed. The state variable is the inventory I and eq. (6) represents a constraint on the inventory.

The number of stages into which the process can be divided is governed by the accuracy desired and the cost of computation incurred. A large number of stages will give a better approximation to a continuous process and yield more accurate results. However, this accuracy is obtained at the cost of increased computational time. For this study, solutions using the conjugate gradient technique were obtained for a 5-stage and 10-stage process.

This technique can only handle constraints on the control variable, and as such equality constraints on state variables given by eq. (6) have to be eliminated by obtaining the partial derivatives of the state variables with

respect to the control variables. This differentiation is used in obtaining the gradient vector.

The cost function given by eq. (7) is for a time interval Δt, but when the integral is evaluated over the entire time interval T, the objective function for a 5-stage process can be written as

$$C_T = [C_I (I_m - I(t + \Delta t))^2 + C_P \exp(P_m - P(t))^2] \Delta t + [C_I (I_m - I(t+2\Delta t))^2 + C_P \exp(P_m - P(t+\Delta t))^2] \Delta t + \ldots\ldots\ldots\ldots \ldots\ldots\ldots + [C_I (I_m - I(t+5\Delta t))^2 + C_P \exp(P_m - P(t+4\Delta t))^2] \Delta t$$

To obtain the gradient at a point we have to differentiate the above cost function with respect to the control variables P at the various stages. As the state variables are related to the control variables by eq. (6), the chain rule is used for the necessary differentiation. As an example the partial derivative of the cost function with respect to the first control variable P(t) is given by

$$\frac{\partial C_T}{\partial P(t)} = -2 C_I \Delta t (I_m - I(t+\Delta t)) \frac{\partial I(t+\Delta t)}{\partial P(t)} - 2 C_P \Delta t (P_m - P(t)) \exp(P_m - P(t))^2 - 2 C_I \Delta t (I_m - I(t+2\Delta t)) \frac{\partial I(t+2\Delta t)}{\partial P(t)} + \ldots\ldots\ldots\ldots\ldots\ldots - 2 C_I \Delta t (I_m - I(t+5\Delta t)) \frac{\partial I(t+5\Delta t)}{\partial P(t)}$$

Consider now the differentiation of the inventory with respect to the production, we have for example

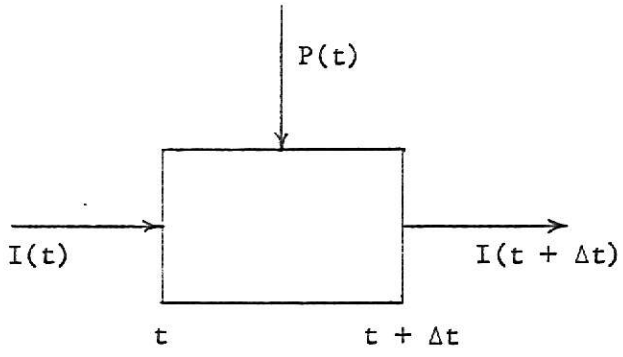
$$\frac{\partial I(t+3\Delta t)}{\partial P(t)} = \frac{\partial I(t+3\Delta t)}{\partial I(t+2\Delta t)} \cdot \frac{\partial I(t+2\Delta t)}{\partial I(t+\Delta t)} \cdot \frac{\partial I(t+\Delta t)}{\partial P(t)}$$

This briefly indicates the manner in which the components of the gradient vector are obtained.

This and the problem that follows were both solved on an IBM 360/50 computer. The flow chart and the computer program written in FORTRAN language are given in the Appendices C and D respectively. The main program includes the cubic interpolation scheme to locate the optimum. The subroutine FUNCT when called, computed the cost function, the gradient vector and the inventory. This problem being one of minimization was solved by the conjugate gradient maximization algorithm by maximizing the negative of the original objective function. The necessary changes made can be noticed in the subroutine FUNCT.

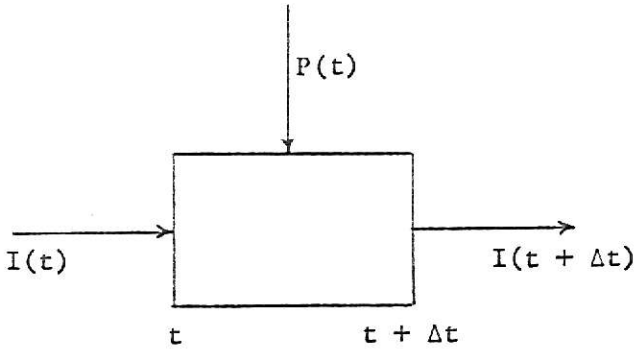
Three different initial points were chosen. It was observed that the optimal values remained the same irrespective of the choice of the initial points. A summary of the results obtained is given in Tables 1 and 2. The convergence rates for the 2 processes with different starting points is given in Tables 3 and 4. The optimal cost was 0.940 for a 5-stage and 0.932 for a 10-stage process. As can be seen, there is no significant difference between the two optimum values. The reason is that the inventory and cost functions are smooth functions as is obvious from Figs. 1 through 4. It is therefore concluded that, for the particular model under consideration, an increase in the number of stages does not significantly improve the value of the optimal cost. This is, however, not the general

Table 1. 5-stage process



t	t + Δt	I(t)	P(t)	I(t + Δt)	C_T	Q(t + Δt)
0.0	0.2	5.000	7.000	5.960	0.420	2.200
0.2	0.4	5.960	7.000	6.880	0.867	2.400
0.4	0.6	6.880	6.947	7.749	0.840	2.600
0.6	0.8	7.749	6.771	8.554	0.913	2.800
0.8	1.0	8.544	6.432	9.230	0.940	3.000

Table 2. 10-stage process



t	t + Δt	I(t)	P(t)	I(t + Δt)	C_T	Q(t + Δt)
0.0	0.1	5.000	7.000	5.490	0.232	2.100
0.1	0.2	5.490	7.000	5.970	0.419	2.200
0.2	0.3	5.970	7.000	6.440	0.569	2.300
0.3	0.4	6.440	7.000	6.900	0.685	2.400
0.4	0.5	6.900	7.000	7.350	0.773	2.500
0.5	0.6	7.350	6.899	7.780	0.836	2.600
0.6	0.7	7.780	6.808	8.191	0.879	2.700
0.7	0.8	8.191	6.691	8.580	0.907	2.800
0.8	0.9	8.580	6.518	8.942	0.924	2.900
0.9	1.0	8.942	6.164	9.258	0.932	3.000

Table 3. Convergence rate for the 5-stage process with different initial points

No. of iterations	Total no. of functional evaluations	No. of constraints in basis	P_1	P_3	P_5	C_T
		Initial point: All $P_i = 1.0$				
0	1	0	1.000	1.000	1.000	889.395
10	11	0	6.732	5.628	4.643	1.094
20	21	3	7.000	7.000	5.255	0.951
30	33	3	7.000	7.000	5.721	0.942
44	54	2	7.000	6.947	6.432	0.940
		Initial point: All $P_i = 5.0$				
0	1	0	5.000	5.000	5.000	1.431
10	11	2	7.000	6.943	5.260	0.967
20	28	3	7.000	7.000	5.571	0.942
30	42	3	7.000	7.000	6.445	0.940
46	77	2	7.000	6.946	6.431	0.940
		Initial point: All $P_i = 7.0$				
0	1	0	7.000	7.000	7.000	0.949
5	6	2	7.000	6.910	6.612	0.940
15	16	2	7.000	6.947	6.434	0.940
25	160	2	7.000	6.947	6.432	0.940

Table 4. Convergence rate for the 10 - stage process with different initial points

No. of iterations	Total no. of functional evaluations	No. of constraints in basis	P_1	P_5	P_{10}	C_T
		Initial point: All $P_i = 1.0$				
0	1	0	1.000	1.000	1.000	8889.355
20	21	3	7.000	6.680	5.148	0.955
40	56	6	7.000	7.000	5.511	0.934
60	81	5	7.000	7.000	5.685	0.932
84	238	5	7.000	7.000	6.164	0.932
		Initial point: All $P_i = 5.0$				
0	1	0	5.000	5.000	5.000	1.414
20	21	5	7.000	7.000	5.157	0.944
40	45	6	7.000	7.000	5.338	0.934
60	66	5	7.000	7.000	5.585	0.932
77	111	4	7.000	6.972	6.165	0.932
		Initial point: All $P_i = 7.0$				
0	1	0	7.000	7.000	7.000	0.943
20	21	4	7.000	6.961	6.319	0.932
44	58	4	7.000	6.973	6.164	0.932

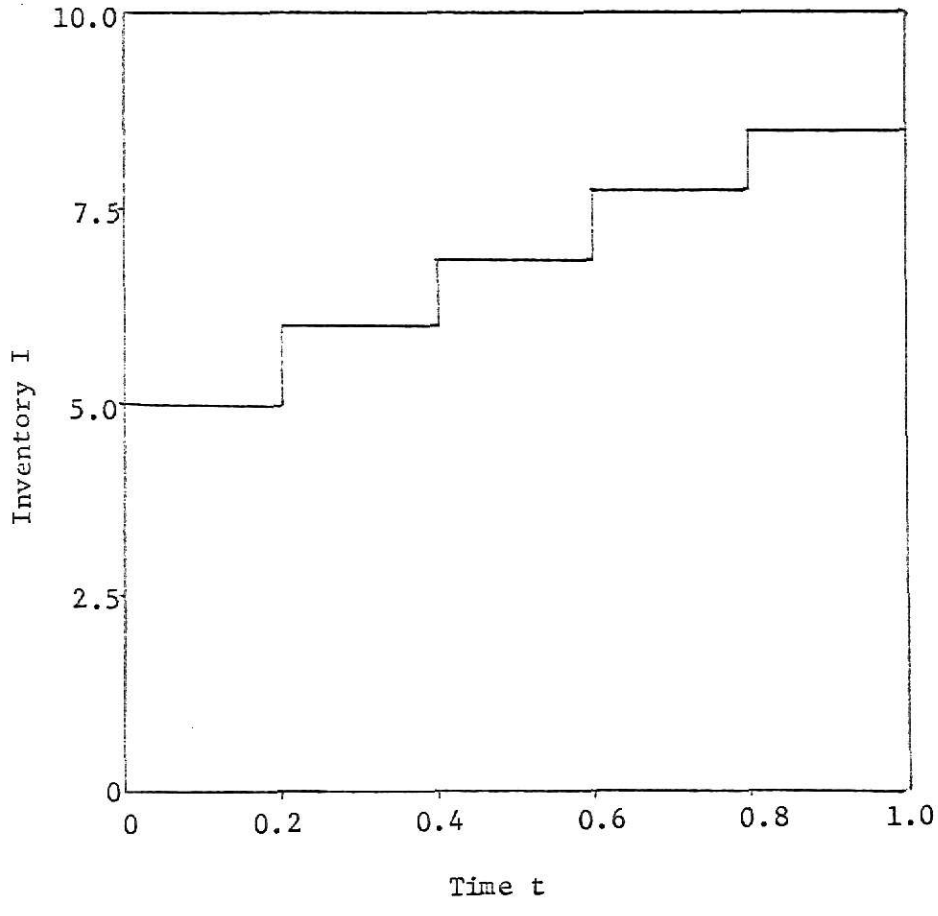


Fig. 1. Inventory for the 5-stage process

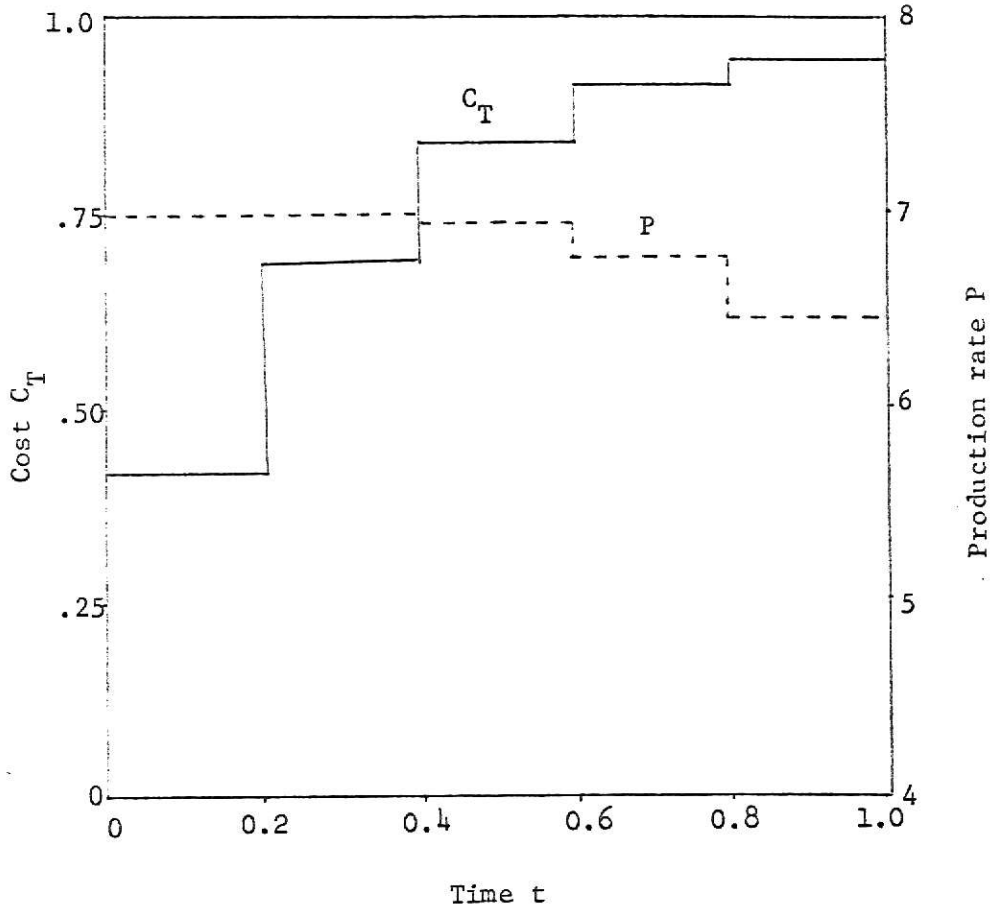


Fig. 2. Cost and production rate for the 5-stage process

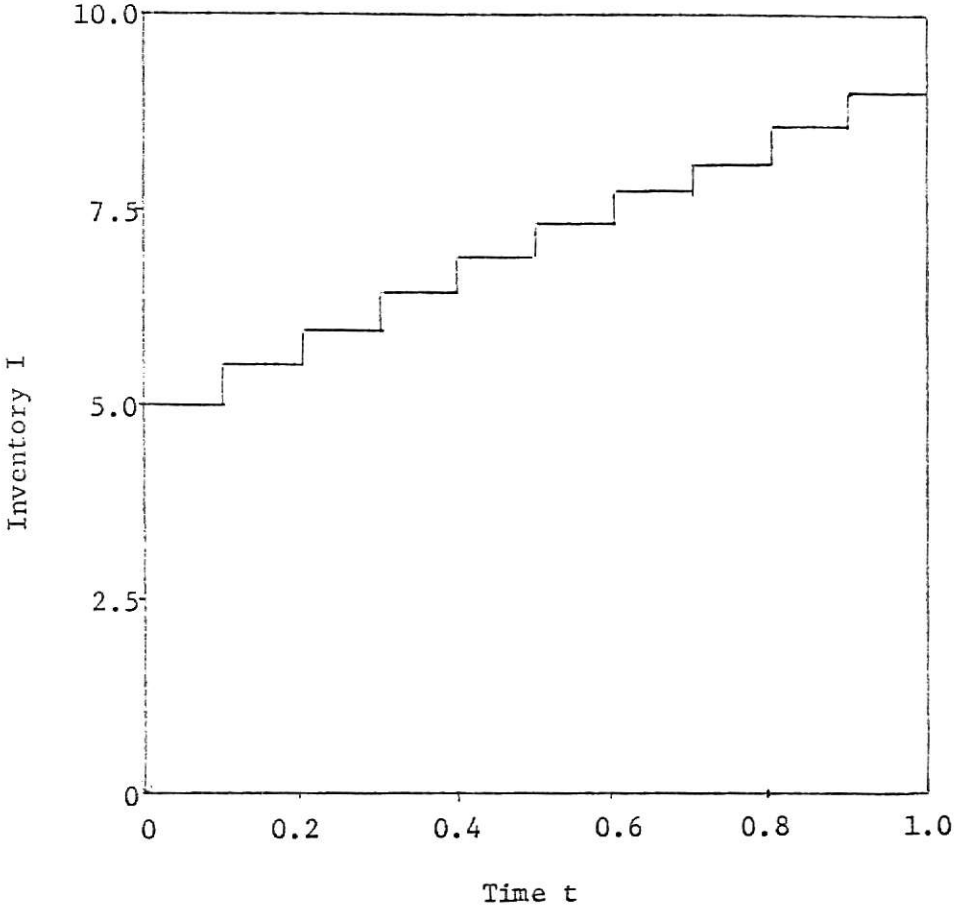


Fig. 3. Inventory for the 10-stage process

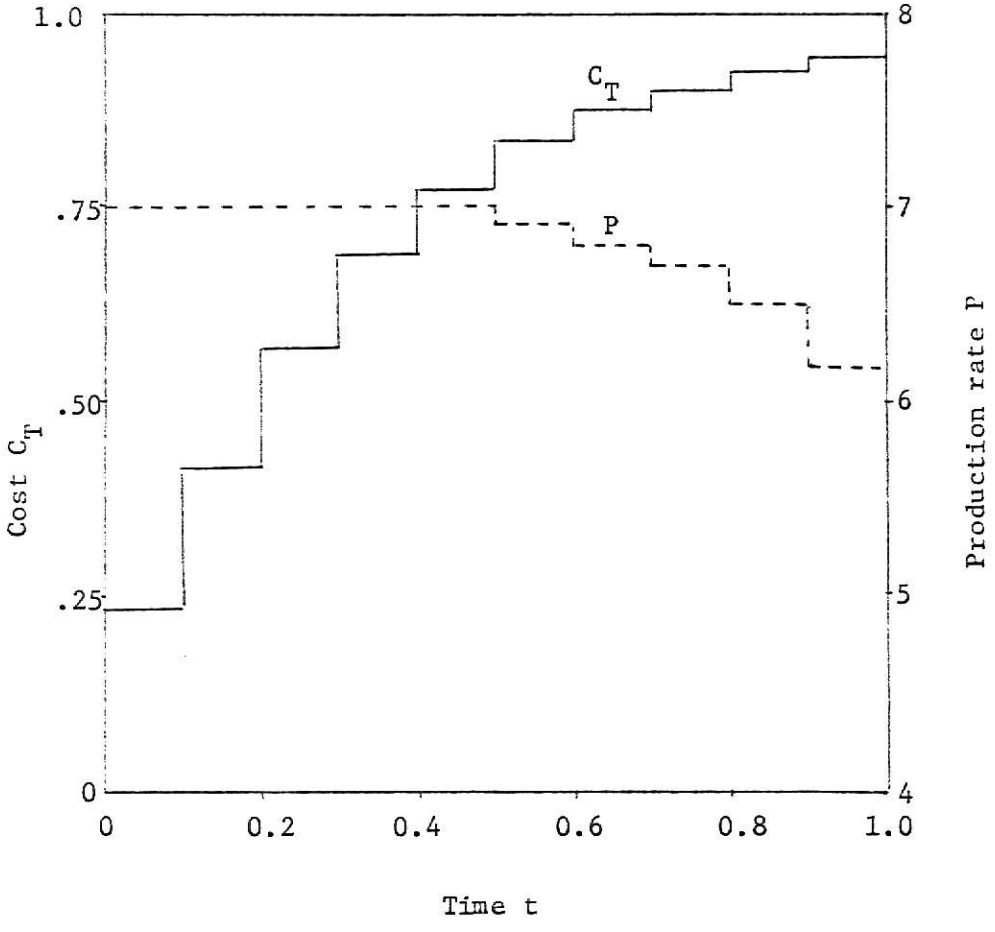


Fig. 4. Cost and production rate for the 10-stage process

case as will be evident in the solution of the second model discussed later.

This model has been solved by Lee and Shaikh [12] using the gradient technique and the value of the optimal cost obtained in their case was 0.943. Thus, there is a minor difference which is reasonable. This variation might be due to the difference in the two techniques employed in obtaining a solution to the above model.

5.2 Test Problem No. 2

An inventory and advertising model with two state variables

This model is an extension of a model formulated by Teichroew [18]. Consider a marketing situation where only a certain number of persons in a group possess certain information about a firm's product. Suppose that the total number of persons in the group under consideration remains constant and that diffusion of information occurs only through personal contract. The number of "contacts" made by an average person in an arbitrary unit of time is given by a "contact coefficient", which is a fixed pure number and is same for allmembers of the group. In a contact, the contactee receives information if he does not already have it; if he already has it, the contact is wasted in so far as increasing the number of informed persons is concerned.

Let $K(0)$ = number of informed persons in the group at time t_0

N = total number of persons in the group

c = contact coefficient; the number of contacts made by one informed person per unit time, and

$K(t)$ = number of informed persons at time t.

Then $K(t)/N$ = proportion of informed persons at time t

$1-K(t)/N$ = proportion of uninformed persons at time t, and

$cK(t)dt$ = contacts made during an interval of time dt.

The increase in total number of informed persons, $dK(t)$, during a short time interval dt is the product of the number of contacts and the proportion of uninformed persons.

Thus $$dK(t) = cK(t)\, dt[1 - K(t)/N]$$

and the process is governed by the differential equation

$$\frac{dK(t)}{dt} = cK(t)\ [1 - K(t)/N] \tag{1}$$

Suppose that the firm can influence the number of contacts by spending money on advertising. In particular, it can increase the number of contacts made by the informed persons (above the ones included in c) by an additional number A per unit of time. Equation (1) then becomes

$$\frac{dK(t)}{dt} = K(t)\ (c + A(t))\ [1 - K(t)/N] \tag{2}$$

If each successful contact results in the sale of n units of the firm's product and if Q(t) denotes the sale at time t, then

$$Q(t) = nK(t) \tag{3}$$

Assuming n = 1 and substituting eq. (3) in (2) we obtain

$$\frac{dQ(t)}{dt} = Q(t)\ (c + A(t))\ [1 - Q(t)/N] \tag{4}$$

Next, the rate of change of the firm's inventory is given by

$$\frac{dI(t)}{dt} = P(t) - Q(t) \tag{5}$$

where P(t) = production rate at time t.

The production rate is given by a linear relation

$$P(t) = a + bt \tag{6}$$

where a and b are known constants. This assumption is used for the purpose of simplifying the model by avoiding a second control variable. The firm's objective is to maximize the profit

$$P_T = \int_0^T [F\ Q(t) - C_I\ (P_I - I(t))^2 - C_A\ A\ Q(t)]\ dt \qquad (7)$$

where P_T is the total net profit, F is the revenue from sale of one unit of the product, and C_I is the inventory carrying cost. P_I can be considered as the available storage capacity for the inventory and C_A is the cost of advertising.

Equations (4) through (7) entirely represent the system. It has two state variables I and Q and one control variable A. To solve this model by the conjugate gradient technique, the systems equations are approximated by difference equations.

Thus, equation (4) is reduced to

$$Q(t + \Delta t) = Q(t) + Q(t)\ (c + A(t))\ [1 - Q(t)/N]\ \Delta t \qquad (8)$$

If this equation is used as it is, $Q(t + \Delta t)$ can exceed N which is theoretically impossible. To avoid this difficulty, the term $(1 - Q(t)/N)$ is replaced by $(1 - Q(t + \Delta t)/N)$. This change eventually leads to the expression

$$Q(t + \Delta t) = \frac{Q(t)[1 + (c + A(t))\Delta t]}{1 + (c + A(t))\ Q(t)\Delta t/N} \qquad (9)$$

Equation (5) is reduced to

$$I(t + \Delta t) = I(t) + P(t) - Q(t)\Delta t \qquad (10)$$

To obtain the profit, eq. (7) has to be computed over the limits $n\Delta t$ to $(n + 1)\Delta t$. If Δt is small, this integral is approximated by

$$\int_{n\Delta t}^{(n+1)\Delta t} [F\ Q(t) - C_I\ (P_I - I(t))^2 - C_A\ A\ Q(t)]dt$$

$$= [F\ Q(t) - C_I\ (P_I - I(t))^2 - C_A\ A\ Q(t)]\ \Delta t$$

The initial conditions and numerical values are given below:

a	= 70	N	= 150	C_I	= 0.15
b	= 100	t_0	= 0	P_I	= 50
c	= 2	T	= 1	I(0)	= 20
F	= 10	C_A	= 1.5	Q(0)	= 20

In addition, due to limited funds for advertising, the firm's management has imposed bounds on the control variable A which should not exceed 6.

Equations (9) and (10) represent constraints on the sales and inventory respectively. The method of eliminating these constraints is similar to that in the previous problem. Due to the relation of Q with A and the relation of I with Q and eventually with A, the chain rule is used to obtain the necessary differentiation. For example, the differentiation of the two state variables with respect to the control variable is given by

$$\frac{\partial Q(t+3\Delta t)}{\partial A(t)} = \frac{\partial Q(t+3\Delta t)}{\partial Q(t+2\Delta t)} \cdot \frac{\partial Q(t+2\Delta t)}{\partial Q(t+\Delta t)} \cdot \frac{\partial Q(t+\Delta t)}{\partial A(t)}$$

and

$$\frac{\partial I(t+3\Delta t)}{\partial A(t)} = \frac{\partial I(t+3\Delta t)}{\partial I(t+2\Delta t)} \cdot \frac{\partial I(t+2\Delta t)}{\partial Q(t+\Delta t)} \cdot \frac{\partial Q(t+\Delta t)}{\partial A(t)} +$$

$$\frac{\partial I(t+3\Delta t)}{\partial Q(t+2\Delta t)} \cdot \frac{\partial Q(t+2\Delta t)}{\partial Q(t+\Delta t)} \cdot \frac{\partial Q(t+\Delta t)}{\partial A(t)}$$

In this way, the gradient of the function can be obtained.

In order to solve this model by the maximization conjugate gradient algorithm, no changes had to be made in the original program, except the

Table 5. Initial and final points for a 5-stage process

	A_1	A_2	A_3	A_4	A_5	P_T
Initial point	1.000	1.000	1.000	1.000	1.000	442.991
Final point	2.379	2.097	1.505	0.878	0.318	524.764
Initial point	4.000	4.000	4.000	4.000	4.000	334.882
Final point	4.522	3.335	1.940	0.752	0.000	582.456
Initial point	6.000	6.000	6.000	6.000	6.000	84.199
Final point	4.472	2.988	1.508	0.505	0.000	584.239

Table 6. Initial and final points for a 10-stage process

	A_1	A_3	A_5	A_7	A_9	A_{10}	P_T
Initial point	1.000	1.000	1.000	1.000	1.000	1.000	466.693
Final point	3.892	3.080	1.949	0.650	0.000	0.000	628.191
Initial point	4.000	4.000	4.000	4.000	4.000	4.000	352.134
Final point	4.234	3.257	1.965	0.852	0.189	0.000	627.885
Initial point	6.000	6.000	6.000	6.000	6.000	6.000	74.240
Final point	4.390	3.011	1.463	0.499	0.091	0.000	631.497

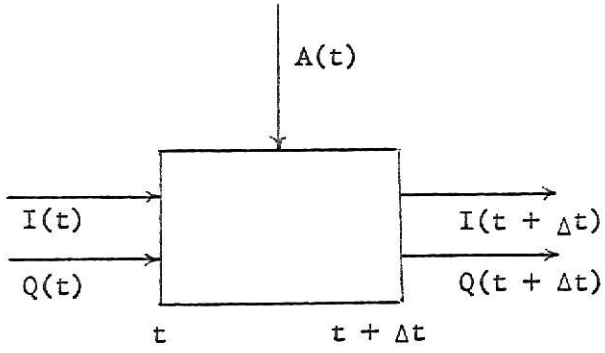
Table 7. Results for a 5-stage process

No. of iterations	Total no. of functional evaluations	No. of constraints in basis	A_1	A_3	A_5	P_T
	Initial point:	All A_i = 1.0, 4.0, 6.0				
0	1	1	2.400	2.400	2.400	447.421
1	2	1	4.029	2.424	0.728	542.021

Table 8. Results for a 10-stage process

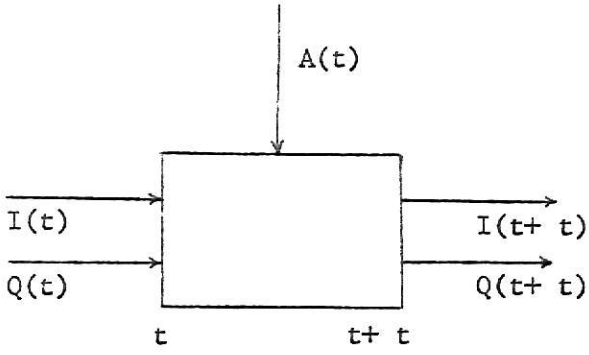
No. of iterations	Total no. of functional evaluations	No. of constraints in basis	A_1	A_5	A_{10}	P_T
	Initial point: All A_i = 1.0, 4.0, 6.0					
0	1	1	2.320	2.320	2.320	480.156
1	2	1	4.633	2.641	0.000	620.792
2	8	2	4.633	2.641	0.000	620.792

Table 9. 5-stage process



t	t+ Δt	I(t)	Q(t)	A(t)	I(t+Δt)	Q(t+Δt)	P(t)	P_T
0.0	0.2	20.000	20.000	4.029	30.000	38.005	70.000	18.078
0.2	0.4	30.000	38.005	3.327	40.399	61.809	90.000	77.246
0.4	0.6	40.399	61.809	2.424	50.037	85.373	110.000	185.900
0.6	0.8	50.037	85.373	1.492	58.963	103.756	130.000	344.550
0.8	1.0	58.963	103.756	0.728	68.212	116.427	150.000	542.021

Table 10. 10-stage process



t	t+Δt	I(t)	Q(t)	A(t)	I(t+Δt)	Q(t+Δt)	P(t)	P_T
0.0	0.1	20.000	20.000	4.633	25.000	30.563	70.000	-0.052
0.1	0.2	25.000	30.563	4.267	29.944	44.088	80.000	9.781
0.2	0.3	29.944	44.088	3.805	34.535	59.525	90.000	31.745
0.3	0.4	34.535	59.525	3.254	38.582	75.135	100.000	68.249
0.4	0.5	38.582	75.135	2.641	42.069	89.255	110.000	121.208
0.5	0.6	42.069	89.255	2.004	45.143	100.942	120.000	191.457
0.6	0.7	45.143	100.942	1.389	48.049	110.053	130.000	278.520
0.7	0.8	48.049	110.053	0.836	51.044	116.934	140.000	380.767
0.8	0.9	51.044	116.934	0.371	54.351	122.091	150.000	495.788
0.9	1.0	54.351	122.091	0.000	58.141	125.999	160.000	620.792

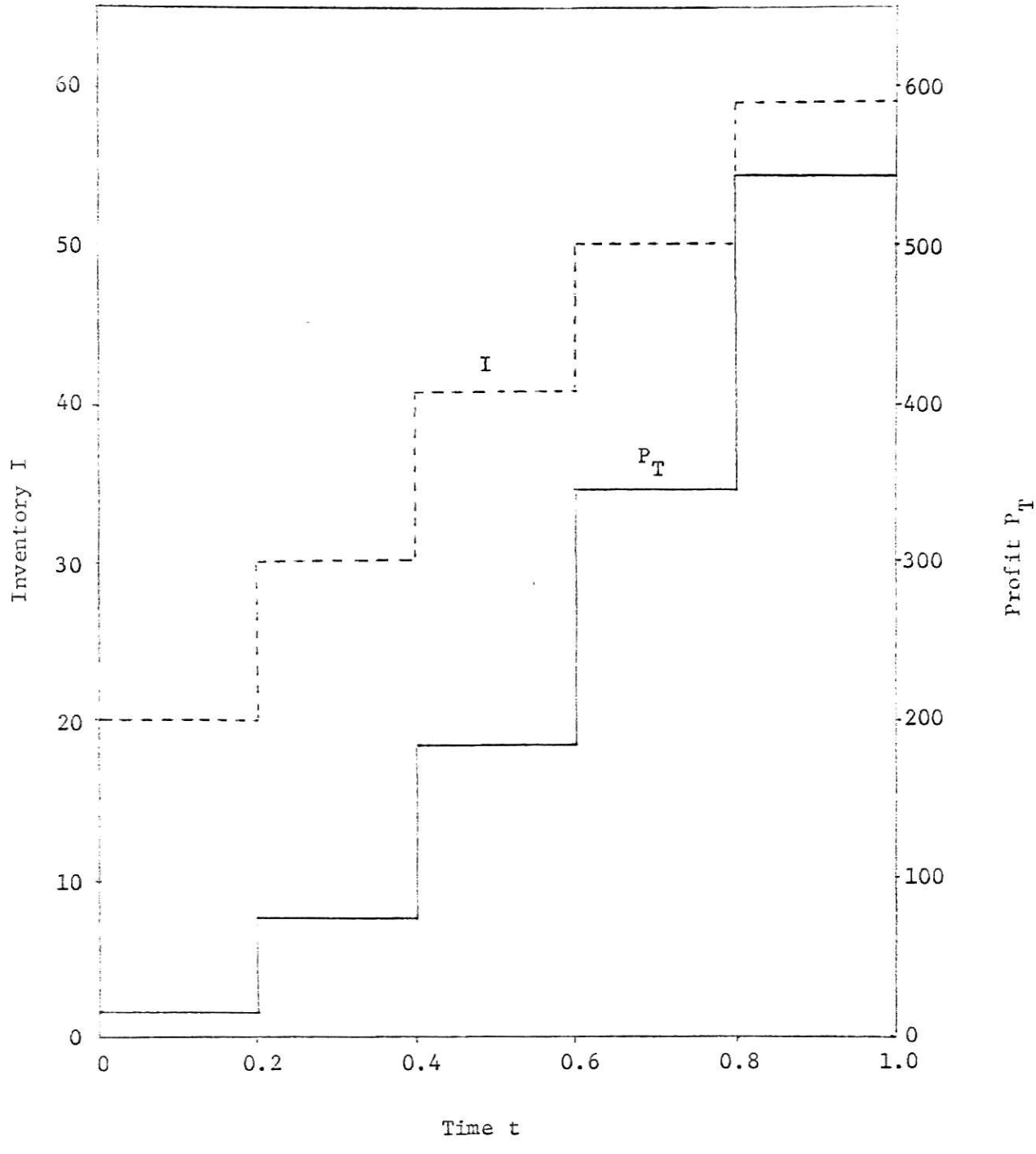


Fig. 5. Inventory and profit for the 5-stage process

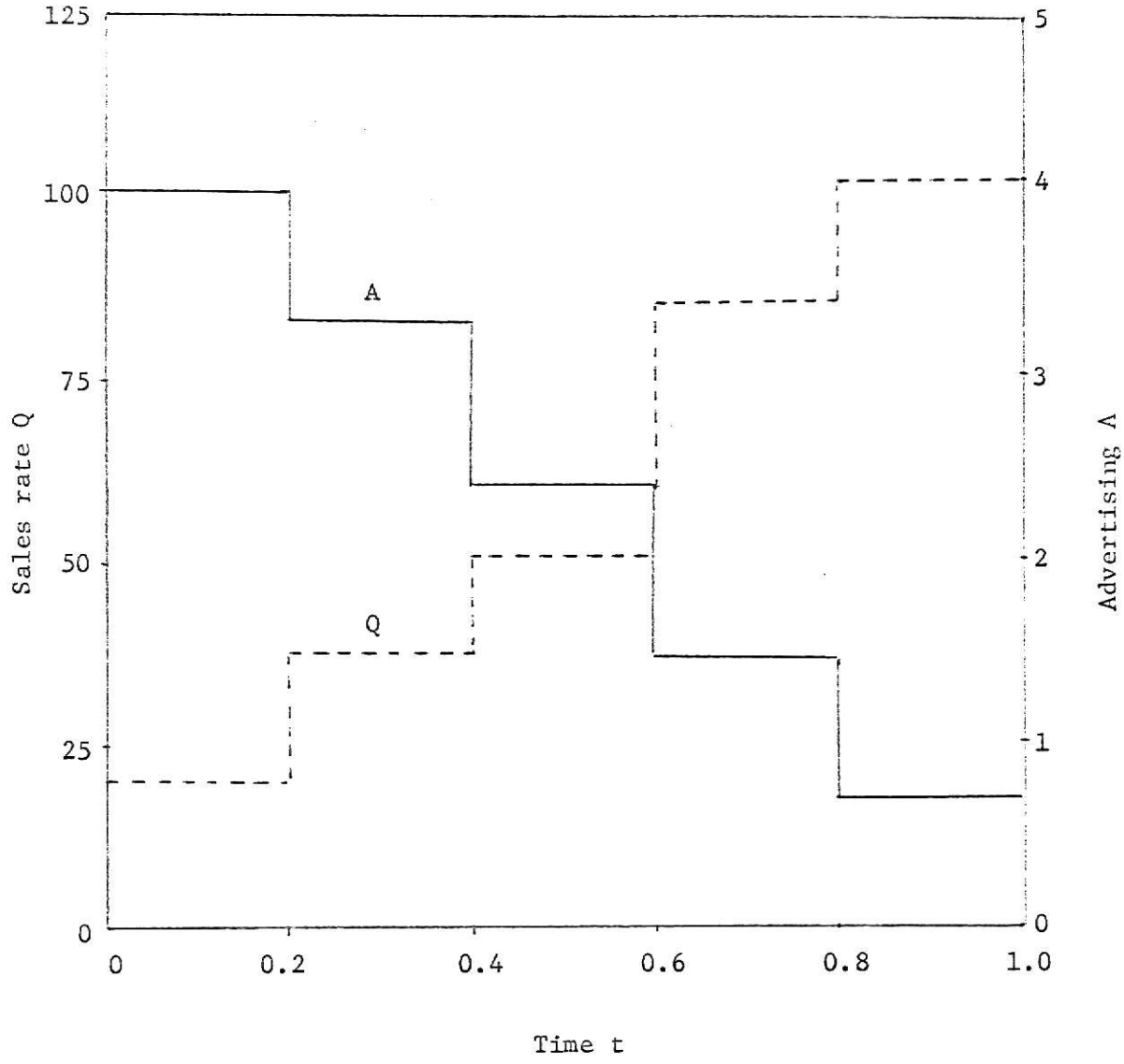


Fig. 6. Sales rate and advertising for the 5-stage process

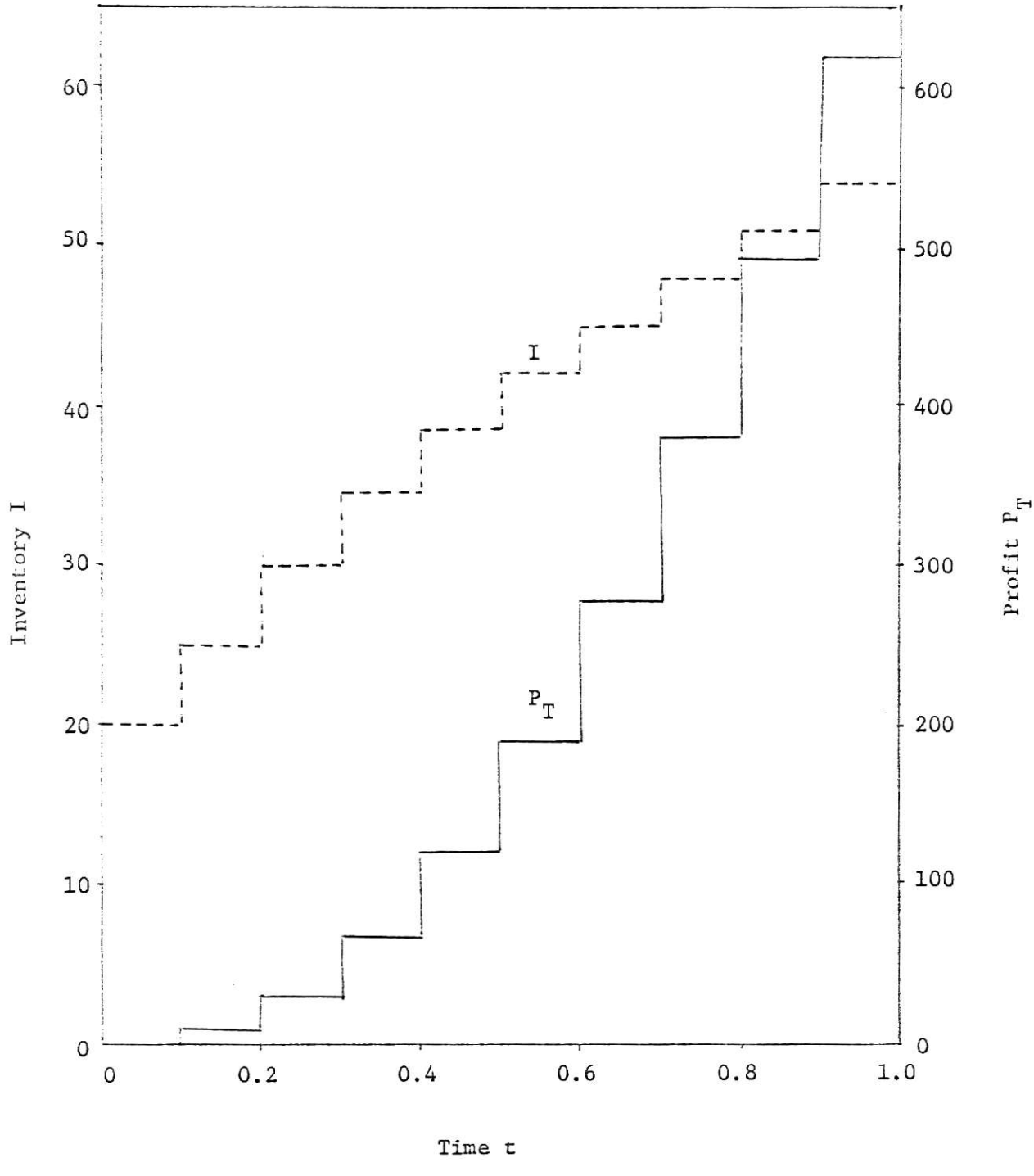


Fig. 7. Inventory and profit for the 10-stage process

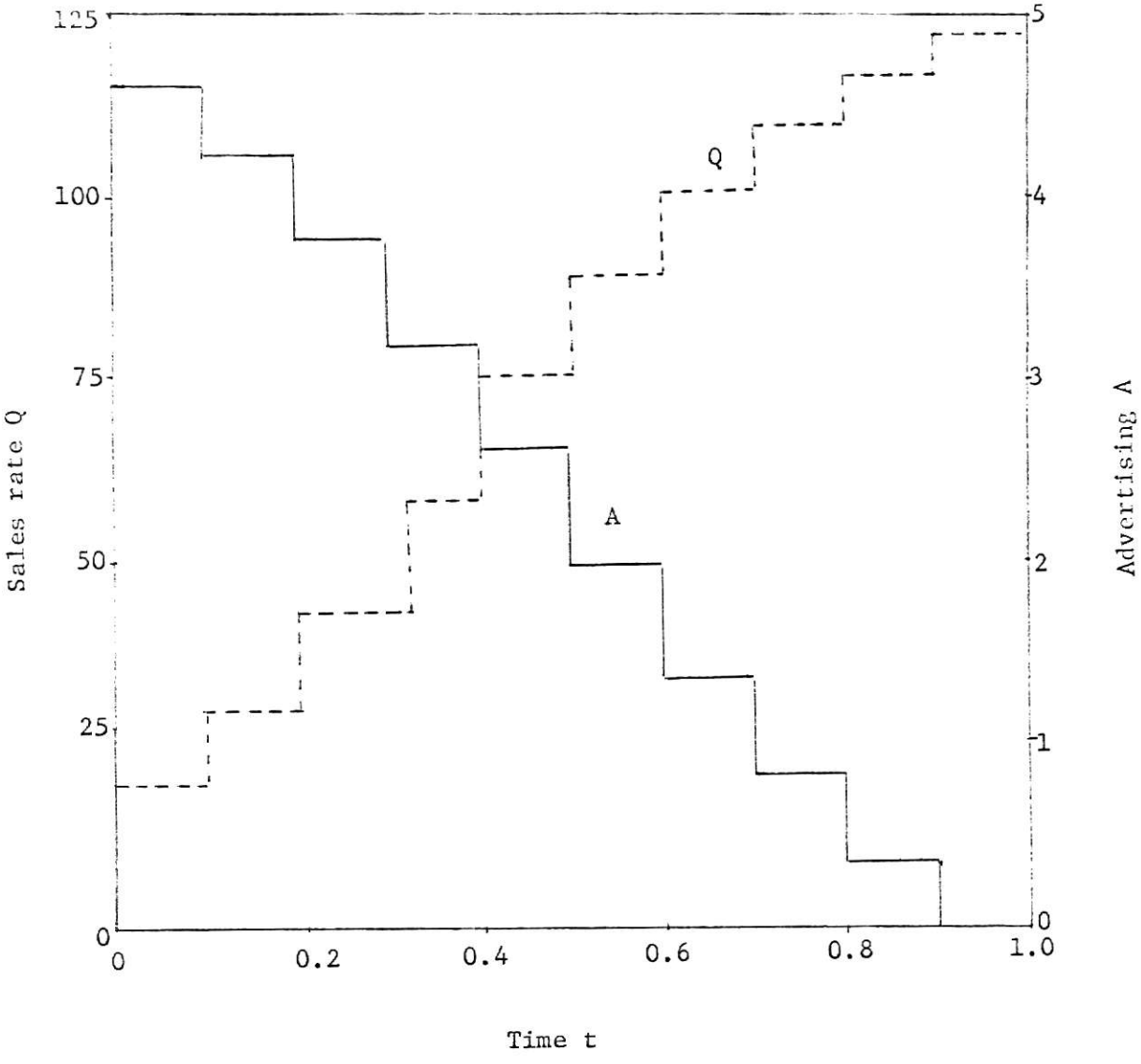


Fig. 8. Sales rate and advertising for the 10-stage process

subroutine FUNCT had to be designed to compute the function value, the gradient at any point, sales, inventory and advertising.

Like the previous model this model was solved for a 5-stage and a 10-stage process. It was however found that the problem failed to converge to the same results with different initial points. The starting points chosen and the results obtained in each case are given in Tables 5 and 6 for the two processes under consideration. The failure in obtaining consistent results might be due to the unstability of the problem. To overcome this difficulty, an equality constraint was imposed on the sum of the A at the various stages. The total was set equal to 12 in the case of a 5-stage process and 23.2 in the case of a 10-stage process. With such a constraint imposed, it was found that the feasible point chosen was the same irrespective of the initial points chosen in both the processes. Hence, the results obtained were identical for the process considered. The profit obtained in the case of a 5-stage process was 542.021 and in the 10-stage process it was 620.792. These results are given in Tables 7 and 8 for the two processes. The final results are tabulated in Tables 9 and 10. The graphs of time vs. inventory, profit, sales and advertising are given in Figs. 5 through 8.

A significant improvement in profit was noticed in the case of the 10-stage process. This was due to the fact that a 5-stage process with two state variables is a very approximate representation of the real situation. In light of the experience gained in trying to solve this model, it can be concluded that the calculation of the gradient was a most tedious job as both inventory and sales were functions of the control variable.

6. COMPUTATIONAL ASPECTS

6.1 Computational Results

For many applications the number of functional evaluations is the most critical criterion of effectiveness, especially in management problems where each functional evaluation may consume considerable time due to complicated relationships between various variables. Frequently, the gradient cannot be obtained analytically and thus the evaluation of the function and gradient at a point requires (m + 1) computations. It is worth noting that the conjugate gradient method requires fewer functional evaluations than does Rosen's gradient projection method. The cubic interpolation scheme used in the former method requires probably fewer interpolations and hence fewer functional evaluations per step to locate the maximum of $f(\underline{x})$ along direction $\underline{s}^i$. The efficiency of this method however lies in the selection of the estimate of the maximum (minimum) of the function. No such priori estimate is used in the Rosen's gradient projection method or any other first order gradient method.

A better way of obtaining the relative effectiveness of this method in comparison with others is by taking the ratio of the number of steps and the ratio of the number of functional evaluations. In highly nonlinear problems this relative superiority will considerably increase. Aside from the merits listed above, the method suffers certain disadvantages.

It necessitates the calculation of the first partial derivatives to obtain the gradient at a point. It is frequently the case that it is laborious or practically impossible to calculate these derivatives and there is a definite need for optimization techniques which do not require them. Unlike other techniques, this technique is not quadratically convergent,

but locates the optimum in a finite number of steps. This, however, is more than compensated by the decrease in computational time.

6.2 Numerical Problems and Corrective Procedures

The major numerical difficulty which may arise in the use of the conjugate gradient technique is the build-up of round-off errors in the matrices $\underline{H}_q$, $\underline{P}_q$ and $(\underline{N}_q^T \underline{N}_q)^{-1}$. After adding or removing hyperplanes to or from the constraint basis the search direction $\underline{H}_q\, g(\underline{x})$ may have small components that do not lie in the subspace M_q. Similarly the matrices $\underline{P}_q$ and $(\underline{N}_q^T \underline{N}_q)^{-1}$ may be inaccurate. This difficulty is also encountered in the gradient projection method and is especially cumbersome if the hyperplanes in the constraint basis tend to be nearly linearly dependent.

Rosen has suggested an error squaring procedure [15], which may also be used to advantage with the conjugate gradient method. If as a result of rounding errors in $\underline{H}_q$ (similar to those that occur in $(\underline{N}_q^T \underline{N}_q)^{-1}$), a point $\underline{x}^0$ is obtained at which some constraint in the basis is not satisfied as an equality, i.e. $|\underline{n}_q^T \underline{x}^0 - b_q| = \delta$ and $||\underline{N}_q^T \underline{x}^0 - \underline{b}|| \leq \delta$, following Rosen's procedure results in a point $\underline{x}^1$ at which $||\underline{N}_q^T \underline{x}^1 - \underline{b}|| \leq (\delta^2)$. Point $\underline{x}^1$ is given by

$$\underline{x}^1 = \underline{x}^0 - \underline{N}_q\, (\underline{N}_q^T \underline{N}_q)^{-1}\, (\underline{N}_q^T \underline{x}^0 - \underline{b})$$

Therefore $(\underline{N}_q^T \underline{x}^1 - \underline{b}) = (\underline{I} - \underline{N}_q^T \underline{N}_q\, (\underline{N}_q^T \underline{N}_q)^{-1})\, (\underline{N}_q^T \underline{x}^0 - \underline{b})$.

Another numerical difficulty may arise when the gradient vector must be approximated by finite differences. Since the iterative scheme for improving $\underline{H}_q$ requires the vector $\underline{y}^i$ which is the difference of two gradient vectors

$\underline{g}^{i+1}$ and $\underline{g}^{i}$, the errors in the estimation of the gradient may be compounded. If the numerical variations in the finite difference estimate of the gradient are too large, the method could, in fact, become unstable.

6.3 Extension of the algorithm to handle nonlinear constraints

The conjugate gradient technique is designed to solve only nonlinear programming problems characterized by linear constraints. This is often a severe limitation. However, combination of this method together with other techniques can lead to algorithms which will solve the more general nonlinear programming problem. One approach is to linearize piecewise all nonlinear constraints and use the conjugate gradient method to solve the resulting problem. This approach can be most easily applied to seperable nonlinear constraints i.e., all nonlinear terms are functions of one variabe [10,14]. However, even in this case the extended set of constraints can become quite large.

7. ACKNOWLEDGEMENT

The author would like to thank Dr. E. S. Lee for providing expert guidance, constructive criticism, helpful suggestions and constant encouragement in the preparation of this master's report.

8. REFERENCES

1. Bodewig, E., Matrix Calculus, Interscience, New York, 1956, pp. 36-38.
2. Carroll, C. W., "The Created Response Surface Technique for Optimizing Nonlinear Restrained Systems," Operations Research, Vol. 9, No. 2, 1961, pp. 169-184.
3. Courant, R. and Hilbert, D., Methods of Mathematical Physics, Interscience, New York, Vol. 1, 1953, pp. 34-36.
4. Davidon, W. C., "Variable Metric Method for Minimization," AEC Research and Development Report, ANL - 5990 Rev., 1959.
5. Fletcher, R. and Powell, M. J. D., "A rapidly convergent descent method for minimization," The Computer Journal, Vol. 6, 1963, pp. 163-168.
6. Fletcher, R. and Reeves, C. M., "Function Minimization by Conjugate Gradients," The Computer Journal, Vol. 7, No. 2, 1964, pp. 149-154.
7. Goldfarb, D., "A Conjugate Gradient Method for Nonlinear Programming," Ph. D. Thesis, Dept. of Chem. Engg., Princeton University, 1966.
8. Goldfarb, D., "Extension of Davidon's Variable Metric Method to Maximization under Linear Inequality and Equality constraints," SIAM J. Appl. Math., submitted for publication, 1968.
9. Goldfarb, D. and Lapidus, L., "Conjugate Gradient Method for Nonlinear Programming with Linear Constraints," I & EC Fundamentals, Vol. 7, No. 1, 1968, pp. 142-151.
10. Hadley, G., Nonlinear and Dynamic Programming, Addison - Wesley, Reading, Massachusetts, 1964.
11. Lee, E. S., "Search Techniques," Lecture notes, Kansas State University, Manhattan, Kansas, 1967.

12. Lee, E. S. and Shaikh, M. S., "Optimal Production Planning by a Gradient technique I. First variations," an unpublished paper, Kansas State University, Manhattan, Kansas, 1968.

13. Merrill, R. P., "Gradient Projection 7090," SHARE Program Library Report No. 3430, 1966.

14. Miller, C. E., The Simplex Method for Local Seperable Programming, Recent Advances in Mathematical Programming, McGraw-Hill, New York, 1963, pp. 89-100.

15. Rosen, J. B., "The Gradient Projection Method for Nonlinear Programming, Part I. Linear Constraints," J. SIAM, Vol. 8, No. 1, 1960, pp. 181-217.

16. Rosenbrock, H. H., "An Automatic Method for Finding the Greatest or Least Value of a Function," The Computer Journal, Vol. 3, 1960, pp. 175-184.

17. Spang, H. A., "A review of Minimization Techniques for Nonlinear Functions," SIAM Review, Vol. 4, No. 4, 1962, pp. 343-365.

18. Teichroew, D., An introduction to Management Science Deterministic Models, John Wiley, New York, 1966.

APPENDIX A

RECURSION RELATIONS FOR OBTAINING THE INVERSE MATRIX

In the course of an optimization calculation using the conjugate gradient algorithm the inverse matrix $(\underline{N}_q^T \underline{N}_q)^{-1}$ is necessary at each iteration. From the outline of the algorithm in section 3, it can be seen that either a new hyperplane is added to or one of the previous hyperplanes is dropped from the constraint basis. For a particular set of vectors $\underline{n}_i$, i = 1,2,----------,q it is always possible to form the matrix $\underline{N}_q^T \underline{N}_q$ first and then invert it to obtain $(\underline{N}_q^T \underline{N}_q)^{-1}$. However, this will involve porhibitive computations especially for large problems.

In order to overcome this difficulty Rosen [15] formulated two recursion relations which permit a hyperplane to be dropped from $(\underline{N}_q^T \underline{N}_q)^{-1}$ with approximately q^2 multiplications and divisions and a hyperplane to be added to $(\underline{N}_{q-1}^T \underline{N}_{q-1})^{-1}$ with approximately $2q^2 + mq$ multiplications and divisions.

In particular, suppose that a (q x q) inverse matrix $(\underline{N}_q^T \underline{N}_q)^{-1}$ is partitioned as

$$(\underline{N}_q^T \underline{N}_q)^{-1} = \underline{B} = \begin{bmatrix} \underline{B}_1 & \underline{B}_2 \\ \underline{B}_3 & \underline{B}_4 \end{bmatrix}$$

where $\underline{B}_1$, $\underline{B}_2$, $\underline{B}_3$ and $\underline{B}_4$ are (q-1 x q-1), (q-1 x 1), (1 x q-1) and (1 x 1) matrices respectively.

Then the required expression for $(\underline{N}_{q-1}^T \underline{N}_{q-1})^{-1}$ in terms of the partitions $\underline{B}_1$, $\underline{B}_2$, $\underline{B}_3$ and $\underline{B}_4$ of $\underline{B}$ from the standard formulas for computing the inverse of a matrix by partitioning (7, 10, 14) is given by

$$(\underline{N}_{q-1}^{T}\,\underline{N}_{q-1})^{-1} = \underline{B}_1 - \underline{B}_2\,\underline{B}_4^{-1}\,\underline{B}_3$$

The procedure for adding a hyperplane to the constraint basis is now described. It is assumed that $(\underline{N}_{q-1}^{T}\,\underline{N}_{q-1})^{-1}$ is known and we wish to compute $(\underline{N}_q^{T}\,\underline{N}_q)^{-1}$ on adding the qth hyperplane.

Compute

$$A_0 = ||\underline{P}_{q-1}\,\underline{n}_q||^2 \tag{1}$$

$$\underline{B}_1 = (\underline{N}_{q-1}^{T}\,\underline{N}_{q-1})^{-1} + A_0^{-1}\,\underline{r}_{q-1}\,\underline{r}_{q-1}^{T}$$

$$\underline{B}_2 = -A_0^{-1}\,\underline{r}_{q-1}$$

$$\underline{B}_3 = \underline{B}_2^{T} \tag{2}$$

$$\underline{B}_4 = A_0^{-1}$$

where $$\underline{r}_{q-1} = (\underline{N}_{q-1}^{T}\,\underline{N}_{q-1})^{-1}\,(\underline{N}_{q-1}^{T}\,\underline{n}_q) \tag{3}$$

and $$\underline{P}_{q-1}\underline{n}_q = \underline{n}_q - \underline{N}_{q-1}\,\underline{r}_{q-1} \tag{4}$$

Thus the procedure is as follows:

1. Compute $\underline{r}_{q-1}$ from (3) as two matrix multiplications
2. Compute A_0 from (4) and (1)
3. Form

$$(\underline{N}_q^{T}\,\underline{N}_q)^{-1} = \begin{bmatrix} \underline{B}_1 & \underline{B}_2 \\ \underline{B}_3 & \underline{B}_4 \end{bmatrix}$$

where the matrices $\underline{B}_i$, i = 1,2,---,4 are given by (2).

Another useful recursion relation is one which computes $\underline{P}_q$ using $\underline{P}_{q-1}$ and $\underline{n}_q$ and is given by

$$\underline{P}_q = \underline{P}_{q-1} - \frac{\underline{P}_{q-1}\,\underline{n}_q\,\underline{n}_q^T\,\underline{P}_{q-1}}{\underline{n}_q^T\,\underline{P}_{q-1}\,\underline{n}_q} \tag{5}$$

This follows from equation (2) section 3 on realizing that $\underline{P}_q$ can be written as

$$\underline{P}_q = \underline{I} - [\underline{N}_{q-1}\ \underline{n}_q]\begin{bmatrix} \underline{B}_1 & \underline{B}_2 \\ \underline{B}_3 & \underline{B}_4 \end{bmatrix}\begin{bmatrix} \underline{N}_{q-1}^T \\ \underline{n}_q^T \end{bmatrix}$$

The above recursion relations facilitate the formulation of the matrices $(\underline{N}_s^T\,\underline{N}_s)^{-1}$ and $\underline{P}_s$ for a set of s linearly independent vectors $\underline{n}_i$, i = 1,2,-----,s, with a minimum of computation. The procedure for obtaining $(N_s^T\,N_s)^{-1}$ is initiated with $\underline{N}_1 = \underline{n}_1$ and $(\underline{N}_1^T\,\underline{N}_1)^{-1} = 1$. Then from equations (3) and (4), $\underline{r}_1 = \underline{n}_1^T\,\underline{n}_2$ and $\underline{P}_1\,\underline{n}_2 = \underline{n}_2 - \underline{r}_1\,\underline{n}_2$, etc. The procedure is applied recursively to form $(\underline{N}_s^T\,\underline{N}_s)^{-1}$ which require approximately $s^3 + \frac{1}{2}ms^2$ multiplications. The recursion relation (5) for $\underline{P}_q$ is started with $\underline{P}_0 = \underline{I}$ giving $\underline{P}_1 = \underline{I} - \underline{n}_1\,\underline{n}_1^T$, etc. The relation is used recursively to obtain $\underline{P}_s$ which approximately takes $2m^2$ s multiplications.

APPENDIX B

LOCATING THE MAXIMUM ALONG A LINE

The cubic interpolation scheme of Davidon [4] is used to locate the maximum in the conjugate gradient algorithm. Any method for obtaining the value of γ^i within the limits $0 \leq \gamma^i \leq \lambda^i$ which maximizes $f(\underline{x}^i + \gamma^i \underline{s}^i)$ along the direction $\underline{s}^i$ could be used, but Davidon's method was preferred.

Initially some point between $\underline{x}^i$ and $\underline{x}^i + \lambda^i \underline{s}^i$ must be selected for interpolation purposes. Since $\underline{H}_q$ is an approximation to $-\hat{\underline{P}}_q\, G^{-1}(\underline{x}^i)$ a point given by $\underline{y}^i = \underline{x}^i + \eta^i \underline{s}^i$ where

$$\eta^i = \min \left\{ 1, \frac{2[f(\underline{x}^i) - f^*]}{(\underline{g}^i)^T \underline{s}^i} \right\}$$

where f^* is some predicted upper bound of $f(x)$ in feasible region R. However η^i may exceed λ^i. Therefore, a suitable choice of the point is given by

$$\hat{\underline{x}}^i = \underline{x}^i + \tau \underline{s}^i$$

where $$\tau = \min \{\eta^i, \lambda^i\}$$

In as much as the interpolation is along a one-dimensional interval it is convenient to plot the function along this direction as a simple graph [4].

The values of $f(\underline{x}^i)$, $f(\hat{\underline{x}}^i)$, $g(\underline{x}^i)$ and $g(\hat{\underline{x}}^i)$ of the function and the gradient at the points $\underline{x}^i$ and $\hat{\underline{x}}^i$ are known. Interpolation for locating the maximum is obtained by choosing the "smoothest" curve satisfying the boundary conditions at $\underline{x}^i$ and $\hat{\underline{x}}^i$. Namely, the value of γ^i is sought at which the maximum is obtained for the curve which minimizes

$$\int_0^{\lambda^i} \left\{ \frac{d^2 f}{d\gamma^{i\,2}} \right\}^2 d\gamma^i$$

over the curve.

This results in a cubic equation and its slope at any $\gamma^i (0 \leq \gamma^i \leq \lambda^i)$ is given by

$$g_s(\gamma^i) = g_s - \frac{2\gamma^i}{\lambda^i}(g_s + z) + \frac{(\gamma^i)^2}{(\lambda^i)^2}(g_s + \hat{g}_s + 2z) \qquad (1)$$

where $z = 3/\tau\,(f(\underline{x}^i) - f(\underline{x}^i)) + g_s + \hat{g}_s$

and $g_s = (\underline{s}^i)^T \underline{g}^i$

$\hat{g}_s = (\underline{s}^i)^T \hat{\underline{g}}^i$

The root of equation (1) that corresponds to a maximum lies between 0 and 1 in virtue of the fact that $g_s > 0$ and either $\hat{g}_s < 0$ or $z > g_s + \hat{g}_s$. It can be expressed as

$$\gamma^i = \tau(1 - d)$$

where $d = \dfrac{w - g_s + z}{g_s - \hat{g}_s + 2w}$

and $w = \sqrt{z^2 - (g_s)(\hat{g}_s)}$

It is necessary to check that $f(\hat{\underline{x}}^i) - f(\underline{x}^i)$ and $\hat{g}_s$ are not both positive before interpolating so as to ensure that $f(\underline{x})$ attains a maximum between

x^i and $\hat{x}^i$. If any one of these quantities is positive, the procedure suggested by Davidon [4] is used. Interpolation with the help of the above formulas may yield a γ^i and a corresponding point $\underline{x}^i + \gamma^i \underline{s}^i$ at which the value of the objective function is less than equal to $\max\{f(\underline{x}^i), f(\hat{\underline{x}}^i)\}$. If such is the case interpolation must be repeated over a small range.

APPENDIX C

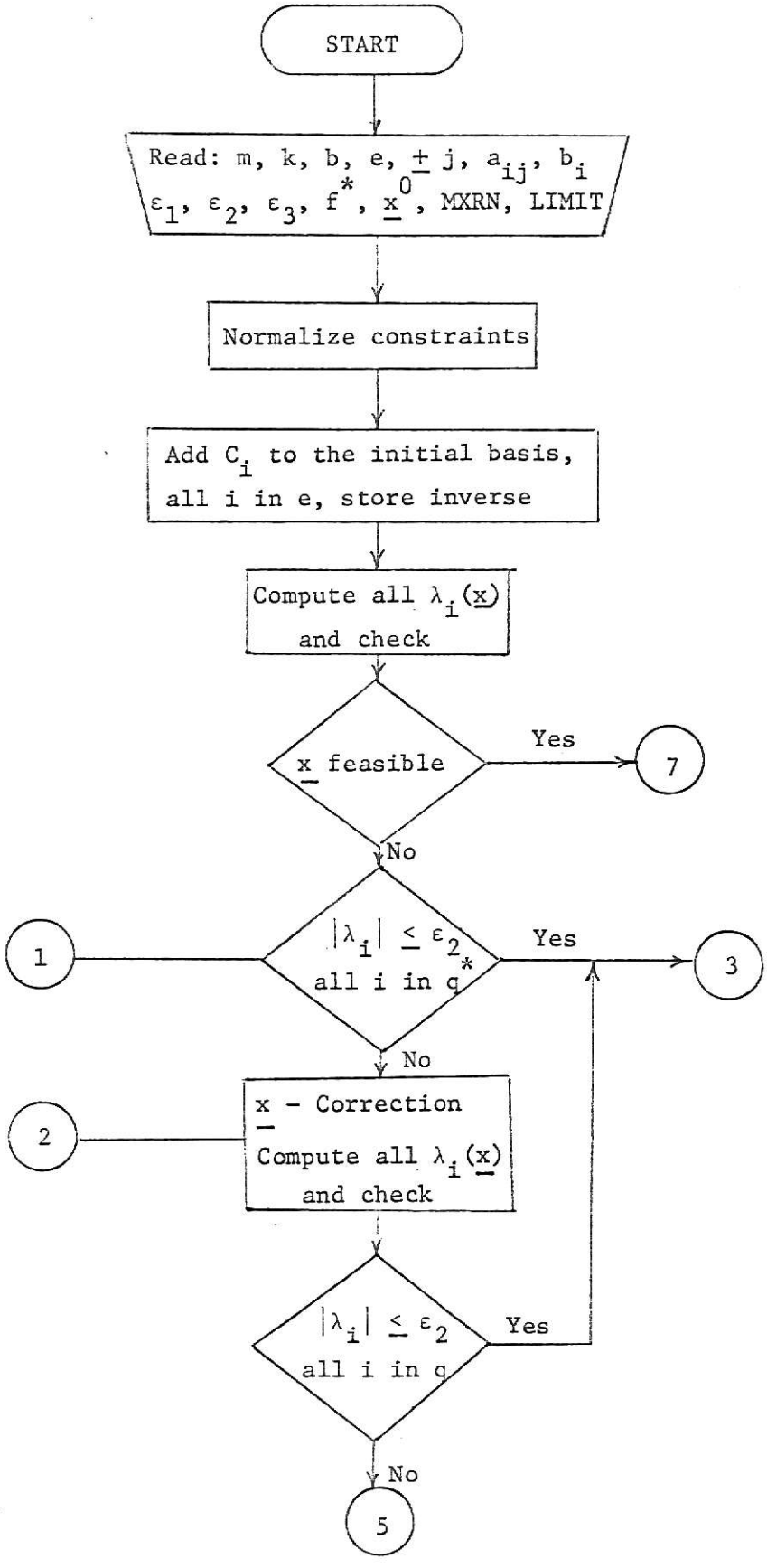
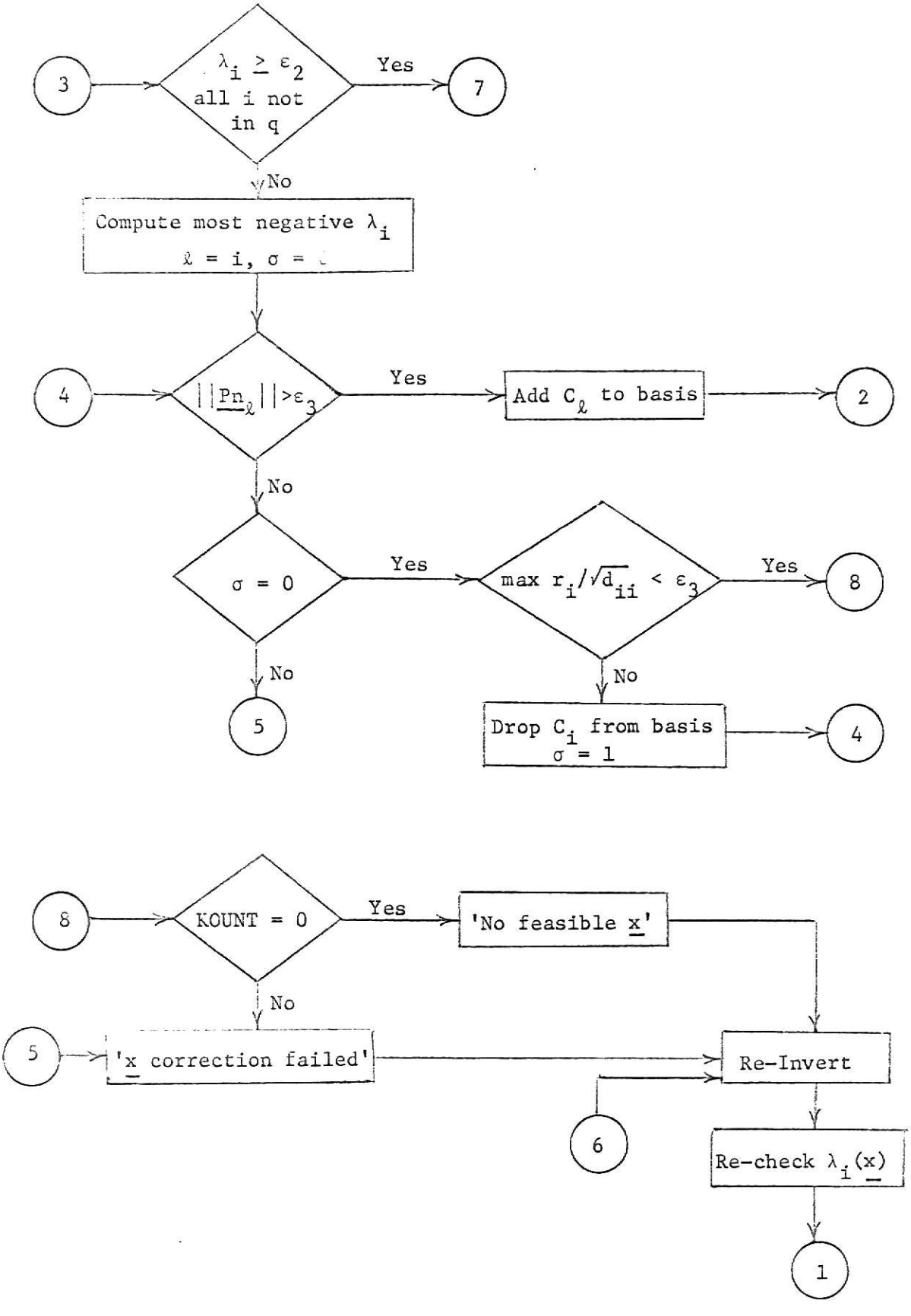
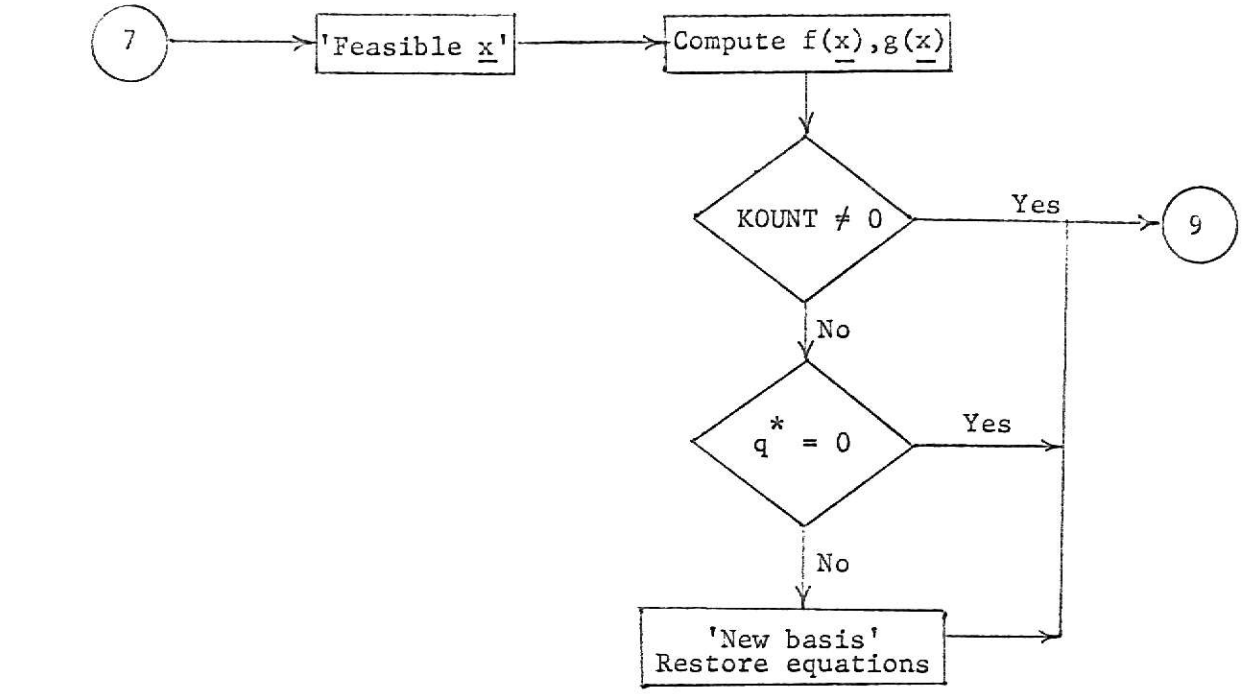
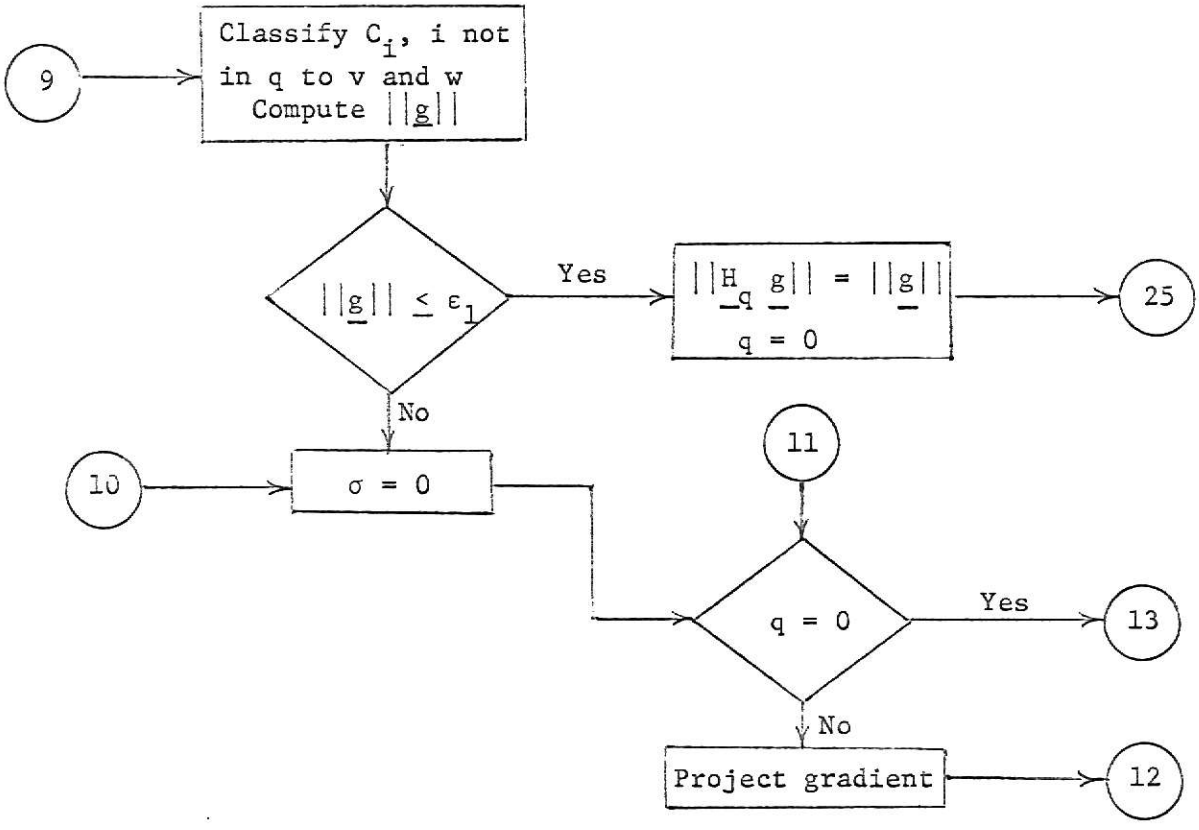
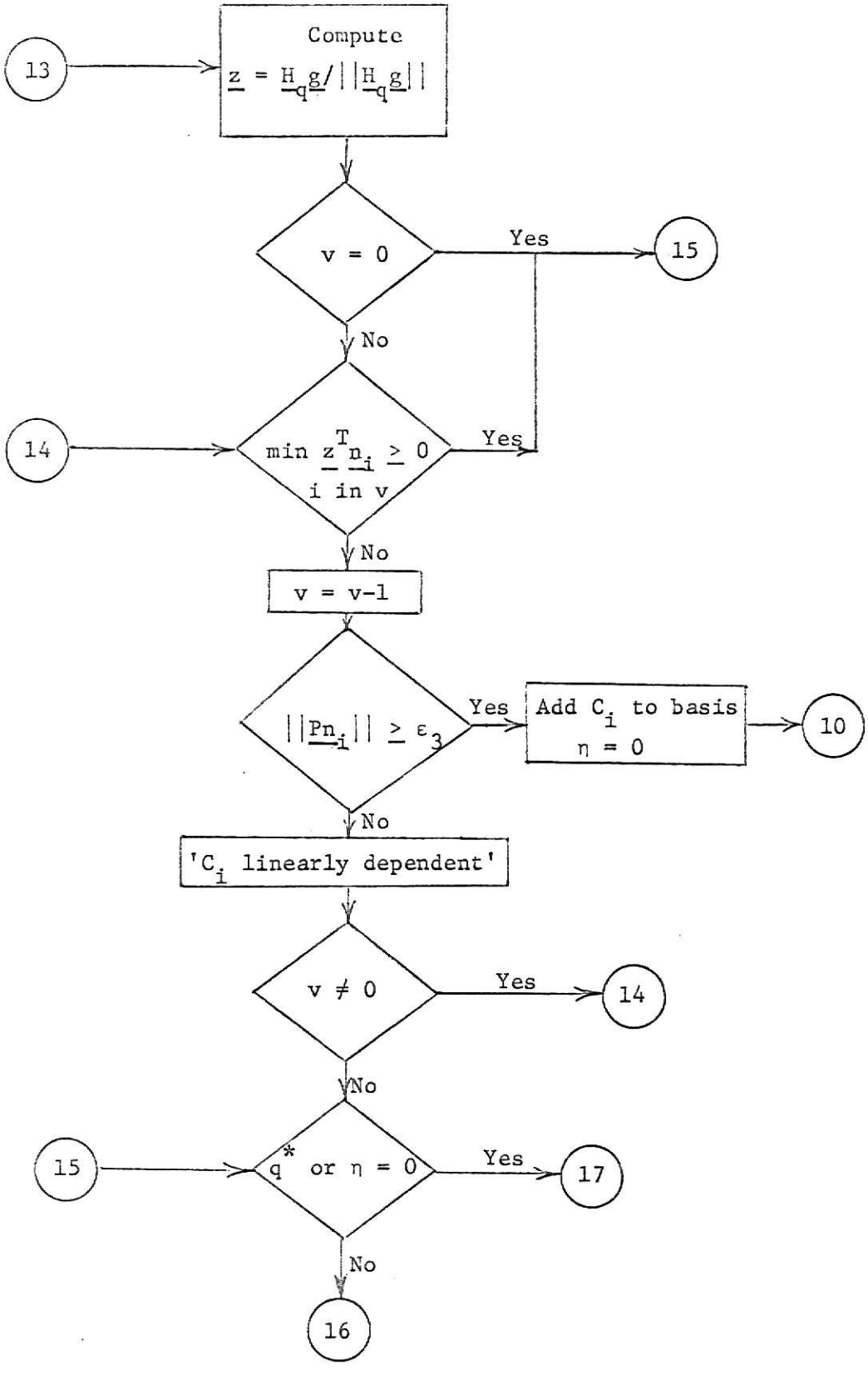


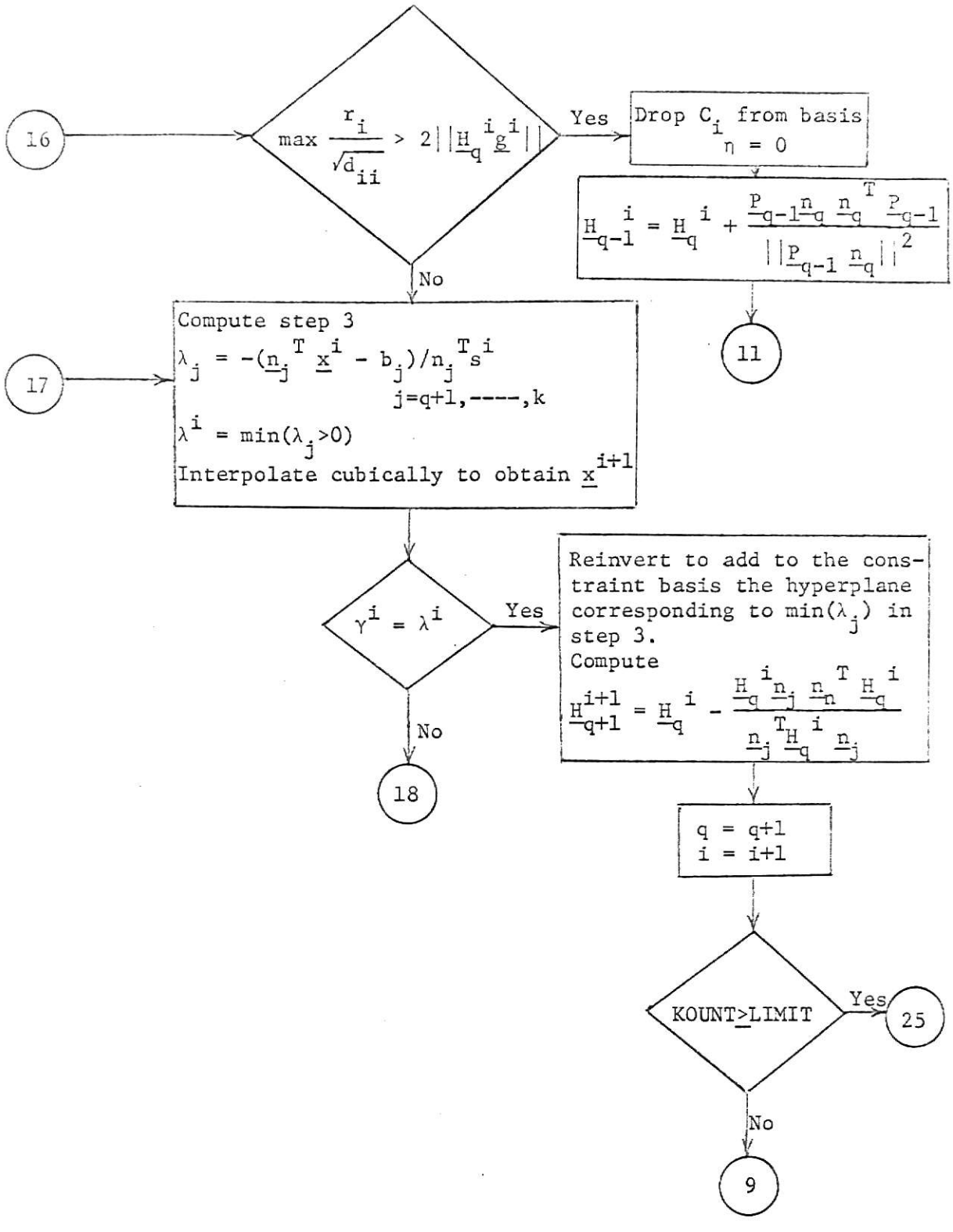
Fig. 9 Flow Chart for the Main Program of the Conjugate Gradient Method

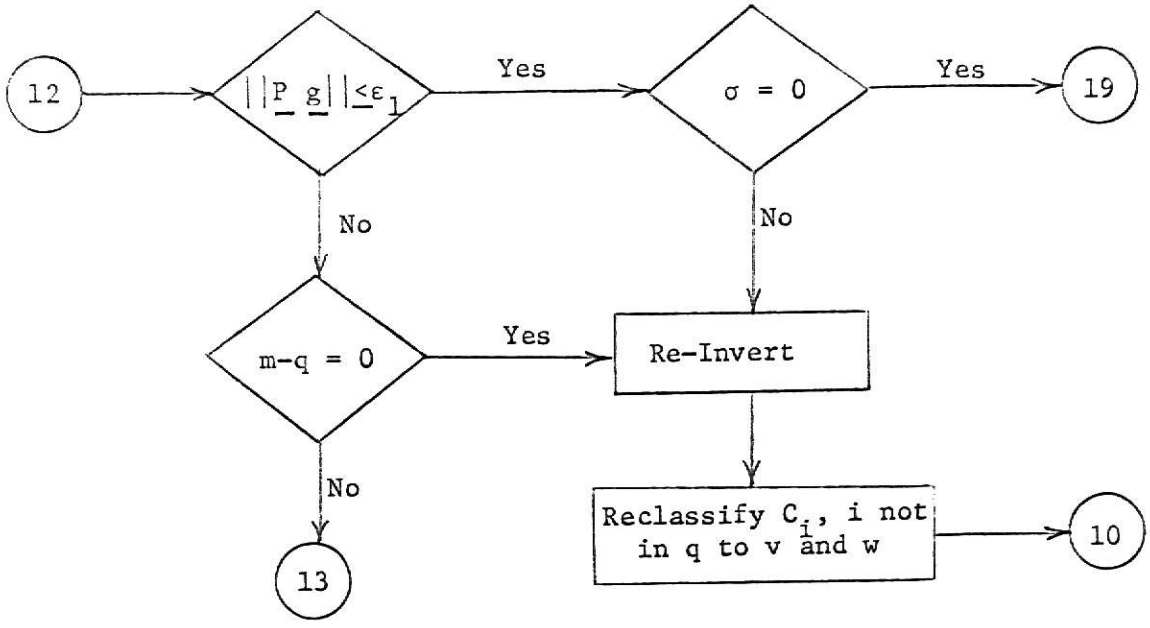

3
$\lambda_i \geq \varepsilon_2$ all i not in q
Yes
7
No
Compute most negative λ_i
$\ell = i$, $\sigma =$
4
$||\underline{Pn}_\ell|| > \varepsilon_3$
Yes
Add C_ℓ to basis
2
No
$\sigma = 0$
Yes
max $r_i/\sqrt{d_{ii}} < \varepsilon_3$
Yes
8
No
5
No
Drop C_i from basis
$\sigma = 1$
4
8
KOUNT = 0
Yes
'No feasible $\underline{x}$'
No
5
'$\underline{x}$ correction failed'
Re-Invert
6
Re-check $\lambda_i(\underline{x})$
1

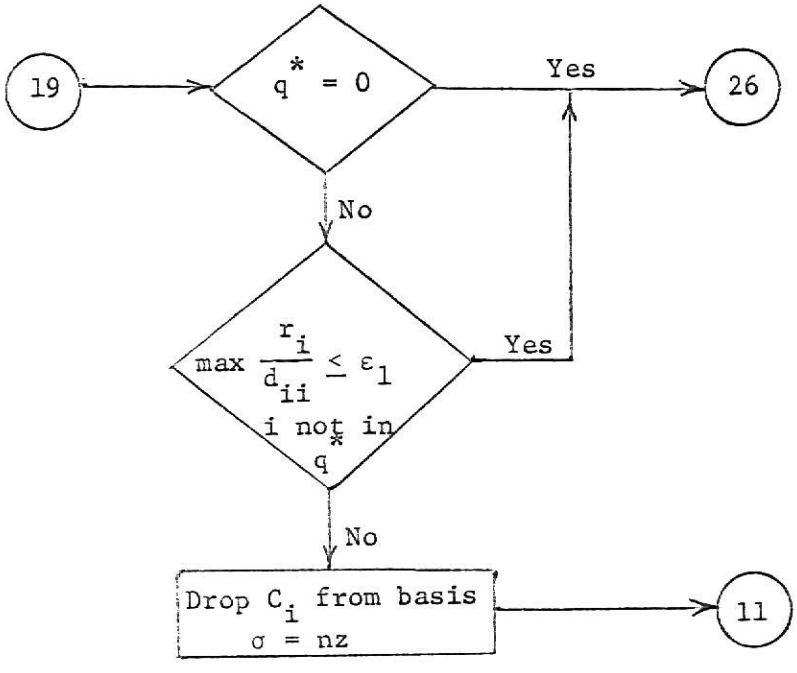

7
'Feasible x'
Compute f(x),g(x)
KOUNT ≠ 0
Yes
9
No
q* = 0
Yes
No
'New basis'
Restore equations

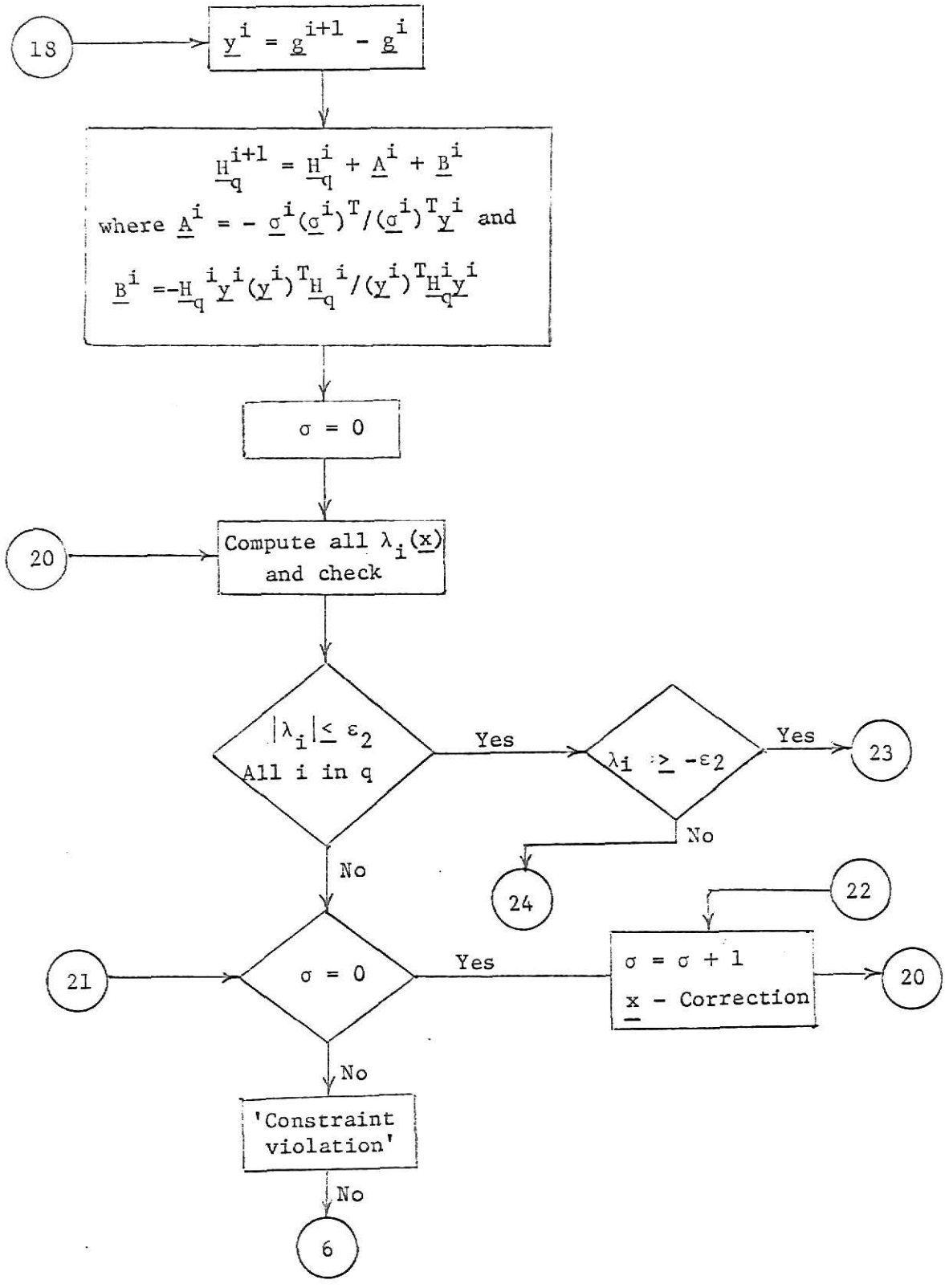

9
Classify C_i, i not
in q to v and w
Compute ||g||
||g|| ≤ ε_1
Yes
||H_q g|| = ||g||
q = 0
25
No
10
σ = 0
11
q = 0
Yes
13
No
Project gradient
12

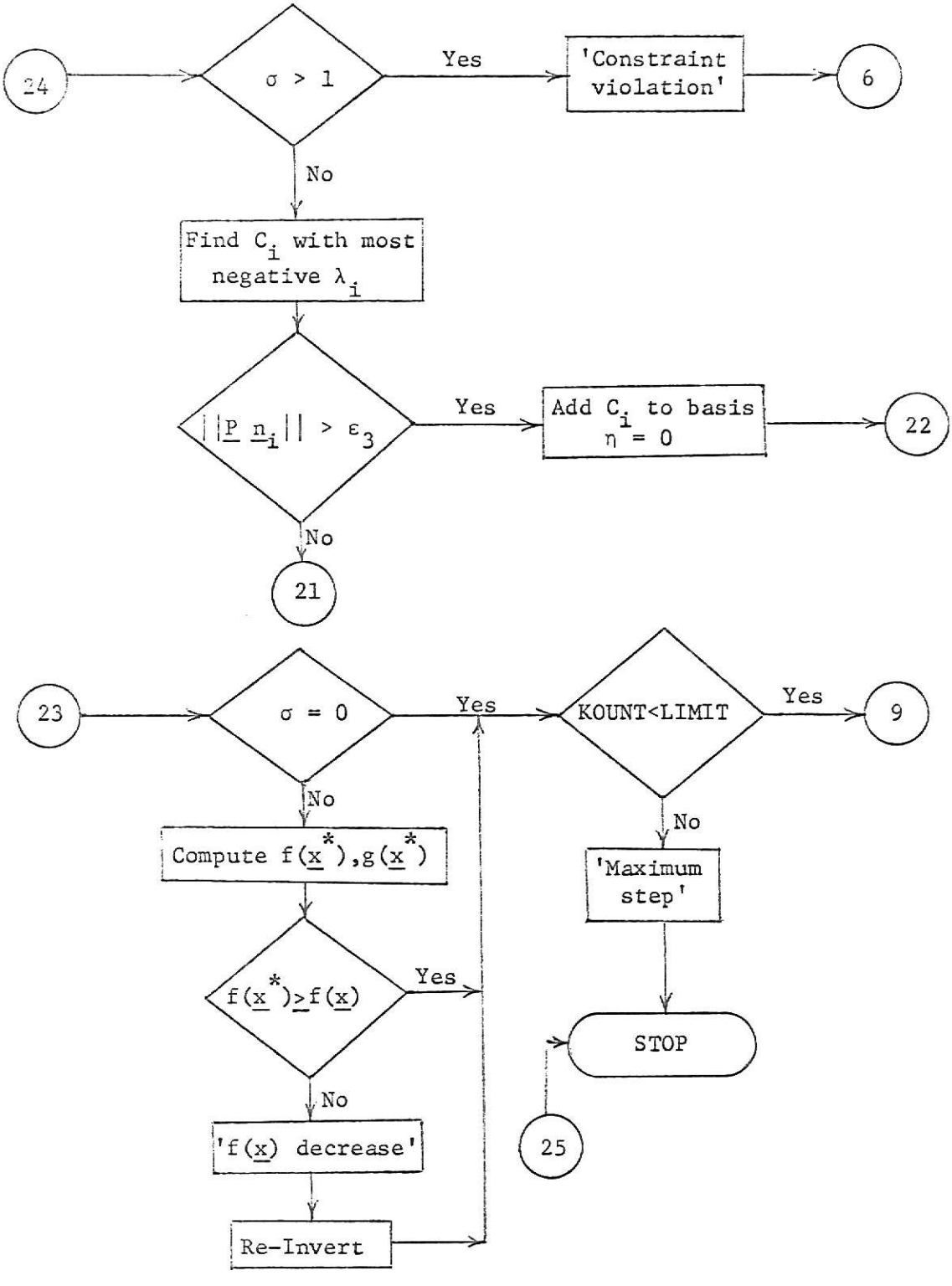

13
Compute
$\underline{z} = \underline{H}_q \underline{g} / ||\underline{H}_q \underline{g}||$
v = 0
Yes
15
No
14
min $\underline{z}^T \underline{n}_i \geq 0$
i in v
Yes
No
v = v-1
$||\underline{Pn}_i|| \geq \varepsilon_3$
Yes
Add C_i to basis
$\eta = 0$
10
No
'C_i linearly dependent'
$v \neq 0$
Yes
14
No
15
q^* or $\eta = 0$
Yes
17
No
16


16
$\max \frac{r_i}{\sqrt{d_{ii}}} > 2||H_q^i g^i||$
Yes
Drop C_i from basis
$\eta = 0$
$H_{q-1}^i = H_q^i + \frac{P_{q-1} n_q n_q^T P_{q-1}}{||P_{q-1} n_q||^2}$
11
No
17
Compute step 3
$\lambda_j = -(n_j^T x^i - b_j)/n_j^T s^i$
$j=q+1,----,k$
$\lambda^i = \min(\lambda_j > 0)$
Interpolate cubically to obtain x^{i+1}
$\gamma^i = \lambda^i$
Yes
Reinvert to add to the constraint basis the hyperplane corresponding to $\min(\lambda_j)$ in step 3.
Compute
$H_{q+1}^{i+1} = H_q^i - \frac{H_q^i n_j n_n^T H_q^i}{n_j^T H_q^i n_j}$
No
18
$q = q+1$
$i = i+1$
KOUNT≥LIMIT
Yes
25
No
9


12
$||P\ g||\leq\varepsilon_1$
Yes
$\sigma = 0$
Yes
19
No
No
$m-q = 0$
Yes
Re-Invert
No
Reclassify C_i, i not in q to v and w
10
13


19
$q^* = 0$
Yes
26
No
$\max \frac{r_i}{d_{ii}} \leq \varepsilon_1$
i not in q^*
Yes
No
Drop C_i from basis
$\sigma = nz$
11


18
$\underline{y}^i = \underline{g}^{i+1} - \underline{g}^i$
$\underline{H}_q^{i+1} = \underline{H}_q^i + \underline{A}^i + \underline{B}^i$
where $\underline{A}^i = -\underline{\sigma}^i(\underline{\sigma}^i)^T/(\underline{\sigma}^i)^T\underline{y}^i$ and
$\underline{B}^i = -\underline{H}_q^i\underline{y}^i(\underline{y}^i)^T\underline{H}_q^i/(\underline{y}^i)^T\underline{H}_q^i\underline{y}^i$
$\sigma = 0$
20
Compute all $\lambda_i(\underline{x})$ and check
$|\lambda_i| \leq \varepsilon_2$ All i in q
Yes
$\lambda_i \geq -\varepsilon_2$
Yes
23
No
24
No
22
21
$\sigma = 0$
Yes
$\sigma = \sigma + 1$
$\underline{x}$ - Correction
20
No
'Constraint violation'
No
6


24
σ > 1
Yes
'Constraint violation'
6
No
Find C_i with most negative λ_i
$||\underline{P}\ \underline{n}_i|| > \varepsilon_3$
Yes
Add C_i to basis $\eta = 0$
22
No
21
23
σ = 0
Yes
KOUNT<LIMIT
Yes
9
No
No
Compute $f(\underline{x}^*), g(\underline{x}^*)$
'Maximum step'
$f(\underline{x}^*) \geq f(\underline{x})$
Yes
STOP
No
25
'f(x) decrease'
Re-Invert

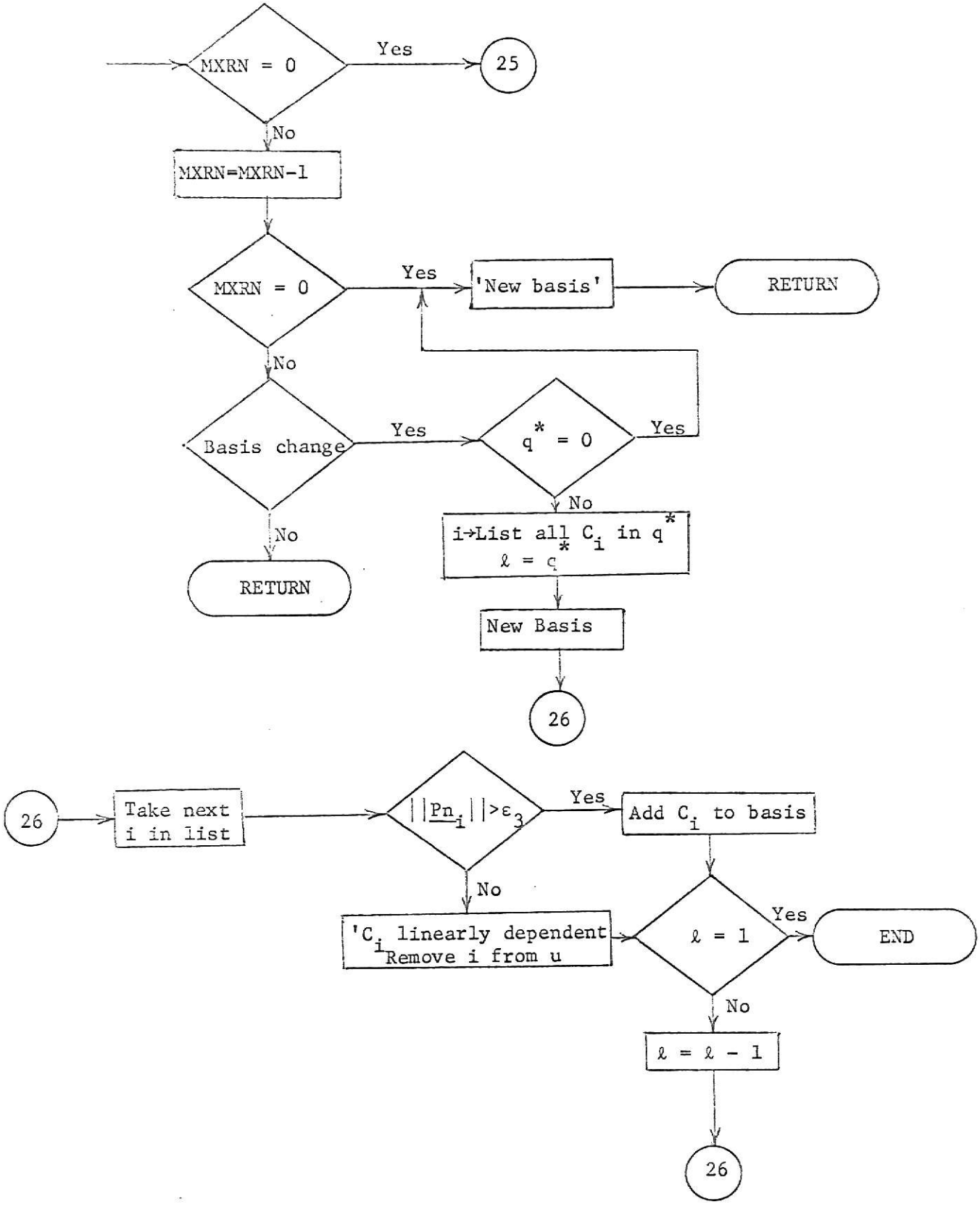


Fig. 10 Flow chart for Subroutine REINV

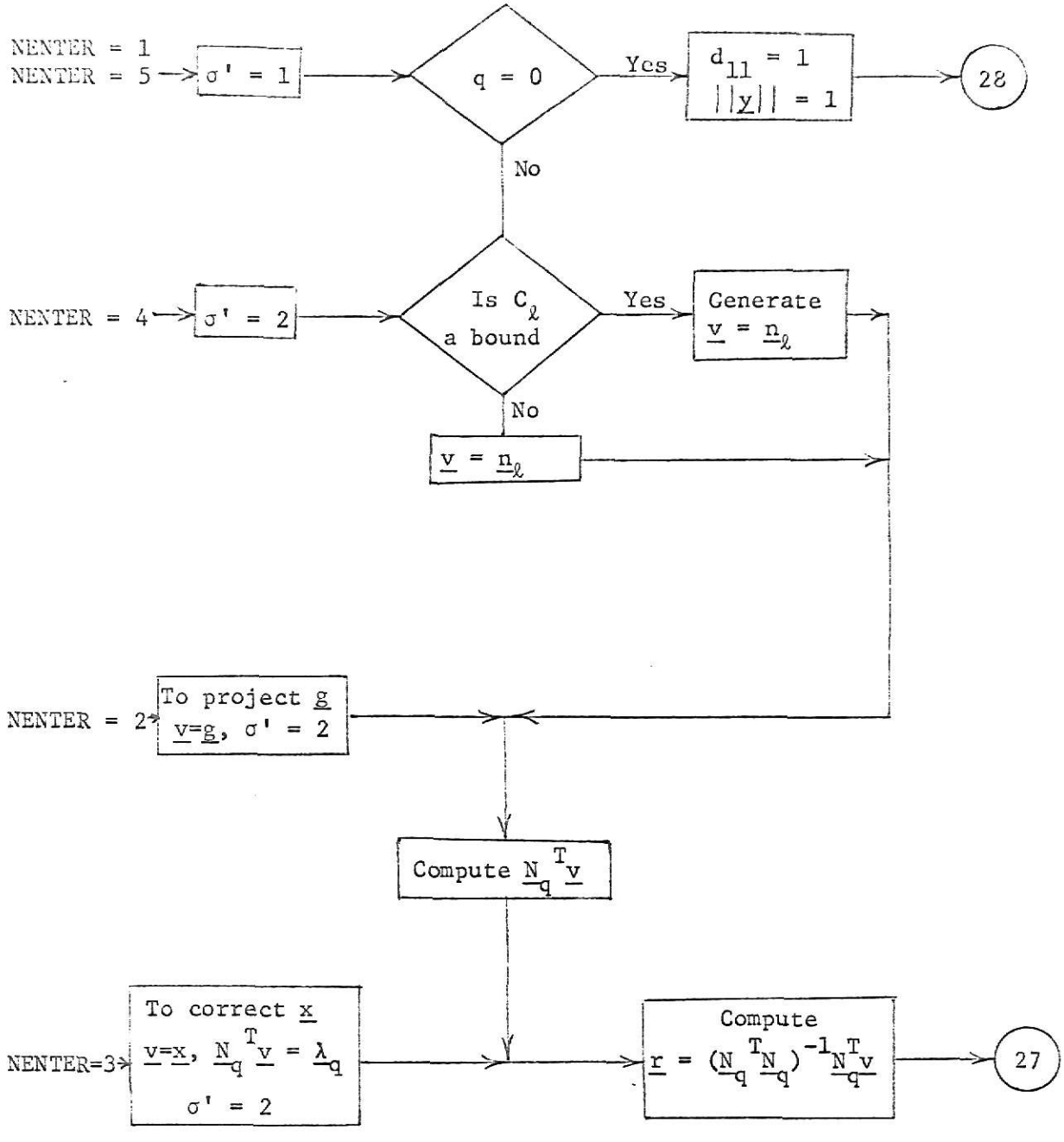
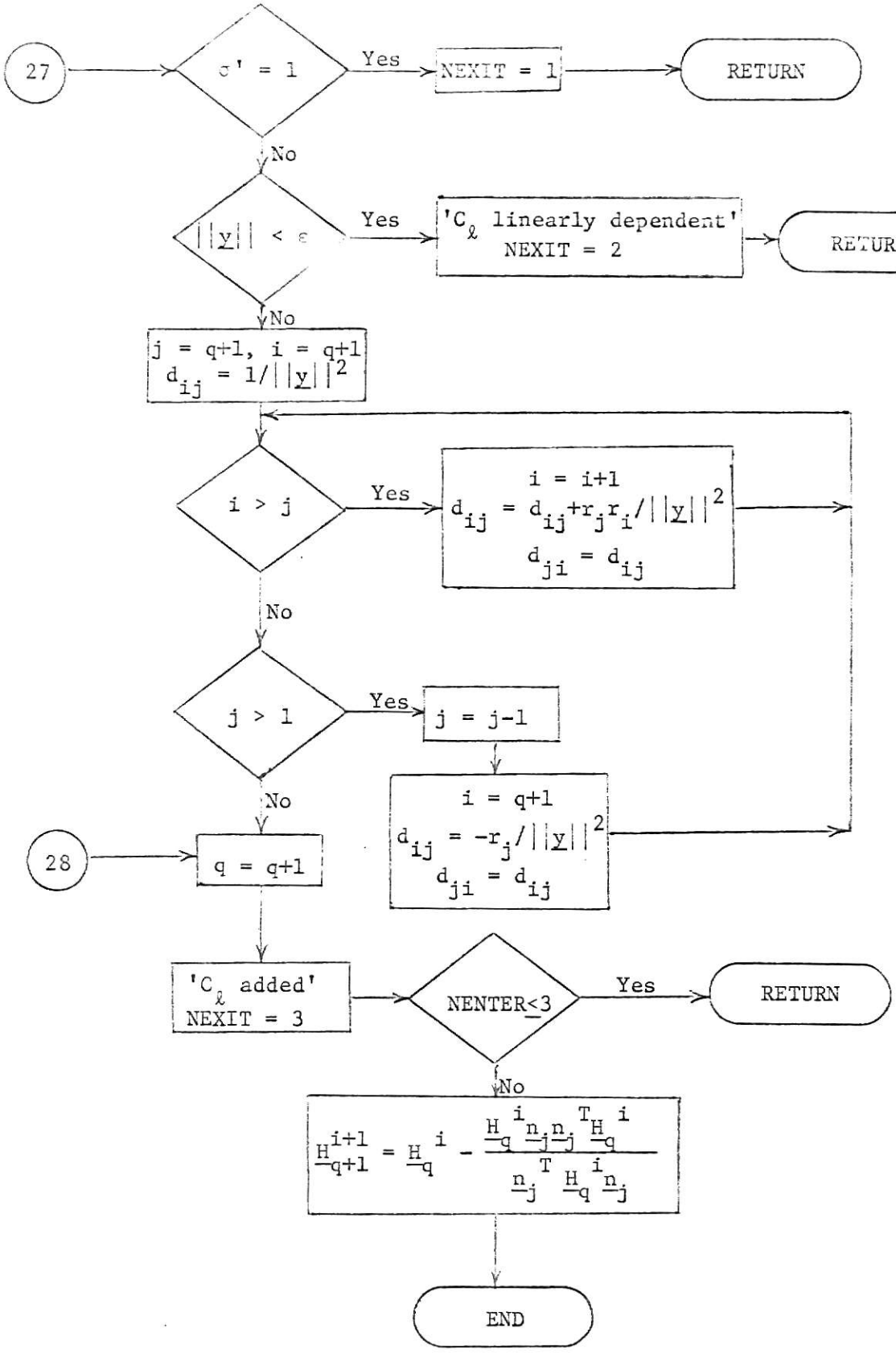


Fig.11 Flow chart for Subroutine COMP1


27
c' = 1
Yes
NEXIT = 1
RETURN
No
||y|| < ε
Yes
'C_ℓ linearly dependent'
NEXIT = 2
RETURN
No
j = q+1, i = q+1
d_ij = 1/||y||^2
i > j
Yes
i = i+1
d_ij = d_ij+r_j r_i/||y||^2
d_ji = d_ij
No
j > 1
Yes
j = j-1
i = q+1
d_ij = -r_j/||y||^2
d_ji = d_ij
No
28
q = q+1
'C_ℓ added'
NEXIT = 3
NENTER≤3
Yes
RETURN
No
H_{q+1}^{i+1} = H_q^i - (H_q^i n_j n_j^T H_q^i) / (n_j^T H_q^i n_j)
END

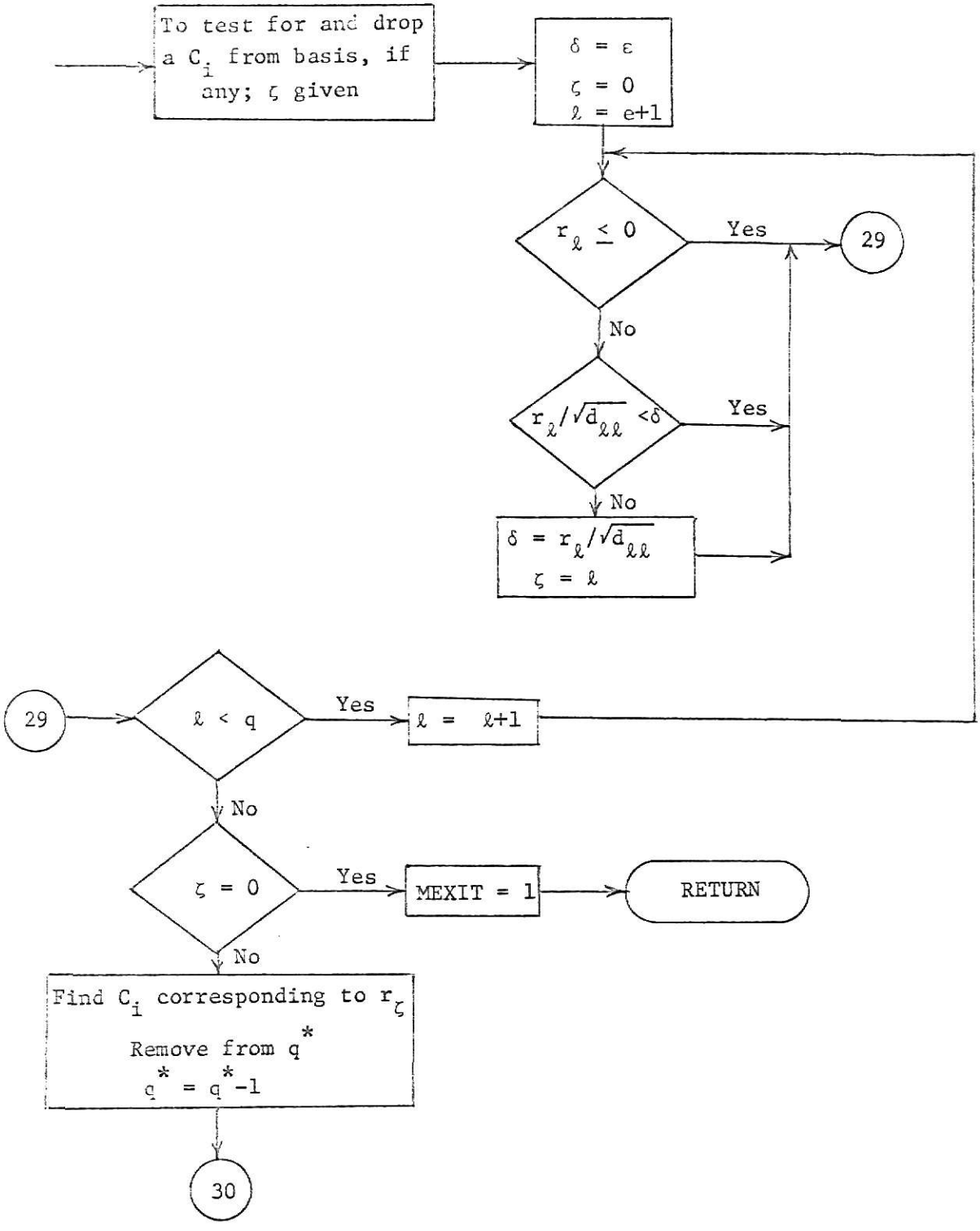
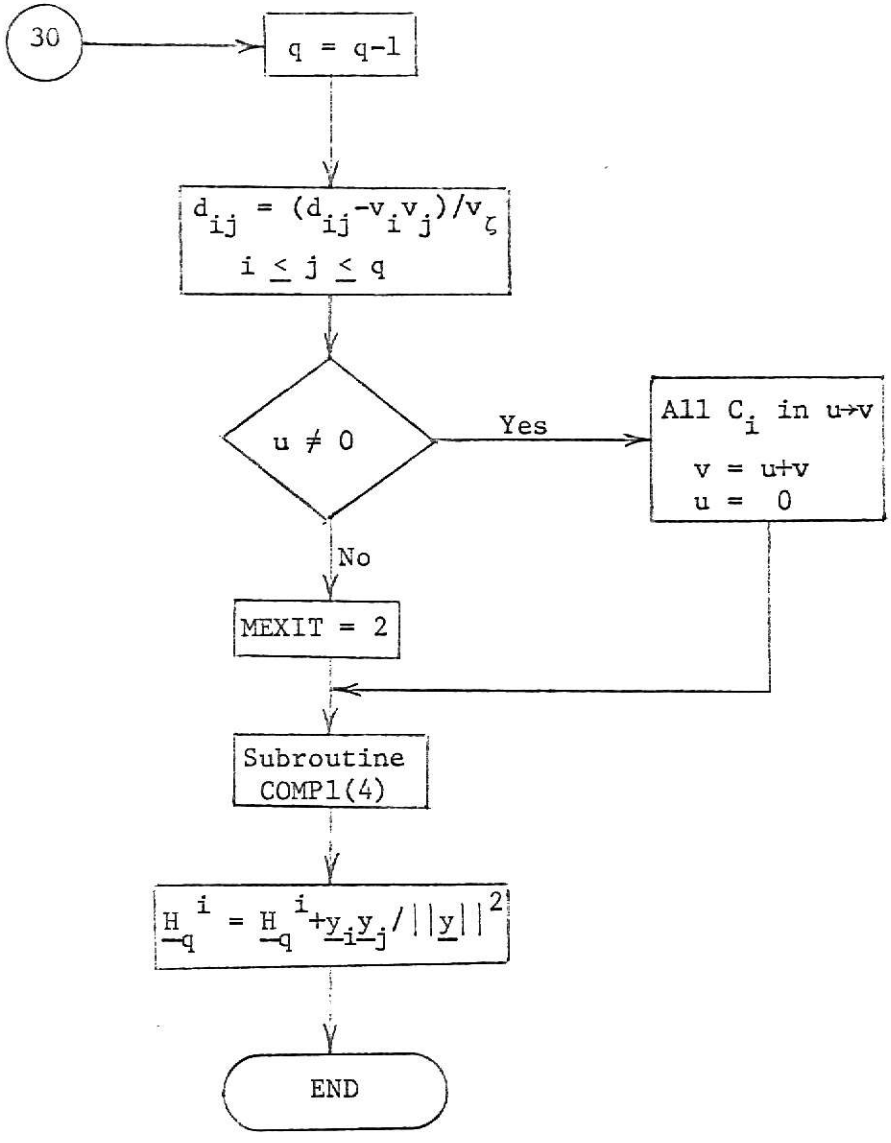


Fig. 12 Flow Chart for Subroutine COMP2


30
q = q-1
$d_{ij} = (d_{ij} - v_i v_j)/v_\zeta$
$i \leq j \leq q$
$u \neq 0$
Yes
All C_i in u→v
v = u+v
u = 0
No
MEXIT = 2
Subroutine
COMP1(4)
$\underline{H}_q^i = \underline{H}_q^i + \underline{y}_i \underline{y}_j / ||\underline{y}||^2$
END

APPENDIX D

```
$JOB            ETIM,RUN=CHECK,TIME=3,PAGES=100,LINES=60
C      COMPUTER PROGRAM FOR THE CONJUGATE GRADIENT METHOD
1      DIMENSION X(10),XO(10),XH(10),X1(10),G(10),GR(10),Y(10),PG(10),
      1JXB(20),IHI(10),IU(10),IV(10),IW(20),B(20),SD(20),P(10),V(10),
      1R(10),RR(10),Z(10),H(10,10),HG(10),AMBDA(20),D(10,10),
      1DN(10,10),A(10,10)
2      COMMON NSIGP,NEXIT,MEXIT,EPSI2,EPSI3,KQ,KEQ,KV,KW,NB,
      1K,KMNB,INV,MXRN,PN,M,IH,LDC,F,X,Y,AMBDA,G,P,
      1D,V,A,JXB,IHI,IU,IV,IW,B,DN,R,H
3      COMMON/INCL/NFUNC
4      COMMON/IND1/KOUNT
5      EQUIVALENCE(Z,HG),(SD,AMBDA),(PN,PGNORM),(Y,PG)
6    1C00 FORMAT(4(I5))
7    1C01 FORMAT('                    STAGE 1  STAGE 2  STAGE 3  STAGE 4
      1TAGE 5')
8    1C20 FORMAT(10I5)
9    1C40 FORMAT(6(F12.6))
10   9991 FORMAT(1H ,'W1 = ',F18.6)
11   9998 FORMAT(1H ,5HERROR)
12     READ 1C00,M,K,NB,NE
13     PRINT 1000,M,K,NB,NE
14     IF(NB .EQ. 0)GO TO 1030
15     READ 1C20,(JXB(I),I=1,NB)
16     PRINT 1020,(JXB(I),I=1,NB)
17   1C30 KMNB=K-NB
18     IF(KMNB .EQ. 0)GO TO 8
19     READ 1C40,((A(I,J),J=1,M),I=1,KMNB)
20     PRINT 1040,((A(I,J),J=1,M),I=1,KMNB)
21     8 CONTINUE
22     READ 1040,(B(I),I=1,K)
23     PRINT 1040,(B(I),I=1,K)
24     READ 1C40,(X(I),I=1,M)
25     PRINT 1040,(X(I),I=1,M)
26     NFUNC=0
27     EPSI1=.001
28     EPSI2=.0001
29     EPSI3=.005
30     LIMIT=50
31     ESTF=500.
32     MXRN=3
33     KQ=0
34     KEQ=0
35     LDC=0
36     INV=1
37     KOUNT=0
38     NBP1=NB+1
39     IF(KMNB .EQ. 0)GO TO 25
40     9 DO 20 I=1,KMNB
41     SD(I)=0.
42     DO 10 J=1,M
43    10 SD(I)=SD(I)+A(I,J)**2
44     SD(I)=SQRT(SD(I))
45     DO 21 J=1,M
46    21 A(I,J)=A(I,J)/SD(I)
47    20 B(I+NB)=B(I+NB)/SD(I)
48     PRINT 1040,((A(I,J),J=1,M),I=1,KMNB)
49     PRINT 1040,(B(I),I=1,K)
50    25 CONTINUE
51     PRINT 1001
52     IF(NE .EQ. 0)GO TO 30
```

```
      IH=NBP1
      DO 610 I=1,NE
  610 IW(I)=0
      GO TO 611
   30 DO 606 I=1,M
      DO 605 J=1,M
  605 H(I,J)=0.
  606 H(I,I)=1.
      GO TO 681
  611 CALL COMP1(1)
      IHI(1)=IH
  615 IF(NE-KQ)675,675,620
  620 N=1
      INDEX=0
      DELTA=EPSI3
  625 N=N+1
      IF(IW(N))630,630,650
  630 DO 635 J=1,M
  635 V(J)=A(N,J)
      CALL COMP1(2)
      PN=0.
      DO 640 J=1,M
  640 PN=PN+Y(J)**2
      PN=SQRT(PN)
      IF(PN-DELTA)650,650,645
  645 DELTA=PN
      INDEX=N
  650 IF(N-NE)625,655,655
  655 IF(INDEX)675,675,660
  660 IH=INDEX+NB
      CALL COMP1(1)
      IW(INDEX)=777
      IHI(KQ)=IH
      GO TO 615
  675 KEQ=KQ
      DO 662 I=1,M
      DO 661 J=1,KEQ
      DN(I,J)=0.
      DO 676 N=1,KEQ
      KK=IHI(N)-NB
  676 DN(I,J)=DN(I,J)+A(KK,I)*D(N,J)
  661 CONTINUE
  662 CONTINUE
      DO 678 I=1,M
      DO 677 J=1,M
      H(I,J)=0.
      DO 677 N=1,KEQ
      KK=IHI(N)-NB
  677 H(I,J)=H(I,J)-A(KK,I)*DN(J,N)
  678 H(I,I)=H(I,I)+1.
      DO 680 I=1,KQ
      DO 680 J=1,KQ
  680 DN(I,J)=D(I,J)
  681 CALL AMDA
      IF(NB .EQ. 0)GO TO 54
      DO 53 I=1,NB
      IF(AMBDA(I)+EPSI2)58,53,53
   53 CONTINUE
   54 IF(NBP1-K)65,65,130
   65 DO 57 J=NBP1,K
```

```
      IF(J-NBP1-NE)55,56,56
   55 IF(ABS(AMBDA(J))-EPSI2)57,57,58
   56 IF(AMBDA(J)+EPSI2)58,57,57
   57 CONTINUE
      GO TO 130
   58 IF(KQ .EQ. 0)GO TO 80
   59 DO 60 I=1,KQ
      J=IHI(I)
      IF(ABS(AMBDA(J))-EPSI2)60,60,70
   60 CONTINUE
      GO TO 80
   70 DO 71 J=1,M
   71 V(J)=X(J)
      CALL COMP1(3)
      DO 72 J=1,M
   72 X(J)=Y(J)
      CALL AMDA
      IF(KQ .EQ. 0)GO TO 80
      DO 73 I=1,KQ
      J=IHI(I)
      IF(ABS(AMBDA(J))-EPSI2)73,73,105
   73 CONTINUE
   80 DO 81 I=1,K
      IF(AMBDA(I)+EPSI2)82,81,81
   81 CONTINUE
      GO TO 130
   82 N=1
      SIGMA=AMBDA(1)
      IF(K-2)86,83,83
   83 DO 85 I=2,K
      IF(AMBDA(I)-SIGMA)84,85,85
   84 N=I
      SIGMA=AMBDA(I)
   85 CONTINUE
   86 SIGMA=0.
      IH=N
   87 CALL COMP1(5)
      GO TO (100,100,90),NEXIT
   90 KQM1=KQ-1
      IF(KQM1)97,97,95
   95 DO 96 I=1,KQM1
   96 AMBDA(I)=0.
   97 INV=1
      GO TO 70
  100 IF(SIGMA)105,101,105
  101 CALL COMP2(EPSI3)
      GO TO (105,103),MEXIT
  103 SIGMA=1.
      INV=1
      IH=N
      GO TO 87
  105 CALL REINV
      IF(NEXIT)59,59,460
  130 CALL FUNCT(X,F,G,KQ)
      IF(KOUNT .GT. 0)GO TO 140
      IF(KQ .LT. KEQ)GO TO 9999
      NETA=0
  140 CALL CLASS
      INV=1
      GNORM=0.
```

```
173          DO 141 J=1,M
174       141 GNORM=GNORM+G(J)**2
175          GNORM=SQRT(GNORM)
176          IF(GNORM-EPSI1)142,142,150
177       142 HGNORM=GNORM
178          KQ=0
179          GO TO 460
180       150 SIGMA=0.
181       160 IF(KQ .GT. 0)GO TO 170
182          HGNORM=0.
183          DO 162 J=1,M
184          HG(J)=0.
185          DO 161 I=1,M
186       161 HG(J)=HG(J)+H(J,I)*G(I)
187       162 HGNORM=HGNORM+HG(J)**2
188          HGNORM=SQRT(HGNORM)
189          GO TO 180
190       170 DO 163 J=1,M
191       163 V(J)=G(J)
192          CALL COMP1(2)
193          HGNORM=0.
194          PGNORM=0.
195          DO 171 J=1,M
196          HG(J)=0.
197          DO 164 I=1,M
198       164 HG(J)=HG(J)+H(J,I)*G(I)
199          HGNORM=HGNORM+HG(J)**2
200       171 PGNORM=PGNORM+PG(J)**2
201          HGNORM=SQRT(HGNORM)
202          PGNORM=SQRT(PGNORM)
203          IF(PGNORM-EPSI1)175,175,172
204       172 IF(M-KQ)9999,173,180
205       173 CALL REINV
206          IF(NEXIT)174,174,460
207       174 CALL CLASS
208          GO TO 150
209       175 IF(SIGMA .NE. 0.)GO TO 173
210          IF(KQ-KEQ)9999,460,178
211       178 CALL COMP2(EPSI1)
212          GO TO (460,179),MEXIT
213       179 INV=1
214          SIGMA=5.
215          GO TO 160
216       180 LDC=0
217          IF(KV)9999,200,190
218       190 KEQP1=KEQ+1
219          IF(KEQP1-KQ)1903,1903,1902
220      1903 CONTINUE
221          DO 1901 J=KEQP1,KQ
222      1901 RR(J)=R(J)
223      1902 L=0
224          INDEX=0
225          DELTA=0.
226       191 L=L+1
227          IF(IV(L)-NB)1912,1912,1915
228      1912 KK=IV(L)
229          J=JXB(KK)
230          IF(J .GT. 0)GO TO 1914
231          ZN=-Z(-J)
232          GO TO 1925
```

```
 1914 ZN=Z(J)
      GO TO 1925
 1915 KK=IV(L)-NB
      ZN=0.
      DO 192 J=1,M
  192 ZN=ZN+Z(J)*A(KK,J)
 1925 IF(ZN-DELTA)193,1935,1935
  193 INDEX=IV(L)
      LL=L
      DELTA=ZN
 1935 IF(KV-L)194,194,191
  194 IF(INDEX)9999,200,195
  195 KV=KV-1
      IF(KV-LL)1995,198,198
  198 DO 199 I=LL,KV
  199 IV(I)=IV(I+1)
 1995 IH=INDEX
      CALL COMP1(5)
      GO TO (197,197,196),NEXIT
  196 INV=1
      GO TO 150
  197 IF(KV)9999,1997,190
 1997 IF(KEQP1-KQ)1998,1998,200
 1998 DO 1999 J=KEQP1,KQ
 1999 R(J)=RR(J)
  200 IF(NETA)201,204,201
  201 IF(KQ-KEQ)202,204,202
  202 CALL COMP2(2.*HGNORM)
      GO TO (204,203),MEXIT
  203 INV=1
      GO TO 160
  204 KQS=KQ-KEQ
      FX=F
      ZGX=0.
      DO 205 I=1,M
      ZGX=ZGX+Z(I)*G(I)
      GR(I)=G(I)
  205 XO(I)=X(I)
      NETA=0
      NETA=NETA+1
      KOUNT=KOUNT+1
      L=0
      TAU=AMIN1(1.,(2.*(ESTF-FX)/ZGX))
      IF(KW)9999,240,231
  231 DO 239 I=1,KW
      J=IW(I)
      IF(J-NB)232,232,235
  232 KK=JXB(J)
      IF(KK)233,9999,234
  233 ZN=-Z(-KK)
      GO TO 2365
  234 ZN=Z(KK)
      GO TO 2365
  235 KK=J-NB
      ZN=0.
      DO 236 N=1,M
  236 ZN=ZN+Z(N)*A(KK,N)
 2365 IF(ZN .GE. 0.)GO TO 239
      PIT=-AMBDA(J)/ZN
      IF(PIT .GE. TAU)GO TO 239
```

```
293          TAU=PIT
294          L=J
295      239 CONTINUE
296      240 DO 241 J=1,M
297          X(J)=XC(J)+TAU*Z(J)
298      241 CONTINUE
299      250 CALL FUNCT(X,F,G,KQ)
300          FXH=F
301          ZG=0.
302          DO 251 J=1,M
303          ZG=ZG+Z(J)*G(J)
304      251 XH(J)=X(J)
305          ZGY=ZG
306          IF(ZGY)260,260,253
307      253 IF(FX-FXH)254,260,260
308      254 IF(TAU-1.)257,255,255
309      255 DO 256 I=1,M
310          DO 256 J=1,M
311      256 H(I,J)=H(I,J)+(Z(I)*Z(J)*TAU)/ZGX
312      257 GO TO 280
313      260 ZZ=(3.*(FX-FXH)/TAU)-ZGX-ZGY
314          W1=ZZ**2-ZGX*ZGY
315          IF(-.0005 .LE. W1 .AND. W1 .LE. 0.)GO TO 9992
316          CONTINUE
317          IF(W1 .LT. 0.)GO TO 9990
318          GO TO 261
319     9990 CONTINUE
320          PRINT 9991,W1
321     9992 W1=0.
322          GO TO 9997
323      261 W=SQRT(W1)
324          D1=(-ZGY+W-ZZ)/(-ZGY+ZGX+2.*W)
325          GAM=TAU*(1.-D1)
326          DO 262 J=1,M
327          R(J)=GAM*Z(J)
328      262 X(J)=XO(J)+R(J)
329          CALL FUNCT(X,F,G,KQ)
330          FX1=F
331          ZGZ=0.
332          DO 263 J=1,M
333          X1(J)=X(J)
334      263 ZGZ=ZGZ+Z(J)*G(J)
335          T1=AMAX1(FX,FXH)
336          IF(T1-FX1)269,269,264
337      264 IF(FX-FXH)267,266,266
338      266 CONTINUE
339          DO 265 J=1,M
340      265 XH(J)=X1(J)
341          TAU=TAU*(1.-D1)
342          ZGY=ZGZ
343          FXH=FX1
344          GO TO 260
345      267 TAU=TAU*D1
346          FX=FX1
347          ZGX=ZGZ
348          DO 268 J=1,M
349      268 XO(J)=X1(J)
350          GO TO 260
351      269 CONTINUE
352      270 DO 271 J=1,M
```

```
353      271 Y(J)=G(J)-GR(J)
354          SIGY=0.
355          YHY=0.
356          DO 273 I=1,M
357          RR(I)=0.
358          DO 272 J=1,M
359      272 RR(I)=RR(I)+Y(J)*H(J,I)
360          SIGY=SIGY+R(I)*Y(I)
361      273 YHY=YHY+RR(I)*Y(I)
362          IF(YHY .EQ. 0.)YHY=.00001
363          CONTINUE
364          IF(SIGY .EQ. 0.)SIGY=.00001
365          CONTINUE
366          DO 275 I=1,M
367          DO 275 J=1,M
368      275 H(I,J)=H(I,J)-(R(I)*R(J))/SIGY-(RR(I)*RR(J))/YHY
369      280 SIGMA=0.
370      290 CALL AMDA
371          IF(KQ .EQ. 0)GO TO 284
372          DO 281 I=1,KQ
373          J=IHI(I)
374          IF(ABS(AMBDA(J)) .GT. EPSI2)GO TO 300
375      281 CONTINUE
376      284 DO 282 I=1,K
377          IF(AMBDA(I)+EPSI2)292,282,282
378      282 CONTINUE
379          GO TO 320
380      292 IF(SIGMA .GT. 1.)GO TO 105
381          L=1
382          RHO=AMBDA(1)
383          IF(K-2)298,299,299
384      299 DO 295 I=2,K
385          IF(AMBDA(I)-RHO)294,295,295
386      294 L=I
387          RHO=AMBDA(I)
388      295 CONTINUE
389      298 IH=L
390          CALL COMP1(5)
391          GO TO (300,300,296),NEXIT
392      296 NETA=0
393          INV=1
394          GO TO 310
395      300 IF(SIGMA .NE. 0.)GO TO 105
396      310 SIGMA=SIGMA+1.
397          DO 311 J=1,M
398      311 V(J)=X(J)
399          CALL COMP1(3)
400          DO 312 J=1,M
401      312 X(J)=Y(J)
402          INV=1
403          GO TO 290
404      320 IF(SIGMA)321,325,321
405      321 CALL FUNCT(X,F,G,KQ)
406          IF(F-FX1)322,325,325
407      322 CONTINUE
408     3333 FORMAT(1H ,'F DECREASE')
409          PRINT 3333
410          CALL REINV
411          IF(NEXIT)325,325,460
412      325 IF((KOUNT-LIMIT) .GE. 0)GO TO 460
```

```
413           GO TO 140
414       460 CALL CLASS
415           GO TO 9997
416      9999 CONTINUE
417           PRINT 9998
418      9997 CONTINUE
419           STOP
420           END
```

```
421           SUBROUTINE COMP1(NENTER)
422           DIMENSION X(10),XO(10),XH(10),X1(10),G(10),GR(10),Y(10),PG(10),
             1JXB(20),IHI(10),IU(10),IV(10),IW(20),B(20),SD(20),P(10),V(10),
             1R(10),RR(10),Z(10),H(10,10),HG(10),AMBDA(20),D(10,10),
             1DN(10,10),A(10,10)
423           DIMENSION TT(10)
424           COMMON NSIGP,NEXIT,MEXIT,EPSI2,EPSI3,KQ,KEQ,KV,KW,NB,
             1K,KMNB,INV,MXRN,PN,M,IH,LDC,F,X,Y,AMBDA,G,P,
             1D,V,A,JXB,IHI,IU,IV,IW,B,DN,R,H
425           GO TO(20,75,100,25,20),NENTER
426          20 NSIGP=1
427           IF(KQ .GT. 0)GO TO 30
428           D(1,1)=1.
429           PN=1.
430           GO TO 260
431          25 NSIGP=2
432          30 IF(IH .LE. NB)GO TO 40
433           I=IH-NB
434           DO 35 J=1,M
435          35 V(J)=A(I,J)
436           GO TO 80
437          40 DO 50 I=1,M
438          50 V(I)=0.
439           JK=JXB(IH)
440           IF(JK .GT. 0)GO TO 70
441           V(-JK)=-1.
442           GO TO 80
443          70 V(JK)=1.
444           GO TO 80
445          75 NSIGP=2
446          80 IF(KQ .GT. 0)GO TO 90
447           DO 85 J=1,M
448          85 Y(J)=V(J)
449           RETURN
450          90 DO 95 I=1,KQ
451           KK=IHI(I)-NB
452           IF(KK)91,91,93
453          91 JBD=IHI(I)
454           JK=JXB(JBD)
455           IF(JK .LT. 0)GO TO 92
456           P(I)=V(JK)
457           GO TO 95
458          92 P(I)=-V(-JK)
459           GO TO 95
460          93 P(I)=0.
461           DO 94 J=1,M
462          94 P(I)=P(I)+A(KK,J)*V(J)
463          95 CONTINUE
464           GO TO 120
465         100 NSIGP=2
466           DO 110 I=1,KQ
467           KK=IHI(I)
468         110 P(I)=AMBDA(KK)
469         120 CONTINUE
470           DO 131 I=1,KQ
471           R(I)=0.
472           DO 130 J=1,KQ
473         130 R(I)=R(I)+D(I,J)*P(J)
474         131 CONTINUE
475           DO 170 J=1,M
```

```
476          Y(J)=0.
477          DO 150 I=1,KQ
478          KK=IHI(I)-NB
479          IF(KK .GT. 0)GO TO 150
480          JBD=IHI(I)
481          JK=JXB(JBD)
482          IF(JK .GT. 0)GO TO 140
483          IF(J+JK)160,135,160
484      135 Y(J)=Y(J)-R(I)
485          GO TO 160
486      140 IF(J-JK)160,145,160
487      145 Y(J)=Y(J)+R(I)
488          GO TO 160
489      150 Y(J)=Y(J)+A(KK,J)*R(I)
490      160 CONTINUE
491      170 Y(J)=V(J)-Y(J)
492          IF(NSIGP .EQ. 1)GO TO 180
493          NEXIT =1
494          RETURN
495      180 PN=0.
496          DO 185 J=1,M
497      185 PN=PN+Y(J)**2
498          PN=SQRT(PN)
499          IF(PN-EPSI3)190,195,195
500      190 LDC=LDC+1
501          IU(LDC)=IH
502          NEXIT=2
503          RETURN
504      195 J=KQ+1
505          I=J
506          D(I,J)=1./(PN**2)
507      200 IF(I-J)240,240,230
508      230 I=I-1
509          D(I,J)=D(I,J)+(R(J)*R(I)/PN**2)
510          D(J,I)=D(I,J)
511          GO TO 200
512      240 IF(J-1)260,260,250
513      250 J=J-1
514          I=KQ+1
515          D(I,J)=-R(J)/PN**2
516          D(J,I)=D(I,J)
517          GO TO 200
518      260 KQ=KQ+1
519          NEXIT=3
520          IHI(KQ)=IH
521          IF(NENTER-3)270,270,290
522      270 RETURN
523      290 IF(IH-NB)300,300,325
524      300 JK=JXB(IH)
525          IF(JK .LT. 0)GO TO 315
526          DO 310 J=1,M
527      310 TT(J)=H(JK,J)
528          H1=H(JK,JK)
529          GO TO 340
530      315 DO 320 J=1,M
531      320 TT(J)=-H(-JK,J)
532          H1=H(-JK,-JK)
533          GO TO 340
534      325 KK=IH-NB
535          H1=0.
```

```
536          DO 335 I=1,M
537          TT(I)=0.
538          DO 330 J=1,M
539      330 TT(I)=TT(I)+A(KK,J)*H(J,I)
540      335 H1=H1+TT(I)+A(KK,I)
541      340 DO 345 I=1,M
542          DO 345 J=1,M
543      345 H(I,J)=H(I,J)-(TT(I)*TT(J)/H1)
544          RETURN
545          END
```

```
      SUBROUTINE COMP2(DEL)
      DIMENSION X(10),XO(10),XH(10),X1(10),G(10),GR(10),Y(10),PG(10),
     1JXB(20),IHI(10),IU(10),IV(10),IW(20),B(20),SD(20),P(10),V(10),
     1R(10),RR(10),Z(10),H(10,10),HG(10),AMBDA(20),D(10,10),
     1DN(10,10),A(10,10)
      COMMON NSIGP,NEXIT,MEXIT,EPSI2,EPSI3,KQ,KEQ,KV,KW,NB,
     1K,KMNB,INV,MXRN,PN,M,IH,LDC,F,X,Y,AMBDA,G,P,
     1D,V,A,JXB,IHI,IU,IV,IW,B,DN,R,H
      DELTA=DEL
      NZ=0
      L=KEQ+1
   10 IF(R(L))13,13,11
   11 RD=R(L)/SQRT(D(L,L))
      IF(RD-DELTA)13,12,12
   12 DELTA=RD
      NZ=L
   13 IF(L-KQ)14,15,15
   14 L=L+1
      GO TO 10
   15 IF(NZ)16,16,17
   16 MEXIT=1
      RETURN
   17 DO 18 I=1,KQ
   18 V(I)=D(I,NZ)
      IH=IHI(NZ)
      KQ=KQ-1
      NZM1=NZ-1
      IF(NZ-KQ)20,20,19
   19 VNZ=V(NZ)
      GO TO 26
   20 DO 24 I=NZ,KQ
      IHI(I)=IHI(I+1)
      IF(NZM1)23,23,21
   21 DO 22 J=1,NZM1
      D(I,J)=D(I+1,J)
   22 D(J,I)=D(I,J)
   23 DO 24 J=NZ,KQ
   24 D(I,J)=D(I+1,J+1)
      VNZ=V(NZ)
      DO 25 I=NZ,KQ
   25 V(I)=V(I+1)
   26 DO 29 I=1,KQ
      DO 29 J=1,KQ
      IF(I-J)27,27,28
   27 D(I,J)=D(I,J)-V(I)*V(J)/VNZ
      GO TO 29
   28 D(I,J)=D(J,I)
   29 CONTINUE
      IF(LDC)32,32,30
   30 DO 31 I=1,LDC
      J=KV+I
   31 IV(J)=IU(I)
      KV=KV+LDC
      LDC=0
   32 MEXIT=2
      CALL COMP1(4)
      PN=0.
      DO 33 J=1,M
   33 PN=PN+Y(J)**2
      DO 34 I=1,M
```

```
601         DO 34 J=1,M
602      34 H(I,J)=H(I,J)+(Y(I)*Y(J)/PN)
603         RETURN
604         END
```

```
605         SUBROUTINE REINV
606         DIMENSION X(10),XO(10),XH(10),X1(10),G(10),GR(10),Y(10),PG(10),
           1JXB(20),IHI(10),IU(10),IV(10),IW(20),B(20),SD(20),P(10),V(10),
           1R(10),RR(10),Z(10),H(10,10),HG(10),AMBDA(20),D(10,10),
           1DN(10,10),A(10,10)
607         COMMON NSIGP,NEXIT,MEXIT,EPSI2,EPSI3,KQ,KEQ,KV,KW,NB,
           1K,KMNB,INV,MXRN,PN,M,IH,LDC,F,X,Y,AMBDA,G,P,
           1D,V,A,JXB,IHI,IU,IV,IW,B,DN,R,H
608         IF(MXRN .GT. 0)GO TO 10
609       9 NEXIT=1
610         RETURN
611      10 NEXIT=0
612         MXRN=MXRN-1
613         IF(MXRN .LE. 0)GO TO 20
614         GO TO 40
615      20 IF(KEQ .EQ. 0)GO TO 35
616      15 DO 30 I=1,KEQ
617         DO 30 J=1,KEQ
618      30 D(I,J)=DN(I,J)
619      35 LDC=0
620         KQ=KEQ
621         INV=0
622         RETURN
623      40 IF(INV)9,9,50
624      50 IF(KQ-KEQ)15,15,51
625      51 L=KQ-KEQ
626         IF(KEQ .GT. 0)GO TO 54
627         DO 53 I=1,M
628         DO 52 J=1,M
629      52 H(I,J)=0.
630      53 H(I,I)=1.
631         GO TO 59
632      54 DO 46 I=1,M
633         DO 45 J=1,KEQ
634         D(I,J)=0.
635         DO 55 N=1,KEQ
636         KK=IHI(N)-NB
637      55 D(I,J)=D(I,J)+A(KK,I)+DN(N,J)
638      45 CONTINUE
639      46 CONTINUE
640         DO 57 I=1,M
641         DO 56 J=1,M
642         H(I,J)=0.
643         DO 56 N=1,KEQ
644         KK=IHI(N)-NB
645      56 H(I,J)=H(I,J)-A(KK,I)*D(J,N)
646      57 H(I,I)=H(I,I)+1.
647         DO 58 I=1,KEQ
648         DO 58 J=1,KEQ
649      58 D(I,J)=DN(I,J)
650      59 LDC=0
651         KQ=KEQ
652         INV=0
653      60 IH=IHI(KQ+1)
654         CALL COMP1(5)
655      65 IF(L .LE. 1)GO TO 70
656         L=L-1
657         GO TO 60
658      70 NEXIT=0
659         RETURN
```

```
660         END
```

```
661          SUBROUTINE AMDA
662          DIMENSION X(10),XO(10),XH(10),XL(10),G(10),GR(10),Y(10),PG(10),
            1JXB(20),IHI(10),IU(10),IV(10),IW(20),B(20),SD(20),P(10),V(10),
            1R(10),RR(10),Z(10),H(10,10),HG(10),AMBDA(20),D(10,10),
            1DN(10,10),A(10,10)
663          COMMON NSIGP,NEXIT,MEXIT,EPSI2,EPSI3,KQ,KEQ,KV,KW,NB,
            1K,KMNB,INV,MXRN,PN,M,IH,LDC,F,X,Y,AMBDA,G,P,
            1D,V,A,JXB,IHI,IU,IV,IW,B,DN,R,H
664          IF(NB .EQ. 0)GO TO 40
665       10 DO 30 I=1,NB
666          J=JXB(I)
667          IF(J .LT. 0)GO TO 20
668          AMBDA(I)=X(J)-B(I)
669          GO TO 30
670       20 AMBDA(I)=-X(-J)-B(I)
671       30 CONTINUE
672       40 IF(KMNB)80,80,50
673       50 DO 70 I=1,KMNP
674          TOTA=0.
675          KK=NB+I
676          DO 60 J=1,M
677       60 TOTA=TOTA+A(I,J)*X(J)
678       70 AMBDA(KK)=TOTA-B(KK)
679       80 RETURN
680          END
```

```
      SUBROUTINE CLASS
      DIMENSION X(10),XO(10),XH(10),X1(10),G(10),GR(10),Y(10),PG(10),
     1JXB(20),IHI(10),IU(10),IV(10),IW(20),B(20),SD(20),P(10),V(10),
     1R(10),RR(10),Z(10),H(10,10),HG(10),AMBDA(20),D(10,10),
     1DN(10,10),A(10,10)
      COMMON NSIGP,NEXIT,MEXIT,EPSI2,EPSI3,KQ,KEQ,KV,KW,NB,
     1K,KMNB,INV,MXRN,PN,M,IH,LDC,F,X,Y,AMBDA,G,P,
     1D,V,A,JXB,IHI,IU,IV,IW,B,DN,R,H
      KW=0
      KV=0
      DO 60 I=1,K
      IF(KQ .EQ. 0)GO TO 30
   10 DO 20 J=1,KQ
      IF(I-IHI(J))20,60,20
   20 CONTINUE
   30 IF(ABS(AMBDA(I))-EPSI2)40,40,50
   40 KV=KV+1
      IV(KV)=I
      GO TO 60
   50 KW=KW+1
      IW(KW)=I
   60 CONTINUE
      RETURN
      END
```

```
700         SUBROUTINE FUNCT(X,F,G,KQ)
      C     SUBROUTINE FUNCT FOR A 5 STAGE INVENTORY CONTROL MODEL WITH ONE
      C     STATE VARIABLE
701         DIMENSION X(10),G(10),VI(20),Q(20),P(20),S(20),U(20),W(20),CT(20)
702         COMMON/IND1/KOUNT
703         COMMON/INCL/NFUNC
704       100 FORMAT('      ITERATION # = ',I3,'      # OF FUNCTIONAL EVALUATIONS
          1= ',I3,'      # OF CONSTRAINTS IN BASIS = ',I3,/)
705       101 FORMAT('      PRODUCTION =  ',5F9.3,/'      INVENTORY  =  ',5F9.3,/'
          1     SALES      =  ',5F9.3,/'      COST       =  ',5F9.3,//)
706         NFUNC=NFUNC+1
707         M=5
708         MP1=M+1
709         A=2.
710         B=1.
711         C=5.
712         CVI=.1
713         CX=.001
714         DT=.2
715         DS=-CVI*(DT**2)
716         VIM=10.
717         XM=5.
718         VI(1)=C
719         Q(1)=A
720         DO 10 I=2,MP1
721         Q(I)=A+B*(I-1)*DT
722         VI(I)=VI(I-1)+(X(I-1)-Q(I))*DT
723         P(I)=2.*VIM-3.*VI(I)/2.
724        10 S(I)=2.*VIM-2.*VI(I)
725         DO 20 I=1,M
726         U(I)=-2.*CX*(XM-X(I))*EXP((XM-X(I))**2)*DT
727        20 W(I)=VIM-VI(I)/2.
728         RP=2.*VIM-3.*VI(M)/2.-VI(MP1)
729         RS=2.*VIM-2.*VI(M)-VI(MP1)
730         RU=VIM-VI(M)/2.-VI(MP1)/2.
731         CT(1)=(CVI*(VIM-VI(1)/2.-VI(2)/2.)**2+CX*EXP((XM-X(1))**2))*DT
732         DO 30 I=2,M
733        30 CT(I)=CT(I-1)+(CVI*(VIM-VI(I)/2.-VI(I+1)/2.)**2+CX*EXP((XM-X(I))
          1**2))*DT
734         PRINT 100,KOUNT,NFUNC,KQ
735         PRINT 101,(X(I),I=1,M),(VI(I),I=2,MP1),(Q(I),I=2,MP1),(CT(I),I=1,M
          1)
736        40 F=-CT(M)
737         G(1)=-(U(1)+DS*(W(1)+P(2)+S(3)+S(4)+RS))
738         G(2)=-(U(2)+DS*(W(2)+P(3)+S(4)+RS))
739         G(3)=-(U(3)+DS*(W(3)+P(4)+RS))
740         G(4)=-(U(4)+DS*(W(4)+RP))
741         G(5)=-(U(5)+DS*RU)
742         RETURN
743         END
```

```
421         SUBROUTINE FUNCT(X,F,G,KQ)
      C     SUBROUTINE FUNCT FOR A 10 STAGE INVENTORY CONTROL MODEL WITH ONE
      C     STATE VARIABLE
422         DIMENSION X(10),G(10),VI(20),Q(20),P(20),S(20),U(20),W(20),CT(20)
423         COMMON/IND1/KOUNT
424         COMMON/INCL/NFUNC
425       100 FORMAT('      ITERATION # = ',I3,'      # OF FUNCTIONAL EVALUATIONS
          1= ',I3,'      # OF CONSTRAINTS IN BASIS = ',I3,/)
426       101 FORMAT('      PRODUCTION = ',10F9.3,/'      INVENTORY  = ',10F9.3,/'
          1     SALES       = ',10F9.3,/'      COST        = ',10F9.3,//)
427         NFUNC=NFUNC+1
428         M=10
429         MP1=M+1
430         A=2.
431         B=1.
432         C=5.
433         CVI=.1
434         CX=.001
435         DT=.1
436         DS=-CVI*(DT**2)
437         VIM=10.
438         XM=5.
439         VI(1)=C
440         Q(1)=A
441         DO 10 I=2,MP1
442         Q(I)=A+B*(I-1)*DT
443         VI(I)=VI(I-1)+(X(I-1)-Q(I))*DT
444         P(I)=2.*VIM-3.*VI(I)/2.
445        10 S(I)=2.*VIM-2.*VI(I)
446         DO 20 I=1,M
447         U(I)=-2.*CX*(XM-X(I))*EXP((XM-X(I))**2)*DT
448        20 W(I)=VIM-VI(I)/2.
449         RP=2.*VIM-3.*VI(M)/2.-VI(MP1)
450         RS=2.*VIM-2.*VI(M)-VI(MP1)
451         RU=VIM-VI(M)/2.-VI(MP1)/2.
452         CT(1)=(CVI*(VIM-VI(1)/2.-VI(2)/2.)**2+CX*EXP((XM-X(1))**2))*DT
453         DO 30 I=2,M
454        30 CT(I)=CT(I-1)+(CVI*(VIM-VI(I)/2.-VI(I+1)/2.)**2+CX*EXP((XM-X(I))
          1**2))*DT
455         CONTINUE
456         PRINT 100,KOUNT,NFUNC,KQ
457         PRINT 101,(X(I),I=1,M),(VI(I),I=2,MP1),(Q(I),I=2,MP1),(CT(I),I=1,M
          1)
458        40 F=-CT(M)
459         G(1)=-(U(1)+DS*(W(1)+P(2)+S(3)+S(4)+S(5)+S(6)+S(7)+S(8)+S(9)+RS))
460         G(2)=-(U(2)+DS*(W(2)+P(3)+S(4)+S(5)+S(6)+S(7)+S(8)+S(9)+RS))
461         G(3)=-(U(3)+DS*(W(3)+P(4)+S(5)+S(6)+S(7)+S(8)+S(9)+RS))
462         G(4)=-(U(4)+DS*(W(4)+P(5)+S(6)+S(7)+S(8)+S(9)+RS))
463         G(5)=-(U(5)+DS*(W(5)+P(6)+S(7)+S(8)+S(9)+RS))
464         G(6)=-(U(6)+DS*(W(6)+P(7)+S(8)+S(9)+RS))
465         G(7)=-(U(7)+DS*(W(7)+P(8)+S(9)+RS))
466         G(8)=-(U(8)+DS*(W(8)+P(9)+RS))
467         G(9)=-(U(9)+DS*(W(9)+RP))
468         G(10)=-(U(10)+DS*RU)
469         RETURN
470         END
```

```
700         SUBROUTINE FUNCT(X,F,G,KQ)
      C     SUBROUTINE FUNCT FOR A 5 STAGE INVENTORY CONTROL WITH ADVERTISING
      C     MODEL WITH TWO STATE VARIABLES
701         DIMENSION X(10),G(10),P(20),Q(20),VI(20),S(20),U(20),PR(20),V(20),
           1E(20),W(20)
702         COMMON/IND1/KOUNT
703         COMMON/INCL/NFUNC
704       100 FORMAT('      ITERATION # = ',I3,'      # OF FUNCTIONAL EVALUATIONS
           1= ',I3,'      # OF CONSTRAINTS IN BASIS = ',I3,/)
705       101 FORMAT('      CON. INC.  =  ',5F9.3,/'      INVENTORY  =  ',5F9.3,/'
           1     PRODUCTION =  ',5F9.3,/'      SALES      =  ',5F9.3,/'      PRO
           1FIT     =  ',5F9.3,//)
706         NFUNC=NFUNC+1
707         M=5
708         MP1=M+1
709         A=70.
710         B=100.
711         C=2.
712         QN=150.
713         F1=10.
714         PI=50.
715         CVI=.15
716         CA=1.5
717         DT=.2
718         DS=-2.*CVI*DT
719         VI(1)=20.
720         Q(1)=20.
721         P(1)=A
722         DO 10 I=2,MP1
723         P(I)=A+B*(I-1)*DT
724         Q(I)=(Q(I-1)*(1.+(C+X(I-1))*DT))/(1.+(C+X(I-1))*Q(I-1)*DT/QN)
725         VI(I)=VI(I-1)+(P(I-1)-Q(I-1))*DT
726         S(I)=-CA*Q(I)*DT
727        10 U(I)=PI-VI(I)
728         PR(1)=(F1*Q(2)-CVI*(PI-VI(2))**2-CA*X(1)*Q(2))*DT
729         DO 20 I=2,M
730        20 PR(I)=PR(I-1)+(F1*Q(I+1)-CVI*(PI-VI(I+1))**2-CA*X(I)*Q(I+1))*DT
731         F=PR(M)
732         PRINT 100,KOUNT,NFUNC,KQ
733         PRINT 101,(X(I),I=1,M),(VI(I),I=2,MP1),(P(I),I=2,MP1),(Q(I),I=2,MP
           11),(PR(I),I=1,M)
734         DO 30 I=1,M
735         V(I)=(Q(I)*(DT**2)*(1.-Q(I)/QN))/((1.+(C+X(I))*Q(I)*DT/QN)**2)
736         E(I)=F1-CA*X(I)
737        30 W(I)=(1+(C+X(I))*DT)/((1+(C+X(I))*Q(I)*DT/QN)**2)
738         DO 35 I=MP1,20
739        35 W(I)=0.
740         L=1
741        40 CONTINUE
742         W1=W(L+1)*W(L+2)
743         W2=W1*W(L+3)
744         W3=W2*W(L+4)
745         W10=1+W(L+1)
746         W11=W10+W1
747         W12=W11+W2
748         GO TO(41,42,43,44,45),L
749        41 G(1)=S(2)+V(1)*(E(1)+E(2)*W(2)+E(3)*W1+E(4)*W2+E(5)*W3+DS*(U(3)+
           1U(4)*W10+U(5)*W11+U(6)*W12))
750         L=2
751         GO TO 40
```

```
752       42 G(2)=S(3)+V(2)*(E(2)+E(3)*W(3)+E(4)*W1+E(5)*W2+DS*(U(4)+U(5)*W10+
             1U(6)*W11))
753          L=3
754          GO TO 40
755       43 G(3)=S(4)+V(3)*(E(3)+E(4)*W(4)+E(5)*W1+DS*(U(5)+U(6)*W10))
756          L=4
757          GO TO 40
758       44 G(4)=S(5)+V(4)*(E(4)+E(5)*W(5)+DS*U(6))
759          L=5
760          GO TO 40
761       45 G(5)=S(6)+V(5)*E(5)
762          RETURN
763          END
```

```
706         SUBROUTINE FUNCT(X,F,G,KQ)
      C     SUBROUTINE FUNCT FOR A 10 STAGE INVENTORY CONTROL WITH ADVERTISING
      C     MODEL WITH TWO STATE VARIABLES
707         DIMENSION X(10),G(10),P(20),Q(20),VI(20),S(20),U(20),PR(20),V(20),
           1E(20),W(20)
708         COMMON/IND1/KOUNT
709         COMMON/INCL/NFUNC
710       100 FORMAT('      ITERATION # = ',I3,'      # OF FUNCTIONAL EVALUATIONS
           1= ',I3,'      # OF CONSTRAINTS IN BASIS = ',I3,/)
711       101 FORMAT('      CON. INC.  = ',10F9.3,/'      INVENTORY  = ',10F9.3,/'
           1     PRODUCTION = ',10F9.3,/'      SALES      = ',10F9.3,/'     PRO
           1FIT     = ',10F9.3,//)
712         NFUNC=NFUNC+1
713         M=10
714         MP1=M+1
715         A=70.
716         B=100.
717         C=2.
718         QN=150.
719         F1=10.
720         PI=50.
721         CVI=.15
722         CA=1.5
723         DT=.1
724         DS=-2.*CVI*DT
725         VI(1)=20.
726         Q(1)=20.
727         P(1)=A
728         DO 10 I=2,MP1
729         P(I)=A+B*(I-1)*DT
730         Q(I)=(Q(I-1)*(1.+(C+X(I-1))*DT))/(1.+(C+X(I-1))*Q(I-1)*DT/QN)
731         VI(I)=VI(I-1)+(P(I-1)-Q(I-1))*DT
732         S(I)=-CA*Q(I)*DT
733        10 U(I)=PI-VI(I)
734         PR(1)=(F1*Q(2)-CVI*(PI-VI(2))**2-CA*X(1)*Q(2))*DT
735         DO 20 I=2,M
736        20 PR(I)=PR(I-1)+(F1*Q(I+1)-CVI*(PI-VI(I+1))**2-CA*X(I)*Q(I+1))*DT
737         F=PR(M)
738         PRINT 100,KOUNT,NFUNC,KQ
739         PRINT 101,(X(I),I=1,M),(VI(I),I=2,MP1),(P(I),I=2,MP1),(Q(I),I=2,MP
           11),(PR(I),I=1,M)
740         DO 30 I=1,M
741         V(I)=(Q(I)*(DT**2)*(1.-Q(I)/QN))/((1.+(C+X(I))*Q(I)*DT/QN)**2)
742         E(I)=F1-CA*X(I)
743        30 W(I)=(1+(C+X(I))*DT)/((1+(C+X(I))*Q(I)*DT/QN)**2)
744         DO 35 I=MP1,20
745        35 W(I)=0.
746         L=1
747        40 CONTINUE
748         W1=W(L+1)*W(L+2)
749         W2=W1*W(L+3)
750         W3=W2*W(L+4)
751         W4=W3*W(L+5)
752         W5=W4*W(L+6)
753         W6=W5*W(L+7)
754         W7=W6*W(L+8)
755         W8=W7*W(L+9)
756         W10=1+W(L+1)
757         W11=W10+W1
758         W12=W11+W2
```

```
759          W13=W12+W3
760          W14=W13+W4
761          W15=W14+W5
762          W16=W15+W6
763          W17=W16+W7
764          GO TO(41,42,43,44,45,46,47,48,49,50),L
765       41 G(1)=S(2)+V(1)*(E(1)+E(2)*W(2)+E(3)*W1+E(4)*W2+E(5)*W3+E(6)*W4+E(7
             1)*W5+E(8)*W6+E(9)*W7+E(10)*W8+DS*(U(3)+U(4)*W10+U(5)*W11+U(6)*W12+
             1U(7)*W13+U(8)*W14+U(9)*W15+U(10)*W16+U(11)*W17))
766          L=2
767          GO TO 40
768       42 G(2)=S(3)+V(2)*(E(2)+E(3)*W(3)+E(4)*W1+E(5)*W2+E(6)*W3+E(7)*W4+E(8
             1)*W5+E(9)*W6+E(10)*W7+DS*(U(4)+U(5)*W10+U(6)*W11+U(7)*W12+U(8)*W13
             1+U(9)*W14+U(10)*W15+U(11)*W16))
769          L=3
770          GO TO 40
771       43 G(3)=S(4)+V(3)*(E(3)+E(4)*W(4)+E(5)*W1+E(6)*W2+E(7)*W3+E(8)*W4+E(9
             1)*W5+E(10)*W6+DS*(U(5)+U(6)*W10+U(7)*W11+U(8)*W12+U(9)*W13+U(10)*
             1W14+U(11)*W15))
772          L=4
773          GO TO 40
774       44 G(4)=S(5)+V(4)*(E(4)+E(5)*W(5)+E(6)*W1+E(7)*W2+E(8)*W3+E(9)*W4+E(1
             10)*W5+DS*(U(6)+U(7)*W10+U(8)*W11+U(9)*W12+U(10)*W13+U(11)*W14))
775          L=5
776          GO TO 40
777       45 G(5)=S(6)+V(5)*(E(5)+E(6)*W(6)+E(7)*W1+E(8)*W2+E(9)*W3+E(10)*W4+DS
             1*(U(7)+U(8)*W10+U(9)*W11+U(10)*W12+U(11)*W13))
778          L=6
779          GO TO 40
780       46  G(6)=S(7)+V(6)*(E(6)+E(7)*W(7)+E(8)*W1+E(9)*W2+E(10)*W3+DS*(U(8)
             1+U(9)*W10+U(10)*W11+U(11)*W12))
781          L=7
782          GO TO 40
783       47 G(7)=S(8)+V(7)*(E(7)+E(8)*W(8)+E(9)*W1+E(10)*W2+DS*(U(9)+U(10)*W10
             1+U(11)*W11))
784          L=8
785          GO TO 40
786       48 G(8)=S(9)+V(8)*(E(8)+E(9)*W(9)+E(10)*W1+DS*(U(10)+U(11)*W10))
787          L=9
788          GO TO 40
789       49 G(9)=S(10)+V(9)*(E(9)+E(10)*W(10)+DS*U(11))
790          L=10
791          GO TO 40
792       50 G(10)=S(11)+V(10)*E(10)
793          RETURN
794          END
```

THE CONJUGATE GRADIENT METHOD AND MANAGEMENT SYSTEMS

by

CHRISTOPHER REBELO NUNES

B.E.(Elec.)(Hons.), B.E.(Mech.)(Hons.), University of Bombay
Bombay, India, 1966, 1967

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1969

In 1966, Goldfarb developed a new method by the combination of the gradient projection method and the conjugate gradient method. This method appears to be a powerful tool for solving a subclass of nonlinear programming problems characterized by linear constraints. The advantage of this technique is its ability to use the local quadratic properties of the general nonlinear objective function effectively and continually without the need of computing the second partial derivatives. This method is approximately quadratically convergent.

The purpose of this report is to apply this technique to industrial management systems and critically analyze the results obtained and the effectiveness of the computational procedure used.

The method is first described in some detail. The versatility of this method is illustrated by considering two production and inventory control models. In the first model, the costs of production and inventory are minimized. In the second model a process involving production and inventory control with advertising is considered. The merits and demerits of the method are analyzed from a nonlinear programming problem solution point of view.