A non-stationary two-dimensional acceleration for the one-dimensional projection method

Howard Dean Pyron
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd
Part of the Computer Sciences Commons

Recommended Citation

This Dissertation is brought to you for free and open access by Digital Repository @ Iowa State University. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Digital Repository @ Iowa State University. For more information, please contact digirep@iastate.edu.
PYRON, Howard Dean, 1933-
A NON-STATIONARY TWO-DIMENSIONAL ACCELERATION FOR THE ONE-DIMENSIONAL PROJECTION METHOD.

Iowa State University, Ph.D., 1971 Computer Science

University Microfilms, A XEROX Company, Ann Arbor, Michigan

THIS DISSERTATION HAS BEEN MICROFLIMED EXACTLY AS RECEIVED
A non-stationary two-dimensional acceleration for the one-dimensional Projection method

by

Howard D. Pyron

A Dissertation Submitted to the Graduate Faculty in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1971
PLEASE NOTE:

Some Pages have indistinct print. Filmed as received.

UNIVERSITY MICROFILMS
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. SYMBOLS AND NOTATIONAL DEFINITIONS</td>
<td>1</td>
</tr>
<tr>
<td>II. INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>III. REVIEW OF THE PROJECTION METHOD</td>
<td>8</td>
</tr>
<tr>
<td>IV. A TWO-DIMENSIONAL PROJECTION ALGORITHM</td>
<td>11</td>
</tr>
<tr>
<td>V. AN OPTIMAL ORDERING ALGORITHM FOR SOLVING $Ax = b$</td>
<td>24</td>
</tr>
<tr>
<td>VI. A NON-STATIONARY ITERATIVE ALGORITHM</td>
<td>30</td>
</tr>
<tr>
<td>VII. CONVERGENCE</td>
<td>35</td>
</tr>
<tr>
<td>VIII. COMPARISONS</td>
<td>44</td>
</tr>
<tr>
<td>IX. CONCLUSIONS AND FUTURE RESEARCH</td>
<td>61</td>
</tr>
<tr>
<td>X. BIBLIOGRAPHY</td>
<td>63</td>
</tr>
<tr>
<td>XI. ACKNOWLEDGEMENTS</td>
<td>65</td>
</tr>
<tr>
<td>XII. PROGRAM IMPLEMENTATION</td>
<td>66</td>
</tr>
<tr>
<td>XIII. APPENDIX</td>
<td>70</td>
</tr>
</tbody>
</table>
I. SYMBOLS AND ROTATIONAL DEFINITIONS

1. A represents the matrix of coefficients
2. b represents the constant vector of the linear system
3. \( a_i \) is the i-th column vector of A
4. \((x,y)\) is the inner product of the vectors x and y
5. \( A_i = (a_i, a_i) \)
6. \( a_i = (r^k, a_i) / A_i \)
7. \( A^i_j = (a_i, a_j) \)
8. \( r^k = (r_k, r_k) \)
9. \( \theta_{ij} \) represents the angle between the vectors \( a_i \) and \( a_j \)
10. \( C^i_j = 1 / (1 - \cos^2 \theta_{ij}), i \neq j \)
11. \( \cos^2 \theta_{ij} = (A^i_j)^2 / (A_i A_j) \)
12. \( n \mod k = k \mod (n) \)
13. \( e_i = (0,0,0, \ldots,1,0,0, \ldots) \); a one in the i-th position
14. \( c_i \) is the i-th row of the matrix C
15. \( x^k \) refers to the k-th approximation to the solution vector x and not to the k-th power of the solution vector
16. \( v = a_i a_i - a_j a_j \)
II. INTRODUCTION

There are many methods for solving a linear algebraic system of equations denoted by $Ax = b$, where $A$ is an $n$ by $n$ matrix, and the column vectors $x$ and $b$ are $n$-tuples. Most of these methods can be included in one of three groups. Group one is comprised of direct methods in which the solution is obtained in a finite, usually fixed, number of steps. One of these direct methods is to find the inverse of $A$ by one of the well-known algorithms, then to find the solution using $A^{-1}b$. Another direct method is elimination or triangularization such as Gauss or Gauss-Jordan (7, p. 60), in which the coefficient matrix is augmented with the column vector $b$. Row operations are then performed on this array until the part corresponding to $A$ is reduced to triangular or diagonal form, from which the solution is readily obtained. This method is more useful if the right-hand side is an $n$ by $m$ matrix $B$ (11, p. 91), rather than a vector. The row eliminations on the augmented system leave the solution vectors, $x_1$, in the column vectors $b_1$. Another advantage with this method is that the row multipliers used to reduce $A$ to a diagonal matrix may be saved to use on another vector or matrix $B$ with the same coefficient matrix $A$.

A second group is comprised of iterative methods. In these the matrix $A$ is broken up so that a system of the form $x^{k+1} = Mx^k + d$ results. The method begins with an arbitrary starting vector $x^0$ (which may be the zero vector), and will
converge or not for a given linear system depending on the spectral radius of $M$ (22, p. 13). The Jacobi method results if $A = D + (L + U)$ and the Gauss-Seidel method if $A = (D + L) + U$, where $D$ is a diagonal matrix and $L$ and $U$ are strictly lower and upper triangular matrices, respectively. The iteration matrix for the Jacobi method is $M = -D^{-1}(L + U)$ and for Gauss-Seidel is $M = -(D + L)^{-1}U$. Jacobi is a total-step method in that all components of the solution vector are changed at each iteration; while Gauss-Seidel is a single-step method in that only one component of the approximation to the solution vector is changed at each iteration step.

The third group is comprised of the relaxation methods, which were first suggested by Gauss (2, p. 146) and later by Southwell (20). Gauss believed that his method was particularly effective for certain systems of linear equations which came from the normal matrix of a linear least squares fit. Although his method has never had the popularity which Gauss hoped it would have (2, p. 145), it could be more popular for particular systems with the use of some of the newer interactive display units.

An advantage of the relaxation approach is that the number of iteration steps can be decreased by making the proper choice of the next residual element to eliminate at each step. However, there are no set rules or formulas to follow in making the choice; as a result, over-relaxation and under-relaxation approaches have been developed. Obtaining
efficient results with a relaxation method requires skill and the user plays a most important role. It is easier for the user to choose which residual or residuals should be eliminated for the next step than to try to build all the different algorithms into a program which operates automatically. The residuals and other important information could be displayed at each step on a unit such as the 2260, enabling the user to make his observations, then input the necessary information for the next change.

Iterative methods are sometimes referred to as stationary or non-stationary, depending on whether or not the change at step $k$ is a function of the step parameter, $k$. Fox (7, p. 190) defines a non-stationary scheme as,

$$x^{k+1} = f_k(x^k) + a_k d,$$

where $a_k$ and $f_k$ are functions of $k$. Relaxation methods tend to be non-stationary since the choice of the next change in the solution vector depends on the residual vector at each step. Methods that proceed cyclically through the components of the $x$ vector are considered stationary; examples of such methods are gradient, Kaczmarz, Jacobi, and Gauss-Seidel (7, p. 189; 16, p. 401).
A. Literature Review

An iterative method for solving linear systems developed by de la Garza (8) in 1951 has become known as the Projection method. It can be used as a single-step method, changing one component at each step, or the columns of A can be grouped in blocks and these blocks used at each step. A brief review of the Projection method will be given in Chapter III.

A comprehensive study of the theory of gradient methods in solving linear systems has been made by Householder (10, 11), Fox (7), Varga (22) and others. Forsythe (5) reviews the methods for solving systems through 1953 and also gives an extensive bibliography of references.

Tewarson (21) considers one of the Projection methods in solving a linear system in which the coefficient matrix is considered sparse, that is, a small percentage of the elements are non-zero. This is a method described by Kaczmarz (2, p. 186) which is represented by the equation

\[ \mathbf{x}^{k+1} = \mathbf{x}^k + \left[ \frac{r^k_i}{(a^i, a^i)} \right] \mathbf{a}^i \mathbf{a}^i \]

where \( a^i \) is the i-th row of A and \( r^k_i \) the i-th element of the residual vector.

The matrix A is partitioned by a permutation matrix P so that
\[ PA = \begin{bmatrix} R \\ N \end{bmatrix} \] or
\[ PAX = \begin{bmatrix} R \\ N \end{bmatrix} X = \begin{bmatrix} W \\ V \end{bmatrix} \quad (2.3) \]

where \( R \) is \( m \) by \( n \) and \( N \) is \((n - m) \) by \( n \) and the rows of \( A \) which are orthogonal have been placed in \( R \). Applying equation 2.2 \( m \) times to the equation \( RX = W \) yields

\[ X^{k+m} = X^k + R^T D^2 (W - RX^k) \quad (2.4) \]

where \( D \) is a diagonal matrix with elements \( d_{ii} = 1 / (R^1, R^1)^{1/2} \).

Equation 2.2 must still be used on \( NX = V \). The use of equation 2.4 saves several steps of calculation and thus saves some computing time.

Keller (13) considers the solutions to a class of singular systems such as the Neuman problem and others by iteration. He shows that the iterative scheme given by

\[ NX^{k+1} = (N - A)X^k + f \quad (2.5) \]

where \( N \) is non-singular, converges to a solution of \( Ax = f \) whenever the solution exists. The iteration matrix for this method is

\[ B = I - N^{-1}A \quad (2.6) \]

and has different convergence properties than when \( A \) is non-singular, that is, all the eigenvalues of \( B \) are less than one.
in modulus.

The major results are best stated by giving the first theorem:

**Theorem 1.** Let \( Ax = f \) have a solution and \( N \) be non-singular. Then the following three statements are equivalent:

a) For every \( x^0 \) the sequence \( \{x^k\} \) of equation 2.5 converges to a solution of \( Ax = f \).

b) For every \( e_0 \) the sequence \( \{e_k\} \) converges to a vector in the null space of \( A \), \( e_{k+1} = Be_k \).

c) For some \( \mathbf{g}(A) \), \( B \) is reduced by \( \left\{ \mathbf{g}(A), \mathbf{f}(A) \right\} \); \( B \) is the identity on \( \mathbf{g}(A) \) and convergent on \( \mathbf{f}(A) \) where \( \mathbf{g}(A) \) denotes the null space of \( A \) and \( \mathbf{f}(A) \) is any complement of \( \mathbf{g}(A) \).

Keller then considers \( A \) to be Hermitian and positive semidefinite and shows that matrix \( B \) of equation 2.6 is convergent if and only if \( A \) is positive semidefinite.

In order to obtain this result he has defined matrix \( B \) to be convergent if and only if part c of Theorem 1 holds. Having a theorem for positive semidefinite matrices, it is an easy step to apply it to positive definite matrices.

Keller's theorems and results seem to be practical for solving an ill-conditioned system; however, he does not give much information on the matrix \( N \) and its relationship to \( A \), other than to say that \( N \) is non-singular.

Forsythe and Moler (6, p. 49) consider iteration as a
means of improving the accuracy of a solution to \( Ax = b \) after a first approximation \( x^1 \) has been found by a direct method such as Gaussian elimination. The key to success is to calculate the residual vectors, \( r^1, r^2, \ldots, r^k \) with a precision of accuracy higher than that used to calculate \( x \). Then the system \( Az^1 = r^1 \) is solved to yield

\[
r^2 = b - Ax^2,
\]

where \( x^2 = x^1 + z^1 \). These steps are continued until the desired accuracy is reached. The cost in time and calculations is only slightly increased since the matrix \( A \) is the same each time and the elimination multipliers of the Gaussian elimination can be retained.
III. REVIEW OF THE PROJECTION METHOD

The projection method for solving an algebraic system of linear equations, defined by $Ax = b$, is a member of the class of methods known as gradient methods. The coefficient matrix $A$ will be considered to be non-singular and $b$ a vector. The gradient methods attempt the solution by minimizing a quadratic form. The choice of the quadratic form leads to different methods of solving the linear system. Some of these methods are projection, gradient, steepest descent, and conjugate gradient. Projection methods are so named because the residual vector at each step is orthogonal to one or more columns of the coefficient matrix $A$. An $N$-dimensional projection method ($N \leq n$) is one such that the residual vector is orthogonal to $N$ column vectors of $A$ (10, p. 51).

A one-dimensional projection method can be derived by minimizing the quadratic form $(r^k, r^k)$ where $r^k$ is the residual vector defined by

$$ r^k = b - Ax^k $$

$x^k$ being the approximation to the true solution vector $x$ at the $k$-th iteration step (7, p. 205). Let the change in $x$ at the $k$-th step be $a_k w^k$ for some scalar $a_k$ and vector $w^k$, then the new approximation to the solution vector $x$ is
\[ x^{k+1} = x^k + \alpha_k w^k \]  \hspace{1cm} (3.2)

with

\[ \alpha_k = \frac{(u^k, r^k)}{(u^k, u^k)} \]  \hspace{1cm} (3.3)

where \( u^k = Aw^k \).

Choosing \( w^k \) as \( e_i \), \( i = 1 + n \backslash k \), \( e_i \) being the \( i \)-th column vector from the identity matrix \( I \), \( Ae_i \) becomes the \( i \)-th column from the matrix \( A \). Thus, equation 3.3 becomes

\[ \alpha_k = \frac{(a_i, r^k)}{(a_i, a_i)} \]  \hspace{1cm} (3.4)

which gives the change for the \( k \)-th step. The residual vector after the \( k \)-th step can be found by

\[ r^{k+1} = r^k - \alpha_k a_i. \]  \hspace{1cm} (3.5)

The square of the length of this vector is given by

\[ (r^{k+1}, r^{k+1}) = (r^k, r^k) - 2\alpha_k (r^k, a_i) + \alpha_k^2 A_i \]
\[ = (r^k, r^k) - 2\alpha_k^2 A_i + \alpha_k^2 A_i \]
\[ = (r^k, r^k) - 2\alpha_k A_i. \]  \hspace{1cm} (3.6)

The convergence for the Projection method is assured for non-singular coefficient matrices; however, it may be quite
slow in some cases (14). In the one-dimensional Projection method a change is calculated at each step for only one component of the solution vector. Hence the method is referred to as a single-step method, as opposed to a total step method such as Jacobi's method in which all components of the solution vector are changed at each step. The columns of A are usually taken in cyclic order, that is, 1, 2, 3, ..., n, since many more calculations and comparisons would be required to choose a different ordering. A cycle in the Projection method would consist of n single steps, usually beginning with the first column; however, there is nothing to prevent the initial column from being any arbitrary column of A.
IV. A TWO-DIMENSIONAL PROJECTION ALGORITHM

The one-dimensional Projection method is considered to be a stationary method in that the columns of the matrix $A$ are used in a cyclic order, independent of the iteration step parameter $k$. A two-dimensional algorithm for accelerating the Projection method was proposed by Shen (18) in his Ph.D. dissertation at Iowa State University, and presented at the SIAM National Meeting in July of 1970 by Keller (15).

The basic idea of the algorithm is that successive corrections are made to two adjacent components of the solution vector; for example, $x_i^k$ and $x_{i+1}^k$, $i = 1, 2, ..., n - 1$; until no further corrections can be made. Then another pair is chosen with $x_{i+1}^k$ and $x_{i+2}^k$ for the same sequence of operations. These sequences of changes are computed by summing an infinite series so that in actuality only one change is computed per pair (18, p. 16). This algorithm using adjacent pairs of columns is equivalent to a two-dimensional Projection method (10, p. 51). The present work is an extension of this idea by looking closer at the matrix $A$. An optimal ordering of the columns of $A$ is sought in order to increase the rate of convergence of the Projection method. This ordering is based on a function involving the cosine squared of the angle between each of the column vectors of $A$. These function values are greater than or equal to one, and will equal one if the columns of $A$ are orthogonal. The function values approach one
for a matrix which is strongly diagonally dominant and symmetric. A general matrix may have one or more large function values and it is the largest of these which plays an important role in this investigation.

Computationally, the original algorithm may be stated as follows, where the subscripts i and j represent any pair of components of the columns of A:

\[
\alpha_{j1} = (r^k, a_j) / A_j
\]

\[
r_{kl} = r^k - \alpha_{j1}a_j
\]

\[
\alpha_{11} = (r_{kl}, a_1) / A_1
\]

\[
r_{k2} = r_{kl} - \alpha_{11}a_1
\]

\[
\alpha_{j2} = (r_{k2}, a_j) / A_j
\]

\[
\Delta x^k_i = \alpha_{11}c^i_j; \quad \Delta x^k_j = \alpha_{j1} + \alpha_{j2}c^i_j
\]

\[
r_{k+1} = r^k - \Delta x^k_i a_i - \Delta x^k_j a_j
\]

The need for computing the intermediate values of \(r_{kl}\) and \(r_{k2}\) can be eliminated by the following substitutions and definitions:
\[
\alpha_j = \frac{(r^k, a_j)}{A_j}
\]

\[
\beta_1 = \frac{((r^k - \alpha_j a_j), a_1)}{A_1} = \frac{[(r^k, a_1) - \alpha_j (a_1, a_j)]}{A_1}
\]

\[
= \frac{[(r^k, a_1) - \alpha_j A_j^1]}{A_1}
\]

\[
\Delta x^k_1 = \beta_1^1 c_j^1
\]

\[
\alpha_{j2} = \frac{(r^{k2}, a_j)}{A_j} = \frac{((r^k - \alpha_j a_j - \beta_1 a_1), a_j)}{A_j}
\]

\[
= \frac{[(r^k, a_j) - (r^k, a_j) - ([(r^k, a_1) - \alpha_j A_j^1] / A_1) A_j^1]}{A_j}
\]

\[
= -\beta_1 A_j^1 / A_j
\]

\[
\Delta x^k_j = \alpha_j + \alpha_{j2} c_j^1 = \frac{\alpha_j - \beta_1 A_j^1 c_j^1}{A_j}
\]

\[
= \alpha_j + \Delta x^k_1 A_j^1 / A_j
\]

\[
r^{k+1} = r^k - \Delta x^k a_1 - \Delta x^k a_j.
\]

Thus we obtain the computational sequence:

\[
\Delta x^k_1 = \frac{[(r^k, a_1) - (r^k, a_j) A_j^1]}{A_j} / A_j^1 / A_1
\]

\[
\Delta x^k_j = \frac{[(r^k, a_j) - \Delta x^k a_j]}{A_j}
\]

\[
r^{k+1} = r^k - \Delta x^k a_1 - \Delta x^k a_j.
\]

Shen (18) uses \( j \) as \( i + 1 \) until \( i = n \), then \( j \) returns to one and cycles in this manner. However, there is nothing in the
algorithm to prevent \( i \) and \( j \) from being any combination between one and \( n \).

The following lemmas will be proved by considering the computational algorithm in equations 4.12. Let us assume that the computation has progressed to the \( k \)-th step and that we are trying to find \( r^{k+1} \) from \( r^k \) by changing \( x_i^k \) and \( x_j^k \) which correspond to columns \( a_i \) and \( a_j \) from the matrix \( A \).

**Lemma 4.1**

The resultant vector \( r^{k+1} \) is orthogonal to both columns \( a_i \) and \( a_j \).

**Proof** This follows from Shen (18, p. 10) or Householder (10, p. 30).

**Lemma 4.2**

For \( k > 0 \) let \( a_i \) be chosen so that it is one of the columns used in the previous step, then the computational algorithm may be reduced to:

\[
\Delta x_j^k = (r^k, a_j) \frac{C_j}{A_j}; \quad \Delta x_i^k = -\Delta x_j^k \frac{A_j}{A_i},
\]

**Proof** In equation 4.12, \((r^k, a_i)\) will be zero by lemma 4.1. Thus,

\[
\Delta x_i^k = -(r^k, a_j) \frac{A_j^1 C_j^1}{A_1 A_j} \quad \text{and}
\]

\[
\Delta x_j^k = \left[ (r^k, a_j) + (r^k, a_j) (A_j^1)^2 \frac{C_j^1}{A_1 A_j} \right] / A_j
\]

\[
= (r^k, a_j) (1 + \frac{(A_j^1)^2}{A_1 A_j - (A_j^1)^2}) / A_j
\]
\[ \Delta x_j^k = (r^k, a_j) C_j^1 / A_j = a_j C_j^1 \quad 4.13a \]

Thus,

\[ \Delta x_1^k = -[(r^k, a_j) C_j^1 / A_j] A_j^1 / A_1 = -\Delta x_j^k A_j^1 / A_1 \quad 4.13b \]

The residual \( r^{k+1} \) is still calculated as given in equation 4.12c. Thus equations 4.12 or 4.13 may be used for calculating the next change, depending on how close \((r^k, a_1)\) is to zero. The change in the magnitude of \( r^k \) to \( r^{k+1} \) is defined as \( \Delta r^k = (r^k, r^k) - (r^{k+1}, r^{k+1}) \).

**Lemma 4.3**

The change \( \Delta r^k \) is \( C_j^1 (v, v) \) where \( v = (a_1, r^k) a_1 / A_1 \)

\[ = (a_j, r^k) a_j / A_j, \] when equations 4.12 are used and is \( C_j^1 (r^k, a_j)^2 / A_j \) when equations 4.13 are used.

**Proof**

For convenience let \( a_1 = (r^k, a_1) / A_1 \); then

\[ \Delta x_1^k = (a_1 - a_j A_j^1 / A_1) C_j^1 \]

\[ \Delta x_j^k = a_j - \Delta x_j^k A_j^1 / A_j. \]

Now consider the magnitude of \( r^{k+1} \):

\[ (r^{k+1}, r^{k+1}) = ((r^k - \Delta x_j^k a_j - \Delta x_j^k a_j), (r^k - \Delta x_j^k a_j - \Delta x_j^k a_j)) \]
\[ \Delta r^k = (r^k, r^k) - ((\Delta x^k_{a_1} + \Delta x^k_{a_j}), r^k) + (\Delta x^k_{a_1} + \Delta x^k_{a_j})^2 \]

\[ \begin{align*}
\Delta r^k &= 2[\Delta x^k_1 (a_1, r^k) + \Delta x^k_j (r^k, a_j)] \\
&\quad - [(\Delta x^k_1)^2 A_1 + 2\Delta x^k_1 \Delta x^k_j A^1_j + (\Delta x^k_j)^2 A_j] \\
&= 2[\Delta x^k_1 a_1 A_1 + (a_j A_j - \Delta x^k_1 A^1_j) a_j] - [(\Delta x^k_1)^2 A_1 \\
&\quad + 2\Delta x^k_1 (a_j - \Delta x^k_1 A^1_j / A_j) A^1_j \\
&\quad + (A_j a^2_j - 2\Delta x^k_1 a^1_j A^1_j + (\Delta x^k_1)^2 (A^1_j)^2 / A_j)] \\
&= 2\Delta x^k_1 a_1 A_1 + a^2_j A_j - 2\Delta x^k_1 a^1_j A^1_j - (\Delta x^k_1)^2 A_1 - 2\Delta x^k_1 a^1_j A^1_j \\
&\quad + 2(\Delta x^k_1)^2 (A^1_j)^2 / A_j - A_j a^2_j + 2\Delta x^k_1 a^1_j A^1_j \\
&\quad - (\Delta x^k_1)^2 (A^1_j)^2 / A_j \\
&= 2\Delta x^k_1 (a_1 A_1 - a^1_j A^1_j) - 2(\Delta x^k_1)^2 ([A^1_j A_1 - (A^1_j)^2] / A^1_j A^1_j) + c^1_j A^1_j. \\
\end{align*} \]

Substituting for \( \Delta x^k_1 \) in terms of \( a_1 \) and \( c^1_j \) and simplifying yields:

\[ \Delta r^k = c^1_j (a^2_1 A_1 - 2a_1 a^1_j A^1_j) + a^2_j (A^1_j c^1_j + A^1_j A^1_j) / A^1_j \]

\[ \begin{align*}
&= c^1_j (a^2_1 A_1 - 2a_1 a^1_j A^1_j) + a^2_j (A^1_j c^1_j + A^1_j A^1_j) / A^1_j \\
&= c^1_j (a^2_1 A_1 - 2a_1 a^1_j A^1_j) + a^2_j c^1_j A^1_j \\
&= c^1_j (a^2_1 A_1 - 2a_1 a^1_j A^1_j) + a^2_j A^1_j \quad 4.14 \\
&= c^1_j ([a_1 a_1 - a_1 a_j], [a_1 a_1 - a_1 a_j]) \quad \text{Q.E.D.} \\
\end{align*} \]

If equations 4.13 are used, then \( a_1 = (r^k, a_1) / A_1 = 0, \)
so that

$$\Delta r^k = c_j^i (a_j^2 A_j) = c_j^i [(r^k, a_j)^2] / A_j$$  \hspace{1cm} 4.15

as required.

Looking at equation 4.14, the order of choosing the columns $a_1$ and $a_j$ is immaterial since the equation is symmetric in the $a$'s and $A$'s. The change $\Delta r^k$ is seen also to be non-negative for any choice of $a_1$ and $a_j$ and can be zero only if $r^k$ is orthogonal to both columns used.

**Lemma 4.4**

If the columns $a_1$ and $a_j$ used for correction at step $k$ are orthogonal, then the algorithm is equivalent to the Projection method at this step.

**Proof**

$$c_j^i = 1 / [1 - (A_j^1)^2 / A_1 A_j].$$ If $a_1$ and $a_j$ are orthogonal then $(a_1, a_j) = A_j^1 = 0$ and $c_j^i = 1$. Thus,

$$\Delta r^k = c_j^i (a_j^2 A_j) = (a_j^2 A_j)$$

which is the same as the change given in equation 3.6 for the Projection method. Keller (15, p. 19) notes that the algorithm used by Shen (18) reduces to the regular Projection method if the columns of $A$ are pair-wise orthogonal.

**Lemma 4.5**

Assume that $k > 0$ and $i$ is fixed for all $k$ so that $(r^k, a_1) = 0$, then the maximum reduction in $(r^{k+1}, r^{k+1})$ at a
given step is obtained by choosing \( j \) such that \( C_j^1 (r^k, a_j)^2 / A_j \) is a maximum for \( j = 1, 2, \ldots, n \).

**Proof**  The proof for lemma 4.5 follows directly from lemma 4.3 since the total change in the magnitude of \( r^{k+1} \) is given by \( C_j^1 (r^k, a_j)^2 / A_j \).

Lemma 4.5 gives a workable acceleration algorithm; however, for large systems it would be quite time-consuming, because \( n - 2 \) inner products and \( n - 2 \) divisions must be calculated at each iteration step.

A second algorithm could be proposed which would not restrict the choice of \( a_j \). This algorithm would use equations 4.12 at each step since \( r^k \) would not necessarily be orthogonal to either column of \( A \) which might be chosen. The maximum change at each step would be found by maximizing the quantity \( C_s^m (v, v) \) with \( v = (a_m a_m - a_s a_s) \) for \( s = 1, 2, \ldots, n \), \( m = 1, 2, \ldots, n \). This would also result in a prohibitively large amount of computation for a general linear system.

**Lemma 4.6**

If \( C_{n+1}^1 \) is undefined then \( r^0 \) is parallel to column \( a_i \) of the coefficient matrix \( A \) and the solution to the linear system can be found without iteration.

**Proof**  Since \( C_j^1 = 1 / (1 - \cos^2 \theta_{i,j}) \), with \( \theta_{i,j} \) being the angle between column \( a_i \) and \( a_j \), \( \cos^2 \theta_{i,j} \) cannot be one for any \( i \) and \( j \) between one and \( n \) or otherwise the coefficient matrix is singular. Thus, this situation will only occur for \( j = n + 1 \). When \( \cos^2 \theta_{i,n+1} = 1 \) then column \( a_i \) and \( r^0 \) are
parallel so that $r^0 = \beta a^i$ for some real scalar $\beta$ which can be found from $\beta = r^0_s / a^i_s$, $a^i_s \neq 0$ so that $x = \beta e^i$.

All of the lemmas have been concerned with a single step involving the two columns $a^i$ and $a^j$ with $i$ and $j$ chosen by some algorithm. A cycle is considered to be a group of $m$ steps with $m \leq n$ with the provision that each column of $A$ must be included at least once in the $m$ steps. The total change in the magnitude of the residual vector per cycle is given by the next lemma.

**Lemma 4.7**

Let $k > 0$ be fixed and consider the next $m$ steps. Assume that the changes in the solution vector for one cycle make use of the columns of $A$ so that the elements of $C$ which are used are $C_{11}, C_{12}, \ldots, C_{jm}$, $m \leq n$, then the total change in the magnitude of $r$ is given by $C_{11} (v^1, v^1) + C_{j2} (v^2, v^2) + \ldots + C_{jm} (v^m, v^m)$.

**Proof** The change in the magnitude of $r$ at the first step is $\Delta r^k = (r^k, r^k) - (r^{k+1}, r^{k+1}) = C_{11} (v^1, v^1)$, where $v^1$ is a function of $a^i, a^j$, and $r^k$.

The change at the second step is

$$\Delta r^{k+1} = (r^{k+1}, r^{k+1}) - (r^{k+2}, r^{k+2}) = C_{j2} (v^2, v^2),$$

where $v^2$ is a function of $a^i, a^j, a^{i2}, a^{j2}$, and $r^{k+2}$.

Thus the change at step $m$ is
\[ \Delta r^{k+m} = (r^{k+m-1}, r^{k+m-1}) - (r^{k+m}, r^{k+m}) = C_{jm}^{im} (v_m, v_m), \]

where \( v_m \) is a function of \( a_{jm}, a_{jm}, \) and \( r^{k+m-1}. \)

The total change for the \( m \) steps is

\[
\Delta r^k + \Delta r^{k+1} + \ldots + \Delta r^{k+m}.
\]

\[
\sum_{s=0}^{m-1} \Delta r^{k+s} = (r^k, r^k) - (r^{k+1}, r^{k+1}) + (r^{k+1}, r^{k+1})
- (r^{k+2}, r^{k+2}) + \ldots + (r^{k+m-1}, r^{k+m-1}) - (r^{k+m}, r^{k+m})
= C_{jl}^{il} (v_1, v_1) + C_{jl}^{i2} (v_2, v_2) + \ldots + C_{jl}^{im} (v_m, v_m).
\]

Thus, the total change is

\[
(r^k, r^k) - (r^{k+m}, r^{k+m}) = \sum_{s=1}^{m} C_{jl}^{is} (v_s, v_s),
\]

which proves the lemma.

---

A. Ill-Conditioned Linear Systems

When using an iterative method to solve the linear system \( Ax = b \) the user may become involved with an ill-conditioned system in which the convergence would be too slow to warrant the use of this method. Thus, it would be helpful to have a built-in method for testing to see if matrix \( A \) is ill-conditioned. Stanton defines an ill-conditioned system as one
whose solution is sensitive to small changes in the coefficient matrix \((19, p. 204)\).

There are several methods of testing a system, but these involve many computations not used in the iterative method; thus, making such a test would diminish any advantage that might be provided by the iteration method. Some of these tests as given by Bodewig \((2, p. 135)\) are:

1. the number \(N(A)N(A^{-1})/n\) where \(N(A) = \sum |a_{1k}|^2\);
2. the number \(n m(A)m(A^{-1})\) where \(m(A) = \max |a_{1k}|\);
3. the number \(\lambda_1/\lambda_n\) where \(\lambda_1\) is the largest eigenvalue and \(\lambda_n\) is the smallest eigenvalue \((\lambda_1\) and \(\lambda_n\) represent the moduli of the respective eigenvalues);
4. the number given by the quotient of the maximum term in \(\det(A)\) with \(\det(A)\) in absolute value.

The first two tests involve the inverse of the matrix, the third the eigenvalues and the last a term from the determinant of \(A\). Only the last could be approximated without a great deal of work; however, the evaluation of the \(\det(A)\) would be equivalent to solving the system by elimination.

A method of testing for ill-conditionedness suggested by Stanton \((19, p. 207)\) can be applied satisfactorily to the acceleration algorithm because the values needed are calculated as part of the computational algorithm. This test considers each equation as a hyperplane and looks at the angles between their normals.

If any of these angles are close to zero then the
hyperplanes are very nearly parallel and the system will be ill-conditioned. When one or more of these angles are small, then the cosine squared of the angle will approach one. The matrix denoted by $C$ in the computational algorithm is a symmetric matrix defined by $C_{ij} = 1 / (1 - \cos^2 \theta_{ij})$, where $\theta_{ij}$ is the angle between the $i$-th and $j$-th column of $A$. Thus when the value of $\cos^2 \theta_{ij}$ is near one, the value of $C_{ij}$ is large; for example, if $\cos^2 \theta_{ij} \geq .99$ then $C_{ij} \geq 100$. From the test cases using this algorithm, any matrix $A$ having a component of $C$ greater than 100 has been very, very slow in convergence. Thus, the value 100 can be used as the cut-off line for considering a system to be ill-conditioned. This does not catch all cases of ill-conditioned systems; for instance, it does not identify a system in which one of the rows is very nearly a linear combination of one or more of the other rows.

The following example given by Bodewig (2, p. 135) illustrates his four tests for ill-conditioned systems versus the maximum component of the matrix $C$:

\[
A = \begin{bmatrix}
5 & 7 & 6 & 5 \\
7 & 10 & 8 & 7 \\
6 & 8 & 10 & 9 \\
5 & 7 & 9 & 10
\end{bmatrix}
\]

Figure 1. An example of a coefficient matrix which is ill-conditioned for solving $Ax = b$. 
Figure 2. Matrix C for the matrix in figure 1 with the entries rounded to integers.

The values for the five tests are:

1. \( N(A)N(A^{-1}) / n \) = 750
2. \( n \text{ m}(A) \text{ m}(A^{-1}) \) = 2700
3. \( |\lambda_1| / |\lambda_n| \) = 3000
4. \( \left| \frac{\text{max term of det}(A)}{\text{det}(A)} \right| \) = 1600
5. \( c_j^A = c_2^A \) = 1360
V. AN OPTIMAL ORDERING ALGORITHM FOR SOLVING $Ax = b$

A non-stationary iterative method for solving the linear system $Ax = b$ can be obtained from lemma 4.5 with a slight modification. The maximum reduction in the magnitude of the residual vector at each step is given by maximizing the product $C_q^s (v,v)_{q,s}$, $q \neq s$, over all of the columns of $A$; however, this leads to a large amount of work for large systems.

Approximately the same reduction can be obtained by considering the values of $C$ as fixed and finding a maximum value for $(v,v)$. Let us consider the following lemma in which a value for $i$ is fixed on a row of $C$ which contains a maximum element. This will insure that the maximum element of $C$ is included as many times as possible in the iteration steps of the algorithm.

Lemma 5.1

Assume that $i$ is fixed, $(r^k, a^k_i) = 0$ and $(r^k, r^k) > \max C^i_j$ for $k > 0$; then the maximum reduction in $(r^{k+1}, r^{k+1})$ at a given step $k$ is attained by choosing $j = s$ such that $(r^k, a^s_k)^2 / A_s$, $s = 1, 2, \ldots, n$, is a maximum value.

Proof From lemma 4.5, the total change is given by $C^i_j (r^k, a^k_j)^2 / A_j$. Let us assume that the inner product of $(r^k, a^s_j)$ is large compared to $C^i_j$ as the iteration begins. The elements of $C$ are fixed by the coefficient matrix $A$, thus the product will be largely a function of the inner product of $r^k$ with the columns of $A$. The cosine squared of the angle
between \( a_j \) and \( r^k \) is \( (r^k, a_j)^2 / (R^k A_j) \), which is < 1. Thus

\[
(r^k, a_j)^2 / A_j < (r^k, r^k),
\]

so that

\[
C_j (r^k, a_j)^2 / A_j < C_j (r^k, r^k).
\]

Therefore, as long as \( (r^k, r^k) \) is greater than the max \( C^i \), the residual vector will have an important role in determining the change at each step.

In like manner, when \( (r^k, r^k) \) becomes less than the min \( C^i \), then the values of \( C^i \) will control the product

\[
C_j (r^k, a_j)^2 / A_j.
\]

Now the value of the next \( j \) cannot be chosen by looking for the largest value from the \( i \)-th row of \( C \) each time, as this will cause the choice of the next column to rotate between two values and the iteration method would probably not converge for the general case.

This last comment leads to the question of what should be done when \( (r^k, r^k) \) is between the max \( C^i \) and the min \( C^i \). To find the maximum change at this stage in the iteration sequence would require more work than can be justified by the amount of reduction which is brought about in the norm of \( r^k \). Thus, it is advantageous to choose some fixed value or function of \( C^i \), for example max \( C^i \), min \( C^i \), or average of these two, and use this value as an arbitrary dividing line.

Lemma 5.1 gives a method for selecting the column \( a_j \) to pair with a fixed column \( a_i \) to obtain successive reductions in
the magnitude of $r^k$ until the magnitude becomes less than the max $C^1$. Now an algorithm is desirable which will include all of the columns of $A$ in some cyclic fashion in order to make it more likely that the method will converge to the true solution. The one-dimensional Projection method uses the columns in cyclic order, that is, $a_1, a_2, ..., a_n$; Shen's two-dimensional Projection method (18) uses the columns in pairs, again in cyclic order, $(1,2), (2,3), ..., (n,1)$. We are looking for an optimal ordering of the columns which will produce the most rapid convergence to the solution of a linear system.

Lemma 4.3 gives the change in the magnitude of $r^k$ from step $k$ to $k + 1$ as $C^i_j (v,v)$ in which the vector $v$ is a function of $a_i, a_j,$ and $r^k$, and lemma 5.1 indicates that the role of $C^i_j$ becomes more important as $(v,v)$ becomes smaller.

Let us assume that we have a convergent method and that there exists a $K$ such that for all $k > K$, the magnitude of the residual vector is less than one so that the values of $C$ become important in the amount of reduction obtained at each step. There are many ways to order the columns of $A$ to obtain an iterative solution to the system $Ax = b$. When a pair of columns $a_i$ and $a_j$ are chosen at a particular step $k$, then the element $C^i_j$ from $C$ is associated with the change in the magnitude of $r^k$. We will consider a cycle to be $m$ steps, $m \leq n$ (in general $m = n - 1$), with the requirement that all $n$ columns of $A$ be included at least one time. Some of the ways in which the elements of $C$ (and the columns of $A$) could be
ordered to obtain a two-dimensional algorithm are:

1. \( \beta_n = \{c_2^1, c_3^2, \ldots, c_{n-1}^n, c_1^n\} \),
2. \( \beta_m = \{c_2^1, c_3^2, \ldots, c_{n-1}^n\} \),
3. \( \beta_l = \{c_2^1, c_3^1, \ldots, c_{n}^1\} \),
4. \( \beta = \{c_2^1, c_{j_2}^{12}, \ldots, c_{jm}^{1m}\} \).

The sequence \( \beta \) has the property that \( c_2^1 \geq c_{j_2}^{12} \geq \ldots \geq c_{jm}^{1m} \) and may be equal to \( \beta_m \) or \( \beta_l \) for certain systems.

The following lemmas indicate ways to choose \( a_1 \) and \( a_j \) at step \( k \) to get a maximum reduction in the magnitude of the residual vector.

**Lemma 5.2**

Assume that the columns of \( A \) are ordered so that \( c_2^1 \) is a maximum for \( A \) and that \( a_1 \) and \( a_2 \) are used at step \( k \) to calculate the respective changes in \( x^k \) and \( r^k \). For a fixed \( r^k \), if \( a_1 \) and \( a_2 \) are not both zero and the ratio \( \frac{c_2^1}{c_q^s} \) is

\( (v,v)_{s,q} / (v,v)_{1,2} \) for \( s = 1, 2, \ldots, n, q = 1, 2, \ldots, n \),

(with \( sq \neq 2 \)), then the change in \( r^k \) will be a maximum for step \( k \).

**Proof** First, we note that \( v = \alpha_1 a_1 - \alpha_j a_j \) for an arbitrary pair of vectors \( a_1 \) and \( a_j \) cannot be zero for a non-singular matrix \( A \) when \( \alpha_1 \) and \( \alpha_j \) are not both zero; however, if \( A \) is ill-conditioned for solving the linear system, \( (v,v) \)
may be close to zero for some choices of $a_1$ and $a_j$. If

\[
\frac{c_2^1}{c_2^s} > \left[ (v,v)_{s,q} / (v,v)_{1,2} \right],
\]

then

\[
c_2^1 (v,v)_{1,2} > c_2^s (v,v)_{s,q}
\]

for all permissible values of $s$ and $q$. The change in the magnitude of $r^k$ is

\[
\Delta r^k = (r^k, r^k) - (r^{k+1}, r^{k+1}) = c_2^1 (v,v)_{1,2}
\]

when $a_1$ and $a_2$ are used at step $k$. Thus we have a maximum change which proves the lemma.

**Lemma 5.3**

Assume that the columns of $A$ are so ordered that $c_2^1$ is a maximum for $A$ and that the $m$ maximum values of $C$ fall on the superdiagonal of $C$ so that $c_2^1 \geq \min C$. For a fixed $r^k$, if $a_1$ and $a_2$ are not both zero, and lemma 5.2 is satisfied, then the reduction in $r^k$ is greater for $n$ steps of the sequence $\beta_m$ than for $n$ steps of the sequence $\beta_n$.

**Proof** The reduction at the end of the first $m$ steps is the same for both sequences $\beta_n$ or $\beta_m$. The reduction for the next step is due to $c_2^{n,2}A_1$ for $\beta_n$, or $c_2^1 (v,v)_{1,2}$, $v = a_1a_1 - a_2a_2$, for $\beta_m$. By lemma 5.2, $c_2^1 (v,v)_{1,2} > c_2^{n,2}A_1$ so that $(r^k, r^k)$ is smaller after $n$ steps of $\beta_m$.

Lemma 5.3 presents the possibility of saving several
iteration steps for solving the linear system by a cyclic two-dimensional method when \( A \) has the proper form.

**Lemma 5.4**

Let \( a_i \) and \( a_j \) be used for the change at step \( k \) and assume that \( \lim_{k \to \infty} \frac{(v,v)_s,q}{(v,v)_i,j} \leq 1 \) for any permissible value of \( s \) and \( q \) at step \( k \), then an iteration method with the columns of \( A \) ordered as in sequence \( \beta \) produces the maximum change in the magnitude of \( r^k \) at each step.

**Proof** The sequence \( \beta \), for ordering the columns of \( A \), has \( C^1_2 \geq C^1_j \geq \ldots \geq C^m_j \); thus the ratio of \( C^i_j \) to any one of those following it is \( \geq 1 \); therefore, applying lemma 5.2, the product \( C^i_j (v,v)_i,j \) at each step is at least as great as any other permissible product and is greater when the strict inequality holds between the values of \( C \).

A non-stationary algorithm results by combining lemmas 5.2 and either 5.3 or 5.4. The magnitude of the residual vector is reduced by choosing the pairs of columns of \( A \) so that \( (v,v)_i,j \) is a maximum at each step until \( (r^k,r^k) \) is less than some predetermined value related to the \( C \) matrix. A cyclic method is then used to be sure all columns of \( A \) are covered. This method could use sequence \( \beta_n \), \( \beta_m \), \( \beta_1 \), \( \beta \) or some other combination. The best method seems to be related to the maximum elements of the \( C \) matrix and the rate at which the inner products of \( r^k \) and the columns of \( A \) approach zero as \( k \to \infty \). We will look at a method in Chapter VI which uses \( \beta_1 \) with lemma 5.2 and compare this method with some other iterative methods.
VI. A NON-STATIONARY ITERATIVE ALGORITHM

A. The Ordering of the Columns of A

The change in the magnitude of the residual vector at the k-th step is given by \( C_j^1(r^k, a_j)^2 / A_j \) (see lemma 4.3). This product is controlled at first by the ratio \( (r^k, a_j)^2 / A_j \) and then by the elements of \( C^1 \) as the elements of the residual vector decrease in absolute value so that \( (r^k, a_j)^2 \) becomes small. From lemmas 4.5 and 5.1 the maximum reduction at step \( k \) is obtained when \( (r^k, a_j)^2 / A_j \) is a maximum, provided \( (r^k, r^k) \) is greater than the max \( C^1 \).

A non-stationary iterative algorithm for solving the linear system \( Ax = b \) is given in two steps. The first step follows from lemma 5.1 and the second step orders the columns of \( A \) so that the computational algorithm has a minimum amount of calculations. The value of \( i \) is chosen and fixed for both steps of the algorithm. This value corresponds to one of the rows of \( C \) which has the maximum element in \( C^1_j \).

The values assigned to \( j \) cause the elements in the \( i \)-th row of \( C \) to be used in forming the reductions in the magnitude of the residual vector. This special case is the optimal ordering of the columns of \( A \) if the maximum values of the matrix \( C \) actually are in the \( i \)-th row, and if lemma 5.4 is satisfied by \( A \) (see Test Problems 3 and 4). This special case is chosen to implement in this paper since a more involved
algorithm is necessary to choose the m maximum values of C and be sure to include all n columns of A. The advantages of this special case are:

(a) Equations 4.13 can be used for the changes in $x^k$ when $k$ is greater than zero, as $(r^k, a_1) = 0$.
(b) Many of the arithmetic operations may be performed only one time.
(c) Each cycle requires $n - 1$ iterative steps.

B. A Two-Step Algorithm

Step_1:

(a) Choose the initial values of i and j as described in Section C below.
(b) Calculate the change in $x^i_1, x^i_j$ and $r^1$ using equations 6.1 and 6.3 with $x^0 = 0$.
(c) Holding i fixed, choose the next value j such that $(r^1, a_j)/A_j$ is a maximum.
(d) Calculate $Ax^1_i, Ax^1_j$ and $r^2$ using equations 6.2 and 6.3.
(e) Repeat steps (c) and (d) until $(r^k, r^k)<\min \mathbf{C}^k$, or until the ratio of two successive norm values becomes greater than a pre-set value which is less than one, for example, 0.9.

Step_2:

(a) i is still the same value as in Step_1, however, the
elements of $c^i$ have been ordered large to small and the corresponding column values stored in a vector of length $n - 1$.

(b) Cycle on the pairs $a_i, a_j$ where the values of $j$ will be some permutation of the integers 1, 2, ..., $i - 1, i + 1, ..., n$.

(c) Continue step (b) until the norm of the residual vector is less than some predetermined value.

In many examples the algorithm remains in Step_1 for only a few iterations. The major difficulty with this first part is that the selection of $j$ could alternate between two columns of $A$ and thus no changes would be generated in the other components of the solution vector. This would tend to cause very small changes in the residual vector and thus very slow convergence in the system. Staying in Step_1 when this happens is expensive in time and calculations, so we move to Step_2 even though $(r^k, r^k)$ is not as small as we would like it to be. Step_2 will cause each of the columns of $A$ to be included which tends to improve the rate of convergence as compared to remaining in Step_1.

C. Initial Calculations

The change in the $i$-th and $j$-th components of $x^k$ is found by using equations 4.12 and 4.13. For $k = 0$ the change in $x^k_1$ is:
\[ \Delta x_1^k = \left[ (r, a_1) - (r, a_j) \right] A_j / A_1 \]
\[ = (a_1 - a_j A_j / A_1) C_j^i / A_1 \]  
6.1a

and the change in \( x_j^k \) is

\[ \Delta x_j^k = \left[ (r, a_j) - \Delta x_1^k A_j \right] / A_j \]
\[ = a_j - \Delta x_1^k A_j / A_j \]  
6.1b

For \( k > 0 \), \( (r, a_1) \) is zero, thus equations 6.1 reduce to

\[ \Delta x_j^k = a_j C_j^i \]  
6.2a

and \( \Delta x_1^k = - \Delta x_j^k A_j / A_1 \).  
6.2b

The new residual vector is calculated by equation 4.12c,

\[ r^{k+1} = r^k - \Delta x_1^k a_1 - \Delta x_j^k a_j \]  
6.3

in either case.

The matrix \( C \) and all of the column inner products used in equations 6.1 or 6.2 should be calculated before the iterative sequence begins. \( C \) is symmetric with the main diagonal not used, thus only \((n^2 - n) / 2\) elements are calculated. The row and column containing the maximum value of \( C \) is found at the same time. Let the maximum element of \( C \) be denoted by \( C^p \).
Since C is symmetric, rows $a$ and $g$ will contain this maximum value.

The next step is to choose a value for $i$ which will remain fixed throughout the iteration sequence. The value for $i$ is chosen as equal to $a$ or $g$ depending on the larger sum, $\sum C^a$ or $\sum C^g$. This row, $C^i$, will be used later so it is stored in a vector for quick access. The initial value of $j$ is then set to $g$ or $a$, whichever is left.

The inner product of $a_i$ with each of the other columns will also be required, so these values are calculated and stored in a vector. The values actually stored are $(a_i, a_s) / A_i$, $s = 1, 2, 3, \ldots, n$, because equation 6.2b uses these values.
VII. CONVERGENCE

A. Proof of Convergence for a Non-Stationary Method

Theorem 7.1

The non-stationary iterative method described in Section B of Chapter VI converges to the solution vector for all linear systems whose coefficient matrix has a non-zero determinant.

Proof Let the residual vector at the end of the k-th step be denoted by \( r^{k+1} \) and defined by equation 6.3. Suppose that \( r^k \) is not the zero vector, then from equation 4.15 the magnitude of \( r^{k+1} \) is given by

\[
(r^{k+1}, r^{k+1}) = (r^k, r^k) - C_j^1 \left[ a_j^2 A_j \right] \text{ for } k > 0.
\]

Since \( C_j^1 \), \( a_j^2 \) and \( A_j \) are all positive quantities, the magnitude of \( r^k \) must be reduced in going from step \( k \) to \( k+1 \) unless \( (r^k, a_j) = 0 \). However, since step 1 of the algorithm chooses \( (r^k, a_j)^2 / A_j \) to be a maximum value, \( a_j^2 \) cannot be zero unless all of the inner products yield zero, but this implies that \( r^k \) is the zero vector which contradicts the assumption that \( r^k \) is not zero. Thus we will assume there exists an integer \( K \) such that for \( k > K \) the iteration has proceeded to step 2 where the columns of \( A \) are chosen in some fixed order. Now, if \( (r^k, a_{j1}) = (r^k, a_{j2}) = \ldots = (r^{k+m-1}, a_{jm}) = 0 \) for \( m = n - 1 \), since
\((r^k, a_1)\) is assumed zero at all times, the residual vector is orthogonal to all columns of \(A\). However, the only vector orthogonal to a set of \(n\) linearly independent vectors spanning \(n\)-space is the zero vector, which contradicts the assumption that \(r^k\) is not the zero vector. Thus the magnitude of the residual vector must be reduced by some positive amount after \(n - 1\) iteration steps. Thus \(\{(r^k, r^k)\}_{k=1}^{\infty}\) is a monotonic, non-increasing sequence which is bounded below by zero and therefore has a limit (17, p. 27).

To complete the proof it is convenient to first prove the following lemma:

**Lemma 7.1**

The \(\lim (\Delta x^k_j)^2 = 0\) and hence the \(\lim (\Delta x^k_1)^2 = 0\) since \(\Delta x^k_1\) is a function of \(\Delta x^k_j\).

**Proof** From equation 4.12c, \(r^{k+1} = r^k - \Delta x^k_1 a_1 - \Delta x^k_j a_j\).

If we express \(\Delta x^k_1\) in terms of \(\Delta x^k_j\) from equation 4.13b, we get the following expression for an element of \(r^{k+1}\):

\[
\begin{align*}
    r^{k+1}_j &= r^k_j + \Delta x^k_j \left[\left(A^1_j / A_1\right) a_1 - a_j\right], \quad j \neq 1, \\
    \text{and } j &= 1, 2, 3, \ldots, n.
\end{align*}
\]

Now \(r^{k+1}_j - r^k_j\) approaches zero as \(k \to \infty\), and \(\left[A^1_j / A_j\right] a_1 - a_j\) cannot be zero for a non-singular matrix, therefore we have

\[
\lim_{k \to \infty} (r^{k+1}_j - r^k_j) = \left[A^1_j / A_j\right] a_1 - a_j \lim_{k \to \infty} \Delta x^k_j = 0,
\]

which proves the lemma.
From lemma 4.1, \( r^{k+1} \) is orthogonal to \( a_i \) and \( a_j \). Also we note that, for \( i \) fixed, \( a_i \) is orthogonal to all \( r^k \) for \( k > 0 \).

Denote the columns of \( A \) which are used in one cycle by \( j_1, j_2, j_3, \ldots, j_m \). A cycle here is defined as \( m = n - 1 \) steps, since \( a_i \) is used each time. Then we may write

\[
(r^k, a_{js}) = (r^k, a_{js}) - (r^{k+1}, a_{js}) = ([r^k - r^{k+1}], a_{js})
\]

for \( s = 1, 2, \ldots, m \), and some value of \( k \).

Let \( k \) be a fixed value so that

\[
(r^k, a_{js}) = a_{js}^t (r^k - r^{k+s}), \quad s = 1, 2, \ldots, m.
\]

Then for \( s = 1 \) we have from equation 4.16c:

\[
r^k - r^{k+1} = \Delta x_i^k a_i + \Delta x_{j_1}^k a_{j_1}
\]

\[
= \Delta x_i^k a_1 + \Delta x_{j_1}^k a_{j_1}
\]

\[
= A (\Delta x_i^k e_1 + \Delta x_{j_1}^k e_{j_1}).
\]

Define a vector \( w_i^k, w_{j_1}^k = \Delta x_i^k \), and a vector \( d_i^k, d_{j_1}^k = \Delta x_{j_1}^k \) then

\[
r^k - r^{k+1} = A(w_i^k + d_{j_1}^k).
\]

To get \( r^k - r^{k+2} \), consider
\[ r^{k+1} - r^{k+2} = \Delta x_1^{k+1} a_1 + \Delta x_{j2}^{k+1} a_2 \]
\[ = A \left( w^{k+1} + d^{k+1} \right) \]  \hspace{1cm} 7.4

and adding equations 7.3 and 7.4, we obtain

\[ r^k - r^{k+2} = A \left( w^k + w^{k+1} + d^k + d^{k+1} \right). \]

Continuing in this way, we obtain

\[ r^k - r^{k+s} = A \left( \sum_{q=0}^{s-1} \left[ w^{k+q} + d^{k+q} \right] \right). \]

Thus,

\[ a_{js}^t r^k = a_{js}^t A \left( \sum_{q=0}^{s-1} \left[ w^{k+q} + d^{k+q} \right] \right) = u_{js} \]  \hspace{1cm} 7.5

Now we define a matrix \( B^\Psi b_1 = a_{j1}, b_2 = a_{j2}, \ldots, b_m = a_4 \). When all of the columns of \( A \) are used in a cycle, equation 7.5 becomes

\[ B^t r^k = B^t A \left( \sum_{q=0}^{s-1} \left[ w^{k+q} + d^{k+q} \right] \right) = u. \]

Now replace \( r^k \) by \( b - Ax^k \) to get

\[ B^t (b - Ax^k) = u. \]

Then pre-multiply by \( (B^t A)^{-1} \) to get
\[ A^{-1}b - x^k = (B^tA)^{-1}u, \text{ or } \]
\[ x^k = A^{-1}b - (B^tA)^{-1}u; \]

then,
\[ \lim_{k \to \infty} x^k = \lim_{k \to \infty} [A^{-1}b - (B^tA)^{-1}u] \]
\[ = A^{-1}b - \lim_{k \to \infty} (B^tA)^{-1}[B^tA\sum_{q=0}^{s-1}(w^{k+q} + d^{k+q})] \]
\[ = A^{-1}b - \lim_{k \to \infty} \sum_{q=0}^{s-1}(c^{k+q} + d^{k+q}). \]

From the definition the single elements of \( w^{k+q} \) and \( d^{k+q} \) are \( \Delta x_i^{k+q} \) or \( \Delta x_{js}^{k+q} \).

\( \Delta x_i^{k+q} \) is a function of \( \Delta x_{js}^{k+q} \) from lemma 4.2, and

\[ \Delta x_{js}^{k+q} \to 0 \text{ as } k \to \infty \text{ from lemma 7.1}; \text{ therefore} \]
\[ \lim_{k \to \infty} \sum_{q=0}^{s-1}(w^{k+q} + d^{k+q}) = 0; \text{ thus } \lim_{k \to \infty} x^k = A^{-1}b = x. \]

Fox (7, p. 314) defines a matrix \( \beta \) which relates \( r^{k+m} \) to \( r^k \) in an iterative process. This relationship is \( r^{k+m} = \beta^* r^k, \)

where \( r^k \) and \( r^{k+m} \) are one complete cycle apart;

\[ \beta^* = \beta_n^* \beta_{n-1}^* \cdots \beta_1^*, \] where \( \beta_1^* = I - a_1a_1^T / (a_1, a_1). \) Let us consider the same relationship for the non-stationary method presented here.

From equation 4.129 and equations 4.13,
\[ r^{k+1} = r^k - \Delta x^k a_j - \Delta x^k a_j \]
\[ = r^k + [(r^k, a_j) c^j / (A_j)] a_j - [(r^k, a_j) c^j / A_j] a_j \]
\[ = r^k + [c^j / (A_j)] [a_i - A_j a_j] \]
\[ = r^k + [c^j / (A_j)] [a_i - A_j a_j] a_j r^k \]
\[ = [I + (c^j / [A_j]) (a_i - A_j a_j) a_j] r^k. \]

Since \( r^k \) to \( r^{k+1} \) corresponds to one iteration step, define

\[ \beta_j = [c^j / (A_j)] [a_i - A_j a_j] a_j; \text{ then } \]

\[ r^{k+m} = (I + \beta_m)(I + \beta_{m-1}) \ldots (I + \beta_1) r^k = \beta r^k. \]  \hspace{1cm} 7.6

The general iterative equation is given by

\[ x^{k+1} = Mx^k + c. \] \hspace{1cm} 7.7

The convergence properties of the system are determined from \( M \). In particular, the method converges provided the spectral radius of \( M, \rho(M) \), is less than one. This is in general a difficult quantity to compute for most methods; therefore it is usually estimated after performing several iteration steps.

Using equation 7.6 and substituting for \( r^{k+m} \) and \( r^k \) gives

\[ b - Ax^{k+m} = \beta (b - Ax^k) \]
When the true solution \( x \) is reached, it will also satisfy this equation, thus

\[
b - Ax = \beta(b - Ax).
\]

Subtracting we get

\[
A(x^{k+m} - x) = \beta A(x^k - x).
\]

For this we have \( AB^{p+1} = \beta AB^p \) where \( B^p = x^k - x \) and \( p \) refers to a cycle. Thus,

\[
B^{p+1} = (A^{-1}\beta A) B^p
\]

where \( (A^{-1}\beta A) \) is equivalent to the matrix \( M \). For further discussion on theoretical rates of convergence, see Shen (18) or Varga (22).

B. A Second Stage Acceleration

For many systems there exists a second stage acceleration which was proposed by Aitken (1) and described by other recent authors such as Fox (7) and Conte (3). The conditions for use of this stage come from equation 7.6 where \( \beta \) is assumed to have a single eigenvalue which is less than one in modulus and is simple in that there are no other eigenvalues of equal
modulus; call this value $\lambda_1$. Shen (18) gives a detailed proof showing that this process does apply to a two-dimensional Projection method.

Fox (7, p. 146) gives the conditions which must be met in order to apply this second stage. These conditions are:

\begin{align*}
a) & \quad E^{p+1} = \lambda_1 E^p; \quad E^p = x - x^p \\
b) & \quad \Delta x^{p+1} = \lambda_1 \Delta x^p \\
c) & \quad r^{p+1} = \lambda_1 r^p.
\end{align*}

Conte (3, p. 195) suggests that when the components of three successive iterations show a decreasing or increasing pattern, the system is a good candidate for acceleration. The two-dimensional method described in Chapter VI seems well fitted to this idea since $a_1$ is held fixed, so that the component $x_1$ would tend to settle down much faster than some of the other components; thus, giving a better estimate of $\lambda_1$ before applying the acceleration. This would require four complete cycles of $m$ iteration steps. The corrected $x$ value may be found from

\begin{align*}
a) & \quad x = x^{k+1} - (x^{k+1} - x^k)^2 \big/ (x^{k+1} - 2x^k + x^{k-1}), \quad 7.10 \\
b) & \quad x = x^k + (x^{k+1} - x^k) \big/ (1 - \lambda_1). \quad 7.11
\end{align*}

A new residual is calculated from $r^{k+1} = b - Ax$. If
\((r^{k+1}, r^{k+1})^{\frac{3}{2}}\) is not less than the convergence criterion, then \(x^{k+1} = x\) and the iteration continues.

If the eigenvalues of \(g\) are complex or if \(|\lambda_1|\) and \(|\lambda_2|\) (\(|\lambda_1| > |\lambda_2|\)) are approximately equal in magnitude, then this acceleration may not be useful. However, a more involved system may be developed using five successive estimates rather than three (7, p. 147). This would be cumbersome to use for the general case; however, if it were necessary to solve systems which were known to yield this property, the increase in the speed of convergence would be worth the additional computation required.
VIII. COMPARISONS

A. Numerical Calculations

We can compare iterative methods for solving $Ax = b$ by considering the number of calculations required at each step, or by considering the number of steps required to reach the desired accuracy of the solution vector or by considering the time each method requires to obtain the solution on a given machine. The numerical calculations are given for the four single-step methods used in this comparison. These methods are: Projection (one-dimensional), Gauss-Seidel, Shen's (two-dimensional) and the non-stationary method presented in Chapter VI. Gauss-Seidel is considered in the form of

\[ dx^k_1 = \frac{r^k_1}{a^i_1}, \quad 8.1a \]

\[ x^{k+1}_1 = x^k_1 + dx^k_1, \quad 8.1b \]

\[ r^{k+1}_1 = r^k_1 - dx^k_1a^i_1. \quad 8.1c \]

The basis for the first comparison is the number of multiplications and additions which are required for a single step of the iteration. The calculations required to start a method are not counted in this comparison as they occur only one time. The number of calculations shown for the non-
stationary method are for step_2. Two things should be noted for the non-stationary method: First, step_1 requires more calculations to choose the value of \( j \) at each step, nevertheless the time spent in step_1 should be relatively short for most systems; second, there are \( n - 1 \) more additions and \( n \) more multiplications in the non-stationary method than in Shen's method, because \( (r^k,a_1) \) is calculated and tested each time to choose between equations 6.1 and 6.2.

Following is the comparison of the four methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Multiplications</th>
<th>Number of Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Projection</td>
<td>2n + 1</td>
<td>2n</td>
</tr>
<tr>
<td>(2) Gauss-Seidel</td>
<td>2n + 1</td>
<td>2n + 1</td>
</tr>
<tr>
<td>(3) Shen's</td>
<td>3n + 2</td>
<td>3n + 1</td>
</tr>
<tr>
<td>(4) Non-Stationary</td>
<td>4n + 2</td>
<td>4n</td>
</tr>
</tbody>
</table>

The methods of Projection and Gauss-Seidel have an advantage in requiring fewer calculations at each step than the other two methods. Thus, unless the two acceleration methods can show some advantages in the other areas of comparison, we may as well continue to use methods (1) and (2).

**B. Iteration Steps**

The comparisons are made on the four single-step iterative methods of Section A. These methods are programmed in
PL/1, compiled and stored on disk for faster execution. The execution was on an I.B.M. 360/65 computer which was running under M.V.T. programming. The programs were run with no other user in the system and the Go-step time used as the execution time for each example. The comparisons are on the number of single steps required to reach the desired accuracy and the time of execution for each.

The two-dimensional Projection methods also have the second acceleration technique of Aitken programmed into them through a parameter in the argument list. A special program is also included to illustrate the advantage of applying the suggestions in Chapter V on ordering the columns of A by using the maximum values of the C matrix. The values of C are chosen to include all of the columns of A, ordered large to small and input into a vector for selection by the program.

C. Test Problems

The test problems include linear systems of orders three, four, five, six, eight, and nine. The coefficient matrix, C matrix, b vector and approximate solution vector are given for each problem. The values of C have been rounded to two decimal places for convenience. All of the norms of the residual vectors are less than $5(10^{-5})$, therefore they are left out of the comparison tables. The comparisons are on the C.P.U. time used by each program and the number of single steps required
to reduce the norm to the desired value. The time is recorded in seconds. The number of steps is denoted by $\text{Iter}$. The value of $I_A$ is used to implement the second acceleration of Aitken. When $I_A = 0$ there is no second acceleration. Nine of the ten test problems given show a significant advantage in using Aitken's method.

The five different methods are identified in the comparison tables by the following notation:

1. $\text{N-S}(g)$ is the non-stationary method presented in Chapter VI.
2. Shen is the two-dimensional Projection method using the sequence $g_n$.
3. $\text{N-S}(\beta)$ is a method using the sequence $\beta$ for illustration.
4. Proj is the single-step, one-dimensional Projection method.
5. G-S is the single-step method of Gauss-Seidel.
Test Problem # 1

\[ A = \begin{bmatrix} 3 & 2 & 2 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} \]

\[ C = \begin{bmatrix} 0.00 & 1.82 & 3.46 \\ 1.82 & 0.00 & 2.00 \\ 3.46 & 2.00 & 0.00 \end{bmatrix} \]

\[ x^* = (1.000011, 0.999998, 0.999988) \]

Comparisons for Test Problem # 1

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S(β₁)</td>
<td>0</td>
<td>8</td>
<td>.5</td>
<td>1</td>
<td>8</td>
<td>.5</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>12</td>
<td>.5</td>
<td>1</td>
<td>12</td>
<td>.4</td>
</tr>
<tr>
<td>3. N-S(β)</td>
<td>0</td>
<td>8</td>
<td>.4</td>
<td>1</td>
<td>8</td>
<td>.4</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>84</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. G=S</td>
<td>0</td>
<td>39</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test Problem # 2

\[ n = 3 \]

\[ A = \begin{bmatrix} 4 & -1 & 2 \\ -1 & 2 & 0 \\ 3 & 0 & 5 \end{bmatrix} \]

\[ b = \begin{bmatrix} 13 \\ 4 \\ 26 \end{bmatrix} \]

\[ c = \begin{bmatrix} 1.38 \\ 0.00 \\ 1.03 \\ 3.35 \end{bmatrix} \]

\[ x' = (1.999971, 2.999964, 4.000007) \]

Comparisons for Test Problem # 2

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S(\beta_1)</td>
<td>0</td>
<td>23</td>
<td>.4</td>
<td>1</td>
<td>13</td>
<td>.4</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>56</td>
<td>.5</td>
<td>1</td>
<td>16</td>
<td>.4</td>
</tr>
<tr>
<td>3. N-S(\beta)</td>
<td>0</td>
<td>23</td>
<td>.4</td>
<td>1</td>
<td>8</td>
<td>.5</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>213</td>
<td>.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>42</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test Problem # 3 \( n = 4 \)

\[
A = \begin{bmatrix}
1.00 & 0.96 & 0.84 & 0.64 \\
0.96 & 0.92 & 0.44 & 0.22 \\
0.84 & 0.44 & 1.00 & 0.34 \\
0.64 & 0.22 & 0.34 & 1.00 \\
\end{bmatrix}
\]
\[
b = \begin{bmatrix}
3.44 \\
2.54 \\
2.63 \\
2.21 \\
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0.00 & 10.89 & 8.38 & 2.97 \\
10.89 & 0.00 & 3.79 & 1.81 \\
8.38 & 3.79 & 0.00 & 2.23 \\
2.79 & 1.81 & 2.23 & 0.00 \\
\end{bmatrix}
\]

\[
X' = (1.000038, 1.000009, 0.999991, 0.999935)
\]

Comparisons for Test Problem # 3

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S((g_1))</td>
<td>0</td>
<td>52</td>
<td>.7</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>156</td>
<td>.7</td>
</tr>
<tr>
<td>3. N-S((g))</td>
<td>0</td>
<td>53</td>
<td>.8</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>1788</td>
<td>3.0</td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>==</td>
<td>---</td>
</tr>
</tbody>
</table>
Test Problem # 4  \( n = 4 \)

\[
A = \begin{bmatrix}
5 & 2 & 5 & 6 \\
2 & 10 & 7 & 8 \\
5 & 7 & 10 & 9 \\
6 & 8 & 9 & 10
\end{bmatrix} \quad b = \begin{bmatrix}
23 \\
32 \\
31 \\
33
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0.00 & 2.89 & 9.18 & 10.16 \\
2.89 & 0.00 & 9.14 & 10.60 \\
9.18 & 9.14 & 0.00 & 79.71 \\
10.16 & 10.60 & 79.71 & 0.00
\end{bmatrix}
\]

\[x' = (-8.530625, -5.171392, -1.655999, 14.045749)\]

Comparisons for Test Problem # 4

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S((\beta_1))</td>
<td>0</td>
<td>1326</td>
<td>8.3</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>2616</td>
<td>5.2</td>
</tr>
<tr>
<td>3. N-S((\beta))</td>
<td>0</td>
<td>1347</td>
<td>8.7</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>--</td>
<td>---</td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>--</td>
<td>---</td>
</tr>
</tbody>
</table>
Test Problem # 5 

\[ n = 4 \]

\[ A = \begin{bmatrix}
67 & 13 & 28 & 26 \\
13 & 69 & 50 & 6 \\
28 & 50 & 156 & 78 \\
26 & 6 & 78 & 110
\end{bmatrix} \]

\[ b = \begin{bmatrix}
134 \\
138 \\
312 \\
220
\end{bmatrix} \]

\[ C = \begin{bmatrix}
0.00 & 1.32 & 1.63 & 1.69 \\
1.32 & 0.00 & 2.38 & 1.25 \\
1.63 & 2.38 & 0.00 & 3.92 \\
1.69 & 1.25 & 3.92 & 0.00
\end{bmatrix} \]

\[ X' = (1.000000, 1.000000, 1.000000, 0.999999) \]

Comparisons for Test Problem # 5

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S(p_1)</td>
<td>0</td>
<td>57</td>
<td>.8</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>72</td>
<td>.4</td>
</tr>
<tr>
<td>3. N-S(p)</td>
<td>0</td>
<td>46</td>
<td>.7</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>560</td>
<td>1.1</td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>92</td>
<td>.3</td>
</tr>
</tbody>
</table>
Test Problem # 6 \( n = 4 \)

\[
A = \begin{bmatrix}
1 & 1 & -1 & 1 \\
2 & 4 & -2 & -4 \\
3 & 3 & -3 & -3 \\
4 & 3 & 5 & 6
\end{bmatrix}
\]

\[
b = \begin{bmatrix}
2 \\
0 \\
0 \\
18
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0.00 & 7.00 & 1.03 & 1.04 \\
7.00 & 0.00 & 1.01 & 1.02 \\
1.03 & 1.01 & 0.00 & 8.01 \\
1.04 & 1.02 & 8.01 & 0.00
\end{bmatrix}
\]

\[
x' = (1.000045, 0.999946, 0.9999997, 0.999975)
\]

Comparisons for Test Problem # 6

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S(g)</td>
<td>0</td>
<td>765</td>
<td>5.3</td>
<td>1</td>
<td>17</td>
<td>.7</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>152</td>
<td>.9</td>
<td>1</td>
<td>20</td>
<td>.6</td>
</tr>
<tr>
<td>3. N-S(g)</td>
<td>0</td>
<td>39</td>
<td>.8</td>
<td>1</td>
<td>12</td>
<td>.6</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>1400</td>
<td>2.7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test Problem # 7  
\[ n = 4 \]

\[ A = \begin{bmatrix} 4 & 2 & 1 & 3 \\ 2 & 3 & 1 & 4 \\ 1 & 1 & 2 & 1 \\ 2 & 1 & 1 & 2 \end{bmatrix} \]

\[ b = \begin{bmatrix} 23 \\ 27 \\ 13 \\ 15 \end{bmatrix} \]

\[ C = \begin{bmatrix} 0.00 & 4.36 & 2.33 & 6.00 \\ 4.36 & 0.00 & 2.56 & 50.00 \\ 2.33 & 2.56 & 0.00 & 2.36 \\ 6.00 & 50.00 & 2.36 & 0.00 \end{bmatrix} \]

\[ X' = (1.000005, 2.000056, 2.999978, 3.999939) \]

**Comparisons for Test Problem # 7**

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S((g_1))</td>
<td>0</td>
<td>30</td>
<td>.8</td>
<td>1</td>
<td>22</td>
<td>.6</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>576</td>
<td>1.4</td>
<td>1</td>
<td>576</td>
<td>1.6</td>
</tr>
<tr>
<td>3. N-S((g))</td>
<td>0</td>
<td>38</td>
<td>.7</td>
<td>1</td>
<td>19</td>
<td>.?</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>4100</td>
<td>6.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>224</td>
<td>.6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test Problem # 8  

\[ n = 5 \]

\[
A = \begin{bmatrix}
21 & 1 & 2 & 1 & -1 \\
3 & 21 & 4 & 1 & 2 \\
5 & 6 & 18 & 0 & 1 \\
1 & 2 & 3 & 18 & 1 \\
2 & 0 & 1 & 3 & 19
\end{bmatrix}
\]

\[
b = \begin{bmatrix}
-6 \\
-10 \\
2 \\
12 \\
-1
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0.00 & 1.06 & 1.15 & 1.01 & 1.00 \\
1.06 & 0.00 & 1.31 & 1.02 & 1.01 \\
1.15 & 1.31 & 0.00 & 1.03 & 1.02 \\
1.01 & 1.02 & 1.03 & 0.00 & 1.05 \\
1.00 & 1.01 & 1.02 & 1.05 & 0.00
\end{bmatrix}
\]

\[
X^* = (-0.337394, -0.520407, 0.386406, 0.686939, -0.145917)
\]

Comparisons for Test Problem # 8

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S(β₁)</td>
<td>0</td>
<td>18</td>
<td>.7</td>
<td>1</td>
<td>18</td>
<td>.7</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>30</td>
<td>.5</td>
<td>1</td>
<td>30</td>
<td>.5</td>
</tr>
<tr>
<td>3. N-S(β)</td>
<td>0</td>
<td>15</td>
<td>.5</td>
<td>1</td>
<td>15</td>
<td>.7</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>45</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>25</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test Problem # 9 \( n = 8 \)

\[
A = \begin{bmatrix}
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\
\end{bmatrix}
\]

\[
b = \begin{bmatrix}
1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0.00 & 2.14 & 1.03 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \\
2.14 & 0.00 & 1.80 & 1.03 & 1.00 & 1.00 & 1.00 & 1.00 \\
1.03 & 1.80 & 0.00 & 1.80 & 1.03 & 1.00 & 1.00 & 1.00 \\
1.00 & 1.03 & 1.80 & 0.00 & 1.80 & 1.03 & 1.00 & 1.00 \\
1.00 & 1.00 & 1.03 & 1.80 & 0.00 & 1.80 & 1.03 & 1.00 \\
1.00 & 1.00 & 1.00 & 1.03 & 1.80 & 0.00 & 2.14 & 0.00 \\
1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 2.14 & 0.00 \\
\end{bmatrix}
\]

\[
X' = (0.777946, -0.555867, 0.333752, -0.111582, -0.110625, 0.332909, -0.555239, 0.777610)
\]

Comparisons for Test Problem # 9

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S((g_1))</td>
<td>0</td>
<td>--</td>
<td>--</td>
<td>1.</td>
<td>1</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>2376</td>
<td>8.7</td>
<td>2.</td>
<td>1</td>
<td>280</td>
<td>1.6</td>
</tr>
<tr>
<td>3. N-S((g))</td>
<td>0</td>
<td>1875</td>
<td>10.1</td>
<td>3.</td>
<td>1</td>
<td>895</td>
<td>5.5</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>11088</td>
<td>21.8</td>
<td>4.</td>
<td>1</td>
<td>895</td>
<td>5.5</td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>456</td>
<td>.9</td>
<td>5.</td>
<td>1</td>
<td>895</td>
<td>5.5</td>
</tr>
</tbody>
</table>
Test Problem # 10 \hspace{1cm} n = 9

\[
\begin{bmatrix}
4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\
-1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\
0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.5 \\
1.0 \\
0.5 \\
0.0 \\
0.0 \\
0.0 \\
0.0 \\
\end{bmatrix}
\]

\[
A = \begin{bmatrix}
0.00 & 1.23 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \\
1.23 & 0.00 & 1.23 & 1.01 & 1.05 & 1.01 & 1.00 & 1.00 \\
1.00 & 1.23 & 0.00 & 1.00 & 1.23 & 1.00 & 1.00 & 1.00 \\
1.23 & 1.01 & 1.00 & 0.00 & 1.22 & 1.00 & 1.23 & 1.01 \\
1.00 & 1.05 & 1.00 & 1.22 & 0.00 & 1.22 & 1.01 & 1.22 \\
1.00 & 1.01 & 1.23 & 1.00 & 1.22 & 0.00 & 1.00 & 1.01 \\
1.00 & 1.00 & 1.00 & 1.23 & 1.01 & 1.00 & 0.00 & 1.23 \\
1.00 & 1.00 & 1.00 & 1.01 & 1.22 & 1.01 & 1.23 & 0.00 \\
1.00 & 1.00 & 1.00 & 1.01 & 1.23 & 1.00 & 1.23 & 0.00 \\
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0.00 & 1.23 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \\
1.23 & 0.00 & 1.23 & 1.01 & 1.05 & 1.01 & 1.00 & 1.00 \\
1.00 & 1.23 & 0.00 & 1.00 & 1.23 & 1.00 & 1.00 & 1.00 \\
1.23 & 1.01 & 1.00 & 0.00 & 1.22 & 1.00 & 1.23 & 1.01 \\
1.00 & 1.05 & 1.00 & 1.22 & 0.00 & 1.22 & 1.01 & 1.22 \\
1.00 & 1.01 & 1.23 & 1.00 & 1.22 & 0.00 & 1.00 & 1.01 \\
1.00 & 1.00 & 1.00 & 1.23 & 1.01 & 1.00 & 0.00 & 1.23 \\
1.00 & 1.00 & 1.00 & 1.01 & 1.22 & 1.01 & 1.23 & 0.00 \\
1.00 & 1.00 & 1.00 & 1.01 & 1.23 & 1.00 & 1.23 & 0.00 \\
\end{bmatrix}
\]

\[
x' = (0.247033, 0.373517, 0.247026, 0.114595, 0.166680, 0.114588, 0.044653, 0.063998, 0.044648)
\]

Comparisons for Test Problem # 10

<table>
<thead>
<tr>
<th>Method</th>
<th>I_A</th>
<th>Iter</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. N-S(\beta_1)</td>
<td>0</td>
<td>324</td>
<td>3.3</td>
</tr>
<tr>
<td>2. Shen</td>
<td>0</td>
<td>198</td>
<td>1.3</td>
</tr>
<tr>
<td>3. N-S(\beta)</td>
<td>0</td>
<td>181</td>
<td>1.7</td>
</tr>
<tr>
<td>4. Proj.</td>
<td>0</td>
<td>405</td>
<td>1.3</td>
</tr>
<tr>
<td>5. G-S</td>
<td>0</td>
<td>108</td>
<td>.6</td>
</tr>
</tbody>
</table>
Test problems # 1 and # 2 are arranged so that the maximum value of C is $C_{3}^{1}$. The two non-stationary methods yield the same results since $\beta_{1} = \beta$. The method using $\beta_{n}$ includes the value $C_{3}^{1}$ but it is last in the cycle whereas the sequences $\beta_{1}$ and $\beta$ use $C_{1}^{3}$ first followed by $C_{2}^{3}$.

Test problem # 3 illustrates an example in which the maximum values of C fall on row one, thus both sequences $\beta_{1}$ and $\beta$ are the same while $\beta_{n}$ is different and does not use the value of $C_{3}^{1} = 8.38$. The iterative methods using $\beta_{1}$ and $\beta$ converge in 52 and 53 steps respectively, while the method using $\beta_{n}$ requires 156 steps to converge. The second acceleration method causes all of the sequences to yield about the same results.

Test problem # 4 has the largest values appearing in row four so that $\beta_{1} = \beta$ again and the methods with these sequences converge in about 1300 steps. The method using $\beta_{n}$ requires about 2600 steps but runs in less time than the other two. The Projection method and Gauss-Seidel do not converge in the allotted step count. The second acceleration method is much better for $\beta_{n}$ in this case. This problem is the same as the matrix of figure 1 except with $a_{1,2} = a_{2,1} = 2$ to remove the large value of $C_{2}^{1}$. The largest value of C is now 79.7 and indicates that the system still tends to be ill-conditioned; however, the acceleration methods still converge.

Test problem # 6 is an example of a system in which the columns of A can be moved so that the maximum values of C will
be on the superdiagonal or the diagonal above the superdiagonal. When they are above the superdiagonal the method with $\beta_n$ requires 1276 steps, the method with $\beta_1$ requires 765 steps and the method with the sequence $\beta$ converges in 39 steps. This example also shows that the maximum values of C cannot always be arranged to be on one row. The sequence $\beta$ needs only two values of C to form a convergent method and cover A.

Test problem #7 is another example of the maximum values of C being above the superdiagonal. If they were on the superdiagonal then the method with $\beta_n$ converges in 44 steps.

Test problem #8 is an example of a linear system which is strictly diagonally dominant. The number of steps range from 15 to 45 and the time varies from .4 seconds to .7 seconds. The maximum value of C is 1.31.

Test problem #9 is an example with the largest values of C on the superdiagonal but not arranged in order of size. When the second acceleration of Aitken is not used the number of steps is reduced from 2376 to 1875 by arranging the columns of A so that the values of C are ordered large to small. This example also illustrates the results of not using the largest values of C in the iterative method. The non-stationary method uses row two which has only two large values out of seven. The other five values are one since the columns are orthogonal to column two.
Test problems # 9 and # 10 are diagonally dominant therefore the elements of C are small. In these cases the method of Gauss-Seidel converges in fewer steps and in less time than the Projection methods.
IX. CONCLUSIONS AND FUTURE RESEARCH

A. Conclusions

The work presented here shows that the one-dimensional Projection method can be accelerated by a two-dimensional method. The rate of convergence of a given algorithm on a linear system is a function of many parameters, some of which are: The magnitude of $r_0$, the length of the column vectors of A, the angles between the columns of A, and the order in which the columns of A are used in the algorithm. The angles between the columns of A are used to determine the C matrix used in this paper. When the elements of C are selected and ordered, the columns of A are automatically ordered for the algorithm. An algorithm based on the matrix C is more effective when some of the elements are large compared to the others; however, if one or more of the values are too large, the system is ill-conditioned and cannot be solved by the Projection method.

When the coefficient matrix is diagonally dominant, the method of Gauss-Seidel converges in fewer steps and less time than the Projection methods.

To choose an optimal order of the columns of the coefficient matrix, one must first calculate the matrix C, then select a cyclic sequence of the elements of C which includes all of the columns of A and the maximum values of C.
B. Future Research

The lemmas in Chapter V indicate the manner of obtaining the maximum reduction at each step. It is the feeling of this author that this will also yield the maximum reduction at each cycle. Thus, the first suggestion for future research is to prove a theorem relating the maximum values of C to the maximum reduction in the magnitude of the residual vector per cycle.

The method suggested here for choosing an algorithm to solve a linear system seems especially suited to an interactive system. The C matrix could be displayed and the user could choose an appropriate algorithm from a given set or have the option of implementing his own. Further work needs to be done on implementing this idea and the algorithms to use.

There is still the possibility of having an ill-conditioned system unless we are willing to invest a large amount of effort before beginning the solution to the linear system. Are there other ways that the coefficient matrix could be used to determine this condition without so much effort?

When the elements of C are small, the best order of the columns of A is not obvious—it may be a function of the length of the vectors of A, the angles between the vectors of A and the residual vector, or it might be possible to derive similar results from the row vectors of A. An additional area of future work is to determine the best order in this case.
X. BIBLIOGRAPHY


XI. ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation for the encouragement and support of Dr. Roy F. Keller during my graduate program at Iowa State University.

I am also grateful to U.M.R., Rolla, Mo. for granting the time off from my regular duties to pursue the course of study in Computer Science and to I.S.U., Ames, Iowa, for the financial support during my graduate program.

I am especially grateful for the support, encouragement, and patience of my family during the course of study and the development of the thesis. A special word of thanks is due my wife, Peggy, for taking the time out of her busy schedule to type the manuscript.
XII. PROGRAM IMPLEMENTATION

The non-stationary iterative method described in Chapter VI has been implemented with a PL/1 program which was run on an I.B.M. 360/65 system. The criterion for convergence is the square root of the magnitude of the residual vector less than EPSI. The test for convergence is made after each step of the iteration instead of at the end of a cycle as with most of the iterative methods. A cycle in this program is defined as N-1 single steps. The following list defines most of the symbols used in the program. Those symbols which are followed by (*) are parameters in the call to the program procedure.

A: An N by N coefficient matrix.
B: An N by (N+1) augmented matrix for input.
C: An N by N symmetric matrix derived from A;
   \( C_j^i = \frac{1}{1 - \cos^2 \theta_{ij}}, \quad i \neq j, \)
   \( i = 1, 2, \ldots, N-1, \quad j = 2, 3, \ldots, N. \)
D: An N-tuple vector containing the scalars \( (a_i, a_i), \)
   \( i = 1, 2, \ldots, N. \)
R: The residual vector at the k-th step; initially \( R = B(*,N+1). \)
X1: The solution vector at the k-th step; initially \( X1 = 0. \)
XX_1: Vector X1 at the previous cycle, \( p-1, \) retained for
the second stage acceleration when I_A > 0.

XX_2: Vector X1 stored at the end of cycle p-2.

N: Order of the matrix A. (*)

I: Refers to the fixed column of A, a_i. (*)

J: Refers to the column of A used with a_i at the k-th step.

DCJ: An N-tuple vector containing the values C_j^i / D_j,
     j = 1, 2, ... , N.

AIJ: An N-tuple vector containing A_j^i, j = 1, 2, ... , N.

AIJI: An N-tuple vector containing AIJ / A_i.

ANORMR: Square root of (R^k, R^k).

EPSI: Test value for accuracy of solution against ANORMR (*)

I_A: This value is set to zero for no second acceleration. (*)

ICT: Maximum number of iteration steps before failure to reach desired accuracy. (*)

ICT_2: An integer for controlling the amount of output desired. (*)

ICOUNT: Number of single steps required for convergence.

This program is written as a subroutine procedure with variable dimensions on all arrays, thus using only the necessary storage and allowing for any size of input system (up to the limit of the storage of a given system). The coefficient matrix, augmented with the vector b, is read in under a format
of \((N+1)P(10,4)\) within the procedure. The initial calculations are made as described in Chapter VI, section C, then the iteration begins with step 1 and proceeds to step 2. The iteration procedure remains in step 1 until \((r^k, \bar{r}^k)^{\frac{1}{2}}\) has been reduced a sufficient amount or until the ratio of the present norm value to the last norm value exceeds some set amount. The value 0.94 seems to be a good cut-off point. If the ratio exceeds this value then the changes in the components of \(x_1\) are small and thus will tend to remain small which causes very slow convergence. Since step 1 chooses the maximum inner product the change at any step would not be zero because of \((r^k, a_j)\) thus the ratio of successive norms should always be less than one.

Step 2 has an inner loop on \(j\), whose values come from the vector \(VJS\), and an outer loop controlled by the maximum number of iterations as given by \(ICT\). Each completion of the inner loop is called a cycle and for \(I_A > 0\) a test is made to see if the second acceleration method of Aitken(1) will apply. The test is made on the 1-th component of three successive approximations to the vector \(x_1\). The first test is that the three values are monotonic in nature, either increasing or decreasing. If this is true then after the third cycle has been completed a test is made on the difference in two successive values of the ratio of \((x_1^k - x_1^{k-1})/(x_1^{k-1} - x_1^{k-2})\). If this difference is small then equation 7.11 is used to
obtain a new approximation for $X_l$. The ratio of the terms involving $X_l^k$ mentioned above begins to approximate the largest eigenvalue of the iteration matrix as $k$ gets large when the iteration matrix has a simple, largest eigenvalue.

Step 2 terminates when the desired accuracy is reached for the residual vector or when the iteration count has been exceeded.
XIII. APPENDIX

AUTO: PROC OPTIONS(MAIN);
DCL LN INIT(I); ON ENDFILE(SYSIN) GO TO FINISH;
DO WHILE('1'B);
GET EDIT(N,ICT,ICT_2,1_A,EPSI)(COL(1),4 F(8),F(12,7)) ;
IF LN > 1 THEN PUT PAGE; ELSE LN=2;
IF N > 0 THEN CALL AUTO_3(N,ICT,ICT_2,1_A,EPSI ) ;
END;
/*
A NON-STATIONARY METHOD FOR ACCELERATING THE PROJECTION
METHOD FOR SOLVING LINEAR SYSTEMS OF EQUATIONS */
AUTO_3: PROC(N,ICT,ICT_2,1_A,EPSI );
DCL A(N,N),B(N,N+1),C(N,N),CC(N,2),RO(N),
X1(N),R(N),DCJ(N),AIJ(N),AIJ(N),VJS(N-1),CC1(2),
XX_1(N),XX_2(N),BJ BIT(1) INIT('0'B),LINE CHAR(80)
ON ZERO DIVIDE GO TO ERR_4; /* THIS MAY HAPPEN IN LINE 53 */
GET EDIT(LINE)(COL(1),A )
PUT EDIT(N,ICT,EPSI,LINE)(COL(5),2 F(8),F(10,6),A) ;
GET EDIT((B(1,*))DO 1=1 TO N)(COL(1),(N+1)F(10,4)) ;
PUT EDIT('ORIGINAL MATRIX; I_A = ',I_A)(COL(10),B,(N)(COL(2),(N+1)F(10,4))) ;
/* ** INITIAL CALCULATIONS ; C, I, J, ORDER C(I,*) */
ANORM = SUM(R*R) ; M = N - 1 ; ANORM1 = 1.0/ ANORM ;
/* ** CALCULATION OF D VECTOR AND CHECK TO SEE IF
THE INITIAL b VECTOR IS PARALLEL TO SOME A(*,J) ** */
DO K = 1 TO N ; AC,D(K) = SUM(A(*,K)** 2 I ;
IF ANORM1 * ABS(SUM(A(*,K)*R))/AC = 1.0 THEN DO;
BJ = '0'B ; IK = K ; END ;
END ;
IF BJ = '0'B THEN GO TO A_Z1 ;
BJ = '0'B ;
DO K = 1 TO N ; IF ABS(A(K,IK)) = 0.0 THEN GO TO A_Z00;
X1(IK) = R(IK) / A(K,IK) ; GO TO NEXT;
A_Z00 :) END ;
A_Z1: LINE_ERR = 53 ; C(N,N) = 0 ;
/* * LOOP1 CALCULATES THE MATRIX C AND FINDS MAX C(I,J) */
LOOP1: DO I = 1 TO M ;
DO J = 1 TO N;
APC = SUM(A(*,I)*A(*,J))**2 / (D(I)*D(J)) ;
APC,C(I,J),C(J,I) = 1.00 / (1.00 - APC) ;
IF APC > AM THEN DO; IM=I; JM=J; AM=APC;
END LOOP1; /* CALCULATION OF C MATRIX */

/* **** END OF LOOP1 *************** */

/* IF MOD(ICT,2) = 1 THEN */
/* * MATRIX C IS NOT PRINTED IF MOD RESULTS IN A 0 */
PUT EDIT(' C MATRIX; 1/(1 - (COS Y)**2') (COL(20), A) (C)((N) (COL(2), (N) F(14,5)))
/* LENGTH OF A(*,K)', D(COL(1), A, (N+1) F(14,5)); */
/* **** FIND MAX ROW SUM; C(I,*); OR C(J,*); : **** */
SUMI = SUM(C(IM,*)); SUMJ=SUM(C(JM,*));
IF SUMI < SUMJ THEN DO; I=JM; JN,J=IM; END;
ELSE DO; I=IM; JN,J=JM; END;
IF C(I,J) > 100.0 THEN PUT EDIT('C(I,J) IS GREATER THAN 100.0, HOWEVER, ITERATION WILL CONTINUE') (COL(5), A);

/* FILL MATRIX CC WITH C(I,*) */
CALCULATE VECTOR AIJ = (A(*,I) , A(*,K))
CALCULATE VECTOR AIJI = AIJ / D(I)
CALCULATE VECTOR DCJ = C(I,*) / D
ORDER CC(*,I) FROM LARGEST TO SMALLEST.
STORE CC(1,*); IN VECTOR VJS FOR LATER USE

/* A_Z7: DO K = 1 TO N;
CC(K,1) = K;
AIJ(K) = SUM(A(*,I)*A(*,K));
CC(K,2) ; DCJ(K) = C(I,K);
END A_Z7;
AIJI = AIJ/D(I);
DCJ = DCJ/D;
IF I = N THEN CC(I,*) = CC(N,*);
*/

/* ORDER C(I,*) FOR USE IN STEP_2; LARGEST TO SMALLEST */
A_Z8: DO K = 1 TO N-2;
DO K1 = K+1 TO M;
IF CC(K,2) < CC(K1,2) THEN DO;CCI = CC(K,*);
CC(K,*); = CC(K1,*);
CCI(K1,*); = CCI;
END;
VJS(K) = CC(K,1);
END A_Z8;
VJS(M) = CC(M,1);
TEST_1 = CC(M,2);
ANORM = 2.0 * ANORM;
IF ABS(SUM(A(*,J)*R)) = 0.0 THEN GO TO A_Z2;

/* STEP_1 OF ITERATION PROCESS: J CHOSEN BY MAX ANGLE */
/* STEP_1: REDUCE BY METHOD 1 UNTIL (R,R) < MIN C(I,*); */
ST: J = JN;
ALP = SUM(A(*,J)*R);
CI = SUM(A(*,I)*R); IF ABS(CI) < .000005 THEN DO ;
DXI = ALP * DCJ(J); /* EQUATIONS 4.13 */
DXJ = ALP * AIJ(J); END;
ELSE DO;
AM = D(J);
ALP = ALP / AM; /* EQUATIONS 4.12 */
DXI = ( CI - ALP * AIJ(J)) * C(I,J) / D(I);
DXJ = (ALP - DXI * AIJ(J) / AM);
END;
R = R - DXI * A(*,I) - DXJ * A(*,J);
ANGRM1 = SUM(R*R);
ANORMR = SQRT(ANGRM1);
X1(I) = X1(I)+DXI;
X1(J) = X1(J)+DXJ;
ICOUNT = ICREMENT + 1;
RCJR = ANORMR / ANORM;
IF RCJR > .94 THEN GO TO ST3;
ANORM = ANORMR;
IF ANORM <= TEST_1 THEN GO TO ST3; /* TEST ON (R,R) */
IF ICREMENT <= ICT_2 THEN
PUT EDIT(I,J,ICOUNT,ANORMR,ANORM1, RCJR = RCJR)
(COL(10),(3)F(5),(2)F(12,5),(2)A,F(12,5))
(X1,R ) (COL(1), (N)F(14,9)) ;
/* IN STEP_1 CHOOSE J SO THAT ANGLE IS MAXIMIZED WITH R */
AR=0;
DO K = 1 TO N;
IF K=1 | K=J THEN GO TO A_Z6;
AC = ABS(SUM(A(*,K)*R)) / D(K);
IF AC > AR THEN DO; JN=K; AR=AC; END;
A_Z6: END;
A_Z3: IF ICREMENT > ICT THEN GO TO ER1; GO TO ST;
/* ** STEP_2 OF ITERATION PROCESS; CYCLE ON J FROM VJS */
ST3: PUT EDIT(I,J,ICOUNT,ANORMR,ANORM1, RCJR = RCJR)
(COL(10),(3)F(5),(2)F(12,5),(2)A,F(12,5))
(X1,R ) (COL(1), (N)F(14,9)) ;
/* ** STEP_2 OF ITERATION PROCESS; CYCLE ON J FROM VJS */
IF J = VJS(1) THEN BJ=1'B;
LINE_ERK = 197;
DO WHILE('1'B);
A_Z4: DO JK2 = 1 TO M;
IF BJ = '1'B THEN DO; BJ='0'B; GO TO A_Z20; END;
J = VJS(JK2);
ALP = SUM(A(*,J)*R);
CI = SUM(A(*,I)*R);
IF ABS(CI) < .0000005 THEN DO;
DXJ = ALP * DCJ(J); /* EQUATIONS 4.13 */
DXI = -DXJ * AIJ(J); END;
ELSE DO:
AM = D(J);
ALP = ALP / AM; /* EQUATIONS 4.12 */
DXI = ( CI - ALP * AIJ(J)) * C(I,J) / D(I);
DXJ = (ALP - DXI * AIJ(J) / AM);
END;
R = R - DXI*A(*,I) - DXJ * A(*,J) ;
ANORM1 = SUM(R*R);
ANORMR = SQRT(ANORM1);
X1(I) = X1(I)+DXI ; X1(J) = X1(J) + DXJ ;
ICOUNT = ICOUNT + 1; /* RECORD OF ITERATION STEPS */
IF ANORMR < EPS1 THEN GO TO NEXT;
A_220: END ; /* END OF INNER LOOP ON J */
RCJR = ANORMR / ANORM;
IF I_A = 0 THEN GO TO A_20; /* NO SECOND ACCELERATION*/
A_30: END;
IF IRZ < 2 THEN DO; IRZ = IRZ + 1 ; GO TO A_25 ; END;
DELX_1 = X1(I) - XX_1(I); DELX_2 = XX_1(I) - XX_2(I);
IF SIGN(DELX_1)* SIGN(DELX_2) < 0 THEN GO TO A_25;
IF ABS(DELX_2) < .00000001 THEN GO TO A_20;
RX_1 = DELX_1 / DELX_2;
IF IRZ < 3 THEN DO; IRZ = IRZ + 1 ; GO TO A_25 ; END;
DO K = 1 TO N;
X_1 = XX_1(K);
DEL_2 = X1(K) - 2.0 * X_1 + XX_2(K);
IF ABS(DEL_2) < .00000001 THEN GO TO A_29;
X1(K) = X1(K) -(X1(K)-X_1)*2 / DEL_2;
A_29: END;
DO K = 1 TO N; R(K) = R(Q(K) - SUM(A(K,*)*X1} ; END;
ANQRMR= SORT(SUM{R«R)); IF ANORMR < EPS1 THEN GO TO NEXT;
A_5: XX_2 = XX_1; XX_1 = XI; RX_2 = RX_1 ;
A_20: IF I_A = 0 THEN GO TO A_25; END; /* END OF OUTER LOOP ON ITERATION COUNT, ICOUNT*/
A_30: END;
ERR_4: PUT EDIT('ITERATION SEQUENCE STOPPED BECAUSE OF ZERO-DIVIDE IN LINE',LINE_ERR,' I = ',I,' J = ',J) ;
/* IF LINE_ERR = 53 THEN THE SYSTEM IS SINGULAR */
GO TO A_230;
NEXT: PUT EDIT('ITERATION CONVERGED')(COL(10),A) ;
A_230: END AUTO_3 ;
FINISH : END AUTO ;