A Generalized Projection method for systems of nonlinear equations

Alexander MacEachern
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Mathematics Commons

Recommended Citation
https://lib.dr.iastate.edu/rtd/4896
MAC EACHERN, Alexander, 1942-
A GENERALIZED PROJECTION METHOD FOR SYSTEMS OF NONLINEAR EQUATIONS.

Iowa State University, Ph.D., 1971
Mathematics

University Microfilms, A XEROX Company, Ann Arbor, Michigan

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED
A Generalized Projection method for systems of nonlinear equations

by

Alexander Mac Eachern

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.

Head of Major Department

Signature was redacted for privacy.

Dean of Graduate College

Iowa State University
Ames, Iowa

1971
A Generalized Projection method for systems
of nonlinear equations

by

Alexander Mac Eachern

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

Approved:

Signature was redacted for privacy.
In Charge of Major Work

Signature was redacted for privacy.
Head of Major Department

Signature was redacted for privacy.
Dean of Graduate College

Iowa State University
Ames, Iowa

1971
A Generalized Projection method for systems
of nonlinear equations

Alexander Mac Eachern

Under the supervision of R. F. Keller
From the Department of Computer Science
Iowa State University of Science and Technology

In this work a Projection method for solving systems of linear
equations is extended to cover the nonlinear case. Projection methods
fall into the broad class of minimization methods, which includes the
various types of gradient techniques for solving systems of equations.
The general idea in minimization methods is to solve the system of
equations \( Fx = 0 \) by reducing the norm (usually the Euclidean norm) of
the residue vector to zero, or, sufficiently close to zero. The tech­
niques are iterative in nature and the approximation vector at the

\((k+1)^{st}\) step is generally expressed in terms of the \(k^{th}\) approximation
vector by the relation \( x^{k+1} = x^k + \alpha_k p_k \), where \( \alpha_k \) is a scalar value and
\( p_k \) is a vector. In the method presented in this study the \( p_k \)'s used are
the columns of the \( nxn \) identity matrix when an \( n^{th} \) order system is being
solved. The \( \alpha_k \)'s are determined at each step so as to reduce the
Euclidean norm of the residue vector, \( R \), a maximum amount. The method is
a single step method since the choice of the \( p_k \)'s allows only one compo­
nent of the approximation vector to change at any step. Under the
current implementation of the method the \( p_k \)'s are chosen in a definite
order - the first component of the approximation vector is altered, then
the second, third, etc., down to the \( n^{th} \) component, for a system of
order n. After the n\textsuperscript{th} component is changed the cycle is repeated starting at the first component again. The process continues until the convergence conditions are satisfied. The stepsize, $\alpha_k$, is determined from an expression which involves inner products of the residue vector and columns of the Jacobian matrix of $F_x$.

A proof of the convergence of the method is presented using basic theorems from functional analysis on continuous mappings. Convergence of the method requires the continuity of the mapping $F$ and the non-singularity of the Jacobian matrix of $F$ on the set of approximation vectors generated by the method.

The method developed in this work has been implemented on an IBM 360, Model 65 using FORTRAN IV. A listing of the program and examples of the subroutines required are given in the Appendix. A set of examples, mainly from journal publications, is given in a set of tables in which computer CPU time and number of cycles are used as comparison norms with two other methods for solving systems of nonlinear equations.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. THE GENERALIZED PROJECTION METHOD</td>
<td>10</td>
</tr>
<tr>
<td>A. Development of the Method</td>
<td>10</td>
</tr>
<tr>
<td>B. The Generalized Projection Method</td>
<td>19</td>
</tr>
<tr>
<td>C. Geometric Description of the Generalized Projection Method</td>
<td>20</td>
</tr>
<tr>
<td>D. Proof of Convergence</td>
<td>23</td>
</tr>
<tr>
<td>III. COMPARISON WITH OTHER METHODS</td>
<td>28</td>
</tr>
<tr>
<td>IV. SUMMARY AND CONCLUSION</td>
<td>49</td>
</tr>
<tr>
<td>A. Summary</td>
<td>49</td>
</tr>
<tr>
<td>B. Future Research</td>
<td>50</td>
</tr>
<tr>
<td>C. Conclusion</td>
<td>52</td>
</tr>
<tr>
<td>V. BIBLIOGRAPHY</td>
<td>53</td>
</tr>
<tr>
<td>VI. ACKNOWLEDGEMENTS</td>
<td>56</td>
</tr>
<tr>
<td>VII. APPENDIX</td>
<td>57</td>
</tr>
<tr>
<td>A. Description of the Implemented Program</td>
<td>57</td>
</tr>
<tr>
<td>B. Listing of Implemented Program with Sample Subroutines</td>
<td>60</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

In the Eighteenth and Nineteenth centuries mathematicians were quite successful in constructing many useful theories for physical phenomena. Generally these theories were constructed with certain restrictions. It was found that by neglecting certain elements which were considered of little importance many of the laws of nature could be expressed linearly.

"Mathematicians, however, have an unfailing urge to stray from trodden paths into the unpaved borderlands, and to step from the linear is to plunge into the domain of the nonlinear. This is a difficult terrain, in which the going has been found rough and slow. But it is a domain in which the demand for a clearing of paths is rising. As in all virgin mathematical territory, so here, the challenge to master difficulties for the glory of the human spirit manifests itself. In this case, however, it is firmly backed up also by more mundane considerations. For technology in its trends toward both refinement and magnification of scope and complexity has been pointing with increasing insistence to the fact that in the formulation of natural laws modern requirements of precision forbid the suppression of nonlinear elements - that suitable formulations are, almost without exception, nonlinear."⁴

One such area where it is common to linearize the problem is in the solution of systems of nonlinear algebraic equations. For such problems the common approach has been to linearize the nonlinear system and solve the linear system as an approximation to the original system. Such an

---

approach commonly requires the solution of a linear system of equations for each solution approximation to the nonlinear case. This technique, although useful, is time consuming, especially for large systems. The Newton-Raphson method uses this approach and example 10, chapter III, gives a good indication of the time involved.

In the past few years there have been numerous articles published which present new methods for solving systems of nonlinear equations or review the known methods from a more theoretical approach. A glance at the bibliography easily confirms this fact. In this work another method shall be presented. This method will fall into the "minimization" class, to be mentioned later, since it is an extension of a minimization method for solving systems of linear equations.

The general problem that is considered is to solve a system of nonlinear equations:

\[ \mathbf{F}(\mathbf{x}) = \mathbf{0} \]  

\[ \begin{align*}
  f_1(x_1, x_2, \ldots, x_n) \\
  f_2(x_1, x_2, \ldots, x_n) \\
  \vdots \\
  f_n(x_1, x_2, \ldots, x_n)
\end{align*} \]  

To date, there is no one method which one can use to solve a large spectrum of problems. Many methods have been developed which may work on certain systems and fail on others. It is hoped that the future will bring more general algorithms into existence and the purpose of this paper is to attempt to make it the very near future. The problem given in (I-1) is a very practical one. Such systems arise from many sources.
In a SIGNUM newsletter\(^1\) Rheinboldt presented the following applications where such systems arise:

"1. Discretized Nonlinear Differential Equations.
   Examples: Nonlinear two point boundary value problems.
   Boundary value problems for nonlinear partial differential equations.
   Typical systems: large \( n \).
   Frequently of the form \( Ax = Gx \) with large sparse matrix \( A \) and nonlinear \( G \).

2. Shooting Method.
   Examples: Nonlinear two point boundary value problems.
   Control problems.
   Typical systems: The mapping \( F \) is not explicitly given but is only known via some numerical algorithm.

3. Discretized Integral Equations.
   Typical systems: Medium-sized and generally "full", that is, each component function \( f \) depends on all \( x \).

4. Optimizations.
   Examples: Nonlinear least squares approximation.
   Discretized variational problems.
   Typical systems: Generally small or at best medium-sized, but always full."

In the same paper the following classifications of methods used to solve nonlinear systems are given:

"A. Generalized One-dimensional Methods.

Typical methods are:

1) Successive approximations \( x^{k+1} = Gx^k, k = 0,1,\ldots \), for solving fixed point equations \( x = Gx \).

2) Chord method: \( x^{k+1} = x^k - A^{-1}Fx^k, k = 0,1,\ldots \), \( A \) a constant matrix.

3) Newton's method: \( x^{k+1} = x^k - (F'(x^k))^{-1}Fx^k, F'(x) \) the Frechet derivative of \( F \).

4) Secant methods.

a) Two point secant methods: Discretize each derivative in \( F'(x) \) separately. There are many variations depending on the choice of discretization.

b) General Secant method: Replace the \( n \) tangent hyperplanes defined by the derivative at \( x^k \) by \( n \) interpolating hyperplanes. Now \( n+1 \) points are needed to compute \( x^{k+1} \); and again there are many variations available depending on the choice of the points and their repetition.

B. Generalized Linear Methods.

Generalized one dimensional methods require at each step the solution of the linear system

\[
A(x^k)(x-x^k) + Fx^k = 0
\]

If \( n \) is large, linear iterative methods have to be used to accomplish this. If the secondary iteration is stopped after
a few steps and the result taken as \( x^{k+1} \), a new method results.

Typical methods of this type are

- Newton - Gauss - Seidel
- Newton - SOR
- Secant - SOR
- Secant ADI, etc.

Another type of generalized linear method is the nonlinear Gauss - Seidel iteration

\[
\begin{align*}
    f_i(x_{i+1}^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^{k}, x_{i+1}^k, \ldots, x_n^k) &= 0 \quad i = 1, \ldots, n
\end{align*}
\]

where the \( i^{\text{th}} \) equation is solved for \( x_i \) and the result taken as \( x_i^{k+1} \).

Similarly nonlinear Jacobi or nonlinear SOR methods are defined. An appropriate secondary iteration can be used to solve each one dimensional component equation. If this secondary iteration is stopped after a few steps, methods are obtained which may be called

- Gauss - Seidel - Newton
- Jacobi - Secant, etc.

C. Minimization Methods.

Problem: Minimize \( g: D \subset \mathbb{R}^n \rightarrow \mathbb{R}^1 \)

leads to the system \( Fx = g'(x)^T = 0 \)

Conversely \( Fx = 0 \) leads, for example, to the problem of minimizing \( g(x) = (Fx)^T Fx \)

Methods used are of the general form
\[ x^{k+1} = x^k - \alpha_k p^k \]

\( \alpha_k \) the steplength, \( p^k \) is a direction vector.

Typical methods are

a) Paraboloid methods: Replace \( g \) at \( x^k \) by a suitable quadratic functional which can be minimized explicitly. This includes, for instance, Newton's method and for the special case of \( g(x) = (Fx)^T Fx \) the Gauss-Newton method.

b) Relaxation methods: Choose \( p^k \) successively or by some rule from among a finite set of given direction vectors.

c) Steepest Descent methods: Choose \( p^k = A_k^{-1} g(x^k)^T \) where \( A_k \) is a positive definite matrix defining the metric of the space at the \( k \)th step.

d) Gradient method: Choose \( p^k = g'(x^k)^T \).

e) Generalized Conjugate Direction methods: methods which in the quadratic case reduce to a conjugate direction method.

In each case the steplength can be chosen in various ways.

Typical steplength algorithms are:

a) Exact minimization: Choose \( \alpha_k \) such that \( g(x^k - \alpha_k p^k) \) is a minimum.

b) Generalized Curry steps: Let \( \alpha_k \) be the solution of

\[ g'(x^k - \alpha p^k)p^k = ug'(x^k)p^k, \quad 0 \leq u < 1 \]

c) Approximate minimization: Replace \( g(x^k - \alpha_k p^k) \) by a suitable function \( \varnothing \) and let \( \alpha_k \) be the minimum of this \( \varnothing \).

d) Compound algorithms: Apply secondary iteration to the solution of either a) or b) and break off after finitely many steps.
e) Range algorithms: Define admissible range for the step length and choose any $a$ in that range.

D. Embedding Methods.

In order to solve $Fx = 0$ find a simpler system $F_0x = 0$ which has the known solution $x^0$. Define an "embedding" $H(x,t)$ where $H(x,0) = F_0x$ and $H(x,1) = Fx$. Then $H(x,t) = 0$ has a solution curve $x(t)$. Move along the solution curve in small steps of $t$--that is, solve

$$H(x,t_{j+1}) = 0$$

iteratively by starting from $x(t_j)$. Other approach, solve the differential equation

$$H_x(x,t)x'(t) + H_t(x,t) = 0$$

by the same approximate process."

The Generalized Projection method presented in this paper is a minimization method which is a combination of paraboloid and relaxation techniques. The $\alpha_k$'s are chosen by exact minimization.

The basic idea in minimization is to solve the system $Fx = 0$ by an iteration technique which determines a sequence $\{x^k\}_{k=1}^\infty$ such that $g(x^k)$ is reduced at each step. The function $g$ is generally chosen to be the norm of $Fx (\|Fx\|)$. The procedure then is to determine a change vector, $dx^k$, so that

$$\|Fx^k\| \geq \|F(x^k+dx^k)\|$$

where $x^{k+1} = x^k + dx^k$.

A more common way to write $dx^k$ is to express it as the product of a scalar, $\alpha_k$, and a vector, $p_k$. Thus

$$x^{k+1} = x^k + \alpha_k p_k.$$
\( \alpha_k \) is called the stepsize and \( p_k \) the direction vector. As was indicated on an earlier page, various choices of \( \alpha_k \) and \( p_k \) give rise to different methods. Of particular interest are the choices of \( p_k \) where \( p_k \) has only one nonzero component and where \( p_k \) has all nonzero components. These two choices give rise to single step methods and total step methods respectively. In a single step method only one component of the approximation is changed while in a total step method all components in the approximation vector may be altered. Thus, for a vector with \( n \) components, \( n \) single steps are needed to accomplish one total step, or, as it is sometimes called, a cycle.

One obvious choice of \( p_k \) to obtain a single step method is to let \( p_k = e_i, i = k \text{mod}(n) \), for an \( n^{\text{th}} \) order system, where \( e_i \) is the \( i^{\text{th}} \) column of the \( nxn \) identity matrix. This particular choice of the direction vector causes a change to occur along the \( i^{\text{th}} \) axis in \( n \) dimensional space.

Householder\(^1\) discusses methods of projection from the approach of subspace mappings and also presents some of the general forms that such methods assume for various subspaces. This discussion along with a functional analysis definition of projections should be most useful to anyone wishing to investigate projection methods for research purposes. In fact, it is the author's opinion that a background in functional analysis is almost a must if progress is to be made in obtaining good methods for solving systems of nonlinear equations.

---

A minimization technique using $e_k$, $k = 1, 2, \ldots, n$, for the choice of the $p_k$ has been developed for systems of linear equations. 1, 2 Convergence has been proven but the rate of convergence was found to be slow for some systems. To overcome this effect an acceleration method was developed 3 and found to be quite effective. In this paper Keller's approach 4 has been used and the method has been extended to cover systems of nonlinear equations.

---


II. THE GENERALIZED PROJECTION METHOD

A. Development of the Method

Let \( F(x) = 0 \) be a system of nonlinear equations, where \( F: x \rightarrow y \), and \( x, y \in \mathbb{E}^n \). This system of equations may be written as follows:

\[
\begin{align*}
    f_1(x) &= 0 \\
    f_2(x) &= 0 \\
    \vdots \\
    f_n(x) &= 0
\end{align*}
\]

where the \( f_i(i = 1, 2, \ldots, n) \) are nonlinear equations and \( x \in \mathbb{E}^n \). Define \( R(x) \) as follows: \( R(x) = F(x) \).

Then \( R_i(x) = f_i(x) \quad i = 1, 2, \ldots, n \).

We shall call \( R(x) \) the residual vector at \( x \) with components \( R_i(x) \).

In the Generalized Projection method to be developed a sequence of approximation vectors \( \{x^{(i)}\}_{i=0}^{\infty} \) shall be generated which approach a solution of \( F(x) = 0 \). The method is iterative and each \( x^{(i)} \) is formed by determining a change vector \( dx^{(i)} \) such that

\[
x^{(i+1)} = x^{(i)} + dx^{(i)} \quad i = 1, 2, \ldots
\]

and

\[
|| R(x^{(i+1)}) ||^2 \leq || R(x^{(i)}) ||^2 \quad \text{for each } i
\]

(i.e. the square of the Euclidean norm of the residue vector forms a monotonic nonincreasing sequence).

In the third part of the chapter it shall be shown that the sequence \( \{|| R(x^{(i)}) ||^2\}_{i=0}^{\infty} \) tends to zero as \( i \) approaches infinity and hence the corresponding sequence of \( x^{(i)} \) converges to a solution of the given system.
The following notation shall be used:\(^1\):

Let \( x(k,i) = (x_1(k+1), x_2(k+1), \ldots, x_i(k+1), x_{i+1}(k), x_{i+2}(k), \ldots, x_n(k) \) 

where \( i = 0,1,2,\ldots,n \) and \( k = 0,1,2,\ldots \); \( k \) denotes the fact that the \( k^{\text{th}} \) cycle is being executed in the iteration scheme and \( i \) indicates the particular component that is currently being changed. Since the Projection method being proposed is a single step method the vector \( x(k,i) \) indicates that \( (k-1) \) cycles plus \( i \) single steps have been completed; that is, \( (k-1)n + i \) single steps in a system of order \( n \) have been executed.

With \( x(k,i) \) as defined above \( R(x(k,i)) \) may be written as 

\[ R(x(k,i)) = R(k,i) = (R_1(x(k,i)), R_2(x(k,i)), \ldots, R_n(x(k,i))) \]

and 

\[ R(k,i) = F(x(k,i)) \]

\[ x(k,n) = x(k+1,0) \]

\[ R(k,n) = R(k+1,0). \]

With these definitions the iteration scheme may be written as follows:

\[ x(k,i+1) = x(k,i) + dx(k,i+1) \]

where \( i = 0,1,2,\ldots,n-1 \) and \( k = 0,1,2,\ldots \).

Since the Generalized Projection method is a single step method the vector \( dx(k,i+1) \) has at most one nonzero component, the \( (i+1)^{\text{st}} \).

In what follows when the term "norm" is used the Euclidian norm is to be understood as the norm that is to be used. An expression shall now

---

\(^1\) Keller, op. cit.
be developed that will be used to find the nonzero component of 
\(dx(k,i+1)\) such that the norm of the residue vector squared 
is reduced at each single step.

Using the above notation the system of equations can be written as
follows:

\[
R(k,i) = \sum_{j=1}^{n} f_j(x^{(k+1)}, x_1^{(k+1)}, \ldots, x_{i+1}^{(k+1)}, x_{i+2}^{(k)}, \ldots, x_n^{(k)})
\]

for \(j = 1, 2, \ldots, n\). (II-1)

Also

\[
R(k,i+1) = \sum_{j=1}^{n} f_j(x^{(k+1)}, x_1^{(k+1)}, \ldots, x_{i+1}^{(k+1)}, x_{i+2}^{(k)}, \ldots, x_n^{(k)})
\]

for \(j = 1, 2, \ldots, n\). (II-2)

Subtracting (II-1) from (II-2) one obtains

\[
R(k,i+1) - R(k,i) = \sum_{j=1}^{n} f_j(x^{(k+1)}, x_1^{(k+1)}, \ldots, x_{i+1}^{(k+1)}, x_{i+2}^{(k)}, \ldots, x_n^{(k)}) - f_j(x^{(k+1)}, x_1^{(k+1)}, \ldots, x_{i+1}^{(k+1)}, x_{i+2}^{(k)}, \ldots, x_n^{(k)})
\]

for \(j = 1, 2, \ldots, n\).

This equation can be written as

\[
R(k,i+1) = R(k,i) + \sum_{j=1}^{n} f_j(x^{(k+1)}, x_1^{(k+1)}, \ldots, x_{i+1}^{(k+1)}, x_{i+2}^{(k)}, \ldots, x_n^{(k)}) - f_j(x^{(k+1)}, x_1^{(k+1)}, \ldots, x_{i+1}^{(k+1)}, x_{i+2}^{(k)}, \ldots, x_n^{(k)})
\]

(II-3)
Assuming that \( F(x) \) and its partial derivatives of order one are differentiable at all points along the line joining \( x^{(k,i)} \) and \( x^{(k,i+1)} \), the functions \( f_j(x^{(k,i+1)}) \) \((j = 1,2,\ldots,n)\) may be written as follows by Taylor's theorem for functions of several variables.

\[
\begin{align*}
\hat{f}_j(x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_i^{(k+1)}, x_{i+1}^{(k+1)}, \ldots, x_{i+2}^{(k+1)}, \ldots, x_n^{(k+1)}) &= \left( f_j(x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_i^{(k+1)}, x_{i+1}^{(k)}, \ldots, x_n^{(k)}) + \right. \\
&\left. \frac{1}{2!} \sum_{i=1}^{n} \hat{f}_j(x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_i^{(k+1)}, x_{i+1}^{(k)}, \ldots, x_n^{(k)}) \right) \left( x_i^{(k+1)} - x_i^{(k)} \right)^2 + O\left( \left( x_i^{(k+1)} - x_i^{(k)} \right)^3 \right)
\end{align*}
\]

where \( \hat{f}_j(x_i^{(k)}) \) denotes the first partial of \( f_j(x_i^{(k)}) \) with respect to the \((i+1)\)th component of \( x^{(k,i)} \) and 0 denotes the remainder in Taylor's theorem.

Letting \( dx_{i+1}^{(k,i+1)} = x_{i+1}^{(k+1)} - x_{i+1}^{(k)} \) and dropping the remainder, \((\text{II-3})\) can be approximated by

\[
R_{j}^{(k,i+1)} = R_{j}^{(k,i)} + dx_{i+1}^{(k,i+1)} \hat{f}_j(x_i^{(k)})
\]

\((\text{II-4})\) gives an expression for the residue vector at any step in terms of the previous residue vector. This expression shall be used to choose \( dx_{i+1}^{(k,i+1)} \) such that \( \|R^{(k,i+1)}\|^2 \leq \|R^{(k,i)}\|^2 \), or, equivalently, it is desired to have

\[
\sum_{j=1}^{n} (R_{j}^{(k,i)})^2 - \sum_{j=1}^{n} (R_{j}^{(k,i+1)})^2 \geq 0 \quad (\text{II-5})
\]
Substituting (II-4) in (II-5) one obtains

\[ \sum_{j=1}^{n} (k,i) - \sum_{j=1}^{n} (R_j + dx \cdot f_{j,(x(k))}) = \sum_{j=1}^{n} (k,i) - \sum_{j=1}^{n} (k,i) + 2 \sum_{j=1}^{n} (dx_j \cdot f_{j,(x(k))}) \]

At this point an expression for \( dx_{(k,i+1)} \) could be found which would give a range of values that yield the desired result of decreasing the norm of the residue vector squared. The next step should be to determine \( dx_{(k,i+1)} \) so that \( \| R_{(k,i)} \|^2 - \| R_{(k,i+1)} \|^2 \) is a maximum value at each step. To do this the usual approach is used - differentiate (II-6) with respect to \( dx_{(k,i+1)} \), equate the result to zero and solve for \( dx_{(k,i+1)} \).

Doing so the following is obtained.

\[ \sum_{j=1}^{n} (k,i) - \sum_{j=1}^{n} (k,i) + 2 \sum_{j=1}^{n} (dx_j \cdot f_{j,(x(k))}) = 0 \]

and thus
\[
\frac{dx(k,i+1)}{i+1} = - \frac{\sum_{j=1}^{n} R_{j}(x(k)) f_j(x(k))}{\sum_{j=1}^{n} (f_j(x(k)))^2}
\]

or

\[
\frac{dx(k,i+1)}{i+1} = - \left[ \begin{array}{cc}
(k,i) & (k,i) \\
R_{i+1}, J_{i+1}(x) & (x) \\
J_{i+1}(x), J_{i+1}(x)
\end{array} \right] \quad \text{(II-7)}
\]

where \( J_{i+1}(x(k,i)) \) is the \((i+1)^{st}\) column vector of the Jacobian matrix of \( F(x) \) at \( x = x(k,i) \) for \( i = 0, 1, 2, ..., n-1 \) and \([r,s]\) denotes the inner product of the vectors \( r \) and \( s \).

Now a rather simple method for determining the change in each component of the approximation vector so that the norm of the residue vector, squared, is decreased a maximum amount at each step is available.

Before proceeding to a convergence proof of the resulting iteration scheme let us note a second form by which the square of the norm of the residue vector may be expressed. Recall that

\[
\left[ \begin{array}{cc}
(k,i) & (k,i) \\
R_{i+1}, J_{i+1}(x) & (x) \\
J_{i+1}(x), J_{i+1}(x)
\end{array} \right] = \left| R_{i+1} \right| \cdot \left| J_{i+1}(x) \right| \left| \cos \theta \right| \quad \text{(II-8)}
\]

where \( \theta \) is the angle between \( R(x(k,i)) \) and \( J_{i+1}(x(k,i)) \).
By (II-6)

\[ \sum_{j=1}^{n} (R_{j}^{(k,i)2}) - \sum_{j=1}^{n} (R_{j}^{(k,i+1)2}) = \]

\[ = \sum_{i+1} (k,i+1) n (k,i) - \sum_{i+1}^{n} f_{j} (x^{(k)}) - (dx^{(k)}) \sum_{i+1}^{n} f_{j} (x^{(k)}) \]

Rewriting this in terms of norms and inner products one has

\[ \|R_{k,i}\|^{2} - \|R_{k,i+1}\|^{2} = -2dx \sum_{i+1}^{n} \left[ R_{i+1} (k,i), J (x_{i+1}) \right] \]

\[ = \sum_{i+1}^{n} \left[ J (x_{i+1}), J (x_{i+1}) \right] \]

(II-9)

Substituting (II-7) in the appropriate places in (II-9) and simplifying the resulting equation the following results are obtained.

\[ \|R_{k,i}\|^{2} - \|R_{k,i+1}\|^{2} = \frac{2}{\sum_{i+1}^{n} \left[ R_{i+1} (k,i), J (x_{i+1}) \right]^{2} - \sum_{i+1}^{n} \left[ J (x_{i+1}), J (x_{i+1}) \right]^{2}} \]

\[ \frac{\left[ R_{i+1} (k,i), J (x_{i+1}) \right]^{2}}{\left[ J (x_{i+1}), J (x_{i+1}) \right]^{2}} \cdot \left[ \left[ R_{i+1} (k,i), J (x_{i+1}) \right]^{2} - \left[ J (x_{i+1}), J (x_{i+1}) \right]^{2} \right] = \]
Now, using (II-8), equation (II-10) can be written as

\[
\frac{\left(\begin{array}{cc}
(k,i) & (k,i) \\
R_{i+1} & J_{i+1}
\end{array}\right)^2}{\left|J_{i+1}(x)\right|^2} - \frac{\left(\begin{array}{cc}
(k,i) & (k,i) \\
R_{i+1} & J_{i+1}
\end{array}\right)^2}{\left|J_{i+1}(x)\right|^2} = \frac{\left(\begin{array}{cc}
(k,i) & (k,i) \\
R_{i+1} & J_{i+1}
\end{array}\right)^2}{\left|J_{i+1}(x)\right|^2}
\]

Hence

\[
\frac{\left(\begin{array}{cc}
(k,i) & (k,i) \\
R_{i+1} & J_{i+1}
\end{array}\right)^2}{\left|J_{i+1}(x)\right|^2} = \frac{\left(\begin{array}{cc}
(k,i) & (k,i) \\
R_{i+1} & J_{i+1}
\end{array}\right)^2}{\left|J_{i+1}(x)\right|^2}
\]

\[
= \left|J_{i+1}(x)\right| \left| R_{i+1,k} \right|^2 - \left|J_{i+1}(x)\right| \left| R_{i+1,k} \right|^2
\]

\[
= \left| J_{i+1}(x) \right| (1 - \cos \theta)
\]

(II-11)
By repeated applications of (II-11) it can be verified that
\(|R(k,i+1)|^2 (i=0,1,...,n-1)\) can be written in terms of \(|R(k,0)|^2\) and
a product of the form

\[
\sum_{i=0}^{n-1} |R(k,i+1)|^2 = \sum_{i=0}^{n-1} \prod_{p=1}^i (1 - \cos \theta_{R,J})
\]

i.e.
\[
|R(k,i+1)|^2 = |R(k,0)|^2 \prod_{p=1}^i (1 - \cos \theta_{R,J})
\]

for \(i = 0,1,2,...,n-1\) and \(k = 0,1,2,...\).

Going still further by using (II-11) repeatedly on \(|R(k,0)|^2\) it is possible to express \(|R(k,i+1)|^2\) in terms of \(|R(0,0)|^2\) and a product of \(2\) elements of the form \((1 - \cos \theta_{R,J})\)

where \(i = 0,1,2,...,n-1\) and \(j = 0,1,2,...,k\).

The result obtained by carrying out this process is

\[
|R(k,i+1)|^2 = \prod_{j=0}^{i} (1 - \cos \theta_{R,J}) \cdot \prod_{m=0}^{k-n+i} (1 - \cos \theta_{R,J})
\]

Relation (II-13) will be used in a later section of this chapter when convergence of the Generalized Projection method is proved.
B. The Generalized Projection Method

Before proceeding to the convergence proof of the Generalized Projection method a description of the process resulting from the development of the method, given above, is presented. There are four steps to the method.

1. Choose an initial approximation vector, \( x^{(0,0)} \).

2. Determine the residual vector for the current approximation vector.

3. Calculate

\[
\begin{align*}
\Delta x_{i+1}^{(k)} &= - \left( \frac{\left( k, i \right)}{R} \right) R_{i+1}^{(k,i)} J(x_{i+1}) \left( x_{i}^{(k,i)} \right) - \frac{\left( k, i \right)}{J(x_{i+1})} J_{i+1}^{(k,i)} J(x_{i+1}) \\
&\quad + \left( x_{i+1}^{(k,i)} - x_{i}^{(k,i)} \right) \\
x_{i+1}^{(k,i+1)} &= x_{i}^{(k,i)} + \Delta x_{i+1}^{(k,i+1)}
\end{align*}
\]

4. Test for convergence - if the approximation vector is close enough to a solution print the results, if not repeat steps 2-4, incrementing the \( i \) superscript by 1 at each step to a maximum of \( n-1 \) (\( n \) the size of the system). For every \( n \) steps increment \( k \) by 1 and reset \( i \) to zero.

The order of these steps may be rearranged. The choice of the ordering depends on the convergence criteria to be used. In the implementation used for this work the relative change in each component after a cycle is used. Hence the above order is convenient. It is also possible to use the square of the norm of the residue vector as a
convergence criteria. If such a test is used it is more convenient to interchange steps 3 and 4.

Using the algorithm described above, for each single step, in a system of order \( n \), it is necessary to perform \( 2n \) function evaluations — \( n \) functions for the residue vector and \( n \) functions for the column vector of the Jacobian matrix. To obtain a new approximation vector in a single step \( 2n + 1 \) multiplications and \( 2n + 1 \) additions are needed. For a cycle then, \( 2n^2 \) function evaluations, \( 2n^2 + n \) multiplications, and \( 2n^2 + n \) additions must be performed.

At this time the residue vector, \( R^{(k,1)} \), is calculated by actually evaluating the functions that form the nonlinear system. From the development of the method it can be seen that it is also possible to approximate the residue vector at any step if the residue vector of the previous step is known. Equation (II-4) of the preceding section gives the relation that can be used for this purpose.

C. Geometric Description of the Generalized Projection Method

To achieve a better grasp of the details of this method a geometric description will prove useful. In Figure 1 and Figure 2 diagrams of the residue vector changes and approximation vector changes, respectively, are presented for a system of order two.

In Figure 1, \( R(x^{(0,0)}) \) is the initial residue vector associated with the initial approximation \( x^{(0,0)} \) in Figure 2. In the first step the first column, \( J_1(x^{(0,0)}) \), of the Jacobian matrix evaluated at \( x^{(0,0)} \) is used with \( R(x^{(0,0)}) \) to determine \( dx^{(0,1)} \). The residual vector that gives a minimum at this step is the vector which is perpendicular to \( J_1(x^{(0,0)}) \).
from the origin 0 - i.e. the vector $R(x^{(0,1)})$. The value $dx^{(0,1)}$ is used to obtain the approximation vector $x^{(0,1)}$, Figure 2. The process is repeated using $R(x^{(0,1)})$ and $J_2(x^{(0,1)})$, the second column of the Jacobian matrix evaluated at $x^{(0,1)}$, to obtain $R(x^{(0,2)})$, which by definition is also $R(x^{(1,0)})$, and also to obtain $dx^{(0,2)}$, which is also $dx^{(1,0)}$. The first cycle is now completed. The second cycle is then started and the process continues until convergence is achieved.

Figure 1. Residue vectors for four steps in a system of order two

Figure 2. Approximation vectors for four steps in a system of order two
The preceding diagram of the residue vectors, Figure 1, is also useful in indicating how the proof of convergence might be carried out. To verify that the method converges it is required to show that the norm of the residue vector, squared, goes to zero as the number of iterations increases. Geometrically it can be shown what will happen if the residue vector does not go to the zero vector. Suppose that \( \| R(x^{(k,l)}) \|^2 > c \) as \( k \to \infty \). For geometric simplicity let us assume that the given system is of order two. Then for each step the residue vector extends outside the circle of radius \( \sqrt{c} \). See Figure 3.

![Figure 3. Residue vectors lie outside the circle of radius \( \sqrt{c} \)](image)

If at any step the residue vector terminates on the circumference of the circle of radius \( \sqrt{c} \), then all the column vectors of the Jacobian matrix are parallel to one another. Otherwise further steps will bring the residue vector inside the circle (see Figure 4). Recall that the Jacobian matrix is singular if any two columns of the matrix are parallel. This will prove very useful in the theoretical proof of convergence which is presented in the next section.
Before giving the actual convergence theorem a few preliminary theorems and definitions will be given.

**Theorem 1**

The sequence \( \{ \| R(x^{(k,i)}) \| ^2 \} _{k=1}^\infty \) \( i = 1, 2, \ldots, n \) converges to a nonnegative value.

**Proof:**

By construction it is known that \( | R^{(k,i)} |^2 \geq | R^{(k,i+1)} |^2 \) and \( | R^{(k,i)} |^2 \geq 0 \). Hence the sequence is monotonic nonincreasing, bounded below by zero and above by \( | R^{(0,0)} |^2 \), which implies convergence to a nonnegative value.

**Definition 1:** A function \( f: A \rightarrow B \) is said to be continuous at \( a \in A \) if for every neighborhood \( G(\epsilon, B) \) of \( f(a) \), \( f^{-1}[G] \) is a neighborhood of
a; f is said to be continuous on A if it is continuous at each point of A.\(^1\)

Definition 2: A semimetric for a set is a function \(d\) of two variables satisfying for all \(a,b,c\), in the set:

(i) \(d(a,b) = d(b,a) \geq 0\)

(ii) \(d(a,a) = 0\)

(iii) \(d(a,c) \leq d(a,b) + d(b,c)\)

The set together with the semimetric is called a semimetric space.\(^2\)

Theorem 2 Let \(A,B\) be semimetric spaces. \(f: A \rightarrow B\) is continuous at \(a \in A\) if and only if whenever \(\{x_n\}\) is a sequence converging to \(a\), \(f(x_n) \rightarrow f(a)\).\(^3\)

Definition 3: The level set of \(f\) at \(x^0\) is the set \(S = \{x \in \mathbb{R}^n: f(x) \leq f(x^0)\}\).\(^4\)

Theorem 3 Let \(f\) be a continuous real function. Then the level set \(S = \{a: f(a) \leq x\}\) is closed for each fixed real number \(x\).\(^5\)

Theorem 4 If \(a_1,a_2,\ldots,a_n,\ldots\) all lie between 0 and 1 then the necessary and sufficient condition for the convergence of the


\(^{2}\)Ibid., p. 60.

\(^{3}\)Ibid., p. 61.


\(^{5}\)Wilansky, op. cit., p. 63.
infinite products \( \prod_{n=1}^{\infty} (1+a_n) \) and \( \prod_{n=1}^{\infty} (1-a_n) \) is the convergence of the series \( \sum_{n=1}^{\infty} a_n \).

Definition 4: The infinite products given in Theorem 4 are said to converge if the products approach a value other than zero or infinity in the limit.

Theorem 5 Let \( F(x) = \emptyset \) be a system of continuous equations (linear or nonlinear) and assume \( s = \{ x : x \in \mathbb{R}^n, \| F(x) \| \leq \| F(x(0)) \| \}, x(0,0) \in \mathbb{R}^n \) is bounded. If the Jacobian matrix of \( F \) exists and is nonsingular on the level set \( (s) \) of \( F \) for some initial vector \( x(0,0) \) then the sequence of vectors \( \{ x(k,i) \}_{k=1}^{\infty} \) generated by the Generalized Projection Method contains a subsequence which converges to a solution of the given system.

Proof: The result shall be verified by contradiction. By Theorem 1 it is known that the sequence \( \{\| R(k,i) \|^2 \}_{k=0}^{\infty} \) converges. Assume that the limit of this sequence is \( c \), where \( c > 0, c \in \mathbb{R} \).

i.e. \( \lim_{k \to \infty} \| R(k,i) \|^2 = c \)

From the development of the Generalized Projection method it was determined that

\[
\| R(k,i) \|^2 = \| R(0,0) \|^2 \frac{k \cdot n + i}{m} (1 - \cos^2 \theta_m)
\]

and thus by the above assumption the infinite product

$$\lim_{k \to \infty} \prod_{m=0}^{k-n+i} (1 - \cos^2 \theta_m)$$

converges.

The proof shall now be sectioned into three cases.

They are:

1. All values of $\cos^2 \theta_m$ lie between 0 and 1.
2. $\cos^2 \theta_m = 0$ for some $m$.
3. $\cos^2 \theta_m = 1$ for some $m$.

Case 1. From Theorem 4 it follows that $\cos^2 \theta_m \to 0$ as $m \to \infty$. By Theorem 2, since $||R(k,i)||^2$ converges to $c$ and $||R(k,i)||^2$ is a continuous function of a continuous function, hence continuous, the sequence $\{x(k,i)\}$ generated by the Generalized Projection method converges to some $x'$. But by Theorem 3 and the fact that $s$ is bounded, $x' \in s$.

Thus, for $x'$, $||R(x')||^2 = c > 0$.

and

$$[R(x'), J_i(x')] = 0 \quad i = 1, 2, \ldots, n$$

since $\cos^2 \theta_m = 0$ in the limit.

But this implies that the Jacobian of $F$ at $x'$ is singular since $R(x') \neq 0$, contradicting the nonsingularity of the Jacobian on $s$. Hence

$$\lim_{k \to \infty} ||R(k,i)||^2 \to 0$$
Case 2. If $\cos^2 \theta_m = 0$ then the residue vector and the column of the Jacobian matrix at this step are orthogonal to one another. The result is that there is no change in the norm of the residue, nor in the approximation vector. Moreover, this situation can occur at most $n-1$ times per cycle for an $n^{th}$ order system. The reason for this is that if it happened $n$ times the Jacobian matrix would be singular. As a result, convergence is not lost but merely slowed down. In fact, these occurrences could be removed from the infinite product.

Case 3. $\cos^2 \theta_m = 1$ implies that the residue vector and the associated column vector at this step are parallel to one another. In this situation the residue vector will be reduced to the zero vector and convergence will be achieved.
III. COMPARISON WITH OTHER METHODS

In this chapter the Generalized Projection method is compared with two other methods for solving systems of nonlinear equations. The first is the Newton-Raphson method,\(^1\) the classical method used to solve such systems; and the second is a method developed by Dr. Kenneth M. Brown\(^2\) in his doctoral dissertation at Purdue in 1966. There are several reasons for choosing these two methods. To date most methods, if not all, have been compared with the Newton-Raphson method. Also, Brown\(^3\) has run comparisons with several of the more recent methods developed for solving systems of nonlinear equations. Hence, with these two choices one obtains comparisons with some of the other methods available.

The comparison norms used in this paper are:

1) Number of iterations: In this case we shall consider a cycle as an iteration. The reason for this is the fact that both the Newton-Raphson method and Brown's method are total step methods.

2) CPU time: This parameter is usually not considered in comparing numerical methods. However, it was felt that the time required to solve a problem on a computing system is something of great importance. To determine whether a method is "good" the total picture should be

---


examined. It is not enough to say that one method converges in fewer iterations than another method. The work involved at each step must be examined. It is quite conceivable that a method could converge in a few iterations, yet the time involved at each step may be considerably long. For example, in the Newton-Raphson method it is necessary to solve a system of linear equations at each step. The amount of time involved in solving the system of linear equations will vary with the method used and the size of the system. As a result, the CPU time for each method used in the comparisons is given in the tables of comparisons. The time values (in seconds) given are the actual time that the CPU was used from the start of the actual computations (i.e., after the input is read) to the point where the approximate solution is printed. This time was determined through the use of a subroutine ALTIME, which is available at the Iowa State University Computation Center\(^1\). Due to the fact that the system at Iowa State is run under MVT, the CPU time may vary for the same job on different runs. The reason for this is that if any cycle stealing is performed during the execution of a task the timer is not stopped. Thus, depending on the amount of cycle stealing, the time value returned by ALTIME depends on the environment at the time of execution. The time given in the tables is the time value obtained from ALTIME when the examples were run as the only jobs in the system. Under these conditions a minimum of interrupts should have occurred.

\(^1\)Kendall White and G. Scranton. ALTIME: assembler language subroutine for absolute time and interval timing. ISU Program Library, Iowa State University of Science and Technology Computation Center, Ames, Iowa. 1970.
In the tables that are to be presented, for each initial vector, the solution to eight decimal places, the residue vector associated with the solution, the number of cycles to reach the desired accuracy and the CPU time are given for each of the three methods considered. The fixed part of each of the three programs was compiled using the IBM 360 FORTRAN G compiler. The resulting object decks were then combined with the proper subroutines and run using FORTRAN G. Storage requests were kept to a minimum. For the execute step the Newton-Raphson method requires 48K (K = 1024) bytes, Brown's method 50K, and the Generalized Projection method, 38K bytes. To be run on the 360 system at Iowa State University, 48K bytes must be allocated for the Newton-Raphson and Generalized Projection methods and 64K for Brown's method. This is due to the fact that core is allocated in blocks of 16K bytes with a minimum of 32 K. Also, the core request for a job is used in determining the cost to execute the job. Each program was set up to allow for a maximum of 30 equations with 30 unknowns. A maximum of 501 cycles were permitted for each case.

The tables on the following pages have five column headings.

1) Method: names the method being used to solve the system.

2) Solution: lists the solution vector obtained by the corresponding method.

3) Res: standing for residue, gives the order of the residue for each equation in the system associated with the solution vector.
4) Iter: standing for iterations, gives the number of cycles required to arrive at the given solution vector.

5) CPU time: gives the amount of time the CPU was used in executing the program after the initial read.

If the method fails to converge a reason is given and the table will not have any other entries for the method. The solution vector is given to eight decimal places, although the actual output for each job was given to sixteen places. As a result the residues may differ slightly between the methods for the same solution.

Note the definition of convergence that is used for the Generalized Projection method. To converge it is necessary that all the elements in the change vector, CHNG, be less than 10^{-15} and the residue vector components be less than 10^{-10}. These conditions demand more than what is usually required for convergence. Because of this, if the method does not converge in 501 cycles, the approximate solution for the last cycle is given along with its associated residue vector. Where this occurs the approximation vector and the residue vector are given in a footnote for the table associated with the example.

Comparison tables:

Test case 1

\[
\begin{align*}
20x_1 - \cos^2(x_2) + x_3 - \sin(x_3) &= 37^1 \\
\cos(2x_1) + 20x_2 + \log_{10}(1 + x_4^2) &= -5 \\
\sin(x_1 + x_2) - x_2 + \tan(x_3) + 19x_3 &= 12 \\
2\tanh(x_2) + e(-2x_3^2 + 0.5) + 21x_4 &= 0
\end{align*}
\]

Table 1.1 Initial vector 2.01, -0.31, 0.63, -0.08

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>1.89651398</td>
<td>$10^{-14}$</td>
<td>6</td>
<td>0.101</td>
</tr>
<tr>
<td></td>
<td>-0.21025166</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.54208732</td>
<td>$10^{-15}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.02388595</td>
<td>$10^{-16}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>1.89651398</td>
<td>$10^{-14}$</td>
<td>3</td>
<td>0.066</td>
</tr>
<tr>
<td></td>
<td>-0.21025166</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.54208732</td>
<td>$10^{-15}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.02388595</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>1.89651398</td>
<td>$10^{-14}$</td>
<td>10</td>
<td>0.117</td>
</tr>
<tr>
<td></td>
<td>-0.21025166</td>
<td>$10^{-15}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.54208732</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.02388595</td>
<td>$10^{-16}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.2 Initial vector 1.0, 1.0, 1.0, 1.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>1.89651398</td>
<td>$10^{-14}$</td>
<td>7</td>
<td>0.117</td>
</tr>
<tr>
<td></td>
<td>-0.21025166</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.54208732</td>
<td>$10^{-15}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.02388595</td>
<td>$10^{-16}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>1.89651398</td>
<td>$10^{-14}$</td>
<td>4</td>
<td>0.101</td>
</tr>
<tr>
<td></td>
<td>-0.21025166</td>
<td>$10^{-15}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.54208732</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.02388595</td>
<td>$10^{-16}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>1.89651398</td>
<td>$10^{-14}$</td>
<td>10</td>
<td>0.101</td>
</tr>
<tr>
<td></td>
<td>-0.21025166</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.54208732</td>
<td>$10^{-15}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.02388595</td>
<td>$10^{-16}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test case 2  \[
\frac{(1-\frac{1}{4\pi}) \left(e^{2t} - 1\right) + 2x_2 - 2x_1}{\pi} = 0
\]
\[
\frac{1}{2} \sin(x_1x_2) - x_2 - x_1 = 0
\]

Table 2.1 Initial vector 0.6, 3.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>0.500000000</td>
<td>10^{-16}</td>
<td>6</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>3.14159265</td>
<td>10^{-16}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>0.50000000</td>
<td>0</td>
<td>6</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>3.14159265</td>
<td>10^{-16}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>0.500000000</td>
<td>10^{-15}</td>
<td>31</td>
<td>0.132</td>
</tr>
<tr>
<td></td>
<td>3.14159265</td>
<td>10^{-15}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2 Initial vector 0.0, 0.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>-0.26059929</td>
<td>10^{-16}</td>
<td>7</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>0.62253089</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>-0.26059929</td>
<td>10^{-16}</td>
<td>5</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>0.62253089</td>
<td>10^{-16}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>-0.26059929</td>
<td>10^{-15}</td>
<td>396</td>
<td>1.617</td>
</tr>
<tr>
<td></td>
<td>0.62253089</td>
<td>10^{-14}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Obviously the Projection method is slow for this problem. For the initial vector (0.6, 3.0) the Jacobian matrix very quickly becomes almost fixed. The determinant of the Jacobian matrix is less than 0.5 after 7 cycles. This indicates that the approximation vector is near a singularity of the Jacobian matrix which will cause the method to proceed slowly. Also, all terms in the matrix are rather small, the largest in absolute value being 0.87. These small values along with a small residue vector yield very small changes in the approximation vector. After 10 cycles the changes are of the order $10^{-6}$.

For the initial vector (0.0, 0.0) the method is quite slow. What happens in this example should not occur too often. The column vectors of the Jacobian matrix have an angle of about $162^\circ$ between them after two cycles and the residue vectors change at each cycle so that the angle between them and the columns of the Jacobian matrix is a little better than $110^\circ$. This is consistently close enough to orthogonality to cause the slowness. This situation could be called a type of "ill-conditioning" for the Generalized Projection method.

Test case 3

\[
\begin{align*}
\frac{x_1^2}{1} - x_2 &= 1.0^1 \\
(x_1 - 2) + (x_2 - 0.5)^2 &= 1.0
\end{align*}
\]

Table 3.1 Initial vector 0.1, 2.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>1.06734608</td>
<td>10^-15</td>
<td>26</td>
<td>0.117</td>
</tr>
<tr>
<td></td>
<td>0.13922766</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>1.06734608</td>
<td>10^-15</td>
<td>7</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>0.13922766</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>1.54634288</td>
<td>10^-15</td>
<td>47</td>
<td>0.101</td>
</tr>
<tr>
<td></td>
<td>1.39117631</td>
<td>10^-15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2 Initial vector 0.0, 0.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>1.06734608</td>
<td>10^-16</td>
<td>8</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>0.13922766</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>1.06734608</td>
<td>10^-15</td>
<td>8</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>0.13922766</td>
<td>10^-15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>1.06734608</td>
<td>10^-16</td>
<td>13</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>0.13922766</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3 Initial vector -1.0, -1.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>failed to converge singular Jacobian matrix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>1.06734608</td>
<td>10^-15</td>
<td>14</td>
<td>0.050</td>
</tr>
<tr>
<td></td>
<td>0.13922766</td>
<td>10^-16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>1.06734608</td>
<td>0</td>
<td>13</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>0.13922766</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In this example both roots of the system of nonlinear equations were found by the Generalized Projection method. This was not the case for the other two methods indicating that there is a difference in the regions of convergence for the three methods. The last approximation vector (1.0, 1.0) demonstrates that the Generalized Projection may in some cases converge in fewer iterations and less time than Brown's method.

\[
\begin{align*}
\text{Test case 4} & & \quad x_1 + ((-x_2 + 5)x_2 - 2)x_2 &= 13^1 \\
& & x_1 + ((x_2 + 1)x_2 - 14)x_2 &= 29
\end{align*}
\]

Table 4.1 Initial vector 15.0, -2.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>5.00000000</td>
<td>0</td>
<td>44</td>
<td>0.167</td>
</tr>
<tr>
<td></td>
<td>4.00000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>5.00000000</td>
<td>10^{-14}</td>
<td>12</td>
<td>0.050</td>
</tr>
<tr>
<td></td>
<td>4.00000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>failed to converge^a</td>
<td>Jacobian matrix nearly singular</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

^aWith the given initial approximation the Projection method went to the vector (11.412, -0.896). The value -0.896 for the second component gives a singular Jacobian matrix. Knowing that a second singularity occurs at \(x_2 = 2.23\) the method was rerun using (11.4, 5.0) as an initial approximation. Convergence was attained in 30 cycles to the solution vector (5.0, 4.0).

^Ibid., p. 568.
Table 4.2 Initial vector 4.0, 4.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>5.00000000</td>
<td>0</td>
<td>2</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>4.00000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>5.00000000</td>
<td>$10^{-15}$</td>
<td>5</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>4.00000000</td>
<td>$10^{-14}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>5.00000000</td>
<td>0</td>
<td>1</td>
<td>0.000$^a$</td>
</tr>
<tr>
<td></td>
<td>4.00000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^a$A time of 0.000 indicates that too few instructions were executed to obtain a time value.

Two points of interest are generated in the above examples. First, there is a distinct advantage to having a feeling for the system of equations and where the points of singularity occur for the Jacobian matrix. Although the Generalized Projection method did not converge in the allotted number of cycles it was still possible to obtain a solution vector by "jumping" the points that yield singular Jacobian matrices. If this system were automated on a 2260 scope then by permitting operator intervention to the routine these "jumps" could easily be executed. Second, the Projection method was able to converge in one cycle for the initial vector (4.0, 4.0). The ideal situation has occurred here. With the given initial approximation the residue vector (-1.0, -1.0) runs parallel to the first column of the Jacobian matrix (1.0, 1.0). Under this condition the change in the first component moves the residue vector to the origin and the solution is obtained in a single step.
Test case 5

\[25x_1 + 2x_2 + x_3 = 69^1\]
\[2x_1 + 10x_2 + x_3 = 63\]
\[x_1 + x_2 + 4x_3 = 43\]

Table 5.1 Initial vector 0.0, 0.0, 0.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>2.000000000</td>
<td>0</td>
<td>2</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>5.000000000</td>
<td>$10^{-14}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9.000000000</td>
<td>$10^{-14}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>2.000000000</td>
<td>$10^{-14}$</td>
<td>2</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>5.000000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9.000000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>2.000000000</td>
<td>$10^{-14}$</td>
<td>20</td>
<td>0.066</td>
</tr>
<tr>
<td></td>
<td>5.000000000</td>
<td>$10^{-13}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9.000000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2 Initial vector 1.0, 1.0, 1.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>2.000000000</td>
<td>0</td>
<td>2</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>5.000000000</td>
<td>$10^{-14}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9.000000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>2.000000000</td>
<td>$10^{-14}$</td>
<td>2</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>5.000000000</td>
<td>$10^{-14}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9.000000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>2.000000000</td>
<td>$10^{-14}$</td>
<td>20</td>
<td>0.066</td>
</tr>
<tr>
<td></td>
<td>5.000000000</td>
<td>$10^{-13}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9.000000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This example was used to verify that the Generalized Projection method works for the special case - a linear system of equations.

As expected the Newton-Raphson method and Brown's method converged in very few cycles. This is so since the Newton-Raphson method solves the linear system directly and Brown's method does also since it is like the Gaussian elimination method for linear systems. Two cycles are needed because of the relative change test for convergence. At the end of the first cycle the solution has been found but the change in each component is too large to satisfy the convergence test. For the second cycle there is no change in any of the components and the convergence test is satisfied.

Test case 6
\[ \sin(x_1) - x_2 = -1.32 \]
\[ -x_1 + \cos(x_2) = -0.85 \]

Table 6.1  Initial vector 1.0, 1.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>0.56732513</td>
<td>0</td>
<td>6</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>10^-15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>0.56732513</td>
<td>10^-15</td>
<td>4</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>10^-16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>0.56732513</td>
<td>10^-15</td>
<td>8</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>10^-16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2 Initial vector -1.0, -1.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>0.56732513</td>
<td>0</td>
<td>32</td>
<td>0.152</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>10^-15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>0.56732513</td>
<td>10^-15</td>
<td>6</td>
<td>0.050</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>0.56732513</td>
<td>0</td>
<td>9</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>10^-15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3 Initial vector 0.0, 0.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>0.56732513</td>
<td>0</td>
<td>8</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>10^-15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>0.56732513</td>
<td>10^-15</td>
<td>4</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>10^-15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>0.56732513</td>
<td>0</td>
<td>8</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>1.85737784</td>
<td>10^-15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this example the Generalized Projection method performs quite well. The best time obtained for an initial vector was the same for Brown's method and the Generalized Projection method (0.015 sec.) while twice as many iterations were used in the latter. This suggests that there is less work per cycle in the Generalized Projection method. If one could now accelerate the rate of convergence of the Generalized Projection method without increasing the work per step by too much this method should aid in taking a giant step forward in the land of the nonlinear.
Test case 7

\[ x_1 \cdot \sin(x_1) \cdot \cosh(x_2) = 0^1 \]
\[ x_2 \cdot \cos(x_1) \cdot \sinh(x_2) = 0 \]

Table 7.1 Initial vector 1.0, 1.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>failed to converge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>solution vector tending to zero vector</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>resulting in singular Jacobian matrix.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>failed to converge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>solution vector tending to zero vector</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>modified Jacobian matrix singular</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>failed to converge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>solution is tending to the zero vector\textsuperscript{a}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\textsuperscript{a}At the end of 39 cycles, when the program stops, the approximation vector is (10^{-7}, 10^{-7}) with a residue vector of (10^{-21}, 10^{-21}).

Table 7.2 Initial vector 5.0, 5.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>7.49767627</td>
<td>10^{-14}</td>
<td>14</td>
<td>0.082</td>
</tr>
<tr>
<td></td>
<td>-2.76867828</td>
<td>10^{-15}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>failed to converge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>solution vector tending to zero vector</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>modified Jacobian matrix singular</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>7.49767627</td>
<td>10^{-14}</td>
<td>9</td>
<td>0.050</td>
</tr>
<tr>
<td></td>
<td>2.76867828</td>
<td>10^{-15}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This example brings out a problem of using relative change methods in testing for convergence. When the components of the approximation vector become small the word size of the computing system plays an important role in the success of the method, or, perhaps it should be stated that the programmer's knowledge of the system limitations on word size is important. Brown's method failed to converge because the approximation vector was nearing the zero vector. For this test case the Jacobian matrix has a very small determinant value for small $x$ and hence the modified Jacobian matrix satisfied the singularity test and the method stops. If the determinant were not small for a small approximation vector the system may stop the job because of overflow conditions occurring when the relative change values were tested. This situation can easily be avoided by adding a few instructions to the program, as was done in the Generalized Projection method.

Test case 8 $10(x_2-x_1^2) = 0$

$$l-x_1 = 0$$

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>1.000000000</td>
<td>$10^{-15}$</td>
<td>3</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>1.000000000</td>
<td>$10^{-16}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>1.000000000</td>
<td>$10^{-15}$</td>
<td>3</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>0.999999999</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>failed to converge$^a$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^a$After 501 iterations the method is slowly converging. The approximate solution at this point is $(0.90654, 0.82182)$ and the associated residue vector is of the order $(10^{-2}, 10^{-1})$.

Table 8.2 Initial vector 0.0, 0.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>1.000000000</td>
<td>0</td>
<td>3</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>1.000000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>1.000000000</td>
<td>0</td>
<td>3</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>0.999999999</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>1.000000000</td>
<td>0</td>
<td>1</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>1.000000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The system of equations used in this example is somewhat like that used in test case 2. That is, for the initial vector (-1.2, 1.0) the column vectors of the Jacobian matrix have an angle of almost 180° between them. Also the residue vector is nearly orthogonal to both column vectors. Thus the inner products of the residue vectors and the columns of the Jacobian matrix are quite small and as a result the norm of the residue vector is reduced very little at each step. It is of interest to note that this system is what is usually referred to as a mildly nonlinear system of equations. The fact that the Jacobian matrix is almost constant, only one element will vary at any step, adds to this phenomena that will be referred to as "ill-conditioning."

In Table 8.2 the Projection method required only one cycle to obtain the solution. The same thing has occurred here as in Table 4.2. At the conclusion of the first single step the residue vector (-10.0, 0.0) is parallel to the second column of the Jacobian matrix (10.0, 0.0). Thus the residue vector can be reduced to the zero vector on the next single step.
Test case 9
\[ f_i(x) = -6 + 2x_i + \sum_{j=1}^{5} x_j \quad i = 1, 2, 3, 4 \]
\[ f_5(x) = -1 + \prod_{j=1}^{5} x_j \]

Table 9.1 Initial vector 0.5, 0.5, 0.5, 0.5, 0.5

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>-0.57904308</td>
<td>0</td>
<td>20</td>
<td>0.484</td>
</tr>
<tr>
<td></td>
<td>-0.57904308</td>
<td>0</td>
<td>20</td>
<td>0.484</td>
</tr>
<tr>
<td></td>
<td>-0.57904308</td>
<td>0</td>
<td>20</td>
<td>0.484</td>
</tr>
<tr>
<td></td>
<td>-0.57904308</td>
<td>0</td>
<td>20</td>
<td>0.484</td>
</tr>
<tr>
<td></td>
<td>8.89521544</td>
<td>0</td>
<td>20</td>
<td>0.484</td>
</tr>
<tr>
<td>Brown's</td>
<td>0.99999999</td>
<td>10^{-15}</td>
<td>7</td>
<td>0.152</td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>10^{-15}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>10^{-15}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td>10^{-14}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>failed to converge[a]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[a\]After 501 iterations the method was still converging, but very slowly. The approximation vector for 501 cycles was (1.00004013, 1.00004361, 1.00004739, 1.00005150, 0.99978083) and the order of the corresponding residue vector components were (10^{-5}, 10^{-5}, 10^{-5}, 10^{-4}, 10^{-4}). The task time was 3.367 sec.

Test case 10

\[ f_i(x) = -11 + 2x_i + \sum_{j=1, j\neq i}^{10} \]

\[ f_{10}(x) = -1 + \prod_{j=1}^{10} x_j \]

Table 10.1 Initial vector 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>1.00000000</td>
<td>0</td>
<td>92</td>
<td>21.50</td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown's</td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td>8</td>
<td>1.06</td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.99999999</td>
<td>(10^{-15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>failed to converge in 501 cycles(^a)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^a\)After 501 cycles the Projection method is slowly converging to a solution. The results to this point are (0.9747, 0.9747, 0.9744, 0.9743, 0.9741, 0.9739, 0.9738, 0.9736, 0.9734, 1.2592), and the order of the corresponding residue vector components are \((10^{-2}, 10^{-2}, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-4}, 10^{-2})\). The task time to achieve this was 8.5 sec.

\(^1\)Ibid., p. 567.
Test cases 9 and 10 are alike except for size. All equations are linear but the last one, so, the system is mildly nonlinear. Since Brown's method appears best suited to nonlinear systems in which the equations are nearly linear it was expected to perform quite well on these two cases and it did. However, the reason for examining these two systems was to obtain information on the work per step increase as the size of the system increases. There is an increase in the work of about 49 times that of the smaller system for the Newton-Raphson method, about 6 times for Brown's method, and around 3 times for the Projection method. Note, however, that the larger system did not get as close to the solution for the Projection method as in the smaller system. Because of the rate at which the Projection method is converging the increase should be a lot more than three times.

The reason for the slow rate of convergence for the Projection method is that in the two systems the residue vector is nearly orthogonal to all the columns of the Jacobian matrix. In test case 9 the final approximation vector given is such that the residue vector and the columns of the Jacobian matrix have angles of about 89° between them. Hence, the inner product values are small and the rate of convergence very slow.

Test case 11 \[ x_1 \log_{10}(x_1) - 1.2 = 0 \]

This example was run to show that the Generalized Projection method works for a single nonlinear equation with one unknown value.
Table 11.1 Initial vector 2.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution</th>
<th>Res</th>
<th>Iter</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>2.74065609</td>
<td>$10^{-15}$</td>
<td>6</td>
<td>0.015</td>
</tr>
<tr>
<td>Brown's Projection</td>
<td>2.74064609</td>
<td>$10^{-15}$</td>
<td>5</td>
<td>0.015</td>
</tr>
</tbody>
</table>

It would hardly seem fitting in comparing the three methods not to compare the number of function evaluations per cycle. In doing so, however, it becomes clear that this is not a valuable norm to use.

As mentioned earlier, in Chapter II, for a system of order n the Generalized Projection method requires $2n^2$ function evaluations per cycle. Brown\(^1\) indicates that his method requires $n^2/2 + 3n/2$ function evaluations per iterative step and Newton's method $n^2 + n$ evaluations. These values indicate that the number of function evaluations per cycle for Brown's method as compared to the Generalized Projection method range in ratio from 1:1 for $n = 1$ to 1:4 as $n \to \infty$. For Brown's method and the Newton-Raphson method the corresponding ratios are 1:1 and 1:2, and for the Newton-Raphson method and the Generalized Projection method 1:1 and 1:2.

Now, if the preceding comparison tables are examined, it will be found for the cases run that the average time per cycle for the Generalized Projection method is less than or equal to that of the other

\(^1\)Kenneth M. Brown and Samuel D. Conte, op. cit.
two methods. Thus it must be concluded that the number of function evaluations in these methods is not the dominant time consumer and hence the number of function evaluations is not always a good indicator of a method's worth.

One other point of interest should be mentioned. Brown\(^1\) indicates that for Case 4 Broyden's methods, I and II, and the Damped Newton algorithm given by Spath\(^2\) failed to converge. As a matter of fact, divergence is indicated for this case.

For those who may be interested, the program was run using \(10^{-9}\) as the relative change test rather than \(10^{-15}\). In general, the number of iterations and time are nearly cut in half. At the same time the order of the residues is around \(10^{-8}\) or better.


IV. SUMMARY AND CONCLUSION

A. Summary

From the results in the tables in the previous chapter it appears that the Generalized Projection method is performing reasonably well. At this time it cannot be stated that it is the method to use in all cases. This can be said of all methods for solving systems of nonlinear equations. However, in all cases run, a rather good approximation to a solution (see cases 8, 9, and 10) if not a solution was found. Where convergence does not occur in the 501 permitted cycles one of the following two conditions will be satisfied:

1) The approximation vector causes the Jacobian matrix to be nearly singular.

2) A type of "ill-conditioning" occurs for the system. This is when the residue vector is nearly orthogonal to all the columns of the Jacobian matrix. This case was not expected to occur very often but in the test cases the condition presented itself three times – in test cases 2, 9, and 10.

The restriction of the nonsingularity of the Jacobian matrix on the level set in Theorem 5, Chapter II, can be relaxed a little. The thing of importance here is the nonsingularity of the Jacobian matrix on the set of approximation vectors generated by the Generalized Projection method. This set is a subset of the level set.

One other point should be emphasized with regard to the convergence proof. The proof yields sufficient conditions for convergence. Nonsingularity of the Jacobian matrix is not necessary for convergence.
As an example the following linear system was "solved" by the Projection method:

\[ \begin{align*}
X_1 + X_2 + X_3 &= 0 \\
X_1 - X_2 + 2X_3 &= 0 \\
3X_1 + X_2 + 4X_3 &= 0
\end{align*} \]

The determinant of the Jacobian matrix

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

of this system is zero and hence singular. The system has an infinite number of solutions. The Projection method yielded the solution vector \((-1.64516129, 0.54838709, 1.09677419)\) for the initial vector \((1, 1, 1)\). The corresponding residue vector was of the order \((10^{-15}, 10^{-15}, 10^{-15})\).

What is required for each cycle is that the residue vector is not orthogonal to all columns of the Jacobian matrix. If this occurs for at least one column per cycle then a reduction in the norm of the residue vector will occur and the process may continue.

Although most of the results to date are satisfying, there are several points of interest that should be pursued. These fall into the category of future research and hopefully will improve the method presented in this work.

B. Future Research

The most immediate goal that could be worked upon is that of accelerating the Generalized Projection method. It is suspected that
this will be a more difficult task than it was for the Linear Projection method. This is due mainly to the fact that the columns of the Jacobian matrix vary at every step and no obvious pattern as to how the changes take place at each step has been noticed. Perhaps Shen's work will be useful, at least for mildly nonlinear systems. A further point to consider here is: can an accelerated version be obtained which will not eliminate the time advantage per cycle that the Generalized Projection method has over the other methods presented?

A second item that can be examined is the effect of replacing the evaluation of the explicit partial derivatives with difference approximations to the partials. Also, it is possible to use relation (II-4) to obtain the new residue vector at each step rather than evaluate the functional equations. The area of concern here is whether the non-increasing norm of the residue vector is maintained at each step, and whether convergence can be guaranteed under these conditions.

Finally, one more area appears quite interesting. What about implementing this method in a conversational mode? There is a strong preference to using a CRT facility that will allow the user to interrupt the program at any time, to enter a new prediction to the solution, which may seem obvious to the user from the calculated approximations, which could be displayed upon request. Hopefully the user would have a minimal amount of work to do in using this implementation. It is hoped that the partial derivative calculations can be built into the program

---

either by the use of difference approximations or some differentiation scheme that could be called upon when needed.

C. Conclusion

A method which appears to be quite useful in the area of solving systems of nonlinear equations has been presented. In doing so a drop has been added to the bucket which will hold results for nonlinear equations in their unmodified forms. Nonlinear theory is still in a juvenile state, is still largely fragmentary. It is hoped, however, that the bucket may soon be filled with many such drops as the one added by this paper. One must be aware that every method added to the bucket will have certain restrictions on the system that can be solved and it is the mathematician that is impelled to round out the theory which invokes restrictive hypotheses.
V. BIBLIOGRAPHY


VI. ACKNOWLEDGEMENTS

The author is greatly indebted to his advisor, Dr. Roy F. Keller, for his advice and guidance during my graduate program at Iowa State University and especially during the development of this work.

I would also like to express my gratitude to my wife, Evelyn, and our children for their patience, encouragement, and prayers while I was struggling through the development and writing of this thesis.
A. Description of the Implemented Program

The Generalized Projection method program implemented at this time was run on an IBM 360, Model 65 system. It was programmed in FORTRAN using double precision accuracy for all variables. The main body of the program, that is, all but the subroutines to determine the components of the residue vector and the components of the columns of the Jacobian matrix, was executed in object deck form. The object deck was obtained via the procedure FORTGC.

The maximum system size currently permitted is 30 equations with 30 unknowns. This bound is easily changed by making the appropriate changes in the DIMENSION statements used in the program. The first step in the program is to input the size of the system to be solved and the initial approximation vector. Calculations to determine changes in the components of the approximation vector along with the residue vector that results for each change are performed. At the end of each cycle a check for convergence is made.

The test for convergence proceeds as follows. The relative change in each component of the change vector, CHNG, for the current cycle is examined to see if all changes are less than $10^{-15}$. If not, convergence has not been completed and a new cycle is started. Should the test be affirmative, then other conditions must be considered. The residue vector associated with the new approximation vector is examined to see whether it lies in a neighborhood of radius $10^{-10}$ of the zero vector. If it does, then the method has converged to a solution. If not, the
Jacobian matrix is nearly singular or the system is what shall be
called, for lack of a better term, "ill-conditioned." In this case a
message is printed indicating this fact.

A maximum of 501 iterations is permitted with this implementation.
Should this bound be reached the program stops executing and prints the
approximation vector to that point, along with the associated residue
vector. This will give the user an idea as to how close to a solution
the method has come.

Two subroutines are used in the program. One, called RESID,
calculates the residue vector for a given vector \( x \), passed as an argu­
ment. The second subroutine, PARDEE, determines a column vector of
the Jacobian matrix for the current value of \( x \), given as an argument.
The particular column to be computed is indicated by the second argument
passed to the subroutine. The actual partial derivatives are used in
this subroutine.

A listing of the main program and, since the subroutines will vary
with each system of equations to be solved, an example of each subroutine
may be found in the Appendix.

The following is a list of the variables used in the program:

- **ANUM** - the result of the inner product \( (R(x^{(k,i)}),DF(x^{(k,i)})) \).
- **CHNG** - a vector containing the amount of change in each component for
  the current cycle.
- **DENOM** - the result of the inner product \( (DF(x^{(k,i)}),DF(x^{(k,i)})) \).
- **DF** - a vector which contains the current column of the Jacobian
  matrix. It is returned from the subroutine PARDEER.
FUDGE - a parameter used in the ALTIME subroutine. It gives the actual time the program uses the CPU, in seconds.

N - the order of the system to be solved.

PARDER - a subroutine to calculate a column of the Jacobian matrix for a given approximation vector $x$.

R - a vector containing the residue for each function in the system being solved. It is returned from the subroutine RESID.

RESID - a subroutine to calculate the residue vector for a given approximation vector.

T - a parameter used in the ALTIME subroutine. It gives the time the job was in the system, in seconds.

X - the approximation vector.
B. Listing of Implemented Program with Sample Subroutines

//B282API  JOB 'I4067,TIME=1,REGION=96K\',SANDY
//STEP1  EXEC FORTGC,PARM,FORT='DECK'
//FORT,SYSIN DD *
C******************************************************************************
C PROGRAM TO SOLVE NONLINEAR SYSTEMS OF EQUATIONS BY A *
C PROJECTION METHOD *
C A.MAC EACHERN JULY 28 1969 *
C *
C REVISED JULY 1970. *
C IN THE REVISION THE CONVERGENCE TEST WAS CHANGED *
C TO MAKE IT COMPATABLE WITH BROWN'S METHOD FOR *
C COMPARISON PURPOSES *
C *
C******************************************************************************
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION X(30),R(30),DF(30),CHNG(30)
C******************************************************************************
C INPUT FORMAT *
C *
C CARD 1 *
C NO. OF VARIABLES IN COLUMNS 1-3 (RIGHT JUSTIFIED) *
C INITIAL APPROXIMATION VECTOR STARTS IN COL. 6. *
C THE ELEMENTS ARE READ UNDER F10.5 FORMAT *
C WITH A MAXIMUM OF 7 IN THE FIRST CARD. *
C CARD 2(IF NECESSARY) *
C REMAINING VARIABLES ARE READ UNDER F10.5 *
C FORMAT. *
C******************************************************************************
77 READ(1,999,END=78) N,(X(I),1=1,N)
999 FORMAT(I3,2X,7F10.5,/(8F10.5))
WRITE(3,998) (X(I),I=1,N)
998 FORMAT('1',*INITIAL APPROXIMATION=*,10(F6.2,2X))
C******************************************************************************
C TURN ON THE TIMER *
C T WILL GIVE THE REAL TIME FROM THIS POINT *
C IN THE JOB UNTIL THE RESULTS ARE PRINTED *
C FUDGE WILL PROVIDE THE TASK TIME FOR THE SAME *
C SECTION OF CODE *
C******************************************************************************
T=0.0
FUDG=0.0
CALL STARTM(T,FUDG)
C******************************************************************************
C BEGINNING OF COMPUTATION SECTION *
C FOR EACH CYCLE THE CURRENT APPROXIMATION *
C VECTOR IS_WRITTEN. *
C******************************************************************************
DO 1 K=1,501
DO 2 I=1,N

C***********************************************************
C OBTAIN THE RESIDUE VECTOR AND THE REQUIRED COLUMN OF THE JACOBIAN FOR THE CURRENT X VECTOR
C***********************************************************
CALL RESID(N,X,R)
CALL PARDER(X,I,DF)
C***********************************************************
C DETERMINE THE CHANGE VALUE FOR THE COMPONENT BEING CHANGED.
C***********************************************************
ANUM=0.0D0
DNOM=0.0D0
DO 3 M=1,N 
ANUM=ANUM+R(M)*DF(M)
DNOM=DNOM+DF(M)*DF(M)
C***********************************************************
C UPDATE THE X VECTOR
C***********************************************************
DX=-(ANUM/DNOM)
CHNG(I)=DX
2 X(I)=X(I)+DX
C***********************************************************
C IS EACH COMPONENT IN THE CHANGE VECTOR LESS THAN 10**-15?
C***********************************************************
DO 33 I=1,N
IF(DABS(CHNG(I)/X(I))-1.0D-15) 33,33,12
33 CONTINUE
C***********************************************************
C COMPUTE THE FINAL RESIDUE
C***********************************************************
CALL RESID(N,X,R)
C***********************************************************
C ARE THE FINAL RESIDUE VALUES SMALL ENOUGH
C***********************************************************
DO 34 I=1,N
IF(DABS(R(I)).GT.1.0D-10) GO TO 7
34 CONTINUE
CONTINUE
GO TO 6
C****************************************************************************** *
C IS THE APPROXIMATION VECTOR TENDING TO THE ZERO VECTOR *
C****************************************************************************** *
12 DO 4 I=1,N
IF(DABS(X(I)).GE.1.0D-7) GO TO 11
4 CONTINUE
WRITE(3,333)
333 FORMAT('SOLUTION IS TENDING TO ZERO')
GO TO 7
C****************************************************************************** *
C PRINT THE APPROXIMATION VECTOR FOR THE CURRENT CYCLE *
C****************************************************************************** *
11 WRITE(3,777) K,(X(I),I=1,N)
777 FORMAT('THE JACOBIAN IS NEARLY SINGULAR OR ILL-CONDITIONED. TRY ANOTHER APPROXIMATION. THE RESULTS TO THIS POINT FOLLOW:')
2 CONTINUE
C****************************************************************************** *
C OUTPUT SECTION FOR GENERALIZED PROJECTION METHOD *
C****************************************************************************** *
7 WRITE(3,776)
776 FORMAT(' THE JACOBIAN IS NEARLY SINGULAR OR ILL-CONDITIONED. TRY ANOTHER APPROXIMATION. THE RESULTS TO THIS POINT FOLLOW:')
6 WRITE(3,996) (X(M),M=1,N)
996 FORMAT(' SOLUTION VECTOR ',/,'D24.16,2X')
WRITE(3,899) (R(M),M=1,N)
899 FORMAT(' FINAL RESIDUES ',/,'D24.16,2X')
C****************************************************************************** *
C TURN THE TIMER OFF AND END SUBROUTINE ALTIME *
C****************************************************************************** *
CALL STOPM(T,FUDG)
WRITE(3,55) T,FUDG
55 FORMAT('2E14.7')
GO TO 77
78 STOP
END
SUBROUTINE RESID(N,X,R)
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION X(30), R(30)

C***********************************************************
C THIS SUBROUTINE CALCULATES THE RESIDUE VECTOR
C USING THE GIVEN SYSTEM OF EQUATIONS.
C***********************************************************

R(1) = 20.0*X(1)-(DCOS(X(2))**2+X(3)-DSIN(X(3))-37.0
R(2) = DCOS(2.0*X(1))+20.0*X(2)+DLOG10(1.0+X(4)**2)+5.0
R(3) = DSIN(X(1)+X(2))=X(2)+DATAN(X(3))+19.0*X(3)-12.0
R(4) = 2.0*DTANH(X(2))+DEXP(-2.0*X(3)**2+0.5)+21.0*X(4)
RETURN
END

SUBROUTINE PARDER(X, I, DF)
IMPLICIT REAL*8 (A-H, O-Z)
DIMENSION DF(30), X(30)

C***********************************************************
C THIS ROUTINE CALCULATES THE REQUIRED COLUMN VECTORS
C OF THE JACOBIAN MATRIX.
C THE COMPUTED GO TO STATEMENT IS A FUNCTION OF
C THE SIZE OF THE SYSTEM OF EQUATIONS.
C STATEMENT NO. N DENOTES THE FIRST ENTRY IN THE
C N-TH COLUMN OF THE JACOBIAN MATRIX.
C***********************************************************

GO TO (1, 2, 3, 4), I
1 DF(1) = 20.0
  DF(2) = -2.0*DSIN(2.0*X(1))
  DF(3) = DCOS(X(1)+X(2))
  DF(4) = 2.0*DCOSH(X(2))
RETURN
2 DF(1) = 2.0*DCOS(X(2))*DSIN(X(2))
  DF(2) = 20.0
  DF(3) = DCOS(X(1)+X(2))-1.0
  DF(4) = 2.0*DCOSH(X(2))**2
RETURN
3 DF(1) = 1.0-DCOS(X(3))
  DF(2) = 0.0
  DF(3) = 19.0+1.0/(1.0+X(3)**2)
  DF(4) = -4.0*X(3)*DEXP(-2.0*X(3)**2+0.5)
RETURN
4 DF(1) = 0.0
  DF(2) = 2.0*X(4)/(1.0+X(4)**2)*.43429
  DF(3) = 0.0
  DF(4) = 21.0
RETURN
END