A COMPARISON OF SOME VECTOR ACCELERATION TECHNIQUES

By

MEI-HUI CHEN

Bachelor of Science

National Chengchi University

Taipei, Taiwan

Republic of China

1979

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements
for the Degree of
MASTER OF SCIENCE
May, 1984

A COMPARISON OF SOME VECTOR ACCELERATION TECHNIQUES

Thesis Approved:

J P Chandler
Thesis Adviser

R. E. Hedrick

Michael J. Folk

Norman N Durham
Dean of Graduate College

PREFACE

This paper discusses the Wynn's vector epsilon algorithm, the modified vector epsilon algorithm, the delta-squared process, and the low-degree generalized secant methods, four acceleration techniques that are used in speeding up some slowly convergent sequences or forcing the convergence of some divergent sequences. A comparison of these acceleration techniques is presented by solving some numerical examples.

I would like to express my deep appreciation to my major adviser, Dr. John P. Chandler, for his intelligent guidance and encouragement.

I am also thankful to the other committee members, Dr. G. E. Hedrick and Dr. Michael Folk, for their advisement and support.

Special thanks is due to Rosemary Fernandes for her kind help in correcting my grammar.

I am grateful to my parents, Mr. and Mrs. Chen for their encouragement and understanding.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The advent of high speed digital computers has caused considerable attention to be paid to the problem of large computations and high accuracy. However, the computer time of thousands of iterations required for some complicated problems may still demand several hours on a large mainframe computer. Use of acceleration techniques with a savings in computer time by a factor of 3 to 5 or more certainly is worthwhile, especially if one has a great number of problems to solve.

Let us focus our attention on the prototype problem

$$F(x) = 0 \tag{1.1}$$

where F is an operator on the n-vector x. For iterative solution, it is convenient to write the equation in the form

$$x = G(x) \tag{1.2}$$

so that any solution of (1.1) is a solution of (1.2) if x is a fixed point of G. G(x) is called an iteration function for solving (1.1). In fact, for the given starting point x_0 , we can calculate successively x_1, x_2, ..., x_n by the basic iteration

$$x_{i+1} = G(x_i) \qquad i=1,\ 2,\ \ldots\ ,\ n. \tag{1.3}$$

The vector sequence x_1, x_2, ..., x_n is assumed to diverge or to converge slowly in this paper, because these are the sequences that are in need of acceleration.

There are several acceleration techniques available for speeding up the convergence of an iterative solution. The methods considered in this paper are the vector epsilon algorithm [35] [36], the modified vector epsilon algorithm [9], the delta-squared process [7] [17], and Anderson's low-degree generalized secant methods [2]. All four methods were tested using the four numerical examples described in chapter 6. The epsilon algorithm [33], which is closely related to the $e_m(S_n)$ transformation [26] and the Padé table [11] [14] [37], provides a powerful technique for transforming slowly convergent or divergent scalar sequences [27] [28] [33] [34]. The vector epsilon algorithm, extended by Wynn [35] from the scalar epsilon algorithm, has been tested and proved by Brezinski [4] [5] [6], Gekeler [13] and McLeod [21]. The delta-squared process, developed by Aitken [1], has been applied to accelerate scalar sequences. Wilkinson [31] points out that it is difficult to develop an efficient program for Aitken's delta-squared process. Modifications have been made by Jennings [17] and Boyle [3] to Aitken's delta-squared process for accelerating the convergence of iterative processes involving matrices.

The next four chapters describe the details of the four acceleration techniques noted above. Chapter 6 considers four numerical examples from the applied literature: a degenerate linear problem [2], nonlinear eigenvalue problems [2] [16], nonlinear integral equations [2] [25], and a boundary value problem [18]. A summary of the results is given in Chapter 7. All of the computations reported here have been done in standard A.N.S.I. FORTRAN 66 using double precision on an IBM 3081D.

CHAPTER II

VECTOR EPSILON ALGORITHM METHODS

The family of nonlinear transformations designated as e_k , e_k^m, $\tilde{e}_k$, and e_d has been discussed by Shanks [26]. Let us now consider e_k^m first.

Many iterations converge slowly but approximately linearly or geometrically:

sequence $x_0, x_1, x_2, \ldots, x_n$ with $n \to \infty$,

$$\lim_{n\to\infty} x_n = \alpha$$

and

$$x_n - \alpha \simeq C(x_{n-1} - \alpha), \tag{2.1}$$

where C is a constant.

For n + 1 iterates we can say

$$x_{n+1} - \alpha \simeq C(x_n - \alpha). \tag{2.2}$$

If we eliminate C between the equation (2.1) and the equation (2.2) and solve for α , we find

$$\alpha \simeq \frac{x_{n+1} x_{n-1} - x_n^2}{x_{n+1} + x_{n-1} - 2x_n} . \tag{2.3}$$

In practice, we don't know whether any particular n is large enough to make a good approximation to α. But we can define a new sequence by [29]

$$y_{n+1} = \frac{x_{n+1} x_{n-1} - x_n^2}{x_{n+1} + x_{n-1} - 2x_n}$$

or

$$y_{n+1} = x_{n+1} - \frac{(x_{n+1} - x_n)^2}{(x_{n+1} - x_n) - (x_n - x_{n-1})}, \qquad (2.4)$$

or

$$y_{n+1} = x_{n-1} - \frac{(x_n - x_{n-1})^2}{(x_{n+1} - x_n) - (x_n - x_{n-1})}. \qquad (2.5)$$

This new sequence may converges faster to α than does the original sequence x_0, x_1, ... In the same way we can form a second derived sequence

$$z_{n+1} = \frac{y_{n+1} y_{n-1} - y_n^2}{y_{n+1} + y_{n-1} - 2y_n},$$

and higher order derived sequences. The equation (2.4) and the equation (2.5) are best for convergent sequences and divergent sequences respectively, in the sense that subtractive cancellation is minimized.

Let us now illustrate these derived sequences y and z by considering the following example.

Example 2.1: Slowly convergent series

The function f(x) = ln x has the Taylor expansion

$$\ln x = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \ldots\ldots - \frac{(-1)^n (x-1)^n}{n} + \frac{(-1)^n (x-1)^{n+1} \xi^{-(n+1)}}{n+1}$$

where 0 < x <= 2, and ξ is between 1 and x.

Table I shows the results of the original sequence and trandformed sequences where x = 2; i.e., ln 2 = 0.69314.... x_{10} is correct to only one figure, but z_{10} is already correct to five figures.

TABLE I

COMPARISON OF ORIGINAL SLOWLY CONVERGENT SEQUENCE X AND TRANSFORMED SEQUENCES Y AND Z

n	x	y	z
o	0.00000		
1	1.00000		
2	0.50000	0.66667	
3	0.83333	0.69999	
4	0.58333	0.69047	0.69259
5	0.78333	0.69444	0.69327
6	0.61666	0.69242	0.69310
7	0.75952	0.69358	0.69316
8	0.63452	0.69285	0.69313
9	0.74563	0.69334	0.69315
10	0.64563	0.69300	0.69314

The transformed sequences y and z belong to e_1^m that are considered in the following.

Let $\{A_n\}$ (n = 0, 1, 2, ...) be a sequence of numbers or functions and let

$$\Delta A_n = A_{n+1} - A_n.$$

Let K be positive integer and let a new sequence $\{B_{k,n}\}$ (n = k, k+1, K+2, ...), "the Kth order transform of $\{A_n\}$", be defined by

$$B_{k,n} = \frac{\begin{vmatrix} A_{n-k} & A_{n-k+1} & \cdots & A_{n-1} & A_n \\ \Delta A_{n-k} & \Delta A_{n-k+1} & \cdots & \Delta A_{n-1} & \Delta A_n \\ \vdots & \vdots & & \vdots & \vdots \\ \Delta A_{n-1} & \Delta A_n & \cdots & \Delta A_{n+k-2} & \Delta A_{n+k-1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 & 1 \\ \Delta A_{n-k} & \Delta A_{n-k+1} & \cdots & \Delta A_{n-1} & \Delta A_n \\ \vdots & \vdots & & \vdots & \vdots \\ \Delta A_{n-1} & \Delta A_n & \cdots & \Delta A_{n+k-2} & \Delta A_{n+k-1} \end{vmatrix}} = e_k(A_n) \qquad (2.6)$$

for each n for which the denominator does not vanish.

Immediately, we will have

$$B_{0,n} = A_n \qquad (n = 0, 1, 2, \ldots)$$

The transform may be written in operator form, thus:

$$B_{k,n} = e_k(A_n)$$

where e_k is the nonlinear operator defined by the right hand side of the equation (2.6).

The e_1^m transform is the first iteration transform, and the following iteration transforms can be defined by

$$C_{k,n} = e_k(B_{k,n}) = e_k^2(A_n) \qquad ,(n >= 2k)$$

$$D_{k,n} = e_k(C_{k,n}) = e_k^2(B_{k,n}) = e_k^3(A_n) \qquad ,(n >= 3k)$$

and so on.

For k = 1, these derived sequences are

$$B_{1,n} = \frac{\begin{vmatrix} A_{n-1} & A_n \\ \Delta A_{n-1} & \Delta A_n \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \Delta A_{n-1} & \Delta A_n \end{vmatrix}} = \frac{A_{n+1}A_{n-1} - A_n^2}{A_{n+1} + A_{n-1} - 2A_n} = e_1^2(A_n), \qquad (2.7)$$

$$C_{1,n} = \frac{\begin{vmatrix} B_{n-1} & B_n \\ \Delta B_{n-1} & \Delta B_n \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \Delta B_{n-1} & \Delta B_n \end{vmatrix}} = \frac{B_{n+1}B_{n-1} - B_n^2}{B_{n+1} + B_{n-1} - 2B_n} = e_1^2(A_n),$$

and so on.

From example 2.1 we know that e_1^m is an accelerator of convergence. The process, using e_1^m for speeding up the convergence of iterative processes, usually is called scalar Aitken's delta-squared process which can be expressed in the form

$$A^* = A_{n+1} - \frac{(\Delta A_n^2)}{\Delta^2 A_{n-1}} \tag{2.8}$$

where $\Delta^2 A_{n-1} = \Delta(\Delta A_{n-1}) = \Delta A_n - \Delta A_{n-1}$.

The right hand side of the equation (2.8) can be written

$$A_{n+1} - \frac{A_{n+1}^2 - 2A_{n+1}A_n + A_n^2}{A_{n+1} - A_n - A_n + A_{n-1}} = \frac{A_{n+1}A_{n-1} - A_n^2}{A_{n+1} + A_{n-1} - 2A_n}$$

which is equivalent to the equation (2.4) and the equation (2.7).

In general e_1^m is a good accelerator of convergence. But e_1^m may converge to a wrong answer in some special series or suffer cancellation error in the denominator; the reader is referred to Olver [23] and Shanks [26]. The following example illustrates that e_K often has better convergence than e_1^m does.

Before we go to the example 2.2, we need to consider how to carry the equation (2.6) into a computer efficiently.

<u>Theorem 2.1:</u> If

$$\epsilon_{-1}(A_n) = 0 \qquad ,(n = 0, 1, \ldots)$$

$$\epsilon_{2K}(A_n) = e_K(A_n) \qquad (n, k = 0, 1, \ldots)$$

and

$$\epsilon_{2k+1}(A_n) = \frac{1}{e_k(\Delta A_n)} \qquad (n,\ k = 0,\ 1,\ \ldots)$$

then

$$\epsilon_{s+1}(A_n) = \epsilon_{s-1}(A_{n+1}) + \frac{1}{\epsilon_s(A_{n+1}) - \epsilon_s(A_n)}. \qquad (n,\ s = 0,\ 1,\ \ldots) \qquad (2.9)$$

provided that none of the quantities $\epsilon_{2k}(A_n)$ becomes infinite. This theorem has been proved by Wynn [33].

The quantities $\epsilon_s^{(m)}$ may be arranged in the following array:

$$\begin{array}{cccccc}
\epsilon_{-1}^{(0)} & & & & & \\
 & \epsilon_{0}^{(0)} & & & & \\
\epsilon_{-1}^{(1)} & & \epsilon_{1}^{(0)} & & & \\
 & \epsilon_{0}^{(1)} & & \ddots & & \\
\epsilon_{-1}^{(2)} & & \epsilon_{1}^{(1)} & & \epsilon_{s}^{(0)} & \\
 & \epsilon_{0}^{(2)} & & \ddots & & \epsilon_{s+1}^{(0)} \\
\epsilon_{-1}^{(3)} & \vdots & \epsilon_{1}^{(2)} & & \epsilon_{s}^{(1)} & \vdots \\
\vdots & & \vdots & & \vdots &
\end{array}$$

The superscript (n) denotes a diagonal and the subscript s denotes a column. The even-numbered columns $\epsilon_{2k}(A_n)$ display the transformed sequences $e_k(A_n)$, and by hypothesis, the transformed sequences converge more rapidly than the original sequence A_n. The odd-numbered columns $\epsilon_{2K+1}(A_n)$ are intermedia; there is no need for the explicit evaluation of $\epsilon_{2K+1}(A_n)$.

The four quantities in the equation (2.9) can be arranged in a lozenge,

$$\begin{array}{ccc} & \epsilon_s^{(n)} & \\ \epsilon_{s-1}^{(n+1)} & & \epsilon_{s+1}^{(n)} \\ & \epsilon_s^{(n+1)} & \end{array}$$

Then the right-most member is obtained by adding the left-most member to the inverse of the difference of the two in the middle.

<u>Example 2.2</u>: Power series

The function $f(x) = (x-1)^{-2}$ has the Taylor expansion

$$f(x) = 1 + \frac{-2}{1!}(-x) + \frac{(-2)(-3)}{2!}(-x)^2 + \frac{(-2)(-3)(-4)}{3!}(-x)^3 + \dots$$

When $x = 2$ then $f(2) = 1$ and $f(2) = 1 + 2*2 + 3*2^2 + 4*2^3 + 5*2^4 + \dots$

Table II shows the results of the original sequence and transformed sequences including e_1^m and e_K^1.

TABLE II

COMPARISON OF ORIGINAL SEQUENCE AND TRANSFORMED SEQUENCES TO A POWER SERIES

		Scalar Aitken's delta-squared process			Scalar epsilon algorithm			
n	A_n	e_1^1	e_1^2	e_1^3	ϵ_1	$\epsilon_2(e_1^1)$	ϵ_3	$\epsilon_4(e_2^1)$
0	1							
1	5				.25000			
2	17	-1.00000			.08333	-1.00000		
3	49	-2.20000			.03125	-2.20000	-.75000	
4	129	-4.33333	.54285		.01250	-4.33333	-.43750	1.00
5	321	-8.14285	.51515		.00521	-8.14285	-.25000	1.00
6	769	-15.00000	.42857	.55589	.00232	-15.00000	-.14062	1.00
7	1793	-27.44445	.27279	.62401	.00097	-27.44445	-.07812	1.00
8	4097	-50.20000	.01918	.67714	.00043	-50.20000	-.04297	1.00

In this special case, the columns of e_1^m don't have the correct answer and the diagonal of e_1^m converges slowly. But e_k^1 has the rapid and correct convergence.

The following algorithm by Wynn [35] extends the scalar epsilon algorithm to the vector epsilon algorithm.

Algorithm 2.1: Vector epsilon algorithm [35]

Let $A_n = G(A_{n-1})$.

Given the initial conditions

$$\epsilon_{-1}^{(n)} = 0 \qquad (n = 1, 2, \ldots)$$

and

$$\epsilon_{0}^{(n)} = A_n, \qquad (n = 0, 1, \ldots)$$

compute

$$\epsilon_{s+1}^{(n)} = \epsilon_{s-1}^{(n+1)} + \{ \epsilon_{s}^{(n+1)} - \epsilon_{s}^{(n)} \}^{-1}. \qquad (n,\ s= 0, 1, \ldots)$$

Test if the distance between $\epsilon_{2m}^{(n)}$ and $\epsilon_{2m}^{(n+1)}$ satisfied.

The inverse of the vector $\{ \epsilon_{s}^{(n+1)} - \epsilon_{s}^{(n)} \}$ can be defined in either of two ways (or in other ways).

(1) Primitive Inverse:

Regarding each component independently, this is equivalent to the simultaneous application of the scalar epsilon algorithm to components of the vector. The inverse of a vector $x = (x_1, x_2, \ldots, x_n)$ is taken to be

$$x^{-1} = (x_1^{-1}, x_2^{-1}, \ldots, x_n^{-1})$$

(2) The Samelson Inverse of a Vector:

This inverse is suggested by K. Samelson. The inverse of a vector $x = (x_1, x_2, \ldots, x_n)$ is taken to be

$$x^{-1} = (\sum_{i=1}^{n} x_i \bar{x}_i)^{-1} (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$$

where $\bar{x}_i$ is the complex conjugate of x_i. If the components of the vector are all real, the Samelson inverse is computed by

$$x^{-1} = (\sum_{i=1}^{n} x_i)^{-1} x.$$

The vector epsilon algorithm has been tested on the solutions of linear equations which are ill-conditioned matrices, on nonlinear integral equations, and on boundary value problems and initial value problems by Wynn [35] [36], Gekeler [13], and Brezinski and Rieu [4] [5] [6]. Generally speaking, all of these test examples have rapidly convergent sequences. But the vector epsilon algorithm can't overcome the divergence of one of nonlinear integral equations in Anderson's paper [2].

CHAPTER III

DYNAMIC DELTA-SQUARED PROCESS

The delta-squared process, one of the nonlinear transformations, has been applied to speeding the convergence of the vectors by Aitken [1]. Chapter II indicates that of drawbacks of e_1^m is cancellation that occurs in the denominator of the equation (2.3). Modifications of vector Aitken's delta-squared process derived by Jennings [17] allow the denominator to be chosen to avoid the cancellation.

Let x_{n-1}, x_n, x_{n+1} be three successive iteration vectors. The equation (2.3), Aitken's prediction, can be written in the form

$$x^* = x_{n+1} - S^*(x_{n+1} - x_n)$$

where

$$s^* = \frac{x_{n+1} - x_n}{(x_{n+1} - x_n) - (x_n - x_{n-1})}.$$

If w is a vector defining the relative weights then the predicted vector is

$$x^* = x_{n+1} - S^*(x_{n+1} - x_n) \tag{3.1}$$

where

$$s^* = \frac{w^T(x_{n+1} - x_n)}{w^T\{(x_{n+1} - x_n) - (x_n - x_{n-1})\}}.$$

First difference modulation (FDM), w is taken as $(x_n - x_{n-1})$ when the convergence is governed by a symmetric iteration matrix. In this case when convergence has already been obtained, the denominator can be vanished. Second difference modulation (SDM) may be used by setting w to $(x_{n+1} - x_n) - (x_n - x_{n-1})$ in unsymmetric iteration matrices. In this case the denominator is the square of the Euclidean norm of $(x_{n+1} - x_n) - (x_n - x_{n-1})$ so cancellation cannot occur in it.

If x^* is a better result than x_{n+1}, then it certainly is wasteful to continue computing x_{n+2}, x_{n+3}, ..., and so on. A better method is to use x^* as the next initial guess. This leads to the following algorithm.

Algorithm 3.1: Dynamic delta-squared process

Given: the initial vector y, iteration equations F(x) and w chosen to be $x_n - x_{n-1}$ or $(x_{n+1} - x_n) - (x_n - x_{n-1})$.

Step 1. $x_0 = y$.

Step 2. $x_1 = F(x_0)$, $x_2 = F(x_1)$, $y = x_2 - s(x_2 - x_1)$

$$\text{where } s = \frac{w^T(x_2 - x_1)}{w^T\{(x_2 - x_1) - (x_1 - x_0)\}}.$$

Step 3. If $x_2 - y$ is small enough then stop; otherwise go to step 1.

This algorithm is similar to Steffensen's iteration [24] in which transformed iterates are dynamically fed back into the iterative process.

Whenever the derived sequence diverges, the reverse dynamic delta-squared process gives the desired results.

This requires the interchange of the values of x_{n+1} and x_{n-1}, and still use the equation (3.1) to compute x*.

The vector Aitken's delta-squared process has been applied successfully to eigenvalue problems, the solution of linear simultaneous equations, and elastic-plastic stress analysis by Jennings and Boyle [3] [17]. It also can cope with the divergence of test examples in Anderson's paper [2].

CHAPTER IV

LOW-DEGREE GENERALIZED SECANT METHODS

The orders of convergence of the delta-squared process and the secant method are $\sqrt{2}$ [20] and 1.618... [10] respectively. In view of the order of convergence, it is desired to generalize the secant method to accelerate the iterative sequence.

The low-degree generalized secant methods are derived from the univariate secant method by Anderson [2]. A similar device for the linear case is due to Khabaza [19]. The univariate secant method, with a secant line through two sample points x_{i-1} and x_i, is a two-point iterative method. The formula is

$$x_{i+1} = \frac{F(x_i)x_{i-1} - F(x_{i-1})x_i}{F(x_i) - F(x_{i-1})}$$

or

$$x_{i+1} = x_i - F(x_i)\frac{x_i - x_{i-1}}{F(x_i) - F(x_{i-1})}$$

for a single function of one variable.

Wolfe [32] generalizes the univariate secant method to a set of n secant hyperlines through n + 1 sample points for the simultaneous solution of a system of nonlinear equations. Instead of using n secant hyperlines, Anderson considers a hyperline through two sample points. This hyperline does not in general intersect the subspace defining the solution but is in some sense closest to the subspace. Derivation of low-degree generalized secant methods start with equation (1.3)

$$x_{i+1} = G(x_i).$$

If y_i and z_i are a coupled pair of iterative sequences related by

$$z_i = G(y_i). \qquad (4.1)$$

The question is how does one define y_{i+1} as a function of z_i, y_i, z_{i-1} and y_{i-1} so that the sequences y_i and z_i converge more rapidly than the basic sequence x_i. Anderson defines y_{i+1} as a linear, parameterized form in previous iterates, defines a quadratic residual, and obtains the free parameters by minimizing the residual. The residual vector r_i is defined by

$$r_i = z_i - y_i. \qquad (4.2)$$

Before we go to the next section, we need to define the inner product of two n-vectors. The inner product of two n-vectors u and v is defined by

$$(u, v) = \sum_{i=1}^{n} u_i v_i w_i.$$

where the weights w_i are positive.

Algorithm 4.1: Extrapolation algorithm [2]

Given: the initial vector y_0 and the iteration equations G(y).

```
Calculate $y_1 = z_0 = G(y_0)$, $r_0 = z_0 - y_0$.
For i = 1, 2, ..., until satisfied, do:
Calculate $z_i = G(y_i)$, $r_i = z_i - y_i$.
Calculate $q_i = (r_i, r_i - r_{i-1}) / (r_i - r_{i-1}, r_i - r_{i-1})$,
          $u_i = y_i + q_i(y_{i-1} - y_i)$,
          $v_i = z_i + q_i(z_{i-1} - z_i)$.
Calculate $y_{i+1} = u_i + b_i(v_i - u_i)$ with $b_i > 0$.     (4.3)
```

Usually the choice $b_i = 1$ is most appropriate. The restriction $b_i > 0$ is to prevent the new iterate y_{i+1} from becoming trapped in the subspace spanned by the previous y_i. The next algorithm, the relaxation algorithm, is identical to the extrapolation algorithm when applied in dynamic fashion, just as Steffenson's iteration is a dynamic Aitken's delta-squared process.

Algorithm 4.2: Relaxation algorithm [2]

Given: the initial vector y_0, and the iteration equation G(y).

```
Calculate $y_1 = z_0 = G(y_0)$, $r_0 = z_0 - y_0$.
For i = 1, 2, ..., until satisfied, do:
Calculate $z_i = G(y_i)$, $r_i = z_i - y_i$.
Calculate $q_i = (z_i - z_{i-1}, r_i - r_{i-1})/(r_i - r_{i-1}, r_i - r_{i-1})$.
Calacuate $y_{i+1} = z_i + q_i r_i$.     (4.4)
```

Equation (4.3) with $b_i = 1$, in the extrapolation algorithm and equation (4.4) in the relaxation algorithm are equivalent to equation (3.1) with $w = (x_{n-1} - x_n) - (x_n - x_{n-1})$ in the dynamic delta-squared process in the first iteration. When extrapolation algorithm and relaxation algorithm are applied in a dynamic fashion, the equation (4.3) and (4.4) can be written as

$$y^* = y_2 + \frac{(y_2 - y_1, y_2 - 2y_1 + y_0)}{(y_2 - 2y_1 + y_0, y_2 - 2y_1 + y_0)}(y_1 - y_2)$$

or

$$y^* = y_2 - \frac{(y_2 - y_1, y_2 - 2y_1 + y_0)}{(y_2 - 2y_1 + y_0, y_2 - 2y_1 + y_0)}(y_2 - y_1) \qquad (4.5)$$

When the weights of inner product w_i equal to 1, the equation (4.5) is equivalent to the equation (3.1).

There are two classes of variants that have been considered by Anderson. The first class concerns the degree of method. The extrapolation algorithm is a second degree method, obtained by minimizing the residual over the hyperline. Higher degree methods are obtained by minimizing over linear subspaces of higher dimension. The u_i, v_i and r_i vector then are defined by

$$u_i = y_i + \sum_{j=1}^{M} q_i^j (y_{i-j} - y_i),$$

$$v_i = z_i + \sum_{j=1}^{M} q_i^j (z_{i-j} - z_i),$$

and $\sum_{j=1}^{M} (r_i - r_{i-k}, r_i - r_{i-j}) q_i^j = (r_i, r_i - r_{i-k})$

for k = 1, 2, ..., M. In Anderson's experience, the low-degree methods are more useful, and it seems best to limit M to about 5.

The second class concerns the choice of the metric of the inner product. We can choose the unit metric, $w_i = 1$, or the residual metric. Usually, it is advantageous to define the residual metric by choosing w_i inversely proportional to some measure of the local size of the solution vector.

The devices of low-degree generalized secant methods have been applied to highly degenerate linear problems, nonlinear eigenvalue problem, and nonlinear integral equations by Anderson [2]. The vector epsilon algorithm, the delta-squared process and the basic iteration have been compared to the low-degree generalized secant methods on these three problems. The low-degree generalized secant methods have the best results in most problems.

CHAPTER V

MODIFIED VECTOR EPSILON ALGORITHM

One of the disadvantages of the vector epsilon algorithm is that transformed iterates are not fed back into the iterative process dynamically. Steffensen's iteration is the dynamic process of Aitken's delta-squared process. The following algorithm is the dynamic process of the vector epsilon algorithm in the same fashion. A similar algorithm is suggested by Cheng [8].

One of the basic properties of the vector epsilon algorithm is that the derived sequences $\epsilon_{2m}^{(n)}$ converge far more rapidly than the basic sequence A_n. Let us choose $\epsilon_{2m}^{(n)}$, with certain values of n and m, as the initial guess of next iteration. This leads to the following algorithm.

Algorithm 5.1: Modified vector epsilon algorithm

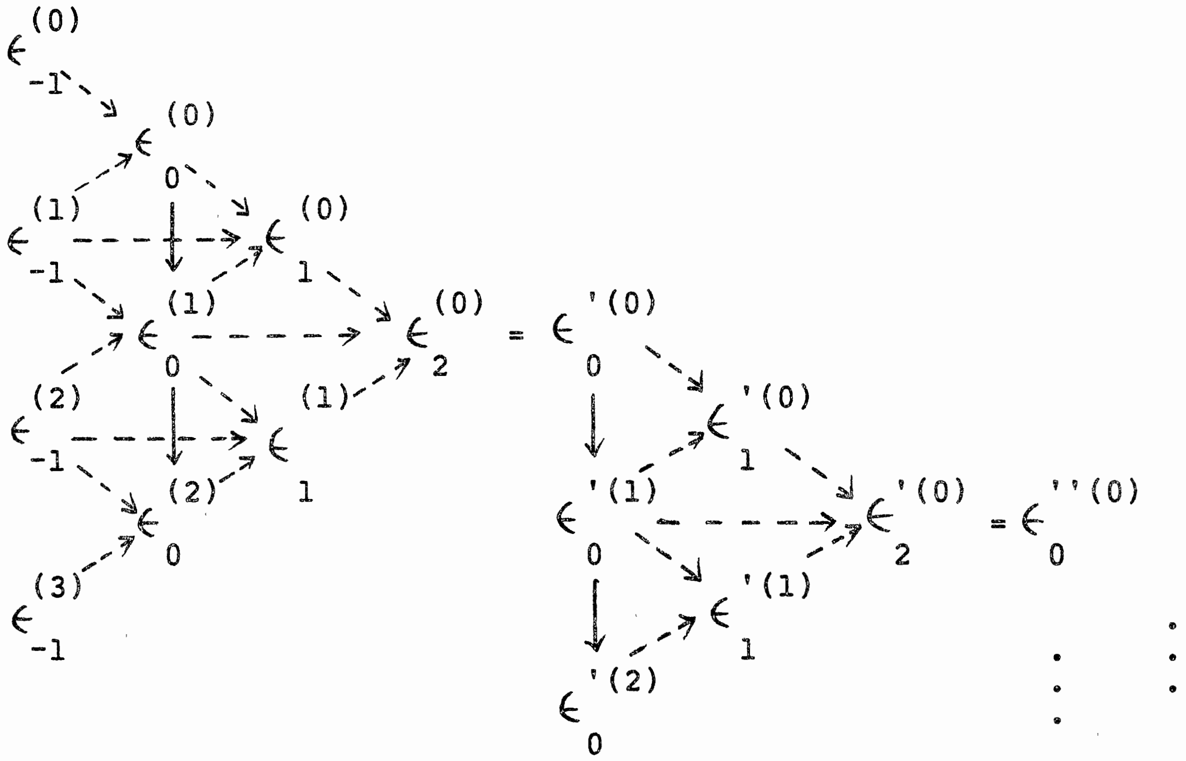
Given: the initial vector x_0, the iteration equations $G(x)$, and the order of transform k.

```
For i = 1, 2, ..., until satisfied, do:
    $\epsilon_{0}^{(0)} = x_0$
    For n = 1, 2, ..., 2k do:
        $\epsilon_{-1}^{(n)} = 0$
        $\epsilon_{0}^{(n)} = x_n = G(x_{n-1})$
    m = n
    For s = 0, 1, ..., n-1
        $\epsilon_{s+1}^{(m)} = \epsilon_{s-1}^{(m+1)} + \frac{1}{\epsilon_{s}^{(m+1)} - \epsilon_{s}^{(m)}}$
        m = m - 1
    $x_0 = \epsilon_{2k}^{(0)}$
```

2k+m+1 pieces of $\epsilon_s^{(m)}$ must be stored in this algorithm. k = 1 usually gives the best result. Usually, the following iterates of sequences are closer to the right answer than the first iterates. Using this hypothesis, it is possible to skip the first iterates, where $G(x_0)$ is $\epsilon_0^{(0)}$, in all computations or only in the first cycle (when i = 1). These two variants both require four pieces of $\epsilon_s^{(m)}$ to be stored.

When k is equal to 1, the quantities $\epsilon_s^{(m)}$ of the modified vector epsilon algorithm may be arranged in the following array:

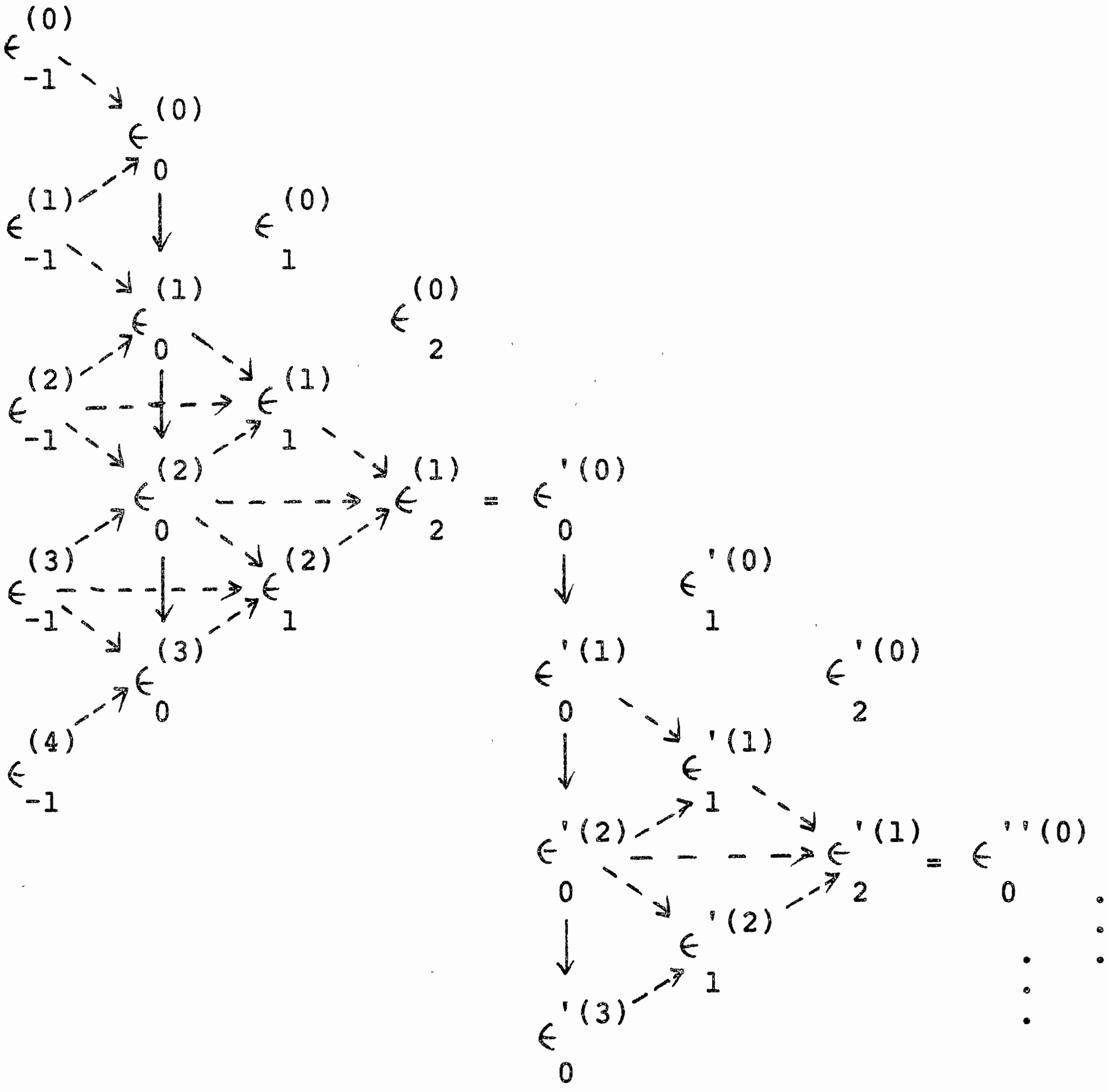
$$\epsilon_{-1}^{(m)} = 0 \qquad \epsilon_{0}^{(m)} = x_m \qquad \epsilon_{1}^{(m)} = \left[\epsilon_{0}^{(m+1)} - \epsilon_{0}^{(m)}\right]^{-1} \qquad \epsilon_{2}^{(m)} = \epsilon_{0}^{(m+1)} + \left[\epsilon_{1}^{(m+1)} - \epsilon_{1}^{(m)}\right]^{-1}$$



Downward arrows indicate generation of elements by iterations. Dash arrows indicate generation of elements by the equation (2.9). Equal signs indicate choosing $\epsilon_{2s}^{(m)}$ with s = 0, m = 0 to be the initial value of the next cycle.

The other two variants can also be described in a similar type of array. The first variant uses $\epsilon_{2s}^{(m)}$ with s = 0, m = 1 to be the initial value of the next cycle, and ignores the values of $\epsilon_{1}^{(0)}$ and $\epsilon_{2}^{(0)}$.

$$\epsilon_{-1}^{(m)} = 0 \qquad \epsilon_{0}^{(m)} = x_m \qquad \epsilon_{1}^{(m)} = \left[\epsilon_{0}^{(m+1)} - \epsilon_{0}^{(m)}\right]^{-1} \qquad \epsilon_{2}^{(m)} = \epsilon_{0}^{(m+1)} + \left[\epsilon_{1}^{(m+1)} - \epsilon_{1}^{(m)}\right]^{-1}$$



The second variant uses $\epsilon_{2s}^{(m)}$ with s = 0, m = 1 to be the initial value of second cycle, then uses $\epsilon_{2s}^{(m)}$ with s = 0, m = 0 in the other cycles.

$$\epsilon_{-1}^{(m)} = 0 \qquad \epsilon_{0}^{(m)} = x_m \qquad \epsilon_{1}^{(m)} = \left[\epsilon_0^{(m+1)} - \epsilon_0^{(m)}\right]^{-1} \qquad \epsilon_{2}^{(m)} = \epsilon_0^{(m+1)} + \left[\epsilon_1^{(m+1)} - \epsilon_1^{(m)}\right]^{-1}$$

$$\begin{tikzcd}
\epsilon_{-1}^{(0)} \arrow[dr, dashed] & & & & & & & & & & \\
& \epsilon_{0}^{(0)} \arrow[dd] & & & & & & & & & \\
\epsilon_{-1}^{(1)} \arrow[ur, dashed] \arrow[dr, dashed] & & \epsilon_{1}^{(0)} & & & & & & & & \\
& \epsilon_{0}^{(1)} \arrow[dd] \arrow[dr, dashed] & & \epsilon_{2}^{(0)} & & & & & & & \\
\epsilon_{-1}^{(2)} \arrow[ur, dashed] \arrow[dr, dashed] \arrow[rr, dashed] & & \epsilon_{1}^{(1)} \arrow[dr, dashed] & & & & & & & & \\
& \epsilon_{0}^{(2)} \arrow[ur, dashed] \arrow[rr, dashed] \arrow[dr, dashed] \arrow[dd] & & \epsilon_{2}^{(1)} & = & \epsilon_{0}^{'(0)} \arrow[dr, dashed] \arrow[dd] & & & & & \\
\epsilon_{-1}^{(3)} \arrow[ur, dashed] \arrow[rr, dashed] \arrow[dr, dashed] & & \epsilon_{1}^{(2)} \arrow[ur, dashed] & & & & \epsilon_{1}^{'(0)} \arrow[dr, dashed] & & & & \\
& \epsilon_{0}^{(3)} \arrow[ur, dashed] & & & & \epsilon_{0}^{'(1)} \arrow[ur, dashed] \arrow[rr, dashed] \arrow[dr, dashed] \arrow[dd] & & \epsilon_{2}^{'(0)} & = & \epsilon_{0}^{''(0)} & \vdots \\
\epsilon_{-1}^{(4)} \arrow[ur, dashed] & & & & & & \epsilon_{1}^{'(1)} \arrow[ur, dashed] & & & \vdots & \\
& & & & & \epsilon_{0}^{'(2)} \arrow[ur, dashed] & & & & &
\end{tikzcd}$$

In Cheng's [9] experience, the delta-squared process converges much more slowly than the cyclic iterative method does in most of his numerical examples, and the cyclic iterative method speeds up the convergence of the basic iteration.

CHAPTER VI

NUMERICAL EXAMPLES

Degenerate Linear Problem

Equation (6.1) presents a system of n equations [2]:

$$Ax = Db \tag{6.1}$$

with $A_{ij} = D \qquad (i = j)$

$\qquad = 1 \qquad (i \neg= j)$

where D, the diagonal matrix of A, is a free parameter and b is chosen such that the solution x has the form $x_i = 2 / i$. The equation (6.1) can be written as a Jacobi iteration [30]:

$$x_{n+1} = D^{-1}(U + L)x_n + b \tag{6.2}$$

with $A = D - U - L$.

where L and U are respectively strictly lower and upper triangular n * n matrices, whose entries are the negatives of the entries of A respectively below and above the main diagonal of A. Let

$H_{ij} = 0 \qquad (i = j)$

$\qquad = -1 / D \qquad (i \neg= j)$.

Equation (6.2) can be written as

$$x_{n+1} = Hx_n + b. \tag{6.3}$$

The matrix A is a highly degenerate matrix defined above. In this example the initial vector x_0 is taken to be

1. For n = 20, D = 15 yields a divergent basic iteration and n = 20, D = 25 yields a slowly convergent basic iteration. Table III contains the results of applying some acceleration techniques. The units in the following tables are counted by calculating each basic iteration

$$x_{n+1} = G(x_n).$$

TABLE III

COMPARISON OF NUMBER OF ITERATIONS TO SOLVE HIGHLY DEGENERATE LINEAR PROBLEM

Algorithm	D = 25	D = 15
Basic iteration	67	∞
Vector ϵ-algorithm[1]	7	7
Dynamic δ^2-process[2]	8	11
Extrapolation(M=1), unit metric	7	7
Extrapolation(M=1), residual metric	5	6
Relaxation, unit metric	6	6
Extrapolation(M=2), unit metric	4	4
Modified vector ϵ-algorithm(s=0,m=0)	13	17
Modified vector ϵ-algorithm(s=0,m=1)	10	10
Modified vector ϵ-algorithm(s=0,m=0,1)	8	8

1: ϵ-algorithm denotes the epsilon algorithm
2: δ^2-process denotes the delta-squared process

All of these acceleration techniques overcome the divergent basic iteration for D = 15, and speed up the slowly convergent basic iteration for D = 25 in this linear problem.

Eigenvalue Problem

Let λ be an eigenvalue of the same matrix A as in example 1 and

$$Az = \lambda z.$$

The n-vector x then is called an eigenvector of A belonging to the eigenvalue. By using the power method, one of the iterative schemes for solving the eigenvalue problem, the eigenvector iterates are [16]

$$z_{i+1} = Az_i$$

and the eigenvalue iterates are

$$\lambda_{i+1} = (z_{i+1}, z_{i+1}) / (z_{i+1}, z_i).$$

For the purpose of applying the power method to these acceleration techniques, a nonlinear operator α together with a coupled pair of iterative sequences are defined by

$$y_i = \alpha x_i = \{\lambda_{i+1}\}^{-1} Ax_i = \left\{\frac{(Ax_i, Ax_i)}{(Ax_i, x_i)}\right\}^{-1} Ax_i.$$

The iterates are normalized and define a basic iteration by

$$x_{i+1} = y_i / (y_i, y_i)^{1/2}.$$

The initial vector is y_i^{-1} and each $y_i = 2 / i$, the free parameter D = 101 and n = 20. The first two largest eigenvalue are $\lambda_1 = 120$, $\lambda_2 = 100$ of the matrix A.

Next we consider another example matrix B where

$$B = \begin{vmatrix} .5d-5 & .8d-5 & .0d0 & .2d-5 & .6d-5 & .1d-5 \\ .7d-5 & .49999d-5 & .1d-5 & .0d0 & .6d-5 & .2d-5 \\ .0d0 & .0d0 & .499d-5 & .4d-5 & .3d-5 & .3d-5 \\ .0d0 & .4d-5 & .2d-5 & .492d-5 & .1d-5 & .0d0 \\ .2d-5 & .3d-5 & .0d0 & .1d0 & .491d-5 & .45d-5 \\ .0d0 & .0d0 & .0d0 & .5d-5 & .0d0 & .49d-5 \end{vmatrix}$$

Matrix B has the eigenvalues $\lambda_1 = .5d-5$ and $\lambda_2 = .49999d-5$, since $B - \lambda I = (.5d-5 - \lambda)*(.49999d-5 - \lambda)*(.499d-5 - \lambda)*$
$(.492d-5 - \lambda)*(.491d-5 - \lambda)*(.49d-5 - \lambda)$.
Using the power method, the basic iteration is a slowly convergent sequence because the absolute values of the two largest eigenvalues are very close. The results are shown on table IV.

These numerical examples are degenerate problems, too. The dynamic delta-squared process accelerates convergence of the matrix A, but does not provide a corresponding acceleration in the convergence of the matrix B. The low-degree generalized secant methods give rapidly convergent sequences for these matrices. The vector epsilon algorithm and the modifications of the vector epsilon algorithm speed up the convergence of the matrix A and the matrix B, but are not comparable to the low-degree generalized secant methods.

TABLE IV

COMPARISON OF NUMBER OF ITERATIONS
TO SOLVE THE EIGENVALUE PROBLEM

Algorithm	Iterations	
	Matrix A	Matrix B
Basic iteration	40	163
Vector ϵ -algorithm	29	28
Dynamic δ^2 -process	12	118
Extrapolation(M=1), unit metric	13	12
Relaxation, unit metric	14	15
Extrapolation(M=2), unit metric	17	16
Modified vector ϵ -algorithm(s=0,m=0)	29	48
Modified vector ϵ -algorithm(s=0,m=1)	13	22
Modified vector ϵ -algorithm(s=0,m=0,1)	29	38

Nonlinear Integral Equations

An attempt to solve the equation [2]

$$\int_a^b k(x,t)f(t)dt = g(x) + f(x)$$

may be made by setting up the classical iterative scheme

$$f_{r+1}(x) = \int_a^b k(x,t)f_r(t)dt - g(x).$$

The following two nonlinear integral equations have the solution cos($x\pi/ 4$).

$$f^{2}(x) = \frac{3\sqrt{2}\pi}{16}\int_{-1}^{1} dt\ f(t)\cos^{2}\frac{\pi|x-t|}{4} - \frac{1}{4} \qquad (6.4)$$

$$f(x) = \frac{3\sqrt{2}}{16}\int_{-1}^{1} dt\ f^{2}(t)\cos\frac{\pi|x-t|}{4} - \frac{1}{4}\cos\frac{\pi x}{4} \qquad (6.5)$$

Using the trapezoidal integration formula [12]

$$\int_{a}^{a+nh} f(t)dt = h\ \{\ \frac{1}{2}f_0 + f_1 + \ldots + f_{n-1} + \frac{1}{2}f_n + c\ \}$$

is transformed into the basic iteration among the vectors

$$(f_r(-1),\ f_r(-1+h),\ \ldots,\ f_r(-1-h),\ f_r(1))$$

with

$$r = 0,\ 1,\ \ldots$$

and

$$h = 2/n.$$

Figure 1 shows the relationship between two initial vectors, $f_0(x_i) = 1$ and $f_0(x_i) = 1 + x_i / 2$, and the solution vector of the equation (6.4). Setting $f_0(x_i) = 1$ and $f_0(x_i) = 1 - x_i^2 / 4$ to solve the equation (6.5), the relationship between these two initial vectors and the solution vector is showed in figure 2.

Table V shows the results of this iterative scheme and the effect of those acceleration techniques when h = 0.02.

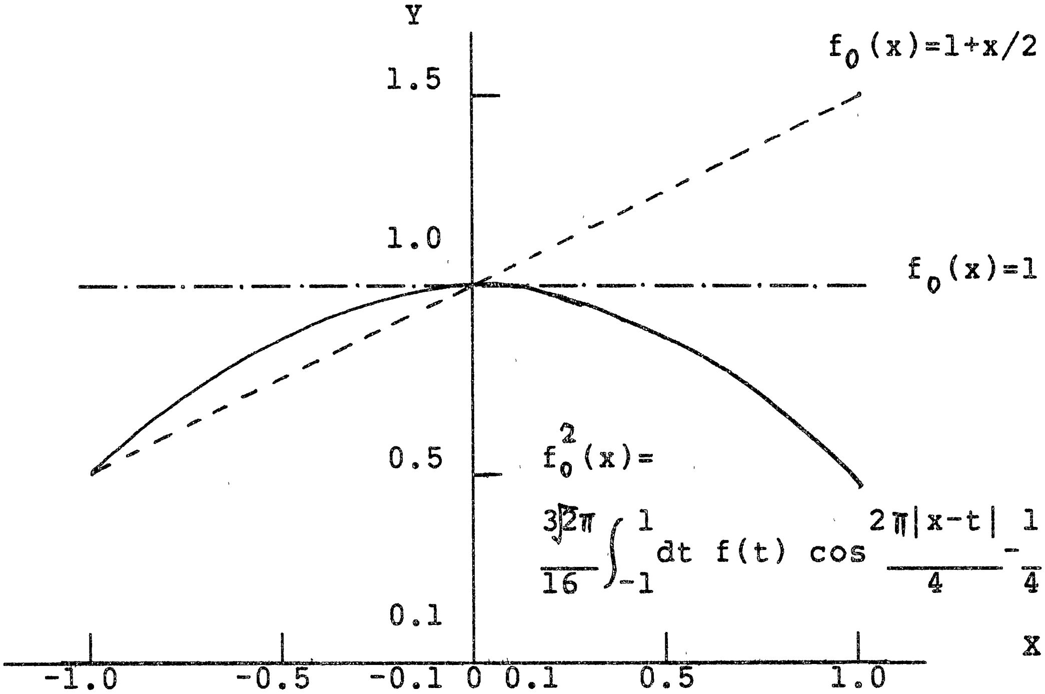


Figure 1. Nonlinear integral equation $f^2(x)$ and two different initial vectors

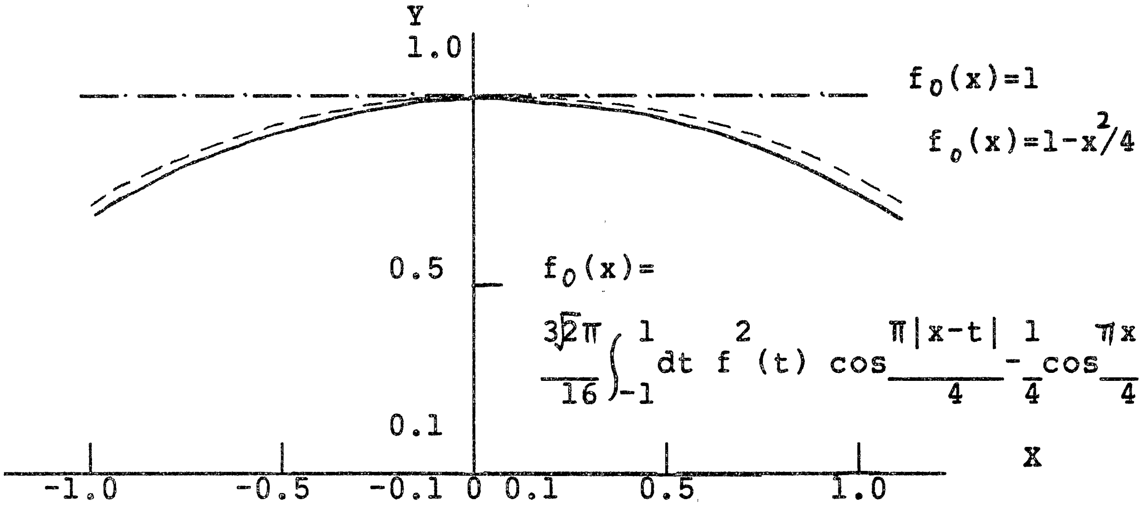


Figure 2. Nonlinear integral equation f(x) and two different initial vectors

TABLE V

COMPARISON OF NUMBER OF ITERATIONS TO SOLVE NONLINEAR INTEGRAL EQUATIONS (6.4) AND (6.5)

Algorithm	Eq. (6.4)		Eq. (6.5)	
	$f(x)$ $=1$	$f(x)$ $=1+x/2$	$f(x)$ $=1$	$f(x)$ $=1-x^2/4$
Basic iteration	22	22	∞	∞
Vector ϵ-algorithm	9	11	∞	∞
Dynamic δ^2-process	9	11	∞	∞
Reverse D. δ^2-process	-	-	24	36
Extrapolation(M=1),unit m.	6	11	7	6
Relaxation, unit metric	8	8	10	6
Extrapolation(M=2),unit m.	6	9	7	6
M. v. ϵ-algorithm,s=0,m=0	12	16	∞	∞
M. v. ϵ-algorithm,s=0,m=1	12	14	22	10
M. v. ϵ-algorithm,s=0,m=0,1	13	17	10	8

The basic iteration is unaffected by the initial guess in this numerical example, but the acceleration techniques depend on the initial guess. The vector epsilon algorithm, the dynamic delta-squared process and the modified vector epsilon algorithm (s=0, m=0) cannot cope with the divergence of the equation (6.5). The reverse dynamic delta-squared process overcomes the divergence of the equation (6.5) but is still not comparable to the low-degree generalized secant methods.

We now consider another nonlinear integral equation [25]

$$F(x) = 1 + \frac{1}{2} \varpi_0 x\, F(x) \int_0^1 \frac{F(x')}{x+x'}\, dx' \qquad 0<=x<=1 \qquad (6.6)$$

which arises in the theory of radiative transfer [8]. F(x) is defined to be

$$F(x) = \frac{1}{x_1 \dots x_n} * \frac{\prod_{j=1}^{n} (x+x_j)}{\prod_{i=1}^{n} (1+k_i x)},$$

where the k_i's are constants.

The quantity ϖ_0 (pi-nought) is a parameter called the albedo, and has the value between 0 and 1. By using the Gaussian integration rule [21] of the form

$$\int_0^1 f(s)ds \simeq \sum_{i=1}^{n} w_i f(s_i),$$

the integral transform in the equation (6.6) can be approximated as

$$\int_0^1 \frac{F(x')}{x+x'}\, dx' \simeq \sum_{j=1}^{n} \frac{w_j}{x+x_j} F(x_j) \qquad 0<=x<=1.$$

The points s_i, $0<=S_i<=1$, i=1, 2, ..., n are called the nodes and the numbers w_i, i=1, 2, ..., n are called the weights of the given rule. Matrix B is defined by

$$B_{ij} = \frac{x_i w_j}{x_i + x_j} \qquad (i,\ j = 1,\ 2,\ \ldots,\ n)$$

and a basic iteration by

$$x_{n+1} = 1 + 1/2\ \tilde{w}_0 \{\ x_n(Bx_n)\}.$$

The nodes and the weights for the Gaussian integration rule of order nine are given in table VI [23].

TABLE VI

NODES AND WEIGHTS FOR THE GAUSSIAN INTEGRATION RULE OF ORDER NINE

i	s	w
1	.0159199	.0406372
2	.0819844	.0903241
3	.1933143	.1303053
4	.3378733	.1561735
5	.5000000	.1651197
6	.6621267	.1561735
7	.8066857	.1303053
8	.9180156	.0903241
9	.9840801	.0406372

The initial vector x_0 has a value of 1 for each component. The parameter ϖ_0 is selected to be 0.8, 0.9, 0.99 and 1.0. When ϖ_0 =1.0, called the conservative case, the resulting sequence is slowly convergent. The results are shown on table VII.

TABLE VII

COMPARISON OF NUMBER OF ITERATIONS TO SOLVE NONLINEAR INTEGRAL EQUATION (6.6)

Algorithm	Iterations			
	ϖ_0=.8	ϖ_0=.9	ϖ_0=.99	ϖ_0=1.0
Basic iteration	19	28	83	974
Vector -algorthm	9	9	17	497
Dynamic -process	9	13	21	157
Extrapolation(M=1),unit metric	19	23	44	38
Relaxation, unit metric	31	29	39	115
Extrapolation(M=2),unit metric	15	18	45	72
Modified vector -algorithm,s=0,m=0	16	24	78	967
Modified vector -algorithm,s=0,m=1	18	21	120	3000
Modified vector -algorithm,s=0,m=0,1	16	24	78	967

As the parameter ϖ_0 increases, the low-degree generalized secant methods have the better results. The vector epsilon algorithm and the delta-squared process have stable, rapidly convergent sequences, but the modifications of the vector epsilon algorithm can't cope with this problem.

Boundary Value Problem

It is convenient to use finite difference methods as an iterative scheme for the solution of boundary value problem [15]. Laplace's equation in two dimensions is a simple numerical example:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad .$$

In general, the discretization of a partial difference equation in a domain of independent variables requires replacement of the domain by a finite number of points, referred to as mesh points. With reference to figure 3, we define that the four edges of a rectangle are used for boundary conditions.

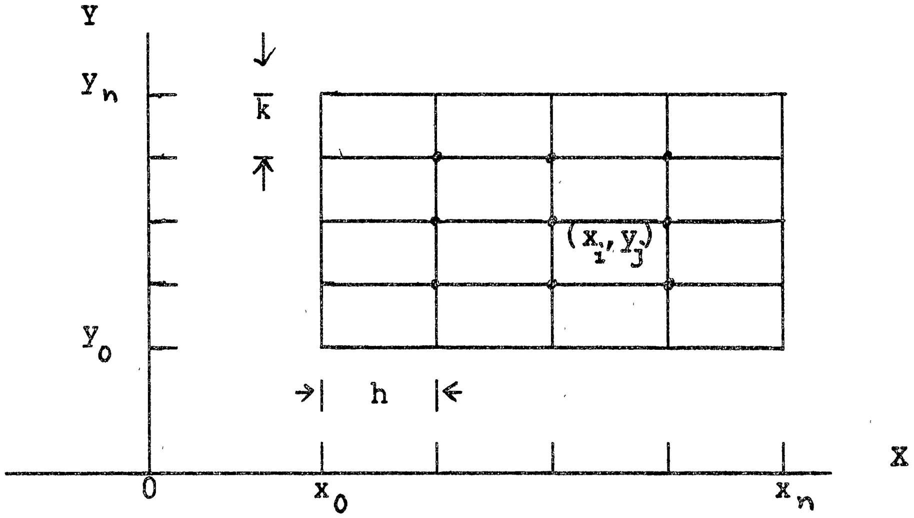


Figure 3. Rectangular finite difference net

Second derivatives with finite difference approximations for a rectangular net are defined

$$\frac{\partial^2 u}{\partial x^2} \simeq \frac{u(x+h,\ y) - 2u(x,y) + u(x-h,y)}{h^2}$$

$$\frac{\partial^2 u}{\partial y^2} \simeq \frac{u(x,\ y+k) - 2u(x,y) + u(x,\ y-k)}{k^2}$$

We then define at mesh point $(x_i,\ y_j)$

$$\Delta u(x_i,\ y_j) = 1/h^2 (u_{i-1,j} - 2u_{ij} + u_{i+1,j}) + 1/k^2 (u_{i,j-1} - 2u_{ij} + u_{i,j+1}). \qquad (6.7)$$

To simplify the structuring of problem, let us set h = k, and rewrite the equation (6.7) to be five point Laplace difference equation

$$\Delta u(x_i,\ y_j) = 1/h^2 (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}).$$

Now let us consider the steady-state temperature distribution in a square, homogeneous isotropic conducting slab with insulated faces when the temperature along the edges is known; see figure 4.

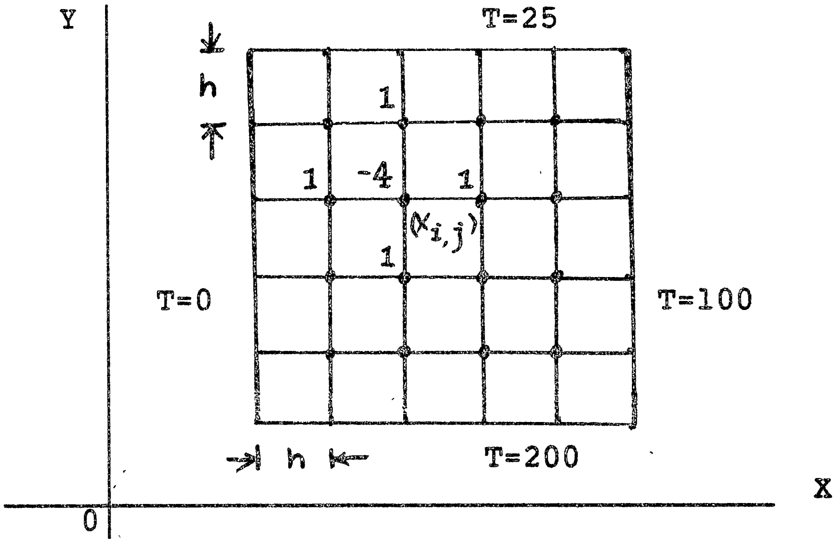


Figure 4. Heat conduction example

Writing the difference equation for the mesh points, we obtain nine equations for the nine unknown values (x_i, y_j), i, j = 1, 2, 3. We also set i, j = 1, 2, ..., 10 and obtain one hundred equations for the one hundred unknown values (x_i, y_j). Table VIII contains the results of this boundary value problem with $(x_i, y_j) = 0$.

TABLE VIII

COMPARISON OF NUMBER OF ITERATIONS
TO SOLVE BOUNDARY VALUE PROBLEM

Algorithm	Iterations	
	n = 9	n = 100
Basic iteration	25	171
Vector ϵ-algorithm	9	38
Dynamic δ^2-process	14	75
Extrapolation(M=1), unit metric	11	52
Relaxation, unit metric	17	50
Extrapolation(M=2), unit metric	10	40
Modified vector ϵ-algorithm(s=0,m=0)	24	192
Modified vector ϵ-algorithm(s=0,m=1)	19	117
Modified vector ϵ-algorithm(s=0,m=0,1)	29	192

The vector epsilon algorithm has the most rapidly convergent sequences. But the modifications of the vector epsilon algorithm (s=0, m=0 and s=0, m=0,1) can't improve the speed of convergence. The dynamic delta-squared process and the low degree generalized secant methods have the better results but are not comparable to the vector epsilon algorithm.

CHAPTER VII

CONCLUSION

The numerical evidence presented here shows that the low-degree generalized secant methods including the extrapolation and the relaxation processes have the most rapid or above average convergent rates. The modified vector epsilon algorithm is more efficient than the vector epsilon algorithm for the nonlinear integral equation (6.5), however it is not as fast as the low-degree generalized secant methods. The dynamic delta-squared process has average results and does not converge for the nonlinear integral equation (6.5). The reverse dynamic delta-squared process converges, but is not comparable to the other acceleration techniques in this problem.

Considering the storage requirements for n equations, the dynamic delta-squared process and the modified vector epsilon algorithm (s=0, m=1) need 3n pieces of data; two other modified vector epsilon algorithm require 4n pieces of data. The extrapolation method (M=1) and the relaxation method require the storage of 6n pieces of data. The worst storage case is the vector epsilon algorithm which requires the storage of $2n^2$ pieces of data.

Considering both storage and computer time, the extrapolation method with M=1 is recommended in the low-degree generalized secant methods. In the class of vector epsilon algorithms, the vector epsilon algorithm (s=0, m=1) is the best choice.

The numerical examples in this paper are both small and simple so as to simplify the exposition. As indicated the acceleration techniques are worth applying when the basic iteration is slowly convergent or divergent, hence it is desirable that further work be done on big and practical problems.

A SELECTED BIBLIOGRAPHY

(1) Aitken, A. C. "On Bernoulli's Numerical Solution of Algebraic Equations." *Proc. Roy. Soc. Edinburgh*, Vol. 46 (1926), 289-305.

(2) Anderson, D. G. "Iterative Procedures for Nonlinear Integral Equations." *J. ACM*, Vol. 12 (Oct. 1965), 547-560.

(3) Boyle, E. F. "Accelerating the Convergence of Elastic-Plastic Stress Analysis." *Int. J. Num. Math. Eng.*, Vol. 7 (1974), 232-235.

(4) Brezinski, C. L. "Numerical Stability of a Quadratic Method for Solving Systems of Non Linear Equations." *Computing*, Vol. 14 (1975), 205-211.

(5) Brezinski, C. "Some Results in the Theory of the Vector ϵ-Algorithm." *Linear Algebra and Its Applications*, Vol. 8 (1974), 77-86.

(6) Brezinski, C. and Rieu, A. C. "The Solution of Systems of Equations Using the ϵ-Algorithm, and an Application to Boundary-Value Problems." *Math. Comp.*, Vol. 28 (July 1974), 731-741.

(7) Chandler, J. P. Subroutine VAITK. Computer Science Dept., Oklahoma State Univ., July, 1981.

(8) Chandrasekhar, S. *Radiative Transfer*. London: Oxford University Press, 1950.

(9) Cheng, H. K. and Hafez, M. M. "Cyclic Iterative Method Applied to Transonic Flow Analyses." *Lecture Notes in Physics No. 47 Padé Application to Mechanics*. Ed. H. Cabannes. New York: Spring-Verlag, 1976, 101-121.

(10) Conte, S. D. and de Boor, C. *Elementary Numerical Analysis*: An Algorithmic Approach. New York: McGraw-Hill, 1980.

(11) Cuyt, A. A. M. "The ϵ-Algorithm and Multivariate Pade-Approximants." *Numerische Mathematik*, Vol. 40 (1982), 39-46.

(12) Davis, P. J. and Rabinowitz, P. Numerical Integration. Massachusetts: Blaisdell Publishing Company, 1967.

(13) Gekeler, E. "On the Solution of Systems of Equations by the Epsilon Algorithm of Wynn." Math. Comp., Vol. 26 (Apr. 1972), 427-435.

(14) Gragg, W. B. "The Padé Table and Its Relation to Certain Algorithms of Numerical Analysis." SIAM Review, Vol. 14 (1972), 1-62.

(15) Holt, J. F. "Numerical Solution of Nonlinear Two-Point Boundary Problems by Finite Difference Methods." Comm. ACM, Vol. 7 (1964), 366-373.

(16) Householder, A. S. The Theory of Matrices in Numerical Analysis. New York: Blaisdell Publishing Company, 1964.

(17) Jennings, A. "Accelerating the Convergence of Matrix Iterative Processes." J. Inst. Math. Applic., Vol. 8 (1971), 99-110.

(18) Kersten, R. D. Engineering Differential Systems. New York: McGraw-Hill, 1969.

(19) Khabaza, I. M. "An Iterative Least-Square Method Suitable for Solving Large Sparse Matrices." Comput. J., Vol. 6 (1963-64), 202-206.

(20) King, R. F. "An Effecient One-point Extrapolation Method for Linear Convergence." Math. Comp., Vol. 35 (Oct. 1980), 1285-1290.

(21) McLeod, J. B. "A Note on the ϵ-Algorithm." Computing, Vol. 7 (1971), 17-24.

(22) Milne, W. E. Numerical Calculus. New Jersey: Princeton University Press, 1949.

(23) Olver, F. W. J. "The Evaluation of Zeros of High-degree Polynomials." Roy. Soc. Phil. Trans., Vol. 244 (1952), 385-415.

(24) Ostrowski, A. M. Solution of Equtions in Euclidean and Banach Spaces. New York: Academic Press, 1973.

(25) Rall, L. B. Computational Solution of Nonlinear Operator Equations. New York: John Wiley & Sons, Inc., 1969.

(26) Shanks, D. "Non-Linear Transformations of Divergent

and Slowly Convergent Sequences." *J. Math. Phys.*, Vol. 34 (Apr. 1955), 1-42.

(27) Smith, D. A. and Ford, W. F. "Numerical Comparison of Nonlinear Convergence Accelerators." *Math. Comp.* Vol. 38 (1982), 481-500.

(28) Thacher, H. C. "Algorithm 215 Shanks." *Comm. ACM*, Vol. 6 (Nov. 1963), 662.

(29) Traub, J. F. *Iterative Methods for the Solution of Equations.* New York: Prentice-Hall Inc., 1964.

(30) Varga, R. S. *Iterative Matrix Analysis.* Englewood Cliffs, New York: Prentice-Hall, 1962.

(31) Wilkinson, J. H. *The Algebraic Eigenvalue Problem.* London: Oxford Univ. Press, 1965.

(32) Wolfe, P. "The Secant Method for Simultaneous Nonlinear Equations." *Comm. ACM*, Vol. 2 (Dec. 1959), 12-13.

(33) Wynn, P. "On a Device for Computing the $e_m(S_n)$ Transformation." *MTAC* (*Math. Tables Aids Comp.*), Vol. 10 (1956), 91-96.

(34) Wynn, P. "The Rational Approximation of Functions which Are Formally Defined by a Power Series Expansion." *Math. Comp.*, Vol. 14 (1960), 147-186.

(35) Wynn, P. "Acceleration Techniques for Iterated Vector and Matrix Problems." *Math. Comp.*, Vol. 16 (July 1962), 301-322.

(36) Wynn, P. "General Purpose Vector Epsilon Algorithm Algol Procedures." *Numer. Math.*, Vol. 6 (1964), 22-36.

(37) Wynn, P. "Upon Systems of Recursions which Obtain Among the Quotients of the Padé Table." *Numer. Math.*, Vol. 8 (1966), 264-269.

APPENDIX A

FORTRAN LISTING OF VECTOR EPSILON ALGORITHM

```
$JOB
C     DENOMSTRATION TEST PROGRAM FOR SUBROUTINES STIT, E1 AND E2.
C     THESE SUBROUTINES ARE TRANSFORMED FROM
C     P. WYNN, "GENERAL PURPOSE VECTOR EPSILON ALGORITHM ALGOL
C     PROCEDURES."NUMERISCHE MATHEMATIK 6(1964), 22-36.
C
C     FLOWCHART:
C                            --------
C                              MAIN
C                            --------
C      --------             ------                   ------
C        STIT                 E1                       E2
C      --------             ------                   ------
C  STRAIGHTFORWARD      DISPLAY APPLICATION     APPLY E-ALGORITHM TO A
C  ITERATE              OF E-ALGORITHM TO       FIXED NUMBER OF ITERATED
C                       ITERATED VECTOR         VECTOR E(2S) S=0,1,.....
C  ---- ------ --  ---- ------ -- ------ -----   ---- ------ -- ----
C  EVEN SAMINV FE  EVEN SAMINV FE TKSMPL PRINT   EVEN SAMINV FE TDIS
C  ---- ------ --  ---- ------ -- ------ -----   ---- ------ -- ----
C
C  ****************     VARIABLE REFERENCE      ********************
C     DELTA             TOLERANCE
C     DINIT(*)          INITIAL VALUES OF THE VECTOR'S COMPONENTS
C     IVEND             AVAILABLE STORAGE
C     LPT               LOGICAL UNIT NUMBER OF THE OUTPUT
C     METHOD            METHOD PARAMETER
C                       =1 STRAIGHTFORWARD ITERATE
C                       =2 VECTOR E-ALGORITHM WITH E(S), S=0,1,...
C                       =3 VECTOR E-ALGORITHM WITH E(2S), S=0,1,...
C     MMAX              THE MAXIMUM NUMBER OF ITERAIONS ALLOWED
C     N                 THE NUMBER OF VECTOR'S COMPONENTS
C     NRD               LOGICAL UNIT NUMBER OF THE INPUT
C     NTRAC             TRACE PARAMETER
C                       =0 PRINT DISTANCES AND FINAL RESULT
C                       =1 PRINT DISTANCES, TRUNCATION ERRORS AND FINAL
C                          RESULT
C                       =2 PRINT DISTANCES, FINAL RESULT AND THE
C                          RESULT OF EACH ITERATION
C
      DOUBLE PRECISION BETA,DELTA,DINIT,D
      COMMON LPT,NRD,BETA(100),DINIT
      COMMON /EXTRA/D
      DATA LPT/6/, NRD/5/
      METHOD=1
      D=2.5D1
C     D=1.5D1
      NTRAC=2
      N=20
      MMAX=70
      DO 10 I=1,N
   10 DINIT(I)=1.D0
      IVEND=400
      DELTA=1.D-5
      IF(METHOD-2) 20,30,40
   20 WRITE(LPT,21)
   21 FORMAT('1','STRAIGHTFORWARD ITERATE'//1X,'INITIAL VECTOR')
      WRITE(LPT,45) (DINIT(I),I=1,N)
      CALL STIT(N,DELTA,MMAX)
      GOTO 50
   30 WRITE(LPT,31)
   31 FORMAT('1','E-ALGORITHM WITH E(S), S=0,1,...'//  1X,
     C'INITIAL VECTOR')
      WRITE(LPT,45) (DINIT(I),I=1,N)
      CALL E1(N,MMAX)
      GOTO 50
   40 WRITE(LPT,41)
   41 FORMAT('1','E-ALGORITHM WITH E(2S), S=0,1,...'/ /1X,
     C'INITIAL VECTOR')
      WRITE(LPT,45) (DINIT(I),I=1,N)
   45 FORMAT(' ',7E15.5)
      CALL E2(N,DELTA,IVEND)
   50 STOP
      END
```

201 , 70

```
      SUBROUTINE STIT(N,DELTA,MMAX)
C
C     THIS SUBROUTINE CARRIES OUT THE STRAIGHTFORWARD ITERATION OF THE
C     GIVEN FUNCATIONAL EQUATION.
C
C  ****************         INPUT     ***********************************
C     N                     THE NUMBER OF VECTOR'S COMPONENTS
C     DELTA                 TOLERANCE
C     MMAX                  THE MAXIMUM NUMBER OF ITERATION ALLOWED
C
C  ****************         OUTPUT    ***********************************
C     MS                    ITERATION COUNTER
C     TERROR                TRUNCATION ERROR
C     DIS                   DISTANCE BETWEEN TWO SUCCESSIVE VECTORS
C     AUX(*,*)              THE VECTOR OF N COMPONENTS
C
      DOUBLE PRECISION AUX(4,100),BETA,DELTA,DIS,TERROR,DINIT
      LOGICAL FTIME
      COMMON LPT,NRD,BETA(100),DINIT(100),NTRAC
C                           SET THE ITERATION COUNTER -- MS=1, THE LABEL
C                           OF CURRENT BOX -- M=1, AND THE LABEL OF NEXT
C                           BOX M1=2. THEN INITIALIZE THE VALUES OF
C                           THESE TWO BOXES.
      MS=1
      M=1
      M1=2       20
      DO 101 I=1,N
      AUX(M,I)=DINIT(I)
      AUX(M1,I)=DINIT(I)
  101 CONTINUE
      FTIME=.TRUE.
  105 IF(MS.GT.1) FTIME=.FALSE.
      IF(MS.GT.MMAX) GOTO 160
C                           BEGIN THE NEXT ITERATION.
C                           X(N+1)=F(X(N)).
      CALL FE(AUX,M1,M,N,DIS,TERROR,FTIME)
      IF(NTRAC.EQ.1) GOTO 126
      WRITE(LPT,125)MS,DIS
  125 FORMAT(' ','ITERATION',I3,5X,'DISTANCE=',E15.5)
      GOTO 128
  126 WRITE(LPT,127)MS,DIS,TERROR
  127 FORMAT(' ','ITERATION',I3,5X,'DISTANCE=',E15.5,5 X,
     C17HTRUNCATION ERROR=,E15.5)
C                           TEST CONVERGENCE. SWAP THE LABELS OF BOXES
C                           THEN GO BACK FOR THE NEXT ITERATION.
  128 IF(DABS(DIS).LE.DELTA) GOTO 135
      MS=MS+1
      M=M1
      M1=3-M
      GOTO 105
  135 IF(NTRAC.NE.1) GOTO 141
      IF(TERROR.GT.DELTA) GOTO 150
C                           PRINT THE FINAL MESSAGE.
  141 WRITE(LPT,147) (AUX(M1,I),I=1,N)
  147 FORMAT(' ','FINAL RESULT',7E15.5)
      GOTO 170
  150 WRITE(LPT,151)
  151 FORMAT('0','TRUNCATION ERROR BIGGER THAN DELTA.')
      GOTO 141
  160 WRITE(LPT,161)
  161 FORMAT('0','THE NUMBER OF ITERATIONS MORE THAN   THE MAXINUM',
     C,' NUMBER OF ITERATIONS ALLOWED.')
      GOTO 141
  170 RETURN
      END
```

V01-150

```
      SUBROUTINE E1(N,MMAX)
C
C     THIS SUBROUTINE DISPLAY THE APPLICATION OF THE EPSILON ALGORITHM.
C
C  ****************         INPUT      ************************************
C     N                     THE NUMBER OF VECTOR'S COMPONENTS
C     MMAX                  THE MAXIMUM NUMBER OF ITERATIONS ALLOWED
C
C  ****************         OUTPUT     ************************************
C     M                     ITERATION COUNTER
C     TERROR(DISPLY(*,2))TRUNCATION ERROR
C     DIS(DISPLY(*,1))      DISTANCE BETWEEN TWO SUCCESSIVE VECTORS
C
      DOUBLE PRECISION DISPLY,RM1,AUX,BETA,DIS,TERROR,DINIT
      LOGICAL EVEN
      DIMENSION RL(200,20)
      COMMON /E1E2/ DISPLY(100,2),RM1(100),AUX(4,100),IONE,ITWO,ITHREE
      COMMON LPT,NRD,BETA(100),DINIT(100),NTRAC
C                           INITIALIZE THE VALUES OF AUXILIARY STORAGE
C                           BOXES.
      MMAX2=MMAX*2
      DO 401 I=1,N
      RM1(I)=DINIT(I)
      AUX(1,I)=DINIT(I)
      AUX(2,I)=DINIT(I)
      AUX(3,I)=DINIT(I)
      RL(1,I)=DINIT(I)
  401 CONTINUE
      DO 450 M=1,MMAX
      IONE=1
      ITWO=2
      ITHREE=3
C                           FIRST MEMBER OF NEW BACKWARD DIAGONAL PUT IN
C                           AUX(ITWO,*).
      DO 402 I=1,N
  402 AUX(ITWO,I)=RM1(I)           M1=0
      M1=M-1
      IF (M.LE.1) GOTO 435
      DO 430 J=1,M1
C
C                            (M)    (M+1)      (M+1)  (M)
C                           E    = E     +1/(E      - E     ) WITH
C                            S+1    S-1         S      S
C
C                            (M)                      (M+1)
C                           E    -- AUX(IONE,*),    E      -- RL(J1,*)
C                            S+1                      S-1
C
C                            (M+1)                  (M)
C                           E      -- AUX(ITWO,*),  E      -- RL(J,*)
C                            S                      S
C
      DO 410 I=1,N
      AUX(4,I)=AUX(ITWO,I)-RL(J,I)
  410 CONTINUE
      CALL SAMINV(IONE,AUX,N)
      J1=J-1
      IF (J.LE.1) GOTO 420
      DO 415 I=1,N
      AUX(IONE,I)=AUX(IONE,I)+RL(J1,I)
      RL(J1,I)=AUX(ITHREE,I)
  415 CONTINUE
C                           CHANGE THE LABELS ON THE BOXES.
  420 ISPL=IONE
      IONE=ITHREE
      ITHREE=ITWO
      ITWO=ISPL
      IF (.NOT.EVEN(J1)) GOTO 430
      MS=J1*(MMAX2-J1)/4+M1
      CALL TKSMPL(MS,M,MMAX,N,DIS,TERROR)
  430 CONTINUE
```

```
C                                  END OF BACKWARD DIAGONAL REACHED.
      IF (M.LE.1) GOTO 435
      DO 432 I=1,N
  432 RL(M1,I)=AUX(ITHREE,I)
  435 DO 436 I=1,N
  436 RL(M,I)=AUX(ITWO,I)
C                                  CHOOSE EVEN COLUMNS TO BE DISPLAYED.
      IF (.NOT.EVEN(M1)) GOTO 450
      ITHREE=ITWO
      MS=M1*(MMAX2-M1+4)/4
      CALL TKSMPL(MS,M,MMAX,N,DIS,TERROR)
  450 CONTINUE
      IN=1
      CALL PRINT(MMAX,N,IN)
      IF(NTRAC.NE.1) GOTO 455
      IN=2
      CALL PRINT(MMAX,N,IN)
  455 WRITE(LPT,456) (AUX(IONE,I),I=1,N)
  456 FORMAT(' ','FINAL RESULT',7E15.5)
      RETURN
      END
      SUBROUTINE E2(N,DELTA,IVEND)
C
C     THIS SUBROUTINE APPLYS THE EPSILON ALGORITHM TO A FIXED NUMBER
C     OF ITERATED VECTORS, E(2S) S=0,1,2,....
C
C  ****************     INPUT      **********************************
C     N                 THE NUMBER OF VECTOR'S COMPONENTS
C     DELTA             TOLERANCE
C     IVEND             AVAILABLE STORAGE
C
C  ****************     OUTPUT     **********************************
C     AUX(ITWO,*)       THE FINAL VECTOR
C
      DOUBLE PRECISION RL,AUX,RM1,BETA,DIS,TERROR,TDIS,DINIT,
     CDELTA
      LOGICAL EVEN,PSUCSS,ISUCSS,FTIME
      DIMENSION RL(10,100),AUX(4,100),RM1(100)
      COMMON LPT,NRD,BETA(100),DINIT(100),NTRAC
C                         SET PERHAPS SUCCESSFUL -- PSCUSS=FALSE, AND
C                         INDEED SUCCESSFUL -- ISUCSS=FALSE. INITIALIZE
C                         THE VALUES OF AUXILITARY STORAGE BOXES.
      DATA ISUCSS/.FALSE./,PSUCSS/.FALSE./
      M=1
      FTIME=.TRUE.
      DO 210 I=1,N
      RM1(I)=DINIT(I)
      AUX(1,I)=DINIT(I)
      AUX(2,I)=DINIT(I)
      AUX(3,I)=DINIT(I)
      RL(1,I)=DINIT(I)
  210 CONTINUE
  220 J=0
      IONE=1
      ITWO=2
      ITHREE=3
      IF(M.GT.1)FTIME=.FALSE.
C                         LAST ITERATE PUT INTO AUX(ITHREE,*) IN
C                         PREPARATION FOR CALLING SUBROUTINE FE.
      DO 221 I=1,N
      AUX(ITHREE,I)=RM1(I)
  221 CONTINUE
      CALL FE(AUX,ITWO,ITHREE,N,DIS,TERROR,FTIME)
      DO 230 I=1,N
  230 RM1(I)=AUX(ITWO,I)
  240 J1=J+1
C                          (M)   (M+1)     (M+1)  (M)
C                         E   = E     +1/(E     - E    ) WITH
C                          S+1    S-1        S     S
C
C                          (M)                     (M+1)
C                         E   -- AUX(IONE,*),     E     -- RL(J,*)
C                          S+1                     S-1
C
C                          (M+1)                   (M)
C                         E     -- AUX(ITWO,*),   E   -- RL(J1,*)
C                          S                       S
      DO 241 I=1,N
      AUX(4,I)=AUX(ITWO,I)-RL(J1,I)
  241 CONTINUE
      CALL SAMINV(IONE,AUX,N)
      IF(J.LE.0) GOTO 260
      DO 250 I=1,N
      AUX(IONE,I)=AUX(IONE,I)+RL(J,I)
      RL(J,I)=AUX(ITHREE,I)
  250 CONTINUE
C                         CHANGE THE LABELS ON THE BOXES.
  260 ISPL=IONE
      IONE=ITHREE
      ITHREE=ITWO
      ITWO=ISPL
      J=J+1
      IF(J.LT.M)GOTO 240
      M1=M+1
      DO 265 I=1,N
      RL(M,I)=AUX(ITHREE,I)
      RL(M1,I)=AUX(ITWO,I)
  265 CONTINUE
      IF(M.LT.2) GOTO 280
C                        END OF BACKWARD DIAGONAL.
      IONE=0
      IF(EVEN(M)) IONE=1
      M3=M1-3+IONE
      M2=M1-1+IONE
C                        TEST CONVERGENCE.
      DIS=TDIS(RL,M3,M2,N)
      IF(DIS.GE.DELTA) GOTO 280
      IF(NTRAC.EQ.1) GOTO 269
      WRITE(LPT,1) M,DIS
    1 FORMAT(' ','ITERATION',I3,5X,'DISTANCE=',E15.5)
      GOTO 270
  269 WRITE(LPT,2) M,DIS,TERROR
    2 FORMAT(' ','ITERATION',I3,5X,'DISTANCE=',E15.5,5X,
     C'TRUNCATION ERROR=',E15.5)
  270 DO 271 I=1,N
  271 AUX(ITHREE,I)=RL(M2,I)
      CALL FE(AUX,ITWO,ITHREE,N,DIS,TERROR,FTIME)
      IF(DABS(DIS).GT.DELTA) GOTO 280
      PSUCSS=.TRUE.
      IF(NTRAC.NE.1) GOTO 274
      IF(TERROR.GT.DELTA) GOTO 276
  274 ISUCSS=.TRUE.
      GOTO 290
  276 ISUCSS=.FALSE.
      GOTO 290
  280 WRITE(LPT,1)M,DIS
      M=M+1
      IF(M*(N).LT.IVEND) GOTO 220
      PSUCSS=.FALSE.
  290 IF(PSUCSS.AND.ISUCSS) GOTO 291
      GOTO 299
C                        PRINT THE FINAL MESSAGE.
  291 WRITE(LPT,292) M
  292 FORMAT(' ','FINAL RESULT'//1X,'M=',I3)
  293 WRITE(LPT,294) (AUX(ITWO,I),I=1,N)
  294 FORMAT(' ',7E15.5)
      GOTO 310
  299 WRITE(LPT,300)
  300 FORMAT('0','TRUNCATION ERROR BIGGER THAN THE TOLERANCE.')
  310 RETURN
      END
      LOGICAL FUNCTION EVEN(INTGR)
C
C     WHEN PARAMETER IS EVEN, FUNCTION VALUE IS TRUE; OTHERWISE
C     FUNCATION VALUE IS FALSE.
C
      EVEN=.FALSE.
      IF(MOD(INTGR,2).EQ.0) EVEN=.TRUE.
      RETURN
      END
```

```
      SUBROUTINE SAMINV(IRES,AUX,LENGTH)
C
C     THIS SUBROUTINE DOES THE SAMELSON INVERSE. WHEN THE COMPONENTS
C     OF THE VECTOR ARE ALL REAL, THE SAMELSON INVERSE IS FORMED
C     MERELY BY DIVIDING THE VECTOR THROUGHOUT BY THE SUM OF THE
C     SQUARES OF ITS COMPONENTS.
C
      DOUBLE PRECISION AUX,HUGE,DENOM
      DIMENSION AUX(4,100)
      DATA HUGE/1.D30/
C                         -1        2
C                        Y  =Y/||Y||
C
      DENOM=0.D0
      DO 620 I=1,LENGTH
  620 DENOM=DENOM+AUX(4,I)*AUX(4,I)
      IF(DENOM.LE.0.D0) GOTO 660
      DO 630 I=1,LENGTH
  630 AUX(IRES,I)=AUX(4,I)/DENOM
      GOTO 670
  660 DO 665 I=1,LENGTH
  665 AUX(IRES,I)=HUGE
  670 RETURN
      END
      DOUBLE PRECISION FUNCTION TDIS(RL,ICURET,INEXT,LENGTH)
C
C     THIS SUBROUTINE FIND THE DISTANCE BETWEEN TWO SUCCESSIVE VECTORS
C
      DOUBLE PRECISION ABST,PTDIS,RL(10,100)
      ABST=0.
      DO 310 I=1,LENGTH
      PTDIS=DABS(RL(ICURET,I)-RL(INEXT,I))
      IF(PTDIS.GT.ABST) ABST=PTDIS
  310 CONTINUE
      TDIS=ABST
      RETURN
      END
```

```
      SUBROUTINE TKSMPL(MS,M,MMAX,N,DIS,TERROR)
C
C     THIS SUBROUTINE CALLS FE TO COMPUTE THE FUNCATIONAL EQUATION,
C     THEN MAPS THE VALUES OF TRUNCATION ERROR AND DISTANCE ONTO THE
C     DISPLAY VECTOR.
C
      DOUBLE PRECISION DISPLY,RM1,AUX,BETA,DIS,TERROR,DINIT
      LOGICAL FTIME,EVEN
      COMMON /E1E2/DISPLY(100,2),RM1(100),AUX(4,100),IONE,ITWO,ITHREE
      COMMON LPT,NRD,BETA(100),DINIT(100),NTRAC
      DATA FTIME/.TRUE./
      IF(M.GT.1) FTIME=.FALSE.
      MS=MS+1
      CALL FE(AUX,IONE,ITHREE,N,DIS,TERROR,FTIME)
      DISPLY(MS,1)=DIS
      IF(NTRAC.NE.1) GOTO 506
      DISPLY(MS,2)=TERROR
C                           IF THE MS INDICES THE FIRST COLUMN OF THE
C                           EPSILON ARRAY, THEN THE NEXT ITERATION IS
C                            ALREADY THE NEXT MEMBER OF THE ORIGINAL
C                           SEQUENCE. IT IS STORED IN RM1(*) FOR FUTURE
C                           USE.
  506 IF(MS.GT.MMAX) GOTO 510
      DO 509 I=1,N
  509 RM1(I)=AUX(IONE,I)
  510 RETURN
      END
      SUBROUTINE PRINT(MMAX,N,IN)
C     THIS SUBROUTINE PRINTS THE VALUES OF TRUNCATION ERROR AND DIS.
      DOUBLE PRECISION DISPLY,RM1,AUX,BETA,DINIT
      DIMENSION INDEX(6)
      COMMON /E1E2/DISPLY(100,2),RM1(100),AUX(4,100),IONE,ITWO,ITHREE
      COMMON LPT,NRD,BETA(100),DINIT(100),NTRAC
      DATA INDEX/0,2,4,6,8,10/
      MMAX2=MMAX/2
      MMAX3=(MMAX+1)/2
C                         PRINT DISTANCES OR TRUNCATION ERRORS BY IN.
      IF(IN.GT.1)GOTO 513
      WRITE(LPT,512)(INDEX(I),I=1,MMAX3)
  512 FORMAT('0','DISTANCES'/1X,3H S ,5I15)
      GOTO 515
  513 WRITE(LPT,514)(INDEX(I),I=1,MMAX3)
  514 FORMAT('0','TRUNCATION ERRORS'/1X,'H',5I15)
  515 WRITE(LPT,516)
  516 FORMAT(' ',' M')
C                         REARRANGE THE DATA OF DISPLAY(*,*) INTO OUTPUT
C                         BUFFER BETA(*).
      DO 540 I=1,MMAX2
      MPLUS=MMAX
      K=1
      IND=I
  525 BETA(K)=DISPLY(IND,IN)
      IF(K.GE.I) GOTO 530
      K=K+1
      IND=IND+MPLUS
      MPLUS=MPLUS-2
      GOTO 525
  530 WRITE(LPT,531)I,(BETA(J),J=1,K)
  531 FORMAT(' ',I2,10E15.5)
  540 CONTINUE
      MMAXL=MMAX-MMAX2
      MI=MMAXL
      DO 560 I=1,MMAXL
      MPLUS=MMAX
      K=1
      IND=I+MMAX2
      ICK=IND
  545 BETA(K)=DISPLY(IND,IN)
      IF(K.GE.MI) GOTO 550
      K=K+1
      IND=IND+MPLUS
      MPLUS=MPLUS-2
      GOTO 545
  550 WRITE(LPT,531)ICK,(BETA(J),J=1,K)
  560 MI=MI-1
  570 RETURN
      END
      SUBROUTINE FE(AUX,INEXT,ICURET,N,DIS,TERROR,FTIME)
C
C     THIS TEST FUNCTION COMES FROM
C     D.G. ANDERSON, ITERATIVE PROCEDURES FOR NONLINEAR INTEGRAL
C     EQUATIONS, JOURNAL OF ASSOCIATION FOR COMPUTING MACHINERY, VOL.12
C     NO.4 (OCTOBER, 1965), 547-560.
C          A*Z=D*B WITH
C                          A(I,J)=D,  I=J
C                                =1,  I¬=J
C          THE BASIC ITERATION IS
C          Z=H*Z+B WITH
C                          H(I,J)=0,  I=J
C                                =-1/D,  I¬=J.
C
      DOUBLE PRECISION DIS,TERROR,BETA,DINIT,D,AUX(4,100)
      LOGICAL FTIME,EVEN
      COMMON LPT,NRD,BETA(100),DINIT(100),NTRAC
      COMMON /EXTRA/D
C                           IF THIS IS THE FIRST TIME CALLING THEN COMPUTES
C                           VECTOR B AND STORES IN BETA(*).
      IF(.NOT.FTIME)GOTO 620
      DO 610 I=1,N
      BETA(I)=0.D0          0.0D0
      DO 605 J=1,N
      IF(I.EQ.J) GOTO 604
      BETA(I)=BETA(I)+2.D0/J
      GOTO 605
  604 BETA(I)=BETA(I)+D*2.D0/J
  605 CONTINUE
      BETA(I)=BETA(I)/D
  610 CONTINUE
C                           COMPUTE Z=H*Z+B.
  620 DIS=0.D0
      DO 630 I=1,N
      AUX(INEXT,I)=0.D0
      DO 625 J=1,N
      IF(I.EQ.J) GOTO 625
      AUX(INEXT,I)=AUX(INEXT,I)+AUX(ICURET,J)
  625 CONTINUE
      AUX(INEXT,I)=-1.D0/D*AUX(INEXT,I)+BETA(I)
C                           FIND THE LONGEST DISTANCE.
      PTDIS=DABS(AUX(INEXT,I)-AUX(ICURET,I))
      IF(PTDIS.LE.DIS) GOTO 630
      DIS=PTDIS
  630 CONTINUE
      IF(NTRAC.LE.1) GOTO 640
      WRITE(6,635) (AUX(INEXT,J),J=1,N)
  635 FORMAT(' ',7E15.5)
  640 RETURN
      END
```

APPENDIX B

FORTRAN LISTING OF DYNAMIC DELTA-SQUARED PROCESS

```
$JOB
C  DEMONSTRATION TEST DRIVER PROGRAM FOR SUBROUTINE VAITK....
C  SOLVES THE DIRICHLET PROBLEM ON A RECTANGLE.
      DOUBLE PRECISION U,TA,TB,COSTH,SMAX,SMIN,DUMAX,COSIN,TOL,DX,DXSQ,
     CSRAW,S,XDX,DU,D,DABS,DSQRT,QABS,QSQRT,OMEGA,USG,UNORM,ALAM
      DIMENSION U(441),TA(441),TB(441)
      DATA METHD/2/, NPREP/0/, NPER/1/, COSTH/0.D0/, SMAX/50.D0/,
     CSMIN/0.D0/
      QABS(ARG)=DABS(ARG)
      QSQRT(ARG)=DSQRT(ARG)
C
      NPREP=0
C                            KW ...  LOGICAL UNIT NUMBER OF THE PRINTER
      KW=6
C
C  NHX IS THE NUMBER OF INCREMENTS IN THE X DIRECTION.
C  NHY IS THE NUMBER OF INCREMENTS IN THE Y DIRECTION.
C  THE INCREMENTS IN THE X AND Y DIRECTIONS ARE ASSUMED TO BE EQUAL.
C  (NHY+1) MAY NOT EXCEED THE DIMENSION OF THE ARRAY U(*).
      NHX=20
      NHY=20
C
C  TOL IS THE CONVERGENCE TOLERANCE ON THE CHANGE IN U(JK).
C  THE ERROR IN U(JK) MAY CONSIDERABLY EXCEED TOL IN MAGNITUDE.
      TOL=1.D-5
C                            MAXIT ...  MAX. NO. OF ITERATIONS ALLOWED
      MAXIT=300
C                            OMEGA ...  RELAXATION PARAMETER
      OMEGA=1.D0
C
C  INITIALIZE U(*). NUINT IS THE NUMBER OF POINTS, INCLUDING THE
C  TOP, LEFT, AND RIGHT BOUNDARIES BUT EXCLUDING THE BOTTOM BOUNDARY
C  AND THE BOTTOM CORNERS.
      NXPLU=NHX+1
      NUINT=NXPLU*NHY
      DO 1 J=1,NUINT
      TB(J)=1.D0
    1 U(J)=1.D0
C
C  SET THE BOUNDARY VALUES INCLUDING THE CORNERS. THE CORNERS ARE
C  NOT USED IN THE ITERATIONS. HOWEVER, THE TOP CORNERS MUST BE
C  INITIALIZED IN ORDER TO AVOID UNDEFFINED ELEMENTS IN THE VECTOR U(*)
C  WHEN SUBROUTINE VAITK IS CALLED.
      DO 2 J=1,NXPLU
C                            TOP BOUNDARY, INCLUDING CORNERS....
      U(J)=0.D0
C                            BOTTOM BOUNDARY, INCLUDING CORNERS....
      JBOT=NUINT+J
    2 U(JBOT)=0.D0
      DO 3 J=2,NHY
C                            LEFT BOUNDARY, EXCLUDING CORNERS....
      JLEFT=(J-1)*NXPLU+1
      U(JLEFT)=0.D0
C                            RIGHT BOUNDARY, EXCLUDING CORNERS....
      JRITE=JLEFT+NHX
    3 U(JRITE)=0.D0
C                            PRINT THE HEADING AND FINISH INITIALIZING.
      WRITE(KW,7)
    7 FORMAT(/1H1,5H ITER,5X,6H DUMAX,12X,6H UNORM,2X,6H KOUNT,4X,
     &6H COSIN,6X,5H SRAW,7X,2H S,10X,4H XDX,12X,5H DXSQ,8X,5H ALAM/1H )
      ITER=1
      DUMAX=1.D35
      KOUNT=0
      KPER=0
C
C                            BEGIN THE NEXT ITERATION.
    6 CONTINUE
      USQ=0.D0
      XDX=0.D0
      DXSQ=0.D0
      DO 100 J=I,NUINT
      USQ=USQ+U(J)**2
      DX=U(J)-TB(J)
      XDX=XDX+U(J)*DX
  100 DXSQ=DXSQ+DX*DX
      UNORM=QSQRT(USQ)
      DU=QSQRT(DXSQ)
      ALAM=-XDX/DXSQ
      CALL VAITK (U,NUINT,TA,TB,METHD,NPREP,NPER,KOUNT,KPER,
     X     COSTH,SMAX,SMIN,    COSIN,SRAW,S)
      IF(COSIN)13,14,13
   14 WRITE(KW,8)ITER,DUMAX,UNORM,KOUNT,COSIN,SRAW
    8 FORMAT(1X,I4,E16.5,E16.5,I5,F14.7,F10.6,F10.6,2E16.5,F12.3)
      GO TO 15
   13 WRITE(KW,8)ITER,DUMAX,UNORM,KOUNT,COSIN,SRAW,S,XDX,DXSQ,ALAM
C  COMPUTE THE NEW VECTOR ITERATE, U(*). FIRST, INITIALIZE DUMAX.
   15 DUMAX=0.
C                            LOOP OVER THE ROWS OF INTERNAL POINTS.
      DO 4 J=2,NHY
      KA=(J-1)*NXPLU
C                            LOOP OVER THE POINTS IN THE ROW, UGS IS
C                                 THE NEW GAUSS-SEIDEL VALUE.
      DO 5 K=2,NHX
      JK=KA+K
C                            U(JPLUK) ... THE POINT BELOW U(JK)
      JPLUK=JK+NXPLU
C                            U(JMIUK) ... THE POINT ABOVE U(JK)
      JMIUK=JK-NXPLU
      UGS=(N(JK-1)+U(JK+1)+U(JPLUK)+U(JMIUK))/4.DO
      DU=UGS-U(JK)
      IF(QABS(DU)-QABS(DUMAX))5,5,9
    9 DUMAX=DU
    5 U(JK)=U(JK)+OMEGA*DU
    4 CONTINUE
C                            END OF ITERATION.  TEST FOR CONVERGENCE.
      IF(DABS(DUMAX)-TOL)11,11,10
   10 ITER=ITER+1
      IF(ITER-MAXIT)6,6,11
   11 STOP
      END
      SUBROUTINE VAITK (X,NX,   TA,TB,    METHD,NPREP,NPER,
     *                  KOUNT,KPER,   COSTH,SMAX,SMIN,   COSIN,SRAW,S)
C
C  VAITK 3.1        A.N.S.I. STANDARD BASIC FORTRAN        JULY 1981
C  SUBROUTINE VAITK PERFORMS DYNAMIC VECTOR AITKEN EXTRAPOLATION,
C  IN AN ATTEMPT TO ACCELERATE THE CONVERGENCE OF A GIVEN SLOWLY
C  CONVERGENT VECTOR ITERATION SCHEME.
C  J. P. CHANDLER, COMPUTER SCIENCE DEPT., OKLAHOMA STATE UNIVERSITY
C
C  A. JENNINGS, J.INST.MATH.APPLIC. 8 (1971) 99-110
C  E. F. BOYLE AND A. JENNINGS, INT.J.NUM.METH.ENG. 7 (1974) 232-235
C  H. K. CHENG AND M. M. HAFEZ, PAGES 101-122 IN
C       -PADE APPROXIMANTS METHOD AND ITS APPLICATIONS TO MECHANICS-,
C       EDITED BY H. CABANNES (SPRINGER-VERLAG 1976)
C  E. GEKELER, MATHEMATICS OF COMPUTATION 26 (1972) 427-436
C  C. BREZINSKI AND A. C. RIEU, MATH. OF COMP. 28 (1974) 731-741
C  C. BREZINSKI, COMPUTING 14 (1975) 205-211
C  P. WYNN, MATH. OF COMP. 16 (1962) 301-322
C
C  INPUT QUANTITIES.....  X(*),NX,METHD,NPREP,NPER,
C                              (KOUNT,KPER),      COSTH,SMAX,SMIN
C  OUTPUT QUANTITIES....  COSIN,SRAW,S,X(*)
C  COUNTERS.............  KOUNT,KPER
C  SCRATCH ARRAYS.......  TA(*),TB(*)
C
C     X(*)     --  INPUT VECTOR ITERATE.  ALSO RETURNS THE EXTRAPOLATED
C                       OUTPUT VECTOR.
C     NX       --  NUMBER OF COMPONENTS IN THE VECTORS X(*), TA(*),
C                       AND TB(*)
C     TA(*)    --  SCRATCH VECTOR OF NX COMPONENTS
C     TB(*)    --  SCRATCH VECTOR OF NX COMPONENTS
C     METHD    --  =1 TO USE THE -FDM- METHOD OF JENNINGS,
C                  =2 TO USE THE -SDM- METHOD (THIS IS THE FIRST STAGE
C                       OF THE VECTOR EPSILON ALGORITHM OF WYNN),
C                  =3 TO USE AN -SDN- (SECOND DIFFERENCE NORM) METHOD,
C                  =4 TO USE A -DFDN- (DIFFERENCE OF FIRST DIFFERENCE
C                       NORMS) METHOD
C                  (SUGGESTION ...  TRY METHD=3, AND PERHAPS 2 AND 4)
C     NPREP    --  NUMBER OF ITERATES DISCARDED BEFORE EACH
C                       EXTRAPOLATION BEGINS
C                  (SUGGESTION ...  SET NPREP=0, OR PERHAPS 1 OR 2)
C     NPER     --  =1 IF THE X(*) ITERATES ARE THOUGHT TO BE CONVERGING
C                       ALONG A LINE, EITHER MONOTONICALLY OR
C                       ALTERNATING IN DIRECTION,
C                  =2 IF THE X(*) ITERATES ARE THOUGHT TO BE ZIGZAGGING
C                       WITH A PERIOD OF TWO ITERATIONS, ETC.
C                  (USE THE SMALLEST VALUE OF NPER SUCH THAT COSIN
C                       APPROACHES +1 OR -1.  NPER=2 GIVES THE
C                       -SQUARED- EXTRAPOLATION METHODS OF JENNINGS.)
C     KOUNT    --  ITERATION COUNTER
C     KPER     --  COUNTER FOR ZIGZAGGING
C     COSTH    --  EXTRAPOLATION IS DONE ONLY IF THE MAGNITUDE OF
C                       COSIN (SEE BELOW) IS .GE. COSTH
C                       (SUGGESTION...  SET COSTH=.95 OR LARGER)
C     SMAX     --  UPPER LIMIT ON THE EXTRAPOLATION FACTOR (SEE BELOW)
C     SMIN     --  LOWER LIMIT ON THE EXTRAPOLATION FACTOR
C     COSIN    --  RETURNS THE COSINE OF THE ANGLE BETWEEN THE TWO
C                       FIRST DIFFERENCE VECTORS
C     SRAW     --  RETURNS THE RAW VALUE OF THE EXTRAPOLATION FACTOR S,
C                       BEFORE COSTH, SMAX, AND SMIN ARE APPLIED
C     S        --  RETURNS THE VALUE OF S ACTUALLY USED
C
C  THE USER CALLS VAITK REPEATEDLY.  THE X(*) VECTORS ARE SUCCESSIVE
C  VECTORS FROM SOME ITERATION SCHEME HAVING ROUGHLY LINEAR CONVERGENCE.
C  THE COUNTERS KOUNT AND KPER MUST BE SET TO ZERO BEFORE THE FIRST
C  CALL TO VAITK FOR A GIVEN PROBLEM, AND NOT CHANGED UNTIL THE NEXT
C  PROBLEM IS TO BE STARTED, EXCEPT AS NOTED BELOW.
C  ON CERTAIN CALLS TO VAITK, THE VECTOR X(*) WILL BE EXTRAPOLATED
C  INSIDE OF VAITK TO AN ESTIMATE OF THE LIMIT VECTOR TOWARD WHICH
C  THE ITERATES X(*) HAVE BEEN CONVERGING.  THIS IS DONE BY ADDING
C  TO X(*) A SCALAR MULTIPLE S OF THE RESULTANT CHANGE IN X(*)
C  DURING THE PREVIOUS NPER ITERATIONS.
```

```
C  IF AN ITERATION DIVERGES, IT IS PROBABLY BEST TO FORCE CONVERGENCE
C  BY APPLYING UNDER-RELAXATION TO IT, AND THEN APPLY VAITK TO THIS
C  CONVERGENT ITERATION.
C
C  MONOTONIC BEHAVIOR OF THE ITERATION IS ASSOCIATED WITH VALUES OF
C  COSIN THAT ARE .GT. ZERO, AND OSCILLATORY BEHAVIOR WITH VALUES THAT
C  ARE .LT. ZERO.  IN ADDITION, WHEN THE VALUE OF COSIN IS NEAR 1.0
C  OR NEAR -1.0, THE VALUE OF S INDICATES THE BEHAVIOR OF SETS OF
C  NPER CONSECUTIVE ITERATIONS, AS FOLLOWS......
C                 S .LE. -1.0          MONOTONIC DIVERGENCE
C       -1.0 .LT. S .LT. -0.5          OSCILLATORY DIVERGENCE
C                 S .EQ. -0.5          OSCILLATION
C       -0.5 .LT. S .LT.  0.0          OSCILLATORY CONVERGENCE
C        0.0 .LE. S                    MONOTONIC CONVERGENCE
C  WHEN COSIN IS NEAR 1.0 OR -1.0, THE FIRST DIFFERENCE VECTOR FOR
C  SETS OF NPER CONSECUTIVE ITERATIONS IS APPROXIMATELY MULTIPLIED
C  BY S/(S+1) ON EACH SUCCESSIVE SET.
C
C  THE PURPOSE OF SMAX AND SMIN IS TO PREVENT A WILD VALUE OF S FROM
C  CAUSING AN ABSURD EXTRAPOLATION.  FOR A SEQUENCE THAT IS KNOWN
C  TO BE NONDIVERGENT, THE USER SHOULD PROBABLY SET SMIN=-0.5 .
C  SMAX MIGHT BE SET TO 10., OR 20., OR 50., OR ....
C
C  TO IMPLEMENT DOUBLE PRECISION ACCUMULATION OF INNER PRODUCTS,
C  ACTIVATE THE FIRST DOUBLE PRECISION STATEMENT BELOW.
C  TO IMPLEMENT COMPLETE DOUBLE PRECISION ARITHMETIC, ACTIVATE ALSO
C  THE SECOND DOUBLE PRECISION STATEMENT AND THE DOUBLE PRECISION
C  FORM OF THE STATEMENT FUNCTIONS BELOW.
      DOUBLE PRECISION DOT,DXLSQ,DXSQ,DXDDX,DDXSQ,DXOLD,DX,DDX
      DOUBLE PRECISION X,TA,TB,COSTH,SMAX,SMIN,COSIN,SRAW,S,
     X   ARG,QABS,DABS,QSQRT,DSQRT,RZERO,
     X   TEMP,SXLSQ,SXSQ,SXDDX,SDXSQ,SXOLD,SX,DENOM
C
      DIMENSION X(441),TA(441),TB(441)
C
C     QABS(ARG)=ABS(ARG)
C     QSQRT(ARG)=SQRT(ARG)
      QABS(ARG)=DABS(ARG)
      QSQRT(ARG)=DSQRT(ARG)
C
      RZERO=0.
      COSIN=RZERO
      SRAW=RZERO
      S=RZERO
      IF(KOUNT-NPREP)10,40,20
   10 KOUNT=KOUNT+1
      GO TO 350
   20 IF(KPER-(NPER-1))30,40,40
   30 KPER=KPER+1
      GO TO 350
   40 KPER=0
      KOUNT=KOUNT+1
      IF(KOUNT-(NPREP+2))290,330,50
C
C                            ACCUMULATE ALL INNER PRODUCTS.
   50 DOT=RZERO
      DXLSQ=RZERO
      DXSQ=RZERO
      DXDDX=RZERO
      DDXSQ=RZERO
      DO 60 J=1,NX
        DXOLD=TB(J)-TA(J)
        DX=X(J)-TB(J)
        DDX=DX-DXOLD
        DOT=DOT+DXOLD*DX
        DXLSQ=DXLSQ+DXOLD*DXOLD
        DXSQ=DXSQ+DX*DX
        DXDDX=DXDDX+DX*DDX
   60   DDXSQ=DDXSQ+DDX*DDX
      SXLSQ=DXLSQ
      SXSQ=DXSQ
      SXDDX=DXDDX
      SDXSQ=DDXSQ
```

```
C                                    COMPUTE COSIN.
      SXOLD=QSQRT(SXLSQ)
      SX=QSQRT(SXSQ)
      DENOM=SXOLD*SX
      IF(DENOM)310,310,70
   70 TEMP=DOT
      COSIN=TEMP/DENOM
C                                    SELECT THE METHOD AND COMPUTE SRAW.
      IF(METHD-2)90,110,80
   80 IF(METHD-4)130,170,170
C
C                                    METHD=1 ...   S = -(DX,DX)/(DDX,DX)
   90 IF(SXDDX)100,310,100
  100 SRAW=-SXSQ/SXDDX
      GO TO 210
C                                    METHD=2 ...   S = -(DX,DDX)/(DDX,DDX)
  110 IF(SDXSQ)120,310,120
  120 SRAW=-SXDDX/SDXSQ
      GO TO 210
C                                    METHD=3 ...   S = SIGN*NORM(DX)/NORM(DDX)
  130 IF(SDXSQ)140,310,140
  140 SRAW=SX/QSQRT(SDXSQ)
      IF(COSIN)160,150,150
  150 IF(SXOLD-SX)160,210,210
  160 SRAW=-SRAW
      GO TO 210
C                                    METHD=4 ...
C                                      S = -NORM(DX)/(NORM(DX)-SIGN*NORM(DXOLD))
  170 IF(COSIN)180,190,190
  180 SXOLD=-SXOLD
  190 DENOM=SX-SXOLD
      IF(DENOM)200,310,200
  200 SRAW=-SX/DENOM
C                                    TEST FOR SUFFICIENT COLINEARITY.
  210 IF(QABS(COSIN)-COSTH)310,220,220
C                                    APPLY THE CONSTRAINTS SMAX AND SMIN.
C
  220 S=SRAW
      IF(S-SMAX)240,240,230
  230 S=SMAX
  240 IF(S-SMIN)250,260,260
  250 S=SMIN
  260 IF(S)270,310,270
C                                    EXTRAPOLATE.
  270 DO 280 J=1,NX
  280    X(J)=X(J)+S*(X(J)-TB(J))
      KOUNT=1
C                                    SAVE X(*).
  290 DO 300 J=1,NX
  300    TA(J)=X(J)
      GO TO 350
C                                    SHIFT THE STORED ITERATES BACK ONE PLACE.
  310 DO 320 J=1,NX
  320    TA(J)=TB(J)
C                                    SAVE X(*).
  330 DO 340 J=1,NX
  340    TB(J)=X(J)
C
  350 RETURN
      END
```

APPENDIX C

FORTRAN LISTING OF LOW-DEGREE GENERALIZED SECANT METHODS

```
$JOB
C     DENOMSTRATION TEST PROGRAM FOR SUBROUTINE EXPO.
C     SUBROUTINE EXPO IS REFERED TO
C     D.G. ANDERSON, "ITERATIVE PROCEDURES FOR NONLINEAR INTEGRAL
C     EQUATIONS." JOURNAL ASSOCIATION FOR COMPUTING MACHINERY,
C     VOL.12 NO.4 (OCTOBER, 1965), 547-560.
C
C     FLOWCHART:
C               --------
C                 MAIN   INPUT DATA
C               --------
C               --------
C                 EXPO   PERFORM LOW-DEGREE GENERALIZED SECANT METHODS
C               --------
C     --------            --------
C        FE                 HIGHD
C     --------            --------
C     COMPUTE TEST        COMPUTE PARAMETERS OF GENERALIZED METHODS
C     FUNCTION            WHEN THE DEGREE IS BIGGER THAN 1
C
C  **************      VARIABLE REFERENCE     *************************
C     B                RELAXATION PARAMETER (0 < B <= 1)
C     DINIT(*)         INITIAL VALUES OF THE VECTOR'S COMPONENTS
C     LPT              LOGICAL UNIT NUMBER OF THE OUTPUT
C     METHOD           =1 EXTRAPOLATION METHOD
C                      =2 RELAXATION METHOD
C     M                THE DEGREE OF GENERALIZED SECANT METHODS
C                      TRY 1 OR 2. THEY ARE MOST USEFUL, AND BETTER NOT
C                      EXCESS 5.( ONLY 1 TO 5 CAN BE USED IN THIS
C                      PROGRAM; OTHERWISE NEED CHANGE THE SIZE OF
C                      DIMENSION )
C     MMAX             THE MAXIMUM NUMBER OF ITERATIONS ALLOWED
C     N                THE NUMBER OF VECTOR'S COMPONENTS
C     NRD              LOGICAL UNIT NUMBER OF THE INPUT
C     NTRAC            TRACE PARAMETER
C                      =0 PRINT THE FINAL RESULT
C                      =1 PRINT THE RESULT OF EACH ITERATION AND THE
C                         FINAL RESULT
C     TOL              TOLERANCE
C     WEIGHT(*)        THE WEIGHTS OF INNDER PRODUCT
C                      =1 UNIT METRIC
C                      OTHERWISE; RESIDUAL METRIC
C
      DOUBLE PRECISION DINIT(100),TOL,BETA,B,D,WEIGHT
      COMMON LPT,NRD,BETA(100),METHOD,M,WEIGHT(100),NTRAC
      COMMON /EXTRA/D
      DATA LPT/6/, NRD/5/
      METHOD=1
      M=4
      NTRAC=1
      D=2.5D1
C     D=1.5D1
      N=20
      DO 10 I=1,N
C  10 WEIGHT(I)=1.D0
   10 WEIGHT(I)=0.1D0*I
      DO 20 I=1,N
   20 DINIT(I)=1.D0
      MMAX=10
      TOL=1.D-5
      B=1.D0
      WRITE(LPT,42) METHOD,M
   42 FORMAT('1','METHOD=',I3,5X,'M=',I3,5X,'UNIT METRIC'//1X,
     C14HINITIAL VECTOR)
C  42 FORMAT('1','METHOD=',I3,5X,'M=',I3,5X,'RESIDURA L METRIC'// ,
C    C1X,'INITIAL VECTOR')
      WRITE(LPT,43) (DINIT(I),I=1,N)
   43 FORMAT(' ',7E15.5)
      CALL EXPO(N,MMAX,DINIT,TOL,B)
      STOP
      END
      SUBROUTINE EXPO(N,MMAX,DINIT,TOL,B)
C
C     THIS SUBROUTINE PERFORMS THE LOW-DEGREE GENERALIZE SECANT
C     METHODS.
C
C  **************       INPUT     ***********************************
C     B                 RELAXATION PARAMETER (0 < B <= 1)
C     DINIT(*)          INITIAL VALUES OF THE VECTOR'S COMPONENTS
C     MMAX              THE MAXIMUM NUMBER OF ITERATIONS ALLOWED
C     N                 THE NUMBER OF VECTOR'S COMPONENTS
C
C  **************       OUTPUT    **********************************
C     NC                ITERATION COUNTER
C     X(*)              THE VECTOR OF N COMPONENTS
C
      DOUBLE PRECISION X(2,100),Y(2,100),R(2,100),TOL,B,BETA,WEIGHT,
     CTHETA,U(100),V(100),TEMP,TP,SVX(5,100),SVY(5,100),SVR(5,100),
     CSVT(5),DINIT(100)
      DIMENSION INDEX(5)
      LOGICAL FTIME
      COMMON LPT,NRD,BETA(100),METHOD,M,WEIGHT(100),NTRAC
      COMMON /SAVE/SVX,SVY,SVR,SVT,INDEX
C                            SET THE INDEX OF CURRENT VECTOR L=1, AND
C                            AND THE INDEX OF NEXT VECTOR L1=2.
      FTIME=.TRUE.
      INDEX(1)=1
      NC=0
      L=1
      L1=2
      IN=1
      DO 100 I=1,N
  100 X(1,I)=DINIT(I)
C                            BEGIN THE NEXT ITERATION.
  105 NC=NC+1
      IF(NC.GT.1) FTIME=.FALSE.
      IF(NC.GT.MMAX) GOTO 160
      CALL FE(X,Y,R,N,IN,FTIME)
      IF(NC.GT.1) GOTO 115
      DO 111 I=1,N
      X(2,I)=Y(1,I)
      SVX(1,I)=X(1,I)
      SVY(1,I)=Y(1,I)
      S\R(1,I)=R(1,I)
  111 CONTINUE
      IF(NTRAC.LT.1) GOTO 113
      WRITE(LPT,151) NC
      WRITE(LPT,152) (X(2,I),I=1,N)
  113 IN=2
      GOTO 105
  115 TEMP=0.D0
      TP=0.D0
C                            SELECT THE METHOD.
C                            METHOD=1
C                            Y(N)=G(X(N))
C                            R(N)=Y(N)-X(N)
C                            THETA (N)=(R(N),R(N)-R(N-I))/
C                                 J    (R(N)-R(N-I),R(N)-R(N-I))
C                                  FOR I=1,2,...,M
C                            U(N)=X(N)+THETA (N)*(X(N-J)-X(N))
C                                           J
C                            V(N)=Y(N)+THETA (N)*(Y(N-J)-X(N))
C                                           J
C                                  FOR J=1,2,...,M
C                            X(N+1)=U(N)+B*(V(N)-U(N)), 0<B<=1
      IF(METHOD.GE.2) GOTO 141
      IF(M.LE.1) GOTO 120
      CALL HIGHD(NC,L,L1,X,Y,R,U,V,N,ITAG)
      IF(NC.LE.M) GOTO 120
      GOTO 131
  120 DO 121 I=1,N
      THETA=R(L1,I)-R(L,I)
      TEMP=TEMP+THETA*R(L1,I)*WEIGHT(I)
      TP=TP+THETA*THETA*WEIGHT(I)
  121 CONTINUE
```

```
      THETA=TEMP/TP
  125 DO 130 I=1,N
      U(I)=X(L1,I)+THETA*(X(L,
      V(I)=Y(L1,I)+THETA*(Y(L,I)-Y(L1,I))
  130 CONTINUE
  131 DO 135 I=1,N
  135 X(L,I)=U(I)+B*(V(I)-U(I))
      GOTO 148
C                             METHOD=2
C                             Y(N)=G(X(N)),  R(N)=Y(N)-X(N)
C                             THETA(N)=(Y(N)-Y(N-I),R(N)-R(N-I))
C                                      (R(N)-R(N-1),R(N)-R(N-I))/
C                             U(N)=Y(N)+THETA(N)*R(N)
C                             V(N)=Y(N-1)+THETA*R(N-1)
C                             X(N+1)=U(N)
  141 DO 145 I=1,N
      THETA=R(L1,I)-R(L,I)
      TEMP=TEMP+THETA*(Y(L,I)-Y(L1,I))
      TP=TP+THETA*THETA
  145 CONTINUE
      THETA=TEMP/TP
      DO 147 I=1,N
  147 X(L,I)=Y(L1,I)+THETA*R(L1,I)
  148 IF(NTRAC.LT.1) GOTO 153
      WRITE(LPT,151)NC
  151 FORMAT(' ','ITERATION',I3)
      WRITE(LPT,152) (X(L,I),I=1,N)
  152 FORMAT(' ','X=',10X,7E15.5)
C                             TEST CONVERGENCE.
  153 TEMP=0.D0
      DO 155 I=1,N
      TP=DABS(X(L,I)-X(L1,I))
      IF(TP.LE.TOL) GOTO 155
      TEMP=TEMP+1.D0
  155 CONTINUE
      IF(TEMP.LE.0.D0) GOTO 170
C                             SWAP THE INDICES OF VECTORS, THEN GO BACK
C                             FOR TO THE NEXT ITERATION.
      L1=L
      L=3-L1
      IN=L1
      GOTO 105
  160 WRITE(LPT,161)
  161 FORMAT(' ','THE NUMBER OF ITERATIONS BIGGER THAN THE',
     C' MAXIMUM NUMBER OF THE ITERATIONS ALLOWED.')
      GOTO 190
  170 WRITE(LPT,171)(X(L,I),I=1,N)
  171 FORMAT(' ','FINAL RESULT',7E15.5)
  190 RETURN
      END
      SUBROUTINE HIGHD(NC,L,L1,X,Y,R,U,V,N,ITAG)
C
C     THIS SUBROUTINE CALCULATES THETA   OF DEGREE J THAT IS MORE
C                                     J
C     THAN 1. LINEAR SYSTEM A * SVT = B CAN BE SOLVED BY USING
C     GAUSS ELIMINATION. SEE
C     CONTE, S.D. AND DE BOOR, CARL. ELEMENTARY NUMERICAL ANALYSIS:
C     AN ALGORITHMIC APPROACH. NEW YORK: MCGRAW-HILL, 1980.
C
      DOUBLE PRECISION X(2,100),Y(2,100),R(2,100),U(100),V(100),
     CSVX(5,100),SVY(5,100),SVR(5,100),SVT(5),A(5,5),B(5),D(5),ROWMAX,
     CCOLMAX,TEMP,SUM,WEIGHT,BETA
      DIMENSION INDEX(5),IPIVOT(5)
      COMMON LPT,NRD,BETA(100),METHOD,M,WEIGHT(100),NTRAC
      COMMON /SAVE/SVX,SVY,SVR,SVT,INDEX
      ITAG=1
      IF(NC.LE.M) GOTO 60
C                            INITIALIZE A(*,*) AND B(*).
      DO 5 I=1,M
      INI=INDEX(I)
      DO 3 J=1,M
      INJ=INDEX(J)
      A(I,J)=0.D0
      DO 2 IJ=1,N
    2 A(I,J)=A(I,J)+(R(L,IJ)-SVR(INI,IJ))*(R(L,IJ)-SVR(INJ,IJ))
    3 CONTINUE
      B(I)=0.D0
      DO 4 IJ=1,N
    4 B(I)=B(I)+R(L,IJ)*(R(L,IJ)-SVR(INI,IJ))
    5 CONTINUE
      M1=M-1
C                            INITIALIZE IPIVOT(*) AND D(*).
      DO 8 I=1,M
      IPIVOT(I)=I
      ROWMAX=0.D0
      DO 6 J=1,M
    6 ROWMAX=DMAX1(ROWMAX,DABS(A(I,J)))
      IF(ROWMAX.NE.0.D0) GOTO 7
      ITAG=0
      ROWMAX=1.D0
    7 D(I)=ROWMAX
    8 CONTINUE
C                            FACTORIZATION.
      DO 20 K=1,M1
C                            DETERMINE PIVOT ROW AND THE ROW ISTAR.
      COLMAX=DABS(A(K,K))/D(K)
      ISTAR=K
      KP1=K+1
      DO 10 I=KP1,M
      TEMP=DABS(A(I,K))/D(I)
      IF(TEMP.LE.COLMAX) GOTO 10
    9 COLMAX=TEMP
      ISTAR=I
   10 CONTINUE
      IF(COLMAX.NE.0.D0) GOTO 12
      ITAG=0
      GOTO 20
   12 IF(ISTAR.LE.K) GOTO 16
C                            MAKE K THE PIVOT ROW BY INTERCHANGING IT
C                            WITH THE CHOSEN ROW ISTAR.
      ITAG=-ITAG
      I=IPIVOT(ISTAR)
      IPIVOT(ISTAR)=IPIVOT(K)
      IPIVOT(K)=I
      TEMP=D(ISTAR)
      D(ISTAR)=D(K)
      D(K)=TEMP
      DO 15 J=1,M
      TEMP=A(ISTAR,J)
      A(ISTAR,J)=A(K,J)
      A(K,J)=TEMP
   15 CONTINUE
```

```
C                                  ELIMINATE X(K) FROM ROWS K+1, ..., N.
   16 DO 19 I=KP1,M
      A(I,K)=A(I,K)/A(K,K)
      DO 18 J=KP1,M
   18 A(I,J)=A(I,J)-A(I,K)*A(K,J)
   19 CONTINUE
   20 CONTINUE
C                                  SOLVE A *SVT = B FOR X BY SOLVING
C                                  TWO TRIANGULAR SYSTEMS.
      IF(A(M,M).EQ.0.D0) ITAG=0
      IF(ITAG.EQ.0) GOTO 70
      IP=IPIVOT(1)
      SVT(1)=B(IP)
      DO 25 I=2,M
      SUM=0.D0
      I1=I-1
      DO 24 J=1,I1
   24 SUM=A(I,J)*SVT(J)+SUM
      IP=IPIVOT(I)
   25 SVT(I)=B(IP)-SUM
      SVT(M)=SVT(M)/A(M,M)
      DO 40 I=1,M1
      K=M-I
      KP1=K+1
      SUM=0.D0
      DO 35 J=KP1,M
   35 SUM=A(K,J)*SVT(J)+SUM
      SVT(K)=(SVT(K)-SUM)/A(K,K)
   40 CONTINUE
C                                  COMPUTE U(*) AND V(*).
      DO 41 I=1,N
      U(I)=X(L1,I)
      V(I)=Y(L1,I)
      DO 41 J=1,M
      INJ=INDEX(J)
      U(I)=U(I)+SVT(J)*(SVX(INJ,I)-X(L1,I))
      V(I)=V(I)+SVT(J)*(SVY(INJ,I)-Y(L1,I))
   41 CONTINUE
C                                  DELETE THE (NC-J)TH VECTOR OF X, Y
C                                  AND R, AND ADD THE CURRENT VECTOR OF
C                                  X, Y AND R TO SAVE BOX.
      DO 47 I=1,M
      IF(INDEX(I).EQ.1) GOTO 46
      INDEX(I)=INDEX(I)-1
      GOTO 47
   46 INDEX(I)=M
   47 CONTINUE
      GOTO 61
   60 J=MOD(NC,M)
      IF(J.EQ.0) J=M
      INDEX(J)=J
   61 DO 65 I=1,N
      SVX(J,I)=X(L,I)
      SVY(J,I)=Y(L,I)
      SVR(J,I)=R(L,I)
   65 CONTINUE
   70 RETURN
      END
      SUBROUTINE FE(X,Y,R,N,IN,FTIME)
C
C     THIS TEST FUNCTION COMES FROM
C     D.G. ANDERSON, "ITERATIVE PROCEDURES FOR NONLINEAR INTEGRAL
C     EQUATIONS." JOURNAL OF ASSOCIATION FOR COMPUTING MACHINERY,
C     VOL.12 NO.4 (OCTOBER, 1965), 547-560.
C          A*Z=D*B WITH
C                       A(I,J)=D, I=J
C                             =1,  I¬=J
C          THE BASIC ITERATION IS
C          Z=H*Z+B WITH
C                       H(I,J)=0, I=J
C                             =-1/D, I¬=J.
C
      DOUBLE PRECISION X(2,100),Y(2,100),R(2,100),BETA,D,WEIGHT
      LOGICAL FTIME
      COMMON LPT,NRD,BETA(100),METHOD,M,WEIGHT(100),NTRAC
      COMMON /EXTRA/D
C                            IF THIS IS THE FIRST TIME CALLING THEN
C                            COMPUTE B AND STORE IN BETA(*).
      IF(.NOT.FTIME)GOTO 620
      DO 610 I=1,N
      BETA(I)=0.D0
      DO 605 J=1,N
      IF(I.EQ.J) GOTO 604
      BETA(I)=BETA(I)+2.D0/J
      GOTO 605
  604 BETA(I)=BETA(I)+D*2.D0/J
  605 CONTINUE
      BETA(I)=BETA(I)/D
  610 CONTINUE
C                            COMPUTE Z=H*Z+B.
  620 DO 630 I=1,N
      Y(IN,I)=0.D0
      DO 625 J=1,N
      IF(I.EQ.J) 625
      Y(IN,I)=Y(IN,I)+X(IN,J)
  625 CONTINUE
      Y(IN,I)=-1.D0/D*Y(IN,I)+BETA(I)
      R(IN,I)=Y(IN,I)-X(IN,I)
  630 CONTINUE
      RETURN
      END
```

APPENDIX D

FORTRAN LISTING OF MODIFIED VECTOR E-ALGORITHM

```
$JOB
C     DENOMSTRATION TEST PROGRAM FOR DYNAMICAL VECTOR E-ALGORITHM.
C
C     SEE REFERENCES:
C     1. CHENG, H.K. AND HAFEZ, M.M. "CYCLIC ITERATIVE METHOD APPLIED
C        TO TRANSONIC FLOW ANALYSIS." LECTURE NOTES IN PHYSICS NO. 47
C        PADE APPROXIMANTS METHOD AND ITS APPLICATIONS TO MECHANICS.
C        ED. H. CABANNES. NEW YORK: SPRINGER-VERLAG, 1976, 101-121.
C     2. SHANKS, D. "NON-LINEAR TRANSFORMATIONS OF DIVERGENT AND
C        SLOWLY CONVERGENT SEQUENCES." J. MATH. PHYS., VOL.34(APR.1955),
C        1-42.
C     3. WYNN, P. "ON A DEVICE FOR COMPUTING THE EM(SN) TRANSFORMATION."
C        MATC, VOL.10(1956),91-96.
C     4. WYNN, P. "ACCELERATION TECHNIQUES FOR ITERATED VECTOR AND
C        MATRIX PROBLEMS." MATH. COMP., VOL.16(JULY 1962), 301-322.
C
C     FLOWCHART:
C                           ---------
C                             MAIN
C                           ---------
C                           ---------
C                            DYNAM
C                           ---------
C                     USE DYNAMIC E-ALGORITHM.
C               -------                       --
C               SAMINV                        FE
C               -------                       --
C    COMPUTE SAMELSON INVERSE.  COMPUTE THE VALUE OF EACH COMPONENT.
C
      DOUBLE PRECISION DELTA,DINIT(100),TEMP,BETA
      LOGICAL FTIME
      COMMON LPT,NRD,NTRAC,METHOD
      COMMON /EXTRA/BETA(100),FTIME
C                        LPT ... LOGICAL UNIT NUMBER OF THE OUTPUT
C                        NRD ... LOGICAL UNIT NUMBER OF THE INPUT
      DATA LPT/6/, NRD/5/
C                        METHOD ... METHOD PARAMETER
C                                   =1 D. V. E-ALGORITHM WITH S,M=0
C                                   =2 D. V. E-ALGORITHM WITH S=0, M=1
C                                   =3 D. V. E-ALGORITHM WITH S=0, M=0,1
      METHOD=1
C                        INITIALIZE DINIT(*). N IS THE NUMBER OF
C                        VECTOR'S COMPONENTS.
      N=20
      DO 10 I=1,N
   10 DINIT(I)=1.D0
C                        DELTA ... TOLERANCE
      DELTA=1.D-5
C                        MMAX ... THE MAX. NO. OF ITERATIONS ALLOWED
      MMAX=70
C                        NTRAC ... TRACE PARAMETER
C                                  =0 PRINT THE FINAL RESULT
C                                  =1 PRINT THE RESULT OF EACH ITERATE
C                                     AND THE FINAL RESULT
      NTRAC=1
C                        PRINT THE HEADING AND INITIAL VALUES.
      IF(METHOD.GE.1)20,30,40
   20 WRITE(LPT,21)
   21 FORMAT('1','DYNAMICAL VECTOR E-ALGORITHM WITH   S=0, M=0.',
     C//1X.'INITIAL VECTOR')
      GOTO 50
   30 WRITE(LPT,31)
   31 FORMAT('1','DYNAMICAL VECTOR E-ALGORITHM WITH   S=0, M=1.',
     C//1X,'INITIAL VECTOR')
      GOTO 50
   40 WRITE(LPT,41)
   41 FORMAT('1','DYNAMICAL VECTOR E-ALGORITHM WITH   S=0, M=0,1.,1X',
     C//1X,'INITIAL VECTOR')
   50 WRITE(LPT,51)  (DINIT(I),I=1,N)
   51 FORMAT(' ',7E15.5)
      CALL DYNAM(N,DELTA,DINIT,MMAX)
      STOP
      END
      SUBROUTINE DYNAM(N,DELTA,DINIT,MMAX)
C
C     THIS SUBROUTINE PERFORMS DYNAMICAL VECTOR E-ALGORITHM.
C
C  ****************         INPUT     ***********************************
C     N                     THE NUMBER OF VECTOR'S COMPONENTS
C     DELTA                 TOLERANCE
C     DINIT(*)              THE INITIAL VALUES OF VECTOR COMPONENTS
C     MMAX                  THE MAXIMUM NUMBER OF ITERATIONS ALLOWED
C
C  ****************         OUTPUT    ***********************************
C     AUX(*,*)              THE VECTOR OF N COMPONENTS
C
      DOUBLE PRECISION AUX(4,100),DELTA,DIS,DINIT(100),BETA
      LOGICAL FTIME
      COMMON LPT,NRD,NTRAC,METHOD
      COMMON /EXTRA/BETA(100),FTIME
C                           INITIALIZE FTIME, IT AND THE LABELS OF AUXILIARY
C                           STORAGE VECTORS.
      FTIME=.TRUE.
      IT=0
      IONE=1
      ITWO=2
      ITHREE=3
      IFOUR=4
      DO 100 I=1,N
  100 AUX(IONE,I)=DINIT(I)
C                           INCREASE IT BY ONE AND TEST IF IT BEYONDS MMAX.
  110 IT=IT+1
      IF(IT.GT.MMAX) GOTO 135
      IF(NTRAC.LT.1) GOTO 117
      WRITE(LPT,115) IT
  115 FORMAT('0','ITERATION:',I3)
      WRITE(LPT,116) (AUX(1,I),I=1,N)
  116 FORMAT(' ','X',5X,7E15.5/7X,7E15.5/7X,7E15.5)
  117 CALL FE(AUX,ITWO,IONE,N,DIS)
      IF(NTRAC.LT.1) GOTO 119
      WRITE(LPT,116)(AUX(2,I),I=1,N)
  119 IF(DIS.LE.DELTA) GOTO 131
      FTIME=.FALSE.
      CALL FE(AUX,ITHREE,ITWO,N,DIS)
      IF(NTRAC.LT.1) GOTO 122
      WRITE(LPT,116)(AUX(3,I),I=1,N)
  122 IF(DIS.LE.DELTA) GOTO 132
      IF(METHOD-2) 129,124,123
C                           METHOD=2 OR METHOD=3 AND IT=1.
C                           COMPUTE    (0)
C                                    E
C                                       0
C
C                                    (1)
C                                    E
C                                       0     (1)
C                                          E
C                                    (2)     1     (1)
C                                    E              E
C                                       0     (2)     2
C                                          E
C                                    (3)     1
C                                    E
C                                       0
C
C                           WITH   (M)       (M+1)        (M+1)     (M)
C                                E      =  E       +  1/E       - E
C                                 S+1       S-1           S         S.
  123 IF(IT.GT.1) GOTO 129
  124 CALL SAMINV(AUX.ITWO,ITHREE,IONE,N)
      CALL FE(AUX,IFOUR,ITHREE,N,DIS)
      IF(NTRAC.LT.1) GOTO 126
      WRITE(LPT,116)(AUX(4,I),I=1,N)
  126 IF(DIS.LE.DELTA) GOTO 134
      CALL SAMINV(AUX,ITHREE,IFOUR,IFOUR,N)
      CALL SAMINV(AUX,IONE,IFOUR,IONE,N)
      DO 128 I=1,N
  128 AUX(IONE,I)=AUX(ITHREE,I)+AUX(IONE,I)
C                             (1)
C                        USE E    TO BE NEW INITIAL VECTOR.
C                              2
      GOTO 110
C                        METHOD=1 OR METHOD=3 AND IT>1.
C                        COMPUTE   (0)
C                                 E
C                                   0     (0)
C                                        E
C                                  (1)     1     (0)
C                                 E             E
C                                   0     (1)     2
C                                        E
C                                  (2)     1
C                                 E
C                                   0
C
C                        WITH   (M)      (M+1)         (M+1)      (M)
C                              E     =  E       +  1/E         - E
C                               S+1      S-1            S          S.
  129 CALL SAMINV(AUX,IONE,ITWO,IONE,N)
      CALL SAMINV(AUX,ITWO,ITHREE,ITHREE,N)
      CALL SAMINV(AUX,IONE,ITHREE,ITHREE,N)
      DO 130 I=1,N
  130 AUX(IONE,I)=AUX(ITHREE,I)+AUX(ITWO,I)
C                        (0)
C                   USE E    TO BE NEW INITIAL VECTOR.
C                         2
      GOTO 110
  131 WRITE(LPT,133)
      WRITE(LPT,116) (AUX(2,I),I=1,N)
      GOTO 140
  132 WRITE(LPT,133)
  133 FORMAT('0','FINAL RESULT:')
      WRITE(LPT,116) (AUX(3,I),I=1,N)
      GOTO 140
  134 WRITE(LPT,116) (AUX(4,I),I=1,N)
      GOTO 140
  135 WRITE(LPT,136)
  136 FORMAT('0','THE NUMBER OF ITERATIONS BIGGER THEN THE',
     C' MAXIMUM NUMBER OF ITERATIONS ALLOWED.')
  140 RETURN
      END
      SUBROUTINE SAMINV(AUX,I1,I2,IOUT,LENGTH)
C
C     THIS SUBROUTINE DOES THE SAMELSON INVERSE. WHEN THE COMPONENTS
C     OF THE VECTPR ARE ALL REAL, THE SAMELSON INVERSE IS FORMED
C     MERELY BY DIVIDING THE VECTOR THROUGHOUT BY THE SUM OF THE
C     SQUARES OF IT COMPONENTS.
C
      DOUBLE PRECISION AUX(4,100),DENOM,HUGE,TEMP(100)
      INTEGER OUT
      DATA HUGE/1.D30/
C                            -1           2
C                           Y   = Y/||Y||
      DENOM=0.D0
      DO 620 I=1,LENGTH
      TEMP(I)=AUX(I2,I)-AUX(I1,I)
      DENOM=DENOM+TEMP(I)*TEMP(I)
  620 CONTINUE
      IF(DENOM.LE.0.D0) GOTO 660
      DO 630 I=1,LENGTH
  630 AUX(IOUT,I)=TEMP(I)/DENOM
      GOTO 670
  660 DO 665 I=1,LENGTH
  665 AUX(IOUT,I)=HUGE
  670 RETURN
      END
      SUBROUTINE FE(AUX,IRNEXT,ICURET,N,DIS)
C
C     THIS TEST FUNCTION COMES FROM
C     D.G. ANDERSON, "ITERATIVE PROCEDURES FOR NONLINEAR INTEGRAL
C     EQUATIONS." JOURNAL OF ASSOCIATION FOR COMPUTING MACHINERY,
C     VOL.12 NO.4 (OCTOBER, 1965), 547-560.
C          A*Z=D*B WITH
C                        A(I,J)=D, I=J
C                              =1, I¬=J
C          THE BASIC ITERATION IS
C          Z=H*Z+B WITH
C                        H(I,J)=0, I=J
C                              =-1/D, I¬=J.
C
      DOUBLE PRECISION DIS,BETA,DINIT,D,AUX(4,100),TEMP
      LOGICAL FTIME
      COMMON LPT,NRD,NTRAC,METHOD
      COMMON /EXTRA/BETA(100),FTIME
C                        IF THIS IS THE FIRST TIME CALLING THEN COMPUTES
C                        VECTOR B AND STORES IN BETA(*).
C     D=1.5D1
      D=2.5D1
      IF(.NOT.FTIME)GOTO 620
      DO 610 I=1,N
      BETA(I)=0.D0
      DO 605 J=1,N
      IF(I.EQ.J) GOTO 604
      BETA(I)=BETA(I)+2.D0/J
      GOTO 605
  604 BETA(I)=BETA(I)+D*2.D0/J
  605 CONTINUE
      BETA(I)=BETA(I)/D
  610 CONTINUE
C                        COMPUTE Z=H*Z+B.
  620 DIS=0.D0
      DO 630 I=1,N
      TEMP=0.D0
      DO 625 J=1,N
      IF(I.EQ.J) GOTO 625
      TEMP=TEMP+AUX(ICURET,J)
  625 CONTINUE
      TEMP=-1.D0/D*TEMP+BETA(I)
      PTDIS=DABS(TEMP-AUX(ICURET,I))
      AUX(IRNEXT,I)=TEMP
      IF(PTDIS.LE.DIS) GOTO 630
      DIS=PTDIS
  630 CONTINUE
      RETURN
      END
```

VITA 2

Mei-Hui Chen

Candidate for the Degree of

Master of Science

Thesis: A COMPARISON OF SOME VECTOR ACCELERATION TECHNIQUES

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Taipei, Taiwan, Republic of China, January 13, 1957, the daughter of Leang-Chyuan Chen and Shwu-Jen Wang Chen.

Education: Graduate from Taipei First Girls' Senior High School, July 1975; received Bachelor of Science in Psychology from National Chengchi University, Taipei, Taiwan, Republic of China, in June, 1979; completed requirements for the Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, in May, 1984.

Professional Experience: Research Assistant, Department of Psychology, National Chengchi University, June, 1979, to December, 1979; Research Assistant, Department of Psychiatry, Veterans General Hospital, January, 1980, to July, 1981.