A STUDY OF THE EPSILON ALGORITHM AND APPLICATIONS

By

HUI-WEN CHIANG

Bachelor of Law

Fu Jen University

Taipei, Taiwan

Republic of China

1978

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
December, 1986

A STUDY OF THE EPSILON ALGORITHM

AND APPLICATIONS

Thesis Approved:

J P Chandler
Thesis Adviser

Donald W Grace

Al Thoreson

Norman N. Durham
Dean of the Graduate College

PREFACE

This thesis describe Wynn and Shanks' epsilon algorithm and its implementations. The main objective of this study is to observe the characteristics of the epsilon algorithm and to formulate an update form of the epsilon algorithm to speed up iterative processes by reducing duplicated steps. The update epsilon algorithm has been tested on several sets of numerical problems and the results have been satisfactory.

The author wishes to express her sincere appreciation to her major adviser, Dr. John P. Chandler, for his lasting patience, constant guidance, and expertise throughout this study. Appreciation is also expressed to the other committee members, Dr. Donald W. Grace and Dr. Sharilyn A. Thoreson, for their cooperation and assistance. Gratitude is also extended to Sharon Steele for her outstanding clerical assistance.

Finally, special thanks is expressed to my parents and to my husband for their continued emotional support, moral encouragement and understanding throughout this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Techniques for accelerating the convergence of iterative processes have been developed almost a century. One of the main strategies of such techniques is to use some kind of transformation on the partial sums of a slowly convergent series to produce a transformed series which converges to the same limit as the original sequence, but faster.

Assume that we want to compute the sum of the series,

$$\begin{aligned} \pi/4 &= 1 - 1/3 + 1/5 - 1/7 + \ldots \\ &= \sum_{j=0}^{\infty} (-1) / (2j + 1). \end{aligned} \tag{1.1}$$

The series converges very slowly. Even after 500 terms there still occur changes in the third decimal. If we do not know that the limit is $\pi/4$, it is very difficult to find the limit to the required tolerance, unless we can do something besides simply adding up partial sums. This is what the acceleration methods are designed for.

There are several acceleration techniques available for speeding up the convergence of an iterative solution. These well-known techniques include Richardson integration, Romberg integration, Euler's method, power series methods, the Q-D algorithm, Aitken's method, the epsilon algorithm, etc. Most of these techniques work well in most of the problems and then fail miserably in some specific cases. However, the epsilon algorithm is considered a relatively simple but powerful technique for accelerating the convergence of slowly convergent sequences and inducing convergence in divergent sequences. The epsilon algorithm may also be used (i) to obtain useful results from divergent series and iterations, (ii) to obtain the limits of iterated vector and matrix sequences, (iii) to aid in the solution of differential and integral equations, (iv) to carry out numerical integration in a new way, (v) to extrapolate, (vi) to fit a curve to a polynomial or to a constant plus a sum of exponentials.

It is the purpose of this thesis to study the properties of the epsilon algorithm and, based on its recursive nature, to modify the epsilon algorithm for speeding up the iterative process in order to reduce some possible duplicate steps by saving previous results for further computation (see Chapter III). Furthermore, the discussion of the singular rule which was suggested by Wynn to overcome the instability of the epsilon algorithm is also included in this paper.

Since the transformations take a very important role in the epsilon algorithm, a brief historical overview, as well as Shanks' motivations of the transforms, and Wynn's epsilon table will be presented at the beginning of Chapter II. Following this will be the descriptions of the implementations and stability of the epsilon algorithm.

The motivation to update the epsilon algorithm will be presented in Chapter III. Meanwhile, the logic to update the epsilon algorithm will be described step by step. (The description of several numerical problems which have been tested both on the "original" and the "update" epsilon algorithm are then illustrated.) (The four sets of the numerical test problems are Fourier series, confluent hypergeometric function, Gauss-Seidel relaxation scheme, and Jacobi relaxation scheme. The comparison of the test results between the "original" epsilon algorithm and the "update" epsilon algorithm with respect to the storage requirements and the time requirements are made with respect to each of the four test problems.)

However, when using the formula of the epsilon algorithm it may sometimes occur that an entity is numerically ill-determined and causes all entities lying in a certain sector to become ill-determined too. P. Wynn's singular rule to overcome these misfortunes is studied and tested in Chapter IV.

In Chapter V, the epsilon algorithm is applied repeatedly in order to obtain a new series expansion in negative powers of n of the magnitude of the error in the partial sums of an infinite series.

The final conclusion of this thesis and the suggestion for further study are given in Chapter VI. Finally, all program listings are collected in the Appendices.

CHAPTER II

THE EPSILON ALGORITHM

Introduction and Historical Overview

The epsilon algorithm is a method of generating non-linear transforms for increasing the rate and expanding the domain of convergence of sequences. The family of non-linear epsilon transforms includes e_k, $e_k^{(m)}$, $\tilde{e}_k$, and e_d. The el transform has been developed by several well-known authors including Delauncy (1926) , Samuelson (1945), Shanks and Walton (1948), Hartree (1949), and Isakson (1949). However, more general discussions of the epsilon transforms and their applications were given by Shanks. But, because of the vast amount of time consumed and labor needed in evaluating the determinant of the entries, the use of these transformations was limited. In 1956, Wynn discovered an easier way to determine the entries of the epsilon table without any determinant calculations. Since then, the epsilon algorithm has become popular. In the next section a summary of Shanks' approach to obtain the transformations will be given.

Motivation of Shanks' Transforms

Let us define a typical numerical sequence A by

$$A = (A_r),\ r = 0,1,2,3,\ldots, \tag{2.1}$$

and draw a smooth curve through the discrete points which are plotted by A versus r. Figure 1 shows the graphs of some sequences that are defined as convergent, divergent, monotonic, and oscillatory sequences. By comparing common points in the sequences of the graph,

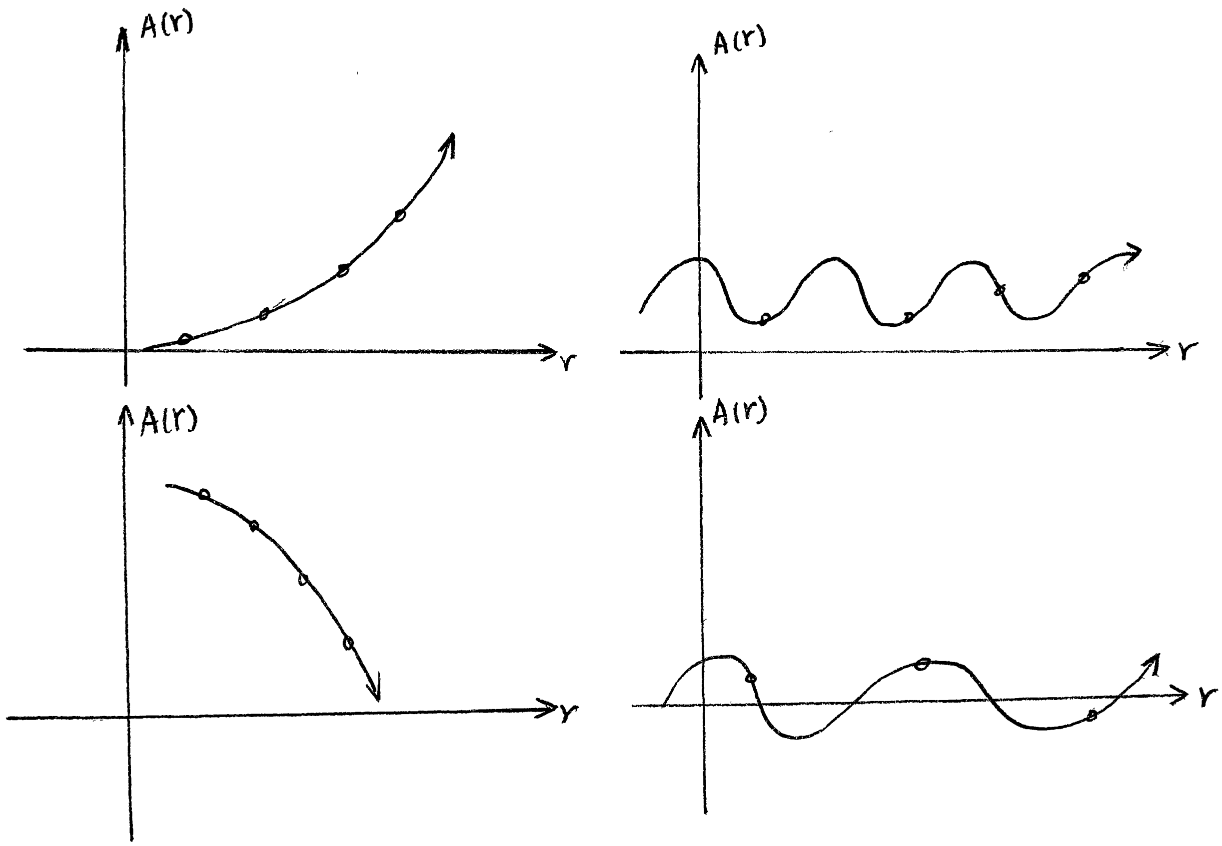


Figure 1: Graphs of the numerical sequences

Shanks devised a function of r in the form of

$$A_r = B + \sum_{i=1}^{k} X_i Q_i^r \qquad (Q_i \neq 1,0) \qquad (2.2)$$

to represent those sequences, where X_i is the spectrum of amplitudes, Q_i is the spectrum of ratios, and B as the "base". Here the prime concern is computing the base B. For if (A_r) is a mathematical transient i.e. if it satisfies (2.1) and if each ratio satisfies $|Q_i| < 1$ then clearly

$$B = \lim_{r \to \infty} A .$$

If (A_r) is a transient and one or more $|Q_i| >= 1$, A_r does not converge; Shanks said that "A_r diverges from B" and calls B the "anti-limit" of (A_r).

Many sequences which arise naturally in analysis are indeed mathematical transients of some finite order k. Other sequences that are of infinite order ($k = \infty$) we can in many cases say that (A_r) is nearly of Kth order for some K -- at least for r greater than some fixed N. Then by analogy with (2.1) Shanks determines a local Kth order base $B_{k,n}$ by solving the 2k+1 equations

$$A_r = B_{k,n} + \sum_{i=1}^{k} X_{i,n} \; Q_{i,n} , \qquad n-k <= r <= n+k, \; n>=k, \; (Q_{i,n} \neq 1,0)$$

or the 2k+1 quantities $B_{k,n}$, $X_{i,n}$, $Q_{i,n}$, (i=1,2,...,k). Algebraically the formula for $B_{k,n}$ is obtain by

$$B_{k,n} = \frac{\begin{vmatrix} A_{n-k} & \text{--------} & A_{n-1} & A_n \\ \Delta A_{n-k} & \text{--------} & \Delta A_{n-1} & \Delta A_n \\ \Delta A_{n-k+1} & \text{--------} & \Delta A_n & \Delta A_{n+1} \\ \vdots & & \vdots & \vdots \\ \Delta A_{n-1} & \text{--------} & \Delta A_{n+k-2} & \Delta A_{n+k-1} \end{vmatrix}}{\begin{vmatrix} 1 & \text{--------} & 1 & 1 \\ \Delta A_{n-k} & \text{--------} & \Delta A_{n-1} & \Delta A_n \\ \Delta A_{n-k+1} & \text{--------} & \Delta A_n & \Delta A_{n+1} \\ \vdots & & \vdots & \vdots \\ \Delta A_{n-1} & \text{--------} & \Delta A_{n+k-2} & \Delta A_{n+k-1} \end{vmatrix}} \tag{2.3}$$

where $\Delta A = A_{n+1} - A_n$, and this is Shanks' "K'th order transform of (A_r)".

The transforms may also be written in operator form:

$$B_{k,n} = e_k (A_n) . \tag{2.4}$$

where e_K is the nonlinear operator defined by the right hand side of the equation (2.3).

Followed by the first iteration transform e_K, the higher order iteration transforms can be gained by

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

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

(2.5)

From the above iteration the operator e_k was transformed. The "Kth order iterated transformation", is defined by:

$$e_k (A_0 ; A_1 ; A_2 ; A_3 ; \ldots) = A_0 ; B_{k,k} ; C_{k,2k} ; D_{k,3k} ; \ldots \tag{2.6}$$

and the operator e_d, the "diagonal transformation", is defined by:

$$e_d(A_n) = B_{n,n} .$$

From Shanks' definition of the epsilon transforms, we may indicate the dependencies on the A_i as in Figure 2 where each transform depends on the A 's directly.

A_0			
A_1	$e_1(A_0)$		
A_2	$e_1(A_1)$	$e_2(A_0)$	
A_3	$e_1(A_2)$	$e_2(A_1)$	$e_3(A_0)$
A_4	$e_1(A_3)$	$e_2(A_2)$	
A_5	$e_1(A_4)$		
A_6			

Figure 2: Configuration for the e_i transforms

Wynn's Epsilon Algorithm

Note that the computation for any particular entry of A proceeds quite independently with an effort similar to any other transformation entry. So with the use of Shanks' transforms in its present form as a sequence to sequence transformation in which entries of the transformed sequence are of the form $e_m(S_n)$ (n=n,n+1,...,m=1,2,...), transform entries may be examined for approaching a limit faster. Therefore, the next transform may save a vast amount of labor.

In 1955, Wynn successfully proved the epsilon theorem which carried out the transformation more efficiently. He calculated the entries from the previous column by saving some auxiliary numbers along the calculations.

Following is the epsilon theorem developed by Wynn.

Epsilon Theoren:

If

$$E_{2m}(S_n) = e_m(S_n)$$

and

$$E_{2m+1}(S_n) = \frac{1}{e_m(\Delta S_n)}$$

then

$$E_{s+1}(S_n) = E_{s-1}(S_{n+1}) + \frac{1}{E_s(S_{n+1}) - E_s(S_n)},$$

$$(n,\ s = 0,\ 1,\ \ldots). \qquad (2.7)$$

provided that the initial conditions of $E_{(-1)}=0$, m=1,2,..., $E_{(0)} = S_{(m)}$, m=0,1,...; and none of the quantities $E_{2k}(A_n)$ becomes infinite.

The quantities Es may be placed in a two-dimensional array in which the suffix s indicates a column number and the superscript m a diagonal (see Figure 3).

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

Figure 3: Epsilon algorithm lozenge diagram

The even-numbered columns $E_{2k}(A_n)$ display the transformed sequences $e_k(A_n)$, and generally the transformed sequences converge to the limit of the A_n more rapidly than the original sequence A_n. The odd-numbered columns $E_{2k+1}(A_n)$ are intermediate and diverge to $\pm\infty$; so there is no need for the printing of $E_{2k+1}(A_n)$.

According to equation 2.7 the four quantities can be arrange in a lozenge, as shown below.

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

The right side entry can be computed by adding the left side entry to the inverse of the difference of the two in the middle. So the quantities Es are constructed by means of the relationship of

$$E_{s+1}^{(m)} = E_{s-1}^{(m+1)} + \left(E_{s}^{(m+1)} - E_{s}^{(m)} \right)^{-1} \qquad (m,s=0,1,\ldots).$$

This is called the fundamental relationship of the epsilon algorithm.

Implementation of the Epsilon Algorithm

Let us the application of

~~In this section we will~~ look at ~~some examples which~~ apply the epsilon algorithm with great success. First, we illustrate the transforms on Leibnitz series:

$$\pi = 4 - 4/3 + 4/5 - 4/7 + \ldots. \qquad (2.8)$$

This a very slowly convergent series but the epsilon algorithm speeds it up considerably. The transformation table is shown in Table I.

The tenth partial sum, A9, is correct to only one figure; it takes about 40,000,000 terms to get eight figures. However, e4 is already corrected to eight figure.

TABLE I

THE EPSILON TABLE FOR LEIBNITZ SERIES,
SUM=3.14159...

n	A(n)	e1	e2	e3	e4
0	4.0000000				
1	2.6666667	3.1666667			
2	3.4666667	3.1333333	3.1421053		
3	2.8952381	3.1452381	3.1441502	3.1415993	
4	3.3396825	3.1396825	3.1416433	3.1415909	3.1415928
5	2.9760462	3.1427129	3.1415713	3.1415933	3.1415927
6	3.2837385	3.1408814	3.1416029	3.1415925	
7	3.0170718	3.1420718	3.1415873		
8	3.2523659	3.1412548			
9	3.0418396				

Aitken "Δ^2" method is considered one of the powerful acceleration techniques and it had been a great success at the geometric series problems. The motivation for the method is to use the ratio of consecutive errors in the partial sums sequence to transform the original sequence to a faster converging sequence. Aitken's method can be written in the forms as below:

$$r = \frac{A_{n+1} - A}{A_n - A} = \frac{A_{n+2} - A}{A_{n+1} - A}, \tag{2.9}$$

or

$$A = \frac{A_n A_{n+2} - A_{n+2}^2}{(A_{n+2} - A_{n+1}) \underset{2}{-} (A_{n+1} - A_n)} \tag{2.10}$$

$$= A_n - \frac{(\Delta A_n)}{(\Delta A_n)} \tag{2.11}$$

$$= A_{n+2} - \frac{(\Delta A_{n+1})^2}{(\Delta A_n)}, \quad (2.12)$$

where r is a constant and A(n) is a numerical sequence.

Aitken's method would seem to have a good chance of success when the ratio of consecutive errors approaches any constant between $r = 0$ and $r = 1$. As it can be shown that the denominators in (2.10), (2.11), and (2.12) go to zero if $r=1$ and this would lead to the method not working well. Overall, Aitken's method is the "perfect" linear convergent accelerator for the geometric series.

The next example illustrates the weakness of Aitken's method in Shanks' "double" geometric series. The sequence of partial sums of A(n) is originally defined as

$$A(n) = f(z) = 1 + 3z/2 + 7z^2/4 + 15z^3/8 + 31z^4/14 + \ldots,$$

and the right side could be written as

$$f(z) = 2 \;/\; (1-z)(2-z)$$

$$= 2 \;/\; (1 + z + z^2 + \ldots) - 1 \;/\; (1 + z/2 + z^2/4 + \ldots). \quad (2.13)$$

The e_2 transform of the epsilon algorithm transforms the sum perfectly on any given five consecutive partial sums. It is not so fortunate with the Aitken's method in this case, especially when $z=10$. See Table II for the Epsilon table of Shanks' double geometric series and Table II for Aitken's transforms.

TABLE II

THE EPSILON TABLE OF SHANKS' DOUBLE GEOMETRIC SERIES
AT X=10. SUM = 2.77777778D-1

n	L(n)	e2	e4
0	.100000000D1		
1	.160000000D2		
2	.191000000D3	-.4062500D0	
3	.206600000D4	-.2014706D1	
4	.214410000D5	-.9892857D1	.2777778D-1
5	.218316000D6	-.4887675D2	.2777778D-1
6	.220269100D7	-.2427850D3	.2777778D-1
7	.221245660D8	-.1209806D4	.2777778D-1
8	.221733941D9	-.6038617D4	.2777778D-1

TABLE III

FAILURE OF AITKEN'S METHOD ON
SHANKS' DOUBLE GEOMETRIC SERIES AT X=10.
SUM=.277777D-1.

n	L(n)				
0	0				
1	.10000D1				
2	.16000D2	-.71D-1			
3	.19100D3	.41D0			
4	.20660D4	-.20D1	.17D-1		
5	.21441D5	-.98D1	.64D-2		
6	.21831D6	-.48D2	-.19D-1	.23D-1	
7	.22026D7	-.24D3	-.83D-1	.24D-1	
8	.22124D8	-.12D4	-.24D0	.24D-1	.24D-1
9	.22173D9	-.60D4	-.64D0	.23D-1	.24D-1

Our last example shows another disadvantage of Aitken's method but another success with the epsilon algorithm. In Lubkin's series:

$$1.13197175\ldots = \pi/4 + \ln(2)/2$$
$$= 1 + 1/2 - 1/3 - 1/4 + 1/5 + 1/6 - \ldots, \tag{2.14}$$

the repeated Aitken's method is completely confused by the ratio of consecutive errors which keeps switching signs and therefore will not find the convergent answer (as shown in Table IV). However, in Table V, we find that the epsilon method soon approaches the correct limit in the later columns.

TABLE IV

FAILURE OF AITKEN'S METHOD ON LUBKIN'S SERIES, SUM = 1.1314...

L(n)			
0			
1.00			
1.50	2.00		
1.1667	1.30		
0.9167	0.1667	3.1308	
1.1167	1.0278	.6560	
1.2833	2.1167	-3.0888	7.9530
1.1405	1.2064	1.6209	-1.0027
1.0155	0.1405	7.4390	-23.098

TABLE V

THE EPSILON TABLE FOR LUBKIN'S SERIES
SUM = 1.131...

L(n)	e1	e2	e3	e4
0				
1.00				
1.50	2.00			
1.17	1.30			
0.917	0.167	1.0755		
1.12	1.03	1.1248		
1.28	2.12	1.1420	1.1504	
1.14	1.21	1.1333	1.1359	
1.02	0.140	1.1285	1.1226	1.1300
1.13	1.07	1.1315	1.1304	1.1317

Stability of the Epsilon Algorithm

From the nature of the epsilon algorithm we may say that it is a recursive process involving repeated subtractions and divisions. And as such one would expect it to be numerically unstable because of the possibility of loss of digits due to cancellations which occur during the transformation. However, we found that in certain circumstances it is quite remarkably stable.

After studying the behavior of some sequences Wynn found that in certain circumstances the epsilon algorithm is a regular (i.e., convergence preserving) transformation: certain types of slowly convergent monotonic sequences are transformed into slowly convergent sequences of single-signed terms and certain type of slowly convergent oscillating sequences are transformed into rapidly convergent oscillating sequences.

Wynn concluded that transformations of monotonic sequences require the repeated subtraction of approximately equal quantities, and that in turn induces instability in which the rounding errors jump up and take complete charge of the computations after a few steps. On the other hand, the transformations of oscillating sequences showed the consistent subtraction of quantities having opposite signs with no loss of digits due to cancellation and the computation appeared to be completely stable.

CHAPTER III

UPDATE FORM OF THE EPSILON ALGORITHM

Motivations

As mentioned in Chapter II the epsilon algorithm uses the lozenge algorithm relationship. The four quantities below form a lozenge of the epsilon-array.

$$\begin{array}{ccc} & E_s^{(m)} & \\ E_{s-1}^{(m+1)} & & E_{s+1}^{(m)} \\ & E^{(m+1)} & \end{array}$$

The computation of the lozenge algorithm requires storing a vector of quantities, not a two-dimensional array; and the auxiliary variables aux0, aux1, and aux2 are implemented in the processes, as shown in Figure 4.

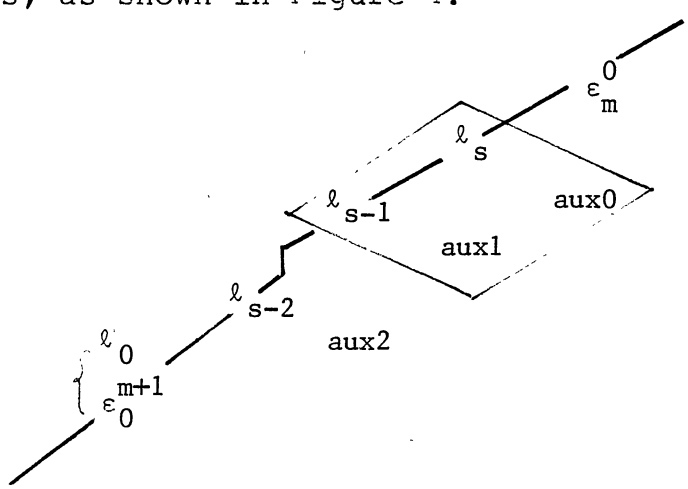


Figure 4: Programming lozenge algorithms

The vector l contains the quantities from Eo to Em, which lie along the thick line in Figure 9. The contents of l(s-1), l(s), aux0, and aux1 form a lozenge in the E-array. The processing starts with the computation of the aux0 quantity from those of l(s-1), l(s), and aux1; then the contents of l(s-1) is replaced by aux2, aux2 by aux1, and aux1 by aux0 accordingly. The value of s is increased by one, and the process is moved to the next step to form a new lozenge; and the above processes are repeated until all of the entries in the new diagonal are computed.

An ALGOL epsilon algorithm procedure which is given by Henry C. Thacher, Jr. in the COMM. A.C.M. Vol. 6, 1963 follows. This algorithm is a revision of the original epsilon algorithm constructed by P. Wynn and is the one used in this thesis.

```
Procedure Shanks(nmin,nmax,kmax,S);
value nmin, nmax, kmax;
integer nmin, nmax, kmax;
array S;

begin integer j,k,limj,limk,two kmax;
 real T0,T1;
 two kmax := kmax + kmax;
 limj := nmax;
for j := nmin step 1 until limj do
 begin T0 := 0;
 limk := j - nmin;
 if limk > two kmax then limk := two kmax; limk := limk - 1;
 for k := 0 step 1 until limk do
  begin T1 := S(j-k) - S(j-k-1);
    if T1 = 0 then T1 := T0 + 1/T1 else
    if S(j-k) = largest number then T1 := T0 else
       T1 := largest number;
    T0 := S(j-k-1)
    S(j-k-1) := T1
 end for k
```

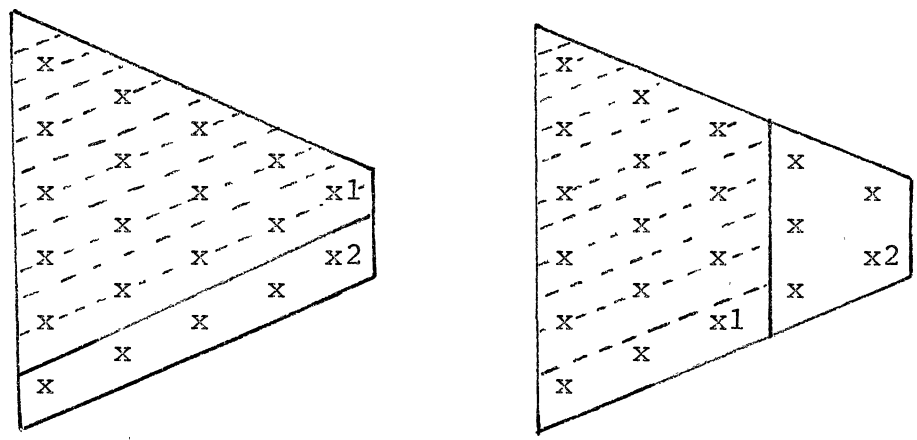
```
 end for j
end Shanks
```

This procedure replaces the elements S(nmin) through S(nmax-2*kmax) of the array S by the e(kmax) transform of the sequence S. The elements S(nmax-2*kmax+1) through S(nmax-1) are destroyed. Note that the array S is the same as vector l in the preceding description.

If, in a slowly convergent sequence, one can transform a certain E from the transformed sequence by applying the epsilon algorithm and we decide that a value of E_i may be a better approximation to the true value, then it is certainly wasteful to start the process from the original sequence. It seems more appropriate to start the process with the transformed sequence as the initial value. We may name this situation the horizontal extended process of the epsilon algorithm.

Or if the m+n elements of the basic sequence are considered after a transformed sequence has been obtained from m elements then it seems more reasonable to continue the process from the m+1th element to reach a transformed position rather than from the first element. This shall be called the vertical extended process of the epsilon algorithm.

Figure 5 shows the examples of two basic situations. The computing processes to obtain the first converted value X1 are included in the dashed zone. However, the steps needed to obtain the second converted value X2 are shown in the thick line zone. It is obvious that a lot of processes in the overlapped area are repeated and wasted. This leads to the following algorithm.



To extend vertically   To extend horizontally

Figure 5: Motivations of the update epsilon algorithm

Update Epsilon Algorithm

Procedure 3.1: Updated epsilon algorithm procedure

```
Procedure Shanks(nmin,nmax,kmax,S,H);
value nmin, nmax, kmax;
integer nmin, nmax, kmax;
array S,H;

begin integer j,k,limj,limk,two kmax;
 real T0,T1;
 two kmax := kmax + kmax;
 limj := nmax;
for j := nmin step 1 until limj do
 begin T0 := H(j);
 limk := j - nmin;
 if limk > two kmax then limk := two kmax; limk := limk - 1;
 for k := 0 step 1 until limk do
```

```
    begin T1 := S(j-k) - S(j-k-1);
      if T1 = 0 then T1 := T0 + 1/T1 else
      if S(j-k) = largest number then T1 := T0 else
         T1 := largest number;
      T0 := S(j-k-1)
      S(j-k-1) := T1
   end for k
   H(j) := T0
  end for j
  end Shanks
```

The variables of nmin and nmax indicate the subscript of the beginning and ending terms of array S. When nmin is not equal to 1, the vertical extended process is applied and more elements are included in the process. The array H, holding the values of the previous Ei column, is defined for the purpose of the horizontal extended process. Kmax indicates the order of the epsilon transform. When the horizontal extending process is applied, the order of the epsilon transform is numbered from the transformed sequence and not from the original sequence.

When both horizontal and vertical extended processes are to be applied to the transformed sequence, it is recommended that one apply the horizontal extended process first, then apply the vertical extended process.

Numerical Test Problems

1. Fourier Series:

A Fourier series may be defined as an expansion of a function or representation of a function in a series of sines and cosines such as

$$f(x) = a_0/2 + \sum_{n=1}^{\infty} a(n) \cos(nx) + \sum_{n=1}^{\infty} b(n) \sin(nx) \qquad (3.1)$$

where a(o),...,a(n) and b(1),...,b(n) are real or complex constants. The conditions imposed on f(x) to make this equation valid are that f(x) has only a finite number of infinite discontinuities and only a finite number of extreme values, maxima and minima.

One of the advantages of a Fourier representation over some other representation, such as a Taylor series, is that it may represent a discontinuous function or a periodic function conveniently.

The first example is to model diffusion in an infinite slab which has plane parallel sides. The boundary conditions at the face of the slab are piecewise constant in time. The method of Fourier series is used to compute one function value and an error estimate. To accelerate the convergence of the series, the epsilon algorithm is applied. The program, called SLAB, is provided by Dr. Chandler.

The update epsilon algorithm is tested by replacing the original epsilon algorithm in the SLAB program. The table VI shows the test results of the SLAB program when time is equal to 0.666111170477D01.

In this case, only the situation of horizontal extension is tested. The above results in Table VI are the same as the results when the original epsilon algorithm is applied twice in the SLAB program.

TABLE VI

THE VALUES OF TWENTY DIFFUSIONS AT TIME =0.666111170477D1

0.1600000D-01	0.9708385D-02	0.6143988D-02
0.4832352D-02	0.4309322D-02	0.3953008D-02
0.3657657D-02	0.3420466D-02	0.3246789D-02
0.3140859D-02	0.3105259D-02	0.3140859D-02
0.3246789D-02	0.3420466D-02	0.3657646D-02
0.3952481D-02	0.4296738D-02	0.4670510D-02
0.5001130D-02	0.5130765D-02	0.5000000D-02

2. The Confluent Hypergeometric Function

The confluent hypergeometric equation

$$zy''(z) + (c-z)y'(z) - ay(z) = 0 \tag{3.2}$$

is simplified from the hypergeometric equation by merging two of its singularities. One solution of the confluent hypergeometric equation is

$$y(z) = {}_1F_1(a,c;z) = M(a,c:z)$$
$$= 1 + \frac{a}{C}\cdot\frac{X}{1!} + \frac{a(a+1)}{C(C+1)}\cdot\frac{X^2}{2!} + \ldots,$$
$$c = 0,-1,-2,\ldots. \tag{3.3}$$

The solution is convergent for all finite z. In terms of the Pochhammer symbols,

$$(a)_n = a(a+1)(a+2)\ldots(a+n-1) = (a+n-1)!/(a-1)!,$$
$$(a)_0 = 1, \tag{3.4}$$

the confluent hypergeometric function becomes

$$M(a,c:z) = {}_1F_1(a,c:z) \tag{3.5}$$
$$= \sum_{n=1}^{\infty} \frac{(A)n}{(C)n} , \frac{X^n}{n} .$$

The leading subscript 1 indicates that one Pochhammer symbol appears in the numerator and the final subscript 1 indicates one Pochhammer symbol in the denominator. If the parameter a is zero or a negative integer, M(a,c;z) becomes a polynomial.

Both the "original" epsilon algorithm and the "update" epsilon algorithm are applied to the confluent hypergeometric series with the initialization of a=1,c=1, and z=(0,1.5707963267949).

The test results are shown in Table VII.

The quantities of the confluent hypergeometric series are first included to reach the column of ei. When processing to column e4, the method to extend horizontally in the updated epsilon algorithm is applied. Furthermore, five more quantities of the confluent hypergeometric series are added into the process to test the vertical extension function. The dashed zones in Table VII indicate the different processing steps. The test results shown in Table VII are similar to the original epsilon algorithm.

TABLE VII

THE EPSILON TABLE IN CONFLUENT HYPERGEOMETRIC FUNCTION

(.10000000D01,.00000000D0)		
(.10000000D01,.15707963D1)	(.23697291D00,.97151625D0)	
(-.23370055D00,.15707963D1)	(.31750537D-1,.10638220D0)	(.11374722D-1,.99993530D0)
(-.23370055D00,.92483222D0)	(-.13923435D-1,.10111385D1)	(.50818081D-3,.10017917D1)
(.19968957D-1,.92483222D0)	(-.28182153D-2,.99736605D0)	(-.25577479D-3,.10001280D1)
(.19968957D-1,.10045248D1)	(.44371597D-3,.99941315D0)	(-.23167201D-4,.99996636D0)
(-.89452299D-3,.10045248D1)	(.10569411D-3,.10000675D0)	(.41047145D-5,.99999646D0)
(-.89452299D-3,.99984310D0)	(-.93874791D-5,.10000168D1)	(.47744989D-6,.10000004D1)
(.24737276D-4,.99984310D0)	(-.24372107D-5,.99999879D0)	
(.24737276D-4,.10000035D1)		

(.21234162D-3,.99999997D0)	
(.48431629D-5,.10000237D1)	(.21315865D-5,.99999999D0)
(-.25077908D-5,.10000009D1)	(.29803821D-7,.10000001D1)
(-.13028108D-6,.99999974D0)	

CHAPTER IV

SINGULAR RULES FOR EPSILON ALGORITHM

Introduction

Regardless of precision, when using the formula of the epsilon algorithm it may sometimes occur that a quantity is numerically ill-determined. As a result of this and because of the way in which the algorithmic formulae are used, this misfortune is propagated throughout a whole sector. Wynn (1963), Cordellier (1977), and Brezinski (1978) have suggested ways to deal with them. We begin the discussion with a simple example.

According to the partial sums of the power series for e^x, its initial members are as follows:

0					
	0				
0		1			
	1		$\frac{1}{(1-X)}$		
0		$\frac{1}{X}$		$\frac{(2-4X+X^2)}{-X^2}$	
	1+X		$\frac{(2+X)}{(2-X)}$		$\frac{2(3+X)}{6-4X+X^2}$
0		$\frac{1}{2X^2}$		$\frac{-2(6-6X+X^2)}{X^3}$	
	$1+X+\frac{X^2}{2}$		$\frac{(6+4X+X^2)}{(6-2X)}$		
0		$\frac{1}{6X^3}$			
	$1+X+\frac{X^2}{2}+\frac{X^3}{6}$				
0					

Figure 6: Numerical example of the singular rule case

When x=2 we construct the above array under the epsilon scheme as shown below:

$$
\begin{array}{ccccc}
0 & & & & \\
 & 1 & & & \\
1 & & -1 & & \\
 & 1/2 & & 1/2 & \\
3 & & \infty & & 5 \\
 & 1/2 & & 1/2 & \\
5 & & 9 & & \\
 & 3/4 & & & \\
19/3 & & & &
\end{array}
$$

Figure 7: Epsilon scheme of the singular case

In this example when one follows the above formulae we find that two entries in the ε_i column have attained the same value (1/2 and 1/2). However, by using the epsilon algorithm we can hardly obtain the value 5, instead infinity in that entry is obtained. This is one of the typical singular cases. When we repeatedly apply the epsilon algorithm, we may forecast that the misfortune is propagated throughout the whole sector. Therefore, the singular rule was introduced by Wynn in 1963 to overcome this difficulty.

Singular Rules

Now suppose that the example in figure 6 has been put into the large lozenge diagram as below,

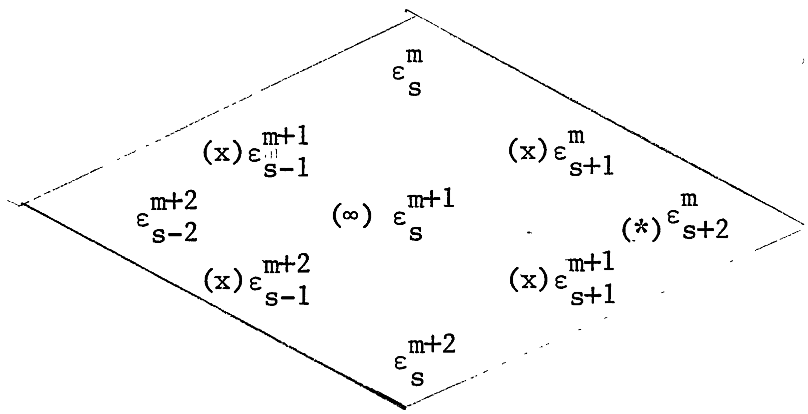


Figure 8: Singular case in the large lozenge diagram

and two entries in the ε^{i}_{s-1} column are both equal to the same value - say x. It is immediately noticeable that becomes finite, ε^{m}_{s+1} and ε^{m+1}_{s-1} are equal to x, and ε^{m}_{s+2} is indeterminate. Further quantities in a sector whose vertex is at ε^{m}_{s+2} remain undetermined. However, if we try to avoid using the entry of ε^{m+1}_{s} to calculate the value of ε^{m}_{s+2}, the indeterminate situation will be eliminated. One can substitute the appropriate values that were originally used to derive the particular entry that progressed to infinity as opposed to using the ill-determined infinity term. This should alleviate indeterminate quantities. This multiple appeal to the epsilon algorithm relationship appears as follows:

$$\varepsilon^{m}_{s+2} = \varepsilon^{m+1}_{s} + \cfrac{1}{\varepsilon^{m+2}_{s} + \cfrac{1}{\varepsilon^{m+2}_{s} - \varepsilon^{m+1}_{s}} - \varepsilon^{m+1}_{s-1} - \cfrac{1}{\varepsilon^{m+1}_{s} - \varepsilon^{m}_{s}}}$$

$$= \varepsilon^{m+1}_{s} + \cfrac{1}{\cfrac{1}{\varepsilon^{m+1}_{s} - \varepsilon^{m+2}_{s}} + \cfrac{1}{\varepsilon^{m+2}_{s} - \varepsilon^{m+1}_{s}} - \cfrac{1}{\varepsilon^{m+1}_{s} - \varepsilon^{m}_{s}}}$$

$$= \varepsilon^{m+1}_{s} - \cfrac{\varepsilon^{m+1}_{s}}{1 - \cfrac{\varepsilon^{m+2}_{s-2}}{\varepsilon^{m+1}_{s} - \varepsilon^{m+2}_{s-2}} + \cfrac{\varepsilon^{m+2}_{s}}{\varepsilon^{m+1}_{s} - \varepsilon^{m+2}_{s}} - \cfrac{\varepsilon^{m+2}_{s}}{\varepsilon^{m+1}_{s} - \varepsilon^{m}_{s}}}$$

$$= \cfrac{\cfrac{\varepsilon^{m+2}_{s}}{1 - \varepsilon^{m+2}_{s}(\varepsilon^{m+1}_{s})^{-1}} + \cfrac{\varepsilon^{m}_{s}}{1 - \varepsilon^{m}_{s}(\varepsilon^{m-1}_{s})^{-1}} - \cfrac{\varepsilon^{m+2}_{s-2}}{1 + \varepsilon^{m+2}_{s-2}(\varepsilon^{m+1}_{s})^{-1}}}{1 - \cfrac{\varepsilon^{m+2}_{s-2}}{\varepsilon^{m+1}_{s} - \varepsilon^{m+2}_{s-2}} + \cfrac{\varepsilon^{m+2}_{s}}{\varepsilon^{m+1}_{s} - \varepsilon^{m+2}_{s}} + \cfrac{\varepsilon^{m}_{s}}{\varepsilon^{m+1}_{s} - \varepsilon^{m}_{s}}} \tag{4.1}$$

From the above illustration we find that ε^{m}_{s+2} becomes large and ill-determined when ε^{m+1}_{s-1} and ε^{m+2}_{s-1} are almost equal. But when ε^{m}_{s+1} and ε^{m+1}_{s+1} are derived from the regular epsilon algorithm and ε^{m}_{s+2} is derived from the singular rule (2.7), they are all quite well determined.

However, when values of ε^{m+1}_{s-1} and ε^{m+2}_{s-1} are exactly equal, the singular rule for the epsilon algorithm becomes quite simple --

$$\varepsilon^{m}_{s+2} = \varepsilon^{m+2}_{s} + \varepsilon^{m}_{s} - \varepsilon^{m+2}_{s-2} \tag{4.2}$$

As in Figure 6, we may obtain the value 5 by applying the singular rule (5=9+(-1)+3).

Since, the more common case is that ε^{m}_{s+2} is highly susceptible to the loss of significant digits via subtraction when ε^{m+1}_{s-1} and ε^{m+2}_{s-1} are almost equal. Thus it is very important to know when the cancellation occurred. Wynn asserts that when a loss of "f" decimal figures takes place at the subtraction of ε^{m+1}_{s-1} and ε^{m+2}_{s-1}, it is the time to apply the singular rule. The general rule to estimate the loss of "f" decimal figures is obtained by

$$f = \log_{10} \left(\frac{\varepsilon^{m+1}_{s-2}}{\varepsilon^{m+2}_{s-2} - \varepsilon^{m+1}_{s-2}} \right). \tag{4.3}$$

E.g., when ε^{m+1}_{s-2} is about ten times as large as the difference between ε^{m+2}_{s-2} and ε^{m+1}_{s+2}, one significant digit is lost.

(In the singular epsilon algorithm procedure a real parameter called - CANCEL, equal in magnitude to 10 ** f, is provided to detect when the singular rule needs to be implemented.)

Applying The Singular Rule To The Epsilon Program

To apply the singular rules to the normal epsilon procedures, the formula of the singular rule in (4.1) may be reformed in several steps.

If (4.4)

$$A = \varepsilon_{s-2}^{m+2} \left(1 - \varepsilon_{s-2}^{m+2} \left(\varepsilon_{s}^{m+1} \right)^{-1} \right)^{-1}$$

(4.5)

$$B = \varepsilon_{s}^{m} \left(1 - \varepsilon_{s-1}^{m} \left(\varepsilon_{s}^{m+1} \right)^{-1} \right)^{-1}$$

$$D = \varepsilon_{s}^{m+2} \left(1 - \varepsilon_{s}^{m+2} \left(\varepsilon_{s}^{m+1} \right)^{-1} \right)^{-1} \qquad (4.6)$$

and $$a = D + B - A \qquad (4.7)$$

then $$\varepsilon_{s+2}^{m} = C = a \left(1 + a * \left(\varepsilon_{s}^{m+1} \right)^{-1} \right)^{-1}. \qquad (4.8)$$

Suppose that cancellation occurs in the formation of ε_{s-1}^{m+2} - ε_{s-1}^{m+1} during the process of computing quantities lying on the diagonal through ε_{s-1}^{m+2}, ε_{s}^{m+1} and ε_{s+1}^{m}. Immediately, we know that we areabout to compute A from (4.4), and store it. The value of S that points to the current position in the E-array needs to be saved too. Next, after the entry of ε_{s+1}^{m} is computed, the value of B can be obtained before the next process. After reaching the end of the E-array, the E-array process is repeated at the other end. When the current value of s is equal to the previous stored value s plus 1 then we do know that we are about to compute D. Then we reach the point to compute ε^{m}_{s+2} by applying the singular rule (4.8) instead of the normal epsilon algorithm and thereby nullify the ill effects of cancellation.

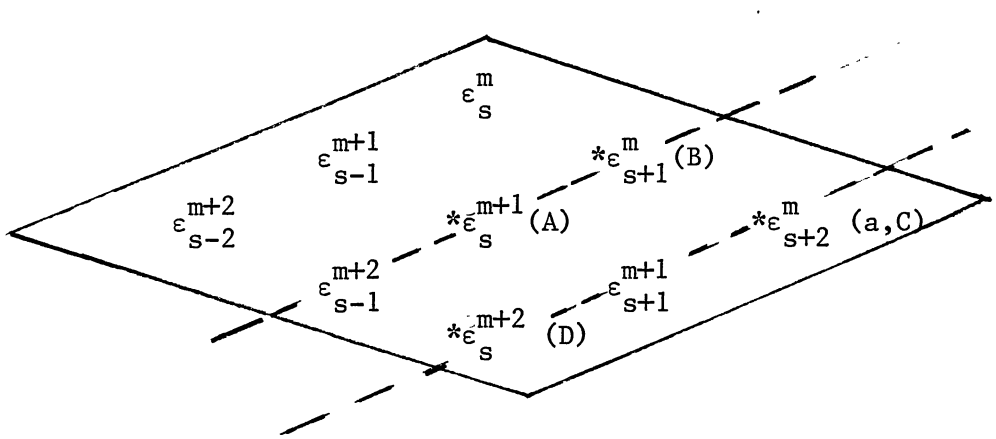


Figure 9: Programming lozenge diagram of singular rule

The entries with an asterisk in figure 9 represent the computing processes necessary to apply the singular rules when the cancellation between the quantities of ε^{m+1}_{s-1} and ε^{m+2}_{s-1} occur. Except consider

However, so far a the single point of instability is concerned. We should apply the above procedure in the case in which there are a number of points in the E-array at which cancellation takes place. Basically the ways to handle the multiple points of instability are the same as the above procedures, but the array S is created to store the multiple instability positions instead of point S. Moreover, the same situation happens at the points A, B, and D.

Wynn's singular rules are appropriate to isolated points of cancellation when two or more instability points occur in the same column. Otherwise, Wynn's singular rules are not appropriate. If the vector case is concerned, the special rules of Cordellier (1977) are suggested for implementation.

TEST RESULTS

The ALGOL procedure that Wynn suggested to apply the multiple singular rule has been rewritten in FORTRAN and the power series for e^x with $x=2$ has been tested and the value 5 has been obtained. See Appendix B.

CHAPTER V

ERROR EXPANSION SERIES AND TEST RESULTS

Series expansions are a very important aid in numerical calculations, especially for quick estimates made in hand calculation - e.g., in evaluating functions, integrals, or derivatives. Solutions to differential equations can often be expressed in terms of series expansions. In practice, one is seldom seriously concerned about a strict error bound when the computed terms reach acceptable accuracy. To use the first neglected term as an estimate of the remainder is very common and easy. However, in numerical analysis sometimes it is not accurate enough. The Euler-Maclaurin summation formula can be used to get the value of the remainder with higher accuracy, but it leads to very complicated calculation.

However, we intend to use a new approach to compute the magnitude of the error in the partial sums of an infinite series, by assuming that there exists a series expansion in negative powers of n. And we apply the epsilon algorithm repeatedly to compute the values of the numerators in the error expansion series. In order to facilitate the discussion, the example below is given.

To get the value of ln2 we apply the following series expansion

$$S(n) = \ln 2 = 1 - 1/2 + 1/3 - 1/4 + \ldots.. \qquad (5.1)$$

or in summation formula

$$\sum_{i=1}^{n} (-1)^{i+1} / i. \qquad (5.2)$$

Suppose there exists an error expansion series which is of the following form:

$$|e(n)| = \frac{C1}{n} + \frac{C2}{n^{**}2} + \frac{C3}{n^{**}3} + \frac{C4}{n^{**}4} + \ldots,$$

$$n=1,2,\ldots N. \qquad (5.3)$$

To compute constant C1 the following steps can be followed.

Compute S(n), n = 1,2,3,...,N.

Compute $e(n) = |\ \ln 2 - S(n)\ |$.

Then, apply equation $C1 = n * |e(n)|$ to get the constant C1, by applying a repeated Aitken, Epsilon, or Romberg algorithm to estimate C1 as $n \dashrightarrow \infty$.

To compute C2.

Use the true value of C1, if it can be seen to converge to a fraction.

Repeat the above process by using equation

$C2 = (|e(n)| - C1/n) * (n**2)$

.

.

.

Finally, the error expansion series for ln2 is gained, as follows:

$$|e(n)| = \frac{(1/2)}{n} + \frac{(-1/4)}{n^{**}2} + \frac{0}{n^{**}3} + \frac{(1/8)}{n^{**}4} + \frac{0}{n^{**}5} + \frac{(-1/4)}{n^{**}6} + \ldots. \qquad (5.4)$$

Notice that the values in each constant C are in fractional form. However, the above expansion series is invalid when n=0.

If the original series is modified as below

$$|e(n)| = \frac{C1}{n+a1} + \frac{C2}{(n+a2)^{**}2} + \frac{C3}{(n+a3)^{**}3} + \ldots, \qquad (5.5)$$

where a1, a2, a3, ... are constant and then apply the original processes with the modified series, we obtain a new series:

$$|e(n)| = \frac{(1/2)}{n+1/2} + 0 + \frac{(-1/8)}{(n+1/2)^3} + 0 + \frac{(5/32)}{(n+1/2)^5} + 0 + \frac{(-61/128)}{(n+1/2)^7} + 0 + \frac{(1385/512)}{(n+1/2)^9} + \ldots, \qquad (5.6)$$

which is valid even when n=0. And in this case the value of "a" is a fixed constant, 1/2. Furthermore, the numerators of the constants C are just Euler numbers $\widetilde{B}$ and the denominators are 2 to the power of n.

Note that due to the cancellation that occurs in the computer process, it runs out of accuracy in the computation of the sixth term of the Euler number $\widetilde{B}$, even using double precision arithmetic (64 bits).

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

The flexibility to extend the epsilon algorithm process vertically or horizontally has been tested on several numerical examples. The numerical results as shown in Chapter III are satisfied. The array H, as mentioned in Chapter III, used in the update epsilon algorithm, is the only additional storage space added into the original epsilon algorithm.

3 Practically speaking, with the application of the update epsilon algorithm, time is saved by erasing duplication of the entire process in order to reach the final state that is desired. It becomes even more obvious when the epsilon algorithm must be applied to a large numerical sequence. Thus, the update epsilon algorithm is a very useful tool indeed.

Since the epsilon algorithm is formed under the lozenge diagram algorithm, it is possible to apply the same principle to other algorithms which are also formed under the lozenge algorithm. Therefore, it is logical to apply this process to the ρ algorithm, Q-D algorithm, etc.

In Chapter V, an alternate method to estimate the error bound in the partial sum series has been developed. The error expansion series of the series ln(2) has been obtained. The author believes that this is a much easier and less complicated method than the Euler-Maclaurin summation formula to estimate the error. Therefore, this method is highly recommended for other partial sum series.

A SELECTED BIBLIOGRAPHY

Aitken, A. C. "On Bernoulli's Numerical Solution of Algebraic Equations." Proceedings of the Royal Society of Edinburgh, 46 (1926), 289-305.

Brezinski, C. "A General Extrapolation Algorithm." Numerische Mathematik, No 35 (1980), pp. 1980.

Brezinski, C. "Convergence Acceleration Methods: The Past Decade." J. Computational and Applied Meath. Vol. 12 (May 1985), 19-36.

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

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

Chandler, J. P. FORTRAN Program "SLAB". Computer Science Department, Oklahoma State University. 1981.

Conte Samuel D./Carl de Boor, Elementary Numerical Analysis: an Algorithmic Approach. 2nd Edition. New Work: McGraw-Hill Book Company, 1972.

Cordellier, F. "Particular Rules for the Vector E-algorithm." Numerische Mathematik, Vol. 27 (1977), 203-207.

Dahlquist Germund and Björck Ake Numerical Methods, Prentice-Hall Series in Automatic Computation, 1974.

Fröberg, Carl-Erik Introduction to Numerical Analysis, Addison-Wesley Publishing Company, 1965.

Hartree, D. R. "Note on Iterative Process" Cam. Phil. Soc., Vol. 45 part 2 (April 1949), 230-236.

Hildebrand, F. B. Introduction to Numerical Analysis. New York: McGraw Hill, 1956.

Isakson, Gabriel "A Method for Accelerating the Convergence of an Iteration Procedure" Jour. Aero. Soc. Vol. 16, No. 7 (July 1949), 443.

Jennings, A. "Accelerating the Convergence of Matrix Iterative Processes." Journal of Institute Mathematic Application, Vol. 8 (1971), 99-110.

Lubkin, S. "A Method of Summing Infinite Series." Journal of Research, National Bureau of Standards, Vol. B48 (1952), 228-254.

MacDonald, J. "Accelerated Convergence, Divergence, Iteration, Extrapolation and Curve Fitting." Journal of Applied Physics, Vol. 35, No. 10 (Oct. 1964), 3034-3041.

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

Richardson, L. F. "The Deferred Approach to the Limit." Royal Society of London Philosophical Transactions, Vol. A226 (1927), 299-349.

Samuelson, P. A. "A Convergent Iterative Process" Jour. of Math. and Phys. Vol. 24 (1945), 131-134.

Sandberg, Dan "0.8660254 = 3/2: An Algorithm that Converts Decimals to Fractions." BYTE, May 1985, 429-464.

Schmidt, R. J. "On the Numerical Solution of Linear Simultaneous Equations by an Iterative Method." The Philosophical Magazine and Journal of Science, 7th Series, Vol. 32 (1941), 369-383.

Shanks, D. "Non-Linear Transformations of Divergent and Slowly Convergent Sequences." Journal of Mathematics and Physics, Vol. 34 (April 1955), 1-42.

Shanks, D. and Walton, T. S. The Use of Rational Functions as Approximate Solutions of Certain Trajectory Problems Naval Ordnance Laboratory Memorandum 9524 (1948) White Oak, Md. P2; P13, equation (23).

Smith, D., and W. Ford. "Numerical Comparisons of Nonlinear Convergence." SIAM Journal on Numerical Analysis, Vol. 16 (1979), 481-499.

Thacher, H. C. "Algorithm 215: Shanks." Comm. ACM, Vol. 6 (November 1963), 662.

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

Wynn, P. "A Comparision Technique for the Numerical Transformation of Slowly Convergent Series Based on the Use of Rational Functions." Numer. Math., Vol. 4 (1962), 8-14.

Wynn, P. "A Note on Programming Repeated Application of the Epsilon Algorithm." Revue Francaise de Traitement de l'Information. Chiffres, 8 (1965), 23-62.

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

Wynn, P. "Acceleration Techniques in Numerical Analysis, with Particular Reference to Problems in One Independent Variable." Information Processing, 1962.

Wynn, P. "An Arsenal of Algol Procedures for Complex Arithmetic" BIT, 2 (1962), 232-255.

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

Wynn, P. "On a Device for Computing the em(sn) Transformation." Mathematical Tables and Other Aids to Computation, Vol. 10 (1956), 91-96.

Wynn, P. "On the Convergence and Stability of the Epsilon Algorithm." SIAM Journal on Numerical Analysis, Vol. 3 (1966), 91-122.

Wynn, P. "Singular Rules for Certain Non-linear Algorithms." BIT, 3 (1963), 175-195.

Wynn, P. "The Epsilon Algorithm and Operational Formulas of Numerical Analysis" Math. of Comp., Vol. 15 (1960), 151-158.

APPENDIX A

PROGRAM LISTING OF "SLAB" WITH

UPDATE EPSILON ALGORITHM

```
$JOB
C  THIS PROGRAM MODELS DIFFUSION IN AN INFINITE SLAB HAVING PLANE
C  PARALLEL SIDES.  THE DIFFUSIVITY, D, MUST BE GIVEN AND FIXED.
C  THE BOUNDARY CONDITIONS AT THE FACES OF THE SLAB MUST BE GIVEN AND
C  BE PIECEWISE CONSTANT IN TIME.
C
C  AUTHOR: J. P. CHANDLER, COMPUTER SCIENCE DEPT.,
C                            OKLAHOMA STATE UNIVERSITY.
C
C  THE DIMENSIONS OF THE ARRAYS MUST BE   BV(NTPMX,2), TFIN(NTPMX),
C       B(2,NTRMX), C(NTPMX-1,NTRMX), U(NDXMX+1) .
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION U(51)
      COMMON /SLAB/ BV(10,2),TFIN(10),B(2,100),C(9,100),
     X     EL,D,T,X,ARGMN,FX,ABERF,NTIMP,NTRMS,KW,NTACT
C
C                             KR AND KW ARE THE LOGICAL UNIT NUMBERS OF
C                                  THE CARD READER AND THE LINE PRINTER.
      KR=5
      KW=6
C                             NTPMX ...  MAXIMUM NUMBER OFTIME PERIODS
      NTPMX=10
C                             NTRMX ...  MAXIMUM NUMBER OF TERMS TO BE
C                                  USED IN THE FOURIER SERIES
      NTRMX=100
C                             NDXMX ...  MAXIMUM NUMBER OF X INTERVALS
      NDXMX=50
C                             ARGMN ...  A NEGATIVE NUMBER SOMEWHAT
C                                  GREATER THAN THE GREATEST VALUE OF
C                                  X FOR WHICH EXP(X) WOULD UNDERFLOW
      ARGMN=-60.
      RZERO=0
C
      READ(KR,10)NTIMP,NTRMS,EL,D
   10 FORMAT(2I5,2D10.5)
      WRITE(KW,20)NTIMP,NTRMS,EL,D
   20 FORMAT(/////42H DIFFUSION IN A SLAB, USING FOURIER SERIES///
     *     8X,26H NUMBER OF TIME PERIODS = ,I5//
     *     8X,34H NUMBER OF TERMS IN EACH SERIES = ,I5//
     *     8X,13H THICKNESS = ,D15.7//8X,15H DIFFUSIVITY = ,D15.7)
      IF(NTIMP.LT.2) GO TO 26
      IF(NTIMP.GT.NTPMX) GO TO 26
      IF(NTRMS.GT.NTRMX) GO TO 26
C
      READ(KR,30)(BV(J,1),BV(J,2),TFIN(J),J=1,NTIMP)
   30 FORMAT(3D10.5)
      WRITE(KW,40)
   40 FORMAT(//////6H  TIME,12X,16H BOUNDARY VALUES,14X,
     *     12H TIME AT END/7H PERIOD,12X,14H DURING PERIOD,16X,
     *     10H OF PERIOD//1H )
      NTMU=NTIMP-1
      DO 50 J=1,NTMU
   50   WRITE(KW,60)J,BV(J,1),BV(J,2),TFIN(J)
   60 FORMAT(1X,I4,D20.7,D16.7,D20.7)
      WRITE(KW,60)NTIMP,BV(NTIMP,1),BV(NTIMP,2)
      IF(NTIMP.EQ.2) GO TO 69
      DO 65 J=2,NTMU
      IF(TFIN(J).LE.TFIN(J-1)) GO TO 26
   65 CONTINUE
C                             COMPUTE AND PRINT THE FOURIER COEFFICIENTS.
   69 CALL CALCC
      DO 70 JB=1,2
   70   WRITE(KW,80)JB,(B(JB,J),J=1,NTRMS)
   80 FORMAT(///3H B(,I1,6H,J).../(10X,7E15.7))
      DO 90 KT=2,NTIMP
   90   WRITE(KW,100)KT,(C(KT-1,J),J=1,NTRMS)
  100 FORMAT(///45H FOURIER COEFFICIENTS FOR TIME PERIOD NUMBER ,I2,
     *     3H...//(8X,7E15.7))
C
  110 READ(KR,10)NDX,NDT,TMIN,TMAX
C
```

```
C                              TEST FOR END OF RUN.
      IF(NDX.LT.0) STOP
  130 WRITE(KW,140)NDX,NDT,TMIN,TMAX
  140 FORMAT(////8X,29H NUMBER OF INCREMENTS IN X = ,I5//
     *       8X,29H NUMBER OF INCREMENTS IN T = ,I5//
     *       8X,16H INITIAL TIME = ,E15.7//
     *       8X,16H FINAL TIME   = ,E15.7///1H )
      IF(NDX.GT.NDXMX) GO TO 26
      DX=RZERO
      IF(NDX.EQ.0) GO TO 142
  141 ENDX=NDX
      DX=EL/ENDX
  142 DT=RZERO
      IF(NDT.LE.0) GO TO 144
  143 ENDT=NDT
      DT=(TMAX-TMIN)/ENDT
  144 NDXPU=NDX+1
      NDTPU=NDT+1
C                              LOOP OVER THE TIME POINTS REQUESTED.
      DO 230 KT=1,NDTPU
        AKTMU=KT-1
        T=TMIN+AKTMU*DT
      IF(KT.EQ.NDTPU) T=TMAX
  160   ERMAX=RZERO
        NTMAX=0
C                              LOOP OVER THE X VALUES REQUESTED.
        DO 220 JX=1,NDXPU
          AJXMU=JX-1
          X=AJXMU*DX
      IF(JX.EQ.NDXPU) X=EL
  180     CALL CALCF
          U(JX)=FX
      IF(NTACT.GT.NTMAX) NTMAX=NTACT
      IF(ABERF.GT.ERMAX) ERMAX=ABERF
  220     CONTINUE
  230   WRITE(KW,240)T,ERMAX,NTMAX,(U(JX),JX=1,NDXPU)
  240 FORMAT(/8H TIME = ,D14.7,9X,24H MAX. ESTIMATED ERROR = ,D14.7,
     *       8X,23H MAX. NO. TERMS USED = ,I3//(1X,7D15.7))
C
C                              GO BACK FOR MORE VALUES OF NDX, ETC.
      GO TO 110
   26 WRITE(KW,987)
  987 FORMAT(///42H THERE IS A FATAL ERROR IN THE DATA ABOVE./1H )
      STOP
      END
      SUBROUTINE CALCC
C
C  COMPUTES THE FOURIER COEFFICIENTS FOR THE SLAB PROGRAM.
C
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON /SLAB/ BV(10,2),TFIN(10),B(2,100),C(9,100),
     X       EL,D,T,X,ARGMN,FX,ABERF,NTIMP,NTRMS,KW,NTACT
C
      QEXP(ARG)=DEXP(ARG)
C
      PI=3.141592653589793D0
      RZERO=0
      UNITY=1
      RTWO=2
      RFOUR=4
C
C  B(1,J) IS THE COEFFICIENT OF SIN(J*PI*X/L) IN THE SINE SERIES
C  FOR F(X)=1.0, ZERO .LT. X .LT. L.
C  B(2,J) IS THE COEFFICIENT IN THE SINE SERIES FOR F(X)=X.
C
      SGN=UNITY
      DO 1000 J=1,NTRMS
        B(1,J)=RZERO
        AJ=J
        B(2,J)=SGN*RTWO*EL/(PI*AJ)
 1000   SGN=-SGN
C
```

```
      DO 1010 J=1,NTRMS,2
        AJ=J
 1010   B(1,J)=RFOUR/(PI*AJ)
C
C  COMPUTE THE PROBLEM-DEPENDENT FOURIER COEFFICIENTS.
C
C                            LOOP OVER THE TIME PERIODS.
      DO 1040 KT=2,NTIMP
      KTMU=KT-1
      AZDIF=BV(KT,1)-BV(KTMU,1)
      AUDIF=((BV(KT,2)-BV(KT,1))-(BV(KTMU,2)-BV(KTMU,1)))/EL
      IF(KT.GE.3) DTDIF=D*(TFIN(KT-1)-TFIN(KT-2))
C
C                            LOOP OVER THE FOURIER COEFFICIENTS FOR
C                                 THIS TIME PERIOD.
 1016   DO 1030 J=1,NTRMS
          TOLD=RZERO
      IF(KT.LT.3) GO TO 1030
 1020     AJ=J
          ARG=-(AJ*PI/EL)**2*DTDIF
      IF(ARG.LT.ARGMN) GO TO 1030
 1025 TOLD=C(KTMU-1,J)*QEXP(ARG)
 1030 C(KTMU,J)=TOLD-AZDIF*B(1,J)-AUDIF*B(2,J)
 1040   CONTINUE
      RETURN
      END
      SUBROUTINE CALCF
C
C  COMPUTES ONE FUNCTION VALUE AND AN ERROR ESTIMATE, FOR THE
C  SLAB PROGRAM.  THE METHOD OF FOURIER SERIES IS USED.
C  TO ACCELERATE THE CONVERGENCE OF THE SERIES, THE EPSILON ALGORITHM
C  IS APPLIED TO THE COMPLEX EXPONENTIAL SERIES, AND THE IMAGINARY
C  PART IS THEN RECOVERED.  (THE REAL EPSILON ALGORITHM DOES NOT WORK
C  WELL IN THIS PROBLEM.)
C
      IMPLICIT REAL*8 (A-H,O-Z)
      COMPLEX*16 S,SAVE,QCMPL,DCMPLX,QCEXP,CARG,CDEXP,CI,XFACT,CXFAC,
     *     SUM,EXTRP,EXTSV,SUMSV,CTERM,HOLD
      DOUBLE PRECISION DSAVE,DHOLD,DEXTRP,DEXTSV
      DIMENSION S(101),SAVE(101),HOLD(101),DSAVE(101),DHOLD(101)
      COMMON /SLAB/ BV(10,2),TFIN(10),B(2,100),C(9,100),
     X     EL,D,T,X,ARGMN,FX,ABERF,NTIMP,NTRMS,KW,NTACT
C
      QABS(ARG)=DABS(ARG)
      QEXP(ARG)=DEXP(ARG)
      QCMPL(ARGA,ARGB)=DCMPLX(ARGA,ARGB)
      QCABS(CARG)=CDABS(CARG)
      QCEXP(CARG)=CDEXP(CARG)
      QIMAG(CARG)=-CI*CARG
C
C                            KRUNC ...  =1 TO USE ALTERNATE PARTIAL
C                                 SUMS (RECOMMENDED VALUE = 1)
      KRUNC=1
      PI=3.141592653589793D0
      RZERO=0
      CZERO=(0.,0.)
      CI=(0.,1.)
C                            COMPUTE MACHINE EPSILON.
      UNITR=1.
      RTEN=10.
      EPS=1.
    1 EPS=EPS/RTEN
      XPLUS=UNITR+EPS
      IF(XPLUS.NE.UNITR) GO TO 1
      EPS=EPS*RTEN
C                            FIND THE TIME PERIOD IN WHICH T LIES.
      NTMU=NTIMP-1
      DO 2000 KT=1,NTMU
      IF(T.LE.TFIN(KT)) GO TO 2010
 2000   CONTINUE
      KT=NTIMP
 2010 NTUSE=1
```

```
      EXTRP=CZERO
      EXTSV=CZERO
      ABERF=RZERO
C                                TEST FOR A POINT ON A FACE OF THE SLAB.
      IF(X.LE.RZERO) GO TO 2150
      IF(X.GE.EL) GO TO 2150
      IF(KT.LE.1) GO TO 2150
      KTMU=KT-1
      DTDIF=D*(T-TFIN(KTMU))
      SUM=CZERO
      S(1)=CZERO
      SAVE(1)=CZERO
      HOLD(1)=CZERO
C                                INITIALIZE FOR THE RECURRENCE RELATIONS.
      TFACT=QEXP(-(PI/EL)**2*DTDIF)
      RBSQ=TFACT**2
      RTFAC=RBSQ*TFACT
      CXFAC=QCEXP(QCMPL(RZERO,PI*X/EL))
      XFACT=CXFAC
C                                LOOP OVER THE TERMS IN THE FOURIER SERIES.
      DO 2090 J=1,NTRMS
      CTERM=QCMPL(C(KTMU,J)*TFACT,RZERO)*XFACT
      ABERF=ABERF+EPS*QCABS(CTERM)
        SUMSV=SUM
      SUM=SUM+CTERM
      IF(QCABS(SUM-SUMSV).NE.RZERO) GO TO 2080
      IF(QABS(C(KTMU,J)).NE.RZERO) GO TO 2070
 2080   S(J+1)=SUM
        SAVE(J+1)=SUM
        HOLD(J+1)=CZERO
        NTUSE=J+1
C                                USE THE RECURRENCE RELATIONS TO COMPUTE
C                                     TFACT AND XFACT FOR TERM J+1.
      AJPL=J+1
        ARG=-(AJPL*PI/EL)**2*DTDIF
      IF(ARG.GE.ARGMN) GO TO 2050
 2070   EXTRP=SUM
        GO TO 2150
 2050 IF(J.EQ.NTRMS) GO TO 2090
      TFACT=TFACT*RTFAC
      RTFAC=RTFAC*RBSQ
C                                AT THIS POINT,  TFACT=QEXP(ARG) .
      XFACT=XFACT*CXFAC
C                                AT THIS POINT,
C                                    XFACT=QCEXP(QCMPL(RZERO,AJPL*PI*X/EL)).
 2090 CONTINUE
C                                SKIP ALTERNATE PARTIAL SUMS IF REQUESTED.
      NKR=NTUSE
      IF(KRUNC.EQ.0) GO TO 2120
 2100 NKR=0
      JMIN=2-(NTUSE-(NTUSE/2)*2)
      DO 2110 J=JMIN,NTUSE,2
        NKR=NKR+1
        S(NKR)=S(J)
 2110   SAVE(NKR)=SAVE(J)
C                                EXTRAPOLATE TO MAXIMUM ORDERS USING THE
C                                     COMPLEX EPSILON ALGORITHM.
 2120 KTOP=(NKR-1)/2
      DO 2130 JJ=1,NKR
         DSAVE(JJ)=QIMAG(SAVE(JJ))
 2130    DHOLD(JJ)=QIMAG(HOLD(JJ))
      DEXTSV=QIMAG(EXTSV)
      DEXTRP=QIMAG(EXTRP)
      WRITE (KW,9)(DSAVE(J),J=1,NKR)
      WRITE (KW,19)(DHOLD(J),J=1,NKR)
      CALL SHANK (DSAVE,1,NKR,KTOP-1,DEXTSV,DHOLD,1)
      NKR=NKR-2*(KTOP-1)
      WRITE (KW,9)(DSAVE(J),J=1,NKR)
      WRITE (KW,19)(DHOLD(J),J=1,NKR)
      CALL SHANK (DSAVE,1,NKR,1,DEXTRP,DHOLD,1)
      WRITE (KW,9)(DSAVE(J),J=1,NKR)
      WRITE (KW,19)(DHOLD(J),J=1,NKR)
```

```
 9     FORMAT (//1X,'SAVE=',6D18.10/(6X,6D18.10))
 19    FORMAT (//1X,'HOLD=',6D18.10/(6X,6D18.10))
       EXTRP=QCMPL(RZERO,DEXTRP)
       PRINT,'EXTRP=',EXTRP
       ABERF=QABS(DEXTRP-DEXTSV)
       PRINT,'DABERF=',ABERF
C
C   EXTRACT THE IMAGINARY PART OF THE EXTRAPOLATED VALUE, AND ADD
C   THE LINEAR PART OF THE SOLUTION.
C
 2150 FX=QIMAG(EXTRP)+BV(KT,1)+(BV(KT,2)-BV(KT,1))*(X/EL)
       NTACT=NTUSE-1
       RETURN
       END
```

```
      SUBROUTINE SHANK (S,NMIN,NMAX,KMAX,EXTRP,H,NPMIN)
C
C
C              UPDATE EPSILON ALGORITHM
C       FOR ACCELERATING THE CONVERGENCE OF A SEQUENCE.
C
C       AUTHOR: HUI WEN CHIANG, COMPUTER SCIENCE DEPT.,
C                               OKLAHOMA STATE UNIVERSITY.
C
C   THIS PROCEDURE IS MODIFIED FROM THE ALOGORITHM 215, COMM.A.C.M. 6
C   (1963) P.662 (AUTHOR: H. C. THACHER, JR.)  TO ALLOW NOT ONLY
C   ACCELERATING THE CONVERGENCE OF A SEQUENCE BUT ALSO ACCELERATING
C   THE CONVERGENCE BY EXTENDING THE ORIGINAL SEQUENCE VERTICALLY
C   (ADD MORE ENTRIES) OR HORIZONTALLY (COMPUTE THE FURTHER ORDER
C   OF EXTRAPOLATED VALUE) BY USING THE PREVIOUS COMPUTED VALUES
C   WITHOUT REPEATING THE ORIGINAL PROCESSES.    09-15-84
C       THE ARRAY H HOLDS THE VALUES IN PREVIOUS E(i) COLUMN
C   FOR EXTENDED PROCESSING PURPOSE.
C
C   HINTS TO EXTEND THE SEQUENCE VERTICALLY:
C          1)  ADD THE ELEMENTS TO THE BOTTOM OF SEQUENCE
C          2)  SPECIFY THE INDEX NO OF THE FIRST CURRENT ADDED ENTRY    N
C                 TO VARIABLE NMIN
C          3)  SPECIFY THE INDEX NO OF THE LAST CURRENT ADDED ENTRY
C                 TO VARIABLE NMAX
C   HINTS TO EXTEND THE SEQUENCE HORIZONTALLY:
C          1)  SPECIFY THE FURTHER ORDER OF EXTRAPOLATED VALUE
C              NEEDED TO VARIABLE KMAX.
C          2)  COMPUTE THE TOTAL ELEMENTS OF THE CURRENT LIST AND MOVE
C              THE FIRST AND LAST ENTRIES TO NMIN, NMAX RESPECTIVELY.
C   HINTS TO EXTEND THE SEQUENCE VERTICALLY AND HORIZONTALLY:
C          1)  EXTEND VERTICALLY FIRST THEN HORIZONTIALL IS SUGGESTED.
C
C   THIS PROCEDURE REPLACES THE ELEMENTS S(NMIN) THROUGH S(NMAX-2*KMAX)
C   OF THE ARRAY S BY THE E(KMAX) TRANSFORM OF THE SEQUENCE S.
C   THE ELEMENTS S(NMAX-2*KMAX+1) THROUGH S(NMAX-1) ARE DESTROYED.
C   THE HIGHEST ORDER ELEMENT OF THE TRANSFORM IS RETURNED IN EXTRP.
C
C   THE MAXIMUM PERMISSIBLE VALUE OF KMAX IS (NMAX-NMIN)/2 .
C   THEREFORE, TO FIND THE HIGHEST ORDER EXTRAPOLATED VALUE OF THE
C   SEQUENCE S(NMIN),...,S(NMAX), PROCEED THUS....
C        CALL SHANK (S,NMIN,NMAX,(NMAX-NMIN)/2,EXTRP)
C   AND THE EXTRAPOLATED VALUE IS RETURNED IN EXTRP.
C
C   REFERENCES ....
C        D. SHANKS, J. MATH. AND PHYSICS 34 (1955) 1-42
C        D. C. JOYCE, S.I.A.M. REVIEW 13 (1971) 435-490
C        W. B. GRAGG, S.I.A.M. REVIEW 14 (1972) 1-62
C
C
C   J. P. CHANDLER, COMPUTER SCIENCE DEPT., OKLAHOMA STATE UNIVERSITY
C
C   H. C. THACHER, JR., ALGORITHM 215, COMM.A.C.M. 6 (1963) P. 662
C
C   THIS PROCEDURE REPLACES THE ELEMENTS S(NMIN) THROUGH S(NMAX-2*KMAX)
C   OF THE ARRAY S BY THE E(KMAX) TRANSFORM OF THE SEQUENCE S.
C   THE ELEMENTS S(NMAX-2*KMAX+1) THROUGH S(NMAX-1) ARE DESTROYED.
C   THE HIGHEST ORDER ELEMENT OF THE TRANSFORM IS RETURNED IN EXTRP.
C
C   THE MAXIMUM PERMISSIBLE VALUE OF KMAX IS (NMAX-NMIN)/2 .
C   THEREFORE, TO FIND THE HIGHEST ORDER EXTRAPOLATED VALUE OF THE
C   SEQUENCE S(NMIN),...,S(NMAX), PROCEED THUS....
C        CALL SHANK (S,NMIN,NMAX,(NMAX-NMIN)/2,EXTRP)
C   AND THE EXTRAPOLATED VALUE IS RETURNED IN EXTRP.
C
C   REFERENCES ....
C        D. SHANKS, J. MATH. AND PHYSICS 34 (1955) 1-42
C        D. C. JOYCE, S.I.A.M. REVIEW 13 (1971) 435-490
C        W. B. GRAGG, S.I.A.M. REVIEW 14 (1972) 1-62
C
C   TO CONVERT THIS ROUTINE FROM COMPLEX TO DOUBLE PRECISION, REPLACE
C   SIX STATEMENTS BELOW BY THE FOLLOWING FIVE...
```

```
C      DOUBLE PRECISION S,EXTRP,HUGE,ZERO,UNITY,TZ,TU,ARG,QTEST
C      QTEST(ARG)=ARG
C      HUGE=1.E35
C      ZERO=0
C      UNITY=1
C
C
C
       DOUBLE PRECISION QTEST,DABS,S,H,EXTRP,HUGE,ZERO,UNITY,TZ,TU,ARG
       DIMENSION S(1),H(1)
       QTEST(ARG)=DABS(ARG)
       HUGE=1.0D35
       ZERO=0.0D0
       UNITY=1.0D0
C
       IF (NMIN.GT.NMAX)GO TO 130
       EXTRP=S(NMAX)
       KT=2*KMAX
       DO 120 J=NMIN,NMAX
          TZ=H(J)
          LIMKK=J-NPMIN
          IF (LIMKK.GT.KT)LIMKK=KT
          IF (LIMKK.LE.0) GO TO 120
          DO 110 KK=1,LIMKK
             JMKK=J-KK
             IF (QTEST(S(JMKK)).EQ.HUGE)GO TO 80
             IF (QTEST(S(JMKK+1)).EQ.HUGE)GO TO 80
             TU=S(JMKK+1)-S(JMKK)
             IF (QTEST(TU).EQ.ZERO)GO TO 90
             TU=TZ+UNITY/TU
             GO TO 100
C
  80         TU=TZ
             GO TO 100
  90         TU=HUGE
  100        TZ=S(JMKK)
  110        S(JMKK)=TU
             H(JMKK)=TZ
  120      CONTINUE
C
           MXMKT=NMAX-KT
           EXTRP=S(MXMKT)
  130      RETURN
           END
```

APPENDIX B

PROGRAM LISTING OF THE EPSILON ALGORITHM WITH THE UPDATE FORM AND SINGULAR RULE

```
$JOB
C*******************************************************************
C                THE   EPSILON ALOGRITHM                           *
C    FOR ACCELERATING THE CONVERGENCN OF A SEQUENCE.                *
C                                                                  *
C    AUTHOR: HUI WEN CHIANG, COMPUTER SCIENCE DEPT.,               *
C                           OKLAHOMA STATE UNIVERSITY.             *
C    THIS PROCEDURE CONTAINS TWO VERSIONS OF THE EPSILON ALGORITHM, *
C    1) THE UPDATE EPSILON ALGORITHM THAT ALLOWS TO EXTEND EPSILON  *
C       ALGORITHM PROCESSING VERTICALLY AND HORIZONTALLY,           *
C    2) THE EPSILON ALGORITHM WITH SINGULAR RULE.                   *
C    ONE OF THE OPTION CAN APPLY RESPECTIVELY, AS THE SINGULAR RULE *
C    DOES NOT WORK PROPERLY WITH SINGULAR RULE.                     *
C    THE VARIABLE "SINGUL" ACTS AS A FLAG BETWEEN THOSE TWO VERSIONS*
C                                                                  *
C*******************************************************************
C
C     VARIABLES:
C
C     NPMIN  --  FIRST ELEMENT NUMBER IN THE S LIST.
C     NMIN   --  FIRST ELEMENT NUMBER IN THE LIST THAT NEEDS TO BE
C               PROCESSED.  IF THE LIST EXTENED VERTICALLY AFTER FIRST
C               CALL SHANK NMIN CONTAINS THE FIRST ADDED ELEMENT NUMBER.
C     NMAX   --  LAST ELEMENT NUMBER IN THE LIST.  IF THE LIST EXTENED
C               HORIZONTIALLY AFTER FIRST CALL SHANK NMAX SHOULD BE
C               COMPUTED BY NMAX = NMAX - 2 * KMAX
C     KMAX   --  ORDER OF EXTRAPOLATED
C     H      --  HOLD THE VALUES OF PREVIOUS E(I) COLUMN
C     S      --  THE EPSILON ARRAY
C     EXTRP  --  HOLD THE EXTRAPOLATED VALUE
C     SINGUL --  FLAG TO APPLY SINGULAR RULES
C                SINGUL = 0      NO SINGULAR RULE APPLIED
C                SINGUL = 1      SINGULAR RULE APPLIED
C
C******************************************************************
C*********VARIABLES THAT APPLIED IN SINGULAR RULES ****************
C***  AP      -  A ABOUT TO BE COMPUTED                            *
C***  BP      -  B ABOUT TO BE COMPUTED                            *
C***  CP      -  C ABOUT TO BE COMPUTED                            *
C***  BOL     -  BEGIN OF LIST INDEX                               *
C***  EOL     -  END OF LIST INDEX                                 *
C***  SA      -  SMALL A                                           *
C***  A(*)    -  A ARRAY                                           *
C***  B(*)    -  B ARRAY                                           *
C***  D       -  D VALUE                                           *
C***  CANCEL  -  CANCELLATION                                      *
C***  S1(*)   -  S1 ARRAY                                          *
C***  NPINS   -  NEW POINT INSTABILITY                             *
C***  LOCINS  -  LOCAL INSTABILITY                                 *
C***  NIS     -  NON ISOLATED SINGULARITY                          *
C******************************************************************
      DOUBLE PRECISION S,H,EXTRP
      DIMENSION S(20),H(20)
C
C
      DOUBLE PRECISION A(5),B(5),SA,D,CANCEL
      INTEGER AP,BP,CP,NIS,LOCINS,BOL,EOL,NPINS,S1(5)
      COMMON /SING/ A,B,SA,D,CANCEL,AP,BP,CP,NIS,
X              LOCINS,BOL,EOL,NPINS,S1
C                              INITIALIZATION
      JCOUNT=1
      KW=6
      KR=5
      NPMIN=1
      NMIN=1
      NMAX=10
      KMAX=4
C                              INITIALIZATION FOR SINGULAR ROUTINE
      CANCEL=1.D-10
      NIS=0
      NPINS=0
      LOCINS=0
```

```
      CP=0
      BOL=1
      EOL=0
C
C                                  READ IN SEQUENCE ELEMENT NUMBER
      READ(KR,2)NSETS
    2 FORMAT(I5)
C                                  COMPUTE THE SERIES FUNCTION
      CALL PATH(S,H,NSETS)
C     CALL ITER(S,H,NSETS)
C     CALL LEIBNZ(S,H,NSETS)
C     CALL LN2(S,H,NSETS)
      WRITE (KW,5)(S(J),J=NPMIN,NSETS)
      WRITE (KW,10)(H(J),J=NPMIN,NSETS)
C                              CALL TO COMPUTE THE EXTRAPOLATED VALUE OF S
      IF (SINGUL.EQ.0) CALL SHANK(S,NMIN,NMAX,KMAX,EXTRP,NPMIN,H)
      IF (SINGUL.EQ.1) CALL SINGULR(S,MIN,NMAX,KMAX,EXTRP)
C
C
      KEND=NMAX-2*KMAX
      WRITE (KW,10)(H(J),J=NPMIN,KEND)
      WRITE (KW,20)(S(J),J=NPMIN,KEND)
      WRITE (KW,30)EXTRP
C
C                                    END OF THE SINGULAR RULE PROCESSES
      IF (SINGUL.EQ.1)GO TO 99
C                                                ADD ONE MORE ELEMENT
C                                                EXTEND VERTICALLY
      NMIN=5
      NMAX=5
      KMAX=2
      WRITE (KW,5)(S(J),J=NMIN,NMAX)
      CALL SHANK(S,NMIN,NMAX,KMAX,EXTRP,NPMIN,H)
      KEND=NMAX-2*KMAX
      WRITE (KW,20)(S(J),J=NPMIN,KEND)
      WRITE (KW,10)(H(J),J=NPMIN,KEND)
      WRITE (KW,30)EXTRP
C
C
C                       CALL TO COMPUTE THE EXTRAPOLATED VALUE OF S
C                       COMPUTE THE FURTHER ORDER OF EXTRAPOLATED VLAUE
C                       EXTEND HORIZONTIALLY
C*    NMIN=NPMIN
C*    NMAX=NMAX-2*KMAX
C     PRVCOL=PRVCOL+2*KMAX
C     NMIN=1
C     NMAX=3
C     KMAX=1
C     CALL SHANK(S,NMIN,NMAX,KMAX,EXTRP,NPMIN,H)
C     KEND=NMAX-2*KMAX
C     WRITE (KW,20)(S(J),J=NPMIN,KEND)
C     WRITE (KW,10)(H(J),J=NPMIN,KEND)
C     WRITE (KW,30)EXTRP
   5  FORMAT (///1X,'S    = ',6E18.10/(7X,6E18.10))
  10  FORMAT (/1X,'H    = ',6E18.10/(7X,6E18.10))
  20  FORMAT (/1X,'E(1)= ',6E18.10/(7X,6E18.10))
  30  FORMAT (/1X,'EXTRP=',E18.10)
  99  STOP
      END
C
```

```
      SUBROUTINE SHANK (S,NMIN,NMAX,KMAX,EXTRP,NPMIN,H)
C
C************ UPDATE EPSILON ALGORITHM ****************************
C  EPSILON ALGORITHM FOR ACCELERATING THE CONVERGENCE OF A SENQUENCE.
C
C  THIS PROCEDURE IS MODIFIED FROM THE ALOGORITHM 215, COMM.A.C.M. 6
C  (1963) P.662 (AUTHOR: H. C. THACHER, JR.)  TO ALLOW NOT ONLY
C  ACCELERATING THE CONVERGENCE OF A SEQUENCE BUT ALSO ACCELERATING
C  THE CONVERGENCE BY EXTENDING THE ORIGINAL SEQUENCE VERTICALLY
C  (ADD MORE ENTRIES) OR HORIZONTIALLY (COMPUTE THE FURTHER ORDER
C  OF EXTRAPOLATED VALUE) BY USING THE PREVIOUS COMPUTED VALUES
C  WITHOUT REPEATING THE ORIGINAL PROCESSES.    09-15-84
C
C  HINTS TO EXTEND THE SEQUENCE VERTICALLY:
C        1)  ADD THE ENTRIES TO THE BOTTOM OF SEQUENCE
C        2)  SPECIFY THE INDEX OF THE FIRST CURRENT ADDED ENTRY
C               TO VARIABLE NMIN
C        3)  SPECIFY THE INDEX OF THE LAST CURRENT ADDED ENTRY
C               TO VARIABLE NMAX
C  HINTS TO EXTEND THE SEQUENCE HORIZONTIALLY:
C        1)  SPECIFY THE FURTHER ORDER OF EXTRAPOLATED VALUE
C            NEEDED TO VARIABLE KMAX.
C            I.E.  KMAX = THE FURTHER ORDER - THE CURRENT ORDER
C        2)  COMPUTE THE TOTAL ELEMENTS OF THE CURRENT LIST AND MOVE
C            THE FIRST & LAST ELEMENT # TO NMIN, NMAX RESPECTIVELY.
C  HINTS TO EXTEND THE SEQUENCE VERTICALLY AND HORIZONTIALLY:
C        1)  EXTEND VERTICALLY FIRST THEN HORIZONTIALL IS SUGGESTED.
C
C  THIS PROCEDURE REPLACES THE ELEMENTS S(NMIN) THROUGH S(NMAX-2*KMAX)
C  OF THE ARRAY S BY THE E(KMAX) TRANSFORM OF THE SEQUENCE S.
C  THE ELEMENTS S(NMAX-2*KMAX+1) THROUGH S(NMAX-1) ARE DESTROYED.
C  THE HIGHEST ORDER ELEMENT OF THE TRANSFORM IS RETURNED IN EXTRP.
C
C  THE MAXIMUM PERMISSIBLE VALUE OF KMAX IS (NMAX-NMIN)/2 .
C  THEREFORE, TO FIND THE HIGHEST ORDER EXTRAPOLATED VALUE OF THE
C  SEQUENCE S(NMIN),....,S(NMAX), PROCEED THUS....
C       CALL SHANK (S,NMIN,NMAX,(NMAX-NMIN)/2,EXTRP)
C  AND THE EXTRAPOLATED VALUE IS RETURNED IN EXTRP.
C
C  REFERENCES ....
C       D. SHANKS, J. MATH. AND PHYSICS 34 (1955) 1-42
C       D. C. JOYCE, S.I.A.M. REVIEW 13 (1971) 435-490
C       W. B. GRAGG, S.I.A.M. REVIEW 14 (1972) 1-62
C
C
      DOUBLE PRECISION QTEST,DABS,S,H,EXTRP,HUGE,ZERO,UNITY,TZ,TU,ARG
      DIMENSION S(1),H(1)

C     QTEST(ARG)=ABS(ARG)
      QTEST(ARG)=DABS(ARG)
      SINGUL=0
      HUGE=1.0E35
      ZERO=0.0E0
      UNITY=1.0E0
C
      IF (NMIN.GT.NMAX)GO TO 130
      EXTRP=S(NMAX)
      KT=2*KMAX
      DO 120 J=NMIN,NMAX
         TZ=H(J)
         LIMKK=J-NPMIN
         IF (LIMKK.GT.KT)LIMKK=KT
         IF (LIMKK.LE.0) GO TO 120
         DO 110 KK=1,LIMKK
            JMKK=J-KK
            IF (QTEST(S(JMKK)).EQ.HUGE)GO TO 80
            IF (QTEST(S(JMKK+1)).EQ.HUGE)GO TO 80
            TU=S(JMKK+1)-S(JMKK)
            IF (QTEST(TU).EQ.ZERO)GO TO 90
            TU=TZ+UNITY/TU
            GO TO 100
   80       TU=TZ
```

```
            GO TO 100
 90         TU=HUGE
 100        TZ=S(JMKK)
 110        S(JMKK)=TU
            H(JMKK)=TZ
 120      CONTINUE
C
 125      MXMKT=NMAX-KT
          EXTRP=S(MXMKT)
 130      RETURN
C
C
```

```
      SUBROUTINE SINGULR(S,NMIN,NMAX,KMAX,EXTRP)
C
C  THIS PROCEDURE APPLIES THE SINGULAR RULE TO THE EPSILON ALGORITHM.
C  THE PROCEDURE IS MODIFIED FROM P. WYNN "SINGULAR RULES FOR CERTAIN
C  NON-LINEAR ALGORITHMS" IN BIT 3 1963, P 175-195.
C    THE SINGULAR RULE AVOIDS THE INSTABILE ENTRY IN THE EPSILON
C  TABLE BY APPLYING THE SPECIAL RULE BUT THE REGULAR EPSILON
C  ALGORITHM.
C
      DOUBLE PRECISION QTEST,DABS,S,EXTRP,HUGE,ZERO,UNITY,TZ,TU,ARG
      DIMENSION S(1)
      INTEGER SINGUL
      DOUBLE PRECISION A(15),B(15),SA,D,CANCEL
      INTEGER AP,BP,CP,NIS,LOCINS,BOL,EOL,NPINS,S1(15)
      COMMON /SING/ A,B,SA,D,CANCEL,AP,BP,CP,NIS,
     X               LOCINS,BOL,EOL,NPINS,S1
C     QTEST(ARG)=ABS(ARG)
      QTEST(ARG)=DABS(ARG)
C     SINGUL=0
      HUGE=1.0E35
      ZERO=0.0E0
      UNITY=1.0E0
C
      IF (NMIN.GT.NMAX)GO TO 130
      EXTRP=S(NMAX)
      KT=2*KMAX
      DO 120 J=NMIN,NMAX
         TZ=0
         LIMKK=J-NMIN
         IF (LIMKK.GT.KT)LIMKK=KT
         IF (LIMKK.LE.0) GO TO 120
         DO 110 KK=1,LIMKK
            JMKK=J-KK
            IF (SINGUL.EQ.1)GOTO 190
C                                      SINGULAR RULE APPLIED
  190       TU=S(JMKK+1)-S(JMKK)
            IF(KK+PRVCOL.GE.J)GOTO 200
            IF(CP.NE.0)GOTO 200
            IF(TU.EQ.ZERO)GOTO 195
            IF(S(JMKK).EQ.ZERO)GOTO 200
            IF(DABS(TU/S(JMKK)).GE.CANCEL)GOTO 200
  195       NPINS=1
            EOL=EOL+1
  200       IF (NPINS.EQ.1)GOTO210
            IF(LOCINS.EQ.1)GOTO230
C                                       COMPUTE AUX0
            IF(TU.NE.ZERO)TU=1.0/TU + TZ
            IF(TU.EQ.ZERO)TU=HUGE
            GOTO 100
C                                       DO ONLY WHEN
C*                                      (LOCINS.EQ.1).AND.(IS.EQ.S1(BOL))
  210       IF (LOCINS.NE.1)GOTO 220
            IF (KK+PRVCOL.NE.S1(BOL))GOTO 220
            NIS=1
            GOTO 125
  220       LOCINS=1
            AP=1
            NPINS=0
            S1(EOL)=KK+PRVCOL
  230       IF(CP.EQ.1)GOTO 240
C
            IF(TU.NE.ZERO)TU=1.0/TU + TZ
            IF(TU.EQ.ZERO)TU=HUGE
            GO TO 250
  240       SA=D+B(BOL)-A(BOL)
C                                           COMPUTE FUNCTION OF
C                                           TU=SA/(1.0+SA/S(JMKK+1))
            IF(TZ.EQ.ZERO)TU=ZERO
            IF(TZ.NE.ZERO)TU=SA/(1.0+SA/TZ)
            CP=0
            BOL=BOL+1
            IF(BOL.GT.EOL)LOCINS=0
```

```
  250       IF(LOCINS.NE.1)GO TO 100
            IF(AP.EQ.0)GO TO 260
            A(EOL)=0.0
            IF(KK.EQ.1)GOTO 255
C                                             COMPUTE FUNCTION OF
C                                           A(EOL)=L(IS-1)/(1.0-L(IS-1)/AUX0)
            IF(TU.EQ.ZERO)A(EOL)=ZERO
            IF(TU.NE.ZERO)A(EOL)=TZ/(1.0-TZ/TU)
  255       AP=0
            BP=1
            GOTO 100
  260       IF (BP.EQ.1)GOTO 270
            IF (KK.EQ.1)GOTO 100
            IF (KK.NE.S1(BOL)+1)GOTO 100
C                                            D=AUX1/(1.0-AUX1/L(IS))
            IF(S(JMKK).EQ.ZERO)D=ZERO
            IF(S(JMKK).NE.ZERO)D=S(JMKK+1)/(1.0-S(JMKK+1)/S(JMKK))
            CP=1
            GOTO 100
  270       IF (BP.NE.1)GOTO 100
C                                            B(EOL)=L(IS)/(1.0-L(IS)/AUX1)
            IF(S(JMKK+1).EQ.ZERO)B(EOL)=HUGE
            IF(S(JMKK+1).NE.ZERO)B(EOL)=S(JMKK)/(1.0-S(JMKK)/S(JMKK+1))
            BP=0
C
C
C                                              END OF SINGULAR RULES
C
  100       TZ=S(JMKK)
  110       S(JMKK)=TU
C
  120     CONTINUE
C
  125     MXMKT=NMAX-KT
          EXTRP=S(MXMKT)
  130     RETURN
          END
END
```

```
C*******************************************************************
C    EXAMPLE #1: LINEARLY CONVERGENT ITERATIVE SCHEME FORMATION OF *
C                THE LAGUERRE POLYNOMIAL SERIES.                   *
C          S(N+1)= 1/4 * (S(N)**2 + 2)                             *
C*******************************************************************
      SUBROUTINE ITER(S,H,NSETS)
C     DOUBLE PRECISION S,H
      DIMENSION S(20),H(20)
C
       S(1)=0.0
       H(1)=0.0
       DO 10 J=2,NSETS
          S(J)=0.25 * (S(J-1)**2 + 2)
          H(J)=0.0
  10   CONTINUE
       RETURN
       END
C******************************************************************
C    EXAMPLE #2: THE VERY SLOWLY CONVERGENT LEIBNITZ SERIES       *
C                             (N-1)                               *
C   TT     S(N)= S(N-1) + (-1) * (4 / (N * 2 - 1))                *
C******************************************************************
C
      SUBROUTINE LEIBNZ(S,H,NSETS)
C     DOUBLE PRECISION S,H
      DIMENSION S(20),H(20)
C
      S(1)=4.0
      H(1)=0.0
      DO 10 J=2,NSETS
         S(J)=S(J-1) + ((-1) ** (J-1)) * (4.0 / (J * 2.D0 - 1.D0))
         H(J)=0.0
  10  CONTINUE
      RETURN
      END
C*******************************************************************
C*     THIS COMPUTE SERIES LN(2)                                   *
C*            &                                                    *
C*     LN(2)= E ((-1) ** M) /  (M+1)                               *
C*           M=0                                                   *
C*******************************************************************
      SUBROUTINE LN2(S,H,NSETS)
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION TERM(2)
      DOUBLE PRECISION S(20),H(20)
C
      S(1)=1.0
      H(1)=0.0
      DO 10 J=2,NSETS
      S(J)=(((-1.)**(J-1)) / J) + S(J-1)
      H(J)=0.0
  10  CONTINUE
      RETURN
      END
C******************************************************************
C    PATHOLOGICAL EXAMPLES    P 177                               *
C    THIS IS TEST SERIES:  U(S) = (X ** S) / S !    S=0,1,...     *
C******************************************************************
      SUBROUTINE PATH(S,H,M)
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION S(20),DENO,H(20)
C
      S(1)=0.0
      H(1)=0.0
      DO 30 K=2,M
      DENO=1.
      IF (K.LT.4)GOTO 20
      DO 10 J=4,K
   10 DENO=DENO*(J-2)
   20 S(K)= (2 ** (K-2)) / DENO + S(K-1)
```

```
      H(K)=0.0
   30 CONTINUE
      RETURN
      END
```

APPENDIX C

PROGRAM LISTING OF ERROR EXPANSION SERIES

```
C*********************************************************************
C****                     ERROR EXPANSION SERIES
C****          AUTHOR: HUI WEN CHIANG, COMPUTER SCIENCE DEPT.,
C****                                OKLAHOMA STATE UNIVERSITY.
C****
C****THIS PROGRAM GENERATE A SERIES EXPANSION IN NEGATIVE POWERS
C****OF N OF THE MAGNITUDE OF THE ERROR IN THE PARTIAL SUMS OF AN
C****INFINITE SERIES.
C****
C*******************************************************************
C     STEPS TO OBTAIN ERROR EXPANSION SERIES:
C  I.
C    Suppose there exists an error expansion series which is
C    of the following form:
C
C                C1       C2           C3
C     [e(n)[ = ---- + --------- + --------- + ...,
C              n+a1   (n+a2)**2    (n+a3)**3
C
C  II.
C   To compute constant C1 the following steps can be followed.
C   Compute S(n), n = 1,2,3,...,N.
C   Compute e(n) = [ ln2 - S(n) [.
C   Then, apply equation C1 = n * [e(n)[ to get the constant
C   C1, by applying a repeated Aitken, Epsilon, or Romberg
C   algorithm to estimate C1 as n--¶ .
C III.
C   To compute C2.
C   Use the true value of C1, if it can be seen to converge
C   to a fraction.
C   Repeat the above process by using equation
C     C2 = ([e(n)[ - C1/n) * (n**2)
C          .
C          .
C*******************************************************************
      IMPLICIT REAL*16 (A-H,O-Z)
C     DOUBLE PRECISION NUMER,DENOM,PRODCT,RATIO,POWRN
C     DOUBLE PRECISION  A(100),XTRAB(100,12),COEF(30),S(100),CONST(30)
C     DOUBLE PRECISION  ARG
      DIMENSION  A(100),XTRAB(100,12),COEF(30),CONST(30),S(100)
      REAL NUMER
C
      LX=100
      NTERMS=2048
      NCOLS=7
      KW=6
      JMNPRT=1
C     JCOEF=10
      JCOEF=9
C                             INITIZATION
      COEF(1)=0.0D0
      COEF(2)=0.0D0
      COEF(3)=0.0D0
      COEF(4)=0.0D0
      COEF(5)=0.0D0
      COEF(6)=0.0D0
      COEF(7)=0.0D0
      COEF(8)=0.0D0
      COEF(9)=0.0D0
C                              INITIZATION
      CONST(1)=0.0D0
      CONST(2)=0.0D0
      CONST(3)=0.0D0
      CONST(4)=0.0D0
      CONST(5)=0.0D0
      CONST(6)=0.0D0
      CONST(7)=0.0D0
      CONST(8)=0.0D0
      CONST(9)=0.0D0
C                           COMPUTE FUNCTION VALUES
      CALL COMPA (A,NTERMS)
      DO 2 J=1,JCOEF
```

```
      IF(J/2*2.EQ.J)GO TO 2
      IF(J.GE.JMNPRT) WRITE(KW,6)J
    6 FORMAT(/'1CONSTANT C  OF N**(-',I2,')'//1X)
      NN=1
      DO 4 N=1,NTERMS
      EN=NN
      POWRN=EN
      SUM=0.0D0
      IF(J.LT.2) GO TO 5
      JCM=J-1
      DO 3 K=1,JCM
C     L=J-K
      SUM=SUM-COEF(K)*(EN**J)/((EN+CONST(K))**K)
    3 POWRN=POWRN*EN
    5 SUM=SUM+A(N)*POWRN
      S(N)=SUM
      XTRAB(N,1)=SUM
    4 NN=NN+NN
      CALL SHANK(S,1,NTERMS,6,EXTRP)
      WRITE(KW,210)EXTRP
      LP=KW
      IF(J.LT.JMNPRT) LP=0
      CALL AITABL (XTRAB,LX,NTERMS,NCOLS,LP)
      RATIO=2.0D0
C     JJ=3
C     K=4
C   1 XTRAB(JJ,1)=XTRAB(K,1)
C     JJ=JJ+1
C     K=K+K
C     IF(K.LT.NTERMS) GO TO 1
C     CALL ROMBEX (XTRAB,LX,JJ-1,RATIO,1,LP)
C     CALL ROMBEX (XTRAB,LX,JJ-1,RATIO,2,LP)
C     CALL AITABL (XTRAB,LX,JJ-1,NCOLS,LP)
      CALL ROMBEX (XTRAB,LX,NTERMS,RATIO,1,LP)
      CALL ROMBEX (XTRAB,LX,NTERMS,RATIO,2,LP)
C
C    SECOND DO LOOP TO COMPUTE VALUE A
C
      IF(J.GE.JMNPRT) WRITE(KW,16)J
   16 FORMAT(/'1CONSTANT A  OF N**(-',I2,')'//1X)
      IF (J.EQ.1)COEF(1)=0.50D0
      IF (J.EQ.3)COEF(3)=-0.125D0
      IF (J.EQ.5)COEF(5)=0.15625D0
      IF (J.EQ.7)COEF(7)=-0.4765625D0
      IF (J.EQ.9)COEF(9)=0.27050D1
      NN=1
      DO 40 N=1,NTERMS
      DENOM=0.0D0
      DO 50 L=1,J
      POWRN=NN
      PRODCT=1.D0
      JJ=J-1
      IF(JJ.LT.1)GO TO 120
      DO 100 M=1,JJ
      IF(L.EQ.M) PRODCT=PRODCT*COEF(M)
C     IF(L.NE.M) PRODCT=PRODCT * (POWRN + CONST(M))
      IF(L.NE.M) PRODCT=PRODCT * ((POWRN + CONST(M))**M)
C     POWRN=POWRN*NN
  100 CONTINUE
  120 IF (L.EQ.J)GO TO 150
      DENOM=DENOM - PRODCT
      GO TO 50
  150 NUMER=PRODCT * COEF(J)
      PRODCT=PRODCT* A(N)
      DENOM=DENOM+PRODCT
   50 CONTINUE
C     WRITE(KW,88)N,NUMER,DENOM,POWRN
C 88  FORMAT(/1X,I4,3X,'NUMBER=',D28.17,2X,'DENOM=',D28.17,5X,D28.17)
C     S(N)=NUMER / DENOM - POWRN
      ARG=NUMER/DENOM
      XTRAB(N,1)=ARG ** (1.D0/J)
      XTRAB(N,1)=XTRAB(N,1) - NN
```

```
      S(N)=XTRAB(N,1)
   40 NN=NN+NN
C   4 XTRAB(N,1)=SUM
      CALL SHANK(S,1,NTERMS,6,EXTRP)
      WRITE(KW,210)EXTRP
  210 FORMAT(/1X,'EXTRP=',D28.17)
      LP=KW
      IF(J.LT.JMNPRT) LP=0
      CALL AITABL (XTRAB,LX,NTERMS,NCOLS,LP)
      RATIO=2.0D0
C     JJ=3
C     K=4
C  10 XTRAB(JJ,1)=XTRAB(K,1)
C     JJ=JJ+1
C     K=K+K
C     IF(K.LT.NTERMS) GO TO 10
C     CALL ROMBEX (XTRAB,LX,JJ-1,RATIO,1,LP)
C     CALL ROMBEX (XTRAB,LX,JJ-1,RATIO,2,LP)
C     CALL AITABL (XTRAB,LX,JJ-1,NCOLS,LP)
      CALL ROMBEX (XTRAB,LX,NTERMS,RATIO,1,LP)
      CALL ROMBEX (XTRAB,LX,NTERMS,RATIO,2,LP)
C
      IF(J.EQ.1)CONST(1)=0.5D0
      IF(J.EQ.3)CONST(3)=0.5D0
      IF(J.EQ.5)CONST(5)=0.5D0
      IF(J.EQ.7)CONST(7)=0.5D0
    2 CONTINUE
      STOP
      END
C
      SUBROUTINE COMPA (A,NTERMS)
C
C     COMPUTES THE MAGNITUDE OF THE ERROR IN THE PARTIAL SUMS OF THE
C     ALTERNATING HARMONIC SERIES.
C
      IMPLICIT REAL*16 (A-H,O-Z)
      DIMENSION A(1)
C
      RTWO=2.0D0
C     XLIM=DLOG(RTWO)
      XLIM=QLOG(RTWO)
      PSUM=0.0D0
      DSIGN=1.0D0
      NN=0
      JJ=1
      DO 1 J=1,NTERMS
      AJ=J
      TERM=DSIGN/AJ
      DSIGN=-DSIGN
      PSUM=PSUM+TERM
      IF (J.NE.JJ)GO TO 1
      NN=NN+1
      A(NN)=QABS(PSUM-XLIM)
      JJ=JJ+J
    1 CONTINUE
      NTERMS=NN
      WRITE(6,10)(A(J),J=1,NTERMS)
   10 FORMAT(/1X,'A = ',4D28.17/(5X,4D28.17))
      RETURN
      END
C
C
      SUBROUTINE AITABL (XTRAB,LX,NROWS,NCOLS,KW)
C
C  COMPUTES A TABLE OF REPEATED AITKEN EXTRAPOLATION AND, IF KW IS
C  POSITIVE, PRINTS THE TABLE ON OUTPUT UNIT NUMBER KW.
C  THE GIVEN INPUT SEQUENCE MUST BE PROVIDED IN (XTRAB$(J,1),J=1,NROWS).
C  LX IS THE FIRST DIMENSION OF THE ARRAY XTRAB.
C
C
      IMPLICIT REAL*16 (A-H,O-Z)
C     DOUBLE PRECISION XTRAB,TINY,DENOM,RZERO
```

```
      DIMENSION XTRAB(LX,NCOLS)
C
      TINY=1.D-35
      RZERO=0.D0
      IF(NCOLS.LT.2) RETURN
      IF(NROWS.LT.3 .OR. NROWS.GT.LX) RETURN
C
C                           COMPUTE THE TABLE.
      DO 1 K=2,NCOLS
      JMIN=2*K-1
      IF(JMIN.GT.NROWS) GO TO 3
      DO 2 J=JMIN,NROWS
      DENOM=(XTRAB(J-2,K-1)-XTRAB(J-1,K-1))-
     X      (XTRAB(J-1,K-1)-XTRAB(J,K-1))
      IF(DENOM.EQ.RZERO) DENOM=TINY
C
C                      USE THE FORM WITH LEAST CANCELLATION FOR
C   CONVERGENT SEQUENCES.
    2 XTRAB(J,K)=XTRAB(J,K-1)-(XTRAB(J,K-1)-XTRAB(J-1,K-1))**2/DENOM
    1 CONTINUE
C
    3 IF(KW.LE.0) RETURN
C                           PRINT THE TABLE.
      DO 4 J=1,NROWS
      KMAX=(J+1)/2
      IF(KMAX.GT.NCOLS) KMAX=NCOLS
    4 WRITE(KW,5)J,(XTRAB(J,K),K=1,KMAX)
    5 FORMAT(/1X,I4,4D28.17/(5X,4D28.17))
      RETURN
      END
C
      SUBROUTINE ROMBEX (R,LR,N,RATIO,JPFRST,KW)
C
      IMPLICIT REAL*16 (A-H,O-Z)
      DIMENSION R(LR,N)
      IF(N.LT.2 .OR. N.GT.LR) RETURN
      UNITR=1
      DO 1 J=1,N
      IF(J.EQ.1) GO TO 2
      RPOWR=RATIO*JPFRST
      DO 3 K=2,J
      R(J,K)=R(J,K-1)+(R(J,K-1)-R(J-1,K-1))/(RPOWR-UNITR)
    3 RPOWR=RPOWR*RATIO
    2 IF(KW.GT.0) WRITE(KW,4)J,(R(J,K),K=1,J)
    4 FORMAT(/1X,I4,4D28.17/(5X,4D28.17))
    1 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE SHANK (S,NMIN,NMAX,KMAX,EXTRP)
C
      IMPLICIT REAL*16 (A-H,O-Z)
C     DOUBLE PRECISION QTEST,DABS,S,H,EXTRP,HUGE,ZERO,UNITY,TZ,TU,ARG
      DIMENSION S(1)
C     QTEST(ARG)=ABS(ARG)
C     QTEST(ARG)=DABS(ARG)
      QTEST(ARG)=QABS(ARG)
      HUGE=1.0E35
      ZERO=0.0E0
      UNITY=1.0E0
C
      WRITE(6,44)(S(J),J=1,NMAX)
   44 FORMAT(/1X,'S = ',4D28.17/(5X,4D28.17))
      IF (NMIN.GT.NMAX)GO TO 130
      EXTRP=S(NMAX)
      KT=2*KMAX
      DO 120 J=NMIN,NMAX
         TZ=0.
         LIMKK=J-NMIN
         IF (LIMKK.GT.KT)LIMKK=KT
         IF (LIMKK.LE.0) GO TO 120
```

```
         DO 110 KK=1,LIMKK
            JMKK=J-KK
            IF (QTEST(S(JMKK)).EQ.HUGE)GO TO 80
            IF (QTEST(S(JMKK+1)).EQ.HUGE)GO TO 80
            TU=S(JMKK+1)-S(JMKK)
            IF (QTEST(TU).EQ.ZERO)GO TO 90
            TU=TZ+UNITY/TU
            GO TO 100
C
  80        TU=TZ
            GO TO 100
  90        TU=HUGE
  100       TZ=S(JMKK)
  110       S(JMKK)=TU
C           H(JMKK)=TZ
  120     CONTINUE
C
          MXMKT=NMAX-KT
          EXTRP=S(MXMKT)
          WRITE(6,21)S(MXMKT)
   21     FORMAT(/1X,'S(MXMKT)=',D28.17)
  130     RETURN
          END
```

\

VITA

HUI-WEN CHIANG

Candidate for the Degree of

Master of Science

Thesis: A STUDY OF THE EPSILON ALGORITHM AND APPLICATIONS

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Taipei, Taiwan, Republic of China, July 14, 1956, the daughter of Hou-Shen Chiang and Chen-Kan Chiang.

Education: Graduate from Taipei First Girls' Senior High School, July 1974; received Bachelor of Law from Fu Jen University, Taipei, Taiwan, Republic of China, in June 1978; completed requirements for the Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, in December, 1986.

Professional Experience: Lawyer Assistant, Formosa Commercial Law Firm, September, 1978 to July, 1981; Programmer/System Analyst, System Design and Computer Services, OAED, Oklahoma State University, March, 1983 to November, 1986.