AN ALGORITHM FOR A GENERALIZATION OF THE
RICHARDSON EXTRAPOLATION PROCESS

by

William F. Ford* and Avram Sidi**

Technical Report #347
November 1984

* Computer Services Division, NASA Lewis Research Center, Cleveland, Ohio, 44135, USA.

** Computer Science Department, Technion - Israel Institute of Technology, Haifa 32000, Israel
ABSTRACT

In this paper we present a recursive method, designated the W(m)-algorithm, for implementing a generalization of the Richardson extrapolation process that has been introduced in [8]. Compared to the direct solution of the linear systems of equations defining the extrapolation procedure, this method requires a small number of arithmetic operations and very little storage. The technique is also applied to solve recursively the coefficient problem associated with the rational approximations obtained by applying the d-transformation of [6,13] to power series. In the course of development a new recursive algorithm for implementing a very general extrapolation procedure is introduced, which is similar to that given in [2,4] for solving the same problem. A FORTRAN program for the W(m)-algorithm is also appended.
1. INTRODUCTION

In a recent paper [8] a generalization of the well known Richardson extrapolation process has been introduced. This generalization - called GREP for short - has proved to be very useful in accelerating the convergence of a large class of infinite sequences. GREP is defined, and so far has been implemented, through the solution of systems of linear equations. As has been mentioned in [8], the matrices of these linear systems become large in dimension if one wishes to increase the accuracy in extrapolation. Thus implementing GREP through the direct numerical solution of the linear equations may become costly. Therefore, it is desirable to have efficient algorithms for implementing GREP. Since the linear equations defining GREP have a very special structure, it seems that any algorithm, in order to be efficient, has to take advantage of this structure. In fact, a very efficient algorithm for a special case of GREP has been given in [12], and has been denoted the W-algorithm.

The purpose of the present work is to present an algorithm for what is probably the most common form of GREP.

Specifically, we develop an algorithm to be designated the $W(m)$-algorithm, for obtaining $A_n^{(m,j)}$, which, with a slight change in the notation of [8], is defined by the linear equations

$$A_n^{(m,j)} = A(y_{\ell}) + \sum_{k=1}^{m} \phi_k(y_{\ell}) \sum_{i=0}^{n_k} \bar{\delta}_{ki} y_{\ell}^i, \quad j \leq \ell \leq j+N, \quad (1.1)$$

where $n$ denotes $(n_1, n_2, \ldots, n_m)$, $N = \sum_k (n_k+1)$, the $y_{\ell}$, $A(y_{\ell})$, and $\phi_k(y_{\ell})$ are given, $r$ is a known positive constant, and $A_n^{(m,j)}$ and the $\bar{\delta}_{ki}$ are the unknowns. The equations in (1.1) differ from those
of [8], given for the most general form of GREP, in that the linear equations defining the latter are obtained from (1.1) by replacing \( r \) by \( 1_k \).

The above mentioned \( W \)-algorithm [12] is a recursive method for implementing GREP when \( m = 1 \) in (1.1). The \( W^{(m)} \)-algorithm of the present work is designed so that it reduces to the \( W \)-algorithm for \( m = 1 \). This is the justification for the designation \( W^{(m)} \).

Below we characterize the sequences whose convergence can hopefully be accelerated by the form of GREP given in (1.1).

**Definition:** We shall say that a function \( A(y) \), defined for \( 0 < y \leq b \), for some \( b > 0 \), where \( y \) can be a discrete or continuous variable, belongs to the set \( F^{(m)} \), if there exist functions \( \phi_k(y) \), \( \beta_k(y) \), \( k = 1, \ldots, m \), and a constant \( A \) such that

\[
A = A(y) + \sum_{k=1}^{m} \phi_k(y) \beta_k(y),
\]

where \( A = \lim_{y \to 0+} A(y) \) whenever this limit exists, and the functions \( \beta_k(y) \), as functions of the continuous variable \( \xi \), are continuous for \( 0 \leq \xi \leq b \), and for some constant \( r > 0 \), have the asymptotic expansions

\[
\beta_k(\xi) \sim \sum_{i=0}^{\infty} \beta_{ki} \xi^{i r} \quad \text{as} \quad \xi \to 0+.
\]

If, in addition, the functions \( B_k(t) \equiv \beta_k(t^{1/r}) \) are infinitely differentiable for \( 0 \leq t \leq b^r \), we shall say that \( A(y) \) belongs to the set \( F^{(m)}_{\infty} \).

When \( \lim_{y \to 0+} A(y) \) does not exist, \( A \) is called the antilimit of \( A(y) \).

The problem is to extrapolate \( A(y) \) to \( y = 0 \) and obtain (or approximate)
A whether \( A \) is the limit or antilimit of \( A(y) \), and this is precisely what we are trying to accomplish through the equations in (1.1) that define \( \text{GREP} \). \( A_n^{(m,j)} \) in (1.1) is taken to be an approximation to \( A \).

Note that the equations in (1.1) are obtained by formally substituting (1.3) in (1.2), truncating the asymptotic expansions for \( \beta_k(y) \) at the terms \( \beta_k y^{\infty} \), replacing \( A \) and the \( \beta_k \) by \( A_n^{(m,j)} \) and \( \tilde{\beta}_k \) respectively, and finally collocating at the points \( y_j, y_{j+1}, \ldots, y_{j+N} \).

Here the \( y_j \) are chosen in such a way that \( b > y_0 > y_1 > \ldots > 0 \), and \( \lim_{\ell \to \infty} y_\ell = 0 \), although this ordering has no effect on the derivation of the \( W^{(m)} \)-algorithm.

Several examples of functions \( A(y) \) in \( F^{(m)} \) are given in [8, Section 2]. Some examples of \( \text{GREP} \) are the \( D \)-transformation of [6] for infinite integrals, the \( \bar{D} \)- and \( \ddot{D} \)-transformations of [10] for oscillatory infinite integrals, the \( W \)-transformation of [11] for very oscillatory infinite integrals, and the \( d \)-transformation of [6] and the \( T \)-transformation of [5] for infinite series. For more information on these methods and their convergence properties see [5 - 13].

In Sections 2-4 we develop the \( W^{(m)} \)-algorithm by which the \( A_n^{(m,j)} \) can be computed recursively without solving numerically the equations in (1.1). This algorithm requires very little storage and computing time.

In Section 5 we shall consider the case in which \( \phi_k(y_\ell) = v_k(\ell)z^\ell \), where \( v_k(\ell) \) are independent of \( z \). This arises, for example, when one applies a modified version of the \( d \)-transformation of [13] to power series in \( z \). The approximations \( A_n^{(m,j)} \) turn out to be rational functions in \( z \). Here, using the techniques of the \( W^{(m)} \)-algorithm
we develop a recursive method for computing the coefficients of the numerator and denominator polynomials of $A_n^{(m,j)}$.

As part of our development, in Section 2 we also devise an algorithm for solving a very general extrapolation problem. This algorithm, although similar in form to the E-algorithm of [2,4], is different than the latter, and requires a smaller number of arithmetic operations. These and other details concerning this algorithm are given in Appendix A. A FORTRAN program for the $W^{(m)}$-algorithm is included in Appendix B.

Before closing this section we shall rewrite the equations in (1.1) in a more convenient form as follows: Let $t = y^R$ and $t_k = y_k^R$, $k = 0, 1, \ldots$, and define $a(t) = A(y)$ and $q_k(t) = \phi_k(y)$, $k = 1, \ldots, m$. Then (1.1) becomes

$$A_n^{(m,j)} = a(t^R) + \sum_{k=1}^{m} q_k(t^R) \sum_{i=0}^{n_k} \beta_i^R t_i^{k^2} t^R, \quad j \leq \ell \leq j+N. \quad (1.4)$$

In the remainder of this work we shall work with these equations.
2. THEORETICAL PRELIMINARIES

Consider the equations in (1.4). Let us denote the \( \varphi_k(t) t_k^i \) and \( \beta_{ki} \), \( 0 \leq i \leq n_k, 1 \leq k \leq m \), (ordered in a manner to be described later) by \( g_u(t) \) and \( \alpha_{u} \), \( 1 \leq u \leq N \), respectively. We note that the exact nature of this ordering is very crucial in the development of the \( \mathcal{W}^{(m)} \)-algorithm. For present purposes, however, this ordering is irrelevant, and in fact, throughout most of this section, the \( g_u(t) \) can be treated as arbitrary. Let us also denote \( A_{n}^{(m,j)} \) by \( A_{N}^{j} \), and \( a(t_k) \) by \( a(t) \). Then equations (1.4) can be expressed as

\[
A_{N}^{j} = a(t) + \sum_{i=1}^{N} a_{i} g_{i}(t), \quad j \leq k \leq j+N. \tag{2.1}
\]

By Cramer's rule \( A_{N}^{j} \) can be expressed as the ratio of two determinants, namely as

\[
A_{N}^{j} = \frac{f_{N}^{j}(a)}{f_{N}^{j}(I)} = \frac{|g_{1}(j) \ldots g_{N}(j) a(j)|}{|g_{1}(j) \ldots g_{N}(j) I(j)|}, \tag{2.2}
\]

where \( I(t) = 1, k \geq 0 \), and \(|u_{1}(j) \ldots u_{p}(j)|\) is the \( p \times p \) determinant

\[
|u_{1}(j) \ldots u_{p}(j)| = \begin{vmatrix}
    u_{1}(j) & \cdots & u_{p}(j) \\
    \vdots & \ddots & \vdots \\
    u_{1}(j+p-1) & \cdots & u_{p}(j+p-1)
\end{vmatrix}. \tag{2.3}
\]

Let us now denote

\[
G_{p}^{j} = |g_{1}(j) \ldots g_{p}(j)|, \quad G_{0}^{j} = 1, \tag{2.4}
\]

and, for arbitrary \( b(t) \), \( k \geq 0 \), define

\[
\psi_{p}^{j}(b) = f_{p}^{j}(b)/G_{p+1}^{j}. \tag{2.5}
\]
Then (2.2) may be reexpressed as

\[ A_N^j = \frac{\psi_N^j(a)}{\psi_N^j(1)}. \]  

(2.6)

Because \( f_p^j(b) \) differs from \( G_{p+1}^j \) by only one column in the determinant arry, it is natural to seek a relation between these quantities. This is accomplished by means of an identity known as Sylvester's Theorem, a proof of which can be found in [1, p.23].

**Theorem 1:** Let \( C \) be a matrix, and let \( C_{\rho\sigma} \) denote the matrix obtained by deleting row \( \rho \) and column \( \sigma \) of \( C \). Also let \( C_{\rho\sigma'} \) denote the matrix obtained by deleting rows \( \rho \) and \( \rho' \) and columns \( \sigma \) and \( \sigma' \) of \( C \). Provided \( \rho < \rho' \) and \( \sigma < \sigma' \)

\[ \det C \det C_{\rho\sigma} \det C_{\rho',\sigma'} = \det C_{\rho\sigma} \det C_{\rho',\sigma}, - \det C_{\rho\sigma}, \det C_{\rho',\sigma'} \]  

(2.7)

If \( C \) is a \( 2 \times 2 \) matrix, then (2.7) holds with \( \det C_{\rho\sigma} = 1 \).

Applying Theorem 1 to the \( (p+1) \times (p+1) \) determinant

\[ f_p^j(b) = \left| g_1(j), \ldots, g_p(j), b(j) \right| \]  

with \( \rho = 1, \sigma = p, \rho' = \sigma' = p+1, \) and using (2.4), we obtain

\[ f_p^j(b) G_{p+1}^{j+1} = f_{p-1}^j(b) G_p^j - f_p^j(b) G_{p-1}^{j+1} \]  

(2.8)

This is the desired relation.

Upon invoking (2.5), and letting

\[ D_p^j = \frac{G_{p+1}^j G_{p-1}^j}{G_p^j G_{p-1}^j}, \]  

(2.9)

(2.8) becomes

\[ \psi_p^j(b) = \left( \psi_{p-1}^{j+1}(b) - \psi_{p-1}^j(b) \right) / D_p^j. \]  

(2.10)
From (2.6) and (2.10), we see that once the $D^j_p$ are known, the
$\psi^j_p(a)$ and $\psi^j_p(I)$ and hence $A^j_p$ can be computed recursively. Therefore, we aim at developing an efficient algorithm for the determination of the $D^j_p$. In the absence of detailed knowledge concerning $g_i(\xi)$, we could proceed by observing that for $b(\xi) = g_{p+1}(\xi)$, (2.5) reduces to

$$\psi^j_p(g_{p+1}) = 1.$$  \tag{2.11}

Consequently (2.10) becomes

$$D^j_p = \psi^{j+1}_p(g_{p+1}) - \psi^j_p(g_{p+1}),$$  \tag{2.12}

which permits a recursive evaluation of $D^j_p$ through the quantities $\psi^j_p(g_i)$, $i \geq p+1$. (Note that $\psi^j_p(g_i) = 0$ for $1 \leq i \leq p$.) Thus (2.10) and (2.12) provide us with a recursive algorithm for solving the general extrapolation problem in (2.1). This algorithm, although similar in form to the E-algorithm of [2,4], is different from it and requires a smaller number of arithmetic operations. For the actual implementation of, and more details on, this new algorithm see Appendix A.

In the present case of GREP, however, the $g_i(\xi)$ have a known structure which may be profitably employed to increase the computational efficiency and reduce the storage requirements. Consider the case of $m = 2$ in (1.2) and (1.4) and suppose we would like to compute $A^j_p = A^{(2,j)}_{n}$ for $n = (0,0), (1,0), (1,1), (2,1), (2,2)$, etc. For this we set

$$g_i(\xi) = \varphi_i(\tau_\xi), \quad i = 1,2, \quad g_i(\xi) = \tau_\xi g_{i-2}(\xi), \quad i = 3,4,\ldots$$

Therefore, for $p \geq 3$

$$d^j_p = |\varphi_1(t_j) \varphi_2(t_j) t_j \varphi_1(t_j) t_j \varphi_2(t_j) \cdots g_p(j)|.$$
If we factor out $t_{j+i-1}$ from the $i$-th row, $i = 1, 2, \ldots, p$, we obtain

$$G_j^p = \left(\prod_{i=1}^{p} t_{j+i-1}\right) | h_1^{j}(j) h_2^{j}(j) s_1^{j}(h) \ldots s_{p-2}^{j}(j) |,$$

where

$$h_i^{j}(h) = \varphi_i^{j}(t)/t_i , \quad i = 1, 2 .$$

The determinant obtained above differs from $G_j^p$ by two columns, and it is not hard to see that there will be $m$ different columns in the general case. We need therefore to develop procedures for evaluating these objects.

We start this by introducing the following generalizations of the $f_j^p(b), \psi_j^p(b)$, and $D_j^p$:

$$F_j^p(h_1, \ldots, h_q) = | s_1^{j}(j) \ldots s_p^{j}(j) h_1^{j}(j) \ldots h_q^{j}(j) | \equiv F_j^p(q) , \quad (2.13)$$

$$\psi_j^p(h_1, \ldots, h_q) = \frac{F_j^{p+1}(q)}{F_j^{p+2}(q-1)} \equiv \psi_j^p(q) , \quad (2.14)$$

$$D_j^p(q) = \frac{F_j^{p+1}(q)}{F_j^{p+1}(q)} , \quad (2.15)$$

the quantities $h_i^{j}(h)$ being arbitrary so far.

As simple consequences of (2.13)-(2.15) we obtain:

$$F_j^p(0) = G_j^p , \quad F_j^p(1) = f_j^p(h_1) , \quad (2.16)$$

$$\psi_j^p(1) = \psi_j^p(h_1) , \quad (2.17)$$

and

$$D_j^p(0) = p_j^p , \quad (2.18)$$

respectively. In addition, we define $F_0^j(0) = 1$. 
We also note that since $F_p^j(q)$ is defined for $p \geq 0$ and $q \geq 0$, $F_p^j(q)$ is defined for $1 \leq q \leq p+1$, and $D_p^j(q)$ is defined for $0 \leq q \leq p-1$.

It may be interesting to note that $F_p^j(q)$ along with the parameters $a_i^j$, $1 \leq i \leq p$, satisfies the set of linear equations

$$(-1)^{q-1} \frac{h_q(\lambda)}{h_q(\lambda)} = \sum_{i=1}^{p-q+1} a_i \delta_i(\lambda) + g_{p-q+2}(\lambda) \delta^j_p(q) + \sum_{i=1}^{q-1} a_{p-q+i+1} h_i(\lambda),$$

where the summation $\sum_{i=1}^{0}$ is taken to be zero for $q = 1$ and $q = p+1$.

The following results will be of use in the development of our algorithm further on.

**Theorem 2:** The $F_p^j(q)$ and $D_p^j(q)$ satisfy

$$F_p^j(q) = \frac{\delta^j_p(q) - \delta^j_{p-1}(q)}{D_p^j(q)}, \quad 1 \leq q \leq p, \quad (2.19)$$

and

$$D_p^j(q) = \frac{\gamma^j_{p-2}(q)}{\gamma^j_{p-1}(q)} + \frac{1}{\gamma^j_{p-1}(q)} \gamma^j_{p-1}(q), \quad 1 \leq q \leq p-1. \quad (2.20)$$

**Proof:** Consider the $(p+q) \times (p+q)$ determinant $F_p^j(q)$. Applying Theorem 1 to this determinant with $\rho = 1$, $\rho' = p+q$, $\sigma = p$, and $\sigma' = p+q$, we obtain

$$F_p^j(q) F_{p-1}^{j+1}(q) = F_p^j(q) F_{p-1}^j(q-1) - F_{p-1}^{j+1}(q-1) F_{p-1}^j(q). \quad (2.21)$$

Replacing $p$ in (2.21) by $p+1-q$, and using (2.14) and (2.15), (2.19) follows. In order to prove (2.20) we start with (2.15). Invoking (2.14), (2.15) becomes

$$D_p^j(q) = \frac{\gamma^j_p(q) \gamma^j_{p-2}(q)}{\gamma^j_{p-1}(q) \gamma^j_{p-1}(q)} D_p^j(q-1). \quad (2.22)$$

Substituting now (2.19) in (2.22), (2.20) follows.
Now $\psi^j_p(p+1)$ cannot be computed using the recursion relation in (2.19) since neither $\psi^j_{p-1}(p+1)$ nor $D^j_p(p)$ is defined. However, it is possible to derive a recursion relation among the $\psi^j_p(p+1)$, and this is given in Theorem 3 below.

**Theorem 3:** Let

$$\hat{\psi}^j_p(b) = \frac{|h_1(j)\ldots h_p(j) b(j)|}{|h_1(j)\ldots h_{p+1}(j)|}$$

(2.23)

Then, for arbitrary $b(\xi)$,

$$\hat{\psi}^j_p(b) = [\hat{\psi}^{j+1}_{p-1}(b) - \hat{\psi}^j_{p-1}(b)]/\hat{D}^j_p$$

(2.24)

where

$$\hat{D}^j_p = \hat{\psi}^{j+1}_{p-1}(h_{p+1}) - \hat{\psi}^j_{p-1}(h_{p+1})$$

(2.25)

and

$$\psi^j_p(p+1) = (-1)^p/\hat{\psi}^j_p(g_1)$$

(2.26)

**Proof:** First (2.26) follows directly from the fact that $\psi^j_p(p+1) = F^j_o(p+1)/F^j_1(p)$. Next, we note that the definition of $\hat{\psi}^j_p(b)$ as given in (2.23) is very similar to that of $\psi^j_p(b)$ as given in (2.5), the only difference being that $g_i(\xi)$ in (2.5) has been replaced by $h_i(\xi)$ in (2.23). This immediately, suggests that (2.10)-(2.12) hold with $\psi^j_p(b)$, $D^j_p$, and $g_i(\xi)$, $i = 1,2,\ldots$, replaced by $\hat{\psi}^j_p(b)$, $\hat{D}^j_p$, and $h_i(\xi)$, $i = 1,2,\ldots$, respectively.

This proves (2.24) and (2.25).

What we see from Theorem 3 is that the computation of the $\psi^j_p(p+1)$ can be achieved through that of the $\hat{\psi}^j_p(g_1)$, which in turn can be achieved by using the algorithm given in Appendix A. As we
shall see in the next section, the \( h_1(t) \) in GREP have no particular structure in general, thus the actual computation of the \( \psi_j^p(p+1) \) in the \( W^{(m)} \)-algorithm is done as described above.

In addition to the relationships in the previous two theorems, we give one more result concerning the \( \psi_j^p(q) \).

**Theorem 4:** The \( \psi_j^p(q) \) satisfy the relation

\[
\psi_j^{p+1-q}(q) = \psi_j^p(1) \psi_j^p(2) \ldots \psi_j^p(q) g_j^{p+1} .
\]  

**Proof:** Let us reexpress (2.14) in the form

\[
\psi_j^{p+1-q}(q) = \psi_j^p(q) \psi_j^{p+1-(q-1)} (q-1). 
\]  

Invoking (2.14), with \( q \) replaced by \( q-1 \), on the right hand side of (2.28), and continuing, we obtain

\[
\psi_j^{p+1-q}(q) = \psi_j^p(q) \ldots \psi_j^p(1) F_j^{p+1}(0).
\]  

(2.27) follows from (2.29) and (2.16). \( \square \)
3. ORDERING OF $\varphi_k(t_k) t_k^i$ AND ITS CONSEQUENCES

Numerical experience and theoretical results suggest that as the $n_k$ in (1.4) become large simultaneously, usually $A_n^{(m,j)} \rightarrow A$ quickly where $A$ is the limit or antilimit of the sequence $a(k)$, $k = 0, 1, 2, \ldots$. The simplest way of achieving this is by letting $n_k = s$, $1 \leq k \leq m$, and increasing $s$. More generally, we can let $n_k = v_k + s$ for some fixed integers $v_k \geq 0$, $1 \leq k \leq m$, and $s = 0, 1, 2, \ldots$. With the $\varphi_k(t_k) t_k^i$, $1 \leq k \leq m$, $i \geq 0$, ordered in an appropriate manner, the $W^{(m)}$-algorithm is actually designed to compute a sequence of $A_n^{(m,j)}$ that includes those $A_n^{(m,j)}$ for which the $n_k$ are as above. We now describe this ordering.

Without loss of generality we may assume $v_1 \geq v_2 \geq \ldots \geq v_m$.

We define the indices $i_k$ by

$$i_{m+1} = \infty, \quad i_1 = 1, \quad i_{k+1} = i_k + 1 + k(v_k - v_{k+1}), \quad 1 \leq k \leq m-1. \quad (3.1)$$

Note that $i_{k+1} \geq i_k + 1$ and thus $i_k \geq k$, $1 \leq k \leq m$. Note also that

$$c_k = (i_{k+1} - i_k - 1)/k, \quad 1 \leq k \leq m, \quad (3.2)$$

are non-negative integers for $1 \leq k \leq m-1$, and $c_m = \infty$.

Let us define

$$c_{k_2}^{(k_1)} = \sum_{j=k_1}^{k_2} c_j, \quad 1 \leq k_1 \leq k_2 \leq m; \quad c_{k_2}^{(k_1)} = 0 \quad \text{otherwise}. \quad (3.3)$$

For each ordered pair of integers $(k,i)$, $1 \leq k \leq m$, $i \geq 0$, we define in one-to-one correspondence, a single integer $\mu \geq 1$ by
\[
\mu = \begin{cases} 
    i_k & \text{if } i = 0 \\
    i_q - (q-k) + (i-c_{q}^{q-1})q & \text{if } i \geq 1 \text{ and } c_{k}^{q-1} < i \leq c_{q}^{q}.
\end{cases}
\]

Finally, we set

\[
\varphi_k(t_k) t_k^i = g_\mu(\xi),
\]

thus determining the desired ordering. We note that the condition

\[c_{k}^{q-1} < i \leq c_{q}^{q}\]

implies that \(i_q < \mu < i_{q+1}\).

In order to illustrate the exact meaning of this ordering let us analyze the following example.

**Example 1.** \(m = 5, v_1 = 7, v_2 = 5, v_3 = 4 = v_4, v_5 = 2\). By (3.1) \(i_1 = 1, i_2 = 4, i_3 = 7, i_4 = 8, i_5 = 17\), and by (3.2) \(c_1 = 2, c_2 = 1, c_3 = 0, c_4 = 2\). In the diagram below, starting from the left, the square in the \(k\)-th row denotes \(\varphi_k(t_k)\), and the \(i\)-th circle in the same row denotes \(\varphi_k(t_k) t_k^i\). The integer below each square or circle is \(\mu \in g_\mu(\xi) = \varphi_k(t_k) t_k^i\). Note that the \(m\) rows have been arranged so that the terms \(\varphi_k(t_k) t_k^i\) are in the same column, and that \(c_k\) denotes the number of columns that contain those \(\mu\)’s satisfying \(i_k < \mu < i_{k+1}\).
With the ordering above the sequence of approximations $A_{n(s)}^{(m,j)}$, where $n(s) = (v_1 + s, \ldots, v_m + s)$, $s = 0, 1, 2, \ldots$, is actually a subsequence of the sequence $A_j^p$, $p = 0, 1, 2, \ldots$, and $A_{n(s)}^{(m,j)} = A_j^{i_0 + ms}$, where $N_0 = \sum_{k=1}^{m} (v_k + 1)$.

As a consequence of this ordering we have

$$g_i(\xi)/t_\xi = \begin{cases} h_k(\xi), & i = i_k, \\ g_{i-k}(\xi), & i_k < i < i_{k+1} \end{cases}, \quad 1 \leq k \leq m, \quad (3.6)$$

where we have defined

$$h_k(\xi) = \varphi_k(t_\xi)/t_\xi = g_{i-k}(\xi)/t_\xi, \quad 1 \leq k \leq m. \quad (3.7)$$

Conversely, (3.6) and (3.7) define the ordering of (3.4) and (3.5).

We note that (3.6) and (3.7) are the key to the development of the $w^{(m)}$-algorithm.

Example 2: An important special case of the construction above is that for which $v_k = 0$, $1 \leq k \leq m$, which, we shall henceforth refer
to as the "normal ordering". For this case we have

\[ i_k = k, \quad 1 \leq k \leq m; \quad c_k = 0, \quad 1 \leq k \leq m-1, \quad (3.8) \]

\[ \varphi_k(t_\lambda^i t_\lambda^{-i}) = g_{k+i-\lambda}(\lambda), \quad 1 \leq k \leq m, \quad i \geq 0, \quad (3.9) \]

\[ g_i(\lambda) = \begin{cases} \varphi_i(t_\lambda), & 1 \leq i \leq m \\ t_\lambda g_{i-m}(\lambda), & i > m \end{cases} \quad (3.10) \]

and

\[ h_k(\lambda) = \varphi_k(t_\lambda^i)/t_\lambda^i = g_k(\lambda)/t_\lambda^i, \quad 1 \leq k \leq m. \quad (3.11) \]

The FORTRAN program given in Appendix B to this work deals with the normal ordering.

Since we have defined \( h_k(\lambda) \) only for \( 1 \leq k \leq m \), we see that \( F^j_p(q) \) is defined only for \( 0 \leq q \leq m, \quad p \geq 0, \quad \psi^j_p(q) \) for \( 1 \leq q \leq \min(p+1,m) \), and \( D^j_p(q) \) for \( 0 \leq q \leq \min(p-1,m) \). Consequently, the recursion relation in (2.19) is valid for \( 1 \leq q \leq \min(p,m) \), and that in (2.20) for \( 1 \leq q \leq \min(p-1,m) \).

Through \( g_i(\lambda) = t_\lambda g_{i-m}(\lambda), \quad i > i_m \), cf. (3.6), the ordering above enables us to relate \( F^j_p(m) \) to \( G^j_p \) and hence \( D^j_p(m) \) to \( D^j_p(0) = D^j_p \), the desired quantity, thus reducing the amount of computation for the \( W^{(m)} \)-algorithm considerably. More generally we have the following results.

**Lemma 5**: Let

\[ \sigma_k = \sum_{s=1}^{k} (i_s - s), \quad 1 \leq k \leq m, \quad (3.12) \]

and

\[ \pi^j_p = \prod_{i=j}^{j+p-1} t_i, \quad p \geq 1. \quad (3.13) \]
Then for \( i_k < p < i_{k+1} \),

\[
G_p^j = (-1)^{k+p} \pi_{p-k}^j (k) . \tag{3.14}
\]

**Proof:** Starting with the determinant representation of \( G_p^j \) in (2.4), and dividing the \( i \)-th row by \( t_{j+i-1} \), \( i = 1, \ldots, p \), and invoking (3.6) and (3.13), we obtain

\[
G_p^j = \pi_p^j \left| h_1 g_1 \ldots g_{i_2-2} h_2 g_{i_2-2} \ldots g_{i_3-3} h_3 g_{i_3-2} \ldots g_{i_4-4} h_4 \ldots g_{i_{k-2}} h_k g_{i_{k-1}} \ldots g_{p-k} \right| , \tag{3.15}
\]

where we have omitted the index \( j \) for simplicity of notation.

Note that when \( p = i_k \), \( h_k \) is the last column in the determinant above. Let us now move \( h_2 \) to the right of \( h_1 \), \( h_3 \) to the right of \( h_2 \), etc. The number of column interchanges required for this is \( \sigma_k \).

Thus

\[
G_p^j = (-1)^{k+p} \pi_p^j \left| h_1(j) \ldots h_k(j) g_1(j) \ldots g_{p-k}(j) \right| . \tag{3.16}
\]

(3.14) now follows from (3.16) and (2.13).

**Theorem 6:** For \( i_k < p < i_{k+1} \) and \( k < i_k = p \cdot p = i_{k+1} - 1 \), \( D_p^j \) can be computed from

\[
D_p^j = D_p^j(0) = D_p^j(k) \times \begin{cases} \psi_{p+1}^j(k) & k < i_k = p \cdot p = i_{k+1} - 1 \\ \psi_{p-2}^j(k) & k < i_k = p \cdot i_{k+1} - 1 \\ 1 & i_k < p < i_{k+1} - 1 \\ (-1)^{i_{k+1}} & i_k < p = i_{k+1} - 1 . \end{cases} \tag{3.17}
\]

**Proof:** By (2.9) \( D_p^j \) is a combination of \( G_p^j, G_p^{j+1}, G_p^j \), and \( G_p^j \). For \( i_k < p < i_{k+1} - 1 \), we also have \( i_k < p = i_{k+1} - 1 \).

Consequently, we can make direct use of Lemma 5 on all four \( G_p^j \), above.
Invoking now (2.15), we obtain \( D^j_p = D^j_p(k) \). When \( i_k < p = i_{k+1} - 1 \), we have \( i_k < p-1 < i_{k+1} \) and \( i_k < p < i_{k+1} \), but \( p+1 = i_{k+1} \). Thus, we can make direct use of Lemma 5 on \( G^j_{p-1} \), \( G^j_p \), and \( G^j_{p+1} \) only. We apply Lemma 5 to \( G^j_{p+1} \) with \( k \) in (3.14) replaced by \( k+1 \). The overall result of this is

\[
D^j_p = (-1)^k \frac{p^j_{p+1-(k+1)}(k+1)p^j_{p-1-k}(k)}{p^j_{p-k}(k)p^j_{p-k}(k)}.
\]

(3.18)

Multiplying both the numerator and the denominator of (3.18) by \( F^j_{p+1-k}(k) \), and invoking (2.14) and (2.15), we obtain

\[
D^j_p = (-1)^k p^j_p(k) F^j_p(k+1). \]

The proof of the rest of (2.17) is accomplished similarly.

\[ \square \]

Note that for \( k = m \) (3.17) reads

\[
D^j_p = D^j_p(m), \quad p > i_m,
\]

(3.19)

and it is this property that makes the \( W^{(m)} \)-algorithm economical.

Also with the help of Theorem 2, (3.17) can be reexpressed as

\[
D^j_p = D^j_p(0) = \begin{cases} 
[y^j_{p-1}(k+1) - y^j_{p-1}(k+1)] / y^j_{p-2}(k) & \text{if } i_k = p = i_{k+1} - 1 \\
(-1)^k [y^j_{p-1}(k) - y^j_{p-1}(k)] & \text{if } i_k = p < i_{k+1} - 1 \\
D^j_p(k) & \text{if } i_k < p < i_{k+1} - 1 \\
(-1)^k [y^j_{p-1}(k+1) - y^j_{p-1}(k+1)] & \text{if } i_k = p = i_{k+1} - 1.
\end{cases}
\]

(3.17)

The only case not covered by Theorem 6 is that of \( p = i_k = k \), and \( D^j_p \) for this case can be computed by using (2.10) and (2.12). If we let \( k_1 \) be that integer for which \( i_k = k \) when \( k \leq k_1 \) and \( i_k > k \) when \( k > k_1 \), then we can use (2.10) and (2.12) for \( 1 \leq p \leq k_1 \),
and Theorem 6 for \( k > k_1 \): In the \( W^{(m)} \)-algorithm given in the next section, however, we choose to compute \( D_i^j \) for \( p = i_k, 1 \leq k \leq m \), using (2.10) and (2.12), thus making the programming simpler.

**Theorem 7:** When \( i_k - 1 < p < i_{k+1} - 2 \)

\[
(-1)^{\pi_{p+1}} \prod_{i=1}^{k} \psi_i^j(i) = 1, \quad (3.20)
\]

**Proof:** Let us put \( q = k \) in (2.27). The result now follows by applying Lemma 5 to the resulting identity.

**APPLICATIONS OF THEOREMS 6 AND 7 TO THE CASES \( m = 1, 2 \).**

**The case \( m = 1 \).** From (3.19) \( D_i^j = D_i^j(1) \) for \( p \geq 2 \). Similarly from (3.20) \( (-1)^{\pi_{p+1}} \prod_{i=1}^{k} \psi_i^j(1) = 1 \) for \( p \geq 0 \). Thus solving for \( \psi_i^j(1) \), and substituting in (2.20) with \( q = 1 \) there, we obtain

\[
\frac{D_i^j}{p} = t_{j+p} - t_j, \quad (3.21)
\]

which is valid for \( p \geq 2 \). Using (2.9), (3.21) is seen to be valid also for \( p = 1 \). Thus we have recovered the \( W \)-algorithm of [12].

**The case \( m = 2 \).** For simplicity we shall consider the normal ordering, for which \( i_1 = 1, i_2 = 2 \). Thus from (3.19) \( D_i^j = D_i^j(2) \) for \( p \geq 3 \). Similarly from (3.20) \( \prod_{p+1}^{p} \psi_i^j(1) \psi_i^j(2) = 1 \) for \( p \geq 1 \).

Solving for \( \psi_i^j(2) \) in terms of \( \psi_i^j(1) = \psi_i^j(h_1) \), and substituting in (2.20) with \( q = 2 \) there, we obtain

\[
D_i^j = \left[ t_j \psi_i^{j+1}(h_1) - t_{j+p} \psi_i^{j+1}(h_1) / \psi_i^{j+1}(h_2) \right], \quad (3.22)
\]

which is valid for \( p \geq 3 \). Using (2.9), (3.22) can be shown to be valid also for \( p = 2 \). As for \( p = 1 \), we use

\[
D_i^j = \frac{g_2(j+1)}{g_1(j+1)} \cdot \frac{g_2(j)}{g_1(j)} = \frac{\varphi_2(t_{j+1})}{\varphi_1(t_{j+1})} - \frac{\varphi_2(t_j)}{\varphi_1(t_j)}, \quad (3.23)
\]
which follows from (2.9). Thus we have actually developed the $W^{(2)}$-algorithm for the normal ordering. The order of computation in this algorithm is as follows: Given $\varphi_1(t_2^k), \varphi_2(t_2^k), k \geq 0$, use (3.23) and (2.10) to obtain the $\psi_j^1(h_1)$. Then for $p = 2, 3, \ldots$, use (3.22) to obtain the $D_p^j$, and then (2.10) to obtain the $\psi_j^1(h_1)$.

Note that (3.22) is also valid for the general case in which $i_2 > 2$ and $p \geq i_2 + 1$, and it can be used for computing $D_p^j$ without having to compute and store $\psi_j^j(2)$ in the algorithm given in the next section.

In Theorem 3 we obtained a recursion relation among the $\psi_p^j(b)$, which, through (2.26) enables us to determine $\psi_p^j(p+1)$. Since we have defined $h_k^p(\xi)$ only for $1 \leq k \leq m$, the $\psi_p^j(p+1)$ are defined and needed only for $0 \leq p \leq m-1$. In general the $\psi_p^j(p+1)$, $0 \leq p \leq m-1$, and the $D_p^j$, $p = i_k$, $1 \leq k \leq m$, are computed by different computational procedures. For the case of normal ordering, however, they can be obtained simultaneously using the same computational procedure, namely the algorithm given in Appendix A, as suggested by the following theorem.

**Theorem 8:** For normal ordering, i.e., $i_k = k$, $1 \leq k \leq m$, we have

$$\psi_p^j(p+1) = (-1)^p / \psi_p^j(g_{m+1}), \quad 0 \leq p \leq m-1 \tag{3.24}$$

and

$$\prod_{i=1}^{p+1} \psi_p^j(i) = 1, \quad 0 \leq p \leq m-1. \tag{3.25}$$

**Proof:** (3.24) follows from Theorem 3, and (3.25) from Theorem 7. 

□
4. THE \( W^{(m)} \)-ALGORITHM

We now describe the \( W^{(m)} \)-algorithm that will enable us to compute the \( A^j_p \) efficiently when the \( \varphi_k (t_\ell) t_\ell^i \) are ordered as in the beginning of Section 3.

Since \( a(\ell), j \leq \ell \leq j+p \), enable us to determine \( A^j_p \), it is readily seen that we can determine all the \( A^j_p \) for \( j+p \leq L \), \( j \geq 0 \), \( p \geq 0 \), whenever \( a(\ell), 0 \leq \ell \leq L \), are given. If we now introduce the element \( a(L+1) \), we can compute those \( A^j_p \) for which \( j+p = L+1 \), in addition to those already computed. That is to say, the calculational flow is along the diagonals \( j+p \) = constant in the \( j-p \) plane.

Along a diagonal all quantities \( \psi_p^j(b) \) in general, and \( \psi_p^j(a) \) and \( \psi_p^j(I) \) in particular, are advanced using the primary recursion in (2.10). For this at each lattice point \( (j,p) \) the denominator \( D_p^j = D_p^j(0) \) must be evaluated, and this is accomplished as follows:

For all values of \( p \) we use Theorem 2 to compute \( D_p^j(q) \) for \( 1 \leq q \leq \min(p-1,m) \) and \( \psi_p^j(q) \) for \( 2 \leq q \leq \min(p,m) \). For \( p \leq m-1 \) we also need \( \psi_p^j(p+1) \), and this can be computed by Theorem 3, \( D_p^j \) in this theorem being computed from Theorem 8 and the new algorithm given in the appendix. We next evaluate \( D_p^j = D_p^j(0) \) by Theorem 6 for \( i_k < p < i_{k+1} \) when \( i_{k+1} > i_k + 1 \), and by (2.12) for \( p = i_k, 1 \leq k \leq m \).

With \( D_p^j \) determined, we compute \( \psi_p^j(1) \) by Theorem 2, and \( \psi_p^j(s_{i_{s+1}}) \), \( k+1 \leq s \leq m \), \( \psi_p^j(a), \psi_p^j(I) \), and \( A^j_p \), thus completing the calculation.

We now express all these steps of the \( W^{(m)} \)-algorithm semiformally in Pascal. For simplicity of notation we shall define \( \bar{g}_k(\ell) = g_{i_{k+1}}(\ell) \), \( 1 \leq k \leq m \). Below by "initialize \((j)\)" we shall mean
\{ read \ t_j, \ a(t_j) = \frac{\lambda^j}{a_0}, \ \psi_k(t_j), \ 1 \leq k \leq m, \ and \ compute \\
\psi^j_0(a) = a(j)/g_1(j); \ \psi^j_0(1) = 1/g_1(j), \ \psi^j_0(l) = 1/t_j, \ \psi^j_0(g_1) = t_j; \\
\psi^j_0(\bar{g}_k(j)) = \frac{\bar{g}_k(j)}{g_1(j)}, \ 1 \leq k \leq m; \ \psi^j_0(h_1) = \frac{\varphi_1(t_j)}{\varphi_1(t_j)}, \ 2 \leq i \leq m}. \\

Note also that statements of the form "for i:=i' to i" do" are not executed if i' > i". We assume that a(\lambda) are given for 0 \leq \lambda \leq L, but are being introduced one by one.
begin
  initialize (0);
  for \( t := 1 \) to \( L \) do
  begin
    initialize \( t \);
    for \( p := 1 \) to \( t \) do
    begin
      \( j := t - p \);
      if \( p \leq m-1 \) then
      begin
        \[
        \begin{align*}
        p_j &= p_{j+1}(h_{p+1}) - p_{j-1}(h_{p+1}) \\
        \text{for } i := p+2 \text{ to } m \text{ do } & \psi_j^p(h_i) := [\psi_j^{p+1}(h_i) - \psi_j^{p-1}(h_i)]/\psi_j^p; \\
        \psi_j^p(g_i) &:= [\psi_j^{p+1}(g_i) - \psi_j^{p-1}(g_i)]/\psi_j^p; \quad \psi_j^{p+1}(p) := (-1)^p/\psi_j^p(g_1).
        \end{align*}
        \]
      end;
      for \( q := 1 \) to \( \min\{p-1, m-1\} \) do
      begin
        \[
        \begin{align*}
        d_j^p(q) &= \psi_j^{q+1}(q)/\psi_j^p(q) - \psi_j^{q+1}(q)/\psi_j^p(q); \quad q' := q+1;
        \psi_j^p(q') &:= [\psi_j^{q+1}(q') - \psi_j^{q+1}(q')] / \psi_j^p(q);
        \end{align*}
        \]
      end;
      if \( p > m \) then
      begin
        \[
        \begin{align*}
        d_j^p(m) &= \psi_j^{q+1}(m)/\psi_j^p(m) - \psi_j^{q+1}(m)/\psi_j^p(m);
        \end{align*}
        \]
      end;
      \{determine that integer \( k, 1 \leq k \leq m \), for which \( i_k \leq p < i_{k+1} \).
      \}(For \( m+1 \leq p \) use the largest positive integer available
      on the computer.);\)
      if \( p = i_k \) then \( d_j^p := \psi_j^{q+1}(g_k) - \psi_j^p(g_k); \)
      if \( i_k < p \) and \( p < i_{k+1} \) then \( d_j^p := d_j^p(k); \)
      if \( p < q \) and \( p < i_{k+1} \) then \( d_j^p := (-1)^k d_j^p(k) \psi_j^p(k+1); \)
      for \( i := k+1 \) to \( m \) do
      begin
        \[
        \begin{align*}
        \psi_j^p(g_i) &:= [\psi_j^{q+1}(g_i) - \psi_j^{q+1}(g_i)] / \psi_j^p(g_i); \\
        \psi_j^p(a) &:= [\psi_j^{q+1}(a) - \psi_j^{q+1}(a)] / \psi_j^p(a); \\
        \psi_j^p(I) &:= [\psi_j^{q+1}(I) - \psi_j^{q+1}(I)] / \psi_j^p(I); \\
        \psi_j^p(p) &:= \psi_j^p(a) / \psi_j^p(I);
        \end{align*}
        \]
      end;
    end;
  end;
end;
We now turn to the questions of storage and operation count for the \( W^{(m)} \)-algorithm.

As can be seen from the above, not everything has to be saved throughout the course of computation. Before \( \ell \) is incremented in the statement "for \( \ell := 1 \) to \( L \) do" the following newly computed quantities are saved: \( \psi^j_p(h_{\ell}) \), \( p+2 \leq i \leq m \), and \( \psi^j_p(g_{\ell}) \) for \( p \leq m-1 \), \( \psi^j_p(q) \), \( 1 \leq q \leq \min(p+1,m) \); \( \psi^j_p(\alpha_{\ell}) \), \( k+1 \leq i \leq m \), for \( i_k \leq p < i_{k+1} \); \( \psi^j_p(a) \) and \( \psi^j_p(I) \); \( j+p = \ell \), \( j \geq 0 \), \( p \geq 0 \). With suitable programming these quantities can occupy the storage locations of those computed in the previous stage of this statement. None of the \( D^j_p(q) \), \( 0 \leq q \leq \min(p-1,m) \) has to be saved. Thus we need to save approximately \( m+2 \) vectors of length \( L \) when \( L \) is large. The rest of the storage required does not increase with \( L \).

As for the operation count, we first observe that there are \( L^2/2 + O(L) \) lattice points \((j,p)\) for \( j+p \leq L \), \( j \geq 0 \), \( p \geq 0 \). At each point \((j,p)\) we compute \( A^j_p \), \( \psi^j_p(a) \), \( \psi^j_p(I) \), \( \psi^j_p(q) \), \( 1 \leq q \leq \min(p+1,m) \), and \( D^j_p(q) \), \( 1 \leq q \leq \min(p-1,m) \). One division is required for \( A^j_p \), one division and one subtraction for each of the \( \psi^j_p(a) \), \( \psi^j_p(I) \), and \( \psi^j_p(q) \), and two divisions and one subtraction for each of \( D^j_p(q) \). Consequently for \( j+p \leq L \), \( j \geq 0 \), \( p \geq 0 \), the computation of the above mentioned quantities requires \( 3(m+1)L^2/2 + O(L) \) divisions and \( 2(m+1)L^2/2 + O(L) \) subtractions. The operation count for the rest of the quantities is \( O(L) \) divisions and subtractions. Thus the final count for the \( W^{(m)} \)-algorithm is \( 3(m+1)L^2/2 + O(L) \) divisions and \( 2(m+1)L^2/2 + O(L) \) subtractions, a total of \( 5(m+1)L^2/L + O(L) \) operations.
For the \( W^{(1)} \) and \( W^{(2)} \) algorithms the storage requirements and
the operation count are reduced considerably by making use of the
results in (3.21) and (3.22). Thus for \( m=1 \) we need to compute
and store only two vectors of dimension \( L \), namely \( \psi_p^j(a) \) and
\( \psi_p^j(I) \) for \( j+p = \text{constant} \), and the operation count is \( 3L^2/2 + 0(L) \)
divisions and \( 3L^2/2 + 0(L) \) subtractions. Similarly, for \( m=2 \)
we need to compute and store only three vectors of dimension \( L \),
namely \( \psi_p^j(a) \), \( \psi_p^j(I) \), and \( \psi_p^j(h_1) = \psi_p^j(I) \) for \( j+p = \text{constant} \),
and the operation count is \( 5L^2/2 + 0(L) \) divisions, \( L^2 + 0(L) \)
multiplications, and \( 2L^2 + 0(L) \) subtractions.

We recall that the sequences \( A_n^{(m,j)} \) with \( j \) fixed and \( n \)
increasing simultaneously have the best convergence properties.
Thus we may be satisfied computing only \( A_p^o \) for \( p = 1,2,\ldots \),
or even for \( p = i_m + ms, s = 0,1,\ldots \). This reduces the number
of divisions by \( L^2/2 + 0(L) \) in all cases above.

Finally, a stopping criterion, based on the values of \( A_p^o \) for
\( p = i_m + ms, s = 0,1,\ldots \), can also be incorporated within the
loop "for \( L=1 \) to \( L \) do". The simplest approach is to estimate
the truncation error from the difference of two successive
approximations, e.g., \( |A_p^{o+m} - A_p^o| \). A better method is to estimate
also the growth of roundoff error in the computations, and stop
when this becomes of the order of the truncation error. One means
of estimating roundoff of \( A_p^j \) is
\[
\left| \sum_{i=0}^{p} \gamma_{p,i} \right|
\]
for \( \gamma_{p,i} \) is the cofactor of \( a(j+i) \) in \( f_p^j(a) \) divided by \( f_p^j(I) \),
and \( \gamma_{p,i} = a(j+i) \). A more accurate but more
laborious method is to compute
\[
\left( \sum_{i=0}^{p} \gamma_{p,i} \right) \frac{\partial A_p^j}{\partial a(\lambda)} \left| \delta a(\lambda) \right|^{1/2}
\]
where \( \delta a(\lambda) \) is...
an estimate of the uncertainty in $a(\ell)$, the $\ell$-th member of the sequence to be accelerated. Because the $W(m)$-algorithm is recursive in nature, formulas for $\frac{\partial A^j}{\partial a(\ell)}$ can be obtained trivially, but are expensive to evaluate. (The idea was implemented in HURRY [3], an adaptive algorithm for accelerating convergence using the $u$-transformation of [5], where it proved to be very accurate.)

A simple and inexpensive way of estimating roundoff, based solely on the computed values of the $A^j_p$ could be as follows: Since for increasing $\ell$ the approximations $A^0_\ell, A^1_\ell, \ldots, A^s_\ell$, for $s$ fixed and small, are close to $A$ and to each other, take (some combination of) the differences $|A^0_\ell - A^j_\ell|$, $1 \leq j \leq s$, as an estimate of the total error (including roundoff) in the computed value of $A^0_\ell$. The extrapolation process can be stopped when the sequence of these errors begins to increase, i.e., when roundoff starts to dominate the computations.

The FORTRAN program for the $W(m)$-algorithm that is given in Appendix B does not include the simpler versions of $W(1)$- and $W(2)$-algorithms. The storage used in the program is about twice the minimal storage described above. We have preferred to give up some of the storage in order to make the program readable. The program also does not include a stopping criterion. All this is left to the interested reader.
5. APPLICATION TO RATIONAL APPROXIMATIONS FROM THE d-TRANSFORMATION

As mentioned in [8], the d-transformation of [6] for accelerating the convergence of infinite series is a form of GREP applied to the sequence of partial sums of these series. When applied to a power series \( \sum_{i=1}^{\infty} c_i z^{i-1} \), this transformation produces approximations \( d_n^{(m,j)}(z) \) to the limit or antilimit that are rational functions in \( z \). In [13] the d-transformation has slightly been modified such that the coefficients of the above mentioned rational approximations are obtained by solving a linear system of equations that are independent of \( z \). Clearly, this results in considerable savings when approximations are required for many different values of \( z \), since otherwise for each \( z \) one has to solve the linear systems of equations that define the d-transformation. For further details see [13].

With a slight change of notation, the approximation \( d_n^{(m,j)}(z) \) to the limit or antilimit of the power series \( \sum_{i=1}^{\infty} c_i z^{i-1} \) is determined by solving the linear system of equations:

\[
d_n^{(m,j)}(z) = S_z + \sum_{k=1}^{m} w_k \sum_{i=0}^{n_k} \frac{\bar{\theta}_{ki}}{(\ell+1)^i}, \quad j \leq \ell \leq j+N, (5.1)
\]

where \( n = (n_1, n_2, \ldots, n_m) \) and \( N = \sum_{k=1}^{m} (n_k+1) \) as before, \( w_k \leq m \) are some fixed integers which can usually be taken to be 0, and

\[
S_o = 0, \quad S_z = \sum_{i=1}^{g} c_i z^{i-1}, \quad \ell \geq 1. \quad (5.2)
\]

For the important questions of when to expect the d-transformation to be efficient, how to determine \( m \) and the integers \( w_k \), when to set \( w_k = 0 \), we refer the reader to [13].
Comparing (5.1)-(5.2) with (1.4), we see that the d-transformation above is indeed a form of GREP for which

\[ t_z = (z+1)^{-1}, \quad a(z) = S_z, \quad \varphi_k(t_z) = \frac{w_k}{c_k + z}, \quad 1 \leq k \leq m, \quad (5.3) \]

and consequently \[ \Phi_n^{(m,j)}(z) = d_n^{(m,j)}(z) \] and \[ \bar{\theta}_k = z^{k-1} \theta_k. \]

It turns out, see [13], that

\[ d_n^{(m,j)}(z) = \sum_{i=0}^{N} \frac{\lambda_i z^{N-i} a(j+1)}{N} \]

where the \( \lambda_i \) are independent of \( z \). Thus \( d_n^{(m,j)}(z) \) is a rational function whose numerator and denominator polynomials are of degree \( j+N-1 \) and \( N \), respectively. The form of \( d_n^{(m,j)}(z) \) in (5.4) can still be achieved if we allow \( \varphi_k(t_z) \) in (5.3) to read more generally

\[ \varphi_k(t_z) = v_k(z) z^k, \quad 1 \leq k \leq m; \tag{5.5} \]

where \( v_k(z) \) can be any numbers. Thus (5.1) becomes

\[ d_n^{(m,j)}(z) = a(z) + \sum_{k=1}^{m} v_k(z) z^k \sum_{i=0}^{n_k} \bar{\theta}_i t_z^i, \quad j \leq z \leq j+N, \tag{5.6} \]

and we shall deal with (5.6) in the remainder of this section.

As in Section 3, let us assume that we would like to compute \( d_n^{(m,j)}(z) \) for \( n = (v_1+s,v_2+s,...,v_m+s) \), \( s = 0,1,2,... \), and that \( v_1 \geq v_2 \geq ... \geq v_m \) are fixed. Define the \( i_k \) as in (3.1), and set

\[ g_{i_k}(z) = v_k(z), \quad 1 \leq k \leq m, \tag{5.7} \]

and define the rest of the \( g_z(z) \) as in (3.6). For \( d_n^{(m,j)}(z) \)
obtained by applying the modified d-transformation to power series as in (5.1), (5.7) is replaced by

\[ g_{i_k}(\ell) = z^{w_k} c_{\ell+k}, \quad 1 \leq k \leq m. \]  

(5.7)

If, in (5.6), we let \( \alpha_{ij}(\ell) = A^j_N \), and replace the \( v^j_k(\ell) t^j_k \) by the appropriate \( g^j_m(\ell) \), and finally multiply (5.6) by \( z^{-\ell} \), we obtain

\[ z^{-\ell} A^j_N = z^{-\ell} a(\ell) + \sum_{i=1}^{N} \alpha^j_i g^j_i(\ell), \quad 1 \leq \ell \leq j+N, \]  

(5.8)

cf. (2.1). Using Cramer's rule, the solution for \( A^j_N \) can be expressed as

\[ A^j_N = \frac{f^j_N(\xi)}{f^j_N(\eta)}, \]  

(5.9)

where \( f^j_N(\xi) \) is as defined by (2.2), and

\[ \xi(\ell) = z^{-\ell} a(\ell), \quad \eta(\ell) = z^{-\ell}, \quad \ell \geq 0. \]  

(5.10)

Dividing the numerator and the denominator of (5.9) by \( C^j_{N+1} \), \( A^j_N \) can also be expressed as

\[ A^j_N = \frac{\psi^j_N(\xi)}{\psi^j_N(\eta)}. \]  

(5.11)

Here \( \psi^j_N(\xi) \) and \( \psi^j_N(\eta) \) can be evaluated recursively by using (2.10), and for this we need to know the \( D^j_p \), which are defined in (2.9). Since \( G^j_p \) are independent of \( z \), so are \( D^j_p \). Thus they have to be computed only once, and this can be accomplished efficiently by using the \( W^{(m)} \)-algorithm. Subsequently they can be used to compute \( \psi^j_N(\xi) \) and \( \psi^j_N(\eta) \) recursively for all values of \( z \).
Expanding $f_j^p(n)$ with respect to its first column, we see that

$\psi_j^p(n)$ is a polynomial in $z^{-1}$ of degree $j+p$ of the form

$$\psi_j^p(n) = \sum_{i=0}^{p} \lambda_j^p, z^{-j-j}.$$  \hspace{1cm} (5.12)

Thus, by expanding $\psi_j^p(\xi)$ with respect to its first column, we obtain

$$\psi_j^p(\xi) = \sum_{i=0}^{p} \lambda_j^p, z^{-j-j}a(j+1).$$  \hspace{1cm} (5.13)

Note that by multiplying $\psi_j^p(\xi)$ and $\psi_j^p(n)$ by $z^{j+p}$, and substituting in (5.11), we recover (5.4). As is seen from (5.12) and (5.13), if we know the $\lambda_j^p, z$, we can use them to compute both $\psi_j^p(n)$ and $\psi_j^p(\xi)$. By (2.10), the $\lambda_j^p, z$ can be computed recursively as follows:

$$\lambda_j^p, z = \begin{cases} 
-\lambda_j^{p-1,0}/D_j^p, & i = 0 \\
(\lambda_j^{p+1,1-1} - \lambda_j^{p-1,1})/D_j^p, & 1 \leq i \leq p-1 \\
\lambda_j^{p-1,p-1}/D_j^p, & i = p 
\end{cases}$$  \hspace{1cm} (5.14)

with

$$\lambda_j^{0,0} = 1/g_{1}(j), \hspace{1cm} j \geq 0.$$  \hspace{1cm} (5.15)

**ACKNOWLEDGEMENT**

The work of the second author was done under NASA contract NAS3-24231 at NASA Lewis Research Center, Cleveland, Ohio.
REFERENCES


APPENDIX A

Prior to Theorem 1 we introduced a recursive algorithm for implementing the general extrapolation procedure defined by (2.1), assuming that we did not have detailed knowledge concerning $g_1(t)$. Actually, the $g_1(t)$ can be assumed to be arbitrary.

In this appendix we give the details of this algorithm, at each stage of which only one new term of the sequence $a(t)$, $t > 0$, is made available. Thus the computational flow is along the diagonals $j + p = \text{constant}$. Here are the stages of the algorithm when we have assumed that $a(t)$ are available for $0 \leq t \leq L$, but again they are being introduced one by one:
\{read \ a(0)=A_o, g_1(0)\}

\psi^0(a) := a(0)/g_1(0); \ \psi^0(I) := 1/g_1(0);

\{save \ \psi^0(a), \ \psi^0(I)\}

\begin{verbatim}
for \ l := 1 \ to \ L \ do

begin

\{read \ \ a(i)=A_{i}, g_1(i), 1 \leq i \leq l, \ g_{l+1}(j), 0 \leq j \leq l\}

\psi^0_{l}(a) := a(l)/g_1(l); \ \psi^0_{l}(I) := 1/g_1(l);

for \ i := 2 \ to \ l \ do

\psi^0_{l}(g_{l+1}) := g_{l+1}(l)/g_1(l);

for \ j := 0 \ to \ l \ do

\psi^0_{l}(g_{l+1}) := g_{l+1}(j)/g_1(j);

for \ p := 1 \ to \ l-1 \ do

for \ j := 0 \ to \ l-1-p \ do

\psi^0_{p}(g_{l+1}) := [\psi^0_{p-1}(g_{l+1}) - \psi^0_{p-1}(g_{l+1})]/D^p_{l};

for \ p := 1 \ to \ l-1 \ do

begin

j := l-p;

p^j_{p-1} := \psi^j_{p-1}(g_{l+1}) - \psi^j_{p-1}(g_{l+1});

for \ i := p+2 \ to \ l+1 \ do

\psi^j_{p}(g_{l+1}) := [\psi^j_{p-1}(g_{l+1}) - \psi^j_{p-1}(g_{l+1})]/D^j_{p};

\psi^j_{p}(a) := [\psi^j_{p-1}(a) - \psi^j_{p-1}(a)]/D^j_{p}; \ \psi^j_{p}(I) := [\psi^j_{p-1}(I) - \psi^j_{p-1}(I)]/D^j_{p};

A^j_{p} := \psi^j_{p}(a)/\psi^j_{p}(I)

end

\{save \ \psi^j_{p}(a), \ \psi^j_{p}(I), \ j+p=l, \ 0 \leq p \leq l, \ \psi^j_{p}(g_{l+1}), \ p+2 \leq l+1, \ j+p=l, \ 0 \leq p \leq l-1, \ and \ discard \ those \ previously \ saved; \ save \ all \ D^j_{p}, \ 1 \leq j \leq l, \ 1 \leq p \leq l\}

end;

\end{verbatim}

Note that statements of the form "for \ k := k_1 \ to \ k_2 \ do" are not executed if \ k_1 > k_2."
The main difference between the present algorithm and the E-algorithm is that in the present algorithm recursion is among the $\psi_p^j(b)$, whereas in the E-algorithm it is among the $B_p^j(b) = f_p^j(b)/f_p^j(I)$. Thus in the notation of the present work the recursion for the E-algorithm as stated in [2] becomes

$$B_p^j(b) = \frac{B_p^{j+1}(b)B_p^{j}(g_p) - B_p^{j}(b)B_p^{j+1}(g_p)}{B_p^{j+1}(g_p) - B_p^{j+1}(g_p)}, \quad (A.1)$$

where $b(\ell)$ is either $a(\ell)$ or $g_i(\ell)$, $i \geq p+1$. Note that $B_p^j(g_i)$, denoted by $g_{p,i}^{(j)}$ in [2], is zero for $i \leq p$. Also note that $B_p^j(g) = A_p^j$. (A.1) can be obtained by writing

$$B_p^j(b) = \frac{f_p^j(b)G_p^{j+1}}{f_p^j(I)G_p^{j+1}} \frac{f_p^j(I)G_p^{j+1}}{f_p^j(I)G_p^{j+1}},$$

applying (2.8) to both the numerator and denominator of this quotient, and finally by dividing both numerator and denominator by $f_p^j(I)G_p^{j+1}(I)$, and realizing that $G_p^j = f_p^j(I)G_p^{j+1}(I)$.

Now most of the computational effort is spent in obtaining the $\psi_p^j(g_i)$ in the present algorithm and the $B_p^j(g_i)$ in the E-algorithm, the rest of the effort being relatively minor in both cases. Given $a(\ell)$, $0 \leq \ell \leq \ell_1$ the computation of the $A_p^j$, $j+p \leq \ell_1$, $j \geq 0$, $p \geq 0$, requires knowledge of $L^3/3 + O(L^2)$ $\psi_p^j(g_i)$'s for the present algorithm and $L^3/3 + O(L^2)$ $B_p^j(g_i)$'s for the E-algorithm. Thus, as mentioned also in [2], if one uses (A.1) to implement the E-algorithm, one needs $2L^3/3 + O(L^2)$ multiplications, $2L^3/3 + O(L^2)$ subtractions, and $L^3/3 + O(L^2)$ divisions. The E-algorithm can be implemented in a more efficient way by rewriting (A.1) in the form

$$B_p^j(b) = \frac{B_p^{j+1}(b) - B_p^j(b)c_p^j}{1 - c_p^j}, \quad (A.1)'$$
where \( c^j_p = \frac{B^j_{p+1}(g_p)}{B^j_p(g_p)} \), and computing \( c^j_p \) and \( 1 - c^j_p \) only once, the operation count becomes \( L^3/3 + O(L^2) \) multiplications, \( L^3/3 + O(L^2) \) subtractions, and \( L^3/3 + O(L^2) \) divisions, a total of \( L^3 + O(L^2) \) operations. For the new algorithm, however, the operation count is \( L^3/3 + O(L^2) \) divisions and \( L^3/3 + O(L^2) \) subtractions, a total of \( 2L^3/3 + O(L^2) \) operations. Thus the new algorithm is about 30% more economical than the E-algorithm even when the latter is implemented using (A.1).

Finally, \( A^j_p \) can be expressed as

\[
A^j_p = \sum_{i=0}^{p} \gamma^{j}_{p,i} a(j+i), \quad (A.2)
\]

where

\[
\gamma^{j}_{p,i} = \frac{\lambda^{j}_{p,i}}{\sum_{s=0}^{p} \lambda^{j}_{p,s}}, \quad 0 \leq i \leq p, \quad (A.3)
\]

and the \( \lambda^{j}_{p,i} \) are computed using the recursion relation in (5.14) and (5.15). The demonstration of (A.3) is left to the reader.
APPENDIX B

In this appendix we give a FORTRAN program that includes the subroutine subprogram WMALGM, for the \( W^{(m)} \)-algorithm with normal ordering. The program illustrates the use of the \( W^{(m)} \)-algorithm as the d-transformation. We recall that the d-transformation is a form of CREP used in accelerating the convergence of infinite series whose terms form sequences in the family \( B^{(m)} \) described in [6]. The program has been written in (standard) FORTRAN 77.

WMALGM

The CALL statement for WMALGM is as follows:

```
CALL WMALGM(MDIM,LDIM,M,LMAX,MLTAG,G,PSIAI,BIGPSI,PSIG,APPROX,EPSDIV)
```

- **MDIM** - a positive integer. (Input of integer type).
- **LDIM** - a positive integer. (Input of integer type).
- **M** - the value of \( m \) in the text. \( M \leq MDIM \). (Input of integer type).
- **LMAX** - the maximum number of terms \( a(l) \) made available to WMALGM.
  
  \( LMAX \leq LDIM \). Starting with \( a(0) \) the terms \( a(1), a(2), \) are introduced one at a time. (Input of integer type).
- **MLTAG** - the name of an external subroutine subroutine the CALL statement for which is

```
CALL MLTAG(M,L,T,A,G)
```

- **M** - The value of \( m \) as before. (Input of integer type).
- **L** - An integer \( \geq 0 \). (Input of integer type).
- **T** - \( t_L \) in the text. (Output of double precision type).
- **A** - \( a(L) \) in the text. (Output of double precision type).
- **G** - A one-dimensional array containing the \( g_i(L) \), \( 1 \leq i \leq m \), in the text. Here \( G(K) = g_K(L) \), \( 1 \leq K \leq M \). (Output of double precision type).

This subroutine subroutine, is to be supplied by the user and must be declared in an EXTERNAL statement in the calling program.
G — Exactly as described in MLTAG.

PSIAI — An array of dimension (0:LDIM,2;2). (Output of double precision type).

BIGPSI — An array of dimension (0:LDIM,MDIM,2). (Output of double precision type).

PSIG — An array of dimension (0:MDIM,2:MDIM+1,2). (Output of double precision type).

APPROX — An array of dimension (0:LDIM,0:LDIM) with APPROX(J,P)=\( A^J_P \) in the text. (Output of double precision type).

EPSDIV — A small positive constant used to avoid division by zero in the computation \( A^J_P = \frac{v^J_P(a)}{\psi^J_P(I)} \). If \( |\psi^J_P(I)| < \text{EPSDIV} \), then \( A^J_P \) is not computed, the sentence "APPROX(J,P) IS NOT DEFINED" is printed instead. (Input of double precision type).

WMAVLGM can be used with any M and LMAX.

In the listing of WMAVLGM below

\[
\begin{align*}
\text{PSIAI}(P,1,*) &= \psi^*_P(a), & \text{PSIAI}(P,2,*) &= \psi^*_P(I), \\
\text{BIGPSI}(P,Q,*) &= \psi^*_P(Q), & \text{PSIG}(P,I,*) &= \psi^*_P(g_I),
\end{align*}
\]  

(B.1)

where

\[
J^* = \begin{cases} 
L-1-P & \text{when } * \text{ denotes CUR} \\
L-P & \text{when } * \text{ denotes TEMP}
\end{cases}
\]  

(B.2)

Note: If we want to save only \( A^0_P \), p \geq 0, we do not have to compute or save APPROX(J,P) for \( 1 \leq J \leq L \), thus reducing the storage requirement substantially. This can be achieved by deleting all reference to APPROX(J,P) for \( J \geq 1 \), and then replacing APPROX(0,P) by APPROX(P). Of course,
APPROX(Ø:LDIM, Ø:LDIM) in the DIMENSION statement should be replaced by APPROX(Ø:LDIM). Also, as mentioned in Section 4, before the statement "B6 CONTINUE" we can insert a stopping criterion.

MLTAG for d-transformation

We now describe an example of the subroutine subprogram MLTAG, namely, for use in accelerating the convergence of infinite series of the form

\[ \sum_{r=0}^{\infty} u_r \]

by the d-transformation. In general, if we select an increasing sequence of integers, \( R_k, 0 \leq R_0 < R_1 < R_2 < \ldots \), then for given \( m \) and \( \ell \)

\[ t_\ell = 1/(R_\ell + 1) \quad \text{and} \quad a(\ell) = \sum_{r=0}^{R_\ell} u_r, \quad \ell \geq 0, \]

and we can set

\[ g_k(\ell) = (R_\ell + 1)^{-k} \Delta_{k-1} u_{R_k}, \quad 1 \leq k \leq m, \quad \ell \geq 0, \]

where \( \Delta \) is the forward difference operator, defined by

\[ \Delta u_1 = u_1, \quad \Delta u_1 = u_{i+1} - u_i, \quad \Delta^k u_i = \Delta(\Delta^{k-1} u_i), \quad k \geq 2. \]

The simplest choice for the \( R_\ell \) is (1) \( R_\ell = \ell, \quad \ell \geq 0 \), although others like (2) \( R_\ell = s \ell, \quad \ell \geq 0, \) for \( s \) any positive integer, or (3) \( R_0 = 0, R_{\ell+1} = [\sigma R_\ell] + 1, \quad \ell \geq 0, \) for \( \sigma \) some positive number (not necessarily integer) > 1, are also possible and useful in appropriate cases. For example (1) is useful for alternating series, (2) with \( s \geq 2 \), for power series, trigonometric Fourier series, Fourier-Legendre series, Fourier-Bessel series, etc., close to singularities of the functions they represent (away from singularities \( s = 1 \) is normally sufficient), and (3) for monotonic series or for those series whose terms can be expressed as \( u_r = u_r^{(1)} + u_r^{(2)} \) such that the \( W^{(m)} \)-transformation, with appropriate values of \( m \), can be applied to both series \( \sum_{r=0}^{\infty} u_r^{(1)} \) and \( \sum_{r=0}^{\infty} u_r^{(2)} \), and at least one of these series is monotonic.
In the listing of MLTAG below, SIGMA and INCR stand for $\sigma$ and $s$ respectively. SIGMA = \(1\) and INCR = \(s, s = 1, 2, \ldots\), give the choices (1) and (2), whereas INCR = \(1\) and SIGMA > \(1\) give the choice (3). Both parameters are passed to MLTAG from the main program by a COMMON statement.

The final value of LSUM in MLTAG is \(R_L\).

The function CF(I) which is called by MLTAG returns the value of $u_I$, i.e., the I-th coefficient of the infinite series \(\sum_{r=0}^{\infty} u_r\).

Note that we have given up computational efficiency in MLTAG in favor of its readability. Of course, efficiency can be achieved by saving previously computed values of CF(I) and $A$.

**MAIN PROGRAM**

The quantities to be supplied by the user are MDIM, LDIM, EPSDIV, M, LMAX, INCR, and SIGMA (all exactly as described above), and they appear in PARAMETER statements. Recall that $M \leq \text{MDIM}$ and $LMAX \leq \text{LDIM}$ must be satisfied.

The program as given below applies the d-transformation to the series

\[
\sum_{i=0}^{\infty} \left( \frac{1}{(i+1)^{3/2}} + \frac{1}{(i+1)^2} \right)
\]

with $R_0 = 0$, $R_{i+1} = (1.3R_i) + 1$, $\ell > 0$, i.e., INCR = \(1\) and SIGMA = \(1.3\). Part of the computer output is given below:

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$R_\ell$</th>
<th>$A_\ell$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.20030000000000000D+01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.35224077499274639D+01</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.43783330679178020D+01</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.42143708741705962D+01</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.42936946135757038D+01</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0.42449082853639200D+01</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0.42579024538962556D+01</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>0.4257227978412741D+01</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>0.4257267050283350D+01</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>0.4257314273661214D+01</td>
</tr>
<tr>
<td>10</td>
<td>27</td>
<td>0.4257310654917329D+01</td>
</tr>
<tr>
<td>11</td>
<td>36</td>
<td>0.4257309211241072D+01</td>
</tr>
<tr>
<td>12</td>
<td>47</td>
<td>0.4257309428939371D+01</td>
</tr>
<tr>
<td>13</td>
<td>62</td>
<td>0.425730942937747D+01</td>
</tr>
<tr>
<td>14</td>
<td>81</td>
<td>0.4257309414616707D+01</td>
</tr>
<tr>
<td>15</td>
<td>106</td>
<td>0.42573094155571107D+01</td>
</tr>
</tbody>
</table>

The exact sum of this series is \(\zeta(3/2) + \zeta(2) = 4.25709415533\ldots\), where $\zeta(z)$ is the Riemann Zeta-function.

Here are the listings of WMALGM, MLTAG, CF, and the main driving program:
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
PARAMETER (NDIM=6,LDIM=50,EPSPDIV=10**77)
PARAMETER (M=2,LMAX=20,INCR=1,SIGMA=1.300)
DIMENSION G(NDIM),PSIAI(0:LDIM,2,2),BIGPSI(0:LDIM,MDIM,2)
DIMENSION PSIG(0:MDIM,2:MDIM+1,2),APPROX(0:LDIM,0:LDIM)
EXTERNAL MLTAG
COMMON SIGMAP,INCRP
WRITE(*,111)
111 FORMAT('SUMMATION OF '1/(1+I)**(3/2)+1/(1+I)**2', I=0 TO I=INFINITY')
    INCRP=INCR
    SIGMAP=SIGMA
    CALL WMALGM(NDIM,LDIM,M,LMAX,MLTAG,G,PSIAI,BIGPSI,PSIG
    APPROX,EPSPDIV)
    WRITE(*,121)
121 FORMAT('J P APPROX(J,P)')
    WRITE(*,131)((L-I,I,APPROX(L-I,I),I=0,L),L=0,LMAX)
131 FORMAT(214.D25,E6)
END

SUBROUTINE WMALGM(MDIM,LDIM,M,LMAX,MLTAG,G,PSIAI,BIGPSI,PSIG.
    APPROX,EPSPDIV)
IMPLICIT DOUBLE, PRECISION(A-H,O-Z)
INTEGER CUR,TMP,P,PM,QP
DIMENSION G(NDIM),PSIAI(0:LDIM,2,2)
DIMENSION BIGPSI(0:LDIM,MDIM,2),PSIG(0:MDIM,2:MDIM+1,2)
DIMENSION APPROX(0:LDIM,0:LDIM)
CUR=1
TEMP=2
CALL MLTAG(M,0,T,A,G), APPROX(0,0)=A
PSIAI(0,1,CUR)=A/G(1)
PSIAI(0,2,CUR)=100/G(1)
BIGPSI(0,1,CUR)=100/TEMP
DO 10 K=2,M
PSIG(0,K,CUR)=G(K)/G(1)
10 CONTINUE
PSIG(0,M+1,CUR)=T
DO 80 L=1,LMAX
CALL MLTAG(M,L,T,A,G)
    APPROX(L,0)=A
    PSIAI(0,1,TMP)=A/G(1)
    PSIAI(0,2,TMP)=100/G(1)
    BIGPSI(0,1,TMP)=100/TEMP
    DO 20 K=2,M
        PSIG(0,K,TMP)=G(K)/G(1)
20 CONTINUE
PSIG(0,M+1,TMP)=T
SIGN=-100
DO 60 P=1,L
    IF (P.LE.M) THEN
        D=PSIG(P-1,P+1,TMP)-PSIG(P-1,P+1,CUR)
        DO 30 I=P+2,M+1
            PSIG(P,I,TMP)=(PSIG(P-1,I,TMP)-PSIG(P-1,I,CUR))/D
        30 CONTINUE
    END IF
60 CONTINUE
IF (P.LT.M) THEN
  BIGPSI(P,P+1,TEMP)=SIGN/PSIG(P,M+1,TEMP)
  SIGN=-SIGN
END IF
PN=MIN(P-1,N-1)
DO 40 Q=1,PN
  PS=BIGPSI(P-2,Q,CUR)
  DQ=PS/BIGPSI(P-1,Q,CUR)-PS/BIGPSI(P-1,Q,TEMP)
Q=Q+1
  BIGPSI(P,QP,TEMP)=(BIGPSI(P-1,QP,TEMP)-BIGPSI(P-1,QP,CUR))/DQ
CONTINUE
40 IF (P.GT.M) THEN
  PS=BIGPSI(P-2,M,CUR)
  D=PS/BIGPSI(P-1,M,CUR)-PS/BIGPSI(P-1,M,TEMP)
END IF
  BIGPSI(P,TEMP)=(BIGPSI(P-1,1,TEMP)-BIGPSI(1,1,CUR))/D
DO 50 I=1,2
  PSIAI(P,I,TEMP)=(PSIAI(P-1,I,TEMP)-PSIAI(P-1,I,CUR))/D
CONTINUE
50 CONTINUE
60 DO 70 P=1,L
  J=L-P
  IF (DABS(PSIAI(P,2,TEMP)).GE.EPSDIV) THEN
    APPROX(J,P)=PSIAI(P,1,TEMP)/PSIAI(P,2,TEMP)
  ELSE
    APPROX(J,P)=10/5
    WRITE(*,'(6,101)') J,P
  101 FORMAT(1X,'APPROX(',I3,',',I3,') IS NOT DEFINED')
END IF
70 CONTINUE
JJ=CUR
CUR=TEMP
TEMP=JJ
80 CONTINUE
RETURN
END
SUBROUTINE MLTAG(M, L, TAG, G)
IMPLICIT DOUBLE PRECISION(A-H, O-Z)
DIMENSION G(M)
COMMON SIGMA, INCR
LSUM=0
DO 10 I=1, L
LSUM=SIGMA*LSUM+INCR
10 CONTINUE
PARSUM=0.0
DO 20 I=0, LSUM
PARSUM=PARSUM+CFL(I)
20 CONTINUE
P=LSUM+1
T=1.0/P
A=PARSUM
DO 30 K=1, M
G(K)=CFL(LSUM+K-1)
30 CONTINUE
DO 50 I=2, M
DO 40 J=I, I-1
G(J)=G(J)-G(J-1)
40 CONTINUE
50 CONTINUE
DO 60 K=1, M
G(K)=G(K)*P
P=P*(LSUM+1)
60 CONTINUE
RETURN
END

FUNCTION CFL(I)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
CFL=1.0/(I+1.0)**1.5D0+1.0/(I+1.0)**2
RETURN
END