SIAM J. NUMER. ANAL.
Vol. 24, No. 5, October 1987

AN ALGORITHM FOR A GENERALIZATION OF THE RICHARDSON EXTRAPOLATION PROCESS*

WILLIAM F. FORD† AND AVRAM SIDI‡

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.

Key words. convergence acceleration, generalized Richardson extrapolation, recursive algorithms

AMS(MOS) subject classifications. 65B05, 40A25

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. It allows, for example, the summation in just a few terms of series as simple as $\sum_{n=0}^{\infty} (1+\sqrt{n})/(1+n)^2$, or as difficult as $\sum_{n=0}^{\infty} P_n(x) \cos ny$. The technique can be applied equally well to integrals, ranging from simple ones like $\int_0^\infty \sin(ax^2+bx)\,dx$ to difficult ones like $\int_0^\infty J_0(x)J_1(ax)J_2(bx)\,dx$.

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

$$A_n^{(m,j)} = A(y_l) + \sum_{k=1}^{m} \phi_k(y_l) \sum_{i=0}^{n_k} \bar{\beta}_{ki} y_l^{ir}, \qquad j \leqq l \leqq j+N, \tag{1.1}$$

where n denotes $(n_1, n_2, \cdots, n_m)$, $N = \sum_{k=1}^{m} (n_k+1)$, the y_l, $A(y_l)$ and $\phi_k(y_l)$ are given, r is a known positive constant, and $A_n^{(m,j)}$ and the $\bar{\beta}_{ki}$ 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 r_k.

* Received by the editors August 19, 1985; accepted for publication (in revised form) September 10, 1986.

† Computer Services Division, National Aeronautics and Space Administration Lewis Research Center, Cleveland, Ohio 44135.

‡ Computer Science Department, Technion-Israel Institute of Technology, Haifa 32000, Israel. The work of this author was supported by National Aeronautics and Space Administration contract NAS3-24231 and was performed at National Aeronautics and Space Administration Lewis Research Center, Cleveland, Ohio 44135.

The above-mentioned W-algorithm of [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 $0<y\leqq b$, for some $b>0$, where y can be a discrete or continuous variable, belongs to the set $F^{(m)}$, if there exist functions $\phi_k(y)$, $\beta_k(y)$, $k=1,\cdots,m$, and a constant A such that

$$A=A(y)+\sum_{k=1}^{m}\phi_k(y)\beta_k(y), \tag{1.2}$$

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

$$\beta_k(\xi)\sim\sum_{i=0}^{\infty}\beta_{ki}\xi^{ir} \quad \text{as } \xi\to 0+. \tag{1.3}$$

If, in addition, the functions $B_k(t)=\beta_k(t^{1/r})$ are infinitely differentiable for $0\leqq t\leqq b^r$, we shall say that $A(y)$ belongs to the set $F_\infty^{(m)}$. When $\lim_{y\to 0+}A(y)$ does not exist, A is called the antilimit of $A(y)$.

In attempting to accelerate the convergence of a sequence that can be identified with $A(y)$, the idea 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 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 $\beta_{kn_k}y^{n_kr}$, replacing A and the β_{ki} by $A_n^{(m,j)}$ and $\bar{\beta}_{ki}$ respectively, and finally collocating at the points $y_j, y_{j+1},\cdots,y_{j+N}$. Here the y_l are chosen in such a way that $b\geqq y_0>y_1>\cdots>0$, and $\lim_{l\to\infty}y_l=0$, 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, § 2]. Some examples of GREP are the D-transformation of [6] for infinite integrals, the $\bar{D}$- and $\tilde{D}$-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 §§ 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 § 5 we shall consider the case in which $\phi_k(y_l)=v_k(l)z^l$, where $v_k(l)$ 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 § 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 from 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 $t_l=y_l^r$, $l=0,1,\cdots$, and define $a(t)\equiv A(y)$ and $\varphi_k(t)\equiv\phi_k(y)$, $k=1,\cdots,m$. Then (1.1) becomes

$$A_n^{(m,j)}=a(t_l)+\sum_{k=1}^{m}\varphi_k(t_l)\sum_{i=0}^{n_k}\bar{\beta}_{ki}t_l^i,\qquad j\leqq l\leqq j+N. \tag{1.4}$$

In the remainder of this paper we shall work with these equations.

2. Theoretical preliminaries. Consider the equations in (1.4). Let us denote the $\varphi_k(t_l)t_l^i$ and $\bar{\beta}_{ki}$, $0\leqq i\leqq n_k$, $1\leqq k\leqq m$ (ordered in a manner to be described later), by $g_\mu(l)$ and α_μ, $1\leqq\mu\leqq N$, respectively. We note that the exact nature of this ordering is crucial in the development of the $W^{(m)}$-algorithm. For present purposes, however, this ordering is irrelevant, and in fact, throughout most of this section, the $g_\mu(l)$ can be treated as arbitrary. Let us also denote $A_n^{(m,j)}$ by A_N^j, and $a(t_l)$ by $a(l)$. Then (1.4) can be expressed as

$$A_N^j=a(l)+\sum_{i=1}^{N}\alpha_i g_i(l),\qquad j\leqq l\leqq j+N. \tag{2.1}$$

By Cramer's rule A_N^j can be expressed as the ratio of two determinants, namely as

$$A_N^j=\frac{f_N^j(a)}{f_N^j(I)}=\frac{|g_1(j)\cdots g_N(j)a(j)|}{|g_1(j)\cdots g_N(j)I(j)|}, \tag{2.2}$$

where $I(l)=1$, $l\geqq0$, and $|u_1(j)\cdots u_p(j)|$ is the $p\times p$ determinant

$$|u_1(j)\cdots u_p(j)|=\begin{vmatrix}u_1(j)&\cdots&u_p(j)\\ \vdots&&\vdots\\ u_1(j+p-1)&\cdots&u_p(j+p-1)\end{vmatrix}. \tag{2.3}$$

Let us now denote

$$G_p^j=|g_1(j)\cdots g_p(j)|,\qquad G_0^j=1, \tag{2.4}$$

and, for arbitrary $b(l)$, $l\geqq0$, define

$$\psi_p^j(b)=f_p^j(b)/G_{p+1}^j. \tag{2.5}$$

Then (2.2) may be reexpressed as

$$A_N^j=\psi_N^j(a)/\psi_N^j(I). \tag{2.6}$$

Because $f_p^j(b)$ differs from G_{p+1}^j by only one column in the determinantal array, 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 ρ and column σ of C. Also let $C_{\rho\rho';\sigma\sigma'}$ denote the matrix obtained by deleting rows ρ and ρ' and columns σ and σ' of C. Provided $\rho<\rho'$ and $\sigma<\sigma'$*

$$\det C\det C_{\rho\rho';\sigma\sigma'}=\det C_{\rho\sigma}\det C_{\rho'\sigma'}-\det C_{\rho\sigma'}\det C_{\rho'\sigma}. \tag{2.7}$$

If C is a 2×2 matrix, then (2.7) *holds with* $\det C_{\rho\rho';\sigma\sigma'}=1$.

Applying Theorem 1 to the $(p+1)\times(p+1)$ determinant $f_p^j(b)=|g_1(j)\cdots g_p(j)b(j)|$ with $\rho=1$, $\sigma=p$, $\rho'=\sigma'=p+1$, and using (2.4), we obtain

$$f_p^j(b)G_{p-1}^{j+1}=f_{p-1}^{j+1}(b)G_p^j-f_{p-1}^j(b)G_p^{j+1}. \tag{2.8}$$

This is the desired relation.

Upon invoking (2.5), and letting

$$D_p^j = \frac{G_{p+1}^j G_{p-1}^{j+1}}{G_p^j G_p^{j+1}}, \tag{2.9}$$

(2.8) becomes

$$\psi_p^j(b) = [\psi_{p-1}^{j+1}(b) - \psi_{p-1}^j(b)]/D_p^j. \tag{2.10}$$

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

$$\psi_p^j(g_{p+1}) = 1. \tag{2.11}$$

Consequently (2.10) becomes

$$D_p^j = \psi_{p-1}^{j+1}(g_{p+1}) - \psi_{p-1}^j(g_{p+1}), \tag{2.12}$$

which permits a recursive evaluation of D_p^j through the quantities $\psi_p^j(g_i)$, $i \geqq p+1$. (Note that $\psi_p^j(g_i) = 0$ for $1 \leqq i \leqq p$.) 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 $g_i(l)$ 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 $A_p^j = A_n^{(2,j)}$ for $n = (0, 0), (1, 0), (1, 1), (2, 1), (2, 2)$, etc. For this we set

$$g_i(l) = \varphi_i(t_l), \quad i = 1, 2, \quad g_i(l) = t_l g_{i-2}(l), \quad i = 3, 4, \cdots.$$

Therefore, for $p \geqq 3$

$$G_p^j = |\varphi_1(t_j)\ \varphi_2(t_j)\ t_j\varphi_1(t_j)\ t_j\varphi_2(t_j) \cdots g_p(j)|.$$

If we factor out t_{j+i-1} from the ith row, $i = 1, 2, \cdots, p$, we obtain

$$G_p^j = \left(\prod_{i=1}^{p} t_{j+i-1}\right)|h_1(j)h_2(j)g_1(j) \cdots g_{p-2}(j)|,$$

where

$$h_i(l) = \frac{\varphi_i(t_l)}{t_l}, \qquad i = 1, 2.$$

The determinant obtained above differs from G_p^j 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 $f_p^j(b)$, $\psi_p^j(b)$, and D_p^j:

$$F_p^j(h_1, \cdots, h_q) = |g_1(j) \cdots g_p(j)h_1(j) \cdots h_q(j)| \equiv F_p^j(q), \tag{2.13}$$

$$\Psi_p^j(h_1, \cdots, h_q) = F_{p+1-q}^j(q)/F_{p+2-q}^j(q-1) \equiv \Psi_p^j(q), \tag{2.14}$$

$$D_p^j(q) = \frac{F_{p+1-q}^j(q)F_{p-1-q}^{j+1}(q)}{F_{p-q}^j(q)F_{p-q}^{j+1}(q)}, \tag{2.15}$$

the quantities $h_i(l)$ being arbitrary so far.

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

$$F_p^j(0)=G_p^j, \qquad F_p^j(1)=f_p^j(h_1), \tag{2.16}$$

$$\Psi_p^j(1)=\psi_p^j(h_1) \tag{2.17}$$

and

$$D_p^j(0)=D_p^j, \tag{2.18}$$

respectively. In addition, we define $F_0^j(0)=1$.

We also note that since $F_p^j(q)$ is defined for $p\geqq 0$ and $q\geqq 0$, $\Psi_p^j(q)$ is defined for $1\leqq q\leqq p+1$, and $D_p^j(q)$ is defined for $0\leqq q\leqq p-1$.

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

$$(-1)^{q-1}h_q(l)=\sum_{i=1}^{p-q+1}\alpha_i g_i(l)+g_{p-q+2}(l)\Psi_p^j(q)+\sum_{i=1}^{q-1}\alpha_{p-q+i+1}h_i(l), \qquad j\leqq l\leqq j+p,$$

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 $\Psi_p^j(q)$ and $D_p^j(q)$ satisfy*

$$\Psi_p^j(q)=[\Psi_{p-1}^{j+1}(q)-\Psi_{p-1}^j(q)]/D_p^j(q-1), \qquad 1\leqq q\leqq p, \tag{2.19}$$

and

$$D_p^j(q)=\Psi_{p-2}^{j+1}(q)\left[\frac{1}{\Psi_{p-1}^j(q)}-\frac{1}{\Psi_{p-1}^{j+1}(q)}\right], \qquad 1\leqq q\leqq p-1. \tag{2.20}$$

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

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

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). When we invoke (2.14), (2.15) becomes

$$D_p^j(q)=\frac{\Psi_p^j(q)\Psi_{p-2}^{j+1}(q)}{\Psi_{p-1}^j(q)\Psi_{p-1}^{j+1}(q)}D_p^j(q-1). \tag{2.22}$$

When we substitute (2.19) in (2.22), (2.20) follows. □

The recursion relations given in (2.19) and (2.20) will be applied, for a given p, with q increasing from 1. When $q=p-1$, (2.20) requires knowledge of $\Psi_{p-2}^{j+1}(p-1)$, and when $q=p$, (2.19) requires knowledge of $\Psi_{p-1}^j(p)$ and $\Psi_{p-1}^{j+1}(p)$. Now $\Psi_p^j(p+1)$ cannot be computed using the recursion relation in (2.19) since neither $\Psi_{p-1}^j(p+1)$ nor $D_p^j(p)$ is defined. However, it is possible to derive a recursion relation among the $\Psi_p^j(p+1)$, and this is given in Theorem 3 below.

THEOREM 3. *Let*

$$\hat{\psi}_p^j(b)=\frac{|h_1(j)\cdots h_p(j)b(j)|}{|h_1(j)\cdots h_{p+1}(j)|}. \tag{2.23}$$

Then, for arbitrary $b(l)$,

$$\hat{\psi}_p^j(b)=[\hat{\psi}_{p-1}^{j+1}(b)-\hat{\psi}_{p-1}^j(b)]/\hat{D}_p^j, \tag{2.24}$$

where

$$\hat{D}_p^j=\hat{\psi}_{p-1}^{j+1}(h_{p+1})-\hat{\psi}_{p-1}^j(h_{p+1}) \tag{2.25}$$

and

$$\Psi_p^j(p+1) = (-1)^p/\hat{\psi}_p^j(g_1). \tag{2.26}$$

Proof. First (2.26) follows directly from the fact that $\Psi_p^j(p+1) = F_0^j(p+1)/F_1^j(p)$. Next, we note that the definition of $\hat{\psi}_p^j(b)$ as given in (2.23) is very similar to that of $\psi_p^j(b)$ as given in (2.5), the only difference being that $g_i(l)$ in (2.5) has been replaced by $h_i(l)$ in (2.23). This immediately suggests that (2.10)-(2.12) hold with $\psi_p^j(b)$, D_p^j and $g_i(l)$, $i=1,2,\cdots$, replaced by $\hat{\psi}_p^j(b)$, $\hat{D}_p^j$ and $h_i(l)$, $i=1,2,\cdots$, respectively. This proves (2.24) and (2.25). □

What we see from Theorem 3 is that the computation of the $\Psi_p^j(p+1)$ can be achieved through that of the $\hat{\psi}_p^j(g_1)$, which in turn can be achieved by using the algorithm given in Appendix A. As we shall see in the next section, the $h_i(l)$ in GREP have no particular structure in general, thus the actual computation of the $\Psi_p^j(p+1)$ 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 $\Psi_p^j(q)$.

THEOREM 4. *The $\Psi_p^j(q)$ satisfy the relation*

$$F_{p+1-q}^j(q) = \Psi_p^j(1)\Psi_p^j(2)\cdots\Psi_p^j(q)G_{p+1}^j. \tag{2.27}$$

Proof. Let us reexpress (2.14) in the form

$$F_{p+1-q}^j(q) = \Psi_p^j(q)F_{p+1-(q-1)}^j(q-1). \tag{2.28}$$

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

$$F_{p+1-q}^j(q) = \Psi_p^j(q)\cdots\Psi_p^j(1)F_{p+1}^j(0). \tag{2.29}$$

Equation (2.27) follows from (2.29) and (2.16). □

3. Ordering of $\varphi_k(t_l)t_l^i$ and its consequences.

Numerical experience and theoretical results suggest that as the n_k in (1.4) become large simultaneously, usually $A_n^{(m,j)} \to A$ quickly where A is the limit or antilimit of the sequence $a(l)$, $l=0,1,2,\cdots$. The simplest way of achieving this is by letting $n_k = s$, $1 \leqq k \leqq m$, and increasing s. More generally, we can let $n_k = \nu_k + s$ for some fixed integers $\nu_k \geqq 0$, $1 \leqq k \leqq m$, and $s = 0,1,2,\cdots$. With the $\varphi_k(t_l)t_l^i$, $1 \leqq k \leqq m$, $i \geqq 0$, 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 $\nu_1 \geqq \nu_2 \geqq \cdots \geqq \nu_m$. We define the indices i_k by

$$i_{m+1} = \infty, \quad i_1 = 1, \quad i_{k+1} = i_k + 1 + k(\nu_k - \nu_{k+1}), \quad 1 \leqq k \leqq m-1. \tag{3.1}$$

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

$$c_k = (i_{k+1} - i_k - 1)/k, \qquad 1 \leqq k \leqq m, \tag{3.2}$$

are nonnegative integers for $1 \leqq k \leqq m-1$, and $c_m = \infty$. Let us define

$$c_{k_1}^{k_2} = \sum_{j=k_1}^{k_2} c_j, \quad 1 \leqq k_1 \leqq k_2 \leqq m, \quad c_{k_1}^{k_2} = 0 \text{ otherwise.} \tag{3.3}$$

For each ordered pair of integers (k,i), $1 \leqq k \leqq m$, $i \geqq 0$, we define, in one-to-one correspondence, a single integer $\mu \geqq 1$ by

$$\mu = \begin{cases} i_k & \text{if } i = 0, \\ i_q - (q-k) + (i - c_k^{q-1})q & \text{if } i \geqq 1 \quad \text{and} \quad c_k^{q-1} < i \leqq c_k^q. \end{cases} \tag{3.4}$$

Finally, we set

$$\varphi_k(t_l)t_l^i = g_\mu(l), \tag{3.5}$$

thus determining the desired ordering. We note that the condition $c_k^{q-1} < i \leqq c_k^q$ 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. $m = 5$, $\nu_1 = 7$, $\nu_2 = 5$, $\nu_3 = 4 = \nu_4$, $\nu_5 = 2$. 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 kth row denotes $\varphi_k(t_l)$, and the ith circle in the same row denotes $\varphi_k(t_l)t_l^i$. The integer below each square or circle is μ in $g_\mu(l) = \varphi_k(t_l)t_l^i$. Note that the m rows have been arranged so that the terms $\varphi_k(t_l)t_l^{\nu_k}$ are in the same column, and that c_k denotes the number of columns that contain those μ's satisfying $i_k < \mu < i_{k+1}$.

0	1	2	3	4	5	6	$\nu_1=7$	8	9
□	○	○	○	○	○	○	○	○	○
$i_1=1$	2	3	5	9	13	18	23	28	33
		0	1	2	3	4	$\nu_2=5$	6	7
		□	○	○	○	○	○	○	○
		$i_2=4$	6	10	14	19	24	29	34
			0	1	2	3	$\nu_3=4$	5	6
			□	○	○	○	○	○	○
			$i_3=7$	11	15	20	25	30	35
			0	1	2	3	$\nu_4=4$	5	6
			□	○	○	○	○	○	○
			$i_4=8$	12	16	21	26	31	36
					0	1	$\nu_5=2$	3	4
					□	○	○	○	○
					$i_5=17$	22	27	32	37

With the ordering above the sequence of approximations $A_{n(s)}^{(m,j)}$, where $n(s) = (\nu_1 + s, \cdots, \nu_m + s)$, $s = 0, 1, 2, \cdots$, is actually a subsequence of the sequence A_p^j, $p = 0, 1, 2, \cdots$, and $A_{n(s)}^{(m,j)} = A_{N_0+ms}^j$, where $N_0 = \sum_{k=1}^m (\nu_k + 1)$.

As a consequence of this ordering we have

$$\frac{g_i(l)}{t_l} = \begin{cases} h_k(l), & i = i_k, \\ g_{i-k}(l), & i_k < i < i_{k+1}, \end{cases} \qquad 1 \leqq k \leqq m, \tag{3.6}$$

where we have defined

$$h_k(l) \equiv \frac{\varphi_k(t_l)}{t_l} = \frac{g_{i_k}(l)}{t_l}, \qquad 1 \leqq k \leqq m. \tag{3.7}$$

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 case of the construction above is that for which $\nu_k = 0$, $1 \leqq k \leqq m$, which we shall henceforth refer to as the "normal ordering." For this case we have

$$i_k = k, \quad 1 \leqq k \leqq m, \qquad c_k = 0, \quad 1 \leqq k \leqq m-1, \tag{3.8}$$

$$\varphi_k(t_l) t_l^i = g_{k+im}(l), \qquad 1 \leqq k \leqq m, \quad i \geqq 0, \tag{3.9}$$

$$g_i(l) = \begin{cases} \varphi_i(t_l), & 1 \leqq i \leqq m, \\ t_l g_{i-m}(l), & i > m, \end{cases} \tag{3.10}$$

and

$$h_k(l) = \frac{\varphi_k(t_l)}{t_l} = \frac{g_k(l)}{t_l}, \qquad 1 \leqq k \leqq m. \tag{3.11}$$

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

Since we have defined $h_k(l)$ only for $1 \leqq k \leqq m$, we see that $F_p^j(q)$ is defined only for $0 \leqq q \leqq m$, $p \geqq 0$, $\Psi_p^j(q)$ for $1 \leqq q \leqq \min\{p+1, m\}$, and $D_p^j(q)$ for $0 \leqq q \leqq \min\{p-1, m\}$. Consequently, the recursion relation in (2.19) is valid for $1 \leqq q \leqq \min\{p, m\}$, and that in (2.20) for $1 \leqq q \leqq \min\{p-1, m\}$.

Through $g_i(l) = t_l g_{i-m}(l)$, $i > i_m$, cf. (3.6), the ordering above enables us to relate $F_{p-m}^j(m)$ to G_p^j and hence $D_p^j(m)$ to $D_p^j(0) = D_p^j$, 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*

$$\sigma_k = \sum_{s=1}^{k} (i_s - s), \qquad 1 \leqq k \leqq m, \tag{3.12}$$

and

$$\pi_p^j = \prod_{i=j}^{j+p-1} t_i, \qquad p \geqq 1. \tag{3.13}$$

Then for $i_k \leqq p < i_{k+1}$

$$G_p^j = (-1)^{\sigma_k + k^2 + pk} \pi_p^j F_{p-k}^j(k). \tag{3.14}$$

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

(3.15)

$$G_p^j = \pi_p^j | h_1 g_1 \cdots g_{i_2-2} h_2 g_{i_2-1} \cdots g_{i_3-3} h_3 g_{i_3-2} \cdots g_{i_4-4} h_4 \cdots g_{i_k-k} h_k g_{i_k-k+1} \cdots g_{p-k} |,$$

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 σ_k. Thus

$$G_p^j = (-1)^{\sigma_k} \pi_p^j | h_1(j) \cdots h_k(j) g_1(j) \cdots g_{p-k}(j) |. \tag{3.16}$$

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

THEOREM 6. *For $i_k < p < i_{k+1}$ and $k < i_k = p$, D_p^j can be computed from*

$$(3.17)\quad D_p^j = D_p^j(0) = D_p^j(k) \times \begin{cases} \Psi_p^j(k+1)/\Psi_{p-2}^{j+1}(k), & k < i_k = p = i_{k+1}-1, \\ (-1)^k/\Psi_{p-2}^{j+1}(k), & k < i_k = p < i_{k+1}-1, \\ 1, & i_k < p < i_{k+1}-1, \\ (-1)^k \Psi_p^j(k+1), & i_k < p = i_{k+1}-1. \end{cases}$$

Proof. By (2.9) D_p^j is a combination of G_{p+1}^j, G_{p-1}^{j+1}, G_p^j and G_p^{j+1}. For $i_k < p < i_{k+1}-1$, we also have $i_k \leqq p \pm 1 < i_{k+1}$. Consequently, we can make direct use of Lemma 5 on all four $G_{p'}^{j'}$ above. Invoking now (2.15), we obtain $D_p^j = D_p^j(k)$. When $i_k < p = i_{k+1}-1$, we have $i_k \leqq p-1 < i_{k+1}$ and $i_k < p < i_{k+1}$, but $p+1 = i_{k+1}$. Thus we can make direct use of Lemma 5 on G_{p-1}^{j+1}, G_p^j, and G_p^{j+1} only. We apply Lemma 5 to G_{p+1}^j with k in (3.14) replaced by $k+1$. The overall result of this is

$$(3.18)\quad D_p^j = (-1)^k \frac{F_{p+1-(k+1)}^j(k+1) F_{p-1-k}^{j+1}(k)}{F_{p-k}^j(k) F_{p-k}^{j+1}(k)}.$$

Multiplying both the numerator and the denominator of (3.18) by $F_{p+1-k}^j(k)$, and invoking (2.14) and (2.15), we obtain $D_p^j = (-1)^k D_p^j(k) \Psi_p^j(k+1)$. The proof of the rest of (2.17) is accomplished similarly. □

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

$$(3.19)\quad D_p^j = D_p^j(m), \qquad p > i_m,$$

and it is this property that makes the $W^{(m)}$-algorithm economical. Observe that in the general algorithm outlined in § 2 and described in detail in Appendix A, the calculation of any D_p^j requires roughly j recursive steps. As more terms are introduced j becomes large and so do the recursive calculations. For the $W^{(m)}$-algorithm, however, calculation of any D_p^j, on account of (3.19), never requires more than m recursive steps.

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

$$(3.17)'\quad D_p^j = D_p^j(0) = \begin{cases} [\Psi_{p-1}^{j+1}(k+1) - \Psi_{p-1}^j(k+1)]/\Psi_{p-2}^{j+1}(k), & k < i_k = p = i_{k+1}-1, \\ (-1)^k[1/\Psi_{p-1}^j(k) - 1/\Psi_{p-1}^{j+1}(k)], & k < i_k = p < i_{k+1}-1, \\ D_p^j(k), & i_k < p < i_{k+1}-1, \\ (-1)^k[\Psi_{p-1}^{j+1}(k+1) - \Psi_{p-1}^j(k+1)], & i_k < p = i_{k+1}-1. \end{cases}$$

The only case not covered by Theorem 6 is that of $p = i_k = k$, and D_p^j 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 $k \leqq k_1$ and $i_k > k$ when $k > k_1$, then we can use (2.10) and (2.12) for $1 \leqq p \leqq k_1$, and Theorem 6 for $k > k_1$. In the $W^{(m)}$-algorithm given in the next section, however, we choose to compute D_p^j for $p = i_k$, $1 \leqq k \leqq m$, using (2.10) and (2.12), thus making the programming simpler.

THEOREM 7. *When $i_k - 1 \leqq p \leqq i_{k+1}-2$*

$$(3.20)\quad (-1)^{\sigma_k + pk} \pi_{p+1}^j \prod_{i=1}^{k} \Psi_p^j(i) = 1.$$

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) $D_p^j = D_p^j(1)$ for $p \geqq 2$. Similarly from (3.20) $(-1)^p \pi_{p+1}^j \Psi_p^j(1) = 1$ for $p \geqq 0$. Thus solving for $\Psi_p^j(1)$, and substituting in (2.20) with $q = 1$ there, we obtain

$$(3.21)\quad D_p^j = t_{j+p} - t_j,$$

which is valid for $p \geqq 2$. Using (2.9), we see that (3.21) is 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) $D_p^j = D_p^j(2)$ for $p \geqq 3$. Similarly from (3.20) $\pi_{p+1}^j \Psi_p^j(1)\Psi_p^j(2)=1$ for $p \geqq 1$. Solving for $\Psi_p^j(2)$ in terms of $\Psi_p^j(1)=\psi_p^j(h_1)$, and substituting in (2.20) with $q=2$ there, we obtain

$$D_p^j = [t_j \psi_{p-1}^j(h_1) - t_{j+p}\psi_{p-1}^{j+1}(h_1)]/\psi_{p-2}^{j+1}(h_1), \tag{3.22}$$

which is valid for $p \geqq 3$. Using (2.9), we can show that (3.22) is valid also for $p=2$. As for $p=1$, we use

$$D_1^j = \frac{g_2(j+1)}{g_1(j+1)} - \frac{g_2(j)}{g_1(j)} = \frac{\varphi_2(t_{j+1})}{\varphi_1(t_{j+1})} - \frac{\varphi_2(t_j)}{\varphi_1(t_j)}, \tag{3.23}$$

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 $\varphi_1(t_l)$, $\varphi_2(t_l)$, $l \geqq 0$, use (3.23) and (2.10) to obtain the $\psi_1^j(h_1)$. Then for $p=2,3,\cdots$, use (3.22) to obtain the D_p^j, and then (2.10) to obtain the $\psi_p^j(h_1)$.

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

In Theorem 3 we obtained a recursion relation among the $\hat{\psi}_p^j(b)$, which, through (2.26) enables us to determine $\Psi_p^j(p+1)$. Since we have defined $h_k(l)$ only for $1 \leqq k \leqq m$, the $\Psi_p^j(p+1)$ are defined and needed only for $0 \leqq p \leqq m-1$. In general the $\Psi_p^j(p+1)$, $0 \leqq p \leqq m-1$, and the D_p^j, $p=i_k$, $1 \leqq k \leqq m$, are computed by different 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.,* $i_k=k$, $1 \leqq k \leqq m$, *we have*

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

and

$$\pi_{p+1}^j \prod_{i=1}^{p+1} \Psi_p^j(i) = 1, \qquad 0 \leqq p \leqq m-1. \tag{3.25}$$

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_p^j efficiently when the $\varphi_k(t_l)t_l^i$ are ordered as in the beginning of § 3.

Since $a(l)$, $j \leqq l \leqq j+p$, enable us to determine A_p^j, it is readily seen that we can determine all the A_p^j for $j+p \leqq L$, $j \geqq 0$, $p \geqq 0$, whenever $a(l)$, $0 \leqq l \leqq L$, are given. If we now introduce the element $a(L+1)$, we can compute those A_p^j 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 $1 \leqq q \leqq \min\{p-1, m\}$ and $\Psi_p^j(q)$ for $2 \leqq q \leqq \min\{p, m\}$. For $p \leqq m-1$ we also need $\Psi_p^j(p+1)$. This can be computed by Theorem 3, $\hat{D}_p^j$ in this theorem being computed from Theorem 8 and the new algorithm given in Appendix A. 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 $p = i_k$, $1 \leqq k \leqq m$. With D_p^j determined, we compute $\Psi_p^j(1)$ by Theorem 2, and $\psi_p^j(g_{i_s+1})$, $k+1 \leqq s \leqq m$, $\psi_p^j(a)$, $\psi_p^j(I)$, and A_p^j, 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 $\bar{g}_k(l) = g_{i_k+1}(l)$, $1 \leqq k \leqq m$. Below by "initialize (j)" we shall mean

{read t_j, $a(t_j) = A_0^j$, $\varphi_k(t_j)$, $1 \leqq k \leqq m$, and compute $\psi_0^j(a) = a(j)/g_1(j)$, $\psi_0^j(I) = 1/g_1(j)$, $\Psi_0^j(1) = 1/t_j$, $\hat{\psi}_0^j(g_1) = t_j$; $\psi_0^j(\bar{g}_k) = \bar{g}_k(j)/g_1(j)$, $1 \leqq k \leqq m$; $\hat{\psi}_0^j(h_i) = \varphi_i(t_j)/\varphi_1(t_j)$, $2 \leqq i \leqq m$}.

Note also that statements of the form "**for** $i := i'$ **to** i'' **do**" are not executed if $i' > i''$. We assume that $a(l)$ are given for $0 \leqq l \leqq L$, but are being introduced one by one.

```
begin
    initialize (0);
    for $l := 1$ to $L$ do
    begin
        initialize ($l$);
        for $p := 1$ to $l$ do
        begin
            $j := l - p$;
            if $p \leqq m - 1$ then
            begin
                $\hat{D}_p^j := \hat{\psi}_{p-1}^{j+1}(h_{p+1}) - \hat{\psi}_{p-1}^j(h_{p+1})$;
                for $i := p+2$ to $m$ do $\hat{\psi}_p^j(h_i) := [\hat{\psi}_{p-1}^{j+1}(h_i) - \hat{\psi}_{p-1}^j(h_i)]/\hat{D}_p^j$;
                $\hat{\psi}_p^j(g_1) := [\hat{\psi}_{p-1}^{j+1}(g_1) - \hat{\psi}_{p-1}^j(g_1)]/\hat{D}_p^j$; $\Psi_p^j(p+1) := (-1)^p/\hat{\psi}_p^j(g_1)$
            end;
            for $q := 1$ to $\min\{p-1, m-1\}$ do
            begin
                $D_p^j(q) := \Psi_{p-2}^{j+1}(q)/\Psi_{p-1}^j(q) - \Psi_{p-2}^{j+1}(q)/\Psi_{p-1}^{j+1}(q)$; $q' := q+1$;
                $\Psi_p^j(q') := [\Psi_{p-1}^{j+1}(q') - \Psi_{p-1}^j(q')]/D_p^j(q)$
            end;
            if $p > m$ then $D_p^j(m) := \Psi_{p-2}^{j+1}(m)/\Psi_{p-1}^j(m) - \Psi_{p-2}^{j+1}(m)/\Psi_{p-1}^{j+1}(m)$;
            {determine that integer $k$, $1 \leqq k \leqq m$, for which $i_k \leqq p < i_{k+1}$. (For $i_{m+1} = \infty$
             use the largest positive integer available on the computer.)};
            if $p = i_k$ then $D_p^j := \psi_{p-1}^{j+1}(\bar{g}_k) - \psi_{p-1}^j(\bar{g}_k)$;
            if $i_k < p$ and $p < i_{k+1} - 1$ then $D_p^j := D_p^j(k)$;
            if $i_k < p$ and $p = i_{k+1} - 1$ then $D_p^j := (-1)^k D_p^j(k)\Psi_p^j(k+1)$;
            for $i := k+1$ to $m$ do $\psi_p^j(\bar{g}_i) := [\psi_{p-1}^{j+1}(\bar{g}_i) - \psi_{p-1}^j(\bar{g}_i)]/D_p^j$;
            $\Psi_p^j(1) := [\Psi_{p-1}^{j+1}(1) - \Psi_{p-1}^j(1)]/D_p^j$;
            $\psi_p^j(a) := [\psi_{p-1}^{j+1}(a) - \psi_{p-1}^j(a)]/D_p^j$;
            $\psi_p^j(I) := [\psi_{p-1}^{j+1}(I) - \psi_{p-1}^j(I)]/D_p^j$;
            $A_p^j := \psi_p^j(a)/\psi_p^j(I)$
        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 l is incremented in the statement "**for** $l := 1$ **to** L **do**" the following newly computed quantities are saved: $\hat{\psi}_p^j(h_i)$, $p+2 \leqq i \leqq m$, and $\hat{\psi}_p^j(g_1)$

for $p \leqq m-1$, $\Psi_p^j(q)$, $1 \leqq q \leqq \min\{p+1, m\}$; $\psi_p^j(\bar{g}_i)$, $k+1 \leqq i \leqq m$, for $i_k \leqq p < i_{k+1}$; $\psi_p^j(a)$ and $\psi_p^j(I)$; $j+p=l$, $j \geqq 0$, $p \geqq 0$. With suitable programming these quantities can occupy the storage locations of those computed in the previous stage of this statement. None of the $D_p^j(q)$, $0 \leqq q \leqq \min\{p-1, m\}$ 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 $j+p \leqq L$, $j \geqq 0$, $p \geqq 0$. At each point (j, p) we compute A_p^j, $\psi_p^j(a)$, $\psi_p^j(I)$, $\Psi_p^j(q)$, $1 \leqq q \leqq \min\{p+1, m\}$, and $D_p^j(q)$, $1 \leqq q \leqq \min\{p-1, m\}$. One division is required for A_p^j, one division and one subtraction for each of the $\psi_p^l(a)$, $\psi_p^j(I)$ and $\Psi_p^j(q)$, and two divisions and one subtraction for each $D_p^j(q)$. Consequently for $j+p \leqq L$, $j \geqq 0$, $p \geqq 0$, 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 $5(m+1)L^2/2+O(L)$ 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 $j+p=$ constant, and the operation count is $3L^2/2+O(L)$ divisions and $3L^2/2+O(L)$ 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 $\psi_p^j(h_1)=\Psi_p^j(1)$ for $j+p=$ constant, and the operation count is $5L^2/2+O(L)$ divisions, $L^2+O(L)$ multiplications, and $2L^2+O(L)$ subtractions.

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

Finally, a stopping criterion, based on the values of A_p^0 for $p=i_m+ms$, $s=0, 1, \cdots$, 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., $|A_{p+m}^0 - A_p^0|$. 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 $\sum_{i=0}^p |\gamma_{p,i}^j|$ as discussed in [8]. Here $\gamma_{p,i}^j$ is the cofactor of $a(j+i)$ in $f_p^j(a)$ divided by $f_p^j(I)$, cf. (2.6), thus $A_p^j = \sum_{i=0}^p \gamma_{p,i}^j\, a(j+i)$. A more accurate but more laborious method is to compute $\{\sum_{l=j}^{j+p} |(\partial A_p^j/\partial a(l))\, \delta a(l)|^2\}^{1/2}$, where $\delta a(l)$ is an estimate of the uncertainty in $a(l)$, the lth member of the sequence to be accelerated. Because the $W^{(m)}$-algorithm is recursive in nature, formulas for $\partial A_p^j/\partial a(l)$ 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_p^j could be as follows: Since for increasing l the approximations A_l^0, $A_{l-1}^1, \cdots, A_{l-s}^s$, for s fixed and small, are close to A and to each other, take (some combination of) the differences $|A_l^0 - A_{l-j}^j|$, $1 \leqq j \leqq s$, as an estimate of the total error (including roundoff) in the computed value of A_l^0. 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

$$d_n^{(m,j)}(z) = S_l + \sum_{k=1}^{m} l^{w_k} c_{l+k} z^{l+k-1} \sum_{i=0}^{n_k} \frac{\bar{\theta}_{ki}}{(l+1)^i}, \qquad j \leqq l \leqq j+N, \tag{5.1}$$

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

$$S_0 = 0, \quad S_l = \sum_{i=1}^{l} c_i z^{i-1}, \quad l \geqq 1. \tag{5.2}$$

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

$$t_l = (l+1)^{-1}, \quad a(l) = S_l, \quad \varphi_k(t_l) = l^{w_k} c_{l+k} z^l, \quad 1 \leqq k \leqq m, \tag{5.3}$$

and consequently $A_n^{(m,j)} = d_n^{(m,j)}(z)$ and $\bar{\beta}_{ki} = z^{k-1} \bar{\theta}_{ki}$.

It turns out, see [13], that

$$d_n^{(m,j)}(z) = \frac{\sum_{i=0}^{N} \lambda_i z^{N-i} a(j+i)}{\sum_{i=0}^{N} \lambda_i z^{N-i}}, \tag{5.4}$$

where the λ_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 $\varphi_k(t_l)$ in (5.3) to read more generally

$$\varphi_k(t_l) = v_k(l) z^l, \qquad 1 \leqq k \leqq m, \tag{5.5}$$

where $v_k(l)$ can be any numbers. Thus (5.1) becomes

$$d_n^{(m,j)}(z) = a(l) + \sum_{k=1}^{m} v_k(l) z^l \sum_{i=0}^{n_k} \bar{\beta}_{ki} t_l^i, \qquad j \leqq l \leqq j+N, \tag{5.6}$$

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

As in § 3, let us assume that we would like to compute $d_n^{(m,j)}(z)$ for $n = (\nu_1 + s, \nu_2 + s, \cdots, \nu_m + s)$, $s = 0, 1, 2, \cdots$, and that $\nu_1 \geqq \nu_2 \geqq \cdots \geqq \nu_m$ are fixed. Define the i_k as in (3.1), and set

$$g_{i_k}(l) = v_k(l), \qquad 1 \leqq k \leqq m, \tag{5.7}$$

and define the rest of the $g_i(l)$ 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

$$g_{i_k}(l)=l^{w_k}c_{l+k}, \qquad 1\leqq k\leqq m. \tag{5.7$'$}$$

If, in (5.6), we let $d_n^{(m,j)}(z)=A_N^j$, and replace the $v_k(l)t_l^i$ by the appropriate $g_\mu(l)$, and finally multiply (5.6) by z^{-l}, we obtain

$$z^{-l}A_N^j=z^{-l}a(l)+\sum_{i=1}^{N}\alpha_i g_i(l), \qquad j\leqq l\leqq j+N, \tag{5.8}$$

cf. (2.1). According to Cramer's rule, the solution for A_N^j can be expressed as

$$A_N^j=\frac{f_N^j(\xi)}{f_N^j(\eta)}, \tag{5.9}$$

where $f_p^j(b)$ is as defined by (2.2), and

$$\xi(l)=z^{-l}a(l), \quad \eta(l)=z^{-l}, \quad l\geqq 0. \tag{5.10}$$

If we divide the numerator and the denominator of (5.9) by G_{N+1}^j, A_N^j can also be expressed as

$$A_N^j=\frac{\psi_N^j(\xi)}{\psi_N^j(\eta)}. \tag{5.11}$$

Here $\psi_p^j(\xi)$ and $\psi_p^j(\eta)$ can be evaluated recursively by using (2.10), and for this we need to know the D_p^j, which are defined in (2.9). Since G_p^j are independent of z, so are D_p^j. 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 $\psi_p^j(\xi)$ and $\psi_p^j(\eta)$ recursively for all values of z.

Expanding $f_p^j(\eta)$ with respect to its first column, we see that $\psi_p^j(\eta)$ is a polynomial in z^{-1} of degree $\leqq j+p$ of the form

$$\psi_p^j(\eta)=\sum_{i=0}^{p}\lambda_{p,i}^j z^{-j-i}. \tag{5.12}$$

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

$$\psi_p^j(\xi)=\sum_{i=0}^{p}\lambda_{p,i}^j z^{-j-i}a(j+i). \tag{5.13}$$

Note that by multiplying $\psi_p^j(\xi)$ and $\psi_p^j(\eta)$ 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 $\lambda_{p,i}^j$, we can use them to compute both $\psi_p^j(\eta)$ and $\psi_p^j(\xi)$. By (2.10), the $\lambda_{p,i}^j$ can be computed recursively as follows:

$$\lambda_{p,i}^j=\begin{cases}-\lambda_{p-1,0}^j/D_p^j, & i=0,\\ (\lambda_{p-1,i-1}^{j+1}-\lambda_{p-1,i}^j)/D_p^j, & 1\leqq i\leqq p-1,\\ \lambda_{p-1,p-1}^{j+1}/D_p^j, & i=p,\end{cases} \tag{5.14}$$

with

$$\lambda_{0,0}^j=1/g_1(j), \qquad j\geqq 0. \tag{5.15}$$

Appendix A. Following 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 $g_i(l)$. Actually, the $g_i(l)$ 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 $a(l)$, $l \geqq 0$, is made available. Thus the computational flow is along the diagonals $j+p=$ constant. Here are the stages of the algorithm when we have assumed that $a(l)$ are available for $0 \leqq l \leqq L$, but again they are being introduced one by one:

```
{read $a(0)=A_0^0, g_1(0)$}
$\psi_0^0(a) := a(0)/g_1(0)$; $\psi_0^0(I) := 1/g_1(0)$;
{save $\psi_0^0(a), \psi_0^0(I)$}
for $l := 1$ to $L$ do
begin
    {read $a(l)=A_0^l, g_i(l), 1 \leqq i \leqq l, g_{l+1}(j), 0 \leqq j \leqq l$}
    $\psi_0^l(a) := a(l)/g_1(l)$; $\psi_0^l(I) := 1/g_1(l)$;
    for $i := 2$ to $l$ do $\psi_0^l(g_i) := g_i(l)/g_1(l)$;
    for $j := 0$ to $l$ do $\psi_0^j(g_{l+1}) := g_{l+1}(j)/g_1(j)$;
    for $p := 1$ to $l-1$ do for $j := 0$ to $l-1-p$ do $\psi_p^j(g_{l+1}) := [\psi_{p-1}^{j+1}(g_{l+1}) - \psi_{p-1}^j(g_{l+1})]/D_p^j$;
    for $p := 1$ to $l$ do
    begin
        $j := l-p$;
        $D_p^j := \psi_{p-1}^{j+1}(g_{p+1}) - \psi_{p-1}^j(g_{p+1})$;
        for $i := p+2$ to $l+1$ do $\psi_p^j(g_i) := [\psi_{p-1}^{j+1}(g_i) - \psi_{p-1}^j(g_i)]/D_p^j$;
        $\psi_p^j(a) := [\psi_{p-1}^{j+1}(a) - \psi_{p-1}^j(a)]/D_p^j$; $\psi_p^j(I) := [\psi_{p-1}^{j+1}(I) - \psi_{p-1}^j(I)]/D_p^j$;
        $A_p^j := \psi_p^j(a)/\psi_p^j(I)$
    end
    {save $\psi_p^j(a), \psi_p^j(I), j+p=l, 0 \leqq p \leqq l, \psi_p^j(g_i), p+2 \leqq i \leqq l+1, j+p=l, 0 \leqq p \leqq l-1$,
     and discard those previously saved; save all $D_p^j$, $1 \leqq j+p \leqq l$, $1 \leqq p \leqq l$}
end;
```

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

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

where $b(l)$ is either $a(l)$ or $g_i(l)$, $i \geqq p+1$. Note that $B_p^j(g_i)$, denoted by $g_{p,i}^{(j)}$ in [2], is zero for $i \leqq p$. Also note that $B_p^j(a) = A_p^j$. Equation (A.1) can be obtained by writing $B_p^j(b) = [f_p^j(b)G_{p-1}^{j+1}]/[f_p^j(I)G_{p-1}^{j+1}]$, applying (2.8) to both the numerator and denominator of this quotient, and finally by dividing both numerator and denominator by $f_{p-1}^j(I)f_{p-1}^{j+1}(I)$, and realizing that $G_p^j = f_{p-1}^j(g_p)$.

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 $a(l)$, $0 \leqq l \leqq L$, the computation of the A_p^j, $j+p \leqq L$, $j \geqq 0$, $p \geqq 0$, 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

$$B_p^j(b)=\frac{B_{p-1}^{j+1}(b)-B_p^j(b)c_p^j}{1-c_p^j}, \tag{A.1$'$}$$

where $c_p^j=B_{p-1}^{j+1}(g_p)/B_{p-1}^j(g_p)$, and computing c_p^j and $1-c_p^j$ only once, the operation count becoming $L^3/3+O(L^2)$ 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_p^j can be expressed as

$$A_p^j=\sum_{i=0}^{p}\gamma_{p,i}^j a(j+i), \tag{A.2}$$

where

$$\gamma_{p,i}^j=\frac{\lambda_{p,i}^j}{\sum_{s=0}^{p}\lambda_{p,s}^j},\qquad 0\leqq i\leqq p, \tag{A.3}$$

and the $\lambda_{p,i}^j$ 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 GREP used in accelerating the convergence of infinite series whose terms form sequences in the family $\tilde{B}^{(m)}$ 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. $M\leqq$ MDIM. (Input of integer type).
LMAX	The maximum number of terms $a(l)$ made available to WMALGM. LMAX$\leqq$LDIM. Starting with $a(0)$ the terms $a(1)$, $a(2)$, $\cdots$, are introduced one at a time. (Input of integer type).
MLTAG	The name of an external subroutine subprogram 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 $\geqq 0$. (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 $g_i(L)$, $1\leqq i\leqq m$, in the text. Here $G(K)=g_K(L)$, $1\leqq K\leqq M$. (Output of double precision type).

	This subroutine subprogram 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:IDIM,2:MDIM+1,2). (Output of double precision type).
APPROX	An array of dimension (0:LDIM,0:LDIM) with APPROX $(J,P)=A_P^J$ in the text. (Output of double precision type).
EPSDIV	A small positive constant used to avoid division by zero in the computation $A_P^J=\psi_P^J(a)/\psi_P^J(I)$. If $\lvert\psi_P^J(I)\rvert<$ EPSDIV, then A_P^J is not computed, the sentence "APPROX (J,P) IS NOT DEFINED" is printed instead. (Input of double precision type).

WMALGM can be used with any M and LMAX.

In the listing of WMALGM below

$$\begin{aligned}&\text{PSIAI}\,(P,1,*)=\psi_P^{J^*}(a), &&\text{PSIAI}\,(P,2,*)=\psi_P^{J^*}(I),\\ &\text{BIGPSI}\,(P,Q,*)=\Psi_P^{J^*}(Q), &&\text{PSIG}\,(P,I,*)=\psi_P^{J^*}(g_I),\end{aligned}\tag{B.1}$$

where

$$J^*=\begin{cases}L-1-P & \text{when * denotes CUR,}\\ L-P & \text{when * denotes TEMP.}\end{cases}\tag{B.2}$$

Note. If we want to save only A_p^0, $p\geqq 0$, we do not have to compute or save APPROX (J,P) for $1\leqq J\leqq L$, thus reducing the storage requirement substantially. This can be achieved by deleting all reference to APPROX (J,P) for $J\geqq 1$, and then replacing APPROX $(0,P)$ by APPROX (P). Of course, APPROX (0:LDIM, 0:LDIM) in the DIMENSION statement should be replaced by APPROX (0:LDIM). Also, as we mentioned in § 4, before the statement "80 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_l, $0\leqq R_0<R_1<R_2<\cdots$, then for given m and l

$$t_l=\frac{1}{R_l+1}\quad\text{and}\quad a(l)=\sum_{r=0}^{R_l}u_r,\qquad l\geqq 0,\tag{B.3}$$

and we can set

$$g_k(l)=(R_l+1)^k\Delta^{k-1}u_{R_l},\qquad 1\leqq k\leqq m,\quad l\geqq 0,\tag{B.4}$$

where Δ is the forward difference operator, defined by

$$\Delta^0u_i=u_i,\quad \Delta u_i=u_{i+1}-u_i,\quad \Delta^k u_i=\Delta(\Delta^{k-1}u_i),\quad k\geqq 2.\tag{B.5}$$

The simplest choice for the R_l is (1) $R_l=l$, $l\geqq 0$, although others like (2) $R_l=sl$, $l\geqq 0$, for s any positive integer, or (3) $R_0=0$, $R_{l+1}=[\sigma R_l]+1$, $l\geqq 0$, for σ 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 $s\geqq 2$, 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). Also, (3) is useful 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 σ and s respectively. SIGMA = 1 and INCR = s, $s = 1, 2, \cdots$, 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 Ith 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 $M \leqq$ MDIM and LMAX $\leqq$ LDIM must be satisfied.

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

$$\sum_{i=0}^{\infty} \left[\frac{1}{(i+1)^{3/2}} + \frac{1}{(i+1)^2}\right] \quad \text{with } R_0 = 0, \quad R_{l+1} = [1.3R_l] + 1, \quad l \geqq 0,$$

i.e., INCR = 1 and SIGMA = 1.3. Part of the computer output is given in Table 1. The exact sum of this series is $\zeta(3/2) + \zeta(2) = 4.25709415533 \cdots$, where $\zeta(z)$ is the Riemann zeta-function.

TABLE 1

l	R_l	A_l^0
0	0	0.2000000000000000D+01
1	1	0.3522407749927483D+01
2	2	0.4378833067917802D+01
3	3	0.4214370874170596D+01
4	4	0.4203604635757008D+01
5	6	0.4244408285363892D+01
6	8	0.4257902453896256D+01
7	11	0.4257227978412741D+01
8	15	0.4257267095028335D+01
9	20	0.4257314273661214D+01
10	27	0.4257310664017329D+01
11	36	0.4257309211241072D+01
12	47	0.4257309428983710D+01
13	62	0.4257309420937747D+01
14	81	0.4257309414416707D+01
15	106	0.4257309415571107D+01

Here are the listings of WMALGM, MLTAG, CF, and the main driving program:

```
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
PARAMETER (MDIM=6,LDIM=50,EPSDIV=1D-77)
PARAMETER (M=2,LMAX=20,INCR=1,SIGMA=1.3D0)
DIMENSION G(MDIM),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(6,111)
111   FORMAT('SUMMATION OF 1/(I+1)**(3/2)+1/(I+1)**2',1X,
     *        'FROM I=0 TO I=INFINITY')
      INCRP=INCR
      SIGMAP=SIGMA
      CALL WMALGM(MDIM,LDIM,M,LMAX,MLTAG,G,PSIAI,BIGPSI,PSIG,
     *   APPROX,EPSDIV)
      WRITE(6,121)
121   FORMAT(3X,'J',3X,'P',3X,'APPROX(J,P)')
      WRITE(6,131)((L-I,I,APPROX(L-I,I),I=0,L),L=0,LMAX)
131   FORMAT(2I4,D25.16)
      END

      SUBROUTINE WMALGM(MDIM,LDIM,M,LMAX,MLTAG,G,PSIAI,BIGPSI,PSIG,
     *   APPROX,EPSDIV)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER CUR,TEMP,P,PM,Q,QP
      DIMENSION G(MDIM),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)=1D0/G(1)
      BIGPSI(0,1,CUR)=1D0/T
      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,TEMP)=A/G(1)
      PSIAI(0,2,TEMP)=1D0/G(1)
      BIGPSI(0,1,TEMP)=1D0/T
      DO 20 K=2,M
      PSIG(0,K,TEMP)=G(K)/G(1)
20    CONTINUE
      PSIG(0,M+1,TEMP)=T
      SIGN=-1D0
      DO 60 P=1,L
      IF (P.LE.M) THEN
         D=PSIG(P-1,P+1,TEMP)-PSIG(P-1,P+1,CUR)
         DO 30 I=P+2,M+1
         PSIG(P,I,TEMP)=(PSIG(P-1,I,TEMP)-PSIG(P-1,I,CUR))/D
30       CONTINUE
      END IF

      IF (P.LT.M) THEN
         BIGPSI(P,P+1,TEMP)=SIGN/PSIG(P,M+1,TEMP)
         SIGN=-SIGN
      END IF
      PM=MIN0(P-1,M-1)
      DO 40 Q=1,PM
      PS=BIGPSI(P-2,Q,CUR)
      DQ=PS/BIGPSI(P-1,Q,CUR)-PS/BIGPSI(P-1,Q,TEMP)
      QP=Q+1
      BIGPSI(P,QP,TEMP)=(BIGPSI(P-1,QP,TEMP)-BIGPSI(P-1,QP,CUR))/DQ
40    CONTINUE
      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,1,TEMP)=(BIGPSI(P-1,1,TEMP)-BIGPSI(P-1,1,CUR))/D
      DO 50 I=1,2
      PSIAI(P,I,TEMP)=(PSIAI(P-1,I,TEMP)-PSIAI(P-1,I,CUR))/D
50    CONTINUE
60    CONTINUE
      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)=1D75
        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,T,A,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=0D0
      DO 20 I=0,LSUM
      PARSUM=PARSUM+CF(I)
20    CONTINUE
      P=LSUM+1
      T=1D0/P
      A=PARSUM
      DO 30 K=1,M
      G(K)=CF(LSUM+K-1)
30    CONTINUE
      DO 50 I=2,M
      DO 40 J=M,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 CF(I)
```

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CF=1D0/(I+1D0)**1.5D0+1D0/(I+1D0)**2
RETURN
END
```

REFERENCES

[1] G. A. Baker, Jr. and P. R. Graves-Morris, *Padé Approximants, Part* I: *Basic Theory*, Vol. 13 in Encyclopedia of Mathematics and its Applications, G. C. Rotta, ed., Addison-Wesley, Reading, MA, 1981.

[2] C. Brezinski, *A general extrapolation algorithm*, Numer. Math., 35 (1980), pp. 175-187.

[3] T. Fessler, W. F. Ford and D. A. Smith, HURRY: *An acceleration algorithm for scalar sequences and series*, ACM Trans. Math. Software, 9 (1983), pp. 346-354.

[4] T. Håvie, *Generalized Neville type extrapolation schemes*, BIT, 19 (1979), pp. 204-213.

[5] D. Levin, *Development of non-linear transformations for improving convergence of sequences*, Internat. J. Comput. Math., B3 (1973), pp. 371-388.

[6] D. Levin and A. Sidi, *Two new classes of nonlinear transformations for accelerating the convergence of infinite integrals and series*, Appl. Math. Comp., 9 (1981), pp. 175-215.

[7] A. Sidi, *Convergence properties of some nonlinear sequence transformations*, Math. Comp., 33 (1979), pp. 315-326.

[8] ———, *Some properties of a generalization of the Richardson extrapolation process*, J. Inst. Math. Appl., 24 (1979), pp. 327-346.

[9] ———, *Analysis of convergence of the* T-*transformation for power series*, Math. Comp., 35 (1980), pp. 833-850.

[10] ———, *Extrapolation methods for oscillatory infinite integrals*, J. Inst. Math. Appl., 26 (1980), pp. 1-20.

[11] ———, *The numerical evaluation of very oscillatory infinite integrals by extrapolation*, Math. Comp., 38 (1982), pp. 517-529.

[12] ———, *An algorithm for a special case of a generalization of the Richardson extrapolation process*, Numer. Math., 38 (1982), pp. 299-307.

[13] A. Sidi and D. Levin, *Rational approximations from the d-transformation*, IMA J. Numer. Anal., 2 (1982), pp. 153-167.