A New Algorithm for the Higher-Order $G$-Transformation

Avram Sidi
Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel

e-mail: asidi@cs.technion.ac.il

December 2000
Abstract

Let the scalars $A^{(j)}_n$ be defined via the linear equations

$$A_l = A^{(j)}_n + \sum_{k=1}^n \bar{\alpha}_k u_{k+l-1}, \quad l = j, j+1, \ldots, j+n.$$  

Here the $A_i$ and $u_i$ are known and the $\bar{\alpha}_k$ are additional unknowns, and the quantities of interest are the $A^{(j)}_n$. This problem arises, for example, when one computes infinite-range integrals by the higher-order $G$-transformation of Gray, Atchison, and McWilliams. One efficient procedure for computing the $A^{(j)}_n$ is the rs-algorithm of Pye and Atchison. In the present work, we develop yet another procedure that combines the FS-algorithm of Ford and Sidi and the qd-algorithm of Rutishauser, and we denote it the FS/qd-algorithm. We show that the FS/qd-algorithm has a smaller operation count than the rs-algorithm. We also show that the FS/qd algorithm can also be used to implement the transformation of Shanks, and compares very favorably with the $\varepsilon$-algorithm of Wynn that is normally used for this purpose.

Mathematics Subject Classification 2000 : 65B05, 65B10, 65D30, 41A21.
1 The Higher Order $G$-Transformation

The $G$-transformation was designed by Gray and Atchison [5] as an extrapolation method for evaluating infinite integrals of the form $\int_a^\infty f(t) \, dt \equiv I[f]$. It was later generalized in different ways in Atchison and Gray [1] and Gray and Atchison [6], the ultimate generalization being given in Gray, Atchison, and McWilliams [7]. This generalization was denoted the higher-order $G$-transformation. The way it is defined in [7], this transformation produces approximations to $I[f]$ that are of the form

$$G_n(x; h) = \begin{vmatrix}
F(x) & F(x+h) & \cdots & F(x+nh) \\
f(x) & f(x+h) & \cdots & f(x+nh) \\
\vdots & \vdots & & \vdots \\
f(x+(n-1)h) & f(x+nh) & \cdots & f(x+(2n-1)h)
\end{vmatrix}, \quad (1)$$

where

$$F(x) = \int_a^x f(t) \, dt. \quad (2)$$

The approximations produced by the $G$-transformation of [5] are simply the $G_1(x; h)$.

It follows from (1) that $G_n(x; h)$ is also the solution of the linear system

$$F(x + ih) = G_n(x; h) + \sum_{k=1}^n \alpha_k f(x + (i + k - 1)h), \quad i = 0, 1, \ldots, n, \quad (3)$$

where $\alpha_k$ are additional unknowns.

It has been shown in [7] that the kernel of the higher-order $G$-transformation is the set of functions $f(x)$ that are integrable at infinity in the sense of Abel and that satisfy linear homogeneous ordinary differential equations of order $n$ with constant coefficients. Thus $f(x)$ is in this kernel if it is of the form $f(x) = \sum_{k=1}^n p_k(x) e^{c_k x}$, where the $c_k \neq 0$ are distinct and $\Re c_k \leq 0$, $p_k(x)$ are polynomials. If $\mu_k$ is the degree of $p_k(x)$ for each $k$, and if $\sum_{k=1}^n (\mu_k + 1) = n$, then $G_n(x; h) = I[f]$ for all $x$ and $h$. On the basis of this result it was concluded in Levin and Sidi [11] that the higher-order
$G$-transformation will be effective on functions of the form $f(x) = \sum_{k=1}^{s} e^{c_k x} u_k(x)$, where $u_k(x) \sim \sum_{i=0}^{\infty} \alpha_{ki} x^{\gamma_k - i}$ as $x \to \infty$, with arbitrary $\gamma_k$, provided $h$ is of a suitable size.

In the present work, we are concerned with the actual computation of the $G_n(x; h)$. Of course, it is not desirable to compute $G_n(x; h)$ via the determinantal representation in (1). Direct solution of the linear systems in (3) is expensive too. A very efficient and elegant procedure for computing the $G_n(x; h)$ was given by Pye and Atchison in [12], and it has been denoted the rs-algorithm in Brezinski and Redivo Zaglia [3]. The derivation of the rs-algorithm makes use of the representation in (1). In the present work, we develop yet another procedure for computing the $G_n(x; h)$, and we call this new procedure the FS/qd-algorithm. We show that the FS/qd-algorithm is more efficient than the rs-algorithm. With proper substitutions to be discussed in Section 3 the FS/qd-algorithm can also be used to implement the transformation of Shanks [15], which is normally implemented via the well-known $\varepsilon$-algorithm of Wynn [17]. We show that, when used for implementing the transformation of Shanks, the FS/qd-algorithm compares very favorably with the $\varepsilon$-algorithm.

2 Algorithms for the Higher Order $G$-Transformation

We start with a review of the algorithm of Pye and Atchison [12]. Actually these authors consider the more general problem in which one would like to compute the quantities $A_{n}^{(j)}$ defined via the linear equations

$$A_l = A_{n}^{(j)} + \sum_{k=1}^{n} \bar{\alpha}_k u_{k+l-1}, \quad l = j, j+1, \ldots, j+n, \quad (4)$$

where the $A_i$ and $u_i$ are known scalars, while the $\bar{\alpha}_k$ are not necessarily known. Comparing (3) with (4) we can draw the analogy $A_i \leftrightarrow F(x + ih)$, $u_i \leftrightarrow f(x + ih)$, and $A_{n}^{(j)} \leftrightarrow G_n(x + jh; h)$.

The algorithm of [12] computes the $A_{n}^{(j)}$ with the help of two sets of auxiliary quantities, $r_{n}^{(j)}$ and $s_{n}^{(j)}$. These quantities are defined by

$$r_{n}^{(j)} = \frac{H_{n}^{(j)}}{K_{n}^{(j)}}, \quad s_{n}^{(j)} = \frac{K_{n+1}^{(j)}}{H_{n}^{(j)}}, \quad (5)$$

2
where $H_n^{(j)}$, the Hankel determinant associated with $\{u_s\}$, and $K_n^{(j)}$ are given as in

$$H_n^{(j)} = \begin{vmatrix} u_j & u_{j+1} & \cdots & u_{j+n-1} \\ u_{j+1} & u_{j+2} & \cdots & u_{j+n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{j+n-1} & u_{j+n} & \cdots & u_{j+2n-2} \end{vmatrix}, \quad H_0^{(j)} = 1, \quad (6)$$

and

$$K_n^{(j)} = \begin{vmatrix} 1 & 1 & \cdots & 1 \\ u_j & u_{j+1} & \cdots & u_{j+n-1} \\ u_{j+1} & u_{j+2} & \cdots & u_{j+n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{j+n-2} & u_{j+n-1} & \cdots & u_{j+2n-3} \end{vmatrix}, \quad K_0^{(j)} = 1. \quad (7)$$

The rs-algorithm computes the $r_n^{(j)}$, and $s_n^{(j)}$ simultaneously by efficient recursions. Once these have been computed the $A_n^{(j)}$ can be computed via a separate recursion.

**The rs-Algorithm**

1. Set
   $$s_0^{(j)} = 1, \quad r_1^{(j)} = u_j, \quad A_0^{(j)} = A_j, \quad j = 0, 1, \ldots$$

2. For $j = 0, 1, \ldots$, and $n = 1, 2, \ldots$, compute recursively
   $$s_n^{(j)} = s_{n-1}^{(j+1)} \left( \frac{r_n^{(j+1)}}{r_n^{(j)}} - 1 \right), \quad r_{n+1}^{(j)} = r_n^{(j+1)} \left( \frac{s_n^{(j+1)}}{s_n^{(j)}} - 1 \right).$$

3. For $j = 0, 1, \ldots$, and $n = 1, 2, \ldots$, set
   $$A_n^{(j)} = \frac{r_n^{(j)} A_{n-1}^{(j+1)} - r_{n-1}^{(j+1)} A_n^{(j)}}{r_n^{(j)} - r_{n-1}^{(j+1)}}.$$
Schneider [14] and the FS-algorithm of Ford and Sidi [4] can be used for computing the $A_n^{(j)}$. Different derivations of the E-algorithm were given by Håvie [8] and Brezinski [2]. Of course, direct application of these algorithms without taking into account the special nature of the $g_k(l)$ is very uneconomical. By taking the nature of the $g_k(l)$ into consideration, it becomes possible to derive fast algorithms for the $A_n^{(j)}$.

Now the E-algorithm produces the $A_n^{(j)}$ by a recursion relation of the form

$$A_n^{(j)} = \frac{R_n^{(j)} A_{n-1}^{(j+1)} - R_{n+1}^{(j+1)} A_{n-1}^{(j)}}{R_n^{(j)} - R_{n+1}^{(j+1)}},$$

the most expensive part of the algorithm being the determination of the $R_n^{(j)}$. Comparing the known expression for $R_n^{(j)}$ when $g_k(l) = u_{k+l-1}$ with that of $r_n^{(j)}$, we realize that $R_n^{(j)} = (-1)^n r_n^{(j)}$. We thus conclude that the rs-algorithm is simply the E-algorithm in which the $R_n^{(j)}$, whose determination forms the most expensive part of the E-algorithm, are computed by a fast recursion. For this point and others, see [3, Section 2.4].

In view of the close connection between the rs- and E-algorithms, it is natural to investigate the possibility of designing another algorithm that is related to the FS-algorithm. This is worth the effort as the FS-algorithm is more economical than the E-algorithm to begin with. To this end we start with a brief description of the FS-algorithm and refer the reader to [4] for details. For a comprehensive summary, see also Sidi [16].

Let us first define the short-hand notation

$$\left| u_1(j) u_2(j) \cdots u_n(j) \right| = \begin{vmatrix} u_1(j) & u_2(j) & \cdots & u_n(j) \\ u_1(j+1) & u_2(j+1) & \cdots & u_n(j+1) \\ \vdots & \vdots & & \vdots \\ u_1(j+n-1) & u_2(j+n-1) & \cdots & u_n(j+n-1) \end{vmatrix}, \quad (9)$$

and set

$$G_n^{(j)} = \left| g_1(j) g_2(j) \cdots g_n(j) \right|, \quad G_0^{(j)} = 1. \quad (10)$$

Next, let us agree to denote the sequence $\{b(l)\}_{l=0}^\infty$ by $b$ for short, and define

$$f_n^{(j)}(b) = \left| g_1(j) g_2(j) \cdots g_n(j) b(j) \right|, \quad f_0^{(j)}(b) = b(j). \quad (11)$$
Finally, let us define

\[ \psi_n^{(j)}(b) = \frac{f_n^{(j)}(b)}{G_{n+1}^{(j)}}. \]  

(12)

Then we have

\[ A_n^{(j)} = \frac{\psi_n^{(j)}(a)}{\psi_n^{(j)}(I)}, \]  

(13)

where \( a \) and \( I \) denote the sequences \( \{a(l) = A_l\}_{l=0}^\infty \) and \( \{I(l) = 1\}_{l=0}^\infty \) respectively.

The FS-algorithm computes the quantities \( \psi_n^{(j)}(b) \) by a recursion of the form

\[ \psi_n^{(j)}(b) = \frac{\psi_{n-1}^{(j+1)}(b) - \psi_{n-1}^{(j)}(b)}{D_n^{(j)}}, \quad D_n^{(j)} = \frac{G_{n+1}^{(j)}G_{n-1}^{(j+1)}}{G_n^{(j)}G_n^{(j+1)}}. \]  

(14)

By this recursion, first the \( \psi_n^{(j)}(a) \) and \( \psi_n^{(j)}(I) \) are computed and then \( A_n^{(j)} \) is determined via (13).

We recall that the most expensive part of the FS-algorithm is the (recursive) determination of the quantities \( D_n^{(j)} \), and we would like to reduce the cost of this part. Fortunately, this can be achieved once we realize that, with \( G_n^{(j)} \) and \( H_n^{(j)} \) defined as in (10) and (6) respectively, and with \( g_k(l) = u_{k+l} \), we have \( G_n^{(j)} = H_n^{(j)} \) in the present case. From this and (14), we obtain the surprising result that

\[ D_n^{(j)} = \frac{H_{n+1}^{(j)}H_{n-1}^{(j+1)}}{H_n^{(j)}H_{n+1}^{(j+1)}} = e_n^{(j)}. \]  

(15)

Here \( e_n^{(j)} \) is a quantity computed by the famous qd-algorithm of Rutishauser [13], a clear exposition of which can also be found in Henrici [9]. Actually, the qd-algorithm computes along with the \( e_n^{(j)} \) also the quantities \( q_n^{(j)} \) that are given as in

\[ q_n^{(j)} = \frac{H_{n-1}^{(j)}H_n^{(j+1)}}{H_n^{(j)}H_{n+1}^{(j+1)}}. \]  

(16)

The \( q_n^{(j)} \) and \( e_n^{(j)} \) serve the construction of the regular \( C \)-fractions, hence the Padé approximants, associated with the formal power series \( \sum_{k=0}^{\infty} u_k z^k \). Regular \( C \)-fractions are continued fractions of a special type. (See Jones and Thron [10].) The qd-algorithm computes the \( q_n^{(j)} \) and \( e_n^{(j)} \) via the recursions

\[ e_n^{(j)} = q_n^{(j+1)} - q_n^{(j)} + e_{n-1}^{(j+1)}, \quad q_n^{(j+1)} = \frac{e_n^{(j+1)}}{e_n^{(j)} q_n^{(j+1)}}, \quad j \geq 0, \quad n \geq 1. \]  

(17)
Figure 1: The qd-table.

with the initial conditions $e_{0}^{(j)} = 0$ and $q_{1}^{(j)} = u_{j+1}/u_{j}$ for all $j \geq 0$. The quantities $q_{n}^{(k)}$ and $e_{n}^{(k)}$ can be arranged in a two-dimensional array as in Figure 1.

This observation enables us to combine the FS- and qd-algorithms to obtain the following economical implementation, the FS/qd-algorithm, for the higher-order $G$-transformation. For simplicity of notation, we will let $\psi_{n}^{(j)}(a) = M_{n}^{(j)}$ and $\psi_{n}^{(j)}(I) = N_{n}^{(j)}$ in the FS-algorithm.

The FS/qd-Algorithm

1. Set

   $e_{0}^{(j)} = 0, \quad q_{1}^{(j)} = \frac{u_{j+1}}{u_{j}}, \quad M_{0}^{(j)} = \frac{A_{j}}{u_{j}}, \quad N_{0}^{(j)} = \frac{1}{u_{j}}.$

2. For $j = 0, 1, \ldots$, and $n = 1, 2, \ldots$, compute recursively

   $e_{n}^{(j)} = q_{n}^{(j+1)} - q_{n}^{(j)} + e_{n-1}^{(j+1)}$, \quad $q_{n+1}^{(j)} = \frac{e_{n}^{(j+1)}}{e_{n}^{(j)}} q_{n}^{(j+1)}$,

   $M_{n}^{(j)} = \frac{M_{n-1}^{(j+1)} - M_{n-1}^{(j)}}{e_{n}^{(j)}}, \quad N_{n}^{(j)} = \frac{N_{n-1}^{(j+1)} - N_{n-1}^{(j)}}{e_{n}^{(j)}}$. 
3. For \( j = 0, 1, \ldots, \) and \( n = 1, 2, \ldots, \) set

\[
A_n^{(j)} = \frac{M_n^{(j)}}{N_n^{(j)}}.
\]

It seems that, when a certain number of the \( u_i \) are given, it is best to compute the associated qd-table columnwise. (Note that the quantities in each of the recursions for \( e_n^{(j)} \) and \( q_{n+1}^{(j)} \) in step 2 of the FS/qd-algorithm above form the four corners of a lozenge in Figure 1.) Following that we can compute the \( M_n^{(j)}, N_n^{(j)}, \) and \( A_n^{(j)} \) columnwise as well.

### 3 Comparison of the rs- and FS/qd-Algorithms

Let us now compare the operation counts of the two algorithms. First, we note that the \( r_n^{(j)} \) and \( s_n^{(j)} \) in the rs-algorithm can be arranged in a table similar to the qd-table of the \( e_n^{(j)} \) and \( q_n^{(j)} \). Thus, given \( A_0, A_1, \ldots, A_L, \) and \( u_0, u_1, \ldots, u_{2L}, \) we can compute \( A_n^{(j)} \) for \( 0 \leq j + n \leq L \). Now the number of the \( e_n^{(j)} \) in the relevant qd-table is \( L^2 + O(L) \) and so is that of the \( q_n^{(j)} \). A similar statement can be made concerning the \( r_n^{(j)} \) and \( s_n^{(j)} \). The number of the \( A_n^{(j)} \) is \( L^2/2 + O(L) \), and so are the numbers of the \( M_n^{(j)} \) and the \( N_n^{(j)} \). Consequently, we have the following operation counts.

| Algorithm | No. of Multiplications | No. of Additions | No. of Divisions |
|-----------|------------------------|------------------|-----------------|
| FS/qd     | \( L^2 + O(L) \)       | \( 3L^2 + O(L) \) | \( 5L^2/2 + O(L) \) |
| rs        | \( 3L^2 + O(L) \)      | \( 3L^2 + O(L) \) | \( 5L^2/2 + O(L) \) |

In case only the \( A_n^{(0)} \) are needed (as they have the best convergence properties), the number of divisions in the FS/qd-algorithm can be reduced from \( 5L^2/2 + O(L) \) to \( 2L^2 + O(L) \). In any case, we see that the operation count of the rs-algorithm is about 30\% more than that of the FS/qd-algorithm.

Finally, we observe that when \( u_k = \Delta A_k = A_{k+1} - A_k \) the higher-order \( G \)-transformation reduces to the Shanks [15] transformation, and therefore the rs- and FS/qd-algorithms can be used for computing the \( A_n^{(j)} \) in this case too. Of course, the most famous and efficient implementation of the Shanks transformation is via the
\( \varepsilon \)-algorithm of Wynn [17], which reads

\[
\varepsilon_{k+1}^{(j)} = \varepsilon_{k-1}^{(j+1)} + \frac{1}{\varepsilon_{k+1}^{(j+1)} - \varepsilon_{k}^{(j)}}, \quad j, k = 0, 1, \ldots, \tag{18}
\]

with the initial values \( \varepsilon_{-1}^{(j)} = 0 \) and \( \varepsilon_{0}^{(j)} = A_j \) for all \( j \geq 0 \). Then \( \varepsilon_{2n}^{(j)} = A_{n}^{(j)} \) for all \( j \) and \( n \). Thus, given \( A_0, A_1, \ldots, A_{2L} \), we can compute \( \varepsilon_{2n}^{(j)} \) for \( 0 \leq j + 2n \leq 2L \). (In particular, we can compute the diagonal approximations \( \varepsilon_{2n}^{(j)} \), \( n = 0, 1, \ldots, L \), that have the best convergence properties.) Since there are \( 2L^2 + O(L) \) of the \( \varepsilon_{k}^{(j)} \) to compute, the operation count of this computation is \( 4L^2 + O(L) \) additions and \( 2L^2 + O(L) \) divisions and no multiplications. It is seen from the table above that the FS/qd-algorithm compares very favorably with the \( \varepsilon \)-algorithm as well.

References

[1] T.A. Atchison and H.L. Gray. Nonlinear transformations related to the evaluation of improper integrals II. *SIAM J. Numer. Anal.*, 5:451–459, 1968.

[2] C. Brezinski. A general extrapolation algorithm. *Numer. Math.*, 35:175–187, 1980.

[3] C. Brezinski and M. Redivo Zaglia. *Extrapolation Methods: Theory and Practice*. North-Holland, Amsterdam, 1991.

[4] W.F. Ford and A. Sidi. An algorithm for a generalization of the Richardson extrapolation process. *SIAM J. Numer. Anal.*, 24:1212–1232, 1987.

[5] H.L. Gray and T.A. Atchison. Nonlinear transformations related to the evaluation of improper integrals I. *SIAM J. Numer. Anal.*, 4:363–371, 1967.

[6] H.L. Gray and T.A. Atchison. The generalized G-transform. *Math. Comp.*, 22:595–605, 1968.

[7] H.L. Gray, T.A. Atchison, and G.V. McWilliams. Higher order G-transformations. *SIAM J. Numer. Anal.*, 8:365–381, 1971.
[8] T. Håvlie. Generalized Neville type extrapolation schemes. *BIT*, 19:204–213, 1979.

[9] P. Henrici. *Applied and Computational Complex Analysis*, volume 1. Wiley, New York, 1974.

[10] W.B. Jones and W.J. Thron. *Continued Fractions: Analytic Theory and Applications*. Addison-Wesley, London, 1980.

[11] D. Levin and A. Sidi. Two new classes of nonlinear transformations for accelerating the convergence of infinite integrals and series. *Appl. Math. Comp.*, 9:175–215, 1981. Originally appeared as a Tel Aviv University preprint in 1975.

[12] W.C. Pye and T.A. Atchison. An algorithm for the computation of the higher order G-transformation. *SIAM J. Numer. Anal.*, 10:1–7, 1973.

[13] H. Rutishauser. Der Quotienten-Differenzen-Algorithmus. *Z. Angew. Math. Phys.*, 5:233–251, 1954.

[14] C. Schneider. Vereinfachte Rekursionen zur Richardson-Extrapolation in Spezialfällen. *Numer. Math.*, 24:177–184, 1975.

[15] D. Shanks. Nonlinear transformations of divergent and slowly convergent sequences. *J. Math. and Phys.*, 34:1–42, 1955.

[16] A. Sidi. On a generalization of the Richardson extrapolation process. *Numer. Math.*, 57:365–377, 1990.

[17] P. Wynn. On a device for computing the $e_m(S_n)$ transformation. *Mathematical Tables and Other Aids to Computation*, 10:91–96, 1956.