A METHOD FOR COMPUTING FUNCTIONS OF A MATRIX BASED ON NEWTON'S DIVIDED DIFFERENCE FORMULA

by

A.C. McCurdy

Memorandum No. UCB/ERL M78/69

October 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Method for Computing Functions of a Matrix Based on Newton's Divided Difference Formula

Contents

Acknowledgment

The author is most grateful to Prof. Beresford Parlett for his comments and help in getting this report written.

Research sponsored by the Office of Naval Research Grant N00014-76-C-0013.

A Method for Computing Functions of a Matrix Based on Newton's Divided Difference Formula

A function of a constant square matrix A, written $f(A)$, may be defined as a polynomial in A. One particular polynomial representation for $f(A)$ is based on Newton's divided difference formula for interpolating polynomials. The coefficients of the polynomial are divided differences of f at the eigenvalues of A. Several methods for computing divided differences will be investigated, and it will also be shown that a table of divided differences is, itself, a function of a very special matrix. Finally, some details in using the Newton polynomial approach for computing $f(A)$ will be discussed in order to point out possible ways to make the computation more efficient.

1. Definitions and Representations of Matrix Functions

1.1 Definition of f(A).

Let A be an $n+1 \times n+1$ complex matrix and $\Lambda_A \equiv \{\lambda_0, \ldots, \lambda_0, \lambda_1, \ldots, \lambda_1, \ldots, \lambda_l, \ldots, \lambda_l\}$ be the set of eigenvalues of A, $l+1$ of which are distinct. Each distinct eigenvalue occurs n_i+1 times, $i=0, 1, \ldots, l$, and Λ_A has $\sum_{i=0}^{l}(n_i+1) = n+1$ elements. The elements of Λ_A are the roots of the characteristic polynomial of A,

$$\chi_A(\lambda) = (\lambda-\lambda_0)^{n_0+1}(\lambda-\lambda_1)^{n_1+1} \cdots (\lambda-\lambda_l)^{n_l+1}. \tag{1.1.1}$$

When the eigenvalues of A are distinct, the definition of $f(A)$ simply requires that $f(\lambda)$ be defined for each $\lambda \in \Lambda_A$. However for greater generality, we must be a bit more careful in order to allow for multiple eigenvalues. With this in mind, f will be required to satisfy the following.

Definition: The function f is said to be "defined on the characteristic values of A" when $f(\lambda_i)$, $f'(\lambda_i)$,..., $f^{(n_i)}(\lambda_i)$ for $i=0, 1, \ldots, l$ are all defined and finite. This set of values is denoted, for brevity, by $f(\Lambda_A)$.

With f satisfying the above, $f(A)$ may be defined as a polynomial in the matrix A.

Definition: Function of a matrix. If f is defined on the characteristic values of A and p is any polynomial such that

$$p(\Lambda_A) = f(\Lambda_A)$$

then

$$f(A) \equiv p(A).$$

The polynomial p is an osculating interpolation polynomial for f on Λ_A. That is, $p(\lambda_i)=f(\lambda_i)$, $p'(\lambda_i)=f'(\lambda_i)$,..., $p^{(n_i)}(\lambda_i)=f^{(n_i)}(\lambda_i)$ for $i=0, 1, \ldots, l$. When the eigenvalues are distinct, the definition of $f(A)$ becomes particularly simple, as then p is just the ordinary

interpolation polynomial for f at the elements of Λ_A. Some authors choose to define $f(A)$ in terms of a power series in A and A's Jordan canonical form [3,18]; however, the above definition is most natural for the purposes of this paper.

The rationale behind the definition of $f(A)$ is that for any two functions f and g, $f(A)$ is indistinguishable from $g(A)$ when $f(\Lambda_A) = g(\Lambda_A)$. The set of zeros of $f(\lambda) - g(\lambda)$ includes the roots of $\chi_A(\lambda)$, and $\chi_A(A) = 0$ by the Cayley-Hamilton theorem. The interpolating polynomial p has degree at least n, as it must satisfy the $n+1$ conditions given in the definition.† Interpolating polynomials p may be chosen to satisfy additional conditions; however, the degree of the polynomial will be increased.

The characteristic polynomial χ_A is an annihilating polynomial for A because $\chi_A(A) = 0$. However, for some matrices A there are polynomials of smaller degree which are also annihilating polynomials. The minimal polynomial μ_A is the non-trivial annihilating polynomial of least degree. If $\mu_A(\lambda)$ has degree $m+1$, then $m \leqslant n$. It is possible to define $f(A)$ in terms of a polynomial p_m of degree m which interpolates f at the $m+1$ roots of the minimal polynomial μ_A. Gantmacher [4] uses this slightly more general approach in his definition of $f(A)$. The set of roots of $\mu_A(\lambda)$ is contained in the set Λ_A of eigenvalues of A. For $m < n$, fewer derivatives of f need be specified; however, μ_A and the multiplicities of its roots are generally difficult and costly to obtain. Thus, we shall not try to form $f(A) = p_m(A)$ for the smallest possible degree m, but shall consider other polynomials. This may lead to polynomials of significantly higher degree than p_m in a few cases, see Fig. 1.2.1, but it also leads to greater simplicity in that less need be known in advance about the matrix A.

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 0 \\ -1 & 1 & 3 \end{bmatrix} \qquad \begin{aligned} \mu_A(\lambda) &= (\lambda-1)(\lambda-2) \\ \chi_A(\lambda) &= (\lambda-1)^2(\lambda-2) \end{aligned}$$

Fig. 1.1.1: Degree of μ_A may be less than degree of χ_A.

That $f(A)$ is representable as a polynomial in A leads to two elementary, but very important, consequences.

(1). For any $n+1 \times n+1$ nonsingular matrix P,

$$f(PAP^{-1}) = P \cdot f(A) \cdot P^{-1}. \tag{1.1.2}$$

In theory, this permits all work to be performed on the simplest matrix similar to A, e.g. the Jordan canonical form. In practice, P may be difficult to compute accurately‡ and some less simple form may be required.

(2). Commutivity.

$$A \cdot f(A) = f(A) \cdot A \tag{1.1.3}$$

Parlett [15] has developed a fast method for computing $f(A)$ based on this property (§4.1).

†A polynomial of degree k can interpolate at, at most, $k+1$ points. In general $k+1$ points uniquely determine a polynomial of degree k through the points; higher degree polynomials are not uniquely determined.
‡Kagstrm [10] gives a method for computing the Jordan form. Golub and Wilkinson [6] discuss limitations in computing the Jordan canonical form.

1.2 Other representations of f(A), the Newton polynomial.

Additional representations of $f(A)$ are obtained from other ways of writing interpolating polynomials p which satisfy $p(\Lambda_A)=f(\Lambda_A)$. One such p is the "Newton polynomial" p_n which is derived directly from Newton's divided difference formula for the interpolating polynomial [12], namely†

$$p_n(\lambda) = \sum_{k=0}^{n} \Delta_0^k f \prod_{j=0}^{k-1} (\lambda - \lambda_j) . \tag{1.2.1}$$

The first few terms of this polynomial are

$$f(\lambda_0) + \Delta_0^1 f \cdot (\lambda - \lambda_0) + \Delta_0^2 f \cdot (\lambda - \lambda_0)(\lambda - \lambda_1) + \Delta_0^3 f \cdot (\lambda - \lambda_0)(\lambda - \lambda_1)(\lambda - \lambda_2) + \cdots .$$

This polynomial satisfies $p_n(\Lambda_A)=f(\Lambda_A)$ where $\Lambda_A=\{\lambda_0, \lambda_1, \ldots, \lambda_n\}$, the eigenvalues having been renumbered.

Newton polynomial for f(A). If f is defined on the characteristic values of A, then taking Λ_A as the set of abscissae for the divided differences,

$$f(A) = \sum_{k=0}^{n} \Delta_0^k f \prod_{j=0}^{k-1} (A - \lambda_j I) . \tag{1.2.2}$$

$$A = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 1 & & & \\ & 1 & 1 & & \\ & & 1 & 1 & \\ & & & 1 & \\ & & & & 2 \end{bmatrix}$$

$$\mu_A(\lambda) = (\lambda-1)(\lambda-2) \qquad \mu_B(\lambda) = (\lambda-1)^4(\lambda-2)$$

$$\chi_A(\lambda) = (\lambda-1)^4(\lambda-2) \qquad \chi_B(\lambda) = \mu_B(\lambda)$$

$$p_m(A) = f(1)I + [f(2)-f(1)](A-I) \qquad p_m(B) = \sum_{k=0}^{3} \frac{f^{(k)}(1)}{k!}(B-I)^k + \Delta_0^4 f\,(B-I)^4$$

$$p_n(A) = \sum_{k=0}^{3} \frac{f^{(k)}(1)}{k!}(A-I)^k + \Delta_0^4 f\,(A-I)^4 \qquad p_n(B) = p_m(B)$$

Fig. 1.2.1: p_m depends on the eigenspaces of the matrix.

Series representations of $f(A)$ are also possible. Following Gantmacher [4], when $\sum_{k=0}^{\infty} u_k(\lambda)$ is defined (converges) on the characteristic values of A to $f(\lambda)$, that is $f(\lambda_i) = \sum_{k=0}^{\infty} u_k(\lambda_i), \ldots, f^{(n_i)}(\lambda_i) = \sum_{k=0}^{\infty} u_k^{(n_i)}(\lambda_i)$ for $i=0, 1, \ldots, l$, denoted $f(\Lambda_A) = \sum_{k=0}^{\infty} u_k(\Lambda_A)$,

$$f(A) \equiv \sum_{k=0}^{\infty} u_k(A) . \tag{1.2.3}$$

In particular, if f can be expanded in a power series about $\lambda = a$

$$f(\lambda) = \sum_{k=0}^{\infty} \gamma_k (\lambda - a)^k \tag{1.2.4}$$

†see §2.1 for details on divided differences and divided difference notation.

with circle of convergence $|\lambda - a| < R$, then when $\Lambda_A \subset \{\lambda \mid R > |\lambda - a|\}$

$$f(A) = \sum_{k=0}^{\infty} \gamma_k (A - aI)^k. \tag{1.2.5}$$

Within the circle of convergence $\gamma_k = f^{(k)}(a)/k!$, the Taylor coefficients.

Taylor series representation for f(A). When f is representable by a Taylor series about a on a domain Ω and $\Lambda_A \subset \Omega$, then

$$f(A) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!} (A - aI)^k. \tag{1.2.6}$$

One other seldom seen representation derives from Newton's divided difference series [5]

$$f(\lambda) = \sum_{k=0}^{\infty} \Delta_0^k f \prod_{j=0}^{k-1} (\lambda - \mu_j) \tag{1.2.7}$$

where the set of abscissae for the divided differences $\mathbf{M} \equiv \{\mu_0, \mu_1, \ldots\}$ is contained in the domain of convergence of the series.

Newton series representation for f(A). When f is representable by a Newton divided difference series about $\mathbf{M}$ on a domain Ω and $\Lambda_A \subset \Omega$, then

$$f(A) = \sum_{k=0}^{\infty} \Delta_0^k f \prod_{j=0}^{k-1} (A - \mu_j I). \tag{1.2.8}$$

If the first $n+1$ elements of $\mathbf{M}$ are the set Λ_A, i.e. $\mu_0 = \lambda_0, \ldots, \mu_n = \lambda_n$, the Newton series (1.2.8) reduces to the Newton polynomial (1.2.2) because $\prod_{j=0}^{n} (A - \lambda_j I) = 0$ by the Cayley-Hamilton theorem.

For some of the elementary functions, such as exp, sin, cosh, the Taylor series and the Newton polynomial representations of $f(A)$ recommend themselves for numerical computation. The Taylor coefficients are easy to compute and are independent of A. Unfortunately, the series may converge slowly for some A. The Newton polynomial is exact after only $n+1$ terms; so convergence is no problem. This great savings in work is diminished by the need to compute the eigenvalues of A. However, if the dimension of A is not large, say $n \leqslant 100$, efficient and accurate algorithms now exist for finding all the eigenvalues [17]. Until recently the eigenvalue problem was thought to be very difficult, but now the need to find the eigenvalues of a "small" matrix is no drawback for a method. The difficulty in employing the Newton polynomial lies in computing its coefficients (i.e. the divided differences). Often, divided differences of f can be obtained quickly and accurately. When this is the case, the Newton polynomial becomes a practical method for computing functions of a matrix. The next section will study the divided difference problem.

2. Divided Differences

2.1 Basic definitions and properties of divided differences.

Divided differences were studied extensively in classical precomputer numerical analysis as part of the finite difference calculus. They saw great use in the tabulation of tables of function values. This subject, along with the general topic of interpolation, is treated in Milne-Thomson [12].

Quite different purposes are envisioned here; however, much of the classical theory is still relevant. Before proceeding to develop formulas for the calculation of divided differences, a few well-known definitions and consequences will be presented. The notation used will be somewhat different from that of other authors, but it is felt to be an improvement. Once understood, it should cause no confusion to those already familiar with divided differences.

Most common notations for divided differences are cumbersome. For clarity we shall begin with such a notation, but later reduce it to a more compact form by suppressing unneeded information. Let f be a function of a single variable ζ and let $f(\zeta)$ be defined, at least, on the set $\mathbf{Z}=\{\zeta_0,\zeta_1,\ldots,\zeta_n\}$ of distinct complex numbers. $\mathbf{Z}$ is called the set of abscissae, or sometimes the set of data points or the set of nodes. The 0-th divided difference of f at ζ_0 is defined as

$$(\Delta^0 f)(\zeta_0) \equiv f(\zeta_0). \tag{2.1.1}$$

The first divided difference of f at ζ_0 is a function of the two variables (abscissae) ζ_0 and ζ_1 and is formed from the 0-th divided difference by the familiar formula

$$(\Delta^1 f)(\zeta_0,\zeta_1) \equiv \frac{(\Delta^0 f)(\zeta_1)-(\Delta^0 f)(\zeta_0)}{\zeta_1-\zeta_0} = \frac{f(\zeta_1)-f(\zeta_0)}{\zeta_1-\zeta_0}. \tag{2.1.2}$$

The k-th order divided difference of f at ζ_0 is a function of the $k+1$ abscissae $\zeta_0,\zeta_1,\ldots,\zeta_k$ and is defined recursively from $k-1$-st order divided differences which are functions of k abscissae.

First recursive definition of divided differences. When f is defined on $\mathbf{Z}$, the k-th order, $0<k\leqslant n$, divided differences of f at ζ_j for $j=0,1,\ldots,n-k$ are

$$(\Delta^k f)(\zeta_j,\zeta_{j+1},\ldots,\zeta_{j+k}) \equiv \frac{(\Delta^{k-1} f)(\zeta_{j+1},\ldots,\zeta_{j+k})-(\Delta^{k-1} f)(\zeta_j,\ldots,\zeta_{j+k-1})}{\zeta_{j+k}-\zeta_j} \tag{2.1.3}$$

$(\Delta^k f)(\zeta_j,\zeta_{j+1},\ldots,\zeta_{j+k})$ has no dependence on abscissae with indices $<j$ or $>j+k$, and so no generality is lost by considering just $(\Delta^n f)(\zeta_0,\zeta_1,\ldots,\zeta_n)$.

Divided differences are very special functions of the data points in $\mathbf{Z}$. Not only does the number of data points used increase with the order of the difference, but the divided difference is symmetric in its arguments. This may be seen by employing an equivalent representation of the divided difference in terms of determinants [12].

$$(\Delta^n f)(\zeta_0,\zeta_1,\ldots,\zeta_n) = \begin{vmatrix} f(\zeta_0) & f(\zeta_1) & \cdot & f(\zeta_n) \\ \zeta_0^{n-1} & \zeta_1^{n-1} & \cdot & \zeta_n^{n-1} \\ \cdot & \cdot & & \cdot \\ 1 & 1 & \cdot & 1 \end{vmatrix} \div \begin{vmatrix} \zeta_0^n & \zeta_1^n & \cdot & \zeta_n^n \\ \zeta_0^{n-1} & \zeta_1^{n-1} & \cdot & \zeta_n^{n-1} \\ \cdot & \cdot & & \cdot \\ 1 & 1 & \cdot & 1 \end{vmatrix} \tag{2.1.4}$$

The abscissae $\mathbf{Z}=\{\zeta_0,\zeta_1,\ldots,\zeta_n\}$ may be arranged in any order without changing the value of $(\Delta^n f)(\zeta_0,\zeta_1,\ldots,\zeta_n)$. This is the very important symmetry property.

Symmetry property. Let π be a permutation on the set of indices $0, 1, \ldots, n$. Then

$$(\Delta^n f)(\zeta_0, \zeta_1, \ldots, \zeta_n) = (\Delta^n f)(\zeta_{\pi(0)}, \zeta_{\pi(1)}, \ldots, \zeta_{\pi(n)}) . \tag{2.1.5}$$

The primary defect in definition (2.1.3) is that the data points must be distinct. However, if f is differentiable (2.1.3) may still be defined even for confluent (i.e. equal) abscissae. In particular, when $\mathbf{Z} = \{\zeta_0, \zeta_0, \ldots, \zeta_0\}$, (2.1.3) is defined if $f^{(n)}(\zeta_0)$ exists. For confluent abscissae the divided difference reduces to

$$(\Delta^n f)(\zeta_0, \zeta_0, \ldots, \zeta_0) = \frac{f^{(n)}(\zeta_0)}{n!} . \tag{2.1.6}$$

Since, in theory, the data points can be arranged in any order without changing the value of the divided difference, it is clear that for suitable f (2.1.3) will always be defined if (2.1.6) is used when confluent abscissae occur. So, the requirement that the abscissae be distinct may be removed.

Definition: Let $\mathbf{Z} \equiv \{\zeta_0, \ldots, \zeta_0, \zeta_1, \ldots, \zeta_1, \ldots, \zeta_l, \ldots, \zeta_l\}$ be the set of abscissae (just a renumbering of the previous $\mathbf{Z}$) where each ζ_i appears n_i+1 times, $\sum_{i=0}^{l}(n_i+1) = n+1$. The function f is said to be "defined on the set of abscissae $\mathbf{Z}$" when $f(\zeta_i)$, $f'(\zeta_i)$,..., $f^{(n_i)}(\zeta_i)$ for $i=0, 1, \ldots, l$ are all defined and finite. This set of values is denoted $f(\mathbf{Z})$.

Before rewriting the definition (2.1.3) in more generality, a more compact notation will be introduced. In the work here, the set of data points $\mathbf{Z} = \{\zeta_0, \zeta_1, \ldots, \zeta_n\}$ is given and, usually, in a fixed order. Hence, reference to $\mathbf{Z}$ may be suppressed. Thus,

$$\Delta_j^k f \equiv (\Delta^k f)(\zeta_j, \zeta_{j+1}, \ldots, \zeta_{j+k}) . \tag{2.1.7}$$

In the event that the set $\mathbf{Z}$ must be emphasized, $\Delta_{\zeta_j}^k f$ will be written for $\Delta_j^k f$.†

Recursive definition of divided differences. When f is defined on the set of abscissae $\mathbf{Z}$, then for $k=1, 2, \ldots, n$

$$\Delta_j^k f \equiv \frac{\Delta_{j+1}^{k-1} f - \Delta_j^{k-1} f}{\zeta_{j+k} - \zeta_j}, \quad j=0, 1, \ldots, n-k , \tag{2.1.8}$$

where $\Delta_j^0 f \equiv f(\zeta_j)$.

This definition of divided differences and the definition of the Newton polynomial in §1.2 are seen to be consistent. Indeed, if $\mathbf{Z} = \Lambda_A$, "defined on the set of abscissae $\mathbf{Z}$" and "defined on the characteristic values of A" are the same.

Divided differences have many useful representations and properties. Several of these are now listed.

Divided difference tables. Divided differences are most conveniently presented in tables. Traditionally, they are arranged in a display like that in Fig. 2.1.1. Each divided difference is computed from its two immediate neighbors in the column to its left. For our purposes, it turns out to be most useful to define the table as an upper triangular matrix.

†Milne-Thomson [12] writes $\Delta_\zeta^k f$ as $[\zeta_0, \zeta_1, \ldots, \zeta_n]$, suppressing the function; Davis [2] uses $f^{[n]}(\zeta_0, \zeta_1, \ldots, \zeta_n)$; and Kahan [11] uses $\Delta f(\zeta_0, \zeta_1, \ldots, \zeta_n)$, which suggested the notation used here.

$$
\begin{array}{llllll}
\zeta_0 & f(\zeta_0) & & & & \\
 & & \Delta_0^1 f & & & \\
\zeta_1 & f(\zeta_1) & & \Delta_0^2 f & & \\
 & & \Delta_1^1 f & & \Delta_0^3 f & \\
\zeta_2 & f(\zeta_2) & & \Delta_1^2 f & & \Delta_0^4 f \\
 & & \Delta_2^1 f & & \Delta_1^3 f & \\
\zeta_3 & f(\zeta_3) & & \Delta_2^2 f & & \\
 & & \Delta_3^1 f & & & \\
\zeta_4 & f(\zeta_4) & & & &
\end{array}
$$

Fig. 2.1.1: Standard divided difference table.

$$
\Delta f \equiv \begin{bmatrix}
f(\zeta_0) & \Delta_0^1 f & \Delta_0^2 f & \cdot & \cdot & \cdot & \Delta_0^n f \\
 & f(\zeta_1) & \Delta_1^1 f & \cdot & \cdot & \cdot & \Delta_1^{n-1} f \\
 & & f(\zeta_2) & \cdot & \cdot & \cdot & \Delta_2^{n-2} f \\
 & & & \cdot & & & \cdot \\
 & & & & \cdot & \cdot & \cdot \\
 & & & & & \cdot & \cdot \\
 & & & & & & f(\zeta_n)
\end{bmatrix} \tag{2.1.9}
$$

The symbol Δf, without the superscript, is used here as a matrix, not a scalar. The elements of the matrix depend on their immediate neighbors in the diagonal below.

$$
\begin{array}{ccccccccc}
f(\zeta_i) & \rightarrow & \Delta_i^1 f & \rightarrow & \Delta_i^2 f & \rightarrow & \Delta_i^3 f & \rightarrow & \Delta_i^4 f \\
 & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 & & f(\zeta_{i+1}) & \rightarrow & \Delta_{i+1}^1 f & \rightarrow & \Delta_{i+1}^2 f & \rightarrow & \Delta_{i+1}^3 f \\
 & & & & \uparrow & & \uparrow & & \uparrow \\
 & & & & f(\zeta_{i+2}) & \rightarrow & \Delta_{i+2}^1 f & \rightarrow & \Delta_{i+2}^2 f \\
 & & & & & & \uparrow & & \uparrow \\
 & & & & & & f(\zeta_{i+3}) & \rightarrow & \Delta_{i+3}^1 f \\
 & & & & & & & & \uparrow \\
 & & & & & & & & f(\zeta_{i+4})
\end{array}
$$

Fig. 2.1.2: Pattern of dependence in a divided difference table.

This causes a "pattern of dependence" where $\Delta_j^k f$ is independent of all table entries in rows before the j-th and columns after the $j+k$-th. This pattern of dependence is characteristic of triangular matrices.

Linearity. If α and β are scalars,

$$\Delta_0^n(\alpha f + \beta g) = \alpha \Delta_0^n f + \beta \Delta_0^n g\,. \tag{2.1.10}$$

Translation invariance. For $\mathbf{Z}-\mathbf{a} \equiv \{\zeta_0 - a, \zeta_1 - a, \ldots, \zeta_n - a\}$ and $f_a(\zeta) \equiv f(\zeta - a)$,

$$\Delta_{\zeta_0}^n f_a = \Delta_{\zeta_0 - a}^n f\,. \tag{2.1.11}$$

$$\begin{bmatrix} 1.000 & 1.718 & 1.476 & .8455 & .3632 \\ & 2.718 & 4.671 & 4.013 & 2.298 \\ & & 7.389 & 12.70 & 10.91 \\ & & & 20.09 & 34.51 \\ & & & & 54.60 \end{bmatrix}$$

Fig. 2.1.3: Divided difference table for $f = \exp$, $\mathbf{Z} = \{0, 1, 2, 3, 4\}$.

For example,

$$\Delta^1_{\zeta_0} f_a = \frac{f_a(\zeta_1) - f_a(\zeta_0)}{\zeta_1 - \zeta_0} = \frac{f(\zeta_1 - a) - f(\zeta_0 - a)}{(\zeta_1 - a) - (\zeta_0 - a)} = \Delta^1_{\zeta_0 - a} f .$$

Mean value representation. When the abscissae are **real**, then [12]

$$\Delta^n_0 f = \frac{f^{(n)}(\zeta)}{n!}, \quad \min_{0 \leqslant j \leqslant n} \zeta_j < \zeta < \max_{0 \leqslant i \leqslant n} \zeta_i \tag{2.1.12}$$

for any f having n continuous derivatives in the interval containing the data points. This representation has no equivalent for the case of complex abscissae. For example, if $f = \exp$ and $\zeta_0 = \xi$, $\zeta_1 = \xi + 2\pi i$, then

$$\Delta^1_0 \exp = \frac{e^{\xi + 2\pi i} - e^{\xi}}{(\xi + 2\pi i) - \xi} = 0 \neq e^{\zeta}$$

for any finite ζ.

Integral representation. Another representation for $\Delta^n_0 f$ when f has a bounded n-th order derivative on a closed convex domain $\overline{\Omega}$ containing $\mathbf{Z}$ is [5]

$$\Delta^n_0 f = \int_0^1 \int_0^{t_1} \cdots \int_0^{t_{n-1}} f^{(n)}[\zeta_0 + (\zeta_1 - \zeta_0) t_1 + \cdots + (\zeta_n - \zeta_{n-1}) t_n] \, dt_n \cdots dt_2 \, dt_1 . \tag{2.1.13}$$

Bound. If f has a bounded n-th derivative on the closed convex domain $\overline{\Omega}$ containing the set of data points $\mathbf{Z}$, then [5]

$$|\Delta^n_0 f| \leqslant \frac{1}{n!} \max_{\zeta \in \overline{\Omega}} |f^{(n)}(\zeta)| . \tag{2.1.14}$$

$\Delta^n_0 f$ may be represented in other ways, such as by contour integrals [5]; however, these are not needed here. The reader is referred to books on the finite difference calculus by Milne-Thomson [12], Gel'fond [5], or Jordan [7] for more details on divided differences and their uses.

2.2 Divided differences for some special functions.

Before proceeding to develop other methods for calculating divided differences of general functions f, it is worthwhile to see how the classical formula (2.1.8) can be employed on some specific functions. Later, these results will be used to develop formulas for more general functions.

Following the notation of Davis [2], let $f = \uparrow^p$ be the p-th power function, $\uparrow^p(\zeta) \equiv \zeta^p$. Because the notation introduced in the previous section suppresses variables, clarity demands that every function have a name. The linearity property of the divided difference allows

immediate extension of results for $\uparrow^p$, any $p \geqslant 0$, to general polynomials. The well-known formula [12] which results from (2.1.8) is

$$\Delta_0^{n}\uparrow^p = \sum_{\substack{k_0+k_1+\cdots+k_n=p-n \\ k_i\geqslant 0}} \zeta_0^{k_0}\zeta_1^{k_1}\cdots\zeta_n^{k_n}, \quad p \geqslant n. \tag{2.2.1}$$

In particular,

$$\Delta_0^{n}\uparrow^p = 0 \quad \text{if } p < n,$$
$$\Delta_0^{n}\uparrow^n = 1,$$
$$\Delta_0^{n}\uparrow^{n+1} = \sum_{j=0}^{n}\zeta_j.$$

Writing (2.2.1) as a system of equations for $\Delta_0^{k}\uparrow^{p+k}$, $0\leqslant k\leqslant n$, yields

$$\begin{bmatrix} \Delta_0^{0}\uparrow^p \\ \Delta_0^{1}\uparrow^{p+1} \\ \cdot \\ \cdot \\ \Delta_0^{n}\uparrow^{p+n} \end{bmatrix} = \begin{bmatrix} \zeta_0 & & & & \\ \zeta_0 & \zeta_1 & & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & & \cdot & \\ \zeta_0 & \zeta_1 & \cdot & \cdot & \zeta_n \end{bmatrix}^p \begin{bmatrix} 1 \\ 1 \\ \cdot \\ \cdot \\ 1 \end{bmatrix}. \tag{2.2.2}$$

In fact, all the differences $\Delta_j^{k}\uparrow^{p+k}$ for $k=0,1,\ldots,n-j$ and $j=0,1,\ldots,n$ can be expressed compactly as a matrix,

$$\begin{bmatrix} \Delta_0^{0}\uparrow^p & \Delta_0^{1}\uparrow^{p+1} & \cdot & \cdot & \Delta_0^{n}\uparrow^{p+n} \\ & \Delta_1^{0}\uparrow^p & \cdot & \cdot & \Delta_1^{n-1}\uparrow^{p+n-1} \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ & & & & \Delta_n^{0}\uparrow^p \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & \cdot & \cdot & 1 \\ & 1 & \cdot & \cdot & 1 \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ & & & & 1 \end{bmatrix} \begin{bmatrix} \zeta_0 & \zeta_0 & \cdot & \cdot & \zeta_0 \\ & \zeta_1 & \cdot & \cdot & \zeta_1 \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ & & & & \zeta_n \end{bmatrix}^p. \tag{2.2.3}$$

$$= \begin{bmatrix} \zeta_0 & \zeta_1 & \cdot & \cdot & \zeta_n \\ & \zeta_1 & \cdot & \cdot & \zeta_n \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ & & & & \zeta_n \end{bmatrix}^p \begin{bmatrix} 1 & 1 & \cdot & \cdot & 1 \\ & 1 & \cdot & \cdot & 1 \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ & & & & 1 \end{bmatrix}. \tag{2.2.4}$$

The alternative form (2.2.4) results from the symmetry property. The matrix representation idea will be employed extensively in §2.6 where another expression similar to (2.2.3) and (2.2.4) will be developed.

Formula (2.2.2) leads directly to a simple, but elegant, procedure for computing the $\Delta_0^{k}\uparrow^{p+k}$, $0\leqslant k\leqslant n$, recursively in p, and so divided differences of any polynomial may be formed.

Algorithm 2.1: Recursive computation of $\Delta_0^{k\uparrow p+k}$.

1. Initialize $\Delta_0^{k\uparrow k}=1$, $k=0,1,\ldots,n$
2. For $p=1,2,\ldots$
   $\Delta_0^{k\uparrow p+k}=\zeta_k\cdot\Delta_0^{k\uparrow p+k-1}+\Delta_0^{k-1\uparrow p+k-1}$, $k=0,1,\ldots,n$.
   $(\Delta_j^k f\equiv 0$ for $k<0)$.

For $p=0$,

$$\Delta_0^{0\uparrow 0}=\Delta_0^{1\uparrow 1}=\Delta_0^{2\uparrow 2}\equiv 1$$

For $p=1$,

$$\Delta_0^{0\uparrow 1}=\zeta_0\cdot\Delta_0^{0\uparrow 0}+\Delta_0^{-1\uparrow 0}=\zeta_0$$
$$\Delta_0^{1\uparrow 2}=\zeta_1\cdot\Delta_0^{1\uparrow 1}+\Delta_0^{0\uparrow 1}=\zeta_1+\zeta_0$$
$$\Delta_0^{2\uparrow 3}=\zeta_2\cdot\Delta_0^{2\uparrow 2}+\Delta_0^{1\uparrow 2}=\zeta_2+\zeta_1+\zeta_0$$

For $p=2$,

$$\Delta_0^{0\uparrow 2}=\zeta_0\cdot\Delta_0^{0\uparrow 1}+\Delta_0^{-1\uparrow 1}=\zeta_0^2$$
$$\Delta_0^{1\uparrow 3}=\zeta_1\cdot\Delta_0^{1\uparrow 2}+\Delta_0^{0\uparrow 2}=\zeta_1^2+\zeta_1\zeta_0+\zeta_0^2$$
$$\Delta_0^{2\uparrow 4}=\zeta_2\cdot\Delta_0^{2\uparrow 3}+\Delta_0^{1\uparrow 3}=\zeta_2^2+\zeta_2\zeta_1+\zeta_2\zeta_0+\zeta_1^2+\zeta_1\zeta_0+\zeta_0^2$$

Fig. 2.2.1: First couple of steps of Algorithm 2.1 for $n=2$.

The computation requires $n+1$ multiplications for each p. A companion procedure for $\Delta_{n-k}^{k\uparrow p+k}$, $0\leqslant k\leqslant n$, comes from (2.2.4).

Algorithm 2.1': Recursive computation of $\Delta_{n-k}^{k\uparrow p+k}$.

1. Initialize $\Delta_{n-k}^{k\uparrow k}=1$, $k=0,1,\ldots,n$
2. For $p=1,2,\ldots$
   $\Delta_{n-k}^{k\uparrow p+k}=\zeta_{n-k}\cdot\Delta_{n-k}^{k\uparrow p+k-1}+\Delta_{n-k+1}^{k-1\uparrow p+k-1}$, $k=0,1,\ldots,n$.

For $p=0$,

$$\Delta_2^{0\uparrow 0}=\Delta_1^{1\uparrow 1}=\Delta_0^{2\uparrow 2}\equiv 1$$

For $p=1$,

$$\Delta_2^{0\uparrow 1}=\zeta_2\cdot\Delta_2^{0\uparrow 0}+\Delta_3^{-1\uparrow 0}=\zeta_2$$
$$\Delta_1^{1\uparrow 2}=\zeta_1\cdot\Delta_1^{1\uparrow 1}+\Delta_2^{0\uparrow 1}=\zeta_1+\zeta_2$$
$$\Delta_0^{2\uparrow 3}=\zeta_0\cdot\Delta_0^{2\uparrow 2}+\Delta_1^{1\uparrow 2}=\zeta_0+\zeta_1+\zeta_2$$

For $p=2$,

$$\Delta_2^{0\uparrow 2}=\zeta_2\cdot\Delta_2^{0\uparrow 1}+\Delta_3^{-1\uparrow 1}=\zeta_2^2$$
$$\Delta_1^{1\uparrow 3}=\zeta_1\cdot\Delta_1^{1\uparrow 2}+\Delta_2^{0\uparrow 2}=\zeta_1^2+\zeta_1\zeta_2+\zeta_2^2$$
$$\Delta_0^{2\uparrow 4}=\zeta_0\cdot\Delta_0^{2\uparrow 3}+\Delta_1^{1\uparrow 3}=\zeta_0^2+\zeta_0\zeta_1+\zeta_0\zeta_2+\zeta_1^2+\zeta_1\zeta_2+\zeta_2^2$$

Fig. 2.2.2: First couple of steps of Algorithm 2.1' for $n=2$.

2.3 Difficulties in computing divided differences.

Divided differences have acquired a poor reputation in numerical analysis. That this reputation is to some extent justified is illustrated by the following example.

Divided differences by recursive formula					
ζ	Δ^0exp	Δ^1exp	Δ^2exp	Δ^3exp	Δ^4exp
0.00	1.000	1.136	6.480E-1	2.347E-1	8.530E-2
0.25	1.284	1.460	8.240E-1	3.200E-1	
0.50	1.649	1.872	1.064		
0.75	2.117	2.404			
1.00	2.718				

Correct value of divided differences to 4 digits					
ζ	Δ^0exp	Δ^1exp	Δ^2exp	Δ^3exp	Δ^4exp
0.00	1.000	1.136	6.454E-1	2.444E-1	6.942E-2
0.25	1.284	1.159	8.287E-1	3.138E-1	
0.50	1.649	1.873	1.064		
0.75	2.117	2.405			
1.00	2.718				

Fig. 2.3.1: Example of loss of accuracy in computing divided differences.

The difficulty is one of finite precision arithmetic. That is, f is smooth enough and the abscissae are close enough together that the implementation of the recursive definition (2.1.8) involves not only subtraction of nearly equal quantities, but also division by small quantities. Computed divided differences which are not small and contain few, or even no correct digits result. The possibility of computing divided differences using schemes other than (2.1.8) will be investigated following a discussion of the above remarks.

Subtractive cancellation. Suppose $\Delta_0^{k+1}f$ is computed using (2.1.8). Let the abscissae $\mathbf{Z}=\{\zeta_0,\zeta_1,\ldots,\zeta_n\}$ and $\Delta_0^k f$ be representable exactly in the computer, and assume all arithmetic is performed exactly. Let $fl(\Delta_1^k f)\equiv\Delta_1^k f(1+\epsilon)$, for some small relative error ϵ, be the available or previously computed floating-point value of $\Delta_1^k f$. Then

$$fl(\Delta_0^{k+1}f)\equiv\Delta_0^{k+1}f(1+\epsilon')=\frac{\Delta_1^k f(1+\epsilon)-\Delta_0^k f}{\zeta_{k+1}-\zeta_0}=\Delta_0^{k+1}f+\frac{\epsilon\Delta_1^k f}{\zeta_{k+1}-\zeta_0}. \qquad (2.3.1)$$

When $|\Delta_1^k f-\Delta_0^k f|=2^{-t}|\Delta_1^k f|$, $|\epsilon'|=2^t|\epsilon|$. For $t\leqslant 0$, $fl(\Delta_0^{k+1}f)$ has, at least, as much relative accuracy as the data it is formed from. However, when t is significantly greater than 0, the relative accuracy of $fl(\Delta_0^{k+1}f)$ is worse than for $fl(\Delta_1^k f)$. $\Delta_1^k f$ and $\Delta_0^k f$ have leading digits in common and subtractive cancellation occurs. That is, about t leading binary digits cancel, and when the resulting answer is normalized to obtain a non-zero leading digit, t zeros are appended at the right. If $\Delta_1^k f$ is given to p correct digits, $\Delta_0^{k+1}f$ is computed to about $p-t$ correct digits.

This phenomenon may have devastating effect on higher order divided differences. When an early difference has already lost t_1 binary digits, $|\epsilon_1|\approx 2^{t_1}|\epsilon|$, and this difference is used to

Subtractive cancellation with small divisors, 6 digit arithmetic						
ζ	Δ^0tan	$fl(\Delta^1$tan)	$fl(\Delta^2$tan)	Δ^0tan	Δ^1tan	Δ^2tan
0	0.00000	1.27324	9.24163E-1	0.00000	1.27324	9.26709E-1
$\pi/4$	1.00000	2.00000		1.00000	2.00200	
$\pi/4+.001$	1.00200			1.00200		

Subtractive cancellation but no small divisors						
ζ	Δ^0tan	$fl(\Delta^1$tan)	$fl(\Delta^2$tan)	Δ^0tan	Δ^1tan	Δ^2tan
0	0.00000	1.27324	-3.23983E-1	0.00000	1.27324	-3.23983E-1
$\pi/4$	1.00000	6.36417E-4		1.00000	6.37055E-4	
$5\pi/4+.001$	1.00200			1.00200		

Fig. 2.3.2: Large relative errors in small differences may not be serious.

compute a subsequent divided difference where t_2 binary digits are lost, t_1+t_2 digits may be lost in the computed result, $|\epsilon_2| \approx 2^{t_1+t_2}|\epsilon|$. Clearly, all accuracy in the recursive computation may be quickly lost.

Loss of relative accuracy may, or may not, be a disaster. If these differences are required as an end in themselves, there is little hope. Usually, however, they are used in additional computations; and errors, even relatively large ones, in quantities which are small relative to others involved in the calculation at hand need not be serious. If a computed divided difference is small relative to other divided differences of the same order in the table or is small relative to some given criterion, this difference may be acceptable, even if inaccurate. Subtraction between nearly equal numbers always leads to a result smaller than either of the original numbers; so subtractive cancellation in (2.1.8) tends to create inaccurate, but small, divided differences.

Small divisors. When can relatively large, but inaccurate, divided differences be obtained? The division by the difference of the abscissae is clearly the culprit. Specifically, if $|\zeta_{k+1}-\zeta_0| = 2^{-q}$, then

$$|fl(\Delta_0^{k+1}f) - \Delta_0^{k+1}f| = 2^q|\epsilon|\,|\Delta_1^k f|\,.$$

For $q > 0$, the absolute error in $fl(\Delta_0^{k+1})$ is large relative to $\Delta_1^k f$; and if $t > 0$ also, the error is large relative to $\Delta_0^{k+1}f$ as well. Because $q > 0$ requires $\Delta_0^{k+1}f$ to be larger in absolute value than $|\Delta_1^k f|$ when cancellation is absent, the combination of close abscissae **and** subtractive cancellation of the intermediate divided differences is what is to be feared.

The devastating effect of close data points is most apparent for divided differences of smooth, slowly varying functions. The divided dfferences, like the derivatives, of such functions are also slowly varying under small changes in the data points; so formula (2.1.8) will experience subtractive cancellation between the intermediate divided differences in its numerator. Fig. 2.3.3 contrasts the effect of the same "close" abscissae for the "rapidly" varying function $f = \exp_{10}$, $\exp_{10}(\zeta) = e^{10\zeta}$, with the less rapidly varying $f = \exp$ of Fig. 2.3.1. Formula (2.1.8) gives results correct to a full four digits for $\exp_{10}$. Hence for smooth, slowly varying

Divided differences of $\exp_{10}$ by the recursive formula					
ζ	$\Delta^0\exp_{10}$	$\Delta^1\exp_{10}$	$\Delta^2\exp_{10}$	$\Delta^3\exp_{10}$	$\Delta^4\exp_{10}$
0.00	1.000	4.472E+1	1.000E+3	1.492E+4	1.668E+5
0.25	1.218E+1	5.449E+2	1.219E+4	1.817E+5	
0.50	1.484E+2	6.638E+3	1.485E+5		
0.75	1.808E+3	8.089E+4			
1.00	2.203E+4				

Fig. 2.3.3: The recursive scheme works well for rapidly varying functions.

functions, close abscissae lead to subtractive cancellation; but exactly what "close" is depends on the function.

There are two fundamental situations in which divided differences appear. (1) Given the abscissae and a table of function values at the abscissae, compute the divided differences. No knowledge of the expression for the function is required; but also if the data are not given to sufficient precision, (2.1.8) will fail and there is no way to obtain desired accuracy in the result. (2) Given an expression for the function f and the ability to compute f's value and derivatives at any point in its domain, compute the divided differences of f. The question must be asked, "What advantage can be taken of this knowledge about f?" The following section will show that for some elementary functions, $\Delta_0^n f$ can be computed accurately in such a way as to avoid the difficulties indicated in this section.

2.4 Power series for $\Delta_0^n f$.

For a special class of functions f an alternative method to (2.1.8) for forming divided differences may be employed. This new method works particularly well when the abscissae are nearly confluent. The special class of functions includes many of the elementary functions. This method will now be presented and error bounds given.

In this section, f is a function which is represented as a power series about the point $\zeta = a$ for all ζ inside the open disk $|\zeta - a| < R$. Expanding on the notation of Davis [2],

$$f = \sum_{j=0}^{\infty} \alpha_j \uparrow_a^j . \tag{2.4.1}$$

Here $\uparrow_a^j(\zeta) \equiv \uparrow^j(\zeta - a) = (\zeta - a)^j$. In complex variable theory it is well-known that on the disk $\Omega_r \equiv \{\zeta \mid r \geqslant |\zeta - a| \text{ and } r < R\}$,

(1) the power series (2.4.1) converges absolutely and uniformly to f;

(2) $\alpha_j = f^{(j)}(a)/j!$ for $j = 0, 1, 2, \ldots$; the power series is the Taylor series of f;

(3) f is holomorphic;

(4) term by term differentiation is possible, giving a power series converging to f' on Ω_r.

This last fact extends directly to divided differences, and yields a formula similar to that in Brent [1] for which Taylor's theorem with remainder was used.

Theorem 2.1: Power series for divided differences. Let f have the power series representation (2.4.1) on the disk $|\zeta - a| < R$. Then if the set of abscissae $\mathbf{Z} = \{\zeta_0, \zeta_1, \ldots, \zeta_n\}$ lies entirely within the disk,

$$\Delta_0^n f = \sum_{j=n}^{\infty} \alpha_j \Delta_0^n \uparrow_a^j . \tag{2.4.2}$$

proof: Let $s_m = \sum_{j=0}^{m} \alpha_j \uparrow_a^j$. By fact (4), for any n $\max_{\zeta \in \Omega_r} |f^{(n)}(\zeta) - s_m^{(n)}(\zeta)| \to 0$ as $m \to \infty$ when $r < R$. As r may be chosen such that $\mathbf{Z} \subset \Omega_r$, we have by (2.1.13)

$$|\Delta_0^n f - \Delta_0^n s_m| \leq \int_0^1 \int_0^{t_1} \cdots \int_0^{t_{n-1}} |f^{(n)}(\omega) - s_m^{(n)}(\omega)| dt_n \cdots dt_2 dt_1$$

$$\leq \frac{1}{n!} \cdot \max_{\zeta \in \Omega_r} |f^{(n)}(\zeta) - s_m^{(n)}(\zeta)| \to 0$$

as $m \to \infty$, where $\omega = \zeta_0 + t_1(\zeta_1 - \zeta_0) + \cdots + t_n(\zeta_n - \zeta_{n-1}) \in \Omega_r$ because Ω_r is convex. □

The computation of divided differences for functions such as exp, sin, and cosh is quite straightforward because the power series coefficients are easily obtained. (2.4.2) may also be used to form divided differences of other functions, such as $\sqrt{}$ and log; however, great care is required to find a series representation whose circle of convergence includes all the abscissae. This limitation to the circle of convergence cannot be over-emphasized. The **same** power series representation must converge to f at each abscissa.

Algorithm. Algorithm 2.1 of §2.2 and Theorem 2.1 lead immediately to a method for computing $\Delta_0^k f$, $0 \leq k \leq n$. Let $s_m = \sum_{j=0}^{m} \alpha_j \uparrow_a^j$ be the partial sums of f. Also recall the translation invariance property (§2.1).

Algorithm 2.2: Power series for $\Delta_0^k f$.

1. Initialize $\Delta_0^k \uparrow_a^k = 1$, $\Delta_0^k s_k = \alpha_k$, $k = 0, 1, \ldots, n$
2. For $p = 1, 2, \ldots$
   $\Delta_0^k \uparrow_a^{p+k} = (\zeta_k - a) \cdot \Delta_0^k \uparrow_a^{p+k-1} + \Delta_0^{k-1} \uparrow_a^{p+k-1}$
   $\Delta_0^k s_{p+k} = \Delta_0^k s_{p+k-1} + \alpha_{p+k} \cdot \Delta_0^k \uparrow_a^{p+k}$, $k = 0, 1, \ldots, n$.

A companion algorithm may also be derived from Algorithm 2.1'.

Algorithm 2.2': Power series for $\Delta_{n-k}^k f$.

1. Initialize $\Delta_{n-k}^k \uparrow_a^k = 1$, $\Delta_{n-k}^k s_k = \alpha_k$, $k = 0, 1, \ldots, n$
2. For $p = 1, 2, \ldots$
   $\Delta_{n-k}^k \uparrow_a^{p+k} = (\zeta_{n-k} - a) \cdot \Delta_{n-k}^k \uparrow_a^{p+k-1} + \Delta_{n-k+1}^{k-1} \uparrow_a^{p+k-1}$
   $\Delta_{n-k}^k s_{p+k} = \Delta_{n-k}^k s_{p+k-1} + \alpha_{p+k} \cdot \Delta_{n-k}^k \uparrow_a^{p+k}$, $k = 0, 1, \ldots, n$.

Exclusive of the coefficient evaluations, the scheme requires $2n+2$ multiplications per p-step.

Error bound. A bound on the computational error in Algorithm 2.2 may be given. Let $\delta = \max_i |\zeta_i - a|$. To keep the form of the bounds simple, assume the arithmetic is such that

For $p=0$,

$$\Delta_0^0\uparrow_a^0 = \Delta_0^1\uparrow_a^1 = \Delta_0^2\uparrow_a^2 \equiv 1$$

$$\Delta_0^0 s_0 = \alpha_0,\ \Delta_0^1 s_1 = \alpha_1,\ \Delta_0^2 s_2 = \alpha_2$$

For $p=1$,

$$\Delta_0^0\uparrow_a^1 = (\zeta_0 - a)\cdot\Delta_0^0\uparrow_a^0 + \Delta_0^{-1}\uparrow_a^0 = \zeta_0 - a$$

$$\Delta_0^0 s_1 = \Delta_0^0 s_0 + \alpha_1\cdot\Delta_0^0\uparrow_a^1 = \alpha_0 + \alpha_1(\zeta_0 - a);$$

$$\Delta_0^1\uparrow_a^2 = (\zeta_1 - a)\cdot\Delta_0^1\uparrow_a^1 + \Delta_0^0\uparrow_a^1 = \zeta_1 + \zeta_0 - 2a$$

$$\Delta_0^1 s_2 = \Delta_0^1 s_1 + \alpha_2\cdot\Delta_0^1\uparrow_a^2 = \alpha_1 + \alpha_2(\zeta_1 + \zeta_0 - 2a);$$

$$\Delta_0^2\uparrow_a^3 = (\zeta_2 - a)\cdot\Delta_0^2\uparrow_a^2 + \Delta_0^1\uparrow_a^2 = \zeta_2 + \zeta_1 + \zeta_0 - 3a$$

$$\Delta_0^2 s_3 = \Delta_0^2 s_2 + \alpha_3\cdot\Delta_0^2\uparrow_a^3 = \alpha_2 + \alpha_3(\zeta_2 + \zeta_1 + \zeta_0 - 3a).$$

Fig. 2.4.1: First step of Algorithm 2.2 for $n=2$.

$$|fl(\sum_i a_i) - \sum_i a_i| \leqslant \epsilon\sum_i |a_i|. \tag{2.4.3}$$

In keeping with Wilkinson [20], $\epsilon \leqslant 1.06\times$ (machine precision) and $fl(g)$ is the computed value of g. The following error bound can be developed (Appendix A).

$$|fl(\Delta_0^n f) - \Delta_0^n f| \leqslant \frac{\epsilon}{n!}\sum_{p=0}\frac{(p+1)(n+p)!|\alpha_{n+p}|\delta^p}{p!} \tag{2.4.4}$$

For example, when $f=\exp_t$, $\exp_t(\zeta)\equiv e^{t\zeta}$,

$$|fl(\Delta_0^n\exp_t) - \Delta_0^n\exp_t| \leqslant \epsilon e^{t\delta}(1+t\delta)\frac{t^n|e^{ta}|}{n!}. \tag{2.4.5}$$

This bound is small relative to $|f^{(n)}(a)|/n! = t^n|e^{ta}|/n!$ when $t\delta$ is small. $t\delta$ can be made small for nearly confluent abscissae by choosing a so as to minimize δ.

Close data points and smooth functions f constitute the difficult case mentioned in §2.3. Algorithm 2.2 appears to fill the gap in computing divided differences for nearly confluent abscissae. One expects the recursive definition (2.1.8) and the results of this section to be sufficient to form divided difference tables for the elementary functions. That this expectation cannot quite be realized is demonstrated in the next section.

2.5 Inadequacy of the series and recursive formulas, the question of "close" abscissae.

The arguments of §2.3 indicate that divided differences computed using the recursive definition (2.1.8) may be unacceptably inaccurate when f is smooth **and** the abscissae are too "close". The Taylor series algorithm 2.2 of the previous section appears to remove this difficulty for some f because the error bound (2.4.4) on the algorithm is smallest when the data points are close. The idea is to order, at least crudely, the abscissae such that close abscissae are clustered together in the ordering, and well-separated abscissae are well-separated in the ordering. Then recalling the pattern of dependence of elements of the divided difference table, the portions of the table depending on the clustered data points may be computed using the Taylor series algorithm and the remainder of the table filled in using the recursive definition. This process is discussed in detail in §2.8 and is illustrated in Fig. 2.8.2.

There are two difficulties here. (1) There must be freedom to order the data points as desired. The amount of freedom depends on the use contemplated for the divided difference

k	abscissae ζ_k	Δ_0^kexp, correct to 7 decimal digits	Δ_0^kexp using recursive scheme after $k=8$	Δ_0^kexp using Taylor series Algorithm 2.2	Δ_0^kexp using Algorithm 2.4 of §2.7
0	-13.0	2.260329E-06			2.260332E-06
1	-12.5	2.932648E-06		2.917451E-06	2.932650E-06
2	-12.0	1.902471E-06		1.905751E-06	1.902472E-06
3	-11.5	8.227822E-07		8.215427E-07	8.227824E-07
4	-11.0	2.668782E-07		2.669856E-07	2.668782E-07
5	-10.5	6.925181E-08		6.925365E-08	6.925180E-08
6	-10.0	1.497504E-08		1.496362E-08	1.497505E-08
7	-9.5	2.775608E-09		2.777049E-09	2.775609E-09
8	-9.0	4.501490E-10	4.501490E-10	4.501040E-10	4.501491E-10
9	-8.5	6.489361E-11	6.489360E-11	6.490737E-11	6.489364E-11
10	-8.0	8.419572E-12	8.419580E-12	8.420343E-12	8.419573E-12
11	-7.5	9.930829E-13	9.930800E-13	9.930225E-13	9.930829E-13
12	-7.0	1.073723E-13	1.073727E-13	1.073735E-13	1.073724E-13
13	-6.5	1.071611E-14	1.071609E-14	1.071603E-14	1.071611E-14
14	-6.0	9.931098E-16	9.931271E-16	9.930869E-16	9.931100E-16
15	-5.5	8.590019E-17	8.590612E-17	8.590039E-17	8.590021E-17
16	-5.0	6.965660E-18	6.951898E-18	6.965612E-18	6.965660E-18
17	-4.5	5.316202E-19	5.402473E-19	5.316202E-19	5.316201E-19
18	-4.0	3.831926E-20	3.486964E-20	3.831920E-20	3.831926E-20
19	-3.5	2.616686E-21	3.650387E-21	2.616686E-21	2.616687E-21
20	-3.0	1.697500E-22	-6.986900E-23	1.697499E-22	1.697500E-22
21	-2.5	1.048766E-23	5.010054E-23	1.048766E-23	1.048766E-23
22	-2.0	6.185062E-25	-1.792933E-24	6.185061E-25	6.185063E-25
23	-1.5	3.489027E-26	-1.236419E-24	3.489027E-26	3.489027E-26
24	-1.0	1.886172E-27	6.496526E-25	1.886172E-27	1.886172E-27
25	-0.5	9.788799E-29	-1.931645E-25	9.788798E-29	9.788797E-29

Fig. 2.5.1: Top row of Δexp using several methods.

table. For the matrix function problem, this point is considered in §3.1. (2) It is not clear what "close" means when speaking of clustering the abscissae. This can be illustrated by the example in Fig. 2.5.1.

The divided differences in the example were computed, from the 9-th difference on, using formula (2.1.8). All divided differences before the 9-th were formed exactly by the formula

$$\Delta_0^k \exp = \frac{1}{k!} e^a [(e^\alpha - 1)/\alpha]^k \tag{2.5.1}$$

where the set of abscissae $\mathbf{Z} = \{a, a+\alpha, \ldots, a+k\alpha, \ldots, a+n\alpha\}$ is evenly spaced. Even with a spread $\delta_n \equiv |\zeta_n - \zeta_0| = 12.5$, Δ_0^nexp cannot be computed accurately by (2.1.8) given the number of digits available. The Taylor series algorithm 2.2 might be employed instead, but the error bound (2.4.5) when $\delta = \delta_n/2 = 6.25$, is unacceptable. Actually, Algorithm 2.2 yields an excellent value for Δ_0^{25}exp because the abscissae are symmetrically spaced about the Taylor series expansion point. However, some of the lower order divided differences are not nearly as accurate. These differences may be computed using Algorithm 2.2 about other expansion points, but the advantage of obtaining all Δ_0^kexp, $k=0, 1, \ldots, 25$, in one computation is lost. So, ζ_n is too close to ζ_0 to use (2.1.8) here; but the abscissae are too well-separated to use Algorithm 2.2

efficiently. To illustrate why this difficulty arises, consider the following two special cases.

1. Suppose the abscissae are real, increasing and evenly spaced. Formula (2.1.8) is used. If $\Delta_0^k\exp = \frac{1}{k!}e^a[(e^\alpha-1)/\alpha]^k$ and $\Delta_1^k\exp = \frac{1}{k!}e^{a+\alpha}[(e^\alpha-1)/\alpha]^k$ are given exactly, then

$$\Delta_0^{k+1}\exp = \frac{\frac{1}{k!}e^{a+\alpha}[(e^\alpha-1)/\alpha]^k - \frac{1}{k!}e^a[(e^\alpha-1)/\alpha]^k}{(k+1)\alpha} = \frac{e^a}{\alpha(k+1)!}\left[\frac{e^\alpha-1}{\alpha}\right]^k(e^\alpha-1) \quad (2.5.2)$$

where common factors have been removed. In binary arithmetic subtractive cancellation will occur when $\alpha < \log 2$. [In principle, $e^\alpha - 1$ can be computed very accurately for small α, e.g. use the Taylor series. However, (2.1.8) avoids explicit use of the function exp and forces subtraction of (possibly) nearly equal quantities.] Thus, even if all k-th differences are given to full machine accuracy, formula (2.1.8) will suffer a loss of information at each further step when $\alpha < \log 2$. If n is large enough, $\delta = \delta_n/2 = n\alpha/2$ will also yield too large a bound on the Taylor series error (2.4.5) to permit safe use of that method. $\Delta_0^n\exp$ can be computed here by neither of the methods yet discussed.

2. Suppose the abscissae are $\mathbf{Z} = \{0, 0, \ldots, 0, \delta_n\}$ where $\delta_n > 0$.

$$\Delta_0^n\exp = \frac{\Delta_1^{n-1}\exp - \Delta_0^{n-1}\exp}{\delta_n} = \frac{1}{\delta_n}\left[\sum_{k=n-1}^{\infty}\frac{\delta_n^{k-n+1}}{k!} - \frac{1}{(n-1)!}\right] = \sum_{k=n}^{\infty}\frac{\delta_n^{k-n}}{k!} \quad (2.5.3)$$

The power series formula (2.4.2) was used to expand $\Delta_1^{n-1}\exp$. In binary arithmetic, subtractive cancellation will occur in the numerator when

$$\delta_n(n-1)!\cdot\sum_{k=n}^{\infty}\frac{\delta_n^{k-n}}{k!} < 1. \quad (2.5.4)$$

n	10	20	30	50	100
δ_n	<6	<11	<16	<26	<51

Fig. 2.5.2: Upper bounds on δ_n for possible cancellation in example 2.

The table in Fig. 2.5.2 indicates that when $\delta_n < \frac{1}{2}n+1$, approximately, subtractive cancellation will occur in using formula (2.1.8).

The examples illustrate that, in general, there is no simple a priori answer to when (2.1.8) will, or will not, fail. In practice, this forces a choice. (1) Try to find an a priori criterion, say $|\zeta_n - \zeta_0| > n$, for when (2.1.8) is assumed to work; or (2) compute the divided difference table by (2.1.8) and monitor the error,and if failure occurs for some difference (i.e. an error estimate becomes too large) do something else to get that difference. The former is simple, but unsure; the latter is sure, but more complicated and possibly more costly. §2.8 expands these ideas and deals with the specifics of computing the entire divided difference table.

The above examples also show that another method for computing divided differences may sometimes be needed. The next sections attempt to supply one.

2.6 The divided difference table as a matrix.

The divided difference table (§2.1) of f for the abscissae $\mathbf{Z}=\zeta_0,\zeta_1,\ldots,\zeta_n$ may be expressed as an $n+1\times n+1$ upper triangular matrix.†

$$\Delta f \equiv \begin{bmatrix} f(\zeta_0) & \Delta_0^1 f & \Delta_0^2 f & \cdot & \cdot & \cdot & \Delta_0^n f \\ & f(\zeta_1) & \Delta_1^1 f & \cdot & \cdot & \cdot & \Delta_1^{n-1} f \\ & & f(\zeta_2) & \cdot & \cdot & \cdot & \Delta_2^{n-2} f \\ & & & \cdot & & & \cdot \\ & & & & \cdot & & \cdot \\ & & & & & \cdot & \cdot \\ & & & & & & f(\zeta_n) \end{bmatrix} \tag{2.6.1}$$

Define also the special $n+1\times n+1$ bidiagonal matrix

$$Z \equiv \begin{bmatrix} \zeta_0 & 1 & & & & \\ & \zeta_1 & 1 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & \zeta_{n-1} & 1 \\ & & & & & \zeta_n \end{bmatrix}. \tag{2.6.2}$$

Opitz [14] calls this a "steigungsmatrix".

Theorem 2.2: "The divided difference table is a matrix function."

$$\Delta f = f(Z) \tag{2.6.3}$$

proof: The conditions for the existence of the divided difference table Δf (§2.1) and the Newton polynomial of $f(Z)$ (§1.2) are identical.

$$f(Z) = f(\zeta_0)I + \Delta_0^1 f(Z-\zeta_0 I) + \cdots + \Delta_0^n f \prod_{k=0}^{n-1}(Z-\zeta_k I)$$

Because $Z-\zeta_k I$, all k, are bidiagonal, $\prod_{k=0}^{m}(Z-\zeta_k I)$ for $m<n-1$ has $(0,n)$ element zero. The $(0,n)$ element of $\prod_{k=0}^{n-1}(Z-\zeta_k I)$ is one; so, the $(0,n)$ element of $f(Z)$ is $\Delta_0^n f$, which is also the $(0,n)$ element of Δf. Because of the pattern of dependence of the elements of the difference table, the result holds for every element of the table. □

The theorem has several important and useful consequences.

1. If $\mathbf{Z}=\{\zeta_0,\zeta_0,\ldots,\zeta_0\}$,

† $\Delta_Z f$ will be written when the set of abscissae must be emphasized. Recall that Δf, no superscript, is a matrix and $\Delta^n f$ is a scalar.

$$\Delta f = \begin{bmatrix} f(\zeta_0) & f'(\zeta_0) & \frac{1}{2!}f''(\zeta_0) & \cdot & \cdot & \cdot & \frac{1}{n!}f^{(n)}(\zeta_0) \\ & f(\zeta_0) & f'(\zeta_0) & \cdot & \cdot & \cdot & \frac{1}{(n-1)!}f^{(n-1)}(\zeta_0) \\ & & f(\zeta_0) & \cdot & \cdot & \cdot & \frac{1}{(n-2)!}f^{(n-2)}(\zeta_0) \\ & & & \cdot & & & \cdot \\ & & & & \cdot & & \cdot \\ & & & & & \cdot & \cdot \\ & & & & & & f(\zeta_0) \end{bmatrix} \tag{2.6.4}$$

This is the well-known special form for the function of a Jordan block.

2. Define $f_t(\zeta) \equiv f(t\zeta)$, then

$$\Delta f_t = f(tZ)\,. \tag{2.6.5}$$

3. Define $D \equiv \mathrm{diag}(1, t, t^2, \ldots, t^n)$, a diagonal matrix, and $t\mathbf{Z} \equiv \{t\zeta_0, t\zeta_1, \ldots, t\zeta_n\}$, then

$$\Delta_{\mathbf{Z}} f_t = D \cdot \Delta_{t\mathbf{Z}} f \cdot D^{-1}\,. \tag{2.6.6}$$

proof: $\Delta_{\mathbf{Z}} f_t \equiv \Delta f_t = f(tZ) = f(DZ_t D^{-1}) = Df(Z_t)D^{-1} = D \cdot \Delta_{t\mathbf{Z}} f \cdot D^{-1}$, where

$$Z_t \equiv \begin{bmatrix} t\zeta_0 & 1 & & & \\ & t\zeta_1 & 1 & & \\ & & \cdot & \cdot & \\ & & & t\zeta_{n-1} & 1 \\ & & & & t\zeta_n \end{bmatrix} \quad \square$$

4. When $f = \uparrow^p$, $\uparrow^p(\zeta) \equiv \zeta^p$, then

$$\Delta \uparrow^p = Z^p. \tag{2.6.7}$$

This leads immediately to algorithms that resemble those of §2.2.

Algorithm 2.3: Recursive computation of $\Delta_0^k \uparrow^p$.

1. Initialize $\Delta_0^0 \uparrow^0 = 1$, $\Delta_0^k \uparrow^0 = 0$, $k = 1, 2, \ldots, n$
2. For $p = 1, 2, \ldots$
   $\Delta_0^k \uparrow^p = \zeta_k \cdot \Delta_0^k \uparrow^{p-1} + \Delta_0^{k-1} \uparrow^{p-1}$, $k = 0, 1, \ldots, n$.
   $(\Delta_j^k f \equiv 0$ for $k < 0)$.

There is also a companion algorithm.

Algorithm 2.3': Recursive computation of $\Delta_{n-k}^k \uparrow^p$.

1. Initialize $\Delta_n^0 \uparrow^0 = 1$, $\Delta_{n-k}^k \uparrow^0 = 0$, $k = 1, 2, \ldots, n$
2. For $p = 1, 2, \ldots$
   $\Delta_{n-k}^k \uparrow^p = \zeta_{n-k} \cdot \Delta_{n-k}^k \uparrow^{p-1} + \Delta_{n-k+1}^{k-1} \uparrow^{p-1}$. $k = 0, 1, \ldots, n$.

Algorithm 2.3 produces the top row of $\Delta \uparrow^p$ for $p = 1, 2, \ldots$, while Algorithm 2.3' gives the right hand column of $\Delta \uparrow^p$.

2.7 A scaling and squaring method for the exponential.

Theorem 2.2 suggests that special properties of the function f may be used to form the divided difference table. When $f=\exp_t$, $\exp_t(\zeta)\equiv e^{t\zeta}$, consequence 2 of the previous section becomes

$$\Delta\exp_t = \exp(tZ) = e^{tZ}. \tag{2.7.1}$$

The expression on the right of (2.7.1) immediately suggests trying to power the difference table;

$$\Delta\exp_t = [\Delta\exp_{2^j t}]^{2^j}, \quad \text{for integer } j \geq 0. \tag{2.7.2}$$

If the table $\Delta\exp_{2^j t}$ is obtainable to suitable accuracy by some method for some $j>0$, while $\Delta\exp_t$ is not, a "scaling and squaring" procedure may be employed. Ward [19] suggests this in general for e^{tA}, but here its use is confined to special matrices.

The table $\Delta\exp_{2^j t}$ may be computed using the Taylor series algorithm 2.2 on $f=\exp_{2^j t}$. A relative error tolerance $\tau_j\epsilon=\epsilon e^{2^{-j}t\delta}(1+2^{-j}t\delta)/(2-e^{2^{-j}t\delta})$ may be given for Algorithm 2.2, where j is chosen as the smallest non-negative integer such that (Appendix B)

$$2^{-j}t\delta \geq 0.2209. \tag{2.7.3}$$

Algorithm 2.2 produces only the 0-th (i.e. the top) row of the divided difference table. However, it is not necessary to employ the algorithm on every row, as sufficient information is now available to fill out the rest of the table by running the recursive scheme (2.1.8) backwards.

Backfilling the table. If the divided differences $\Delta_0^k f$ for $k=0,1,\ldots,n$ are known, $\Delta_i^k f$ for $k=0,1,\ldots,n-i$ and $i=1,2,\ldots,n$ may be computed recursively by

$$\Delta_i^k f = (\zeta_{i+k}-\zeta_{i-1})\Delta_{i-1}^{k+1} f + \Delta_{i-1}^k f. \tag{2.7.4}$$

$$\begin{bmatrix} x & x & x & x & x & x \\ & 5 & 4 & 3 & 2 & 1 \\ & & 9 & 8 & 7 & 6 \\ & & & \cdot & \cdot & \cdot \\ & & & & \cdot & \cdot \\ & & & & & \cdot \end{bmatrix}$$

Fig. 2.7.1: Possible order of backfilling using (2.7.4) given 0-th row of table.

As use of the backfill scheme is contemplated only when the forward scheme (2.1.8) fails, there is no danger of introducing large errors due to subtractive cancellation. The reason for this is quite simple. If (2.1.8) fails, then (§2.3) subtractive cancellation between some $\Delta_i^k f$ and $\Delta_{i-1}^k f$ has occured. This means that $\Delta_i^k f \approx \Delta_{i-1}^k f$, at least in the first few digits. But by (2.7.4), $(\zeta_{i+k}-\zeta_{i-1})\Delta_{i-1}^{k+1} f$ must be significantly smaller in magnitude than $\Delta_{i-1}^k f$. Hence, subtractive cancellation in (2.7.4) cannot occur when subtractive cancellation occurs in (2.1.8).

A method for computing $\Delta\exp_t$ for any set of abscissae $\mathbf{Z}=\{\zeta_0,\zeta_1,\ldots,\zeta_n\}$ is now evident.

Algorithm 2.4: Scaling and squaring for $\Delta\exp_t$.

1. Find j to satisfy (2.7.3).
2. Compute the $\Delta_0^k\exp_{2^{-j}t}$, $k=0, 1, \ldots, n$, by Algorithm 2.2.
3. Backfill the table $\Delta\exp_{2^{-j}t}$ using (2.7.4).
4. Square the divided difference table matrix $\Delta\exp_{2^{-j}t}$ j times.

This method requires $O(\frac{1}{6}jn^3)$ multiplications. Most of the work is in step 4; the first three steps require only $O(n^2)$ operations. A similar algorithm may be developed from Algorithm 2.2'.

Error bound. A quite satisfactory error bound is possible for Algorithm 2.4 when all the abscissae are real. In this case, $\Delta\exp_t \geqslant 0$.† If ϵ is as defined in §2.4 and the arithmetic satisfies the following condition, also given in §2.4,

$$|fl(\sum_i a_i) - \sum_i a_i| \leqslant \epsilon \sum_i |a_i|, \tag{2.7.5}$$

the following, element by element, bound on the error in the computed divided difference table $fl(\Delta\exp_t)$ is possible (Appendix B).

$$|fl(\Delta\exp_t) - \Delta\exp_t| \leqslant \epsilon[2^j(\tau_j+1) - 1]\cdot\Delta\exp_t \tag{2.7.6}$$

In particular, the relative error for each computed divided difference in the table is bounded by

$$\epsilon[2^j(\tau_j+1) - 1]. \tag{2.7.7}$$

This indicates a loss of about j binary digits in the squaring operation. Fig. 2.5.1 gives an example of Algorithm 2.4 in the real case. Here $j=4$ and $\tau_j=3.936$. In arithmetic to seven decimal digits, the relative error in using Algorithm 2.4 is always less than $78\epsilon \leqslant 4\times10^{-5}$.

If the abscissae are complex, such uniformly satisfactory bounds are not possible. The bound analogous to (2.7.6) is

$$|fl(\Delta\exp_t) - \Delta\exp_t| \leqslant \epsilon[2^j(\tau_j+1) - 1]\cdot|\Delta\exp_{2^{-j}t}|^{2^j}. \tag{2.7.8}$$

This is satisfactory only when the elements of $|\Delta\exp_{2^{-j}t}|^{2^j}$ are not much larger than the corresponding elements of $|\Delta\exp_t|$.

When $\mathbf{Z}=\{\zeta_0, \zeta_1, \ldots, \zeta_n\}$ and $\mathbf{X}\equiv\{\mathrm{Re}(\zeta_0), \mathrm{Re}(\zeta_1), \ldots, \mathrm{Re}(\zeta_n)\}$, the integral representation (2.1.13) yields

$$|\Delta_{\zeta_0}^n\exp_t| \leqslant \Delta_{\mathrm{Re}(\zeta_0)}^n\exp_t. \tag{2.7.9}$$

This gives another bound on the error,

$$|fl(\Delta_{\mathbf{Z}}\exp_t) - \Delta_{\mathbf{Z}}\exp_t| \leqslant \epsilon[2^j(\tau_j+1) - 1]\cdot\Delta_{\mathbf{X}}\exp_t. \tag{2.7.10}$$

†Matrix inequalities $B \geqslant C$ hold element by element; $|B|$ is the matrix whose elements are the absolute values of the elements of B.

2.8 The complete divided difference table.

The previous sections have presented methods which permit the computation of the complete divided difference table for some elementary functions f. This section now outlines the entire computation.

(1) Ordering the abscissae. As discussed in §2.5, the abscissae should be ordered such that close abscissae are also close in the ordering, and well-separated abscissae are well-separated in the ordering. This is to facilitate both the use of the Taylor series algorithm 2.2 on the clusters of close abscissae and the use of the recursive formula (2.1.8) on the remainder of the table. Only a crude ordering (clustering) is needed; as best orderings of complex numbers are seldom obvious, this leeway is valuable.

If the given problem requires the abscissae be in an order where close and well-separated abscissae may interlace, none of the presented methods is sure to work well. However, if it is possible to form a divided difference table for some desireable ordering of the abscissae, then either (2.1.8) or (2.7.4) may be used with the symmetry property to construct, element by element, the required table from the one at hand.

(2) Natural clustering. If there is a simple a priori estimate of when (2.1.8) may be used safely, say $|\zeta_i-\zeta_j| > g(i-j,f)$ for some non-negative function g, the procedure becomes quite straightforward. Given an ordering of the abscissae, let all abscissae ζ_j, $\zeta_{j+1},\ldots,$ ζ_i satisfying $|\zeta_i-\zeta_j| \leqslant g(i-j,f)$ be a cluster upon which (2.1.8) must not be used. Instead, the Taylor series algorithm 2.2 and the backfill formula (2.7.4) may be employed to form the block of the divided difference table corresponding to the cluster. [If the spread of the abscissae in the cluster is so great that the series error bound (2.4.4) is excessive, the ideas of §2.6 may sometimes be employed.] Each cluster can be treated in this manner. The divided difference table will now resemble the matrix in Fig. 2.8.1.

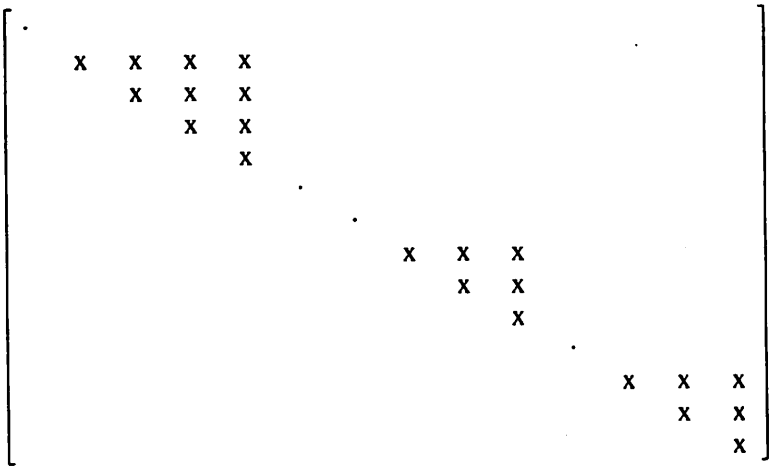

Fig. 2.8.1: Block structure of Δf corresponding to clusters.

(3) Analytic computation of the diagonal. The diagonal (0-th order divided differences) of the table may always be computed directly from the function; i.e. $\Delta^0_{i,i} f = f(\zeta_i)$ for $i=0, 1, \ldots, n$. For some functions, the first super-diagonal may also be computed analytically. For example,

abscissae	Given Divided Difference Table				
0.00	1.000	1.136	6.454E-1	1.183	4.503
0.25		1.284	1.459	6.559	4.622E+1
0.50			1.649	3.261E+1	4.572E+2
5.00				1.484E+2	4.376E+3
10.00					2.203E+4

abscissae	Correct Rearranged Divided Difference Table				
0.00	1.000	1.297	6.263	4.509E+1	4.503
0.50		1.649	3.261E+1	4.572E+2	4.622E+1
5.00			1.484E+2	4.376E+3	4.456E+2
10.00				2.203E+4	2.259E+3
0.25					1.284

abscissae	Rearranged Table Using Recursive Scheme Only				
0.00	1.000	1.298	6.262	4.509E+1	3.640
0.50		1.649	3.261E+1	4.572E+2	4.600
5.00			1.484E+2	4.376E+3	4.457E+2
10.00				2.203E+4	2.259E+3
0.25					1.284

abscissae	Rearranged Table Using Both Schemes				
0.00	1.000	1.298	6.262	4.509E+1	4.503
0.50		1.649	3.261E+1	4.572E+2	<u>4.622E+1</u>
5.00			1.484E+2	4.376E+3	4.457E+2
10.00				2.203E+4	2.259E+3
0.25					1.284

Underlined element was computed by the backfill scheme (2.7.4) using the given table and the symmetry property to obtain 4.503

Fig. 2.8.2: Rearranging the divided difference table $\Delta\exp$.

$$\Delta_i^1 \exp_t = e^{t(\zeta_i+\zeta_{i+1})/2}\frac{\sinh[t(\zeta_{i+1}-\zeta_i)/2]}{(\zeta_{i+1}-\zeta_i)/2}, \quad \zeta_i \neq \zeta_{i+1} \tag{2.8.1}$$

$$= te^{t\zeta_i}, \quad \zeta_i = \zeta_{i+1}.$$

(4) Recursive formula. The results of (2) and (3) permit the remainder of the table to be filled by the recursive formula (2.1.8) by working outwards towards the upper right. The diagram in Fig. 2.8.3 illustrates one way of proceeding.

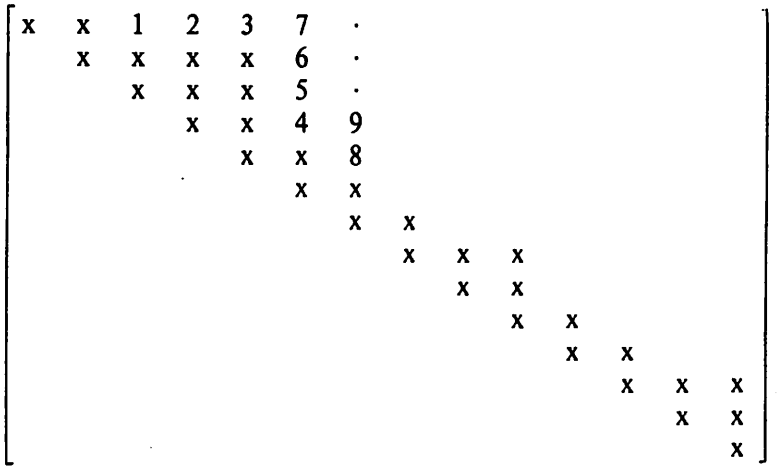

Fig. 2.8.3: Filling out the table by the recursive formula.

Summary. The above ideas are summarized in the following algorithm.

Algorithm 2.5: The divided difference table Δf.

1. Order the abscissae $\mathbf{Z}=\{\zeta_0,\zeta_1,\ldots,\zeta_n\}$ such that close abscissae are close in the ordering, well-separated abscissae are well-separated in the ordering.
2. Use an a priori estimate to determine clusters of "close" abscissae. Use Algorithm 2.2 and formula (2.7.4) (or the ideas of §2.6) to form blocks of the table corresponding to the clusters. [If $f=\exp_t$, use Algorithm 2.4.]
3. Compute all 0-th (and first) order divided differences analytically. [If $f=\exp_t$, use (2.8.1).]
4. Fill the remainder of the table by the recursive formula (2.1.8).

Adaptive computation of Δf. If the a priori estimate of step 2 is not employed, step by step monitoring of error growth for (2.1.8) is required. For example, if

$$|fl(\Delta_i^k f)-\Delta_i^k f|\leqslant\epsilon_i^k \text{ and } |fl(\Delta_{i+1}^k f)-\Delta_{i+1}^k f|\leqslant\epsilon_{i+1}^k \tag{2.8.2}$$

where $\epsilon_i^k\equiv 0$ for $k<0$, then

$$|fl(\Delta_i^{k+1}f)-\Delta_i^{k+1}f|\leqslant\epsilon_i^{k+1}\equiv\frac{\epsilon_{i+1}^k+\epsilon_i^k}{|\zeta_{i+k+1}-\zeta_i|}+\epsilon\cdot|fl(\Delta_i^{k+1}f)| \tag{2.8.3}$$

for $i=0,1,\ldots,n-k-1$ and $k=0,1,\ldots,n$.

The idea is as follows. The diagonal (and first super-diagonal) of the table is formed analytically and the initial error bounds ϵ_i^0 (ϵ_i^1) are obtained from (2.8.3). The recursive formulas (2.1.8) and (2.8.3) are now used.† If at any step, say for $\Delta_i^k f$, ϵ_i^k exceeds some prescribed error tolerance, the computed $\Delta_i^k f$ is rejected and recomputed using Algorithm 2.2 (or the ideas of §2.6). ϵ_i^k is reset to the error bound (2.4.4) of the series and the computation is resumed with formulas (2.1.8) and (2.8.3). [If $f=\exp_t$, Algorithm 2.4 may be used to get $\Delta_i^k\exp_t$ and ϵ_i^k reset according to (2.7.8).] Algorithms 2.2 and 2.4 reproduce some already computed differences as a bonus. These new values, if more accurate, may replace the old

†work outwards by diagonals or rightwards by rows to the top or upwards by columns to the right, as illustrated in Fig. 2.8.3.

ones. If ϵ_i^k is still too large, the computation may have failed; however, as only the error bound is too large, there is no assurance that the computed divided differences are in fact unacceptable. These ideas may now be summarized.

Algorithm 2.5': The divided difference table Δf with error bound.

1. Order the abscissae (as before).
2. Compute all 0-th (and first) divided differences analytically. Obtain the initial error bounds, $\epsilon_i^0 = |\Delta_i^0 f|\epsilon$ for $i=0, 1, \ldots, n$.
3. Fill the table by (2.1.8) and compute an error bound for each element by (2.8.3). If the error bound exceeds a given tolerance for some difference, recompute that difference by Algorithm 2.2 (or the ideas of §2.6); reset the error bound by (2.4.4) and resume filling the table.

Whether the clustering approach or the error monitoring approach is more efficient depends on the problem at hand. The former may fail without warning, and if the clusters are larger than required or great accuracy is not needed, the very fast recursion (2.1.8) may be avoided unnecessarily. The latter approach may indicate failure when, in fact, failure has not occurred. Also it is very fast, $O(n^2)$ operations, when (2.1.8) may be used exclusively, but can be very slow, up to $O(n^4)$ operations, when recomputation of the differences is required at each step.

The discussion of divided differences shows that for some elementary functions divided differences may be computed. This is what is required to make the Newton polynomial a practical approach for computing matrix functions. Now that forming the Newton polynomial coefficients has been considered, the remainder of this paper is devoted to other details of the computation of $f(A)$.

3. Computing the Newton Polynomial of f(A)

3.1 Complex matrices.

The previous discussion indicated that for some elementary functions f, the coefficients of the Newton polynomial for $f(A)$ may be computed. Indeed, these coefficients are just the 0-th (i.e. top) row of the divided difference table of f at the set of abscissae $\Lambda_A = \{\lambda_0, \lambda_1, \ldots, \lambda_n\}$, the eigenvalues of A.

$$\Delta f = \begin{bmatrix} \underline{f(\lambda_0)} & \underline{\Delta_0^1 f} & \underline{\Delta_0^2 f} & \underline{\Delta_0^3 f} & \underline{\Delta_0^4 f} & \underline{\Delta_0^5 f} & \underline{\Delta_0^6 f} \\ & f(\lambda_1) & \Delta_1^1 f & \Delta_1^2 f & \Delta_1^3 f & \Delta_1^4 f & \Delta_1^5 f \\ & & f(\lambda_2) & \Delta_2^1 f & \Delta_2^2 f & \Delta_2^3 f & \Delta_2^4 f \\ & & & f(\lambda_3) & \Delta_3^1 f & \Delta_3^2 f & \Delta_3^3 f \\ & & & & f(\lambda_4) & \Delta_4^1 f & \Delta_4^2 f \\ & & & & & f(\lambda_5) & \Delta_5^1 f \\ & & & & & & f(\lambda_6) \end{bmatrix}$$

Fig. 3.1.1: Newton polynomial coefficients (underlined) for $\Lambda_A = \{\lambda_0, \lambda_1, \ldots, \lambda_6\}$.

The basic algorithm to obtain $f(A)$ is now clear.

Algorithm 3.1: Newton polynomial algorithm for $f(A)$.

1. Find the eigenvalues of A.
2. Order the abscissae (eigenvalues) as described in §2.8 and form the divided difference table for f.
3. Pick out the Newton polynomial coefficients from the table and compute $f(A) = \sum_{k=0}^{n} \Delta_0^k f \prod_{j=0}^{k-1} (A - \lambda_j I)$.

This is an $O(n^4)$ procedure, the bulk of the work being in the matrix multiplications of step 3. Only three $n+1 \times n+1$ storage arrays are required: one for A, one to accumulate $f(A)$, and one for the matrix products. The divided difference table may be formed in the, as yet, unused array for $f(A)$. If error bounds $\tau_0^k \epsilon$, $k=0, 1, \ldots, n$, on the coefficients are available (§2.8),

$$\| fl[f(A)] - f(A) \| \leqslant \epsilon \sum_{k=0}^{n} \tau_0^k \| \prod_{j=0}^{k-1} (A - \lambda_j I) \| \tag{3.1.1}$$

bounds the norm of the error in the final result if the matrix products are formed accurately.

In practice, f often depends on a non-negative parameter t, $f(tA) \equiv f_t(A)$, where A remains fixed, but many different values of t are given. Only the coefficients of the Newton polynomial for $f_t(A)$ depend on t, so, it seems worthwhile to study how the computation of $f_t(A)$ can be arranged to lessen the work at each value of t. Such economies, and others, are now discussed in order that a more practical algorithm may be presented.

(1) The Schur form. When n is not large, say $n \leqslant 100$, the eigenvalues of A are usually found by the QR algorithm which performs a sequence of unitary similarity transformations on A in order to transform it into upper (lower) triangular form T. Wilkinson [21] gives a treatment of

the theory behind the *QR* algorithm; it is implemented in the EISPACK [17] package of computer algorithms. T is the Schur form of A, and the eigenvalues of A appear on the diagonal of T.

$$A = P^*\cdot T\cdot P \quad \text{or} \quad T = P\cdot A\cdot P^*, \quad P^* = P^{-1} \tag{3.1.2}$$

From (1.1.2),

$$f_t(A) = P^*\cdot f_t(T)\cdot P \quad \text{or} \quad f_t(T) = P\cdot f_t(A)\cdot P^*. \tag{3.1.3}$$

Because $\Lambda_A\ (=\Lambda_T)$, P and T are independent of the parameter t, the Schur factorization need be performed only once at the beginning of the computation for $f_t(A)$. $f_t(T)$ may be computed at each step and $f_t(A)$ formed by (3.1.3) only when required. Thus, the problem may be reduced to computing a function of a triangular matrix.

(2) Ordering the eigenvalues. Typically, the *QR* algorithm produces a matrix T whose eigenvalues appear in almost monotonically decreasing order by absolute value along the diagonal. Parlett [16] has developed routines for reordering the eigenvalues of a triangular matrix into any desired order by using unitary similarity transformations. Hence, T's eigenvalues may be put in any order and the transformations necessary to achieve this may be absorbed into P.

(3) Products of triangular matrices. The problem is now to compute

$$f_t(T) = \sum_{k=0}^{n} \Delta_0^k f_t \prod_{j=0}^{k-1} (T-\lambda_j I)\,. \tag{3.1.4}$$

The matrix factors $\prod_{j=0}^{k-1}(T-\lambda_j I)$ for $k=0,1,\ldots,n$ are all upper triangular; and thus, $f_t(T)$ is also. The matrix products need $O(\frac{1}{6}n^3)$ operations per multiplication, as opposed to $O(n^3)$ for full matrices.

The matrix multiplications for the factors $\prod_{j=0}^{k-1}(T-\lambda_j I)$ for $k=2,3,\ldots,n$ do not depend on the parameter t and may, if storage is available, be performed once and stored for future use at each t-step. This requires an additional $\frac{1}{2}(n^3+n^2-n-2)$ storage locations. Then at each t-step, only $\frac{1}{2}(n^3+3n^2+2n)$ multiplications are required to form $f_t(T)$, once the coefficients of the polynomial are known. As shown below, this figure for storage locations can be reduced considerably. If the reduction is still not sufficient, the matrix multiplications may be performed at each t-step.

(4) Analytic computation of the diagonal. The diagonal elements of $f_t(T)$ can be computed analytically. As $T_{j,j}=\lambda_j$,

$$f_t(T)_{j,j} = f_t(\lambda_j) \quad \text{for } j=0,1,\ldots,n. \tag{3.1.5}$$

For some f, the first super-diagonal of $f_t(T)$ can also be formed analytically. For example, for $j=0,1,\ldots,n-1$

$$\exp_t(T)_{j,j+1} = T_{j,j+1}\Delta_j^1\exp_t = T_{j,j+1}e^{t(\lambda_j+\lambda_{j+1})/2}\frac{\sinh[t(\lambda_{j+1}-\lambda_j)/2]}{(\lambda_{j+1}-\lambda_j)/2}, \quad \lambda_j\neq\lambda_{j+1} \tag{3.1.6}$$

$$= T_{j,j+1}te^{t\lambda_j}, \quad \lambda_j=\lambda_{j+1}.$$

This reduces the additional storage requirements to $\frac{1}{2}(n^2-n)$ locations and the number of multiplications at each t-step (exclusive of the above evaluations and the coefficient evaluations) to $\frac{1}{2}(n^3+n^2)$.

(5) Special form of the $\prod_{j=0}^{k-1}(T-\lambda_j I)$. The matrix factors $\prod_{j=0}^{k-1}(T-\lambda_j I)$ for $k=2,3,\ldots,n$ are

not full upper triangular matrices. This is one of the features that makes the Newton polynomial so very attractive and may be demonstrated as follows. Define $S^{(1)} \equiv T - \lambda_0 I$. $S^{(1)}$ has (0,0) element $S^{(1)}_{0,0} = 0$. $S^{(2)} \equiv (T-\lambda_0 I)(T-\lambda_1 I)$ has not only $S^{(2)}_{0,0} = 0$, but also $S^{(2)}_{0,1} = 0$ and $S^{(2)}_{1,1} = 0$. In general, the upper triangular matrix

$$S^{(k)} \equiv \prod_{j=0}^{k-1} (T - \lambda_j I) \quad \text{has} \quad S^{(k)}_{i,j} = 0, \quad j < k. \tag{3.1.7}$$

$$\begin{bmatrix} \cdot & & & x & x & x & x & x \\ & \cdot & & x & x & x & x & x \\ & & \cdot & x & x & x & x & x \\ & & & \cdot & x & x & x & x & x \\ & & & x & x & x & x & x \\ & & & & x & x & x & x \\ & & & & & x & x & x \\ & & & & & & x & x \\ & & & & & & & x \end{bmatrix}$$

Fig. 3.1.2: Matrix $S^{(4)}$ when $n = 8$.

With this knowledge, only

$$\begin{aligned} &\tfrac{1}{3}(n^3 - n) && \text{storage locations} \\ &\tfrac{1}{3}n^3 + \tfrac{1}{2}n^2 + \tfrac{1}{6}n && \text{multiplications} \end{aligned} \tag{3.1.8}$$

are needed. This nice property requires that the divided difference abscissae used to form the polynomial coefficients be in exactly the same order as the eigenvalues along the diagonal of T. The remarks under (2) show that the eigenvalues may be ordered to best advantage.

Summary. The complete algorithm may now be summarized.

Algorithm 3.2: Newton polynomial algorithm for $f_t(A)$, with ample storage.

1. Use the QR algorithm to compute T and P.
2. Use Parlett's swap routines to order the eigenvalues of T to aid in forming the divided difference table of f; update T and P.
3. Form and store the matrix products $\prod_{j=0}^{k-1}(T-\lambda_j I)$ for $k=2,3,\ldots,n$ using the economies indicated in (4) and (5).

At each t-step,

4. Compute the Newton polynomial coefficients (§2).
5. Use the results of steps 3 and 4 to compute the off-diagonal elements of $f_t(T)$,
$$f_t(T)_{i,j} = \sum_{k=1}^{n} \Delta_0^k f_t \prod_{l=0}^{k-1} (T-\lambda_l I)_{i,j} \quad \text{for } i < j.$$
6. Fill the diagonal (and first super-diagonal) of $f_t(T)$ analytically; e.g. use (3.1.5) and (3.1.6).
7. If desired, transform $P^* \cdot f_t(T) \cdot P = f_t(A)$.

The entire algorithm requires, with care, about $\frac{1}{3}n^3+3n^2$ storage locations, and $O(\frac{1}{3}n^3)$ operations per t-step (exclusive of step 7).

3.2 Real matrices.

All the remarks of the previous section apply equally well when the matrix A is real. However, if the methods presented there are used without change, complex arithmetic is usually required. This section will discuss how complex arithmetic can be avoided when computing $f_t(A)$ for real A.

(1) The real Schur form. When A is real, its eigenvalues are either real or complex conjugate pairs. The QR algorithm may be used to reduce A to a nearly upper triangular form T by use of orthogonal (real) similarity transformations [21].

$$A = P^T \cdot T \cdot P \quad \text{or} \quad T = P \cdot A \cdot P^T, \quad P^T = P^{-1} \text{ (real)} \tag{3.2.1}$$

T is real and upper triangular except for some non-consecutive non-zero elements on its first sub-diagonal.

$$T = \begin{bmatrix} x & x & x & x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x & x & x & x \\ & & x & x & x & x & x & x & x & x & x \\ & & x & x & x & x & x & x & x & x & x \\ & & & & x & x & x & x & x & x & x \\ & & & & & x & x & x & x & x & x \\ & & & & & & x & x & x & x & x \\ & & & & & & x & x & x & x & x \\ & & & & & & & & x & x & x \\ & & & & & & & & & x & x \\ & & & & & & & & & x & x \end{bmatrix}$$

Fig. 3.2.1: Example of nearly upper triangular real Schur form.

We may speak of T as being block upper triangular, with 1×1 or 2×2 blocks along its diagonal. The 1×1 blocks are the real eigenvalues of A (and T); that is if $T_{i,i}$ is a 1×1 block, $\lambda_i = T_{i,i}$. The 2×2 blocks correspond to the conjugate pair eigenvalues. Let $\begin{matrix} T_{i,i} & T_{i,i+1} \\ T_{i+1,i} & T_{i+1,i+1} \end{matrix}$ be such a block. Orthogonal P may be chosen such that $T_{i,i} = T_{i+1,i+1}$, and this normalization will be assumed. Then the corresponding conjugate pair eigenvalues λ_i and $\lambda_{i+1} = \bar{\lambda}_i$ are the roots of

$$\lambda^2 - 2T_{i,i}\lambda + (T_{i,i}^2 - T_{i+1,i}T_{i,i+1}) = 0. \tag{3.2.2}$$

It follows that $T_{i,i} = T_{i+1,i+1} = \mathrm{Re}(\lambda_i)$ and $\mathrm{Im}(\lambda_i) = \pm[-T_{i+1,i}T_{i,i+1}]^{1/2}$.

The product of two real Schur matrices of the same structure is again a real Schur matrix of that structure. Hence, $f_t(T)$ is a Schur matrix with the same structure as T. Since

$$f_t(A) = P^T \cdot f_t(T) \cdot P \quad \text{or} \quad f_t(T) = P \cdot f_t(A) \cdot P^T, \tag{3.2.3}$$

$f_t(T)$ may be computed in order to avoid use of the full matrix A.

(2) The real Newton polynomial. The set of eigenvalues Λ_A now consists of real and conjugate pair elements. Generally, the divided differences formed from Λ_A and f_t will be complex; however, because of the symmetry property there are some important relationships.

$$\Delta f = \begin{bmatrix} f(\lambda_0) & \underline{\Delta_0^1 f} & \underline{\Delta_0^2 f} & \Delta_0^3 f & \underline{\Delta_0^4 f} \\ & \overline{f(\lambda_0)} & \Delta_1^1 f & \Delta_1^2 f & \Delta_1^3 f \\ & & \underline{f(\lambda_1)} & \Delta_2^1 f & \underline{\Delta_2^2 f} \\ & & & f(\lambda_2) & \underline{\Delta_3^1 f} \\ & & & & \overline{f(\lambda_2)} \end{bmatrix}$$

$$\Delta \bar{f} = \begin{bmatrix} \overline{f(\lambda_0)} & \underline{\Delta_0^1 f} & \underline{\Delta_0^2 f} & \overline{\Delta_0^3 f} & \underline{\Delta_0^4 f} \\ & f(\lambda_0) & \overline{\Delta_1^1 f} & \overline{\Delta_1^2 f} & \overline{\Delta_1^3 f} \\ & & \underline{f(\lambda_1)} & \overline{\Delta_2^1 f} & \underline{\Delta_2^2 f} \\ & & & \overline{f(\lambda_2)} & \underline{\Delta_3^1 f} \\ & & & & f(\lambda_2) \end{bmatrix}$$

Fig. 3.2.2: Divided difference table for $\Lambda_A = \{\lambda_0, \bar{\lambda}_0, \lambda_1, \lambda_2, \bar{\lambda}_2\}$ is conjugate of table for $\bar{\Lambda}_A = \{\bar{\lambda}_0, \lambda_0, \lambda_1, \bar{\lambda}_2, \lambda_2\}$. (real elements underlined)

The above example shows that $(\Delta^4 f)(\lambda_0, \bar{\lambda}_0, \lambda_1, \lambda_2, \bar{\lambda}_2) = (\Delta^4 f)(\bar{\lambda}_0, \lambda_0, \lambda_1, \bar{\lambda}_2, \lambda_2)$ is real, while $(\Delta^3 f)(\lambda_0, \bar{\lambda}_0, \lambda_1, \lambda_2) = (\Delta^3 f)(\bar{\lambda}_0, \lambda_0, \lambda_1, \bar{\lambda}_2)$. This suggests forming the Newton polynomial of $f_i(T)$ using both the given ordering and its conjugate ordering, and then averaging the two resulting polynomials. Using the abscissae of the above example,

$$\begin{aligned} f_i(T) = f(\lambda_0)I &+ \Delta_0^1 f(T-\lambda_0 I) + \Delta_0^2 f(T-\lambda_0 I)(T-\bar{\lambda}_0 I) + \Delta_0^3 f(T-\lambda_0 I)(T-\bar{\lambda}_0 I)(T-\lambda_1 I) \\ &+ \Delta_0^4 f(T-\lambda_0 I)(T-\bar{\lambda}_0 I)(T-\lambda_1 I)(T-\lambda_2 I) \\ f_i(T) = f(\bar{\lambda}_0)I &+ \Delta_0^1 f(T-\bar{\lambda}_0 I) + \Delta_0^2 f(T-\bar{\lambda}_0 I)(T-\lambda_0 I) + \overline{\Delta_0^3 f}(T-\bar{\lambda}_0 I)(T-\lambda_0 I)(T-\lambda_1 I) \\ &+ \Delta_0^4 f(T-\bar{\lambda}_0 I)(T-\lambda_0 I)(T-\lambda_1 I)(T-\bar{\lambda}_2 I), \end{aligned}$$

and then the real Newton polynomial is

$$\begin{aligned} f_i(T) = \mathrm{Re}[f(\lambda_0)]I &+ \Delta_0^1 f[T-\mathrm{Re}(\lambda_0)I] + \Delta_0^2 f(T-\lambda_0 I)(T-\bar{\lambda}_0 I) \\ &+ \mathrm{Re}(\Delta_0^3 f)(T-\lambda_0 I)(T-\bar{\lambda}_0 I)(T-\lambda_1 I) \\ &+ \Delta_0^4 f(T-\lambda_0 I)(T-\bar{\lambda}_0 I)(T-\lambda_1 I)[T-\mathrm{Re}(\lambda_2)I] \end{aligned} \qquad (3.2.4)$$

The coefficients of the real Newton polynomial are just the real parts of the coefficients of the general Newton polynomial.

(3) Ordering the abscissae. In order for T to remain real, the conjugate pair eigenvalues must be ordered together. Parlett [16] has also developed routines for swapping the 1×1 and 2×2 blocks along the diagonal of the real Schur T by orthogonal similarity transformations; so, the eigenvalue pairs may be ordered as desired. If some of the eigenvalues have large imaginary parts, the abscissae for the divided differences cannot be taken in the same order without making a reasonable clustering impossible. It is suggested that the conjugate pair abscissae be ordered separately as in Fig. 3.2.3. The idea is that if λ_i and λ_j are close, so are $\bar{\lambda}_i$ and $\bar{\lambda}_j$. The differences giving the Newton coefficients no longer appear in the top row of the table, but go almost diagonally up to the right.

(4) Products of real Schur matrices. The matrix factors, e.g. $S^{(4)} \equiv (T-\lambda_0 I)(T-\bar{\lambda}_0 I)(T-\lambda_1 I)[T-\mathrm{Re}(\lambda_2)I]$, are all real Schur matrices; so their formation requires little more work than for triangular matrices. Also, they exhibit the same special

$$\Delta f = \begin{bmatrix} f(\bar{\lambda}_2) & \Delta_0^1 f & \Delta_0^2 f & \Delta_0^3 f & \underline{\Delta_0^4 f} \\ & f(\bar{\lambda}_0) & \underline{\Delta_1^1 f} & \underline{\Delta_1^2 f} & \underline{\Delta_1^3 f} \\ & & \underline{f(\lambda_0)} & \Delta_2^1 f & \Delta_2^2 f \\ & & & f(\lambda_1) & \Delta_3^1 f \\ & & & & f(\lambda_2) \end{bmatrix}$$

Fig. 3.2.3: Divided difference table for conjugate pairs, abscissae $\Lambda_A = \{\bar{\lambda}_2, \bar{\lambda}_0, \lambda_0, \lambda_1, \lambda_2\}$.
(Newton coefficients are underlined)

property, the zeroing out of columns, that was seen in (5) of §3.1. In (3.2.4), $S^{(2)} \equiv (T-\lambda_0 I)(T-\bar{\lambda}_0 I)$ has $S^{(2)}_{i,j}=0$ for $j<2$ $(i,j=0,1,\ldots,4)$; $S^{(3)} \equiv (T-\lambda_0 I)(T-\bar{\lambda}_0 I)(T-\lambda_1 I)$ has $S^{(3)}_{i,j}=0$ for $j<3$; and $S^{(4)}$ has $S^{(4)}_{i,j}=0$ for $j<3$. The slight difference from the purely triangular case of §3.1 is due to the presence of the 2×2 blocks.

(5) Analytic computation of the diagonal blocks. As before, the diagonal blocks of $f_t(T)$ may be computed analytically. If $T_{i,i}=\lambda_i$ is a 1×1 block,

$$f_t(T)_{i,i} = f_t(\lambda_i)\,. \tag{3.2.5}$$

If $\begin{matrix} T_{i,i} & T_{i,i+1} \\ T_{i+1,i} & T_{i+1,i+1} \end{matrix}$ where $T_{i,i}=T_{i+1,i+1}=\mathrm{Re}(\lambda_i)$ is a 2×2 block corresponding to the eigenvalues $\lambda_i, \lambda_{i+1}=\bar{\lambda}_i$,

$$f_t(T)_{i,i} = f_t(T)_{i+1,i+1} = \mathrm{Re}[f(\lambda_i)] \tag{3.2.6}$$

and

$$\begin{aligned} f_t(T)_{i,i+1} &= T_{i,i+1}\cdot \mathrm{Im}[f_t(\lambda_i)]/\mathrm{Im}(\lambda_i) \\ f_t(T)_{i+1,i} &= T_{i+1,i}\cdot \mathrm{Im}[f_t(\lambda_i)]/\mathrm{Im}(\lambda_i)\,. \end{aligned} \tag{3.2.7}$$

In particular, if $f_t=\exp_t$ and $\lambda_i \equiv \alpha_i + i\beta_i$, then

$$\begin{aligned} \exp_t(T)_{i,i} &= \exp_t(T)_{i+1,i+1} = e^{t\alpha_i}\cos t\beta_i \\ \exp_t(T)_{i,i+1} &= T_{i,i+1} e^{t\alpha_i}(\sin t\beta_i)/\beta_i \\ \exp_t(T)_{i+1,i} &= T_{i+1,i} e^{t\alpha_i}(\sin t\beta_i)/\beta_i\,; \end{aligned} \tag{3.2.8}$$

and if $\lambda_i = T_{i,i}$ and $\lambda_{i+1} = T_{i+1,i+1}$ are two adjacent distinct real eigenvalues,

$$\exp_t(T)_{i,i+1} = T_{,i+1} e^{t(\lambda_i+\lambda_{i+1})/2} \frac{\sinh[t(\lambda_{i+1}-\lambda_i)/2]}{(\lambda_{i+1}-\lambda_i)}\,. \tag{3.2.9}$$

With the modifications suggested above, Algorithm 3.2 may be carried over directly to the case of real matrices A. The arithmetic is now real, except for the divided difference table formation. The only complication is that special attention must be paid to the peculiar form of T.

4. Use of the Newton Polynomial Method

4.1 Other methods for computing f(A).

There are many possible ways to compute $f(A)$. Moler and Van Loan [13] discuss nineteen different ways to compute $\exp_t(A)$; but unfortunately, many of the methods are unsatisfactory for numerical computation, and all of them prove unsatisfactory in some cases. Nevertheless, a combination of methods may prove the best way to proceed. One such possibility will be explored briefly here.

The Newton polynomial method presented in §§3.1 and 3.2 is not intended as a general routine for computing $f_t(A)$, but as a part of a larger program. When the matrix A has $n=100$, or even $n=50$, the Newton polynomial approach by itself is likely to be very costly. Sufficient fast storage for the matrix products is unlikely to be available (for $n=50$, $\frac{1}{3}n^3=41,667$; for $n=100$, $\frac{1}{3}n^3=333,333$) and $O(n^4)$ operations will be required for each value of t. Only when desired accuracy can be obtained from just the first few terms of the polynomial can this approach be considered attractive.

Parlett [15] has proposed a fast recursive method for computing $f_t(A)$ using the fact that A and $f_t(A)$ commute. In brief, A is transformed to an upper triangular Schur matrix T and the diagonal of $f_t(T)$ is obtained directly, as in (3.1.5). The off-diagonal elements of $f_t(T)$ may be solved for uniquely by working outwards super-diagonal by super-diagonal in the equation

$$T\cdot f_t(T) - f_t(T)\cdot T = 0. \tag{4.1.1}$$

The general recursion is

$$T_{i,i}f_t(T)_{i,j} - f_t(T)_{i,j}T_{j,j} = \sum_{k=0}^{j-i-1}[f_t(T)_{i,i+k}T_{i+k,j} - T_{i,j-k}f_t(T)_{j-k,j}] \tag{4.1.2}$$

where $i<j$. The pattern of dependence here is exactly the same as that described in §2.1. In fact when $T=Z$, the "steigungsmatrix" of §2.6, the recursion (4.1.2) reduces to recursion (2.1.8), giving another way to define the divided difference table matrix.

Recursion (4.1.2) requires only $O(\frac{1}{6}n^3)$ multiplications to compute $f_t(T)$ and storage for just the three matrices T, $f_t(T)$ and the transformation matrix P where $T=PAP^*$. When T is a real Schur matrix, (4.1.2) may be interpreted as a block matrix equation.

Formula (4.1.2) fails numerically whenever T has confluent or nearly confluent eigenvalues. Kagstrom [8] points this out in his analysis of this method. However since the diagonal elements of T may be ordered as desired (§3.1), the eigenvalues may be arranged such that close ones are clustered in the ordering along T's diagonal and well-separated ones are well-separated in the ordering. If a suitable clustering is decided upon, the blocks of $f_t(T)$ corresponding to the eigenvalue clusters may be computed by the Newton polynomial method; the remainder of $f_t(T)$ may be formed by the fast recursion (4.1.2). The basic idea is exactly the same as that proposed in (2) of §2.8 for divided difference tables.

Finally, the discussion to this point has assumed that the given data, i.e. the matrix A or A's eigenvalues, are exact. However, this is seldom true. For example, A is often determined from experiment, and the QR Algorithm produces a matrix T which is correct for some slightly perturbed matrix. So, we actually seek to compute $f(\tilde{A})$ for some $\tilde{A}$ close to A. The question of the condition of the problem, that is how sensitive $f(A)$ is to small perturbations in A, is important as we wish $f(\tilde{A})$ to be close to $f(A)$. The condition is not dependent on the method selected to compute $f(A)$; however, the reliability of the computed solution depends strongly upon the condition. Kagstrom [9] develops some criteria for deciding on the condition of $\exp_t(A)$.

Appendix

A. Error bound for Algorithm 2.2.

In order that the bounds to be developed remain simple in form, the arithmetic will be assumed to satisfy

$$|fl(\sum_i a_i) - \sum_i a_i| \leqslant \epsilon \sum_i |a_i| \tag{A.1}$$

where $\epsilon \leqslant 1.06 \times$ (machine precision) and $fl(g)$ is the computed floating-point value of the expression g. Condition (A.1) holds when double precision accumulation of sums and inner products is available.† The important simplification in this case is that the error bound on the sum is independent of the number of quantities summed. The leeway given in ϵ allows all error terms containing powers of ϵ to be absorbed into terms linear in ϵ.

Algorithm 2.2 is based on Theorem 2.1,

$$\Delta_0^n f = \sum_{p=0}^{\infty} \alpha_{n+p} \Delta_0^n \uparrow_a^{n+p}, \tag{A.2}$$

where $\uparrow_a^j(\zeta) \equiv (\zeta - a)^j$ and where the set of abscissae $\mathbf{Z} \equiv \{\zeta_0, \zeta_1, \ldots, \zeta_n\}$ is within the circle of convergence of the power series $f = \sum_{j=0}^{\infty} \alpha_j \uparrow_a^j$ about the expansion point $\zeta = a$. Let

$$\delta \equiv \max_i |\zeta_i - a| . \tag{A.3}$$

We first develop bounds on the divided differences of the power functions $\Delta_0^k \uparrow_a^{p+k}$, $k=0, 1, \ldots, n$, and on the error in using Algorithm 2.1. These results are then used to bound the error in computing (A.2) by Algorithm 2.2.

Lemma 1: For $p=1, 2, \ldots$

$$|\Delta_0^k \uparrow_a^{p+k}| \leqslant \frac{(p+k)!\,\delta^p}{k!\,p!}, \quad k=0, 1, \ldots, n. \tag{A.4}$$

proof: For $p=1$, $\Delta_0^k \uparrow_a^{k+1} = \sum_{i=0}^{k} (\zeta_i - a)$ and so

$$|\Delta_0^k \uparrow_a^{k+1}| \leqslant (k+1)\delta, \quad k=0, 1, \ldots, n.$$

If for some $p \geqslant 1$

$$|\Delta_0^k \uparrow_a^{p+k-1}| \leqslant \frac{(p+k-1) \cdots (k+1)}{(p-1)!} \delta^{p-1}$$

for $k=0, 1, \ldots, n$, then since $\Delta_0^k \uparrow_a^{p+k} = \sum_{i=0}^{k} (\zeta_i - a) \cdot \Delta_0^i \uparrow_a^{p+i-1}$

$$|\Delta_0^k \uparrow_a^{p+k}| \leqslant \frac{\delta^p}{(p-1)!} \sum_{i=0}^{k} [(p+i-1) \cdots (i+1)] = \frac{(p+k) \cdots (k+1)}{p!} \delta^p$$

for $k=0, 1, \ldots, n$. □

†see Wilkinson [20] for a general treatment of rounding error analysis.

Lemma 2: If for $p=1,2,\ldots$ the $\Delta_0^{k\uparrow}{}_a^{p+k}$ are computed according to Algorithm 2.1, then for $k=1,2,\ldots,n$

$$|fl(\Delta_0^{k\uparrow}{}_a^{p+k}) - \Delta_0^{k\uparrow}{}_a^{p+k}| \leqslant \frac{(p+k)!\,\delta^p}{k!\,(p-1)!}\epsilon \tag{A.5}$$

proof: By condition (A.1) for $p=1$,

$$|fl(\Delta_0^{k\uparrow}{}_a^{k+1}) - \Delta_0^{k\uparrow}{}_a^{k+1}| \leqslant \epsilon\sum_{i=0}^{k}|\zeta_i - a| \leqslant (k+1)\delta\epsilon, \quad k=0,1,\ldots,n.$$

If for some $p \geqslant 1$,

$$|fl(\Delta_0^{k\uparrow}{}_a^{p+k-1}) - \Delta_0^{k\uparrow}{}_a^{p+k-1}| \leqslant \frac{(p+k-1)\cdots(k+1)}{(p-2)!}\delta^{p-1}\epsilon$$

for $k=0,1,\ldots,n$, then

$$\begin{aligned}|fl(\Delta_0^{k\uparrow}{}_a^{p+k}) - \Delta_0^{k\uparrow}{}_a^{p+k}| &\leqslant |fl[\sum_{i=0}^{k}(\zeta_i - a)fl(\Delta_0^{i\uparrow}{}_a^{p+i-1})] - \sum_{i=0}^{k}(\zeta_i - a)fl(\Delta_0^{i\uparrow}{}_a^{p+i-1})| \\ &\quad + \sum_{i=0}^{k}|\zeta_i - a||fl(\Delta_0^{i\uparrow}{}_a^{p+i-1}) - \Delta_0^{i\uparrow}{}_a^{p+i-1}| \\ &\leqslant \epsilon\delta^p\left[\frac{1}{(p-1)!} + \frac{1}{(p-2)!}\right]\sum_{i=0}^{k}[(p+i-1)\cdots(i+1)] \\ &= \frac{(p+k)\cdots(k+1)}{(p-1)!}\delta^p\epsilon\end{aligned}$$

for $k=0,1,\ldots,n$. □

Theorem: If the arithmetic satisfies condition (A.1), then the error in using Algorithm 2.2 satisfies

$$|fl(\Delta_0^n f) - \Delta_0^n f| \leqslant \frac{\epsilon}{n!}\sum_{p=0}\frac{(p+1)(n+p)!|\alpha_{n+p}|\delta^p}{p!}. \tag{A.6}$$

proof: Let $\Delta_0^n s_{n+m} \equiv \sum_{p=0}^{m}\alpha_{n+p}\Delta_0^{n\uparrow}{}_a^{n+p}$ be the partial sums of (A.2). The error may be bounded by three terms,

$$\begin{aligned}|fl(\Delta_0^n f) - \Delta_0^n f| &\leqslant |fl(\Delta_0^n f) - \sum_{p=0}^{m}\alpha_{n+p}fl(\Delta_0^{n\uparrow}{}_a^{n+p})| \\ &\quad + |\sum_{p=0}^{m}\alpha_{n+p}fl(\Delta_0^{n\uparrow}{}_a^{n+p}) - \sum_{p=0}^{m}\alpha_{n+p}\Delta_0^{n\uparrow}{}_a^{n+p}| + |\Delta_0^n s_{n+m} - \Delta_0^n f|.\end{aligned}$$

In the first term, $fl(\Delta_0^n f) = fl[\sum_{p=0}^{m}\alpha_{n+p}fl(\Delta_0^{n\uparrow}{}_a^{n+p})]$ and so by Lemma 1 and condition (A.1),

$$\begin{aligned}|fl(\Delta_0^n f) - \sum_{p=0}^{m}\alpha_{n+p}fl(\Delta_0^{n\uparrow}{}_a^{n+p})| &\leqslant \epsilon\sum_{p=0}^{m}|\alpha_{n+p}||fl(\Delta_0^{n\uparrow}{}_a^{n+p})| \\ &\leqslant \frac{\epsilon}{n!}\sum_{p=0}^{m}\frac{(n+p)!|\alpha_{n+p}|\delta^p}{p!}.\end{aligned}$$

By Lemma 2 and condition (A.1), the second term satisfies

$$\left|\sum_{p=0}^{m}\alpha_{n+p}fl(\Delta_0^{n\uparrow}{}_a^{n+p})-\sum_{p=0}^{m}\alpha_{n+p}\Delta_0^{n\uparrow}{}_a^{n+p}\right| \leqslant \sum_{p=0}^{m}|\alpha_{n+p}|\,|fl(\Delta_0^{n\uparrow}{}_a^{n+p})-\Delta_0^{n\uparrow}{}_a^{n+p}|$$

$$\leqslant \frac{\epsilon}{n!}\sum_{p=1}^{m}\frac{(n+p)!|\alpha_{n+p}|\delta^p}{(p-1)!}.$$

Finally, m may be chosen so large that the truncation error is as small as we like, and so it may be considered insignificant compared with the round-off error bounds. Adding the bounds gives (A.6). □

Corollary: If $f=\exp$, $\exp_t(\zeta)\equiv e^{t\zeta}$, then the error in using Algorithm 2.2 satisfies

$$|fl(\Delta_0^n\exp_t)-\Delta_0^n\exp_t| \leqslant \epsilon e^{t\delta}(1+t\delta)\frac{t^n|e^{ta}|}{n!}. \tag{A.7}$$

proof: For $f=\exp_t$, $\alpha_{n+p}=t^{n+p}e^{ta}/(n+p)!$. Inserting this into (A.6) yields

$$|fl(\Delta_0^n\exp_t)-\Delta_0^n\exp_t| \leqslant \epsilon\frac{t^n|e^{ta}|}{n!}\sum_{p=0}^{m}\frac{(p+1)}{p!}t^p\delta^p$$

$$\leqslant \epsilon e^{t\delta}(1+t\delta)\frac{t^n|e^{ta}|}{n!}. \quad \square$$

B. Error bound for Algorithm 2.4.

Algorithm 2.4 computes

$$\Delta\exp_t = [\Delta\exp_{2^{-j}t}]^{2^j} \tag{B.1}$$

given that $\Delta\exp_{2^{-j}t}$ has already been computed according to Algorithm 2.2. A bound of the form†

$$|fl(\Delta\exp_t)-\Delta\exp_t| \leqslant \epsilon\tau|\exp_{2^{-j}t}|^{2^j} \tag{B.2}$$

is possible for some $\tau>0$ and integer $j\geqslant 0$. In the event that the divided difference abscissae are real, every element of $\Delta\exp_t$ and $\Delta\exp_{2^{-j}t}$ is non-negative; and so the bound (B.2) becomes a relative error bound

$$|fl(\Delta\exp_t)-\Delta\exp_t| \leqslant \epsilon\tau\Delta\exp_t. \tag{B.3}$$

Bound (B.2) will be developed here under the same conditions used in Appendix A.

Lemma: If $t\delta<\log 2$, the relative error in each element of $\Delta\exp_t$ computed by Algorithm 2.2 is bounded by

$$\epsilon e^{t\delta}\frac{1+t\delta}{2-e^{t\delta}}. \tag{B.4}$$

proof: From equation (A.2) with $\alpha_{n+p}=t^{n+p}e^{ta}/(n+p)!$,

$$\Delta_0^n\exp_t = t^n e^{ta}\sum_{p=0}^{\infty}\frac{t^p\Delta_0^{n\uparrow}{}_a^{n+p}}{(n+p)!};$$

†Matrix inequalities of the form $B\geqslant C$ hold element by element; the matrix $|B|$ is the matrix whose elements are the absolute values of the elements of B.

and so by (A.4)

$$|\Delta_0^n \exp_t| = t^n |e^{ta}| \left| \frac{1}{n!} + \sum_{p=1}^{\infty} \frac{t^p \Delta_0^{n\uparrow a^{n+p}}}{(n+p)!} \right|$$

$$\geqslant \frac{t^n |e^{ta}|}{n!} \left[1 - \sum_{p=1}^{\infty} \frac{t^p \delta^p}{p!} \right] = \frac{t^n |e^{ta}|}{n!} (2 - e^{t\delta}) .$$

When $t\delta < \log 2$, inequality (A.7) gives

$$|fl(\Delta_0^n \exp_t) - \Delta_0^n \exp_t| \leqslant \epsilon e^{t\delta} \frac{1+t\delta}{2-e^{t\delta}} |\Delta_0^n \exp_t|$$

where the quantity

$$\epsilon e^{t\delta} \frac{1+t\delta}{2-e^{t\delta}}$$

bounds the relative error in using Algorithm 2.2. This bound is independent of n and so holds for every element of the divided difference table. □

Under the condition (A.1), the operation of squaring a matrix B satisfies the condition

$$|fl(B^2) - B^2| \leqslant \epsilon |B|^2 . \tag{B.5}$$

Combining this with the lemma yields the sought for error bound.

Theorem: For any non-negative integer j such that $2^{-j} t\delta < \log 2$, the error in using Algorithm 2.4 satisfies

$$|fl(\Delta \exp_t) - \Delta \exp_t| \leqslant \epsilon [2^j(\tau_j + 1) - 1] |\Delta \exp_{2^{-j}t}|^{2^j} \tag{B.6}$$

where

$$\tau_j \equiv e^{2^{-j} t\delta} \frac{1 + 2^{-j} t\delta}{2 - e^{2^{-j} t\delta}} . \tag{B.7}$$

proof: From the lemma,

$$|fl(\Delta \exp_{2^{-j}t}) - \Delta \exp_{2^{-j}t}| \leqslant \epsilon \tau_j |\Delta \exp_{2^{-j}t}| .$$

Then the first matrix squaring yields

$$\begin{aligned} |fl(\Delta \exp_{2^{-j+1}t}) - \Delta \exp_{2^{-j+1}t}| &\leqslant |fl(\Delta \exp_{2^{-j+1}t}) - [fl(\Delta \exp_{2^{-j}t})]^2| \\ &\quad + |[fl(\Delta \exp_{2^{-j}t})]^2 - \Delta \exp_{2^{-j+1}t}| \\ &\leqslant \epsilon |\Delta \exp_{2^{-j}t}|^2 + 2\epsilon \tau_j |\Delta \exp_{2^{-j}t}|^2 \\ &= \epsilon [2(\tau_j + 1) - 1] |\Delta \exp_{2^{-j}t}|^2 . \end{aligned}$$

Repeating the squaring operation j times yields (B.6). □

The non-negative integer j may be chosen to minimize the error coefficient in (B.6),

$$c_j \equiv 2^j \left[e^{2^{-j} t\delta} \frac{1 + 2^{-j} t\delta}{2 - e^{2^{-j} t\delta}} + 1 \right] - 1 . \tag{B.8}$$

Coefficient (B.8) has a minimum for $2^{-j} t\delta \approx 0.3297$. Since $2^{-j} t\delta = 0.3297$ is probably not true for integer j, we ask that j satisfy $c_{j+1} \geqslant c_j$ and $c_{j-1} \geqslant c_j$. This condition is true for the smallest integer j such that

$$2^{-j} t\delta \geqslant 0.2209 . \tag{B.9}$$

References

[1] R. P. Brent, *Algorithms for Minimization without Derivatives,* Prentice-Hall (1973).

[2] C. Davis, Explicit functional calculus, *Linear Algebra and its Applications* **6** (1973), 193-199.

[3] D. T. Finkbeiner, *Introduction to Matrices and Linear Transformations,* Freeman, San Francisco (1960).

[4] F. R. Gantmacher, *Theory of Matrices,* v.1, Chelsea, New York (1959).

[5] A. O. Gel'fond, *Calculus of Finite Differences,* Hindustan, India (1971).

[6] G. H. Golub and J. H. Wilkinson, Ill-conditioned eigensystems and the computation of the Jordan canonical form, *SIAM Rev.* **18** (1976), 578-619.

[7] C. Jordan, *Calculus of Finite Differences,* Chelsea, New York (1950).

[8] B. Kagstrom, Numerical computation of matrix functions, Revised report UMINF-58.77, Dept. of Information Processing, Umea Univ., Sweden, July 1977.

[9] B. Kagstrom, Bounds and perturbation bounds for the matrix exponential, *BIT* **17** (1977), 39-57.

[10] B. Kagstrom and A. Ruhe, An algorithm for numerical computation of the Jordan normal form of a complex matrix, Revised report UMINF-59.77, Dept. of Information Processing, Umea Univ., Sweden, Sept. 1976.

[11] W. Kahan and I. Farkas, Algorithm 167, calculation of confluent divided differences, *Commun. Assoc. Comput. Mach.* **6** (1963), 164-165.

[12] L. M. Milne-Thomson, *The Calculus of Finite Differences,* Macmillan, London (1933).

[13] C. B. Moler and C. F. Van Loan, Nineteen ways to compute the exponential of a matrix, Technical Report 76-283, Dept. of Computer Science, Cornell Univ., July 1976.

[14] G. Opitz, Steigungsmatrizen, *ZAMM* **44** (1964), T52-54.

[15] B. N. Parlett, A recurrence among the elements of functions of triangular matrices, *Linear Algebra and its Applications* **14** (1976), 117-121.

[16] B. N. Parlett, A program to swap diagonal blocks, Memo. No. UCB/ERL M77/66, Univ. of Calif. at Berkeley, Nov. 1977.

[17] B. T. Smith et al., Matrix eigensystem routines - EISPACK guide, *Lecture Notes in Computer Science,* v.6, Springer-Verlag (1974).

[18] H. W. Turnbull and A. C. Aitken, *An Introduction to the Theory of Canonical Matrices,* Blackie and Son, London (1932).

[19] R. C. Ward, Numerical computation of the matrix exponential with accuracy estimate, *SIAM J. Numer. Anal.* **14** (1977), 600-609.

[20] J. H. Wilkinson, *Rounding Errors in Algebraic Processes,* Prentice-Hall (1963).

[21] J. H. Wilkinson, *The Algebraic Eigenvalue Problem,* Oxford (1965).