On Multipoint Numerical Interpolation

NAI-KUAN TSAO and ROSE MARIE PRIOR
Wayne State University

An efficient algorithm for storing and constructing a Neville-type divided difference table for general Hermite interpolation is described. This table contains all the information needed for multipoint interpolation where for better accuracy a new interpolation polynomial is desired such that for each given point $\bar{x}$ the interpolating points $x_i (i = 0, 1, \ldots, n)$ are reordered to cluster around $\bar{x}$.

Key Words and Phrases: polynomial, multipoint interpolation, Hermite interpolation, divided difference
CR Categories: 5.12, 5.13, 5.25

1. INTRODUCTION

Given a set of distinct points (x_i, f_i), where $f_i \equiv f(x_i)$ and $i = 0, 1, \ldots, n$, an nth degree polynomial $p(x)$ can be constructed so that $p(x_i) = f_i$, $i = 0, 1, \ldots, n$. The polynomial $p(x)$ is usually expressed in ordinary form as

$$p(x) = a_0 + a_1 x + \cdots + a_n x^n \tag{1}$$

or in Newton's form defined recursively as

$$\left.\begin{aligned} \pi_0(x) &= 1, \quad p_0(x) = c_0\pi_0(x) \\ \pi_k(x) &= (x - x_{k-1})\pi_{k-1}(x) \\ p_k(x) &= p_{k-1}(x) + c_k\pi_k(x) \end{aligned}\right\} \quad k = 1, 2, \ldots, n$$
$$p(x) = p_n(x) \tag{2}$$

where $c_k = f_{012\ \ldots\ k} \equiv f[x_0, x_1, \ldots, x_k]$, $k = 0, 1, \ldots, n$, is the divided difference of the kth order usually obtained recursively through Aitken's scheme stated as: for $k = 1, 2, \ldots, n$,

$$f_{01\ldots i,k} = (f_{01\ \ldots\ i} - f_{01,\ \ldots,i-1,k})/(x_i - x_k), \quad i = 0, 1, \ldots, k-1, \tag{3}$$

This work was supported in part by the U.S. Air Force under Grant AFOSR 76-3020.
Authors' address: Computer Science Section, Department of Mathematics, Wayne State University, Detroit, MI 48202

or through Neville's scheme stated as: for $i = 1, 2, \ldots, n$,

$$f_{i-j,i-j+1,\ldots,i} = (f_{i-j,i-j+1,\ldots,i-1} - f_{i-j+1,i-j+2,\ldots,i})/(x_{i-j} - x_i), \quad j = 1, 2, \ldots, i. \quad (4)$$

The intermediate results in eq. (3) or (4) are usually not stored to save storage spaces. Note also the divided differences generated in eq. (4) are those of consecutive points.

In general Newton's form is recommended since algorithms for evaluating the c_k (see [1, 4, 5]) require only about half (more for [1]) of the multiplicative operations needed by algorithms for evaluating a_k (see [2, 3]). Furthermore in evaluating $p(x)$ at a given point $x = \bar{x}$ the recursive nature of Newton's form makes it possible to terminate the computation of $p_k(\bar{x})$ at an earlier stage if the process "converges" rapidly. For this to happen it is usually advisable [5] to sort the x_k so that

$$|\bar{x} - x_{k+1}| \geq |\bar{x} - x_k|, \quad k = 0, 1, \ldots, n-1. \quad (5)$$

In other words, the set of points

$$\{x_0, x_1, \ldots, x_k\}, \quad k = 0, 1, \ldots, n,$$

is the closest one to $\bar{x}$ in the chosen order for each k. However, for multipoint interpolation, eq. (5) cannot be satisfied for each $\bar{x}$ unless eq. (3) or (4) is recomputed, which is expensive for each ordering of x_k corresponding to a given $\bar{x}$. We describe in this paper an efficient algorithm for multipoint interpolation satisfying eq. (5) for all possible $\bar{x}$ without having to recompute the coefficients c_k for each new $\bar{x}$. This algorithm is a generalized Neville's form of divided difference table which can also be used for Hermite interpolation if functional derivative values are also available.

2. MULTIPOINT INTERPOLATION

Let us assume that f_i and x_i are given such that $x_0 < x_1 < \cdots < x_n$ and a divided difference table in Neville's form is generated using eq. (4) with all intermediate results stored. Thus for $n = 4$ we have the following table:

$$\begin{array}{lllll} f_0 & & & & \\ f_1 & f_{01} & & & \\ f_2 & f_{12} & f_{012} & & \\ f_3 & f_{23} & f_{123} & f_{0123} & \\ f_4 & f_{34} & f_{234} & f_{1234} & f_{01234} \end{array}$$

Note the subscripts are all consecutive integers. Now if we denote $\{0, 1, \ldots, k\}$ as the set of all integers from 0 to k, then for a given $\bar{x}$ we can always find $i_0, i_1, \ldots, i_n$ such that $S_n \equiv \{i_0, i_1, \ldots, i_n\} = \{0, 1, 2, \ldots, n\}$ and

$$|\bar{x} - x_{i_k}| \geq |\bar{x} - x_{i_{k-1}}|, \quad k = 1, 2, \ldots, n. \quad (6)$$

So the desired polynomial is

$$p(x) = f_{i_0} + f_{i_0 i_1}(x - x_{i_0}) + \cdots + f_{i_0 i_1 \ldots i_n}(x - x_{i_0})(x - x_{i_1}) \cdots (x - x_{i_{n-1}}). \quad (7)$$

We now assert that $S_k \equiv \{i_0, i_1, \ldots, i_k\}$ is simply a collection of $k+1$ consecutive integers. Specifically,

$$\begin{aligned} S_k &\equiv \{i_0, i_1, \ldots, i_k\} = \{j_k, j_k+1, \ldots, j_k+k\}, \quad k = 0, 1, \ldots, n, \\ j_k &= \min_{i_l \in S_k} i_l, \quad k = 0, 1, \ldots, n. \end{aligned} \tag{8}$$

To see this we assume that eq. (8) is true for $k = \mathbf{k}$ and so

$$\{x_{i_0}, x_{i_1}, \ldots, x_{i_\mathbf{k}}\} = \{x_{j_\mathbf{k}}, x_{j_\mathbf{k}+1}, \ldots, x_{j_\mathbf{k}+\mathbf{k}}\}$$

is the set of $\mathbf{k}+1$ points closest to $\bar{x}$. Now $\bar{x}$ can either be inside the interval $[x_{j_\mathbf{k}}, x_{j_\mathbf{k}+\mathbf{k}}]$ or outside it. In the latter case $\bar{x}$ satisfies either

$$x_{j_\mathbf{k}-1} < \bar{x} < x_{j_\mathbf{k}}, \quad |\bar{x} - x_{j_\mathbf{k}}| \leq |\bar{x} - x_{j_\mathbf{k}-1}|$$

or

$$x_{j_\mathbf{k}+\mathbf{k}} < \bar{x} < x_{j_\mathbf{k}+\mathbf{k}+1}, \quad |\bar{x} - x_{j_\mathbf{k}+\mathbf{k}}| \leq |\bar{x} - x_{j_\mathbf{k}+\mathbf{k}+1}|.$$

In all cases the next point $x_{i_{\mathbf{k}+1}}$ must be either $x_{j_\mathbf{k}-1}$ or $x_{j_\mathbf{k}+\mathbf{k}+1}$ and so $S_{\mathbf{k}+1}$ satisfies eq. (8). By induction, eq. (8) is true for all k.

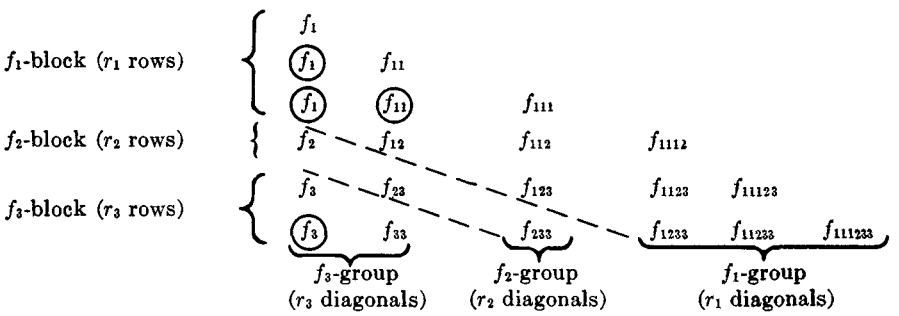
From eq. (8) it follows immediately that

$$\begin{aligned} f_{i_0 i_1 \ldots i_k} &= f_{j_k, j_k+1, \ldots, j_k+k}, \quad k = 0, 1, \ldots, n, \\ j_k &= \min_{i_l \in S_k} i_l, \quad k = 0, 1, \ldots, n. \end{aligned} \tag{9}$$

Thus the coefficients of $p(x)$ in eq. (7) for many $x_{i_0}, x_{i_1}, \ldots, a_{i_n}$ satisfying eq. (6) can be obtained from the computed Neville's table of divided differences. For multipoint interpolation the table can be generated only once and used over and over again. The only expense is the $(n+1)(n+2)/2$ storage spaces for the table which is more than the usual $(n+1)$ storage spaces for c_k.

3. THE ALGORITHMS

For programming convenience, let us consider the case where (x_i, f_i) are given for $i = 1, 2, \ldots, n$. If the derivative values of $f(x)$ are also available at $x_1, x_2, \ldots, x_n$, then a generalized Neville's table of divided differences can also be constructed for Hermite interpolation. For this purpose we denote $f_i^{(j)} \equiv f^j(x_i)$ and assume that $f_i, f_j^{(1)}, \ldots, f_i^{(r_i-1)}$ are given for $i = 1, 2, \ldots, n$. For $n = 3$ with $r_1 = 3$, $r_2 = 1$, $r_3 = 2$ the table is constructed as follows (see [4]):



(10)

where $f_{ii}(i = 1, 3)$ and $f_{iii}(i = 1)$ are generalized divided differences defined as $f_i^{(1)}/1!$ and $f_i^{(2)}/2!$, respectively. We see from this table there are three distinct

groups of rows and diagonals. Let us call them f_i-block (or group) as each row (or diagonal) within the block (or group) is headed by $f_i(i = 1, 2, 3)$ in the table. We observe that if the encircled redundant items are deleted, then there are equal numbers of items N_i in each row of the f_i-block. In fact,

$$N_1 = 1, \quad N_i = 1 + \sum_{k=1}^{i-1} r_k, \qquad i = 2, 3.$$

Furthermore, those items belonging to both f_i-block and f_j-group are divided differences formed by using $x_j - x_i$ as divisor. Based on these observations we have the following algorithm where a one-dimensional array T is used to store row-wise the essential items of a Neville table. (Here we assume that the array f contains the functional values in the order of $f_1, f_1^{(1)}, \ldots, f_1^{(r_1-1)}, f_2, \ldots, f_2^{(r_2-1)}, \ldots, f_n, \ldots, f_n^{(r_n-1)}$.)

Algorithm TABLE

1. For $i = 1, 2, \ldots, r_1$, $T_i = f_i/(i-1)!$
2. end $\leftarrow 0$, $N_1 = 1$, $m \leftarrow r_1$, $s \leftarrow r_1$
3. For $k = 2, 3, \ldots, n$ $(n \geq 2)$
   - 3.1. For $j = 1, 2, \ldots, r_k$
     - 3.1.1. $m \leftarrow m + 1$, $T_m = f_{s+j}/(j-1)!$
     - 3.1.2 For $i = k-1, k-2, \ldots, 1$
       - 3.1.2.1. $\Delta \leftarrow 1$
       - 3.1.2.2. If $j = 1$ and $i = k - 1$, then $\Delta \leftarrow N_{k-1}$
       - 3.1.2.3. For $l = 1, 2, \ldots, r_i$
         - 3.1.2.3.1. $m \leftarrow m + 1$, end $\leftarrow$ end $+ \Delta$, $d \leftarrow x_k - x_i$
         - 3.1.2.3.2. $T_m = (T_{m-1} - T_{\text{end}})/d$
     - 3.1.3. end $\leftarrow$ end $+ 1$
   - 3.2. $N_k = N_{k-1} + r_{k-1}$, end $\leftarrow$ end $- r_k * N_k$, $s \leftarrow s + r_k$

Once the table is constructed, then for a given $\bar{x}$ there exist unique $i_1, i_2, \ldots, i_n$ such that eq. (6) is satisfied. The desired polynomial $p(x)$ is also unique if only $f_i(i = 1, 2, \ldots, n)$ are given. This is not so if, in addition, derivative values are also given. For example, if $n = 3$, $i_1 = 3$, $i_2 = 2$, $i_3 = 1$, then $p(x)$ can be constructed by those coefficients one encounters when traveling from s to t along one of the five possible routes in the directed graph shown by Figure 1. However, since the generalized divided differences formed in step 3.1.1 are less likely to be affected by rounded-off errors, one should choose the set of coefficients f_3, f_{33}, f_{233}, f_{1233}, f_{11233}, f_{111233} for $p(x)$. Thus in the setup of (10) if k coefficients $d_1, d_2, \ldots, d_k$, where $d_i(i = 1, 2, \ldots, k)$ is the $(i-1)$th divided difference, have been chosen and there are two possible choices (at most two!) for d_{k+1}, then choose the one

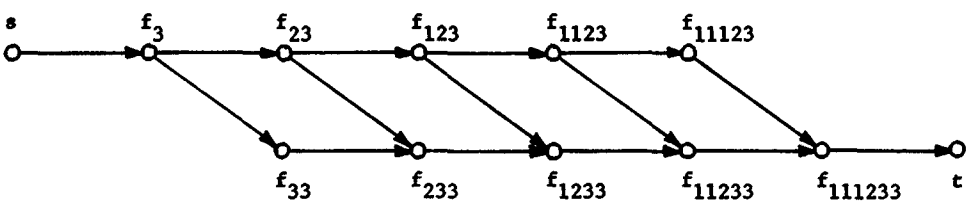


Fig. 1

along the same diagonal as d_k. Thus if $n = 3$, $i_1 = 2$, $i_2 = 3$, and $i_3 = 1$, then the coefficients for $p(x)$ are f_2, f_{23}, f_{233} (not f_{123}), f_{1233}, f_{11233}, and f_{111233}. In the following algorithm the desired ordering $i_1, i_2, \ldots, i_n$ are generated from a given $\bar{x}$ by a bisection-like strategy and then the subscript of the appropriate $(j-1)$th divided difference is stored in $d_j (j = 1, 2, \ldots, r_1 + r_2 + \cdots + r_n)$ using the output arrays N, r, and T from the algorithm TABLE.

Algorithm RETRIEVE

Part A. (Finding $i_1, i_2, \ldots, i_n$ for a given $\bar{x}$)

1. If $\bar{x} < x_1$, then set $i_k = k$ $(k = 1, 2, \ldots, n)$; go to step 8;
2. If $\bar{x} > x_n$; then set $i_k = n - k + 1$ $(k = 1, 2, \ldots, n)$; go to step 8;
3. $a \leftarrow 1$, $b \leftarrow n$;
4. If $a + b$ is even then set $m \leftarrow (a + b)/2$;
   else set $m \leftarrow (a + b + 1)/2$; go to step 6 if $m = b$;
5. If $x_a \leq \bar{x} \leq x_m$, then set $b \leftarrow m$, go to step 4;
   else set $a \leftarrow m$, go to step 4;
6. For $k = 1, 2, \ldots, n - 1$;
   if $|\bar{x} - a| < |\bar{x} - b|$, then set $i_k \leftarrow a$, $a \leftarrow a - 1$,
   else set $i_k \leftarrow b$, $b \leftarrow b + 1$;
7. If $a = 0$, then set $i_n = b$;
   else set $i_n = a$;

Part B. (Finding $d_j (j = 1, 2, \ldots, r_1 + r_2 + \cdots + r_n)$)

8. $d_1 \leftarrow 1$, $m \leftarrow 1$;
9. For $k = 2, 3, \ldots, i_1 (i_1 \geq 2)$
   9.1. $d_1 \leftarrow d_1 + r_{k-1} * N_{k-1}$;
10. For $k = 2, 3, \ldots, r_{i_1} (r_{i_1} \geq 2)$
    10.1. $m \leftarrow m + 1$, $d_m = d_{m-1} + N_{i_1}$;
11. For $k = 2, 3, \ldots, n$ $(n \geq 2)$
    11.1. $\Delta \leftarrow 1$
    11.2. If $i_k > i_{k-1}$, then set $\Delta \leftarrow N_{i_k}$
    11.3. For $j = 1, 2, \ldots, r_{i_k}$
    $m \leftarrow m + 1$, $d_m = d_{m-1} + \Delta$,

Finally, the polynomial $p(x)$ can be evaluated at $x = \bar{x}$ in full using all the coefficients obtained from the algorithm RETRIEVE by the following algorithm.

Algorithm VALUE

1. $p \leftarrow 0$, $\pi \leftarrow 1$, $m \leftarrow 0$
2. For $k = 1, 2, \ldots, n$
   2.1. For $j = 1, 2, \ldots, r_{i_k}$
   2.1.1. $m \leftarrow m + 1$, $p \leftarrow p + T_{d_m} * \pi$, $\pi \leftarrow \pi * (\bar{x} - x_{i_k})$;

4. CONCLUDING REMARKS

To see how our approach compares with others in the literature, let us consider the following problem: given the values of a function $f(x)$ and its first derivative at $n + 1$ distinct points $x_i (i = 0, 1, \ldots, n)$, a polynomial $p(x)$ of degree $2n + 1$ is desired to approximate $f(x)$ at a given point $\bar{x}$. For this case Table I gives the number of arithmetic operations required by the various approaches to find the coefficients of $p(x)$. For simplicity only the dominant terms for large n are given.

From Table I we see that our approach is comparable to the most efficient one

Table I. Number of Arithmetic Operations Required for Constructing $p(x)$

	Our Approach	Krogh's Approach	Bartels & Steingart's Approach	Björck & Elfving's Approach
Multiplicative Operations	$2n^2$	$2n^2$	$3n^2$	$4n^2$
Additive Operations	$2.5n^2$	$4n^2$	$2.5n^2$	$6n^2$

by Krogh where an Aitkin-type divided difference table is computed row by row. Note the polynomial $p(x)$ is in Newton's form of eq. (2) for both our approach and that of Krogh, which accounts for their more favorable arithmetic counts over the one by Björck and Elfving where $p(x)$ is in its ordinary form of eq. (1).

Now if $p(x)$ is needed for many points $\bar{x}_j$, say $j = 1, 2, \ldots, k$, and it is also desired that the set of $x_i (i = 0, 1, \ldots, n)$ be ordered for each $\bar{x}_j$ so that eq. (5) is satisfied for "better" accuracy, then our approach is far more efficient than the others as our table generated by using (x_i, f_i), $i = 0, 1, \ldots, n$, for $x_0 < x_1 < \cdots < x_n$ contains sufficient information to construct a new $p(x)$ for any ordering of x_i satisfying eq. (5). This is not true in the other approaches where the coefficients of a new $p(x)$ have to be recomputed for each ordering of x_i. This is certainly not economical for multipoint interpolation.

However, we should also emphasize that in our approach extra storage spaces are needed to store the final table. This number is essentially $\sum_{i=0}^{n} r_i N_i$ for the general case and is approximately $2(n + 1)^2$ for the above problem. Once the table is constructed, it also requires approximately $\log_2 n$ divisions and n multiplications to retrieve the proper subscripts in the array T for the coefficients of $p(x)$. To evaluate $p(x)$ is rather inexpensive as it requires only a multiple of n arithmetic operations in all approaches.

REFERENCES

1. Bartels, R., and Steingart, A. Hermite interpolation using a triangular polynomial basis. *ACM Trans Math. Software 2*, 3 (Sept. 1976), 252–256.
2. Björck, Å., and Elfving, T. Algorithms for confluent Vandermonde systems. *Numer. Math. 21* (1973), 130–137.
3. Gustafson, S.-Å. Rapid computation of general interpolation formulas and mechanical quadrature rules. *Comm. ACM 14*, 12 (Dec 1971), 797–801.
4. Jones, T.G. An algorithm for the numerical application of a linear operator. *J. ACM 9*, 4 (Oct 1962), 440–449.
5. Krogh, F.T. Efficient algorithms for polynomial interpolation and numerical differentiation. *Math. Comp. 24* (1970), 185–190.

Received January 1977; revised May 1977