Efficient Algorithms for Polynomial Interpolation and Numerical Differentiation*

By Fred T. Krogh

Abstract. Algorithms based on Newton's interpolation formula are given for: simple polynomial interpolation, polynomial interpolation with derivatives supplied at some of the data points, interpolation with piecewise polynomials having a continuous first derivative, and numerical differentiation. These algorithms have all the advantages of the corresponding algorithms based on Aitken-Neville interpolation, and are more efficient.

Introduction. The polynomial of nth degree passing through the points $(x_i, f(x_i))$, $i = 0, 1, \cdots, n$, is given by the Newton interpolation formula

$$p_n(x) = f[x_0] + \pi_1 f[x_0, x_1] + \pi_2 f[x_0, x_1, x_2] + \cdots + \pi_n f[x_0, x_1, \cdots, x_n] \tag{1}$$

where $\pi_i = (x - x_0)(x - x_1) \cdots (x - x_{i-1})$ and $f[x_0, x_1, \cdots, x_i]$ is the ith divided difference of f defined by**

$$f[x_k] = f(x_k)$$

$$f[x_0, x_k] = \frac{f[x_0] - f[x_k]}{x_0 - x_k} \tag{2}$$

$$f[x_0, x_1, \cdots, x_i, x_k] = \frac{f[x_0, x_1, \cdots, x_i] - f[x_0, x_1, \cdots, x_{i-1}, x_k]}{x_i - x_k}, \qquad i \geqq 1.$$

Although Newton's interpolation formula is well known, it is not widely used due to the popular misconception that it is inefficient. Usually recommended for interpolation are the methods of Aitken [1], Neville [2], or Lagrange's formula. Algorithms which make use of derivative values for interpolation are given in [2], [3], and [4]; and algorithms for numerical differentiation are given in [3], [5], and [6]. In all cases the algorithms given here are more efficient than other algorithms in the literature.

Simple Interpolation. The algorithms given below follow naturally from Eqs. (1) and (2). In these algorithms

Received February 12, 1968, revised June 25, 1969.

AMS Subject Classifications. Primary 6515, 6555.

Key Words and Phrases. Interpolation, numerical differentiation, Newton's interpolation formula, Aitken interpolation, Neville interpolation, Lagrange interpolation, Hermite interpolation, spline function.

* This work was done while the author was employed by TRW Systems.

** The recursion used in this definition gives the same results as obtained with the more usual $f[x_0, x_1, \cdots, x_k] = \{f[x_0, \cdots, x_{k-1}] - f[x_1, \cdots, x_k]\}/(x_0 - x_k)$. It has the minor advantages of letting one save the divided differences used in Eq. (1) with no extra storage requirement, and of giving indexes which do not decrease in the implementation of the algorithms (a convenience with some FORTRAN compilers).

$$
\begin{aligned}
V_{ik} &= f(x_k)\,, \quad i = 0\,, \\
&= f[x_0, x_1, \cdots, x_{i-1}, x_k]\,, \quad i = 1, 2, \cdots, k\,, \\
\omega_k &= x - x_k\,, \\
\pi_0 &= 1\,, \\
\pi_k &= (x - x_0)(x - x_1) \cdots (x - x_{k-1})\,.
\end{aligned} \tag{3}
$$

In the algorithms a statement of the form $x = f(x, y, \cdots)$ means compute f using the numbers in locations $x, y, \cdots$, and store the result in location x.

Algorithm I.

$$
\begin{aligned}
&\pi_0 = 1\,, \\
&V_{0,0} = f(x_0)\,, \\
&p_0 = V_{0,0}\,, \\
&\left.\begin{aligned}
V_{0,k} &= f(x_k)\,, \\
V_{i+1,k} &= \frac{V_{i,i} - V_{i,k}}{x_i - x_k}\,, \qquad i = 0, 1, \cdots, k-1 \\
\omega_{k-1} &= x - x_{k-1}\,, \\
\pi_k &= \omega_{k-1}\pi_{k-1}\,, \\
p_k(x) &= p_{k-1}(x) + \pi_k V_{k,k}\,,
\end{aligned}\right\} \quad k = 1, 2, \cdots, n\,.
\end{aligned}
$$

Note that in a computer program the ω's, π's, and p's can be scalars since in each case they are only used immediately after being computed, and similarly the array V can be replaced by a vector c with $c_\nu = V_{\mu,\nu}$.

An important feature of the Newton (or Aitken-Neville) algorithm is that one can select the value of n based on the convergence of the sequence $p_k(x)$, $k = 0, 1, \cdots$, without any additional computation (except for that involved in selecting n). If one is taking advantage of this fact, the x_k should be selected so that $|x - x_{k+1}| \geqq |x - x_k|$, cf. [7, p. 50].

Algorithm II.

$$
\begin{aligned}
&V_{0,0} = f(x_0)\,, \\
&\left.\begin{aligned}
V_{0,k} &= f(x_k)\,, \\
V_{i+1,k} &= \frac{V_{i,i} - V_{i,k}}{x_i - x_k}\,, \qquad i = 0, 1, \cdots, k-1
\end{aligned}\right\} \quad k = 1, 2, \cdots, n \\
&V_{k-1,k-1} = V_{k-1,k-1} + (x - x_{k-1})V_{k,k}\,, \qquad k = n, n-1, \cdots, 1 \\
&p_n(x) = V_{0,0}\,.
\end{aligned}
$$

Hermite Interpolation. Equations (1) and (2) hold (see e.g., Steffensen [8]) for $x_j = x_{j+1} = \cdots = x_{j+s}$ provided one avoids division by zero in Eq. (2) and defines

$$f[x_j, x_j, \cdots, x_j] = \frac{1}{s!}\frac{d^s f}{dx^s}\bigg|_{x=x_j} \tag{4}$$

where x_j is repeated $s+1$ times. To simplify the exposition we assume the data is given in the form

k	x_k	y_k
0	ξ_0	$a_{0,0}$
1	ξ_0	$a_{0,1}$
$\vdots$	$\vdots$	$\vdots$
q_0	ξ_0	a_{0,q_0}
q_0+1	ξ_1	$a_{1,0}$
$\vdots$	$\vdots$	$\vdots$
q_0+q_1+1	ξ_1	a_{1,q_1}
q_0+q_1+2	ξ_2	$a_{2,0}$
$\vdots$	$\vdots$	$\vdots$

where

$$a_{r,s} = \frac{1}{s!}\frac{d^s f}{dx^s}\bigg|_{\xi=\xi_r}.$$

In Algorithm III

$$\begin{aligned} V_{ik} = y_k &= f[\xi_r, \xi_r, \cdots, \xi_r], && i = 0 \\ &= f[x_0, x_1, \cdots, x_{i-1}, \xi_r, \xi_r, \cdots, \xi_r], && i > 0, \end{aligned} \tag{6}$$

where ξ_r is repeated $s+1$ times, and r and s are the subscripts of the a associated with y_k.

Algorithm III.

$$\pi_0 = 1$$
$$c_0 = f(x_0)$$
$$p_0 = c_0$$

$$\left.\begin{aligned} V_{0k} &= y_k \\ V_{i+1,k} &= \frac{c_i - V_{ik}}{x_i - x_k}, \qquad s = 0 \\ &= \frac{V_{i+1,k-1} - V_{ik}}{x_i - x_k}, \qquad s > 0 \end{aligned}\right\} \begin{array}{l} i = 0, 1, \cdots, k-1-s \\ (\text{If } s \geqq k, \text{ do nothing.}) \end{array} \Bigg\} k = 1, 2, \cdots, n.$$

$$c_k = V_{k-s,k}, \quad \omega_{k-1} = x - x_{k-1},$$
$$\pi_k = \omega_{k-1}\pi_{k-1}, \quad p_k(x) = p_{k-1}(x) + \pi_k c_k \Bigg\}$$

In the implementation of this algorithm $V_{\mu\nu}$ can be replaced by d_μ. Algorithm II can be extended to do Hermite interpolation in a similar way.

An Interpolating Function in C^1. If $n = 2m - 1$ $(m > 1)$, and the x_k are always selected so that m of them are on either side of x, then it is easy to construct an interpolating function which is composed of nth degree polynomials between tabular points and has a continuous first derivative where the polynomials are joined.

Let the table be given by $(\xi_i, f(\xi_i))$, $i = 1, 2, \cdots, N$, with $\xi_k < \xi_{k+1}$. Define $P^L_{n-1}(x)$ and $P^R_{n-1}(x)$ as the polynomials passing through points with indices $i = k - m + 1, k - m + 2, \cdots, k + m - 1$ and $i = k - m + 2, k - m + 3, \cdots, k + m$ respectively, where $\xi_k \leqq x < \xi_{k+1}$. If x is so close to one end of the table that there are not m points on either side of x, then $P^L_{n-1}(x) = P^R_{n-1}(x) =$ the polynomial passing through the n points nearest that end of the table. It is easy to verify that the function

$$C_n(x) = P^L_{n-1}(x) + \frac{x - \xi_k}{\xi_{k+1} - \xi_k}[P^R_{n-1}(x) - P^L_{n-1}(x)] \tag{7}$$

has the desired properties.

Let $x_{2j} = \xi_{k-j}$, $x_{2j+1} = \xi_{k+j+1}$, $j = 0, 1, \cdots, m - 1$. With this labeling of the x's, Eq. (1) yields

$$P^R_{n-1}(x) - P^L_{n-1}(x) = (x - x_0)(x - x_1)\cdots(x - x_{n-2})\{f[x_0, \cdots, x_{n-2}, x_n] - f[x_0, \cdots, x_{n-1}]\} \tag{8}$$

and thus

$$\begin{aligned} C_n(x) &= P^L_{n-1} + (x - x_0)\cdots(x - x_{n-2})\frac{(x - x_0)}{x_1 - x_0}\{f[x_0, \cdots, x_{n-2}, x_n] \\ &\qquad - f[x_0, \cdots, x_{n-1}]\} \qquad (9) \\ &= P^L_{n-1} + (x - x_0)\cdots(x - x_{n-2})(x - x_0)\frac{x_n - x_{n-1}}{x_1 - x_0} f[x_0, x_1, \cdots, x_n]\,. \end{aligned}$$

If the x_k are selected as indicated above, then the function $p_n(x)$ defined by the following algorithm has a continuous first derivative.

Algorithm IV.

$n = 2m - 1$ if there are m points on both sides of x

$= 2m - 2$ otherwise.

Do everything as in Algorithm I, except replace $\omega_{k-1} = x - x_{k-1}$ in Algorithm I with

$$\begin{cases} \omega_{k-1} = x - x_{k-1} & \text{if } k < 2m - 1\,, \\ \omega_{k-1} = x - x_0 & \\ V_{k,k} = V_{k,k}\left(\dfrac{x_k - x_{k-1}}{x_1 - x_0}\right) & \text{if } k = 2m - 1\,. \end{cases}$$

Numerical Differentiation. The algorithm for numerical differentiation is easily obtained by repeatedly differentiating Eq. (1).

Algorithm V. Start by performing Algorithm I (or IV) with $V_{\mu,\nu}$ replaced by c_ν. (Or if extra derivatives are available perform Algorithm III.) In any case save the π_k's and ω_k's, and then do the following

$$c_0 = p_n(x)$$

$$\left.\begin{aligned} \pi_i &= \omega_{k+i-1}\pi_{i-1} + \pi_i \,, \\ c_k &= c_k + \pi_i c_{k+i} \,, \end{aligned}\right\} \quad i = 1, 2, \cdots, n-k \,, \quad \left\} \begin{array}{l} k = 1, 2, \cdots, n-1 \\ \\ (\text{If } n \leqq 1 \text{ do nothing.}) \end{array}\right.$$

The c_k's give the coefficients of the interpolating polynomial expanded about x. (Of course, one need not compute all of them.) That is

$$p_n(\xi) = \sum_{k=0}^{n} c_k(\xi - x)^k \tag{10}$$

and

$$\left.\frac{d^k p_n(\xi)}{d\xi^k}\right|_{\xi=x} = k!c_k \,. \tag{11}$$

Lyness and Moler [6] give a brief discussion of the errors in a process such as this, and point out that iterative refinement can be used to reduce the rounding error introduced by the algorithm.

Comparison of Methods for Simple Interpolation. Table 1 gives the number of arithmetic operations required by the algorithms for simple polynomial interpolation. The computation in the Lagrangian case is assumed to be carried out according to the formula

$$p_n(x) = \left(\prod_{i=0}^{n} \omega_i\right) \sum_{k=0}^{n} \frac{f(x_k)}{A_{nk}\omega_k} \,, \tag{12}$$

where $\omega_i = x - x_i$ and $A_{nk} = \prod_{j=0}^{\prime n} (x_k - x_j)$. (The prime indicates that the term with $j = k$ is not included in the product.) Formula (12) is an efficient computational form of Lagrange's formula. It has the drawback that exceptional cases may cause overflow or underflow. Lagrange's formula is most efficient if polynomial interpolation of *fixed* degree is to be performed on several components of a vector valued function.

TABLE 1. *Number of Operations Required to Compute* $p_n(x)$

	Number of Divisions	*Number of Multiplications*	*Number of Additions and Subtractions*
Lagrange	$n + 1$	$(n + 1)(n + 2)$	$(n + 1)(n + 2)$
*Aitken or Neville****	$\frac{1}{2}n(n + 1)$	$n(n + 1)$	$(n + 1)^2$
Algorithm I	$\frac{1}{2}n(n + 1)$	$2n$	$(n + 1)^2 + n - 1$
Algorithm II	$\frac{1}{2}n(n + 1)$	n	$(n + 1)^2 + n - 1$

*** See footnote on p. 190.

The arithmetic operations required by the interpolation algorithm are an increasingly less important factor in determining the efficiency of a subroutine. The optimal algorithm in a given case will depend on the value of n and programming considerations. For example, Algorithm I requires more multiplications than Algorithm II, but it has simpler indexing and gives the difference between successive approximations.

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91103

1. A. C. AITKEN, "On interpolation by iteration of proportional parts, without the use of differences," *Proc. Edinburgh Math. Soc.*, v. 3, 1932, pp. 56–76.
2. E. H. NEVILLE, "Iterative interpolation," *J. Indian Math. Soc.*, v. 20, 1934, pp. 87–120.
3. M. GERSHINSKY & D. A. LEVINE, "Aitken-Hermite interpolation," *J. Assoc. Comput. Mach.*, v. 11, 1964, pp. 352–356. MR **29** #2938.
4. A. C. R. NEWBERY, "Interpolation by algebraic and trigonometric polynomials," *Math. Comp.*, v. 20, 1966, pp. 597–599. MR **34** #3752.
5. D. B. HUNTER, "An iterative method of numerical differentiation," *Comput. J.*, v. 3, 1960/61, pp. 270–271. MR **22** #8657.
6. J. N. LYNESS & C. B. MOLER, "Van Der Monde systems and numerical differentiation," *Numer. Math.*, v. 8, 1966, pp. 458–464.
7. F. B. HILDEBRAND, *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956. MR **17**, 788.
8. J. F. STEFFENSEN, *Interpolation*, Chelsea, New York, 1950. MR **12**, 164.
9. L. B. WINRICH, "Note on a comparison of evaluation schemes for the interpolating polynomial," *Comput. J.*, v. 12, 1969, pp. 154–155. (For comparison with the results given in Table 1 of this reference, our Algorithm II involves $n(n+1)$ subtractions and $\frac{1}{2}n(n+1)$ divisions for setup, and n additions, n subtractions, and n multiplications for each evaluation.)

*** This is for the usual procedure which is based on linear interpolation with Lagrange's formula. The referee points out that if this interpolation is given in the form of Newton's formula, the algorithm requires $\frac{1}{2}n(n+1)$ fewer multiplications and $\frac{1}{2}n(n+1)$ more additions. At the same time the round-off characteristics are improved. Thus for Aitken's algorithm.

$$P_{0,0} = f(x_0)$$
$$\omega_0 = x - x_0$$
$$P_{k,0}(x) = f(x_k)$$
$$\omega_k = x - x_k$$
$$P_{k,i+1}(x) = \left\{ \begin{array}{ll} \dfrac{\omega_k P_{i,i} - \omega_i P_{k,i}}{x_i - x_k} & \text{(Lagrange)} \\ & i = 0, 1, \cdots, k-1 \\ P_{i,i} + \dfrac{\omega_i}{x_i - x_k}\,[P_{i,i} - P_{k,i}] & \text{(Newton)} \end{array} \right\} \quad k = 1, 2, \cdots, n.$$
$$p_n(x) = P_{n,n}$$