A comparison of algorithms for polynomial interpolation

Allan J. Macleod

ABSTRACT

Eight different algorithms for polynomial interpolation are compared with respect to stability and computational efficiency. Large differences in performance are described and certain methods are shown to be very undesirable.

1. INTRODUCTION

The concept of polynomial interpolation is one of the first to be introduced in most numerical analysis textbooks. Mention is usually made of two or three different methods, yet hardly any work seems to have been done to compare the available algorithms. Winrich [8] and Krogh [3] discuss the problem of computational efficiency but only of a small subset of possible methods. Powell [6] contains a short discussion on the effect of rounding errors on the Lagrangian form and on the Newton form. In this paper eight algorithms are tested for stability and efficiency, and the results analysed to see if any one method is significantly better than the others.

2. DESCRIPTION OF THE METHODS

Let (x_i,f_i), $i=0,...,n$, be $(n+1)$ data points with $x_i < x_{i+1}$, $i = 0,...,n-1$, and $f_i = f(x_i)$ for some function f.

Let $x_0 < t < x_n$. Then f(t) is approximated by $P_n(t)$, the value at t of the unique polynomial of degree n through the data points.

The problem is : given (x_i,f_i), $i=0,...,n$, and t, evaluate $P_n(t)$.

Method 1. Lagrangian form, see Ralston and Rabinowitz [7]

$$P_n(t) = \sum_{i=0}^{n} L_i(t) f_i, \tag{1}$$

where

$$L_i(t) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(t - x_j)}{(x_i - x_j)} \tag{2}$$

To facilitate computation, (1) can be rewritten as

$$P_n(t) = \sum_{i=0}^{n} \left[A_i f_i \left[\prod_{\substack{j=0 \\ j\neq i}}^{n} (t - x_j) \right] \right], \tag{3}$$

with $A_i = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{1}{(x_i - x_j)}$

This form is useful if more than one interpolation is to be done with the same data set, as the A_i need only be calculated once.

Method 2. Barycentric form, see Hamming [2]

This expresses the interpolation polynomial in the form

$$P_n(t) = \frac{\sum_{i=0}^{n} \frac{A_i f_i}{(t - x_i)}}{\sum_{i=0}^{n} \frac{A_i}{(t - x_i)}} \tag{4}$$

with A_i as defined in (3).

Method 3. Aitken's, see Aitken [1]

There are many ways of describing this method. The following was given by Winrich.

Let $P_{k,0} = f_k$, $k = 0,...,n$.

$$\text{Define } P_{k,d+1} = \frac{(x_k - t) P_{d,d} - (x_d - t) P_{k,d}}{x_k - x_d} \tag{5}$$

for $k = d+1,...,n$, and $d = 0,...,n-1$.

Then $P_n(t) = P_{n,n}$.

Method 4. Modified Aitken, see Noble [5]

Noble pointed out that method 3 could suffer from rounding error, and recommended the following equivalent formulation as being more stable.

Let $P_{k,0} = f_k$, $k = 0,...,n$.

$$\text{Define } P_{k,d+1} = P_{d,d} + \frac{(x_d - t)(P_{d,d} - P_{k,d})}{x_k - x_d} \tag{6}$$

for $k = d+1,...,n$, and $d = 0,...,n-1$.

Then $P_n(t) = P_{n,n}$.

It is worth noting that, in both methods 3 and 4, $P_{d,d}$ is

A. J. Macleod, Department of Management Studies, Scottish College of Textiles, Galashiels, United Kingdom.

a fixed quantity during the inner k loop, and it can therefore be extracted and stored seperately to increase efficiency.

Method 5. Neville's, see Neville [4]

The particular formulation used here is :

let $P_{k,0} = f_k$, $k = 0,...,n$.

Define $$P_{k,d+1} = \frac{(x_k - t)P_{k-1,d} - (x_{k-d-1} - t)P_{k,d}}{x_k - x_{k-d-1}} \quad (7)$$

for $k = d+1,...,n$, and $d = 0,...,n-1$.

Then $P_n(t) = P_{n,n}$.

Method 6. Modified Neville

As in method 4, Neville's method can be rewritten, though not in such an obvious way.

Let $P_{k,0} = f_k$, $k = 0,...,n$.

Define $$P_{k,d+1} = P_{k-1,d} + \frac{(x_{k-d-1} - t)(P_{k-1,d} - P_{k,d})}{x_k - x_{k-d-1}} \quad (8)$$

for $k = d+1,...,n$, and $d = 0,...,n-1$.

Then $P_n(t) = P_{n,n}$.

This modified version has exchanged a multiplication for an addition, so it should be somewhat faster on most machines. The lack of a fixed element, such as $P_{d,d}$, means that Neville's method will be less efficient than the corresponding Aitken method.

The final two methods are based on algorithms given in the paper by Krogh. These are based on the Newton divided-difference form of the interpolation polynomial. The algorithms are not given exactly as in Krogh, but more in the form used to program them. Both algorithms have two parts – the first calculates the divided-differences, the second evaluates the polynomial.

Method 7. Krogh One

Let $V_0 = f_0$

$$\left.\begin{aligned} V &= f_k \\ V &= \frac{V_i - V}{x_i - x_k}, \quad i = 0,...,k-1 \\ V_k &= V \end{aligned}\right\} \quad k = 1,...,n$$

Let $Q_0 = 1$, $P_0 = f_0$.

$$\left.\begin{aligned} Q_k &= (t - x_{k-1})\,Q_{k-1} \\ P_k &= P_{k-1} + Q_k V_k \end{aligned}\right\} \quad k = 1,...,n$$

Then $P_n(t) = P_n$.

Method 8. Krogh Two

Define V_k, $k = 0,...,n$, as in method 7.

Let $V_{k-1} = V_{k-1} + (t - x_{k-1})\,V_k$, $k = n,n-1,...,1$.

Then $P_n(t) = V_0$

The difference between methods 7 and 8 can be thought of as being the difference in evaluating a polynomial in its power form, and evaluating using nested multiplication. Thus one would expect method 8 to be faster than method 7.

3. TEST RESULTS

The eight algorithms given above were tested in two ways. Firstly their stability with respect to rounding errors was compared, and then their computational efficiency assessed.

To perform the first test the following experiment was performed. For various functions f, and various values of n, data pairs (x_i, f_i), $i = 0,1,...,n$, were formed in single precision, with $0 = x_0 < x_1 < ... < x_n = 1$, and $f_i = f(x_i)$. Another random point t was generated in (0,1), and the interpolation value $P_n(t)$ calculated by all the methods. The data pairs and the point t were then transformed to double precision and the double precision result $P_n^*(t)$ calculated. This whole procedure was repeated 100 times for each value of n, resulting in the following error measure,

$$\sqrt{\frac{1}{100}\sum_{i=1}^{100}[P_n^*(t_i) - P_n(t_i)]^2},$$

which was calculated for each method and for each value of n.

Tables 1 and 2 give the results obtained for two specific functions. These results show wide differences between the eight algorithms. The methods based on the Aitken approach show the worst instability, with the divided-difference methods also performing badly. Of the others the modified Neville method clearly seems the most stable.

The results for computational efficiency are given in table 3. These are ratios of time taken by each method as against the time taken by method 1. Clearly the algorithms of Krogh are the fastest with the modified Aitken a close third. These comparisons can be supported by an analysis of each method in terms of operation count, including array look-ups.

4. CONCLUSION

The results given in this paper make the recommendation of one specific method very difficult. Since interpolation, even in large tables, is usually done using only a few points in the neighbourhood of the desired point, it would appear that for less than 10 points method 8 is best, but for between 10 and 20 points the modified Neville method is best. The use of polynomial interpolation with more than 20 points seems a very dubious enterprise, and some other method should be used.

REFERENCES

1. AITKEN A. C. : 'On interpolation by iteration of proportional parts, without the use of differences'. Proc. Edinburgh Math. Soc. 3 (1932) 56-76.

TABLE 1. Error results for $f(x) = \ln(1+x)$

n	Method 1	2	3	4	5	6	7	8
3	.86 E-6	.11 E-5	.50 E-6	.21 E-6	.33 E-6	.10 E-6	.73 E-7	.53 E-7
7	.26 E-4	.33 E-4	.79 E-4	.34 E-4	.35 E-5	.21 E-6	.90 E-6	.88 E-6
11	.93 E-2	.77 E-2	.78 E-1	.23 E+0	.25 E-3	.23 E-5	.16 E-2	.16 E-2
15	.32 E-1	.53 E-1	.18 E+1	.32 E-1	.76 E-3	.13 E-3	.44 E-3	.47 E-3
19	.74 E+2	.25 E+0	.69 E+5	.91 E+1	.96 E+1	.15 E-1	.81 E+1	.81 E+1
23	.51 E+2	.15 E+2	.69 E+5	.21 E+4	.23 E+1	.20 E-1	.36 E+2	.36 E+2
27	.25 E+2	.21 E+2	.28 E+7	.25 E+6	.18 E+1	.55 E-1	.56 E+4	.55 E+4

TABLE 2. Error results for $f(x) = |x - 0.4|$

n	Method 1	2	3	4	5	6	7	8
3	.34 E-6	.40 E-6	.48 E-6	.48 E-6	.31 E-6	.32 E-6	.49 E-6	.64 E-6
7	.75 E-5	.11 E-3	.78 E-3	.82 E-4	.11 E-5	.33 E-6	.82 E-4	.82 E-4
11	.74 E-4	.15 E-2	.18 E-1	.23 E-2	.20 E-4	.95 E-5	.54 E-3	.54 E-3
15	.14 E-2	.42 E+1	.10 E+1	.15 E+0	.42 E-2	.23 E-3	.22 E-1	.22 E-1
19	.29 E+1	.15 E+2	.91 E+4	.86 E+4	.76 E+0	.67 E-2	.13 E+2	.13 E+2
23	.12 E+2	.19 E+4	.76 E+5	.14 E+5	.35 E+2	.97 E-1	.22 E+3	.20 E+3
27	.10 E+3	.73 E+4	.11 E+10	.28 E+6	.38 E+1	.28 E+0	.39 E+5	.39 E+5

TABLE 3. Timing ratios for each method

Method	n = 8	n = 16
1	1.000	1.000
2	1.025	1.012
3	0.835	0.869
4	0.658	0.485
5	0.987	1.385
6	0.684	0.808
7	0.646	0.496
8	0.570	0.469

2. HAMMING R. W. : *Numerical methods for scientists and engineers*. McGraw-Hill New York (1962).
3. KROGH F. T. : 'Efficient algorithms for polynomial interpolation and numerical differentiation'. Maths. Comp. 24 (1970) 185-190.
4. NEVILLE E. H. : 'Iterative interpolation'. J. Indian Math. Soc. 20 (1934) 87-120.
5. NOBLE B. : *Numerical methods vol. 2*, Oliver & Boyd Edinburgh (1964).
6. POWELL M. J. D. : *Approximation theory and methods*. Cambridge University Press, Cambridge (1981).
7. RALSTON A. and RABINOWITZ P. : *A first course in numerical analysis*. McGraw-Hill, New York (1978).
8. WINRICH L. B. : 'Note on a comparison of evaluation schemes for the interpolating polynomial'. Computer Journal 12 (1969) 154-155.