Lagrange polynomial interpolation with derivatives and error indication

José Luís Martins

Departamento de Física, Instituto Superior Técnico,

Universidade de Lisboa, Av. Rovisco Pais, 1049-001 Lisboa, Portugal and

Instituto de Engenharia de Sistemas e Computadores –

Microsistemas e Nanotecnologias (INESC MN) Lisbon,

Portugal, Rua Alves Redol 9, 1000-029 Lisboa, Portugal

(Dated: November 16, 2021)

Abstract

A code for Lagrange polynomial interpolation in either Fortran or C is described. It includes the calculation of the derivatives of the interpolating function and an indication of the error through the last correction of those values. It is based on a modification of Neville's algorithm to keep track of differences between recursion levels, and Hunter's method for differentiation.

I. INTRODUCTION

Lagrange polynomial interpolation is sufficiently simple that for low fixed order of the polynomial it can be coded essentially in a single line. Code for the general case with arbitrary order can be found in many libraries, and is usually short. The Fortran code form Numerical Recipes[1], that includes an indication about the error is less than 40 lines.

Codes that give the coefficients of the interpolating polynomial are available and from them one can obtain the coefficients of the respective derivatives, but this is an indirect method with serious numerical accuracy problems. Numerical Recipes[1] after showing such a code asks the reader: "Are you still quite certain that the *coefficients* are what you want?"

The NAG algorithm `nagf_interp_dim1_cheb`[2] gives the interpolating polynomial as a Chebyshev series, which do not have the accuracy problems of the raw polynomial coefficients, `nagf_fit_dim1_cheb_deriv`[3] gives the Chebyshev series of the derivative (and can be used repeatidly), ant the series can be evaluated with `nagf_interp_dim1_cheb_eval` at the desired point.[4]

There are also many codes to differentiate a function by a wise choice of interpolation points. The problem addressed here is for the case where the data points are given and the underlying function is not explicitly available.

I was unable to find open source codes that perform a polynomial interpolation and give directly the values of the derivatives. At the risk of reinventing the wheel I figured out it was faster and funnier to code such an algorithm. The Fortran and C code are available for download.[5]

I should note that if the data set has some noise, polynomial interpolation should be avoided, as a more robust spline interpolation is probably better, and it gives for free the first and second derivatives of the interpolating function. Also the user should avoid the idea that higher order means higher accuracy. But for good quality data polynomial interpolation of moderate order can be quite useful. The present code was developed to calculate derivatives of data generated by solving the radial Schrödinger equation in atoms.

II. METHOD

We will follow closely the discussion of D. B. Hunter[6] on numerical differentiation. Suppose $y(x)$ is a function that takes the values $y_0, y_1, y_2, \ldots, y_n$ when $x = x_0, x_1, x_2, \ldots, x_n$. Then if $i, j, \ldots, p, q$ is any set of integers in the range $(0, n)$, let us denote

$$y_{i,j,\ldots,p,q}(x)$$

the polynomial of least degree in x that takes the values $y_i, y_j, \ldots, y_p, y_q$ when $x = x_i, x_j, \ldots, x_p, x_q$. The Lagrange interpolation formula can be concisely written for the particular case

$$y_{0,1,2,\ldots,m}(x) = \sum_{k=0}^{m} y_k \, \frac{\prod_{j\neq k}(x_j - x)}{\prod_{j\neq k}(x_j - x_k)}. \tag{1}$$

More useful is the recurrence expression

$$y_{i,j,\ldots,p,q}(x) = \frac{1}{x_q - x_i}\Big((x_q - x)\, y_{i,j,\ldots,p}(x) - (x_i - x)\, y_{j,\ldots,p,q}(x)\Big). \tag{2}$$

which can be trivially generalized to any pair of indices, as the order of the x_j is not specified. Note that the polynomials $y_{i,j,\ldots,p,q}$ are invariant by a permutation of the $i, j, \ldots, p, q$ as a trivial consequence of its definition.

If Eq. 2 is differentiated it gives the result

$$y'_{i,j,\ldots,p,q}(x) = \frac{1}{x_q - x_i}\Big((x_q - x)\, y'_{i,j,\ldots,p}(x) - (x_i - x)\, y'_{j,\ldots,p,q}(x) + y_{j,\ldots,p,q}(x) - y_{i,j,\ldots,p}(x)\Big).$$

More generally if Eq. 2 it is differentiated r times we obtain the recursion expression[6]

$$y^{(r)}_{i,j,\ldots,p,q}(x) = \frac{1}{x_q - x_i}\Big((x_q - x)\, y^{(r)}_{i,j,\ldots,p}(x) - (x_i - x)\, y^{(r)}_{j,\ldots,p,q}(x) + r\,\big(y^{(r-1)}_{j,\ldots,p,q}(x) - y^{(r-1)}_{i,j,\ldots,p}(x)\big)\Big). \tag{3}$$

Numerical inaccuracy occurs for the subtraction of similar large numbers. As $y^{(r-1)}_{j,\ldots,p,q}(x)$ and $y^{(r-1)}_{i,j,\ldots,p}(x)$ are approximations of the same quantity, the last term of Eq. 3 is a case of concern. It is safer to work with the small differences that are defined in Numerical Recipes[1] by

$$\begin{aligned} c_{i,j,\ldots,p,q}(x) &= y_{i,j,\ldots,p,q}(x) - y_{i,j,\ldots,p}(x) \\ d_{i,j,\ldots,p,q}(x) &= y_{i,j,\ldots,p,q}(x) - y_{j,\ldots,p,q}(x), \end{aligned} \tag{4}$$

and which satisfy the recurrence relations

$$c_{i,j,\ldots,p,q}(x) = (x_i - x)\ \frac{c_{j,\ldots,p,q}(x) - d_{i,j,\ldots,p}(x)}{x_i - x_q}$$
$$d_{i,j,\ldots,p,q}(x) = (x_q - x)\ \frac{c_{j,\ldots,p,q}(x) - d_{i,j,\ldots,p}(x)}{x_i - x_q}. \tag{5}$$

At each step one can decide if one uses c or d to update the y.

The equations 4 and 5 can be generalized to the case of derivatives by defining

$$c^{(r)}_{i,j,\ldots,p,q}(x) = y^{(r)}_{i,j,\ldots,p,q}(x) - y^{(r)}_{i,j,\ldots,p}(x)$$
$$d^{(r)}_{i,j,\ldots,p,q}(x) = y^{(r)}_{i,j,\ldots,p,q}(x) - y^{(r)}_{j,\ldots,p,q}(x), \tag{6}$$

which now satisfy the recurrence relations

$$c^{(r)}_{i,j,\ldots,p,q}(x) = (x_i - x)\ \frac{c^{(r)}_{j,\ldots,p,q}(x) - d^{(r)}_{i,j,\ldots,p}(x)}{x_i - x_q} - r\ \frac{c^{(r-1)}_{j,\ldots,p,q}(x) - d^{(r-1)}_{i,j,\ldots,p}(x)}{x_i - x_q}$$
$$d^{(r)}_{i,j,\ldots,p,q}(x) = (x_q - x)\ \frac{c^{(r)}_{j,\ldots,p,q}(x) - d^{(r)}_{i,j,\ldots,p}(x)}{x_i - x_q} - r\ \frac{c^{(r-1)}_{j,\ldots,p,q}(x) - d^{(r-1)}_{i,j,\ldots,p}(x)}{x_i - x_q}. \tag{7}$$

III. IMPLEMENTATION

The algorithm was implemented in Fortran in a subroutine that takes as input the order n of the interpolation, the maximum order $r_{\max} \leq n$ of the requested derivatives, and the $n+1$ grid points $x_0, x_1, x_2, \ldots, x_n$ and the corresponding $n+1$ values $y_0, y_1, y_2, \ldots, y_n$ of the function to be interpolated and differentiated. The output is the estimate of the function and its derivatives $Y_r = y^{(r)}_{0,1,2,\ldots,n}(0)$ for $r = 0, 1, \ldots, r_{\max}$ at the origin. The value δY_r of the last update to Y_r is an indication of the possible order of magnitude of the error. For any other point x one just enters the values of $x_0 - x, x_1 - x, x_2 - x, \ldots, x_n - x$, that is apply a rigid shift of the function.

It is clear from Eq. 2 and our remark that the order of the indices of the polynomial $y_{i,j,\ldots,p,q}(x)$ is irrelevant, that there are many paths from the order 0 polynomials $y_i(x) = y_i$ to the target polynomial $y_{0,1,2,\ldots,n}(x)$.[7] The final result only depends on the path through round-off errors. Without loss of generality we can assume that the input points are ordered, and in

that case it makes some sense to work only with polynomials of the form $y^{(r)}_{i,i+1,i+2,\ldots,i+m}(x)$, with $0 \le m \le n-i$. Defining

$$\begin{aligned} C_m(r,i) &= c^{(r)}_{i,i+1,\ldots,i+m}(0) \\ D_m(r,i) &= d^{(r)}_{i,i+1,\ldots,i+m}(0), \end{aligned} \tag{8}$$

the recursion equation 7 becomes

$$\begin{aligned} C_m(r,i) &= x_i \, \frac{C_{m-1}(r,i+1) - D_{m-1}(r,i)}{x_i - x_{i+m}} - r \, \frac{C_{m-1}(r-1,i+1) - D_{m-1}(r-1,i)}{x_i - x_{i+m}} \\ D_m(r,i) &= x_{i+m} \, \frac{C_{m-1}(r,i+1) - D_{m-1}(r,i)}{x_i - x_{i+m}} - r \, \frac{C_{m-1}(r-1,i+1) - D_{m-1}(r-1,i)}{x_i - x_{i+m}}, \end{aligned} \tag{9}$$

which is the recursion implemented in the code.

The implemented path from the 0-th order to n-th order polynomial is the same as in Numerical Recipes[1], which tries to keep the partial approximations centered on the target (in this case $x = 0$). Its first step is to identify the point nearest to the origin, n_s that satisfies $|x_{n_s}(0)| \le |x_i|$ for all i.

Algorithm 1 presents the implementation, with the omission of initialization of variables to zero. For the details it is better to read the working code.

As a common use of interpolation is to go from data on one grid to another grid, it is also included in the available code such a procedure, which also checks that the arguments make sense. Test examples are also made available.

The interpolating Fortran subroutine was converted to C by the f2c software[8] after it was recast in Fortran77. The result was slightly edited to only need standard C libraries, and be easier to read and is also available.[5]

r	$y^{(r)}(12.3)$	$f^{(r)}(12.3)$	Error	$\lvert\delta y^{(r)}\rvert$
0	3.507135526	3.507135583	-0.000000058	0.000000350
1	0.142566367	0.142566487	-0.000000120	0.000000881
2	-0.005794807	-0.005795386	0.000000579	0.000002771
3	0.000707463	0.000706754	0.000000708	0.000008058
4	-0.000147733	-0.000143649	-0.000004084	0.000011792
5	0.000039307	0.000040876	-0.000001569	

TABLE I: Results of the interpolation of $y(x)$ of the function $f(x) = \sqrt(x)$ for $x = 12.3$ starting from the points $x_i = 10 + i$, for $i = 0, 1, \ldots, 5$, including the estimates of its derivatives of order r. The last column shows the last update to $y^{(r)}$ in the algorithm and gives an indication of the error.

IV. EXAMPLES

The example of D. B. Hunter's article[6] is to calculate the value of $\sqrt{12.3}$ with a fifth order polynomial interpolation using the points $x_i = 10 + i$, for $i = 0, 1, \ldots, 5$. This is a well behaved function, and even though the starting points are reasonably spread, the accuracy is quite good as can be seen in Table I. Obviously the accuracy is reduced for the highest derivatives as should be expected, but even for the fifth derivative the estimate is still quite good. The important point to stress here is that although the value of $|\delta y^{(r)}|$ is an overestimate of the true error it still conveys quite well its magnitude. By construction the last update of the n-th derivative of the n-th order interpolation is equal to the interpolated value, and therefore is omitted from the table as it cannot be considered an indication of the accuracy.

To illustrate how an interpolation can go wrong the second example is almost the same as the first except that two of the grid points are almost the same. As can be best seen from Eq. 1 division by zero occurs if two points are exactly the same. Table II shows the results for $x_1 = 10, 11, 12, 12 + 1/10^{12}, 14, 15$. The accuracy is much lower than in the previous example and and the value of $|\delta y^{(r)}|$ is an underestimate of the true error. Notice that $|\delta y^{(n-1)}| \gg |y^{(n-1)}|$ which should be considered an indication that there is a problem with the interpolation, and shows the usefulness of $|\delta y^{(r)}|$ to assess the reliability of the

r	$y^{(r)}(12.3)$	$f^{(r)}(12.3)$	Error	$\lvert\delta y^{(r)}\rvert$
0	3.507280620	3.507135583	0.000145037	0.000016271
1	0.143085623	0.142566487	0.000519136	0.000118496
2	-0.005732078	-0.005795386	0.000063307	0.000483019
3	-0.001089820	0.000706754	-0.001796574	0.000405480
4	-0.000570623	-0.000143649	-0.000426974	0.002134104
5	0.004268208	0.000040876	0.004227332	

TABLE II: Results of the interpolation of $y(x)$ of the function $f(x) = \sqrt(x)$ for $x = 12.3$ starting from the points $x_i = 10, 11, 12, 12+1/10^{12}, 14, 15$, including the estimates of its derivatives of order r. The last column shows the last update to $y^{(r)}$ in the algorithm and gives an indication of the error. The poor choice of two almost coincident points makes the interpolation worse than in the previous example.

calculation.

Another problem occurs when the interpolated function has features on a scale smaller than the spacing between points used for the interpolation. Table III shows the results for interpolating the Gaussian function $f(x) = \exp(-2x^2)$ sampled at the points $x_i = -3 + i$, $i = 0, 1, \ldots, 6$ for $x = 0.6$. With such a poor sampling of the function the interpolated results are quite bad, in particular the sign is even wrong for the second derivative. However the important message of this example is that the large values of $|\delta y^{(r)}|$, in particular for the second and third derivatives, are a clear indication that there are problems with the quality of the interpolation.

V. CONCLUSIONS

The examples show that the code works fine for the cases where the available data points describe well the function to be interpolated. The most important point is that the information on the last update of the function and its derivatives can be used to identify the problematic cases where the interpolation is not accurate.

r	$y^{(r)}(12.3)$	$f^{(r)}(12.3)$	Error	$\lvert\delta y^{(r)}\rvert$
0	0.6229	0.4868	0.1362	0.1114
1	-1.1274	-1.1682	0.0408	0.0290
2	-1.0389	0.8567	-1.8955	1.1239
3	3.9712	7.2896	-3.3184	1.5785

TABLE III: Results of the interpolation of $y(x)$ of the function $f(x) = \exp(-2x^2)$ for $x = 0.5$ starting from the points $x_i = -3 + i,$, $i = 0, 1, \ldots, 6$. The last column shows the last update to $y^{(r)}$ in the algorithm and gives an indication of the error. The poor quality of the interpolation resulting from the sparseness of the grid points is indicated by the large values of $|\delta y^{(r)}|$.

VI. ACKNOWLEDGMENTS

The author wishes to acknowledge Portuguese Science and Technology Foundation (FCT) for funding of the Research Unit INESC MN (UID/05367/2020) through Plurianual, Base and Programatico financing.

Algorithm 1 Lagrange interpolation with derivatives

$n_s \leftarrow 0$

for $i = 1, \ldots, n$ **do**

  if $|x_i| < |x_{n_s}|$ **then**

    $n_s \leftarrow i$ ▷ Nearest point to origin

  end if

end for

$Y(0) = y_{n_s}$ ▷ Start of initialization and order 0

$n_s \leftarrow n_s - 1$

for $i = 0, \ldots, n$ **do**

  $C(0, i) \leftarrow y_i$

  $D(0, i) \leftarrow y_i$

end for

for $m = 1, \ldots, n$ **do** ▷ Loop on order of interpolation

  for $i = 0, \ldots, n - m$ **do** ▷ Loop on first sub index

    for $r = 0, \ldots, \min(r_{\max}, m)$ **do** ▷ Loop on order of derivative

      $T(r) \leftarrow (D(r, i) - C(r, i + 1))/(x_{i+m} - x_i)$

      $C(r, i) \leftarrow x_{i+m} T(r) - rT(r - 1)$

      $C(r, i) \leftarrow x_i T(r) - rT(r - 1)$

    end for

  end for

  if $2n_s + 1 < n - m$ **then** ▷ Chooses C or D

    $\delta y(r) \leftarrow C(r, n_s + 1)$; for $r = 0, \ldots, \min(r_{\max}, m)$

  else

    $\delta y(r) \leftarrow D(r, n_s)$; for $r = 0, \ldots, \min(r_{\max}, m)$

    $n_s \leftarrow n_s - 1$

  end if

  $y(r) \leftarrow y(r) + \delta y(r)$; for $r = 0, \ldots, \min(r_{\max}, m)$

end for

References

---

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran 90 (2nd Ed.): The Art of Parallel Scientific Computing* (Cambridge University Press, USA, 1996).

[2] The Numerical Algorithms Group Ltd., *NAG FL Interface e01aef (dim1 cheb)*, `https://www.nag.com/numeric/nl/nagdoc_latest/flhtml/e01/e01aef.html`.

[3] The Numerical Algorithms Group Ltd., *NAG FL Interface e02ahf (dim1 cheb deriv)*, `https://www.nag.com/numeric/nl/nagdoc_latest/flhtml/e02/e02ahf.html`.

[4] The Numerical Algorithms Group Ltd., *NAG FL Interface e02akf (dim1 cheb eval)*, `https://www.nag.com/numeric/nl/nagdoc_latest/flhtml/e02/e02aef.html`.

[5] Martins, J. L., *polynomial_interpolation*, `https://github.com/jlm785/polynomial_interpolation/`.

[6] D. B. Hunter, The Computer Journal **3**, 270 (1961).

[7] The notation for the order 0 polynomials may be a bit confusing, but is consistent with the article by D. B. Hunter.

[8] Feldman, S. I. and Gay, D. M. and Maimone, M. W. and Schryer, N. L., *A fortran to c converter*, `https://www.netlib.org/f2c/`.