NAG Fortran Library Chapter Introduction

D04 – Numerical Differentiation

Contents

1 Scope of the Chapter

This chapter is concerned with calculating approximations to derivatives of a function f, where the user can supply a routine representing f.

2 Background to the Problems

2.1 Description of the Problem

The problem of numerical differentiation does not receive very much attention nowadays. Although the Taylor series plays a key role in much of classical analysis, the poor reputation enjoyed by numerical differentiation has led numerical analysts to construct techniques for most problems which avoid the explicit use of numerical differentiation.

One may briefly and roughly define the term numerical differentiation as any process in which numerical values of derivatives $f^{(s)}(x_0)$ are obtained from evaluations $f(x_i)$ of the function $f(x)$ at several abscissae x_i near x_0. This problem can be stable, well-conditioned, and accurate when complex-valued abscissae are used. This was first pointed out by Lyness and Moler (1967). An item of numerical software for this appears in Lyness and Ande (1971). However, in many applications the use of complex-valued abscissae is either prohibitive or prohibited. The main difficulty in using real abscissae is that amplification of round-off error can completely obliterate meaningful results. In the days when one relied on hand calculating machines and tabular data, the frustration caused by this effect led to the abandonment of serious use of the process.

There are several reasons for believing that, in the present-day computing environment, numerical differentiation might find a useful role. The first is that, by present standards, it is rather a small-scale calculation, so its cost may well be negligible compared with any overall saving in cost that might result from its use. Secondly, the assignment of a step length h is now generally open. One does not have to rely on tabular data. Thirdly, although the amplification of round-off error is an integral part of the calculation, its effect can be measured reliably and automatically by the routine at the time of the calculation.

Thus the user does not have to gauge the round-off level (or noise level) of the function values or assess the effect of this on the accuracy of the results. A routine does this automatically, returning with each result an error estimate which has already taken round-off error amplification into account.

We now illustrate, by means of a very simple example, the importance of the round-off error effect. A very simple approximation of $f'(0)$, based on the identity

$$f'(0) = (f(h) - f(-h))/2h + (h^2/3!)f'''(\xi), \tag{1}$$

is

$$(f(h) - f(-h))/2h.$$

If there were no precision problem, this formula would be the only one needed in the theory of first-order numerical differentiation. We could simply take $h = 10^{-40}$ (or $h = 10^{-1000}$) to obtain an excellent approximation based on two function values. It is only when we consider in detail how a finite length machine comes to calculate $(f(h) - f(-h))/2h$ that the necessity for a sophisticated theory becomes apparent.

To simplify the subsequent description we shall assume that the quantities involved are neither so close to zero that the machine underflow characteristics need be considered nor so large that machine overflow occurs. The approximation mentioned above involves the function values $f(h)$ and $f(-h)$. In general no computer has available these numbers exactly. Instead it uses approximations $\hat{f}(h)$ and $\hat{f}(-h)$ whose relative accuracy is less than some tolerance ϵ_{f}. If the function $f(x)$ is a library function, for example $\sin x$, ϵ_f may coincide with the machine accuracy parameter ϵ_{m}. More generally the function $f(x)$ is calculated in a user-supplied routine and ϵ_f is larger than ϵ_m by a small factor 5 or 6 if the calculation is short or by some larger factor in an extended calculation. This factor is not usually known by the user.

$\hat{f}(h)$ and $\hat{f}(-h)$ are related to $f(h)$ and $f(-h)$ by

$$\hat{f}(h) = f(h)(1+\theta_1\epsilon_f), \qquad |\theta_1| \leq 1$$

$$\hat{f}(-h) = f(-h)(1+\theta_2\epsilon_f), \quad |\theta_2| \leq 1.$$

Thus even if the rest of the calculation were carried out exactly, it is trivial to show that

$$\frac{\hat{f}(h)-\hat{f}(-h)}{2h} - \frac{f(h)-f(-h)}{2h} \simeq 2\phi\epsilon_f\frac{f(\xi)}{2h}, \quad |\phi| \leq 1.$$

The difference between the quantity actually calculated and the quantity which one attempts to calculate may be as large as $\epsilon_f f(\xi)/h$; for example using $h = 10^{-40}$ and $\epsilon_m = 10^{-7}$ this gives a 'conditioning error' of $10^{33}f(\xi)$.

In practice much more sophisticated formulae than (1) above are used, and for these and for the corresponding higher-derivative formulae the error analysis is different and more complicated in detail. But invariably the theory contains the same overall feature. In a finite length calculation, the error is composed of two main parts: a discretisation error which increases as h becomes large and is zero for $h = 0$; and a 'conditioning' error due to amplification of round-off error in function evaluation, which increases as h becomes small and is infinite for $h = 0$.

The routine in this chapter has to take into account internally both these sources of error in the results. Thus it returns pairs of results, DER(j) and EREST(j) where DER(j) is an approximation to $f^{(j)}(x_0)$ and EREST(j) is an estimate of the error in the approximation DER(j). If the routine has not been misled, DER(j) and EREST(j) satisfy

$$|\mathrm{DER}(j) - f^{(j)}(x_0)| \leq \mathrm{EREST}(j).$$

In this respect, numerical differentiation routines are fundamentally different from other routines. The user does not specify an error criterion. Instead the routine provides the error estimate and this may be very large.

We mention here a terminological distinction. A fully automatic routine is one in which the user does not need to provide any information other than that required to specify the problem. Such a routine (at a cost in computing time) decides an appropriate internal parameter such as the step length h by itself. On the other hand a routine which requires the user to provide a step length h, but automatically chooses from several different formulae each based on the specified step length, is termed a semi-automatic routine.

The situation described above must have seemed rather depressing when hand machines were in common use. For example in the simple illustration one does not know the values of the quantities $f'''(\xi)$ or ϵ_f involved in the error estimates, and the idea of altering the value of h and starting again must have seemed appalling. However, by present-day standards, it is a relatively simple matter to write a program which carries out all the previously considered time-consuming calculations which may seem necessary. None of the routines envisaged for this chapter are particularly revolutionary in concept. They simply utilise the computer for the sort of task for which it was originally designed. It carries out simple tedious calculations which are necessary to estimate the accuracy of the results of other even simpler tedious calculations.

2.2 Examples of Applications for Numerical Differentiation Routines

(a) One immediate use to which a set of derivatives at a point is likely to be put is that of constructing a Taylor series representation:

$$f(x) = f(x_0) + \sum_{j=1}^{n}\frac{f^{(j)}(x_0)}{j!}(x-x_0)^j + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}, \quad |\xi - x_0| \leq x.$$

This infinite series converges so long as $|x - x_0| < R_c$ (the radius of convergence) and it is only for these values of x that such a series is likely to be used. In this case in forming the sum, the required accuracy in $f^{(j)}(x_0)$ diminishes with increasing j.

The series formed from the Taylor series using elementary operations such as integration or differentiation has the same overall characteristic. A technique based on a Taylor series expansion may be quite accurate, even if the individual derivatives are not, so long as the less accurate derivatives are associated with known small coefficients.

The error introduced by using n approximate derivatives DER(j) is bounded by

$$\sum_{j=1}^{n} \mathrm{EREST}(j)\ |x - x_0|^j / j!$$

Thus, if the user is prepared to base the result on a truncated Taylor series, the additional error introduced by using approximate Taylor coefficients can be readily bounded from the values of EREST(j). However, in an automatic code the user must be prepared to introduce some alternative approach in case this error bound turns out to be unduly high.

In this sort of application the accuracy of the result depends on the size of the errors in the numerical differentiation. There are other applications where the effect of large errors EREST(j) is merely to prolong a calculation, but not to impair the final accuracy.

(b) A simple Taylor series approach such as described in (a) is used to find a starting value for a rapidly converging but extremely local iterative process.

(c) The technique known as 'subtracting out the singularity' as a preliminary to numerical quadrature may be extended and may be carried out approximately. Thus suppose we are interested in evaluating

$$\int_0^1 x^{-(1/2)} \phi(x)\, dx,$$

we have an automatic quadrature routine available, and we have a routine available for $\phi(x)$ which we know to be an analytic function. An integrand function like $x^{-(1/2)}\phi(x)$ is generally accepted to be difficult for an automatic integrator because of the singularity. If $\phi(x)$ and some of its derivatives at the singularity $x = 0$ are known one may effectively 'subtract out' the singularity using the following identity:

$$\int_0^1 x^{-(1/2)} \phi(x) dx = \int_0^1 x^{-(1/2)} (\phi(x) - \phi(0) - Ax - Bx^2/2)\, dx + 2\phi(0) + 2A/3 + B/5 \qquad (2)$$

with $A = \phi'(0)$ and $B = \phi''(0)$.

The integrand function on the right of (2) has no singularity, but its third derivative does. Thus using numerical quadrature for this integral is much cheaper than using numerical quadrature for the original integral (in the left-hand side of (2)).

However, (2) is an identity whatever values of A and B are assigned. Thus the same procedure can be used with A and B being approximations to $\phi'(0)$ and $\phi''(0)$ provided by a numerical differentiation routine. The integrand would now be more difficult to integrate than if A and B were correct but still much less difficult than the original integrand (on the left-hand side of (2)). But, assuming that the automatic quadrature routine is reliable, the overall result would still be correct. The effect of using approximate derivatives rather than exact derivatives does not impair the accuracy of the result. It simply makes the result more expensive to obtain, but not nearly as expensive as if no derivatives were used at all.

(d) The calculation of a definite integral may be based on the Euler–Maclaurin expansion

$$\int_0^1 f(x) dx = \frac{1}{m} \sum_{j=0}^{m}{}'' f(j/m) - \sum_{s=1}^{l} \frac{B_{2s}}{2s!} \frac{(f^{(2s-1)}(1) - f^{(2s-1)}(0))}{m^{2s}} + O(m^{(-2l-2)}).$$

Here B_{2s} is a Bernoulli number. If one fixes a value of l then as m is increased the right-hand side (without the remainder term) approaches the true value of the integral. This statement remains true whatever values are used to replace $f^{(2s-1)}(1)$ and $f^{(2s-1)}(0)$. If no derivatives are available, and this formula is used (effectively with the derivatives replaced by zero) the rate of convergence is slow (like m^{-2}) and a large number of function evaluations may be used in calculating the trapezoidal rule sum for large m before a sufficiently accurate result is attained. However, if approximate derivatives are used, the initial rate of convergence is enhanced. In fact, in this example any derivative approximation which is closer than the approximation zero is helpful. Thus the use of inaccurate derivatives may have the effect of shortening the overall calculation, since a sufficiently accurate result may be obtained using a smaller value of m, without impairing the accuracy of the result. (The resemblance with Gregory's formula is superficial. Here l is kept fixed and m is increased, ensuring a convergent process.)

The examples given above are only intended to illustrate the sort of use to which approximate derivatives may be put. Very simple illustrations have been used for clarity, and in such simple cases there are usually more efficient approaches to the problem. The same ideas applied in a more complicated or restrictive setting may provide an efficient approach to a problem for which no simple standard approach exists.

3 Recommendations on Choice and Use of Available Routines

Note: refer to the Users' Note for your implementation to check that a routine is available.

(a) At the present time there is only one numerical differentiation routine available in this chapter, D04AAF. This is a semi-automatic routine for obtaining approximations to the first fourteen derivatives of a real valued function $f(x)$ at a specified point x_0. The user provides a FUNCTION representing $f(x)$, the value of x_0, an upper limit $n \leq 14$ on the order of the derivatives required and a step length h. The routine returns a set of approximations DER(j) and corresponding error estimates EREST(j) which hopefully satisfy

$$|\mathrm{DER}(j) - f^{(j)}(x_0)| \leq \mathrm{EREST}(j), \quad j = 1, 2, \ldots, n \leq 14.$$

We term this routine a semi-automatic routine because the user provides a step length h and this is not needed to specify the problem.

It is important that the error estimate EREST(j) is taken into consideration by the user before any use of DER(j) is made. The actual size of EREST(j) depends on the analytic structure of the function, on the word length of the machine used and on the step size h, and is difficult to predict. Thus the user has to run the routine to find out how accurate the results are. Usually the user will find the higher-order derivatives are successively more inaccurate and that past a certain order, say 10 or 11, the size of EREST(j) actually exceeds DER(j). Clearly when this happens the approximation DER(j) is unusable.

(b) We have investigated a fully automatic routine, which has the same calling sequence with the exception that a step length is not required. This routine finds an appropriate step length h for itself. The cost seems to be greater by a factor of 3 to 5 but the returned values of EREST(j) are usually smaller. It is our intention to develop such a routine only if there is a demand for it in which case the experience of users with the presently available semi-automatic routine will be very helpful.

(c) There is available in the algorithm section of CACM (Lyness and Ande (1971)) a semi-automatic Fortran routine for numerical differentiation of an analytical function $f(z)$ at a possibly complex abscissa z_0. This is a stable problem. It can be used for any problem for which D04AAF might be used and produces more accurate results, and derivatives of arbitrary high order. However, even if z_0 is real and $f(z)$ is real for z, this routine requires a user-supplied FUNCTION which evaluates $f(z)$ for complex values of z and it makes use of complex arithmetic.

(d) Routines are available in Chapter E02 to differentiate functions which are polynomials (in Chebyshev series representation) or cubic splines (in B-spline representation).

4 Routines Withdrawn or Scheduled for Withdrawal

None.

5 References

Lyness J N and Ande G (1971) Algorithm 413, ENTCAF and ENTCRE: Evaluation of normalised Taylor coefficients of an analytic function *Comm. ACM* **14 (10)** 669–675

Lyness J N and Moler C B (1967) Numerical differentiation of analytic functions *SIAM J. Numer. Anal.* **4 (2)** 202–210