

## Notes 3: Interpolation, numerical derivatives and numerical integration

### 3.1 Interpolation

#### 3.1.1 Lagrange polynomials

Suppose we are given  $n + 1$  points  $\{x_0, x_1, \dots, x_n\}$  in an interval  $[a_1, a_2]$  and  $n + 1$  associated values,  $\{y_0, y_1, \dots, y_n\}$ , which we assume are samples of the value of an unknown function,  $f(x)$ , at these values. A common computational task is to construct a function  $g(x)$  which approximates  $f(x)$  at any value of  $x$  in the interval  $[a_1, a_2]$ . This is known as interpolation. Of course, there is no way to solve the task as stated: an infinite amount of information can be encoded in  $f(x)$  which cannot be recovered from a finite number,  $n + 1$ , of function values. A reasonable candidate for  $g(x)$  is the polynomial of degree  $N$  which passes through all  $n + 1$  known points  $\{(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)\}$ . Such a polynomial exists and is unique. It is known as the Lagrange polynomial:

$$P(x) = \sum_{i=0}^n y_i p_i(x) \quad (3.1)$$

where

$$p_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}.$$

You should check that this formula does as it says. Note that it includes the special cases of linear interpolation ( $n = 1$ ) which will be familiar to everyone. Fig. 3.1 shows the Lagrange polynomial approximating the Gaussian function,

$$f(x) = e^{-\frac{x^2}{2}}, \quad (3.2)$$

using  $n = 5, 7, 9$  and  $11$  equally spaced samples in the interval  $[-3, 3]$ . Obviously, every interpolating polynomial can also be used for extrapolation of  $f(x)$  outside of the sample interval. Fig. 3.1 illustrates that this is (usually) not a terribly good idea.

#### 3.1.2 Neville's algorithm

The question of how to compute Eq. (3.1) efficiently is a non-trivial one. From a numerical point of view, it is much more common to compute the value of the interpolating polynomial at the required point  $x$  directly than to compute the coefficients and then evaluate the resulting polynomial at  $x$ . There is some detailed discussion in [1, chap.3] about why this is so. The best algorithm for computing  $P_n(x)$  for any value  $x$  is known as Neville's algorithm. It is a recursive procedure which builds the required polynomial order by order from intermediate polynomials of lower order, starting from constants (zeroth order polynomials!). We use the representation of Neville's algorithm shown in [1, chap. 3]. If you find this difficult to understand, it might be worth reading the Wikipedia article on Neville's algorithm which shows a slightly different representation [2] in which the intermediate polynomials are indexed differently.

The idea is to construct the tree shown in Fig. 3.2 from the  $n+1$  points  $\{(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)\}$ . The tree has  $n + 1$  levels, labeled from  $0$  to  $n$  starting from the right. At level  $i$  in the tree there are  $n + 1 - i$  nodes. Let us index the nodes by  $(i, j)$  where  $i$  is the level in the tree and  $j$  is the number of the node within that level. Each node,  $(i, j)$  is associated with a polynomial,  $P_{j, j+1 \dots j+i}(x)$ . The algorithm is specified by the following inductive argument:

- Let us assume that  $P_{j, j+1 \dots j+i}(x)$  is the unique polynomial of degree  $i$  which passes through the points  $\{(x_j, y_j), (x_{j+1}, y_{j+1}), \dots, (x_{j+i}, y_{j+i})\}$ .

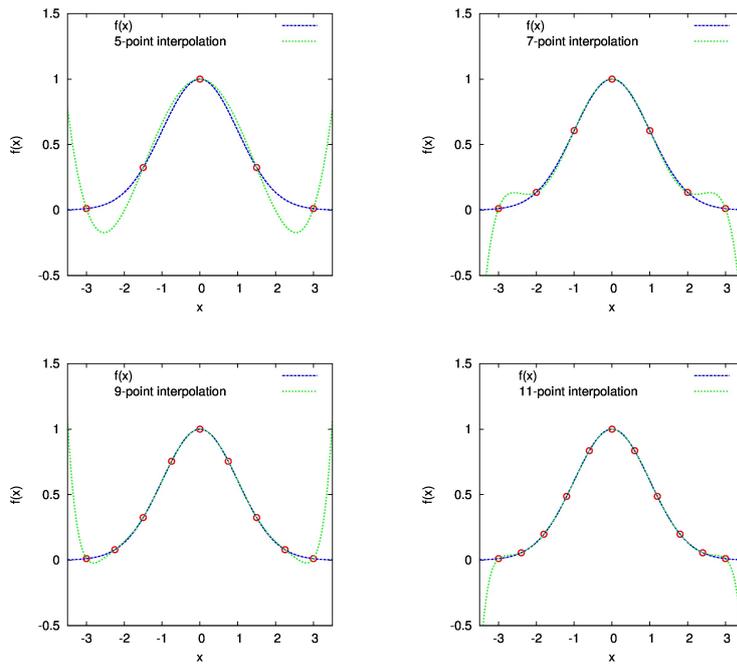


Figure 3.1: Lagrange interpolation of the Gaussian function, Eq. (3.2), with equally spaced sample points in the interval

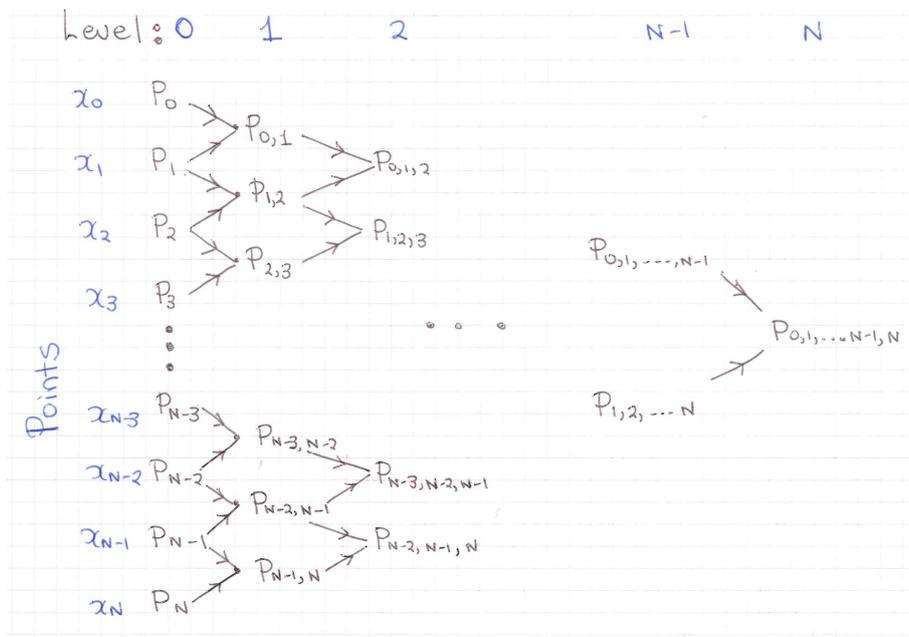


Figure 3.2: Graphical representation of the recursive structure of Neville's algorithm.

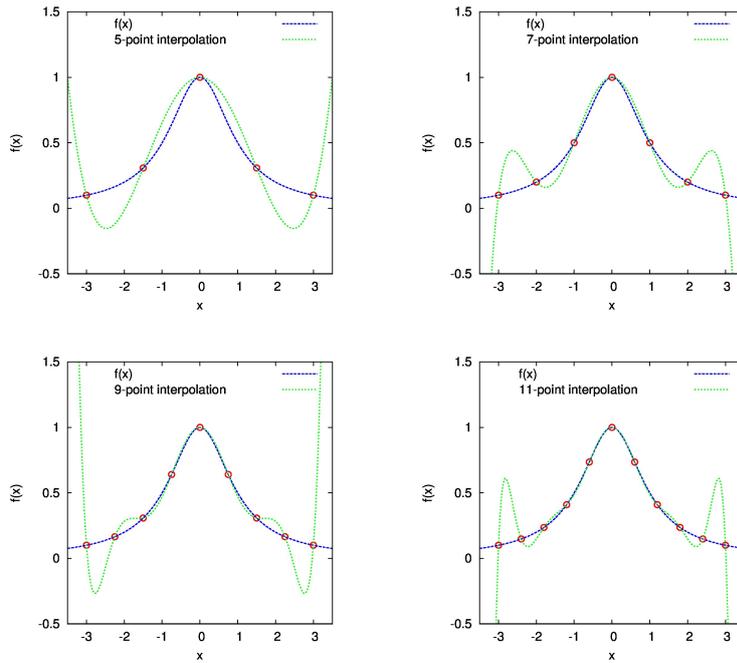


Figure 3.3: Lagrange interpolation of the Runge function, Eq. (3.5), with equally spaced sample points in the interval  $[-3, 3]$ .

- With this assumption, the polynomial

$$P_{j,j+1\dots j+i+1}(x) = \frac{(x - x_{j+i+1}) P_{j,j+1\dots j+i}(x) - (x - x_j) P_{j+1,j+2\dots j+i+1}(x)}{x_j - x_{j+i+1}} \tag{3.3}$$

is the unique polynomial of degree  $i+1$  which goes through the points  $\{(x_j, y_j), (x_{j+1}, y_{j+1}), \dots, (x_{j+i+1}, y_{j+i+1})\}$ . Check it to make sure it works!

- The recursion relation (3.3) gives us a way to step from level  $i$  to level  $i + 1$  in Fig. 3.2. All that remains is to provide the starting condition at level 0:

$$P_i(x) = y_i. \tag{3.4}$$

This is clearly the unique polynomial of degree 0 passing through the point  $(x_i, y_i)$ . By induction, the polynomial  $P_{0,1\dots N}(x)$  at level  $N$  is the required Lagrange polynomial.

### 3.1.3 The concept of conditioning

The condition number of a function with respect to an argument measures how much the output value of the function can change for a small change in the input argument. It is used to measure how sensitive a function is to changes or errors in the input. This concept is often applied in a context where the "function" is the solution of a numerical problem and the "argument" is the data going into the problem. In the present context, the problem is the calculation of the interpolating polynomial and the argument is the set of sample points. A problem with a low condition number is said to be well-conditioned, while a problem with a high condition number is said to be ill-conditioned. It is important to realise that the condition number is a property of the problem itself, not of the algorithm used to solve it or the precision with which this algorithm is implemented. For ill-conditioned problems, however, roundoff error coming from finite precision can act as a source of error in the input which can lead to large and unpredictable variations in the output.

In the context of interpolation, we expect intuitively that as the number of samples increases (assuming that we keep the interval  $[a_1, a_2]$  fixed) then the interpolating polynomial,  $P(x)$ , will converge to  $f(x)$ . Although this seemed to be the case with the example shown in Fig. 3.1, such convergence

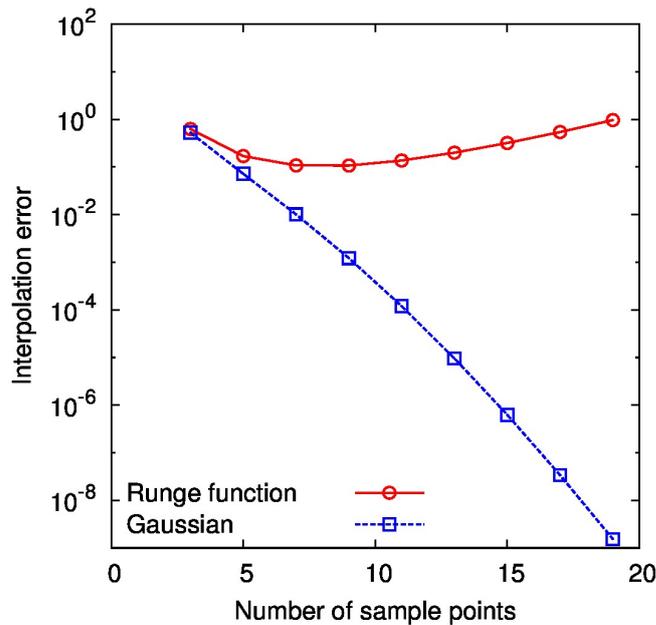


Figure 3.4: Interpolation error as a function of number of sample points for Lagrange interpolation of the Runge function, Eq. (3.5), and Gaussian function, Eq. (3.2), with equally spaced sample points in the interval  $[-3, 3]$ . For the Gaussian function the interpolation error decreases monotonically with the order and the interpolating polynomial converges to the true function. For the Runge function, the error goes down at first but then starts to grow again!

is in fact not assured. Consider the so-called Runge function [3]

$$f(x) = \frac{1}{1+x^2}, \quad (3.5)$$

which is not all that different from the Gaussian example used above. Fig. 3.3 shows the Lagrange polynomial approximating this function using  $n = 5, 7, 9$  and 11 equally spaced samples in the same interval  $[-3, 3]$ . Despite the fact that Eq. (3.5) is not qualitatively very different from Eq. (3.2),  $f(x)$  becomes increasingly badly represented near the edges of the interval as the number of sample points,  $n$ , increases. The oscillations between the sample points actually get larger as  $n$  increases. The interpolation error can be quantified by calculating

$$E_n = \sqrt{\int_{-a}^a |P_n(x) - f(x)|^2 dx}. \quad (3.6)$$

This is plotted as a function of  $n$  for the Gaussian and Runge examples in Fig. 3.4. The fact that larger  $n$  does not necessarily imply smaller error was an important discovery in the history of numerical analysis attributed to Runge. The oscillations of the interpolating polynomial responsible for this effect are known as the Runge Phenomenon. These oscillations are not due to numerical instability but are reflecting the fact that the problem of Lagrange interpolation with equally spaced sample points is ill-posed in the sense that small variations in the input parameters (the sample points) produce large variations in the resulting interpolating polynomial. The problem of Lagrange interpolation is an ill-conditioned problem. From a numerical perspective Lagrange interpolation at high orders is a dangerous business since there is the omnipresent danger that small numerical errors in the sample values will be amplified by this intrinsic instability. This effect is closely related to the problem of overfitting which you encountered in a different guise in MA930. There is a large literature on interpolation methods which go beyond Lagrange in an attempt to get around these difficulties.

## 3.2 Numerical differentiation

We now turn our attention to another common computational task: the evaluation of the derivative of a function,  $f(x)$ , at a point,  $x$ .

### 3.2.1 Finite difference approximations of derivatives

Suppose we have a list of the values,  $\{f_i : i = 1 \dots N\}$  of a function,  $f(x)$ , at a set of discrete points,  $\{x_i\}$ . Let us further assume, for simplicity, that these points are equally separated with interval  $h$ :  $x_i = x_0 + ih$ . Intuitively, derivatives of  $f$  at a point can be approximated by differences of the  $f_i$  at adjacent points:

$$\frac{df}{dt}(x_i) \approx \frac{f_{i+1} - f_i}{h}.$$

This intuition is made precise using Taylor's Theorem:

**Theorem 3.2.1.** *If  $f(x)$  is a real-valued function which is differentiable  $n + 1$  times on the interval  $[x, x + h]$  then there exists a point,  $\xi$ , in  $[x, x + h]$  such that*

$$\begin{aligned} f(x+h) &= f(x) + \frac{1}{1!} h \frac{df}{dx}(x) + \frac{1}{2!} h^2 \frac{d^2f}{dx^2}(x) + \dots \\ &\quad + \frac{1}{n!} h^n \frac{d^n f}{dx^n}(x) + h^{n+1} R_{n+1}(\xi) \end{aligned} \quad (3.7)$$

where

$$R_{n+1}(\xi) = \frac{1}{(n+1)!} \frac{d^{n+1}f}{dx^{n+1}}(\xi).$$

Note that the theorem does not tell us the value of  $\xi$ . We will use Theorem (3.2.1) a lot. Adopting "primed" notation to denote derivatives, Taylor's Theorem to first order ( $n = 1$ ) tells us that

$$f(x+h) = f(x) + h f'(x) + O(h^2).$$

In the light of the above discussion of approximating derivatives using differences, we can take  $x = x_i$  and re-arrange this to give

$$f'(x_i) = \frac{df}{dx}(x_i) = \frac{f_{i+1} - f_i}{h} + O(h). \quad (3.8)$$

This is called the first order forward difference approximation to the derivative of  $v$ . Taylor's theorem makes the sense of this approximation precise: we make an error proportional to  $h$  (hence the terminology "first order"). Thus it is ok when  $h$  is "small".

We could equally well use the Taylor expansion of  $f(x-h)$ :

$$f(x-h) = f(x) - h f'(x) + O(h^2)$$

Rearranging this gives what is called the first order backwards difference approximation:

$$f'(x_i) = \frac{f_i - f_{i-1}}{h} + O(h). \quad (3.9)$$

It also obviously makes an error of order  $h$ . To get a first order approximation of the derivative requires that we know the values of  $f$  at two points. The key idea of finite difference methods is by using more points we can get better approximations. Better means the error made in the approximation is proportional to a higher power of  $h$ . The general method is to consider a linear combination of the values of the function at a set of points surrounding the point  $x$  at which we want to approximate the derivative. We use Taylor's theorem to express everything in terms of the value of the function and its derivatives at the point  $x$  and then choose the coefficients in the linear combination to cancel as many powers of  $h$  as possible. This is best illustrated by an example. Suppose we wish to derive a finite difference approximation for the derivative of  $f(x)$  at  $x_i$  using the values of  $f(x)$  at the three points  $x_{i-1} = x_i - h$ ,  $x_i$  and  $x_{i+1} = x_i + h$ . We will need the Taylor expansions at  $x_{i-1}$  and  $x_{i+1}$ :

$$\begin{aligned} f_{i+1} &= f_i + h f'(x_i) + \frac{1}{2} h^2 f''(x_i) + O(h^3) \\ f_{i-1} &= f_i - h f'(x_i) + \frac{1}{2} h^2 f''(x_i) + O(h^3). \end{aligned}$$

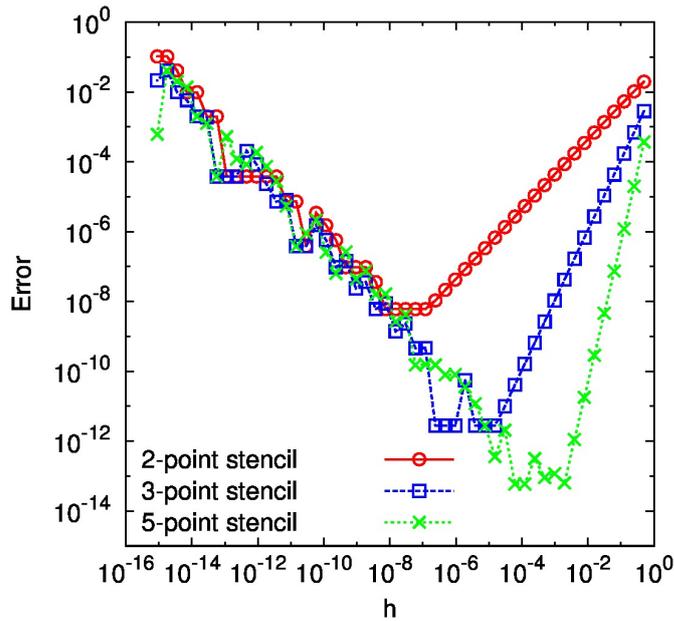


Figure 3.5: Finite difference approximation to the derivative of  $f(x) = \sqrt{x}$  at  $x = 2$ . The plot shows the error as a function of  $h$  for several different finite difference formulae.

We now consider the linear combination  $a_1 f_{i-1} + a_2 f_i + a_3 f_{i+1}$  and use these Taylor expansions. We obtain

$$a_1 f_{i-1} + a_2 f_i + a_3 f_{i+1} = (a_1 + a_2 + a_3) f_i + (a_3 - a_1) h f'(x_i) + \frac{1}{2} (a_3 + a_1) h^2 f''(x_i) + O(h^3).$$

We now choose the values  $a_1, a_2$  and  $a_3$  to satisfy the equations

$$\begin{aligned} a_1 + a_2 + a_3 &= 0 \\ a_3 - a_1 &= 1 \\ a_3 + a_1 &= 0. \end{aligned}$$

This cancels the terms proportional to  $h^0$  and  $h^2$ . We obtain  $a_1 = -\frac{1}{2}, a_2 = 0$  and  $a_3 = \frac{1}{2}$ . Solving for  $f'(x_i)$  then gives

$$f'(x_i) = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2). \tag{3.10}$$

This is called the centred difference approximation. This procedure generalises easily to unequally spaced points although the formulae are not so neat. An astute reader will have noticed that we could have chosen  $a_1, a_2$  and  $a_3$  so as to cancel the terms proportional to  $h^0$  and  $h^1$  and obtained an approximation for  $f''(x_i)$ . This is indeed how finite difference formulae for higher order derivatives are obtained.

The set of points underpinning a finite-difference approximation is known as the "stencil". Notice that by using a 3-point stencil, we obtained an approximation which is second order accurate in  $h$ . A straightforward extension to stencils with more points allow higher order approximations to be derived. For example, a 5-point stencil leads to a 4th order accurate finite difference formula:

$$f'(x_i) = \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12h} + O(h^4). \tag{3.11}$$

From this discussion, it may seem optimal to choose  $h$  as small as possible in order to reduce the error. Due to roundoff error, the situation is not so simple when finite difference methods are implemented in finite precision arithmetic. In fact it can be quite difficult to determine a-priori what is

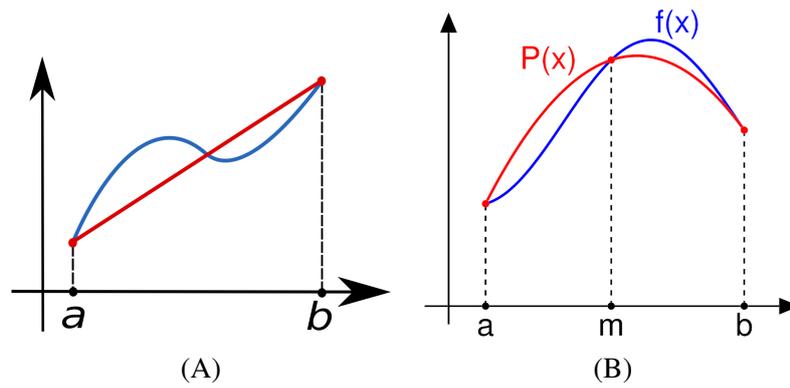


Figure 3.6: (from Wikipedia) Graphical illustration of (A) the Trapezoidal Rule and (B) Simpson's Rule.

the optimal value of  $h$ . See Fig. 3.5 to see an illustration of this. With higher order finite difference formulae,  $h$  does not have to be very small (around  $h = 0.001$  for Eq. (3.11)) before we get to a point where decreasing  $h$  further starts to make the estimated value of the derivative worse rather than better!

### 3.3 Numerical integration

We now turn to the other basic operation of elementary calculus: integration. A common computational task is to estimate the definite integral of some function,  $f(x)$  over an interval  $[a, b]$ :

$$I = \int_a^b f(x) dx.$$

#### 3.3.1 Trapezoidal Rule and Simpson's Rule

The classical numerical integration rules construct an approximation from the values of  $f(x)$  sampled at  $m + 1$  points,  $x_0, \dots, x_m$ , in the interval  $[a, b]$ . We think of the interval as being fixed:  $x_0 = a$  and  $x_n = b$ . Thus as  $m$  increases, the points get closer together. We further assume that the points are equally spaced with separation  $h = (b - a)/n$ . The trapezoidal rule, shown in Fig 3.6(A), uses two values ( $m = 1$ ),  $f_0$  and  $f_1$  :

$$I = \frac{h}{2} [f_0 + f_1] + O(h^3). \tag{3.12}$$

Simpson's rule, shown in Fig 3.6(B), uses three values ( $m = 2$ ),  $f_0$ ,  $f_1$  and  $f_2$ :

$$I = \frac{h}{3} [f_0 + 4f_1 + f_2] + O(h^5). \tag{3.13}$$

These formulae and higher order ones (involving the use of more points inside the interval  $[a, b]$ ) can be derived by using Eq. (3.1) to interpolate the function  $f(x)$  in the interval  $[a, b]$  and then use the fact that polynomials are easily integrated analytically. The determination of the error estimate is less straightforward and is detailed (for those interested) for the trapezoidal rule in [4].

In practice, the region of integration consists of more than a single interval. We can divide a larger interval into  $n$  non-overlapping subintervals each of length  $h = (b - a)/n$ . Eq.(3.12) or (3.13) can then be applied to each subinterval and the result summed to estimate the integral over the entire interval. It is instructive to write out the formulae explicitly to help see the general pattern. Applying Eq.( 3.12) to the pair of intervals  $(x_0, x_1), (x_1, x_2)$  gives

$$I = h \left[ \frac{f_0}{2} + f_1 + \frac{f_2}{2} \right] + 2O(h)^3$$

which generalises with  $n$  intervals to the so-called "extended" trapezoidal rule:

$$I = h \left[ \frac{f_0}{2} + \sum_{i=1}^{n-1} f_i + \frac{f_n}{2} \right] + O\left(\frac{(b-a)^3}{n^2}\right). \tag{3.14}$$

The error term comes from making  $n$  errors of  $O(h^3)$  (see Eq. (3.12)) with  $h = (b - a)/n$ . Similarly, applying Eq. (3.13) to the pair of pairs of intervals  $(x_0, x_1)$ ,  $(x_1, x_2)$  and  $(x_2, x_3)$ ,  $(x_3, x_4)$  gives

$$I = h \left[ \frac{f_0}{3} + \frac{4f_1}{3} + \frac{2f_2}{3} + \frac{4f_3}{3} + \frac{f_4}{3} \right] + 2O(h)^5$$

which generalises with  $n$  intervals to the so-called "extended" Simpson's rule (we assume for convenience of notation that  $n$  is even):

$$I = h \left[ \frac{f_0}{3} + \frac{2}{3} \sum_{i=1}^{n/2-1} f_{2i} + \frac{4}{3} \sum_{i=1}^{n/2} f_{2i-1} + \frac{f_n}{3} \right] + O\left(\frac{(b-a)^5}{n^4}\right). \tag{3.15}$$

The error term comes from making  $n$  errors of  $O(h^5)$  (see Eq. (3.13)) with  $h = (b - a)/n$

A simple algorithm like Eq. (3.15) does a remarkably good job for many integrals. If you want to be really clever, it is possible to recursively nest extended integration rules inside each other on a set of successively finer subdivisions of the interval to produce a very rapidly converging scheme known as Romberg's method.

### 3.3.2 Numerical estimation of "improper" integrals

Integration rules like Eq. (3.14) and (3.15) are of no immediately use if the domain of integration is infinite (either  $a = \infty$  or  $b = \infty$ ) or if the function  $f(x)$  has a singularity on the boundary of the domain of integration. Any such singularity is necessarily integrable. Otherwise the integral would not exist. If a singularity exists at a known point,  $c$ , in the interior of the domain then we can split the domain into two  $([a, c]$  and  $[c, b])$  so that the singularity is on the boundary. Such integrals are referred to as "improper" integrals.

While one can think of approaches to attempt to estimate such integrals by brute force using the tools above, a much more elegant approach is to use a change of variables which turns the improper integral into a proper one. For example, if the domain of integration is infinite, we can use the change of variables  $x = y^{-1}$ :

$$\int_a^b f(x) dx = \int_{1/b}^{1/a} y^{-2} f(y^{-1}) dy \tag{3.16}$$

which produces a finite domain of integration as either  $a$  or  $b$  go to infinity.

Now suppose  $f(x)$  has a singularity at the lower limit of integration:  $f(x) \sim (x - a)^{-\gamma}$  as  $x \rightarrow a$ . We must have  $\gamma < 1$  in order for this singularity to be integrable. We can use the change of variables

$$y = (x - a)^{1-\gamma}$$

to obtain the integral

$$I = \int_a^b f(x) dx = \int_0^{(b-a)^{1-\gamma}} F(y) dy$$

where

$$F(y) = \frac{1}{1-\gamma} y^{\frac{\gamma}{1-\gamma}} f\left(y^{\frac{1}{1-\gamma}} + a\right)$$

and  $F(y) \rightarrow 1$  as  $y \rightarrow 0$ . In both these cases, Eq. (3.14) or (3.15) can be directly applied to the transformed integral. This is an example of a general principle of numerical simulation: if it is possible to use a change of variables to remove explicit infinities from a problem before putting it on a computer then life will likely go much more smoothly.

### 3.3.3 Equivalence of numerical integration and solution of ordinary differential equations

If there is a singularity at an unknown point in the interior of the domain of integration then we cannot apply any of the above methods. For such problems it is better to go "back to the drawing board" and exploit the fact that computing the integral

$$I = \int_a^b f(x) dx$$

is equivalent to solving the differential equation

$$\frac{dy}{dx} = f(x),$$

with initial condition  $y(a)=0$ . The required integral is  $I = y(b)$ . Check it yourself. An advantage of this equivalence is that there are powerful general numerical methods available to solve ordinary differential equations which we shall get to later.

### Bibliography

- [1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical recipes: The art of scientific computing (3rd ed.). Cambridge University Press, New York, NY, USA, 2007.
- [2] Neville's algorithm, 2014. [http://en.wikipedia.org/wiki/Neville's\\_algorithm](http://en.wikipedia.org/wiki/Neville's_algorithm).
- [3] Runge's phenomenon, 2014. [http://en.wikipedia.org/wiki/Runge's\\_phenomenon](http://en.wikipedia.org/wiki/Runge's_phenomenon).
- [4] D. Cruz-Uribe and C. J. Neugebauer. An elementary proof of error estimates for the trapezoidal rule. Mathematics Magazine, 76(4):303, 2003. [http://www.maa.org/sites/default/files/An\\_Elementary\\_Proof30705.pdf](http://www.maa.org/sites/default/files/An_Elementary_Proof30705.pdf).