INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

H4.SMR/854-5

College on Computational Physics

15 May - 9 June 1995

Introductory Lectures

C. Rebbi

Boston University
Boston, USA

1.4. Numerical Integration

We would like to evaluate the integral of a function $f(x)$ in the range $x_a - x_b$. The numerical algorithms make use of the values of the function, assumed either known or calculable, at definite points in the interval of integration and can be generally grouped into three classes, according to whether the points are equally spaced, regularly spaced but not equidistant, or chosen at random. We will deal with the former class of algorithms here, leaving the other two (Gaussian integration and generalizations, and Monte Carlo integration) to later chapters.

Let us assume then that we know $f(x)$ for the set of equidistant values, $x_0 = x_a$, $x_1, x_2, \ldots, x_n = x_b$, of the argument. Let $\delta = (x_n - x_0)/n$ be the spacing between points. The simplest method, known as the *trapezoidal rule*, consists in approximating $f(x)$ by a straight line between x_i and x_{i+1}. This gives

$$f(x) \approx [f(x_i)(x_{i+1} - x) + f(x_{i+1})(x - x_i)]\frac{1}{\delta} \tag{1.4.1}$$

which can be integrated to give,

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = \frac{1}{2}[f(x_i) + f(x_{i+1})]\,\delta \ . \tag{1.4.2}$$

Adding the contributions from all subintervals making up the interval of integration we get,

$$\int_{x_0}^{x_n} f(x)\,dx = \left[\frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right]\delta \ . \tag{1.4.3}$$

How accurate is this result? To answer this question, expand $f(x)$ about x_i,

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \mathcal{O}\left((x - x_i)^2\right)$$

and estimate f' from the same expression, to finally get

$$f(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{\delta}(x - x_i) + \mathcal{O}(\delta^2) \tag{1.4.4}$$

which coincides with the linear interpolation formula we used before, but with the additional information that the error is of order δ^2. We can therefore rewrite Eq. (1.4.2) more correctly as

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = \frac{1}{2}[f(x_i) + f(x_{i+1})]\,\delta \ + \mathcal{O}(\delta^3) \tag{1.4.5}$$

Adding the contributions from all the subintervals will result in the accumulation of the N intervals of length δ, so that, in general, the error in the trapezoidal rule will be of order δ^2.

An alternative formula for the trapezoidal rule can be derived when the function to be integrated is known at the midpoints, $x_i+\delta/2$, $(i=0,\ldots,n-1)$ of the discretization intervals. In this case, we can also write

$$f(x)=f\left(x_i+\frac{\delta}{2}\right)+f'\left(x_i+\frac{\delta}{2}\right)\left(x-x_i-\frac{\delta}{2}\right)+\mathcal{O}\left(\delta^2\right). \tag{1.4.6}$$

Integrating this equation over the interval $[x_i,x_{i+1}]$ we see that the second term, proportional to f', vanishes by symmetry, so we are left with

$$\int_{x_i}^{x_{i+1}} f(x)\,dx=f\left(x_i+\frac{\delta}{2}\right)\delta+\mathcal{O}\left(\delta^3\right) \tag{1.4.7}$$

leading to

$$\int_{x_0}^{x_n} f(x)\,dx=\left[f\left(x_0+\frac{\delta}{2}\right)+f\left(x_1+\frac{\delta}{2}\right)+\cdots+f\left(x_{n-1}+\frac{\delta}{2}\right)\right]\delta+\mathcal{O}\left(\delta^2\right). \tag{1.4.8}$$

Is it possible to obtain approximations with an accuracy higher than $\mathcal{O}(\delta^2)$? There are several ways of doing this. The simplest is to Taylor expand $f(x)$ to higher order, and proceed as we did for the trapezoidal rule. Consider expanding $f(x)$ about x_i to fourth order in δ:

$$f(x)=f(x_i)+f'(x_i)(x-x_i)+\frac{1}{2}f''(x_i)(x-x_i)^2+\frac{1}{6}f'''(x_i)(x-x_i)^3+\mathcal{O}\left(\delta^4\right), \tag{1.4.9}$$

which integrates to,

$$\int_{x_i}^{x_{i+1}} f(x)\,dx=f(x_i)\,\delta+\frac{1}{2}f'(x_i)\,\delta^2+\frac{1}{6}f''(x_i)\,\delta^3+\frac{1}{24}f'''(x_i)\,\delta^4+\mathcal{O}\left(\delta^5\right). \tag{1.4.10}$$

Of course, the error in Eq. (1.4.10) will be of order δ^5 only if we are able to calculate the derivatives on the right-hand side with a sufficient degree of accuracy. The equation, as it stands, calls for an accuracy $\mathcal{O}(\delta^3)$ in f', $\mathcal{O}(\delta^2)$ in f'' and $\mathcal{O}(\delta)$ in f'''. However, we can get by without having to calculate $f'(x_i)$ and $f'''(x_i)$ to this degree of accuracy, indeed, without having to calculate them at all, if, rather than using the expansion in Eq. (1.4.9) to calculate the integral from x_i to x_{i+i}, we use it for calculating the integral in the larger interval from x_{i-1} to x_{i+1}. Because of the symmetry, then, the terms which are linear and cubic in $(x-x_i)$ do not contribute to the integral, and we are left with,

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\;dx=2f(x_i)\,\delta+\frac{1}{3}f''(x_i)\,\delta^3+\mathcal{O}\left(\delta^5\right), \tag{1.4.11}$$

with the only requirement that we must calculate $f''(x_i)$ with error of order δ^2 in order to achieve a total local error of order δ^5. To proceed further, let us put aside for a moment the problem of numerical integration to turn to that of the numerical calculation of a derivative.

1.5. Numerical differentiation

Sometimes the derivatives of a given function can be calculated analytically and their analytic expressions used directly in the context of a numerical calculation. Often, however, this is either not possible or not practical; the function may be known only through the values it takes at definite points, and no analytic expression may be available, or the analytic form of the function may be complex enough that, even if the derivatives can be calculated analytically, it is numerically too time–consuming to calculate their values. But even if this is not the case, the explicit calculation of the derivatives may simply be redundant. The result we obtained in Eq. (1.4.11) is a case in point. A calculation of the left–hand side with error of order δ^5 requires only knowledge of $f''(x_i)$ with error of order δ^2 and nothing is gained in precision (because of the other terms which are neglected on the right–hand side) from a more accurate calculation of f''. As we shall soon see, f'' can be calculated with error of order δ^2 from the values of f at x_i, x_{i-1} and x_{i+1}, which are, in any case, required to find the approximate integral over the whole domain. Therefore, an explicit numerical evaluation of $f''(x_i)$ would only be a waste of time.

The Taylor expansion formula in Eq. (1.4.9) constitutes the basis for the numerical calculation of derivatives. By choosing for x the values of neighboring points x_{i+1}, x_{i-1}, etc, where the function is assumed known, and truncating the expansion, one can obtain one or more equations where the derivatives, $f^{(k)}(x_i)$, appear as unknowns. The approximate value of the k^{th} derivative can then be computed in terms of the values of the function which enter in the expansion.

Thus, for example, taking $x = x_{i+1}$ and truncating the expansion to $\mathcal{O}(\delta^2)$ we get

$$\begin{aligned} f(x_{i+1}) &= f(x_i) + f'(x_i)(x_{i+1} - x_i) + \mathcal{O}(\delta^2) \\ &= f(x_i) + f'(x_i)\,\delta + \mathcal{O}(\delta^2) \end{aligned} \tag{1.5.1}$$

Solving this for f' we recover the result already obtained in Eq. (1.4.4), namely,

$$f'(x_i) = [f(x_{i+1}) - f(x_i)]\,\frac{1}{\delta} + \mathcal{O}(\delta)\,.$$

This expression is known as the *forward difference* approximation to the first derivative.

Similarly, by setting instead $x = x_{i-1}$, we can derive the equation

$$f'(x_i) = [f(x_i) - f(x_{i-1})]\,\frac{1}{\delta} + \mathcal{O}(\delta)\,. \tag{1.5.2}$$

This is known as the *backward difference* approximation to the first derivative. Both the forward difference and the backward difference approximations have an error of order δ.

One can obtain a more accurate expression for the first derivative by carrying the Taylor series expansion to order δ^2 and combining the equations one gets setting $x = x_{i+1}$ and $x = x_{i-1}$.

These give,

$$f(x_{i+1}) = f(x_i) + f'(x_i)\,\delta + \frac{1}{2}f''(x_i)\,\delta^2 + \mathcal{O}(\delta^3) \tag{1.5.3}$$

and

$$f(x_{i-1}) = f(x_i) - f'(x_i)\,\delta + \frac{1}{2}f''(x_i)\,\delta^2 + \mathcal{O}(\delta^3)\,. \tag{1.5.4}$$

Subtracting these two expressions, the terms with the unknown $f''(x_i)$ drop out and we find,

$$f(x_{i+1}) - f(x_{i-1}) = 2f'(x_i)\,\delta + \mathcal{O}(\delta^3)\,, \tag{1.5.5}$$

or,

$$f'(x_i) = [f(x_{i+1}) - f(x_{i-1})]\,\frac{1}{2\delta} + \mathcal{O}(\delta^2)\,. \tag{1.5.6}$$

This is the *central difference* approximation to the derivative and is one order of magnitude more accurate in δ than either the forward or the backward approximations.

If we add Eqs. (1.5.4) and (1.5.5) instead of subtracting them, then the terms with f' drop out and we are left with an equation for $f''(x_i)$. Indeed, also the terms $\mathcal{O}(\delta^3)$ cancel if we add the equations, because they come from the term $\frac{1}{6}f'''(x_i)(x - x_i)^3$ which has opposite signs for $x = x_{i+1}$ and $x = x_{i-1}$. Thus we find

$$f(x_{i+1}) + f(x_{i-1}) = 2f(x_i) + f''(x_i)\,\delta^2 + \mathcal{O}(\delta^4)\quad, \tag{1.5.7}$$

which then leads to,

$$f''(x_i) = [f(x_{i+1}) + f(x_{i-1}) - 2f(x_i)]\,\frac{1}{\delta^2} + \mathcal{O}(\delta^2)\quad, \tag{1.5.8}$$

which is a useful and accurate approximation to f''.

Proceeding along similar lines and making use of the values $f(x)$ takes on various neighboring points one can obtain numerical approximation formulas for the higher derivatives and also formulas which are accurate to higher orders in δ. We will not write down any of these expressions. It is a straightforward exercise to derive them, generalizing the formulas we have derived so far, and they can also be found in many books specifically devoted to numerical methods. Insofar as the first and second derivatives are concerned, the formulas in Eqs. (1.5.6) and (1.5.8) provide approximations which are adequate for most applications.

In an actual computer calculation, the error in the evaluation of a derivative will not only depend on the terms neglected in the Taylor expansion, but also on the magnitude of the roundoff errors. The presence of roundoff errors is a crucial factor in any numerical calculation. If it were not for these, one could obtain arbitrarily accurate values for the derivative with the sophisticated formulas by simply taking δ small enough.

The roundoff errors affect the mantissa of the floating–point numbers in the computer, thus they typically represent relative errors in the values of $f(x)$. Let such errors be of order ϵ. For the standard representation of single–precision numbers, with an 8–bit exponent, 1 sign bit and a 23–bit mantissa, $\epsilon \approx 10^{-7}$. Then, if we consider, for instance, the central difference formula for f', we can expect a roundoff error of order $|f(x_i)|\epsilon$ in the difference $f(x_{i+1}) - f(x_{i-1})$. The total error will then be

$$\Delta f' \approx |f|\frac{\epsilon}{\delta} + c|f|\delta^2\quad. \tag{1.5.9}$$

with c some constant.

On the right–hand side we have assumed that the coefficient of the $\mathcal{O}(\delta^2)$ term, which is proportional to the third derivative, f''', has the same order of magnitude as f. Minimizing the $\Delta f'$ with respect to δ, we find a relative error,

$$\frac{\Delta f'}{|f|} \approx c' \epsilon^{\frac{2}{3}} \tag{1.5.10}$$

Thus, we cannot do better than $10^{-\frac{14}{3}}$, or roughly 10^{-5}, for the relative error in the numerical calculation of f' by the central difference formula, if we calculate in single precision. Moreover, the minimum occurs at $\delta \approx \epsilon^{\frac{1}{3}}$, that is, $\delta \approx 10^{-2}$ to 10^{-3}. Trying to achieve a higher accuracy by taking δ too small will only worsen the result.

These are essential notions for the practitioner of computational physics, comparable in value to a good knowledge of the laboratory apparatus for an experimentalist. It is extremely useful and instructive to verify the above considerations by experimenting with the computer. You will soon develop a strong feeling for the possibilities and limitations of the computational methods that characterizes the accomplished computational scientist by writing a few lines of code to test the performance of the various algorithms as you go through this book.

1.6. Simpson's formula

We now return to Eq. (1.4.11) and to the problem of developing a more accurate scheme for numerical integration. Inserting into that equation the expression found in Eq. (1.5.8) for $f''(x_i)$ one gets,

$$\begin{aligned} \int_{x_{i-1}}^{x_{i+1}} f(x)\, dx &= 2f(x_i)\,\delta + \frac{1}{3}\left[f(x_{i+1}) + f(x_{i-1}) - 2f(x_i)\right]\delta + \mathcal{O}(\delta^5) \\ &= \left[\frac{1}{3}f(x_{i+1}) + \frac{4}{3}f(x_i) + \frac{1}{3}f(x_{i+1})\right]\delta + \mathcal{O}(\delta^5) \quad . \end{aligned} \tag{1.6.1}$$

This can now be extended to a formula for the integral over a finite domain, provided only that this region is divided into an even number of subintervals. Doing this one obtains an expression for the integral known as *Simpson's rule*:

$$\begin{aligned} \int_{x_0}^{x_{2n}} f(x)\, dx = \Big[&\frac{1}{3}f(x_0) + \frac{4}{3}f(x_1) + \frac{2}{3}f(x_2) + \frac{4}{3}f(x_3) + \cdots + \\ &+ \frac{2}{3}f(x_{2n-2}) + \frac{4}{3}f(x_{2n-1}) + \frac{1}{3}f(x_{2n})\Big]\delta + \mathcal{O}(\delta^4) \quad , \end{aligned} \tag{1.6.2}$$

with $\delta = (x_{2n} - x_0)/2n$ and $x_\ell = x_0 + \ell\delta$.

Simpson's rule provides a very useful and accurate formula for approximate numerical integration. The alternation of coefficients with different magnitudes may

appear strange at first sight, but there is nothing magical about it. Indeed, it is possible to derive an equally accurate formula where one is not restricted to an even number of subintervals and all the interior coefficients are equal to 1, but with more elaborate end point corrections. The derivation of this result is left as an exercise. As we have seen, the premises for all such formulas are found in a careful treatment of the expansion of the function in the neighborhood of the integration points. Once these are well understood, the workings of all integration formulas become clear.

... the possible treatments of end–point singularities (integrable singularities, of course) for finite values of the variable of integration. There could be an integrable singularity, such as $1/\sqrt{x}$ for $x = 0$, making the integrand infinite at one of the points used in the numerical integration formulas: this of course renders their straightforward application impossible. But even if the nature of the singularity, for instance an integrand behaving like $\sqrt{x}$ for $x = 0$, were not such as to lead to infinities in the integration formulas, one should still use caution. Indeed, the derivation of the formulas and the estimate of the error are all based on the assumption that a sufficient number of derivatives of the integrand exist and are bounded. If this condition is not fulfilled, as in the case of $\sqrt{x}$, the error can be much worst than one would estimate for a normal integrand. Then again one should remedy to the situation either by changing variable of integration so as to produce a non-singular integrand (setting $x = y^2$, for instance, gives $dx/\sqrt{x} = 2\,dy$ and $\sqrt{x}\,dx = 2y^2\,dy$, both regular at $y = 0$), or by separating the integral into two parts, one without any singularity, which will be dealt with numerically, and one containing the singularity, which will be treated analytically after the integrand has been simplified by some suitable expansion.

It is actually of some interest to see how a formula like Simpson's formula fails to give an accurate result in presence of a singularity. To look at this, let us consider the integral from 0 to 1 of the very simple expression $\sqrt{x}$. Let us divide the interval into $2N$ subintervals of width $\delta = 1/2N$ and consider the approximation provided by Simpson's formula for the interval between $x = 2n\delta$ and $x = (2n+2)\delta$. This is

$$\begin{aligned} \Delta I_{approx} &= \frac{\delta}{3}\left[\sqrt{2n\delta} + 4\sqrt{(2n+1)\,\delta} + \sqrt{(2n+2)\,\delta}\right] \\ &= \frac{\delta^{\frac{3}{2}}}{3}\left[\sqrt{2n} + 4\sqrt{2n+1} + \sqrt{2n+2}\right] \end{aligned} \tag{1.7.1}$$

This is to be contrasted with the exact expression

$$\Delta I_{exact} = \frac{2}{3}\delta^{\frac{3}{2}}\left[(2n+2)^{\frac{3}{2}} - (2n)^{\frac{3}{2}}\right]. \tag{1.7.2}$$

Expanding for large n we see that in the difference, $\Delta I_{approx} - \Delta I_{exact}$, many terms cancel, leaving

$$\delta^{\frac{3}{2}} n^{-\frac{7}{2}} \frac{\sqrt{2}}{1536}\left[-1 + \frac{7}{4n} - \frac{69}{32n^2} + \mathcal{O}\left(\frac{1}{n^3}\right)\right] \tag{1.7.3}$$

The cancellations are of course to be expected, since $1/n$ is of order δ and thus the right–hand side of Eq. (1.7.3) confirms that the error within the subinterval is of order δ^5.

Let us assume that we integrate $\sqrt{x}$ not from 0 to 1 but from some cutoff value x_c to 1. This means that we are adding the contributions from $2n_c = x_c/\delta$ to $2N$ and the errors in Eq. (1.7.3) add up to a global error

$$I_{approx} - I_{exact} = -\frac{\sqrt{2}}{3840}\delta^{\frac{3}{2}} n_c^{-\frac{5}{2}} + \mathcal{O}\left(n_c^{-\frac{7}{2}}\right) = -\frac{1}{480}\delta^4 x_c^{-\frac{5}{2}}\left(1 + \mathcal{O}\left(\frac{\delta}{x_c}\right)\right) \tag{1.7.4}$$

We see therefore that, so long as we integrate from a finite value of x on up, Simpson's formula produces a result accurate to order δ^4, as expected. But if we let the lower limit x_c approach small values of the order of δ itself (in particular if we try to use the formula for the whole range 0 to 1), the approximation deteriorates and the error becomes of order $\delta^{\frac{3}{2}}$. And the situation would be even worse with an integrable singularity, like $1/\sqrt{x}$, in which case the error would be of order $\delta^{\frac{1}{2}}$.

These arguments allow us also to compare the accuracies of the two procedures described above, namely of changing the variable of integration versus using Simpson's formula only from some x_c on, approximating the integrand by an expression that can be handled analytically for $x < x_c$. We have to think, of course, that while the integrand behaves like $\sqrt{x}$ for $x \approx 0$, its form is given by a more complex expression, that we can expand as $\sqrt{x}(a_0 + a_1 x + \cdots)$ up to an error of order $\sqrt{x}\, x^m$. Integrating up to x_c, the terms we neglected in the expansion of the integrand will produce an error of order $x_c^{(m+3/2)}$. Combining this with the result obtained in Eq. (1.7.4) we see that the total error will be of order

$$c\delta^4 x_c^{-\frac{5}{2}} + c' x_c^{m+\frac{3}{2}} \tag{1.7.5}$$

where c and c' are some constants.

This is minimized by $x_c \approx \delta^{\frac{4}{m+4}}$ with a resulting error of order $\delta^{\frac{4m+6}{m+4}}$.

If we change the variable of integration so as to eliminate the end point singularity, we expect an error of order δ^4, so we would conclude that changing variable of integration is more accurate that partitioning off the singularity and approximating the integrand in its neighborhood. Of course, one must also be guided by common sense in the implementation of numerical techniques, and for many applications the accuracy obtained with either method and a sufficiently small δ may be accurate enough. The important thing is to develop a sound understanding of how the numerical methods work so as to be able to adapt them to exceptional situations with critical sense, rather than using the various formulas as black boxes, which most of the times will work, but on occasions, and unbeknownst to the unsuspecting user, can produce catastrophic results.

Again, as we mentioned at the end of section 4, it is a useful and amusing exercise to verify running a numerical experiment that the actual behavior of the error follow the analytical expectations. We encourage you to calculate a simple integral that can be evaluated exactly and has an end point singularity by the two methods described above. You should use an expansion in the neighborhood of the singularity which produces an error of the same sign as the one introduced by Simpson's formula in the rest of the integration range and determine by a trial method the value of x_c that minimizes the error (the two errors must be of the same sign, of course, otherwise there will always be a value of x_c that will produce an artificially accurate result). You can then verify that the optimal cutoff value and the resulting error scale with delta as expected. The diskette of programs includes a code that "experiments" with the integrand $\sqrt{x + x^2}$).

1.10. Other elementary numerical methods

In this last section of an already too long chapter we shall discuss three further topics of basic use in computational physics. We start by exploring classical methods for finding zeroes of a function of one variable. This topic is also applicable to the problem of finding extrema of such functions by locating the zeroes of their derivatives.

The simplest method, is based on a search–and–partition algorithm. Imagine we are interested in finding zeroes of a function, $f(x)$ in a finite interval $[x_0, x_n]$. We subdivide this interval into n subintervals, and determine a subinterval, say $[x_i, x_{i+1}]$, where $f(x)$ changes sign, that is, where

$$f(x_i)\, f(x_{i+1}) \leq 0\,. \tag{1.10.1}$$

If x_m lies in the center of this subinterval, at least one of the following will hold:

$$f(x_i)\, f(x_m) \leq 0\,,$$

or

$$f(x_m)\, f(x_{i+1}) \leq 0\,.$$

In general only one of these conditions will be satisfied, but both could be true if f vanishes at one of the end points of the subinterval and changes sign in the subinterval defined by x_m and the other end point, or if f just happens to vanish at x_m. Of course, one could test to see whether f vanishes at one of the end points, or whether $f(x_i)\, f(x_{i+1}) < 0$ separately, but this is really not necessary, since the algorithm always produces at least one zero of f. Depending on which of the conditions above holds, we then refine the corresponding subinterval, and repeat the procedure.

Since the width of the subinterval where f changes sign decreases by a factor of 2^m in m iterations, with a sufficiently large m we can locate an arbitrarily small subinterval where f changes sign and, therefore, if f is continuous, a zero of f with essentially arbitrary precision –within the limits of numerical accuracy of the computer, of course.

Although this simple algorithm is very robust, its rate of convergence is impractical. However, once a sufficiently small neighborhood of a zero of f has been found and if f is continuous with continuous derivatives, as is usual in most applications, there are far more efficient algorithms to find the root. We will discuss two of the most popular of these algorithms.

The Newton–Raphson method uses information contained both in $f(x)$, as well as in its first derivative, $f'(x)$ by approximating $f(x)$ near x_n as by

$$f(x) = f(x_n) + (x - x_n)\, f'(x_n)\,. \tag{1.10.2}$$

The first derivative may be known analitically, or may be computed numerically. The iterative approach to the root is generated by taking as iterates of x the successive approximations to $f(x_{n+1}) = 0$ from Eq. (1.10.2), that is,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}\,. \tag{1.10.3}$$

If f is sufficiently linear near the root, this method converges very rapidly, but it is easy to run into trouble with this method if the starting point is not in the basin of attraction of the root. If the initial point is good, though, the convergence of the method is extremely fast. To see this, consider a case where the root is at the origin, and expand

$$f(x) = \alpha x + \frac{\beta}{2}x^2 + \mathcal{O}(x^3)$$

for small x. Applying the Newton–Raphson algorithm, Eq. (1.10.2), we find,

$$x_{n+1} = \frac{\beta x_n^2}{2(\alpha + \beta x_n)}$$

which is well approximated by

$$x_{n+1} = \frac{\beta x_n^2}{2\alpha}$$

for sufficiently small x_n. This recursion is easily solved by defining $y_n = \beta x_n/(2\alpha)$, giving, $y_{n+1} = y_n^2$, that is,

$$x_n = \frac{2\alpha}{\beta}\left(\frac{\beta x_0}{2\alpha}\right)^{2^n}$$

which is fast indeed!

An alternative to the Newton–Raphson method, of comparable popularity, is the so–called Secant method. The method uses previous values of the function at an iterate, $f(x_{n-1})$ to interpolate $f(x)$ near the root,

$$f(x) = \frac{x - x_n}{x_{n-1} - x_n} f(x_{n-1}) + \frac{x - x_{n-1}}{x_n - x_{n-1}} f(x_n) \ . \tag{1.10.4}$$

The next iterate, x_{n+1}, is found by talking $f(x) = 0$ in Eq. (1.10.4), which leads to,

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \ . \tag{1.10.5}$$

Another problem which often comes up is determining where a given x falls realative to a discretization of the interval $[x_0, x_n]$, $x_0, x_1, \ldots, x_n$. We actually had to solve this problem when we constructed the the histogram defining the density of states.

If the x_i are uniformly spaced, the solution is trivially found. Let $x_m = x_0 + m\delta$. Defining $y = (x - x_0)/\delta$, the original sequence, $x_0, x_1, \ldots, x_n$ goes over into $y_0 = 0, y_1 = 1, \ldots, y_n = n$, and

$$k = \text{int}(y) = \text{int}\left(\frac{x - x0}{\delta}\right) \tag{1.10.6}$$

labels the interval, $[x_i, x_{i+1}]$ where x falls.

It is very easy to generalize the method to the case where the x_i are not uniformly spaced, but are spaced according to some simple functional relation. As an example, consider the x_i spaced in geometric progression,

$$x_0 \quad x_1 = rx_0 \quad \ldots \quad x_n = r^n x_0 \quad (r > 0) \,,$$

then the variables $y_0, \ldots, y_n$ defined by $y_i = \ln x_i$ are uniformly spaced, and we can proceed as in the simplest case.

For an arbitrary discretization of the search interval, finding the subinterval where a given x falls requires multiple comparisons. If n is not small, though, the intervals should not be scanned sequentially, which would require $\mathcal{O}(n)$ tests, but rather by a sequence of bisections, which only require $\mathcal{O}(\ln_2 n)$ comparisons.

To see this, look at the following algorithm. Assuming that we have tested that z lies in the interval $[x_0, x_n]$, proceed as follows

```
      m = int(n/2)
      x ≥ x(m) ?
yes:  consider the sequence x_m x_{m+1} ... x_n
 no:  consider the sequence x_0 x_1 ... x_m
```

and repeat this procedure until the whole sequence collapses to a single interval.

Here's a simple Fortran implementation of the above algorithm: Assume we've dimensioned the array of end points, `x(0:n)`,

```
m1 = 0
m2 = n
DO WHILE ( m2 .GT. m1+1 )
   m = (m1+m2)/2
   IF( z .GT. x(m) ) THEN
      m1 = m
   ELSE
      m2 = m
   ENDIF
END DO
```

As a variant of the above algorithm, if several intervals corresponding to a sequence of variables, $x^{(1)} < x^{(2)} < x^{(3)} < \ldots$ must be located, after $x^{(1)}$ has been placed between x_i and x_{i+1}, we may test whether $x^{(2)} < x_{i+1}$ first, then test for $x^{(2)} < x_{i+2}$, $x^{(2)} < x_{i+4}$, $x^{(2)} < x_{i+8}$, and so on, in progressively larger steps until some bounding x_{i+2^m} is found. The last interval encountered before the bound is then searched using the simple search algorithm illustrated above. This variant, as opposed to considering the entire interval again for $x^{(2)}$, will be more efficient if there is a reasonable expectation that the sequence of $x^{(k)}$ is not too sparse

The last topic we will examine in this chapter is one we've also encountered briefly in our calculation of the density of states, namely, the problem of finding the unique

polynomial, $P_{n-1}(x)$, of degree $n-1$ which takes the values, $f(x_1), f(x_2), \ldots, f(x_n)$ at the points $x_1, x_2, \ldots, x_n$. The classic formula for this polynomial is due to Lagrange, and bears his name. It is given by,

$$\begin{aligned} P(x) = & \frac{(x-x_2)(x-x_3)\cdots(x-x_n)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_n)} f(x_1) \\ & + \frac{(x-x_1)(x-x_3)\cdots(x-x_n)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_n)} f(x_2) \\ & + \ldots + \\ & + \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(x_n-x_1)(x_n-x_3)\cdots(x_n-x_{n-1})} f(x_n) \end{aligned} \tag{1.10.7}$$

This expression, requiring $O(n^2)$ operations can be used to calculate $P(x)$ given x. A more elegant, and slightly more economical algorithm accomplishes the same task by iteration.

Imagine that two polynomials, $Q(x)$, and $R(x)$, of degree $n-2$ interpolate $f(x)$ through the points, $x_1, \ldots, x_{n-1}$ with $Q(x)$, and through the points $x_2, \ldots, x_n$ with $P(x)$. Then

$$P(x) = \frac{(x-x_n)}{(x_1-x_n)} Q(x) + \frac{(x-x_1)}{(x_n-x_1)} R(x) \tag{1.10.8}$$

The proof of this statement is trivial. To see this, evaluate $P(x)$ at x_1 and x_n; there it coincides with the corresponding values taken by Q and R respectively, but, by assumption, Q and R interpolate f at these points. At the rest of the x_i,

$$\begin{aligned} P(x_i) &= \frac{(x_i-x_n)}{(x_1-x_n)} Q(x_i) + \frac{(x_i-x_1)}{(x_n-x_1)} R(x_i) \\ &= \frac{(x_i-x_n)}{(x_1-x_n)} f(x_i) + \frac{(x_i-x_1)}{(x_n-x_1)} f(x_i) \\ &= f(x_i) \ , \end{aligned} \tag{1.10.9}$$

where the second equality follows from the fact that Q and R both interpolate $f(x)$.

Since P is of order $n-1$, it coincides with the (unique) interpolating polynomial. This gives origin to the following iterative procedure.

(i) Build a table of 0–degree polynomials taking the correct values at $x_1 \ldots x_n$, that is, $f(x_1) \ldots f(x_n)$.

(ii) Call these constant polynomials $P_1^{(0)} P_n^{(0)}$.

(iii) From neighboring pairs of these polynomials, construct $n-1$ polynomials of degree 1 interpolating f at the pairs of neighboring points. That is, construct

$$P_{i,j}^{(1)} = \frac{(x-x_j)}{(x_i-x_j)} P_1^{(0)} + \frac{(x-x_i)}{(x_j-x_i)} P_2^{(0)} \tag{1.10.10}$$

for every pair $\{i,j\}$ up to $P_{n-1,n}^{(1)}$

The procedure is continued by combining pairs of the $n-1$ polynomials of degree 1 into $n-2$ polynomials of degree 2, $P^{(2)}_{1,2,3}\, P^{(2)}_{2,3,4}\, P^{(2)}_{n-2,n-1,n}$ by linear interpolation.

The recursion obviously ends after $n-1$ steps, where the resulting polynomial, $P^{(n-1)}_{1,2,\ldots,n}$, coincides with $P(x)$.

The number of operations involved in this construction is still $\mathcal{O}(n^2)$, but is smaller than that required for the naive evaluation. Furthermore, as an added advantage, one can keep track of the corrections between the interpolations of lower and higher degree.

The recursive procedure also allows one to calculate efficiently the coefficients of the interpolating polynomial, in case one wants these quantities instead (or as well as) the actual value $P(x)$. Having these coefficients, the calculation of $P(x)$ for a different value of x requires only $\mathcal{O}(n)$ operations, as we will see shortly. A word of caution which we will repeat often (not more often than necessary, though) is in order here: Consider the problem of interpolating a set of 20 numbers whose values are of order one for x taking values between 1 and 20. The interpolating polynomial will either require cancellations between very large numbers, or have very small coefficients, or both. In all such cases one has to be very careful and not trust results blindly, since they are likely to be fraught with round-off errors.

Our final comment here concerns the efficient evaluation of a polynomial,

$$c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n \,.$$

The straightforward code

```
P = C(0)
DO I=1,n
   P = P + C(I)*x**I
END DO
```

is very inefficient, involving $\mathcal{O}(n^2)$ operations . The correct way of evaluating a polynomial is by recursion,

```
P = C(n)
DO I=1,n
   P = P*x + C(I-1)
END DO
```

which requires only $\mathcal{O}(n)$ operations.