Chapter 3

Approximation

Chapter 3

3.1 Interpolation

So far, we have discussed numerical methods to solve simple equations. In this and a few following sections we will consider the second fundamental concept in this course which is the concept of a function. Functions $f(x)$ have been used previously to describe nonlinear equations. These functions where then used to create a second class of functions $F(x^0, x^1, \ldots, x^n)$ which defined the numerical (iterative) procedure to compute a solution to the equation $f(x) = 0$.

In the following sections a function $u(x)$ is the primary object. Ultimately, we will discuss differential equations which have a function $u(x)$ as their solution. We will consider first the representation of functions in computers, their approximation by polynomial functions, operations defined on the functions like integration and differentiation before we enter in the methods for the solution of differential equations.

3.1.1 Functions in scientific computing

Functions may occur as results of (lengthy) numerical computations. Often, the such a final result is a relatively simple function while more complicated functions are evaluated throughout the computation. The final results are used in decision and design processes which involve prediction, diagnosis, and optimisation.

The type of functions used depends on their applications. Some functions may depend on time. This includes the average temperature in some city or the blood pressure of a patient. Such functions depend on one variable. A large class of functions are spatial including the hyperspectrum of some pasture or forest in agriculture or ecology. Another example is the velocity of flows in oceans and floods. Such functions depend on one to three spatial variables. A third class of functions depend on parameters defining a certain design or process. For example, porous flow depends on the porosity of the media. These functions may depend on a large number (some times in the hundreds) of material and design parameters. A generalisation of spatial functions are functions which define on some state including density of particles over space and velocity or probability densities depending on the values of (multidimensional random) variables.

There are now a variety of ways on how to determine a function.

1. In the simplest case, the function is provided by a specific function evaluation formula like $u(x) = \exp(-2x)$.
2. In many cases, the function is determined from a model defined by (differential) equations.
3. Functions may also be obtained by fitting data to some class of functions.

In many applications one has both a model and data to determine a function. This is the case of data assimilation.

Larger project in scientific computing involve the study of functions $u(x, \lambda)$ in terms of the their parameters λ. These functions are computed given some λ from models and data. One might wish to find parameters λ such that some property like the energy or risk related to a specific function is minimised. As the determination of $u(x, \lambda)$ for every λ typically involves large scale computations which may take many hours one aims to only compute the functions for a finite number of parameter values $\lambda = \lambda_1, \ldots, \lambda_m$. (Note that the λ_k can also be vectors.) When trying to find the best λ or investigating the effect of the parameter λ on the solution one uses interpolation to get approximates for $u(x, \lambda)$ and other values of λ.

Fundamentally, functions are represented by evaluation procedures which take as input some value and return some other value. In Python, one has two basic ways to represent functions. Simple functions are defined by simple expressions as in

```
u = lambda x : x*x
```

or defining a subroutine as in

```
def u(x):
    y = x*x
    return y
```

A large collection of pre coded functions are available in Python modules like math and scipy using for example

```
from math import exp
```

or

```
from scipy import exp
```

Computers can read numbers from tables, can evaluate arithmetic expressions like $x + y$ and can determine results based on decisions:

```
if x > 14:
  y = 5
else:
  y = 2
```

It follows that any computed result is a piecewise rational function of the input values of the independent variables and the parameters. A piecewise rational function of x is actually a collection of rational functions. The code then selects the appropriate one to evaluate based on the value of the inputs. Consequently, even simple functions like $\exp(x)$ cannot be exactly represented in a computer. Instead one replaces this function with an approximation.

In this section we will consider the simplest case of polynomial functions $u(x)$ of one variable x for $x \in [a, b]$ and following that we consider functions which may evaluate to a different polynomial on each subinterval $[\xi_n, \xi_{n+1}]$ of $[a, b]$.

3.1.2 Polynomials, their representation and evaluation

A fundamental tool of scientific computing is the class of polynomial functions. Recall that a polynomial of degree n is defined by $n + 1$ coefficients $a_0, \ldots, a_n$ and evaluates to

$$p_n(x) = a_0 + a_1 x + \cdots + a_n x^n.$$

A simple Python code for this evaluation is

```
def pn(x,a):
   y = a[0] + a[1]*x
   xk = x
   for k in range(2,len(a)):
     xk = x*xk
     y  = y + a[k]*xk
   return y
```

One sees that one requires $n + 1$ real numbers $a_0, \ldots, a_n$ to represent the function p_n. The evaluation at one point requires n additions and $2n - 1$ multiplications. Furthermore, we need to access the $n + 1$ coefficients a_k from memory. In the loop, the values computed for some k depend on the values computed in the previous step $k - 1$. Depending on how one determines the complexity (essentially the cost or time) of these computations they are $2n - 1$ times the time taken for one multiplication plus n times the time for one addition. In the early days the time for the additions could be neglected as they were much cheaper than the multiplications. Later they were the same cost but could be done in parallel with multiplications (as addition and multiplication units were independent) and could be neglected again. Today, the main cost of polynomial evaluation is for the cost of accessing the coefficients a_k, the arithmetic is basically "for free".

While the costs of scientific computations has changed substantially over the years it is still interesting to investigate the scope for saving. Often code transformations which result in big savings are done automatically by compilers (for languages like C or Fortran). A major idea used is to block loops and data access into blocks which can be done in parallel. From the arithmetic point of view, one can use the humble distributive law

$$(a + b)c = ac + bc$$

to achieve considerable computational savings. In particular, a repeated application of the distributive law (together with the associative and commutative laws) provide

$$p_n(x) = a_0 + x(a_1 + x(a_2 + x(\cdots))).$$

One sees directly, that while the number of additions is still n one now only has to multiply n times with x. One thus saves half the number of multiplications. A possible Python code of this is

```
def pn(x,a):
    n = len(a)-1
    y = a[n]
    for k in range(n-1,-1,-1):
      y  = a[k] + x*y
    return y
```

Note that in addition to reducing the number of multiplications one also does not require the intermediate values xk and in particular does not not need to store and retrieve them.

The class of polynomial functions of degree at most n is an $n+1$ dimensional vector space. One can thus add polynomial functions and multiply them with numbers and gets polynomials again. The product of two polynomials of degree n results in a polynomial of degree $2n$. If the coefficients drop sufficiently fast going from a_0 to a_n one can approximate this product by a polynomial of degree n. Furthermore, the derivative a polynomial of degree n is a polynomial of lesser degree $n-1$. Using these ingredients one may derive algorithms so solve equations involving polynomials. However, only a small proportion of all functions occurring in applications are polynomials. We will now show that a substantially larger class can be approximated reasonably well with polynomial functions.

3.1.3 Polynomial approximation and the Taylor polynomial

Weierstrass' theorem (from calculus) states that every continuous function over a finite interval $[a, b]$ and be approximated arbitrarily well by a polynomial $p_n(x)$ of sufficiently high degree. We do not know in advance, however, how high the degree of the approximating polynomial has to be to achieve a certain accuracy. In practice, it turns out that polynomial approximation is most suited to very smooth functions. (In which case the approximation does work very well.)

Again from calculus we know that for a function which is $n+1$ times continuously differentiable one has the Taylor remainder theorem

$$u(x) = u(a) + u'(a)(x-a) + \frac{u''(a)}{2}(x-a)^2 + \cdots + \frac{u^{(n)}(a)}{n!}(x-a)^n + \frac{u^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}$$

for some $\xi \in [a, x]$. The Taylor polynomial for $x = a$ is then given by

$$a_k = \frac{u^{(k)}(a)}{k!}, \quad k = 0, \ldots, n.$$

For a bounded $(n+1)$st derivative the error is (up to a constant factor) bounded by $(b-a)^{n+1}/(n+1)!$. The advantage of the Talor approximation is that one only needs to evaluate u in one point $x = a$ but the disadvantage is that one requires n derivatives at that point. For very smooth functions u, however, Taylor approximations are successfully utilised in a variety of algorithms and also for error bounds.

3.1.4 Polynomial Interpolation

Polynomial Collocation

The simplest and oldest case is polynomial collocation. The basic result is this:

Theorem 1. *Given any $n+1$ distinct numbers $x_0, \ldots, x_n$, and any set of numbers $y_0, \ldots, y_n$, there is exactly one polynomial $p_n(x)$ of degree n or less that satisfies the collocation (or interpolation) conditions*

$$p_n(x_i) = y_i, \qquad 0 \le i \le n.$$

This theorem will be proven constructively by computing the interpolation polynomial in three different ways.

Polynomial Interpolation

Suppose we are given a polynomial $p_n(x) = a_0 + a_1 x + \cdots + a_n x^n$, the condition that $p_n(x_i) = y_i$ $(0 \le i \le n)$ leads to the following matrix equation $X\,a = y$:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

The matrix X is a *Vandermonde matrix*. If no two x_i are the same then it can be demonstrated that X is invertible. This is equivalent with the statement that a polynomial of degree n which is zero at $n+1$ different points is identically zero.

Example Polynomial Interpolation

Example 1 (Second Order Interpolation)**.** *Suppose we need to find the polynomial $p_2(x) = a_0 + a_1 x + a_2 x^2$ that passes through the following three points;*

i	*0*	*1*	*2*
x_i	*0*	*0.5*	*2*
y_i	*0.2*	*0.6*	*-1.0*

This leads to the system of equations

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0.5 & 0.25 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.6 \\ -1 \end{bmatrix}.$$

Solving the above system of equations gives $a_0 = 1/5$, $a_1 = 19/15$ and $a_2 = -14/15$, so $p_2(x) = 1/5 + 19/15x + -14/15x^2$. *Important:* always check your answer by ensuring $p_2(x_i) = y_i$ for all i.

Cardinal Functions

An alternative to the solution of $Xa = y$ for the coefficients a_i of the monomials x^j uses *cardinal functions* $l_j(x)$ which satisfy

$$l_j(x_i) = \begin{cases} 1, & i = j \\ 0, & \text{otherwise.} \end{cases}$$

Each l_j has the n roots $x_i, i \ne j$ and being a polynomial of degree at most n these are all the roots. Thus the l_j must have the form

$$l_j(x) = c_j(x - x_0)(x - x_1)\cdots(x - x_{j-1})(x - x_{j+1})\cdots(x - x_n).$$

Next, the constants c_j can be determined by evaluating l_j at x_j where $l_j(x_j) = 1$. One obtains

$$c_j = \frac{1}{\prod_{i=0, i\ne j}^{n}(x_j - x_i)}$$

which gives

$$\begin{aligned} l_j(x) &= \frac{(x - x_0)\cdots(x - x_{j-1})(x - x_{j+1})\cdots(x - x_n)}{(x_j - x_0)\cdots(x_j - x_{j-1})(x_j - x_{j+1})\cdots(x_j - x_n)} \\ &= \prod_{i=0, i\ne j}^{n} \frac{x - x_i}{x_j - x_i}. \end{aligned}$$

Note that we have by construction that $l_j(x_i) = 1$ if $i = j$, and is 0 otherwise. Also each of these polynomials is uniquely determined by the collocation conditions.

Example Polynomial Interpolation

Example 2 (Second Order Interpolation)**.** *The three cardinal functions corresponding to the data points*

i	0	1	2
x_i	0	0.5	2
y_i	0.2	0.6	-1.0

are

$$l_0(x) = \frac{(x-0.5)(x-2)}{(0-0.5)(0-2)} = (x-0.5)(x-2),$$

$$l_1(x) = \frac{(x-0)(x-2)}{(0.5-0)(0.5-2)} = -\frac{4}{3}x(x-2),$$

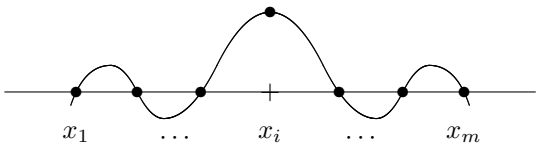
$$l_2(x) = \frac{(x-0)(x-0.5)}{(2-0)(2-0.5)} = \frac{1}{3}x(x-0.5).$$

Lagrange Form

Once the cardinal (or Lagrangian) functions $l_j(x)$ have been determined, the interpolant $p_n(x)$ is simply

$$p_n(x) = y_0 l_0(x) + \cdots + y_n l_n(x) = \sum_{k=0}^{n} y_k l_k(x).$$

This form of the polynomial is known as the *Lagrangian Form* of the interpolation polynomial.



Example of Lagrangian function $l_i(x)$

Example Polynomial Interpolation

Example 3 (quadratic interpolation)**.** *The polynomial that passes through the data points given in the previous example would be*

$$p_2(x) = 0.2 * l_0(x) + 0.6 * l_1(x) - l_2(x).$$

To verify that this constructed polynomial is indeed a solution, first notice that as a sum of constant multiples of polynomials of degree $\leq n$, it is certainly a polynomial with degree of at most n. To check the collocation condition evaluate at each x_j:

$$\begin{aligned} p_n(x_j) &= y_0 l_0(x_j) + \cdots + y_j l_j(x_j) + \cdots + y_n l_n(x_j) \\ &= y_0.0 + \cdots + y_j.1 + \cdots + y_n.0 \\ &= y_j. \end{aligned}$$

The uniqueness of this solution is not obvious, but is easily confirmed. For suppose $p(x)$ and $q(x)$ are both polynomial solutions of this collocation problem, then $r(x) = p(x) - q(x)$ is a polynomial of degree at most n, but with the $n+1$ roots $x_0 \ldots x_n$. Thus r must be identically zero, and thus p and q are the same function.

Newtons Method

Both interpolation methods discussed so far establish $p_n(x)$ for one fixed n. In contrast, Newton's method determines all k-th degree interpolation polynomials $p_k(x)$ which interpolate $(x_0, y_0), \ldots, (x_k, y_k)$.

It follows that the constant polynomial $p_0(x) = y_0$. Furthermore, for any known $p_{k-1}(x)$ one gets $p_k(x)$ from the observation that the difference $p_k(x) - p_{k-1}(x)$ is a k-th degree polynomial which is zero for $x = x_0, \ldots, x_{k-1}$. Consequently

$$p_k(x) = p_{k-1}(x) + c_k(x - x_0)(x - x_1)\cdots(x - x_{k-1})$$

and the first k interpolation conditions $p_i(x_i) = y_i$ hold. From $y_k = p_k(x_k) = p_{k-1}(x_k) + c_k(x_k - x_0)\cdots(x_k - x_{k-1})$ one obtains $c_k = (y_k - p_{k-1}(x_k))/\prod_{j=0}^{k-1}(x_k - x_j)$. With this one has

$$\begin{aligned} p_0(x) &= c_0, \\ p_1(x) &= c_0 + c_1(x - x_0), \\ p_2(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1), \\ &\ldots \end{aligned}$$

Example Polynomial Interpolation

Example 4 (Newton's interpolation formula)**.** *Consider the same data points as previously:*

i	0	1	2
x_i	0	0.5	2
y_i	0.2	0.6	-1.0

to ge the interpolation functions in Newton's form as

$$\begin{aligned} p_0(x) &= c_0 = 0.2, \\ p_1(x) &= c_0 + c_1(x - 0) = 0.2 + 0.8x, \\ p_2(x) &= c_0 + c_1(x - 0) + c_2(x - 0)(x - 0.5) \\ &= 0.2 + 0.8x - \frac{14}{15}x(x - 0.5). \end{aligned}$$

Examples of Polynomial Collocation

Example - Newton's Form

Consider the polynomial $p_3(x) = 4x^3 + 35x^2 - 84x - 954$. Show that the four points with coordinates $(5, 1)$, $(-7, -23)$, $(-6, -54)$ and $(0, -954)$ are on the graph of p_3.

Example 5 (Newtons' Form)**.** *An application of Newton's interpolation method gives then*

$$1 + 2(x - 5) + 3(x - 5)(x + 7) + 4(x - 5)(x + 7)(x + 6).$$

Example - Lagrange's Form

Consider again the polynomial $p_3(x) = 4x^3 + 35x^2 - 84x - 954$ and the four points $(5, 1)$, $(-7, -23)$, $(-6, -54)$ and $(0, -954)$.

Example 6 (Lagrange Form)**.** *The cardinal functions for this example are*

$$l_0(x) = \frac{(x + 7)(x + 6)x}{(5 + 7)(5 + 6)5} \qquad l_1(x) = \frac{(x - 5)(x + 6)x}{(-7 - 5)(-7 + 6)(-7)}$$

$$l_2(x) = \frac{(x - 5)(x + 7)x}{(-6 - 5)(-6 + 7)(-6)} \qquad l_3(x) = \frac{(x - 5)(x + 7)(x + 6)}{(0 - 5)(0 + 7)(0 + 6)}$$

and Lagrange's method gives

$$p_3(x) = l_0(x) - 23l_1(x) - 54l_2(x) - 954l_3(x).$$

The two function forms (Lagrange and Newton's) may appear different, but are in fact representations of the same function. They are called the *interpolation polynomials*. In terms of computations, Newton's form appears the more simple in that a polynomial of the next order builds nicely on the previous polynomial. It lends itself to nested multiplication:

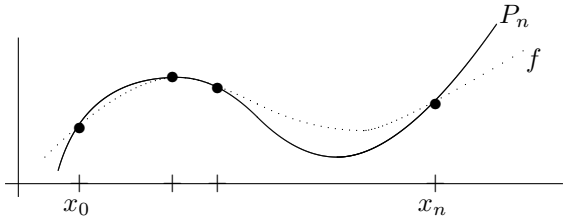
$$p_k(x) = (...(((c_k)d_{k-1} + c_{k-1})d_{k-2} + c_{k-2})d_{k-3} + ... + c_1)d_0 + c_0$$

where the c_i's are known coefficients and $d_i = (x - x_i)$.

3.1.5 Error Formulae for Polynomial Collocation

Basic Error Formula

The polynomial interpolation polynomial p_n are used as approximation for f in a variety of numerical procedures.



The error can be obtained from Rolle's theorem as

$$e_n(x) := f(x) - p_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{i=0}^{n} (x - x_i),$$

where ξ_x is between the $x_0, \ldots, x_n$ and depends on x.

When $f(x)$ is computed by a complicated numerical procedure, one does not know much about f let alone any higher derivatives. Nonetheless, the error bound does give some information about the dependence of the error on the data points x_i in the case where f is $n+1$ times differentiable.

Divided differences and proof of Basic Error Formula

Let $f[x_0, \ldots, x_k] = c_k$ be the coefficients for the Newton interpolation formula so that

$$p_n(x) = \sum_{k=0}^{n} f[x_0, \ldots, x_k] w_k(x)$$

is the interpolation polynomial with $p_n(x_j) = f(x_j)$ for $j = 0, \ldots, k$ and where

$$w_k(x) = \prod_{j=0}^{k-1} (x - x_j), \quad k = 1, \ldots$$

and $w_0(x) = 1$. The coefficients in the Newton interpolation formula are often called divided differences. Now let $q_{n+1}(t)$ be the polynomial of degree $n + 1$ which interpolates f in the points $t = x_0, \ldots, x_n$ and also in $t = x$. The Newton interpolation formula gives then

$$q_{n+1}(t) = p_n(t) + f[x_0, \ldots, x_n, x] w_{n+1}(t).$$

As $q_{n+1}(x) = f(x)$ one has then

$$f(x) = p_n(x) + f[x_0, \ldots, x_n, x]\, w_{n+1}(x)$$

or in terms of the error

$$e_n(x) = f(x) - p_n(x) = f[x_0, \ldots, x_n, x]\, w_{n+1}(x).$$

One could now show that the divided difference is a multiple of the $n + 1$st derivative to get the error formula. Here, we will get this result directly from Rolle's theorem (and in this way also prove a formula for the divided difference).

Theorem 2. *Let $f \in C^{n+1}$ and $p_n(x)$ be the n-th degree polynomial which inerpolates f in the points $x = x_0, \ldots, x_n$. Then there exists a ξ_x in the span of $x_0, \ldots, x_n$ and x such that*

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} w_n(x).$$

Proof. Let $e_n(x) = f(x) - p_n(x)$ and define

$$g(t) = e_n(t) - \frac{e_n(x)}{w_n(x)} w_n(t).$$

This function is well defined if $x \neq x_k$ for all $k = 0, \ldots, n$. It does have $n+2$ zeros $t = x_0, \ldots, x_n$ and $t = x$. Thus, by Rolle's theorem, there exists a ξ_x such that

$$g^{(n+1)}(\xi_x) = 0.$$

Now, as $p_n^{(n+1)} = 0$ and $w_n^{(n+1)} = (n+1)!$ one gets

$$g^{(n+1)}(t) = f^{(n+1)}(t) - \frac{e_n(x)}{w_n(x)}(n+1)!$$

and the error formula follows by choosing $t = \xi_x$. □

The proof of this theorem is based on Rolle's theorem from calculus which states that if a continuous function $f(x)$ defined on an interval $[a, b]$ is differentiable in the interval and satisfies $f(a) = f(b)$ then there exists a point x such that $a < x < b$ and $f'(x) = 0$. In this form, Rolle's theorem is a special case of the mean value theorem. Actually, the mean value theorem is typically proved using Rolle's theorem and Rolle's theorem itself is proved considering extremal values.

Using induction one can then prove the more general result of Rolle's theorem used in the proof above.

When simply fitting to a set of points, there is no well-defined error: usually we are simply concerned with the plausibility of the resulting function. However when the data are considered to be given by the values of some possibly unknown function as $y_i = f(x_i)$ then there are formulae describing the possible errors. These also lead to some useful observation on the plausibility of values given by a collocation polynomial in various situations, and some ideas about how to choose the points x_i, if that is possible.

Example

If $f(x) = \sin x$, the derivative term is a sine or cosine and $|f^{(n+1)}| \leq 1$. Consequently, in this case, we are mainly interested in getting a bound on the size of the polynomial factor

$$w_{n+1}(x) := \prod_{i=0}^{n} (x - x_i).$$

On the interval $[0, 1]$ this product is bounded above by 1, and thus with 10 nodes and x in the interval $[0, 1]$

$$|\sin x - p(x)| \leq \frac{1}{10!} < 2.8 \times 10^{-7}.$$

More often, all one can do is assume some vague bound $|f^{(n+1)}| \leq M$ (which is true for some M if this derivative is continuous on the interval containing the data points) and try to bound the remaining terms, meaning in particular $|w_{n+1}(x)|$.

Indeed it is surprisingly common for the error over the whole interval spanned by the collocation points to be roughly proportional to this polynomial, so choosing those points so as to keep $w_{n+1}(x)$ small for the x values of interest is a useful strategy, which will be considered next.

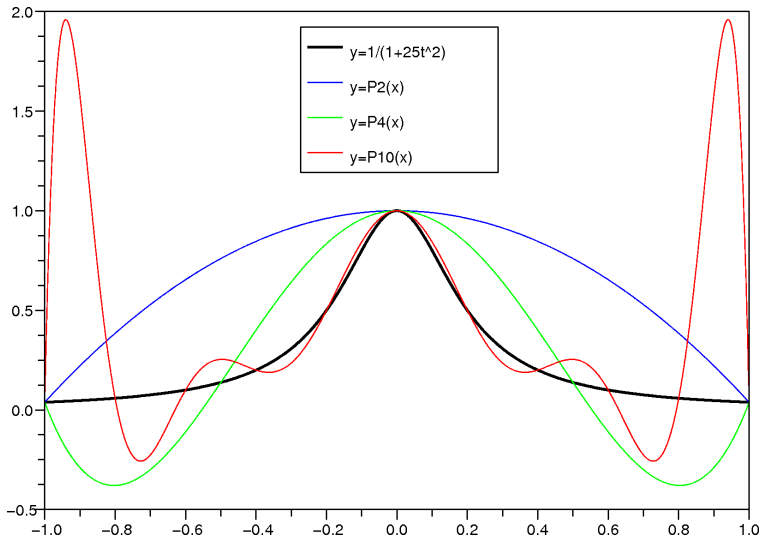


Figure 3.1: Several polynomial fits to Runge's function.

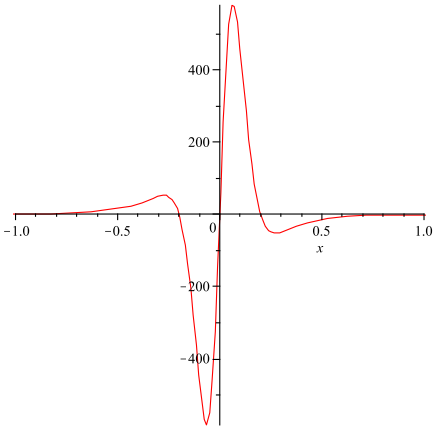


Figure 3.2: Third derivate of Runge's function

Example: Runge's function $f(t) = \frac{1}{1+25\,t^2}$

Figure 3.1 shows the results of fitting polynomials of different degree to Runge's function. A second degree (`P2`), fourth degree (`P4`) and tenth degree (`P10`) polynomial have been fitted to the function. Observe that the higher order polynomial fits tend to be highly oscillatory and vary greatly over a small interval.

The third, fifth and eleventh derivative of Runge's function was found using Maple. A plot of the functions are given in Figures 3.2, 3.3 and 3.4. Observe that the maximum values of $|f^{(n+1)}|$ increases as n is increased, which suggests that e_n may be large for high order polynomials, as we see in Figure 3.1

3.1.6 Chebyshev Interpolation

Example: The function $w(x) = (x - x_1)(x - x_2)\cdots(x - x_m)$

Lets consider another example of a high order polynomial fit. Figure 3.5 shows the results of fitting a polynomial of degree 11 to the function $w(t) = (t - t_1)(t - t_2)\cdots(t - t_m)$. Observe that the polynomial passes through equally spaced points.

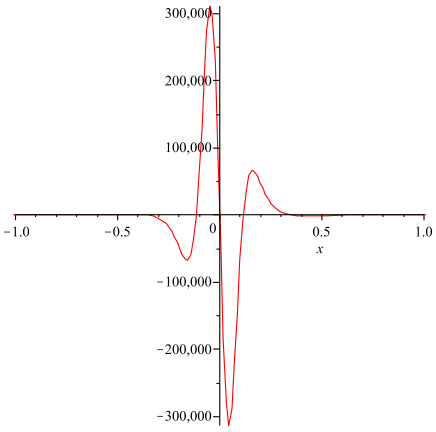


Figure 3.3: Fifth derivate of Runge's function

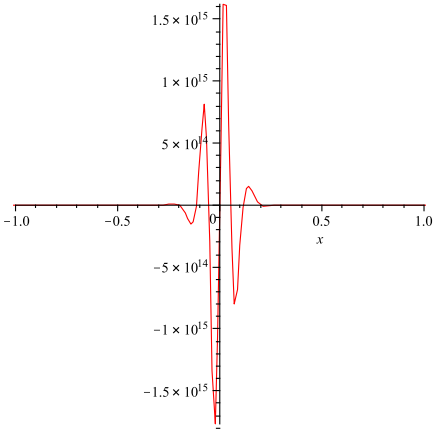


Figure 3.4: Eleventh derivate of Runge's function

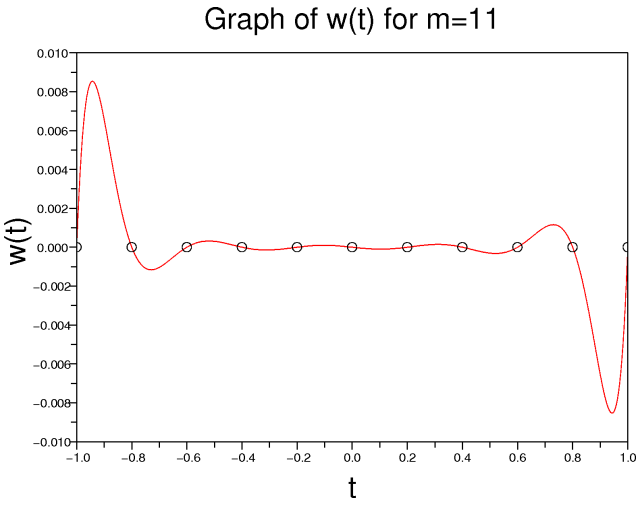


Figure 3.5: Polynomial $w(x)$ for equidistant x_i.

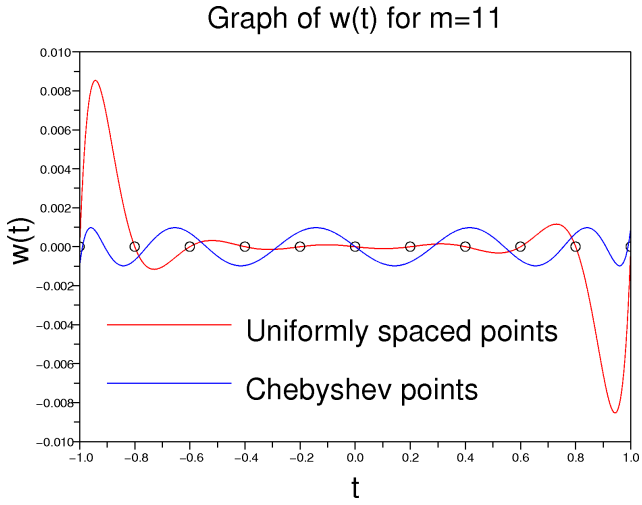


Figure 3.6: Polynomial $w(x)$ for Chebyshev points.

By being more careful about the choice of interpolation points it is possible to find a less oscillatory polynomial to approximate the function. See Figure 3.6.

Polynomial interpolants using Chebyshev points

In some situations, one can choose the nodes x_i to use in polynomial collocations and a natural objective is to minimise the worst case error over some interval $[a, b]$ on which the approximation is to be used. As discussed previously, the best one can do in most cases is to minimise the maximum absolute value of the polynomial $w_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)$ arising in the error formula.

The usual idea of using equally spaced points is not optimal as w_{n+1} reaches considerably larger values between the outermost pairs of nodes. Better intuition suggests that moving the nodes closer in these regions of large error will reduce the maximum error there, while not increasing it too much elsewhere. Compare Figures 3.1 and 3.7. Further it would seem that this strategy is possible so long as the maximum amplitude in some of the intervals between the nodes is larger than others: the endpoints a and b need not be nodes so there are $n + 2$ such intervals.

Chebyshev Points

The *Chebyshev points* are given by the formula

$$\frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{2i+1}{2n+2}\pi\right), \qquad 0 \le i \le n. \tag{3.1}$$

These points correspond to the roots of the *Chebyshev polynomials*. These polynomials are important because they have the minimum (smallest) largest value of any polynomials of degree $n + 1$ on the interval $[-1, 1]$.

To understand this result, consider the case where the interval of interest is $[-1, 1]$, so that these special nodes are $\cos\left(\frac{2i+1}{2n+2}\pi\right)$. The general case then follows by using the change of variables $x = (a + b)/2 + t(b - a)/2$. The reason that this works is that these are the roots of the function

$$T_{n+1}(x) = \cos((n + 1)\cos^{-1} x)$$

which turns out to be a polynomial of degree $n + 1$ that takes its maximum absolute value of 1 at the $n + 2$ points $\cos\left(\frac{i}{n+1}\pi\right), 0 \le i \le n + 1$.

There are a number of claims here: most are simple consequences of the definition and what is known about the roots and extreme values of cosine. The $T_n(x)$ is a polynomial of degree n,

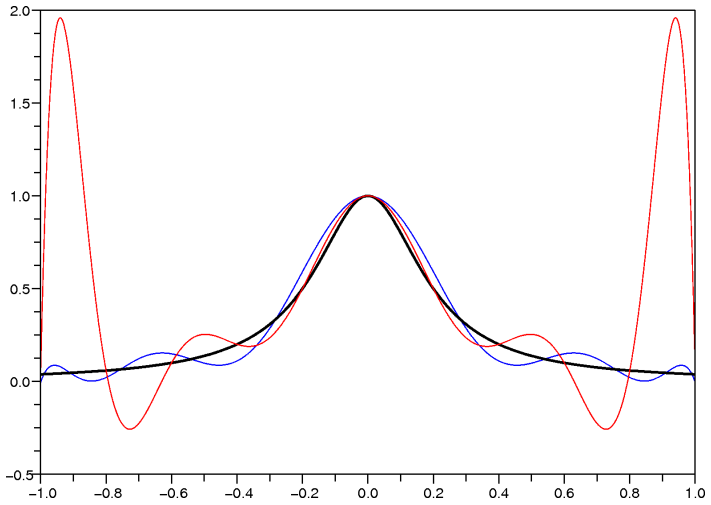


Figure 3.7: Polynomial interpolants of Runge's Function using equally spaced x_i (red) and Chebyshev points x_i (blue) respectively.

known as a *Chebyshev polynomial*. (The notation comes from an old translation, Tchebycheff, of this Russian name.)

By induction, the first few cases are easy to check: $T_0(x) = 1$, $T_1(x) = x$ and $T_2(x) = \cos 2\theta = 2\cos^2\theta - 1 = 2x^2 - 1$. In general, let $\theta = \cos^{-1} x$ so that $\cos\theta = x$. Then trigonometric identities give

$$\begin{aligned} T_{n+1}(x) &= \cos(n+1)\theta \\ &= \cos n\theta\cos\theta - \sin n\theta\sin\theta \\ &= T_n(x)x - \sin n\theta\sin\theta \end{aligned}$$

and similarly

$$\begin{aligned} T_{n-1}(x) &= \cos(n-1)\theta \\ &= \cos n\theta\cos\theta + \sin n\theta\sin\theta \\ &= T_n(x)x + \sin n\theta\sin\theta \end{aligned}$$

Thus $T_{n+1}(x) + T_{n-1}(x) = 2xT_n(x)$ or

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$

Since T_0 and T_1 are known to be polynomials, the same follows for each successive n from this formula. The induction also shows that

$$T_n(x) = 2^{n-1}x^n + \quad \text{(terms involving lower powers of } x\text{)}$$

so in particular the degree is n.

With this information, the error formula can be written in a special form. Firstly w_{n+1} is then a polynomial of degree $n+1$ with the same roots as T_{n+1}, so is a multiple of the latter function. Secondly, the leading coefficient of w_{n+1} is 1, compared to 2^{n+1} for the Chebyshev polynomial, so $w_{n+1} = T_{n+1}/2^n$.

Finally, the maximum of w_{n+1} is seen to be $1/2^n$ and we have the following result.

Theorem 3. *When a polynomial approximation $p(x)$ to a function $f(x)$ on the interval $[-1,1]$ is constructed by collocation at the roots of T_{n+1}, the error is bounded by*

$$|f(x) - p(x)| \le \frac{1}{2^n(n+1)!} \max_{-1\le t\le 1} |f^{(n+1)}(t)|.$$

When the interval is $[a, b]$ and the collocation points are appropriately rescaled as given in (3.1)

$$|f(x) - p(x)| \leq \frac{(b-a)^{n+1}}{2^{2n+1}(n+1)!} \max_{a \leq x \leq b} |f^{(n+1)}(x)|.$$

Properties of the Chebyshev Polynomials

Chebyshev polynomials

The Chebyshev polynomials are are given as

$$T_n(x) = \cos(n \arccos(x)).$$

Applying trigonometric identities we have

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1 \\ &\vdots \end{aligned}$$

and the recursion

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad n \geq 1$$

Zeros, Maxima and Minima

$T_n(x)$ has n zeros in $[-1, 1]$ at

$$x = \cos\left(\frac{\pi(k - \frac{1}{2})}{n}\right) \qquad k = 1, 2, 3...n.$$

In the same interval there are $n + 1$ extrema (maxima & minima) at

$$x = \cos\left(\frac{\pi k}{n}\right)$$

and at each maximum $T_n = 1$, minimum $T_n = -1$. It is this property which makes these polynomials so useful for the polynomial approximation of functions.

Discrete Orthogonality

If x_k for $k = 1, 2, ...m$ are m zeros of $T_m(x)$, and assuming that $i, j < m$ then

$$\sum_{k=1}^{m} T_i(x_k)T_j(x_k) = \begin{cases} 0 & i \neq j \\ \frac{m}{2} & i = j \neq 0 \\ m & i = j = 0 \end{cases}$$

which is a discrete orthogonality relation.

Continuous Orthogonality

It also satisfies a continuous orthogonality relation on the interval $[-1, 1]$, and over a weight $(1 - x^2)^{-1/2}$.

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0 & i \neq j \\ \frac{\pi}{2} & i = j \neq 0 \\ \pi & i = j = 0 \end{cases}.$$

Constructing a Chebyshev Interpolating Polynomial

Theorem 4. *If $f(x)$ is an arbitrary function in the interval $[-1, 1]$, and if N coefficients c_j with $j = 0, ..., N-1$ are defined by*

$$\begin{aligned} c_j &= \frac{2}{N}\sum_{k=1}^{N} f(x_k)T_j(x_k) \\ &= \frac{2}{N}\sum_{k=1}^{N} f\left[\cos\left(\frac{\pi(k-\frac{1}{2})}{N}\right)\right]\cos\left(\frac{\pi j(k-\frac{1}{2})}{N}\right) \end{aligned}$$

then the approximation formula

$$f(x) \approx \left[\sum_{k=0}^{N-1} c_k T_k(x)\right] - \frac{1}{2}c_0$$

is exact for x equal to all the N zeros of $T_N(x)$.

This representation may not necessarily be the *best* polynomial exact at all nodes, however it can be truncated (and thus easily calculated) to provide a polynomial of degree m much less than N, which does provide the best approximation at degree m.

Consider the truncated series

$$f(x) \approx \left[\sum_{k=0}^{m-1} c_k T_k(x)\right] - \frac{1}{2}c_0$$

of a function which is nearly perfectly represented by the infinite series. Since we have $|T_k(x)| \leq 1$, the error can be no larger than the sum of the truncated c_k's. If these c_k's are decreasing (a typical scenario) then the dominant term is $c_m T_m(x)$, which is oscillatory with $m+1$ extrema equally spaced over the interval. This smooth distribution is extremely important as it close to minimises the maximum deviation from the true function.

Thus, in order to implement this process for the estimation of some function $f(x)$, we need to evaluate the c_j's of the theorem definition, and from inspection decide on a truncating value m. Then f can be computed. For functions between some arbitrary limits $[a, b]$, the change of variable

$$y \equiv \frac{x - \frac{1}{2}(b-a)}{\frac{1}{2}(b-a)}$$

(which we saw above) and estimate of f by a Chebyshev polynomial in y, will provide the result.

This method works well in many cases. Further, it is known that any continuous function on any interval $[a, b]$ can be approximated arbitrarily well by polynomials, in the sense that the maximum error over the whole interval can be made as small as one likes [this is the Weierstrass Approximation Theorem]. However, collocation at these Chebyshev nodes will not work for all continuous functions: indeed no choice of points will work for all cases (for proof see Theorem 6 on page 288 of Kincaid and Cheney). One way to understand the problem is that the error bound relies on derivatives of ever higher order, so does not even apply to some continuous functions.

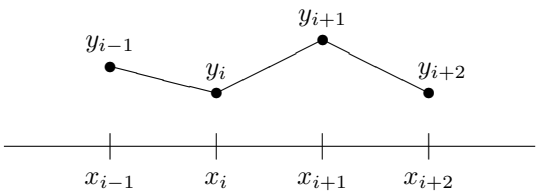
This suggests a new strategy: break the interval $[a, b]$ into smaller intervals, approximate on each interval by a polynomial of some small degree, and join these polynomials together. Hopefully, the errors will only depend on a few derivatives, and so will be more controllable, while using enough nodes and small enough intervals will allow the errors to be made as small as desired.

3.1.7 Piecewise linear interpolation

Piecewise Linear Interpolation

The idea of approximating a function (or interpolating between a set of data points) with a function that is piecewise polynomial takes its simplest form using *continuous piecewise linear*

functions. Indeed, this is the method most commonly used to produce a graph from a large set of data points: the PYTHON matplotlib command `plot` does it for example. The idea is simply to draw straight lines between each successive data point.



Example 7 (Piecewise Linear Interpolation of sine function)**.** *An example piecewise linear interpolation of the sine function is given in Figure 3.8.*

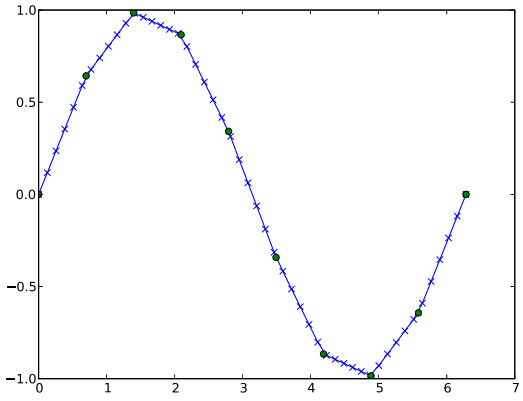


Figure 3.8: Piecewise linear interpolation of the sine function.

The PYTHON code used to obtain Figure 3.8 is given below.

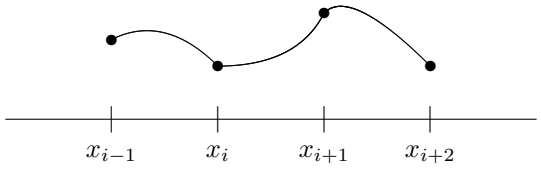
```
from numpy import linspace, sin, pi, interp
from matplotlib.pyplot import plot, show

# construct the data points

xp = linspace(0, 2*pi, 10)
yp = sin(xp)

# fit the polynomial

x = linspace(0, 2*pi, 50)
y = interp(x, xp, yp)

# plot the results

plot(x, y, '-x')
plot(xp, yp, 'o')
show()
```

There are many other methods of approximating functions some of which are discussed in, for example, [**?**, Chapter 7.4]. We will now look at the use of spline functions.

3.1.8 Cubic spline interpolation

Smooth Interpolation

The general strategy of spline interpolation is to approximate with a piecewise polynomial function, of some fixed degree k, that is as smooth as possible across the joins between different polynomials. Smoothness is measured by the number of continuous derivatives that the function has at the knots.



If a piecewise linear approximation is constructed to pass through a given set of $n+1$ points or *knots*

$$(t_0, y_0), \ldots, (t_n, y_n)$$

and is linear in each of the n interval between them; the 'smoothest' curve that one can get is the continuous one given by using linear interpolation between each consecutive pair of points. Less smooth functions are possible, for example the piecewise constant approximation where $L(x) = y_i$ for $x_i \leq x < x_{i+1}$.

Cubic Splines

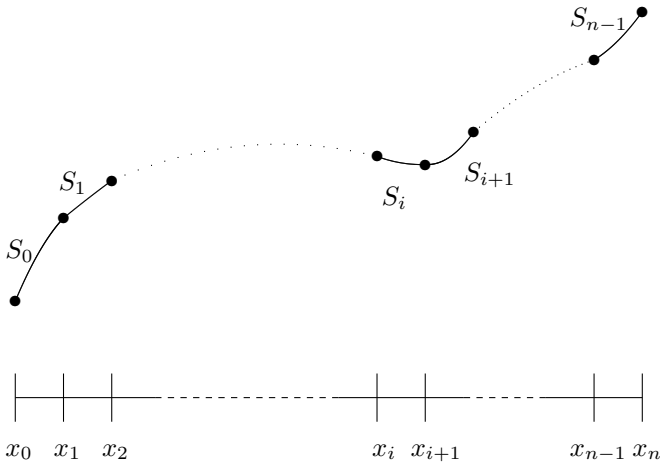
Cubic splines are piecewise cubic polynomials $S \in C^2$, i.e., they are twice differentiable and have a continuous second derivative $S''(x)$.

$$S(x) = S_i(x), \quad x_i \leq x \leq x_{i+1}, \quad 0 \leq i < n.$$

The interpolation conditions are

$$S_i(x_i) = y_i, \quad S_i(x_{i+1}) = y_{i+1}, \qquad 0 \leq i < n.$$

Example Cubic Spline



To simplify the calculations lets assume that

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3.$$

As the cubic spline shown in the previous example must pass through the knots we know that

$$S_i(x_i) = y_i \quad (= a_i) \tag{3.2}$$

and

$$S_i(x_{i+1}) = y_{i+1}. \tag{3.3}$$

The smoothness condition ($S \in C^2$) also needs to be enforced as

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \tag{3.4}$$

and

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}). \tag{3.5}$$

Equations (3.2), (3.3), (3.4) and (3.5) can be combined to form a system of equations. This system can *almost* be used to find a_i, b_i, c_i and d_i.

It turns out that just using Equations (3.2), (3.3), (3.4) and (3.5) results in an under-determined system, appropriate boundary conditions make the system well-posed.

Natural Splines

Natural or *free* splines are characterised by their zero second derivative boundary conditions

$$S''(x_0) = S''(x_n) = 0. \tag{3.6}$$

Clamped Splines

When the spline is to be used to interpolate a function $f(x)$ an alternative choice of boundary conditions is to specify the derivative of the spline function to match that of f at the endpoints:

$$S'(x_0) = f'(x_0), \qquad S'(x_n) = f'(x_n). \tag{3.7}$$

This is called a *clamped spline*. When the function f or its derivatives are not known, they can be approximated from the data itself. Thus a generalisation of the last condition is

$$S'(t_0) = d_0, \qquad S'(t_n) = d_n \tag{3.8}$$

for some approximations of the derivatives.

Though the algorithm for natural cubic spline interpolation is widely available in software [such as the MATLAB command `spline`], it is worth knowing the details. In particular, it is then easy to consider minor changes, like different conditions at the end points.

Example 8 (Cubic Interpolation on sine function)**.** *An example natural cubic spline interpolation of the sine function is given in Figure 3.9. Compare this to Figure 3.8.*

3.1.9 Piecewise cubic Hermite interpolation

Hermite Interpolation with Piecewise Cubic Functions

A Hermite interpolant $H(x)$ of a function $f(x)$ satisfies two interpolation conditions per point, one for the function values $f(x_i)$ and one for the derivatives. This gives 4 conditions per interval $[x_i, x_{i+1}]$:

$$\begin{aligned} H(x_i) = f(x_i) = y_i, \qquad & H'(x_i) = f'(x_i) = y_i' \\ H(x_{i+1}) = f(x_{i+1}) = y_{i+1}, \qquad & H'(x_{i+1}) = f'(x_{i+1}) = y_{i+1}'. \end{aligned}$$

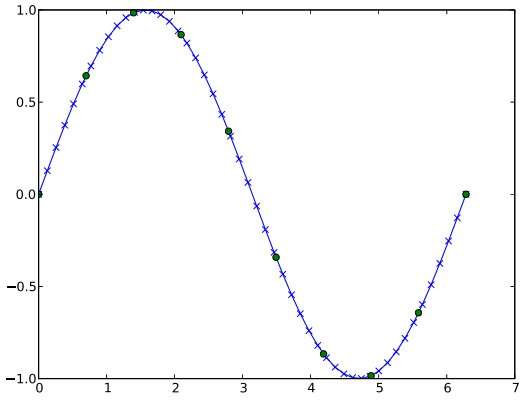


Figure 3.9: Cubic spline interpolation of the sine function.

These four conditions characterise the four coefficients of the cubic polynomial in $[x_i, x_{i+1}]$. This can be seen by using the parametrisation:

$$H_i(x) = a_i + b_i(x - x_i) + (x - x_i)^2[c_i + d_i(x - x_{i+1})].$$

With $h_i = x_{i+1} - x_i$ the four interpolation conditions give

$$\begin{aligned} a_i &= y_i, \qquad b_i = y_i', \\ c_i &= \frac{y_{i+1} - y_i}{h_i^2} - \frac{y_i'}{h}_i, \qquad d_i = \frac{y_{i+1}' + y_i'}{h_i^2} - \frac{2(y_{i+1} - y_i)}{h_i^3}. \end{aligned}$$

Hermite interpolation in general consists of finding a polynomial $H(t)$ to approximate a function $f(t)$ by giving a set of points $t_0, \ldots, t_n$ and requiring that the value of the polynomial and its first few derivatives match that of the original function.

With more points, one could look for higher order polynomials, but it is useful in some cases to construct a piecewise cubic approximation, with the cubic between each consecutive pair of nodes determined only by the value of the function and its derivative at those nodes.

The accuracy is about as good as for clamped splines: the trade off is that the Hermite approximation is less smooth (only one continuous derivative at the nodes), but the error is 'localised'. That is, if the fourth derivative of f is large or non-existent in one interval, the accuracy of the Hermite approximation only suffers in that interval, not over the whole domain.

However this comparison is a bit unfair, as the Hermite approximation uses the extra information about the derivatives of f. This is also often impractical: either the derivatives are not known, or there is no known function f but only a collection of values y_i.

We have written some PYTHON routine to fit a natural cubic spline. An example code to call the routine is given below (the routine is available with the tutorials).

```
import sys
sys.path.append("/home/stals/math3511/python_tut")
from natural_cubic import cubic_spline
from numpy import linspace, sin, pi
from matplotlib.pyplot import plot, show

# construct the data points

xp = linspace(0, 2*pi, 10)
yp = sin(xp)
```

```
# fit the polynomial

x = linspace(0, 2*pi, 50)
y = cubic_spline(x, xp, yp)

# plot the results

plot(x, y, '-x')
plot(xp, yp, 'o')
show()
```

3.7 Quadrature

3.7.1 Integrals

The determination of integrals is an essential component of scientific computing. A simple fundamental problem is the determination of the integral

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

over a finite intervall $[0, 1]$ for a given function $f(x)$. For the applications here one assumes that the function f is either given by some computational procedure which allows us to determine $f(x)$ from any given $x \in [a, b]$ or given by a collection of (approximate) function values $y_i = f(x_i)$ for some (approximately) known data points x_i. These values may orginate from (lengthy) computations or from observations. Often the input data (x_i, y_i) depends on both calculations and data. In applications one often has the (slightly) more general case where

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

for some (explicitely) given density function ρ.

Quadrature or the determination of the values of integrals is another frequent task in scientific computing. It is often related to geometric questions relating to lengths, areas or volumes of geometric objects. More generally in physical problems integrals might relate to the mass of some object, its energy or the force excerted on an object. For any area using probability the determination of the probability and the expection boils down to computing an integral. Connected concepts from data analysis are averages, covariance and cumulative effects. In financial applications one encounters (expected) costs and values of assets and in decision making one needs to determine expected risks of certain decisions. Regarding weather and the environment one computes average rainfall and other averages over both time and space.

A very important source of quadrature problems originates from scientific computing itself. In particular, whenever one solves integral equations or differential equations with finite element, boundary element or spectral methods, computing integrals is a key component. The solution of intial value problems for ordinary differential equations often starts by reformulating the differential equations into integral equations. For example by integrating the initial value problem

$$\frac{du(x)}{dx} + k(x)u(x) = f(x), \quad u(0) = u_0$$

one gets the integral equation

$$u(x) + \int_0^x k(t)u(t)\,dt = u_0 + \int_0^x f(t)\,dt.$$

This suggests that numerical methods for the solution of this equation will require methods for the deterimination of the integral term.

The mathematics of integration has had a long and fruitful history over several thousand years and is still an active area of resarch. As has been the case over most of history, applications and theoretical development have been closely linked.

Assume that we are given a continous function $f(x)$ and would like to determine $\int_a^b f(x)\,dx$ for some real numbers a and b. If we are now able to find a Function $F(x)$ such that $f(x)$ is the derivative, i.e.,

$$f(x) = \frac{dF(x)}{dx}$$

then one has an explicit formula for the integral I as

$$I = \int_a^b f(x)\,dx = F(b) - F(a).$$

Using this one is able to compute integrals for a good collection of functions $f(x)$ including polynomials and trigonometric functions. Several numerical quadrature methods (like the trapezoidal rule or the Newton-Cotes methods) make use of this fact. The quadrature method is obtained interpolating the data with a simpler function (like a polynomial) for which the antiderivative or indefinite integral $F(x)$ is known. Note that while $F(x)$ is only determined up to a constant this constant disappears when computing the difference $F(b) - F(a)$. We will discuss this further later on. The connection between derivatives and integrals stated above is theoretically very important and is typically referred to as the fundamental theorem of calculus.

Note that an antiderivative $F(x)$ is only known for a limited number of continuous functions $f(x)$ even though we know that it exists for every continuous function. A consequence is that numerical techniques are essential for their computation. Interestingly, the theory of integration itself is based on a numerical quadrature method, the Riemannian sums. We will investigate the definition and numerical properties Riemannian sums later when we discuss numerical methods.

From what has been said so far one sees that there is a close connection between quadrature and function approximation and in particular interpolation. In particular, the Taylor remainder theorem will play an important role in error analysis. As an aside, the inventor of a set of axioms for the integers, the italian mathematician Peano also came up with a very elegant error formula for a large class of numerical integration methods.

Integration is one mathematical task where an exact formula for the solution is rarely available: even products and compositions of easily integrated functions often have an indefinite integral not expressible in terms of standard functions. One famous and important example is the expression for the "error function" arising in the study of the normal distribution in probability

$$\operatorname{erf}(x) := \frac{1}{\sqrt{2\pi}} \int_0^x e^{-t^2/2}\,dt.$$

On the other hand, the definition of the integral by Riemann sums can easily be used to compute numerical approximations: it is far from the best method, but makes a good starting point.

Since we are dealing with numerical values, all integrals from now on will be definite ones, so we are seeking ways to accurately, efficiently and reliably compute approximations of

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

We mentioned the close connection between integration, probability and statistics. In fact, one can use sampling to approximate integrals. This is basic idea behind the Monte Carlo methods which are used widely in scientific computing. For example, if one has given a density function $\rho(x) \geq 0$ satisfying $\int_a^b \rho(x)\,dx = 1$ one may approximate the expectation of the random variable X with probability density $\rho(x)$ by

$$E(X) = \int_a^b x\rho(x)\,dx \approx \frac{1}{n}\sum_{i=1}^{n} X_i$$

where the X_i are random samples of the random variable X. One can show that this estimator has a variance proportional to $1/\sqrt{n}$. This gives a very simple basic approximation of the integral. We will see that most numerical techniques have a substantially better convergence rate than $O(n^{-1/2})$ but require that the density is relatively smooth.

3.7.2 Quadrature and Python

Fundamentally, a quadrature method takes as an input a function $f(x)$ and the integration boundaries a and b and returns a number I. There is a simple function "quad" in scipy.integrate which does exactly this. For example, to determine the integral

$$\int_3^4 \frac{\exp(x)}{(1+x^2)^{-3}}\,dx$$

one calls

```
import scipy.integrate as sci
import math as m
I = sci.quad(lambda x : m.exp(x)/(1.0+x*x)**3, 3.0, 4.0)
```

The resulting I is has then two components: I[0] is the resulting numerical approximation, while I[1] an estimate for the error of I[0]. For the computation it uses methods from the Quadpack quadrature package which are of high numerical quality. For the simple approach taken above all one needs is the concept of function evaluations which has been covered in the previous section.

Many quadrature methods take the form

$$Q(f) = \sum_{i=1}^{n} w_i f(x_i).$$

Note that even the Monte Carlo method is of this form with $w_i = 1$. Usually the weights are $w_i > 0$. The computational cost of determining $Q(f)$ is almost exclusively due to the cost of evaluation of the functions $f(x_i)$. However, if one has a very large number of function evaluations one might wish to evaluate f in parallel so that the sum will require moving the results of the function evaluations between processors. This can in the end also be an important part of the computations. While evaluating the functions requires a total amount of work which is proportional to n which can be done in parallel so that if one has n processors this can be done in the time it takes one processor to get the result. Summing up, however, cannot be done in parallel and requires $\log_2(n)$ steps. The weights are typically precomputed. They will be discussed later.

3.7.3 Composite rules – the trapezoidal rule

Riemannian sums

Riemann Sum

The Riemann sum is fundamental in the definition of the Riemann integral which is just the limit of Riemann sums. It is arguably the simplest quadrature rule and is also used in the form of the midpoint rule.

In a first step the quadrature points x_i are selected such that

$$a = x_0 < x_1 < \cdots < x_n = b.$$

The integral is then the sum of all the integrals over the subintervals given by the quadrature points. If f is approximated by a piecewise constant function with values $f(\bar{x}_i)$ for the interval $[x_i, x_{i+1}]$ one gets the approximant

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \approx (x_{i+1} - x_i)\, f(\bar{x}_i) \tag{3.9}$$

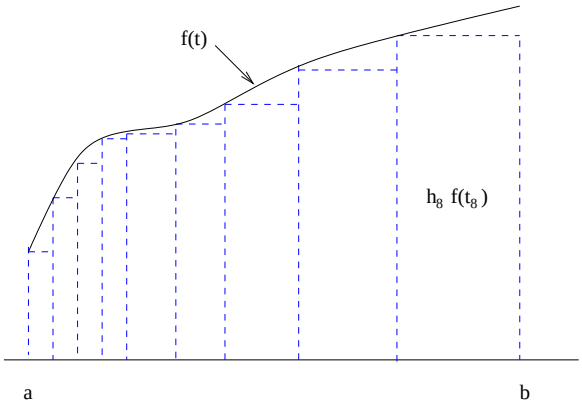


Figure 3.10: Example use of the Riemann sum with $\bar{x}_i = x_i$ to approximate the area under the curve

and the resulting composite quadrature rule is then

$$I = \int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} h_i f(\bar{x}_i) \tag{3.10}$$

where $h_i := x_{i+1} - x_i$ as usual.

Figure 3.10 illustrates the idea behind the Riemann sum. In this example $f(\bar{x}_i) = f(x_i)$.

It is known that as the intervals get smaller, in that $h := \max h_i \to 0$, these approximations will converge to the exact value so long as f is continuous on the interval $[a, b]$.

Unlike differentiation, rounding errors are not a major problem for integration. Integration is typically a well-posed problem and there is no division by small terms.

Riemann Sum - Error Formula

An error formula will be derived, first for (3.9) and then for the composite rule (3.10) that follows from it.

This does require an extra assumption: that f is differentiable, so that Taylor polynomials can be used. Then on each interval

$$f(x) = f(x_i) + f'(\xi_x)(x - x_i).$$

If f is twice differentiable and we consider the limit as $h \to 0$, $f(x) = f(x_i) + f'(x_i)(x - x_i) + O(h^2)$ which can make the estimation of errors a bit simpler.

The error in (3.9) is given by integrating this:

$$\int_{x_i}^{x_{i+1}} f(x)\,dx - h_i f(\bar{x}_i) \quad = \quad \int_{x_i}^{x_{i+1}} (f(x) - f(\bar{x}_i))\,dx.$$

Defining $M := \max\{|f'(x)| : x \in [a, b]\}$, the integrand is bounded by $|f(x) - f(\bar{x}_i)| \le M h_i \le M h$ and so the integral can be bounded

$$\left| \int_{x_i}^{x_{i+1}} f(x)\,dx - h_i f(\bar{x}_i) \right| \le M h_i^2 \le M h^2.$$

For a given number of intervals n, the smallest h is achieved by using equally spaced points

$$x_i = a + ih, \quad h = (b - a)/n.$$

While hardly used in practice, we choose here $\bar{x}_i = x_i$.

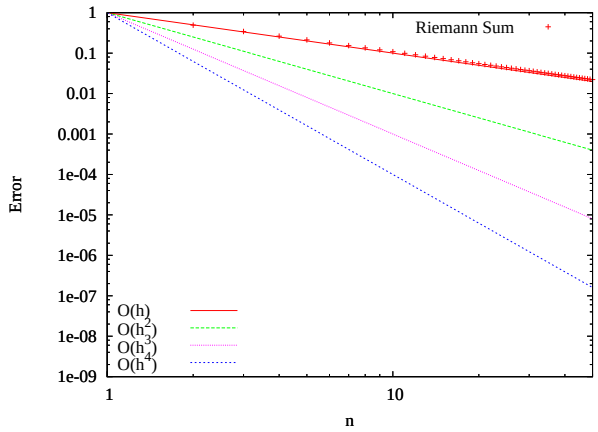


Figure 3.11: Example of the approximation found the Riemann sum.

However, as with the use of Chebyshev points for interpolations, there are examples where it is more appropriate to choose unequally spaced points.

An error bound for (3.10) is given simply by summing these error bounds, so that

$$\begin{aligned}\left|I-\sum_{i=0}^{n-1}h_i f(\bar{t}_i)\right| &= \left|\sum_{i=0}^{n-1}\int_{t_i}^{t_{i+1}} f(t)dt-\sum_{i=0}^{n-1}h_i f(\bar{t}_i)\right| \\ &\leq \sum_{i=0}^{n-1}\left|\int_{t_i}^{t_{i+1}} f(t)dt-h_i f(\bar{t}_i)\right| \\ &\leq Mh^2 n, \\ &= \frac{M(b-a)^2}{n}=M(b-a)h,\end{aligned}$$

for equally spaced points.

This result is rather poor: for example if M and $b-a$ are of order 1, computing to the fifteen decimal places of accuracy available with typical 64-bit machine arithmetic requires a completely impractical $n \approx 10^{15}$. Even a modest six decimal places of accuracy requires millions of operations. A substantially better approach is obtained by choosing $\bar{x}_i = x_i + h_i/2$.

Example 9 (The integral of $x\,(2\sin(x)+x\cos(x))$ from $x=0$ to $x=1$)**.** *Figure 3.11 shows a log-log plot of the error if the Riemann sum is used to approximate the integral.*

The foundation of almost all better methods is to replace the piecewise constant approximation of f that underlies the Riemann sum by more accurate approximating functions, and then integrate them. Polynomial and piecewise polynomial approximations are natural choices and even the next simplest idea of using piecewise linear functions can be used to build a very good algorithm.

The Trapezoidal Rule

Trapezoidal Rule (trapezoid rule, trapezium rule)

Again the first step is to approximate the integral over each interval $[x_i, x_{i+1}]$, this time using the linear approximation through the values at the endpoints:

$$f(x) \approx p_1(x) = f(x_i)\frac{x_{i+1}-x}{h_i} + f(x_{i+1})\frac{x-x_i}{h_i}$$

with the integral

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \approx \int_{x_i}^{x_{i+1}} p_1(x)\,dx = h_i\frac{f(x_i)+f(x_{i+1})}{2}.$$

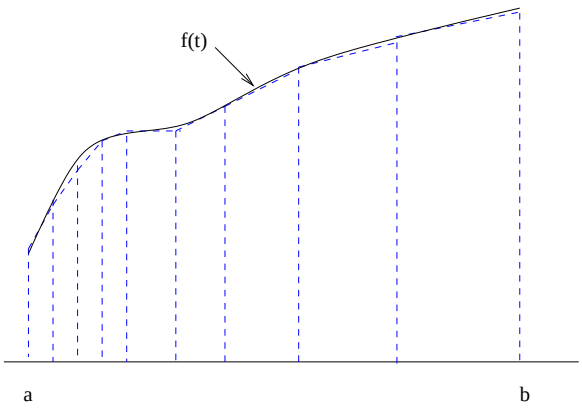
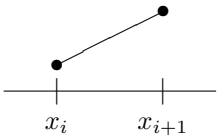


Figure 3.12: Example use of the trapezoid rule to approximate the area under the curve



This is also the area of the trapezoid lying between the linear interpolating function and the axis over this interval, hence its name: the *trapezoid rule*. In the most important case of equally spaced points, summing these approximations can be simplified by combining the two occurrences of $f(t_i)$ at each internal point: the result is

$$I \approx T(f,h) := \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right]. \tag{3.11}$$

This is properly called the *composite trapezoid rule*, though as it is the one used in practice, its name is often shortened to just "the trapezoid rule".

Comparing Figure 3.10 to Figure 3.12 demonstrates the improved accuracy of the trapezoid rule.

The composite trapezoidal rule is defined as the integral over the piecewise linear polynomial interpolant.

Trapezoid Rule - Error Formulas

Error formulas for the trapezoid and composite trapezoid rules can be derived much as was done for the Riemann sums, this time needing the assumption that f has two continuous derivatives. However there are several ways of getting better results.

Start with the error formula for the polynomial interpolation

$$f(x) - p_1(x) = -\frac{f''(\xi_x)}{2}(x_{i+1} - x)(x - x_i)$$

and integrate.

Noting that the quadratic term is positive, this integrand lies in the range

$$\begin{aligned} -\frac{M}{2}(x_{i+1} - x)(x - x_i) &\leq f(x) - p_1(x) \\ &\leq -\frac{L}{2}(x_{i+1} - x)(x - x_i), \end{aligned}$$

where L and M are lower and upper limits on the second derivative in this interval:

$$\begin{aligned} L &= \min\{f''(x) : x_i \leq x \leq x_{i+1}\} \\ M &= \max\{f''(x) : x_i \leq x \leq x_{i+1}\}. \end{aligned}$$

Integrating gives corresponding limits on the error for the trapezoid rule. It contains a coefficient in the range $[L, M]$, and due to the continuity of f'' this is equal to the value of f'' at some point in the interval, giving

$$\begin{aligned}\int_{x_i}^{x_{i+1}} f(x)\,dx - h\frac{f(x_i)+f(x_{i+1})}{2} &= \int_{x_i}^{x_{i+1}} -\frac{f''(\xi_{x,i})}{2}(x_{i+1}-x)(x-x_i)\,dx \\ &= \frac{-f''(\xi_i)}{2}\int_{x_i}^{x_{i+1}}(x_{i+1}-x)(x-x_i)\,dx \\ &= -\frac{1}{12}f''(\xi_i)h^3.\end{aligned}$$

We have used the error formula for linear interpolation along with the mean value theorem for integration.

Theorem 5 (Mean Value Theorem for Integration)**.** *If $G : [a,b] \to \mathbb{R}$ is continuous and $\psi : [a,b] \to (0,\infty)$ is integrable then*

$$\int_a^b G(x)\psi(x)\,dx = G(\xi)\int_a^b \psi(t)\,dt$$

for some $\xi \in [a,b]$.

Proof. If $m \le G(x) \le M$ for $x \in [a,b]$ then

$$m\int_a^b \psi(x)\,dx \le \int_a^b G(x)\psi(x)\,dx \le M\int_a^b \psi(x)\,dx$$

or

$$m \le \frac{1}{\int_a^b \psi(x)\,dx}\int_a^b G(x)\psi(x)\,dx \le M.$$

Let

$$d = \frac{1}{\int_a^b \psi(x)\,dx}\int_a^b G(x)\psi(x)\,dx.$$

Since G is continuous and $m \le d \le M$, by the intermediate value theorem there exists an $\xi \in [a,b]$ such that $G(\xi) = d$. □

For simplicity, we assume that the points are equally spaced from now on. Then summing the error terms above for each interval gives an error formula for the composite trapezoid rule containing all the values $f''(\xi_i)$:

$$I - T(f,h) = -\frac{h^3}{12}\sum_{i=0}^{n-1} f''(\xi_i). \tag{3.12}$$

These terms can be replaced by a single term $f''(\xi)$ by considering the continuity of f'' again, giving

$$I - T(f,h) = -\frac{1}{12}(b-a)f''(\xi)h^2, \quad a < \xi < b. \tag{3.13}$$

Thus the errors are $O(h^2)$, which is already a big improvement over the result for Riemann sums. Six decimal places of accuracy are now typically given a somewhat more reasonable several thousand points, but the cost of fifteen decimal places is still quite prohibitive at some tens of millions.

Another useful error formula can be derived by rearranging (3.12) as

$$I - T(f,h) = -\frac{h^2}{12}\left[\sum_{i=0}^{n-1} hf''(\xi_i)\right].$$

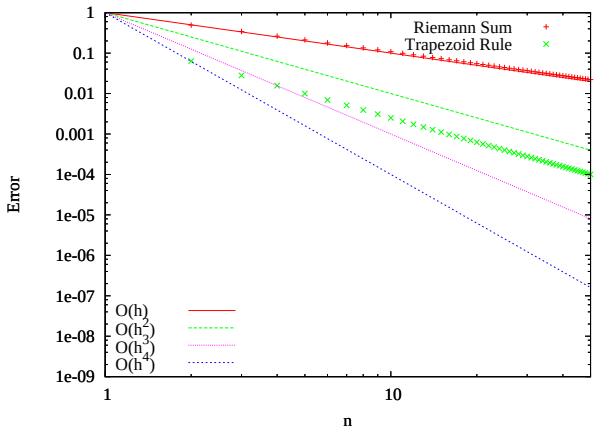


Figure 3.13: Example of the approximation found using the trapezoid rule.

The sum in brackets is a Riemann sum approximating the integral of the second derivative and using the above error formula for such approximations,

$$\begin{aligned} I - T(f,h) &= -\frac{h^2}{12}\left(\int_a^b f''(t)\,dt + O(h)\right) \\ &= -\frac{h^2}{12}[f'(b) - f'(a)] + O(h^3). \end{aligned}$$

This shows that the constant in front of h^2 is independent of h, unlike equation (3.13) where ξ does depend on h. Having this sort of error formula means that Richardson extrapolation can be applied, at least once. One would like to be able to do this repeatedly: this can in fact be justified, and leads to the next topic of Romberg integration.

Example 10 (The integral of $x\,(2\sin(x) + x\cos(x))$ from $x = 0$ to $x = 1$)**.** *Figure 3.13 shows a log-log plot of the error if the trapezoid rule is used to approximate the integral.*

3.7.4 Newton-Cotes methods

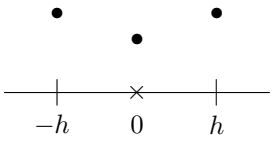
Quadratic Interpolation

The (composite) trapezoid rule was based on the integration of (piecewise) linear interpolating functions. The next step is to use quadratic approximations, and this turns out to be more accurate than one might expect at first.

Start first with a single quadratic through equally spaced points. Since the independent variable can be shifted by a constant, it is convenient to consider first

$$I = \int_{-h}^{h} f(x)\,dx$$

and use an approximating polynomial given by collocation through the points $-h, 0, h$.



Later it will be seen that these are not the optimal points. A better distribution of points is given by Gaussian quadrature, just as the Chebyshev points are better for interpolation.

One can see that the second order interpolation polynomial is

$$p_2(t) = f(0) + \frac{f(h) - f(-h)}{2h}t + \frac{f(-h) - 2f(0) + f(h)}{h^2}\frac{t^2}{2}$$

whose integral gives the approximation

$$I \approx \int_{-h}^{h} p_2(x)\,dx = \frac{h}{3}[f(-h) + 4f(0) + f(h)] \tag{3.14}$$

which is *Simpson's Rule* for a single *panel*, consisting of two of the intervals between data points.

Before extending Simpson's rule to the composite rule needed in practice, note that the derivation can also be done by the method of undetermined coefficients: this will be essential in more difficult cases later.

Simpsons Rule by the Method of Undetermined Coefficients

We seek an approximation

$$I \approx A_0 f(-h) + A_1 f(0) + A_2 f(h).$$

Indeed, one expects the coefficients to be proportional to the length of the interval, but not to change if the x-variable is simply translated. So it is reasonable to guess that $A_i = ha_i$ gives true constants a_i. Anyway, this will be confirmed in the calculation.

Exactness for quadratics means exactness for $f(x) = 1, x, x^2$ and inserting each of these in turn gives

$$\begin{aligned} 2h &= h(a_0 + a_1 + a_2) \\ 0 &= h^2(-a_0 + a_2) \\ \frac{2}{3}h^3 &= h^3(a_0 + a_2). \end{aligned}$$

The solution is $a_0 = a_2 = 1/3, a_1 = 4/3$, confirming the expected scaling by h of the coefficients and giving (3.14) again.

Example 11 (The integral of $x\,(2\sin(x) + x\cos(x))$ from $x = 0$ to $x = 1$)**.** *Figure 3.14 shows a log-log plot of the error if Simpson's rule is used to approximate the integral.*

The surprise with Simpson's rule comes when its error for cubics is examined (see Figure 3.14). $f(x) = x^3$ gives the error

$$\begin{aligned} &\int_{-h}^{h} x^3\,dx - \frac{h}{3}(f(-h) + 4f(0) + f(h)) \\ &= 0 - \frac{h}{6}(-h^3 + 0 + h^3) \\ &= 0 \end{aligned}$$

which with exactness for quadratics gives exactness for all cubics. This leads to higher order accuracy when the error is examined using Taylor series.

Exactness for cubics means that in order to get any information about errors from Taylor series, one must go at least to fourth powers, so substitute in

$$f(x) = f(0) + f'(0)x + \cdots \frac{f^{(4)}(0)}{4!}x^4 + \frac{f^{(5)}(\xi_x)}{5!}x^5.$$

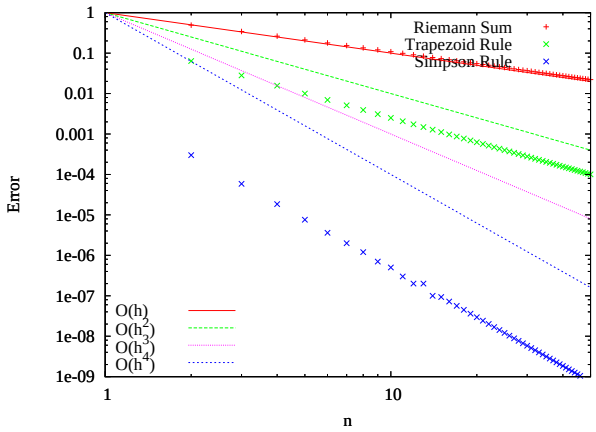


Figure 3.14: Example of the approximation found using the trapezoid rule.

On the domain of integration the last term is $O(h^5)$ and in both the integral and the approximation formula contributes only a term $O(h^6)$. On the other hand the terms up to x^3 contribute no error. Thus the error comes mostly from the x^4 term, and is

$$\begin{aligned}
&\frac{f^{(4)}(0)}{4!}\left[\int_{-h}^{h} x^4\,dx - \frac{h}{3}(-h)^4 + 4.0 + h^4\right] + O(h^6) \\
&= \quad \frac{f^{(4)}(0)}{4!}h^5\left[\frac{2}{5} - \frac{2}{3}\right] + O(h^6) \approx -\frac{f^{(4)}(0)}{90}h^5.
\end{aligned}$$

To derive the composite rule and its error formula, consider the points $x_i = a+ih, h = (b-a)/n$. As Simpson's rule applies to a pair of intervals, there must be an even number of intervals; that is, n must be even. The integral is then expressed as the sum of the integrals

$$\int_{x_i}^{x_{i+2}} f(x)\,dx$$

for i even. This has the same width $2h$ as the interval considered above, so the change of variables $\tilde{x} = x - x_{i+1}$ gives the approximation

$$\int_{x_i}^{x_{i+2}} f(x)\,dx = \frac{h}{3}(f(x_i) + 4f(x_{i+1}) + f(x_{i+2}))$$

with error of

$$-\frac{f^{(4)}(x_{i+1})}{90}h^5.$$

Summing these gives the *composite Simpson's rule*

$$\begin{aligned}
\int_a^b f(x)\,dx &\approx S(f,h) \\
&= \frac{h}{3}[f(a) + 4f(a+h) + 2f(a+2h) + \cdots \\
&\quad +2f(b-2h) + 4f(b-h) + f(b)] \\
&= \frac{h}{3}\left[f(a) + 2\sum_{i=2}^{n/2} f(x_{2i-2}) + 4\sum_{i=1}^{n/2} f(x_{2i-1}) + f(b)\right].
\end{aligned}$$

Summing the errors for each part gives the total error of

$$-\frac{h^5}{90}\sum_{i=1}^{n/2} f^{(4)}(x_{2i-1}) + nO(h^6) \tag{3.15}$$

Dividing the sum at right by $n/2$ gives an average of values of the fourth derivative, which for a continuous function must be equal to its value at some point ξ in the domain of integration: the sum is $f^{(4)}(\xi)n/2$. As $n = (b-a)/h$, the total error is

$$I - S(f,h) = -\frac{b-a}{180} f^{(4)}(\xi)h^4 + O(h^5).$$

In fact a more careful calculation shows that ξ can be chosen in a way that eliminates the term $O(h^5)$, giving

$$I - S(f,h) = -\frac{b-a}{180} f^{(4)}(\xi)h^4.$$

Another way to simplify (3.15) is to note that the sum involved is related to a Riemann sum for the integral of the fourth derivative, with intervals of width $2h$. Thus the error can be rewritten as

$$\begin{aligned} & -\frac{h^4}{180}\sum_{i=1}^{n/2} 2h f^{(4)}(x_{2i-1}) + nO(h^6) \\ \approx\ & -\frac{h^4}{180}\int_a^b f^{(4)}(x)\,dx \\ =\ & -\frac{h^4}{180}[f'''(b) - f'''(a)]. \end{aligned}$$

This resembles the h^4 term in the error formula for the trapezoid rule, and there is a connection through Richardson extrapolation.

Richardson extrapolation from the composite trapezoid rule gives another derivation of both Simpson's rule and its error formula the composite trapezoid rule. Using the trapezoid values for one and two intervals and setting $h = (b-a)/2$, this extrapolation gives

$$\begin{aligned} & \frac{4T(f,(b-a)/2) - T(f,(b-a))}{3} \\ =\ & \frac{4h/2[f(a) + 2f((a+b)/2) + f(b)] - h[f(a) + f(b)]}{3} \\ =\ & \frac{h}{3}[f(a) + 4f((a+b)/2) + f(b)] \end{aligned}$$

which is Simpson's rule again. Extrapolating with more intervals gives the sum of these extrapolations so that

$$S(f,h) = \frac{4T(f,h) - T(f,2h)}{3}$$

Newton-Cotes formulas

The method for deriving Simpson's rule could be continued to use higher order polynomial approximations, fitted to more points. Allowing the points to be chosen more flexibly than with equal spacing, a general form for the approximate integral is

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} A_i f(x_i).$$

So far we have chosen the points x_i first and then determined the coefficients A_i in order to give a good approximation, and this strategy in general is the basis of the *Newton-Cotes formulas*, which use various selections of equally spaced points.

In this case, the coefficients can be determined straightforwardly by the method of undetermined coefficients: substituting in $f(x) = 1, x, x^2, \ldots, x^n$ and demanding that the result be exact gives a set of simultaneous linear equations for the coefficients, and results in a formula that is exact for polynomials of degree up to n, and has an error of the form $Cf^{(n+1)}(\xi)h^{n+1}$ or better.

3.7.5 Gaussian Quadrature

Gaussian Quadrature

Similar to the case of interpolation with Chebyshev points, a carefull choice of the quadrature points x_i can improve approximation performance of quadrature rules.

Gaussian Quadrature is the method of choosing both A_i and x_i such that the quadrature rule is exact for polynomials of degree up to N, for the largest N for which this is possible. As there are $2n+2$ unknowns, it is not surprising that the resulting formulas are exact for all polynomials of degree up to $2n+1$.

Midpoint Rule $(n = 0)$

The simplest case of $n = 0$ gives the approximation

$$\int_a^b f(x)\,dx \approx (b-a)f((a+b)/2)$$

which is exact for linear functions: This is *the midpoint rule*.

Taylor series analysis shows that it is comparable to the trapezoid rule in accuracy: in particular it leads to a composite rule with errors of $O(h^2)$.

Note that in its composite form, the midpoint rule is a Riemann sum with $\hat{x}_i = 0.5(x_i + x_{i+1})$. In fact, this is the only Riemann sum which is computationally relevant.

Gaussian Quadrature for $n = 1$

The next case is $n = 1$. It is easiest to work initially on the symmetrical interval $[-1, 1]$ and then change coordinates to get the general result. Thus the approximation is

$$\int_{-1}^1 f(x)\,dx \approx A_0 f(x_0) + A_1 f(x_1).$$

Requiring that this be exact for the first four powers of x gives

$$\begin{aligned} 2 &= A_0 + A_1, \\ 0 &= A_0 x_0 + A_1 x_1, \\ 2/3 &= A_0 x_0^2 + A_1 x_1^2, \\ 0 &= A_0 x_0^3 + A_1 x_1^3. \end{aligned}$$

Combining the second and fourth equations

$$x_0^2 = x_1^2.$$

They cannot be equal as that would give the midpoint rule, not exact for quadratics, so they are negatives. The second equation then gives $A_0 = A_1$ and the first shows that each of these equals 1. Then the third equation gives $x_1 = \pm 1/\sqrt{3}$ and without loss of generality one can take $x_1 = 1/\sqrt{3} = -x_0$. That is

$$\int_{-1}^1 f(x)\,dx \approx f(-1/\sqrt{3}) + f(1/\sqrt{3}).$$

This formula is exact for cubics, and considerably more accurate than the trapezoid rule which would have arisen if the points x_i were fixed at the endpoints.

Gaussian Quadrature - General n

Given an interval $[a, b]$ and a natural number n, we want to find constants A_i and $x_i \in [a, b]$ such that the approximation

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} A_i f(x_i) \tag{3.16}$$

is exact for all polynomials of degree up to $2n + 1$.

Solving for these constants by the method of undetermined coefficients is hard for all but small values of n, due to the simultaneous non-linear equations involved. There is an easier way of computing them, using *orthogonal polynomials*.

Orthogonal Polynomials

Definition 1 (Orthogonal Polynomials)**.** *Two functions f and g are orthogonal on the interval $[a, b]$ if*

$$\int_a^b f(x)g(x)\,dx = 0.$$

Legendre Polynomials

To see examples of orthogonal polynomials, it can be verified that every two members of the following set are orthogonal on $[-1, 1]$: they are the first few *Legendre polynomials*, which we will see are central to Gaussian quadrature.

$$\begin{aligned} q_0(x) &= 1, \\ q_1(x) &= x, \\ q_2(x) &= x^2 - \frac{1}{3}, \\ q_3(x) &= x^3 - \frac{3}{5}x, \\ q_4(x) &= x^4 - \frac{6}{7}x^2 + \frac{3}{35}. \end{aligned}$$

Quadrature rules and orthogonal polynomials

Orthogonal polynomials are useful because of the following result:

Theorem 6. *If a polynomial q of degree $n+1$ is orthogonal to all polynomials of lower degree on the interval $[a, b]$ then*

1. It has $n + 1$ distinct roots x_i with $a < x_0 < \cdots x_n < b$

2. If one uses these roots to determine the weights A_i in the approximate integration formula (3.16) so that it is exact for all polynomials of degree up to n, then it is in fact exact for all polynomials of degree up to $2n + 1$

3. No approximate integration formula of the form (3.16) can be exact for all polynomials of degree $2n + 2$, so in a sense this is the best possible method using $n + 1$ points.

Proof. **Part 1.** Suppose that q does not have $n+1$ distinct roots in the interval $[a, b]$ as claimed. Then it can only change sign at m points in this interval, with $m \leq n$: call these points $x_1, \ldots, x_m$. With the appropriate choice of the sign, the polynomial

$$s(x) = \pm(x - x_1) \cdots (x - x_m)$$

has the same sign as q through the interval $[a, b]$, and so

$$\int_a^b q(x)s(x)\,dx > 0.$$

But the degree of $s(x)$ is at most n, so this contradicts the orthogonality property of q: this polynomial must have $n+1$ roots in the interval.

Part 2. Let f be any polynomial of degree at most $2n+1$. Polynomial division by q gives $f = qp + r$ where p and r are of degree at most n. Then

$$\begin{aligned}
\int_a^b f(x)\,dx &= \int_a^b p(x)q(x)\,dx + \int_a^b r(x)\,dx \\
&= \int_a^b r(x)\,dx \quad \text{as } q \text{ is orthogonal to } p \\
&= \sum_{i=0}^{n} A_i r(x_i) \quad \text{as degree of } r \text{ is low - approximation exact} \\
&= \sum_{i=0}^{n} A_i p(x_i)q(x_i) + \sum_{i=0}^{n} A_i r(x_i) \quad \text{as } q \text{ vanishes at each } x_i \\
&= \sum_{i=0}^{n} A_i f(x_i).
\end{aligned}$$

So the approximation is exact for f as required.

Part 3. For any choice of the $n+1$ points x_i in this interval, define the polynomial

$$w(x) = (x - x_0)^2 \cdots (x - x_n)^2$$

of degree $2n+2$. Clearly its integral over the interval is positive, but it vanishes at the x_i so that the approximation formula (3.16) gives 0. So no such formula can be exact for all polynomials of degree $2n+2$. □

Quadrature using Orthogonal Polynomials

Using orthogonal polynomials gives the following procedure for computing the above "optimal" $n+1$ point approximate integrals:

1. Find a polynomial of degree $n+1$ orthogonal to all polynomials of lower degree.
2. Find all its roots x_i.
3. Compute the weights A_i by the method of undetermined coefficients.

The second step can be done by searching the interval $[a, b]$ to find $n+1$ smaller interval in which the sign of q changes, and then computing the root in each such interval by any of the methods discussed earlier in the semester.

The third step gives a system of $n+1$ simultaneous linear equations, the conditions that each of $1, x, x^2, \ldots, x^n$ are integrated exactly by the formula. This is also straightforward, using Gaussian elimination.

Transforming $[a, b]$ onto $[-1, 1]$

Thus only step one is left to deal with. This problem can be reduced to the same problem on the standard interval $[-1, 1]$, by using the change of variables

$$\tilde{x} = \frac{2}{b-a}\left(x - \frac{a+b}{2}\right)$$

to convert the integral to one on $[-1, 1]$.

This then leads to a quadrature formula for $[a, b]$ with

$$x_i = \frac{b-a}{2}\tilde{x}_i + \frac{a+b}{2}, \quad A_i = \frac{b-a}{2}\tilde{A}_i$$

where $\tilde{x}_i$ and $\tilde{A}_i$ are the points and weights for the Gaussian rule on $[-1, 1]$.

Using the inverse of the basis transform, one can see that the transformed Legendre polynomials are orthogonal.

Constructing the Legendre Polynomials

The following result is needed:

Proposition 1. *Given any set of polynomials $p_n(x) = x^n + \cdots$ of degree exactly n, any polynomial f of degree n can be written [uniquely] as a linear combination of the polynomials $p_i, i \le n$:*

$$f(x) = c_0 p_0(x) + \cdots + c_j p_j(x) + \cdots + c_n p_n(x). \tag{3.17}$$

suggestion: have a go at proving this result

Proof. Write each of the polynomials as

$$p_j(x) = u_{0j} + \cdots + u_{ij}x^i + \cdots + x^j \tag{3.18}$$

and

$$f(x) = f_0 + \cdots + f_i x^i + \cdots + f_n x^n \tag{3.19}$$

Substituting (3.19) on the left of (3.17) and (3.18) on the right, and matching the coefficients of each power of x gives the equations

$$f_i = \sum_{j=i}^{n} c_j u_{ij}, \quad 0 \le i \le n$$

where $u_{ii} = 1$.

This system of equations has the form

$$\begin{bmatrix} 1 & u_{01} & & & & \\ 0 & 1 & u_{12} & & & \\ & \ddots & \ddots & \ddots & & \\ & & 0 & 1 & u_{i,i+1} & \\ & & & \ddots & \ddots & \\ & & & & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_i \\ \vdots \\ f_n \end{bmatrix}.$$

This matrix is unit upper triangular so there is a unique solution, giving f in terms of the p_n. □

Equation (3.17) guarantees that if a function is orthogonal to each $q_i, i \le n$, it is orthogonal to all polynomials of degree up to n.

Now a method for computing each q_n in turn can be given, by starting at $n = 0$ and choosing each to be orthogonal to all the previous q_i.

Firstly, $q_0(x) = 1$. Then assuming that each $q_i, i \le n$ has been computed, seek q_{n+1} in the form

$$q_{n+1}(x) = x^{n+1} + c_n q_n(x) + \cdots + c_0 q_0(x). \tag{3.20}$$

The orthogonality condition is

$$\begin{aligned} 0 &= \int_{-1}^{1} q_{n+1}(x) q_i(x)\, dx \\ &= \int_{-1}^{1} x^{n+1} q_i(x) + c_n \int_{-1}^{1} q_n(x) q_i(x)\, dx + \cdots + c_0 \int_{-1}^{1} q_0 q_i\, dx \\ &= \int_{-1}^{1} x^{n+1} q_i(x) + c_i \int_{-1}^{1} q_i^2(x)\, dx \end{aligned}$$

as all the other integral vanish due to orthogonality.

Thus

$$c_i = -\frac{\int_{-1}^{1} x^{n+1} q_i(x)}{\int_{-1}^{1} q_i^2(x)\, dx} \tag{3.21}$$

and q_{n+1} is determined by (3.20) and (3.21).

Here is the computation of the first few Legendre polynomials after $q_0(x) = 1$.

$$q_1(x) = x + c_0 q_0(x) = x + c_0$$

$$c_0 = -\frac{\int_{-1}^{1} x q_0(x)}{\int_{-1}^{1} q_0^2(x)\,dx} = -\frac{\int_{-1}^{1} x\,dx}{\int_{-1}^{1} 1\,dx} = 0,$$

so $q_1(x) = x$.

Next

$$q_2(x) = x^2 + c_0 + c_1 x$$

with

$$\begin{aligned} c_0 &= -\frac{\int_{-1}^{1} x^2 \times 1\,dx}{\int_{-1}^{1} 1\,dx} = -\frac{1}{3} \\ c_1 &= -\frac{\int_{-1}^{1} x^2 \times x\,dx}{\int_{-1}^{1} x^2\,dx} = 0, \end{aligned}$$

giving

$$q_2(x) = x^2 - \frac{1}{3}.$$

Similarly one can verify that $q_3(x)$ and $q_4(x)$ are as given above.

The First Few Gaussian Integration Formulas

Computing the weights A_i: case of $n = 0$

Recall that the quadrature points x_i for Gaussian quadrature are the zeros of Legendre polynomials q_n

For $n = 0$, the root of q_1 is $x_0 = 0$. Also

$$\int_{-1}^{1} 1\,dx = A_0 f(0)$$

so $A_0 = 2$, giving the midpoint rule seen earlier:

$$\int_{-1}^{1} f(x)\,dx \approx 2f(0).$$

Case of $n = 1$

For $n = 1$ the roots of $q_2(x) = x^2 - 1/3$ are $\pm 1/\sqrt{3}$. Noting that

$$2 = \int_{-1}^{1} 1\,dx = A_0 + A_1$$

and

$$0 = \int_{-1}^{1} x\,dx = (-A_0 + A_1)/\sqrt{3}$$

shows $A_0 = A_1 = 1$.

This gives the rule

$$\int_{-1}^{1} f(x)\,dx \approx f(-1/\sqrt{3}) + f(1/\sqrt{3})$$

derived in the previous section.

Case of $n = 2$

The next case $n = 2$ has $x_i = -\sqrt{3/5}, 0, \sqrt{3/5}$ and a bit of computation gives the weights $5/9, 8/9, 5/9$:

$$\int_{-1}^{1} f(x)\,dx \approx \frac{5}{9}f(-\sqrt{3/5}) + \frac{8}{9}f(0) + \frac{5}{9}f(\sqrt{3/5}).$$

With some patience, it can be confirmed that this is exact for all polynomials of degree up to 5.

The values of t_i and A_i have been accurately computed for a large range of n values: for example the book [**?**] lists them for all $n \leq 512$.

Error Estimates and Convergence for Gaussian Quadrature

Error Estimates for Gaussian Quadrature

Gaussian quadrature has more reliable convergence than Romberg integration and polynomial collocation. In this respect:

Theorem 7. *For any function f continuous on an interval $[a, b]$, the succession of approximate integrals given by Gaussian quadrature with n points in the interval converges to the correct answer as $n \to \infty$.*

Nevertheless it is often more convenient and efficient to use composite Gaussian integration. That is, fix the number n that determines the number of points used in the Gaussian quadrature formula, divide the interval $[a, b]$ into m interval of length $h = (b - a)/m$, and use the same Gaussian integration formula on each interval, with the points x_i and weights A_i appropriately translated and scaled.

Romberg integration had the nice property that for any continuous function, each column converged. That is, each of the methods produced by a fixed number M of levels of extrapolation converges, even when they do not have the expected rate of convergence $O(h^{2M+2})$ due to lack of differentiability of the function.

However, the approximations on the diagonal of the table, got by doing all possible extrapolations from the function values at spacing $h = (b - a)/2^n$ are not known to converge. There was a similar lack of guaranteed convergence with successive collocating polynomials of increasing degree.

Convergence Rate for Gaussian Quadrature

The basis for the convergence rate is the error formula for any Gaussian quadrature formula:

Theorem 8. *The error in Gaussian Quadrature is given by:*

$$\int_a^b f(x)dx - \sum_{i=0}^{n} A_i f(x_i) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\int_a^b w(x)dx$$

where ξ is some point in the domain of integration and

$$w(x) = \prod_{i=0}^{n}(x - x_i)^2.$$

One can see that the quadrature rule cannot be exact for general polynomials of degree $2n+2$.

This approach can be shown to have errors that diminish at a rate $O(h^{2n+2})$ so long as the derivative $f^{(2n+2)}$ exists and is continuous on the whole integration interval. Note that in the case $n = 9$, doubling m and so doubling the amount of computation and halving h reduces the error by a factor of 2^{20} which is about one million! Thus only a few doubling will reduce errors to the lower limit imposed by rounding error.

What happens to this error when the length of the interval is reduced by a factor h? The points x_i move closer by this factor, and the x values involved in the integration of q become closer by the same factor, so each term in the product giving q is reduced by a factor h, which

reduces w by a factor of h^{2n+2}. The domain of integration is shorter by this factor, so the integral above of w is reduced by $h.h^{2n+2}$. However, in the composite rule for a fixed interval, the number of sub-intervals m increases by a factor $1/h$, so summing the errors for each sub-interval gives a total error proportional to h^{2n+2}, as claimed.

3.7.6 Romberg Integration

The Euler-Maclaurin Formula

In order to apply the full Richardson extrapolation scheme to the trapezoidal rule $T(f,h)$ one requires an expansion in terms of the discretisation parameter h of the form

$$T(f,h) = I + C_1 h + C_2 h^2 + \cdots$$

where the C_i only depend on f, a and b, but not on h. It turns out that such an expansion holds without the odd coefficients $C_1 = C_3 = C_5 = \cdots = 0$:

Theorem 9 (The Euler-Maclaurin Formula)**.** *If f and its derivatives of order up to $2r+2$ are continuous on the interval $[a,b]$ then*

$$T(f,h) = \int_a^b f(x)\,dx + C_2 h^2 + C_4 h^4 + \cdots + C_{2r} h^{2r} + O(h^{2r+2})$$

with $C_2 = [f'(b) - f'(a)]/12$ and in general $C_{2m} = c_{2m}[f^{(2m-1)}(b) - f^{(2m-1)}(a)]$.

The derivation is given in Section 7.7 of Kincaid and Cheney, [**?**] for a slightly different version of the result.

Aside: integrating periodic functions. For the sake of using Richardson extrapolation all that matters is that the C_i do not depend on h. However their specific form gives a very nice result about integrating periodic functions.

Theorem 10. *If f is periodic of period $b-a$, and it and its derivatives of order up to $2r+2$ are continuous on the interval $[a,b]$ then the composite trapezoid rule approximation has an error of size $O(h^{2r+2})$. In particular if f is infinitely differentiable, the error goes to zero faster than any power of h.*

Proof. Since $f(t+b-a) = f(t)$, it follows that $f'(t+b-a) = f\prime(t)$ and so on for all derivatives that exist. In particular $f'(b) = f'(a)$ and so on: all derivatives are equal at the two endpoints. The Euler-Maclaurin formula then has all the $C_i = 0$, leaving just the error term $O(h^{2r+2})$. □

This indicates that for periodic functions, the trapezoid rule is the best method to use. Since $f(a) = f(b)$ in this case this gives

$$I \approx h[f(a) + f(a+h) + \cdots + f(b-h)]$$

as a simple and very good formula for approximate integration.

Trapezoidal rules for hierarchical grids

Similar to the algorithm used for differentiation we will evaluate the trapezoidal rule $T(f,h)$ for $h = (b-a)/2^k$ and $k = 0, 1, 2, \ldots$. We see in the following table that

$$T(f,h/2) = \frac{T(f,h)}{2} + \frac{h}{2}\sum_{i=1}^{n} f(a + (i-0.5)h)$$

where $h = (b-a)/n$.

$$\begin{aligned}
T(f, b-a) &= \frac{b-a}{2}[f(a)+f(b)], \\
T(f,(b-a)/2) &= \frac{b-a}{4}\left[f(a)+2f\left(a+(b-a)/2\right)+f(b)\right] \\
&= \frac{1}{2}T(f,b-a)+\frac{b-a}{2}f\left(a+(b-a)/2\right), \\
T(f,(b-a)/4) &= \frac{b-a}{8}[f(a)+2f(a+(b-a)/4)+2f\left(a+(b-a)/2\right) \\
&\quad +2f\left(a+3(b-a)/4\right)+f(b)] \\
&= \frac{1}{2}T(f,(b-a)/2) \\
&\quad +\frac{b-a}{4}\left[f(a+(b-a)/4)+f(a+3(b-a)/4)\right] \\
&\vdots \\
T(f,(b-a)/(2p)) &= \frac{T(f,(b-a)/p)}{2}+\frac{b-a}{2p}\sum_{i=1}^{p} f\left(a+(i-0.5)\frac{b-a}{p}\right)
\end{aligned}$$

Romberg Integration = Richardson Extrapolation for the Trapozoidal Rule

As the Trapezoidal rule is second order (can be seen from the Euler-MacLaurin expansion) one has $I - T(f,h/2) \approx (I - T(f,h))/4$ so that one step of Richardson extrapolation gives

$$\frac{4T(f,h/2)-T(f,h)}{3}.$$

The Romberg scheme sets $R_{0,0} = T(f, b-a)$ and

$$R_{n,0} = T(f,h_n) = \frac{R_{n-1,0}}{2} + h_n \sum_{i=1}^{2^{n-1}} f(a+(i-0.5)h_{n-1})$$

with $h_n = (b-a)/2^n$. The remaining Romberg approximations $R_{n,m}$ are

$$R_{n,m} = \frac{4^m R_{n,m-1} - R_{n-1,m-1}}{4^m - 1}, \quad m \le n.$$

The Euler-MacLaurin formula guarantees that $I = R_{n,m} + O(h^{2m})$ if f is sufficiently smooth.

It can be checked by induction that each $R_{m,n}$ depends on values of f at spacing h_n: that is, the values at $a, a+h_n, \ldots$. This is true for $m = 0$, and if it is true for $m-1$, the above formula depends on values at spacing h_n for the first term and with spacing $h_{n-1} = 2h_n$ for the second. The latter are just a subset of the points with spacing h_n, so indeed these are the only ones involved.

As in the algorithm for differentiation, these steps can be arranged in an order that minimises the amount of computation involved. In this case, this means computing all the approximations possible for a given set of f values and checking the accuracy in order to decide whether it is worth continuing: in other words, computing all the approximations $R_{n,m}$ that are wanted for a given n and then deciding whether to continue with $n+1$.

One plausible (generally pessimistic) error estimate for any $R_{n,m}$ is the difference between it and the approximation $R_{n-1,m}$ which has the same order of accuracy $O(h^{2m+2})$ but twice the spacing of x values. This is only defined for $m < n$, but for a given n it is a reasonable strategy to use $|R_{n,m-1} - R_{n-1,m-1}|$ as an error estimate and stopping conditions for the iterations.

Romberg Integration Algorithm - First Version

One possible algorithm for the Romberg method is

Algorithm 1. *Romberg Method*

1: $h \leftarrow (b-a)$
2: $R_{0,0} \leftarrow \dfrac{(b-a)}{2}[f(a)+f(b)]$
3: ***for*** $n = 1$ *to* M ***do***
4: $\quad h \leftarrow h/2$
5: $\quad R_{n,0} \leftarrow \dfrac{1}{2}R_{n-1,0} + h\displaystyle\sum_{i=1}^{2^{n-1}} f(a+(2i-1)h)$
6: $\quad R_{n,m} \leftarrow \dfrac{4^m R_{n,m-1} - R_{n-1,m-1}}{4^m - 1}$
7: ***end for***
8: $I \leftarrow R_{n,n}$
9: ***if*** $|R_{n,n-1} - R_{n-1,n-1}| \leq \varepsilon$ ***then***
10: $\quad$ *stop*
11: ***end if***

There are some possible problems with this algorithm that suggest various modifications.

Firstly, as was seen with polynomial collocation, it can be dangerous to assume that a higher power of h in the error formula guarantees greater accuracy. That is true here too, because increasing m gives an error formula containing higher derivatives of f, and these derivatives might grow rapidly, or cease to exist after some point. Thus one variation is to impose an upper limit on m. A value that works well in practice is 6, so I will use it in the following discussion. One still continues on the higher n values as needed, and the error estimate becomes $|R_{n,6} - R_{n-1,6}|$ for $n \geq 7$.

Secondly, it is often more appropriate to specify a bound on the relative error rather than the absolute error as done above, or more precisely an error bound that grows with the limitation imposed by rounding error. As will be explained below, this error is roughly proportional to $B := \int_a^b |f(t)|\,dt$, suggesting that an error tolerance could be used of the form $\epsilon\tilde{B}$ for some approximation $\tilde{B}$ of B.

Combining these ideas suggests the following modifications to the above algorithm.

Romberg Integration Algorithm - Modification

1. Always go to at least $n = 6$ (64 intervals), so start by using the above algorithm for $n \leq 6$ without error testing.
2. Use the values of $f(x)$ computed in these steps to compute an approximation of B by the composite trapezoid rule. That is, $\tilde{B} = T(|f|, (b-a)/64)$
3. Continue using the above algorithm from $n = 7$ on, with the inner loop replaced by "**for** $m = 1$ **to** 6", the next line giving the best approximation for a given n replaced by "$I \leftarrow R_{n,6}$" and the stopping condition replaced by "**if** $|R_{n,6} - R_{n-1,6}| \leq \epsilon\tilde{B}$ **stop**".

Neither version of the algorithm is "definitive": they are given to illustrate some of the considerations that go into the design of a good algorithm for any numerical method.

The Effects of Rounding Error and Convergence of the Romberg Method

The Effects of Rounding Error

Contrary to what was seen for numerical differentiation, methods for numerical integration usually have good behaviour in the presence of errors due to rounding or inaccuracy in the evaluation of the values of f needed.

A proper analysis of rounding errors will be done in later sections of this course.

This is true in particular for the Romberg method due to the following result.

Theorem 11. *Each $R_{n,m}$ has the form*

$$R_{n,m} = \sum_{i=0}^{2^n} \alpha_i f(t_i)$$

with each $\alpha_i > 0$ and $\sum_{i=0}^{2^n} \alpha_i = 1$.

Some error bounds following from this are:

Theorem 12. *If the Romberg method is applied using approximations $\tilde{f}_i$ of the values $f(x)$, and the absolute error of each approximation is bounded by some constant M, the error introduced into each $R_{n,m}$ is at most $(b-a)M$.*

More generally, if $|f(t_i) - \tilde{f}_i| \leq M_i$ there is the error bound

$$|I - R_{n,m}| \leq \sum_{i=0}^{2^n} \alpha_i M_i.$$

This can be applied to the situation where the errors are due to rounding error in the evaluation of f. Typically the effects of floating point arithmetic make these errors proportional to the magnitude of $f(x)$ giving bounds $M_i = k|f(t_i)|$. (k is typically a fixed multiple of machine epsilon).

Thus for $R_{n,0}$, given by the trapezoid rule, one gets errors roughly bounded by

$$\frac{h}{2}\left[k|f(a)| + 2k|f(a+h)| + \cdots + k|f(b)|\right] \approx k\int_a^b |f(t)|\,dt$$

The effect of the successive extrapolation steps will be roughly to amplify these errors by the factor $(4^m+1)/(4^m-1)$ which are $5/3$, $17/15$, $65/63$ so quickly become very close to 1. Regardless of this detail, the formula above shows that one expects the error to be bounded roughly by a multiple of $\int_a^b |f(t)|\,dt$ as suggested earlier.

Convergence of the Romberg Method

The Euler-Maclaurin formula gives some guarantees of convergence for sufficiently differentiable functions. Specifically, if f has $2m+2$ continuous derivatives, the errors for the approximation in the column corresponding to m go to zero as $n \to \infty$ (which gives $h \to 0$).

That is

$$\lim_{n\to\infty} R_{n,m} = I. \tag{3.22}$$

It is *not* guaranteed that the sequence $R_{n,n}$ of "best" approximations for each n converge to the integral, even though they often will, and faster than the above sequence.

The utility of the above result is rather restricted, since one often does not know much about the differentiability of f. Fortunately, this convergence still occurs even when the Euler-Maclaurin formula does not apply.

Theorem 13. *The convergence property of (3.22) holds so long as f is continuous on the domain of integration $[a,b]$.*

Proof. For $m = 0$ this is convergence of the trapezoid rule. The result is true in this case by noting that the trapezoid is a Riemann sum, with the ends of the intervals at the points

$$a, a + h/2, a + 3h/2, \ldots, b - h/2, b$$

to give the lengths $h/2, h, h, \ldots, h, h/2$. Then the convergence of Riemann sums for any continuous function gives the result for $m = 0$. The result for successive m values can be shown by induction. If the result is true for $m - 1$ then

$$\begin{aligned}
\lim_{n \to \infty} R_{n,m} &= \lim_{n \to \infty} \frac{4^m R_{n,m-1} - R_{n-1,m-1}}{4^m - 1} \\
&= \frac{4^m}{4^m - 1} \lim_{n \to \infty} R_{n,m-1} - \frac{1}{4^m - 1} \lim_{n \to \infty} R_{n-1,m-1} \\
&= \frac{4^m}{4^m - 1} I - \frac{1}{4^m - 1} I \\
&= I.
\end{aligned}$$

□

The fact that convergence is guaranteed "down columns", but not "down the diagonal" is another reason why a cautious approach fixes the maximum m as in the second algorithm above, so taking its approximations from one column.

3.7.7 Comparisons of the Accuracy of Various Numerical Integration Methods

Composite Quadrature rules For the interval $[a, b]$ - Reminder

Composite quadrature: $I = \int_a^b f(x)\,dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx$, $(h = x_{i+1} - x_i)$;

- Riemann sum :
  $I = \sum_{i=0}^{n-1} f(x_i)h + O(h).$

- Composite midpoint rule:
  $I = \sum_{i=0}^{n-1} f(m_i)h + \frac{1}{24}\left[f'(b) - f'(a)\right]h^2 + O(h^4).$

- Composite trapezoid rule:
  $I = T(f, n) - \frac{1}{12}\left[f'(b) - f'(a)\right]h^2 + O(h^4).$

- Composite Simpson's rule:
  $I = S(f, n) - \frac{1}{180}\left[f'''(b) - f'''(a)\right]h^4 + O(h^6).$

Error Terms - Reminder

Can write the Taylor series explicitly with higher order terms, leading to

$$\int_{x_i}^{x_{i+1}} f(x)\,dx - T(f, 1) = -\frac{1}{12} f''(m_i)h^3 - \frac{1}{480} f^{(4)}(m_i)h^5 + O(h^7)$$

$$\int_a^b f(x)\,dx - T(f, n) = -\frac{1}{12}\left[f'(b) - f'(a)\right]h^2 + O(h^4).$$

Can absorb the higher order terms into the h^2 term by truncating the original Taylor series

$$\int_{x_i}^{x_{i+1}} f(x)\,dx - T(f, 1) = -\frac{1}{12} f''(\xi_i)h^2, \qquad \xi_i \in (x_i, x_{i+1})$$

$$\begin{aligned}\int_a^b f(x)\,dx - T(f,n) &= -\frac{1}{12}\sum_{i=0}^{n-1} f''(\xi_i)h^3 \\ &= -\frac{1}{12}nf''(\xi)h^3, \qquad \xi \in (a,b) \\ &= -\frac{(b-a)}{12}f''(\xi)h^2.\end{aligned}$$

Comparisons of Various Numerical Integration Methods

Consider the approximate evaluation of

$$I = \int_1^{1.5} e^{-x^2}\,dx \approx 0.1093643$$

by the various methods seen so far. Cost comparisons are best done by counting the number of evaluations of f required, and errors of "zero" only mean accuracy to all seven decimal places given.

n	Evaluations	Trapezoid Rule	Simpson's Rule
0	2	-.0089554	
1	3	-.0021984	.0000539
2	5	-.0005471	.0000033
3	9	-.0001366	.0000002

Table 3.1: Errors for trapezoidal rule and Simpson's rule

Evaluations	1	2	3	4
Error	.0045586	-.0000360	0.00000001	0

Table 3.2: Errors in Gaussian Quadrature

One can see the superior performance of Gaussian quadrature. Also, recall that Simpson's rule is part of the Romberg rule and one can see that further extrapolation gives zero quadrature error up to 7 digits considered here.

3.8 Numerical Differentiation

3.8.1 Introduction

Motivation

- next two sections on numerical differentiation and integraton mirror the core topics of calculus
- approximations of derivatives are used in the solution of nonlinear equations, ordinary and partial differential equations and in data analysis
- the determination of derivatives from measured data is the prototypical example of an ill-posed problem
- objective: determine $f'(x)$ from finite number of values $f(x_i)$
- fundamental idea from definition of derivative as limit of finite differences:

$$f'(x) \simeq \frac{f(x+h)-f(x)}{h}.$$

- other fundamental idea: take derivative of interpolating function

Truncation Error

To assess the accuracy, expand the right hand side above using the first order Taylor polynomial with error term:

$$\begin{aligned}\frac{f(x+h)-f(x)}{h} &= \frac{[f(x)+hf'(x)+f''(\xi)h^2/2]-f(x)}{h} \\ &= f'(x)+\frac{f''(\xi)}{2}h \\ &\quad \text{for some } \xi \text{ between } x \text{ and } x+h.\end{aligned}$$

For bounded f'' one has for $h \to 0$ (i.e. small h):

$$e_h(x) := f'(x) - \frac{f(x+h)-f(x)}{h} = O(h) \to 0.$$

This error is called the *truncation error* [as the Taylor series for f has been truncated after one term to get it], and must be considered in conjunction with the rounding error in evaluating this approximation.

Example 12 (The derivative of $x\sin(x)$ at $x=1$)**.** *Consider the function*

$$f(x) = x\sin(x)$$

where $f'(x) = \sin(x) + x\cos(x)$*. Figure 3.15 shows a log-log plot of the absolute value of the error*

$$|e_h(x)| = \left| f'(x) - \frac{f(x+h)-f(x)}{h} \right|$$

for different values of h *when* $x = 1$*. Observe that as* h *is reduced we see the* $O(h)$ *convergence rate until* h *is approximately* 10^{-7} *when round-off becomes a problem. This counter-intuitive growth of the error is actually a consequence of the ill-posedness of differentiation as the number of operations used to determine derivatives does not depend on* h*.*

One reason why the above approximation is not good enough is that we want to use approximate derivatives in situations like piecewise cubic approximation where the other errors are $O(h^4)$.

Another reason is that we cannot always increase accuracy to the level needed by simply using smaller h values, due to rounding error and catastrophic cancellation.

The rounding error in the numerator is bounded by $\epsilon(|f(x)| + |f(x+h)|)$ where ϵ is machine epsilon. Dividing by h, even ignoring any further error in this operation, will give an absolute rounding error bounded by

$$\frac{|f(x)| + |f(x+h)|}{h}\epsilon \simeq \frac{2|f(x)|}{h}\epsilon.$$

Adding this to the truncation error gives a total error bounded approximately by

$$\frac{|f''(x)|}{2}h + \frac{2|f(x)|\epsilon}{h}.$$

The problem is now clear: reducing one part of the error (by reducing or increasing h) increases the other term. In fact this approximate error bound has a minimum of $\sqrt{|f''(x)f(x)|}\sqrt{\epsilon}$ when $h = \sqrt{4|f(x)/f''(x)|}\sqrt{\epsilon}$. For example with $\epsilon \simeq 10^{-16}$ as with MATLAB, and f and f'' of order one, the square root gives a best possible error of order 10^{-8}. In general the rough rule is that half the significant digits are lost.

This can be overcome, and with sufficient effort, approximations can be found with errors roughly proportional to ϵ, which is the best one could hope for. To see the path to such improvements, consider the *centred difference approximation*

$$f'(x) \simeq \frac{f(x+h)-f(x-h)}{2h}$$

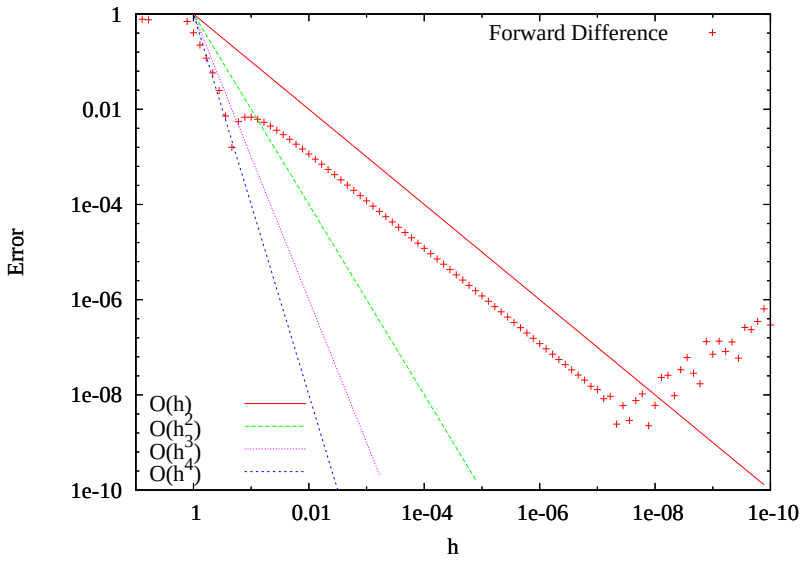


Figure 3.15: Example of forward difference approximation to the derivative

which is the slope of the secant line to the graph of f through two points at equal distances to either side of x.

Using the second order Taylor polynomial approximations both each of $f(x \pm h)$ with error terms gives

$$\begin{aligned} f'(x) - \frac{f(x+h) - f(x-h)}{2h} &= -\frac{h^2}{12}[f'''(\xi_1) + f'''(\xi_2)] \\ &\simeq -\frac{f'''(x)}{6}h^2, \\ &= O(h^2) \end{aligned}$$

To get even more accurate formulas, one approach is to continue with Taylor polynomials at several different points near x, computing linear combinations that cancel out all higher order derivatives so as to express $f'(x)$ in terms of those nearby function values.

3.8.2 Finite Difference Approximations

First Order Approximations of First Derivative

Consider the two Taylor series expansions

$$f(x+h) = f(x) + h\,f'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O\big(h^4\big)\,, \tag{3.23}$$

and

$$f(x-h) = f(x) - h\,f'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O\big(h^4\big)\,. \tag{3.24}$$

Using Equation (3.23) we get the following *forward difference* approximation to the derivative

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)\,.$$

Equation (3.24) gives the *backward difference* approximation to the derivative

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h)\,.$$

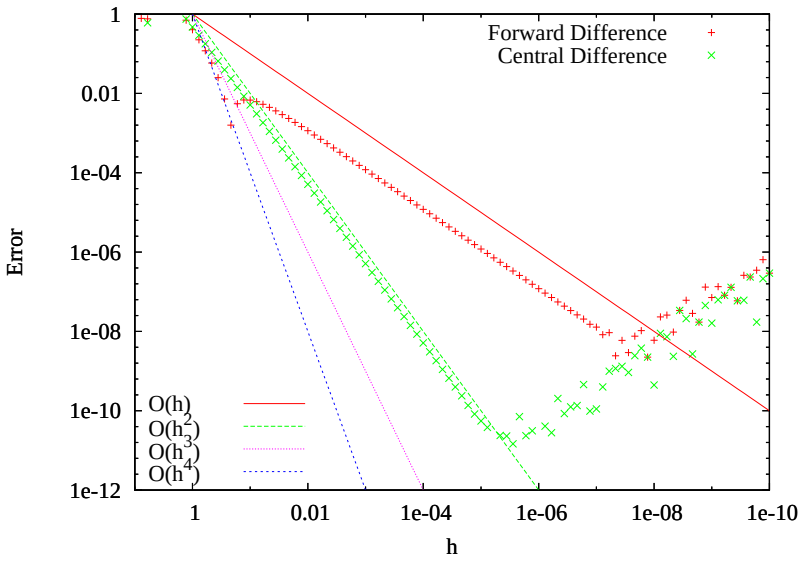


Figure 3.16: Example of the central difference approximation to the derivative

Second Order Approximations of First Derivative

When subtracting (3.24) from (3.23) one sees that the terms with h^2 cancel. Thus the resulting central difference approximation of the first derivative is second order accurate:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2\,h} - O\big(h^2\big)\,.$$

Example 13 (The derivative of $x\sin(x)$ at $x = 1$)**.** *Figure 3.16 shows a log-log plot of the error for the central difference approximation.*

Second Derivatives

To find an approximation of the second derivatives further expansion of the Taylor series is required,

$$f(x+h) = f(x) + h\,f'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f''''(x) + O\big(h^5\big)\,,$$

and

$$f(x-h) = f(x) - h\,f'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f''''(x) + O\big(h^5\big)\,.$$

Add the equations to get

$$f(x+h) + f(x-h) = 2\,f(x) + h^2\,f''(x) + \frac{h^4}{12}f''''(x) + O\big(h^5\big)$$

So

$$f''(x) = \frac{f(x+h) - 2\,f(x) + f(x-h)}{h^2} - \frac{h^2\,f''''(x)}{12} + O\big(h^3\big)\,.$$

Higher order formulas can be obtained. However there are several other more convenient methods, including the use of collocation polynomials, the *method of undetermined coefficients*, and *Richardson extrapolation*, which will be considered in turn.

3.8.3 The Method of Undetermined Coefficients

The Method of Undetermined Coefficients

Interpolate f at points x_0, x_1, x_2, $\cdots$, x_n by a polynomial of degree n

$$p_n(x) = f(x_0)l_0(x) + f(x_1)l_1(x) + \cdots + f(x_n)l_n(x)$$

where $l_i(x_j) = \delta_{i,j}$. The $l_i(x)$ are the Lagrangian or cardinal polynomials:

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

The approximation to the derivative at some $x = b$ is

$$f'(x)|_{x=b} \approx p_n'(x)|_{x=b} = a_0 f(x_0) + \cdots + a_n f(x_n)$$

where $a_i = l_i'(b)$. The examples already seen for approximations of the first derivative all have the form

$$f'(x_0) \simeq a_1 f(x_1) + \cdots + a_k f(x_k) \tag{3.25}$$

where the points x_i are known (such as $x_0 - h, x_0, x_0 + h$) and the a_i follow when the collocation polynomial is differentiated. In fact $a_i = l_i'(x_0)$ where l_i is the polynomial of degree $k-1$ that has $l_i(x_i) = 1$ and vanishes at the other collocation points x_i.

Determination of the coefficients a_i

While one has an explicit formula $a_i = l_i'(b)$ the computation of these values may be cumbersome. Note, however, that the formula is exact for all n-th degree polynomials $p_n(x)$ for which

$$p_n'(x)|_{x=b} = a_0 p_n(x_0) + a_1 p_n(x_1) + \cdots + a_k p_n(x_k).$$

Thus for any monomial x^k with $k = 0, \ldots, n$ one has

$$k b^{k-1} = a_0 x_0^k + \cdots + a_n x_n^k.$$

This is just a system of $n+1$ linear equations. (Question: when do we have a unique solution?)

The procedure can be simplified by a couple of changes. Firstly, express the polynomials in terms of powers of $x - x_0$, so that

$$p_n(x) = b_0 + b_1(x - x_0) + \cdots + b_{k-1}(x - x_0)^{k-1}.$$

Secondly, apply the condition separately for each of the k simple powers: that is, require that the approximation in (3.25) be exact for each case $p_m(x) = (x - x_0)^m, 0 \leq m \leq k-1$. Each of these conditions will give one of the equations that can then be solved to get the required coefficients.

The most important case is that of equally spaced points, on one or both sides of x_0: $x_i = x_0 + ih$, with the points used in the approximation being those with $L \leq i \leq M$. Then as seen several times above, the coefficients will be proportional to $1/h$ and it is convenient to determine the approximation formula in the form

$$\begin{aligned} f'(x_0) &\simeq \frac{1}{h}(a_L f(x_L) + \cdots + a_M f(x_M)) \\ &= \frac{1}{h}(a_L f(x_0 + Lh) + \cdots + a_M f(x_0 + Mh)). \end{aligned}$$

Example 14 (Method of Undetermined Coefficients)**.** *Consider the approximation*

$$f'(x)|_{x=x_0} \approx a_0 f(x_0) + a_1 f(x_1) + a_2 f(x_2) + a_3 f(x_3)$$

where $x_i = x_0 + ih$ for some grid size h.

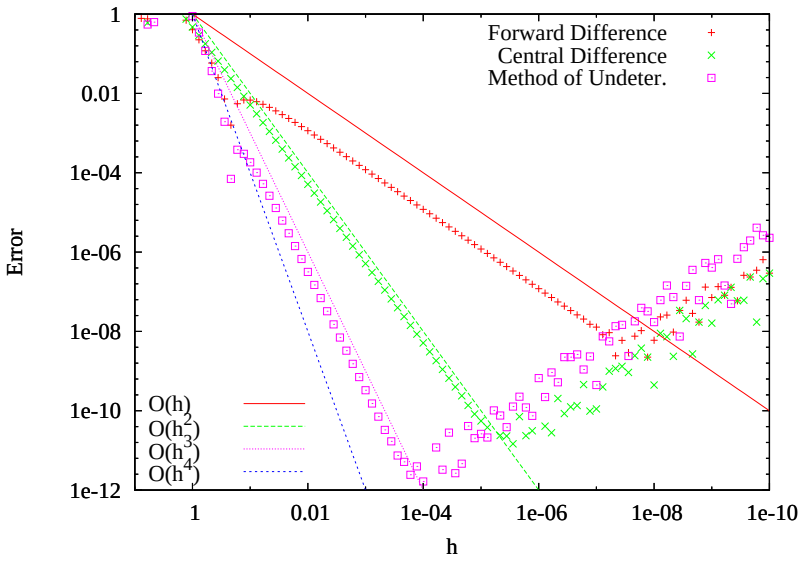


Figure 3.17: Example of the approximation found using the method of undetermined coefficients

This approximation must be exact for the following polynomials

$$\begin{aligned} f_0(x) &= 1, \\ f_1(x) &= x - x_0, \\ f_2(x) &= (x - x_0)^2, \\ f_3(x) &= (x - x_0)^3. \end{aligned}$$

Note that any third order polynomial can be written as a linear combination of these four polynomials.

Using the fact that

$$f_j'(x)\big|_{x=x_0} = a_0 f_j(x_0) + a_1 f_j(x_1) + a_2 f_j(x_2) + a_3 f_j(x_3)$$

for $0 \leq j \leq 3$ we get the following system of equations

$$\begin{aligned} 0 &= a_0 + a_1 + a_2 + a_3 \quad \text{from } f_0(x), \\ 1 &= ha_1 + 2ha_2 + 3ha_3 \quad \text{from } f_1(x), \\ 0 &= h^2 a_1 + (2h)^2 a_2 + (3h)^2 a_3 \quad \text{from } f_2(x), \\ 0 &= h^3 a_1 + (2h)^3 a_2 + (3h)^3 a_3 \quad \text{from } f_3(x), \end{aligned}$$

Solve to find

$$a_0 = \frac{-11}{6h}, \quad a_1 = \frac{3}{h}, \quad a_2 = \frac{-3}{2h}, \quad a_3 = \frac{1}{3h}.$$

NB This is an example of a so-called stencil or finite difference scheme and is identical with what one gets using Taylor series approximation.

Example 15 (The derivative of $x\sin(x)$ at $x = 1$)**.** *Figure 3.17 shows a log-log plot of the error for the coefficients found in Example 14.*

Analogous to the error formulas for polynomial collocation, there are results that say how closely the derivatives of a collocation polynomial approximate the derivatives of the original function. However there is an easier approach when the points are equally spaced, giving formulas that show that the error will vary roughly as a power of h when h is small.

To better understand how this works lets look at another example.

Example 16 (Method of Undetermined Coefficients With Taylor Series)**.** *Consider the approximation*

$$f'(x)|_{x=x_0} \approx a_0 f(x_0) + a_1 f(x_1) + a_2 f(x_2) \tag{3.26}$$

where $x_i = x_0 + ih$ for some grid size h.

From Taylor series approximation we have

$$f(x_1) = f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{3!} f'''(\xi_1)$$

and

$$f(x_2) = f(x_0 + 2h) = f(x_0) + 2hf'(x_0) + \frac{4h^2}{2} f''(x_0) + \frac{8h^3}{3!} f'''(\xi_2),$$

where $x_0 \le \xi_1 \le x_1$ and $x_0 \le \xi_2 \le x_2$. Substituting these expansions into Equation (3.26) gives

$$\begin{aligned} f'(x)|_{x=x_0} &\approx a_0 f(x_0) \\ &\quad + a_1 \left(f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{3!} f'''(\xi_1) \right) \\ &\quad + a_2 \left(f(x_0) + 2hf'(x_0) + \frac{4h^2}{2} f''(x_0) + \frac{8h^3}{3!} f'''(\xi_2) \right) \\ &= (a_0 + a_1 + a_2) f(x_0) + h (a_1 + 2a_2) f'(x_0) + \frac{h^2}{2} (a_1 + 4a_2) f''(x_0) \\ &\quad + \frac{h^3}{3!} (a_1 f'''(\xi_1) + 8a_2 f'''(\xi_2)) . \end{aligned}$$

We require the approximation to be exact when f is a polynomial of degree 2. If f is a polynomial of degree 2 then $f'''(x) = 0$, so the last term drops out, and after collecting like terms we get the following system;

$$\begin{aligned} a_0 + a_1 + a_2 &= 0 \\ a_1 + 2a_2 &= h^{-1} \\ a_1 + 4a_2 &= 0. \end{aligned}$$

Consequently $a_0 = -3/(2h)$, $a_1 = 2/h$ and $a_2 = -1/(2h)$.

Lets come back to the question of the error term

$$f'(x)|_{x=x_0} - \frac{1}{2h} [-3f(x_0) + 4f(x_1) - f(x_2)] .$$

If we replace $f(x_1)$ and $f(x_2)$ by their Taylor series expansion, then based on the way we calculated the coefficients a_0, a_1 and a_2, we are left with

$$f'(x)|_{x=x_0} - \frac{1}{2h} [-3f(x_0) + 4f(x_1) - f(x_2)] = \frac{-h^2}{3} (4f'''(\xi_1) - f'''(\xi_2)) .$$

So we have an $O(h^2)$ approximation.

The pattern is clear: using data from k points x_i gives an approximation of the derivative that is $O(h^{k-1})$ as $h \to 0$. Another way to derive the error formulas is to use the degree k Taylor polynomial approximation with its error in the simpler form

$$f(x_i) = f(x_0 + ih) = f(x_0) + ihf'(x_0) + \cdots + i^k h^k f^{(k)}(x_0) + O(h^k).$$

The terms $O(h^k)$ will all be seen to be negligible (except when $f^{(k)}(x_0) = 0$), so in most practical situations, one can just use Taylor polynomials themselves and ignore the details of their error terms.

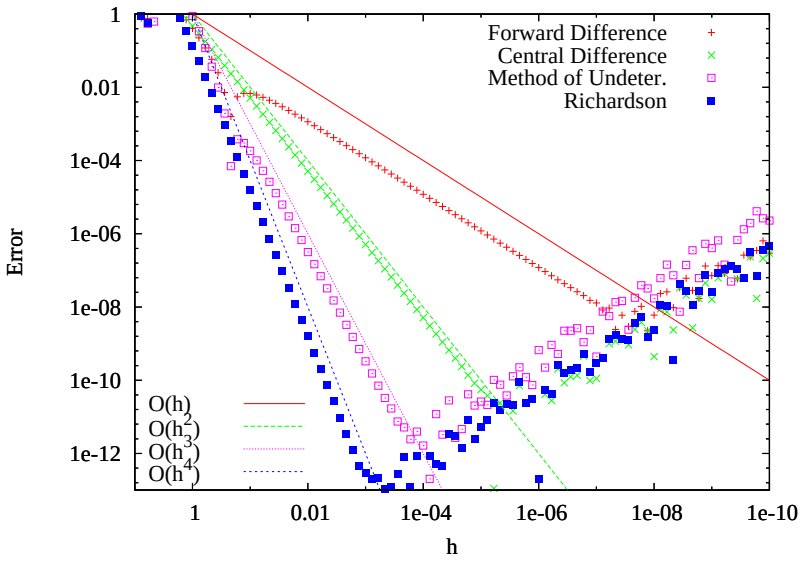


Figure 3.18: Example of a fourth order approximation

3.8.4 Richardson Extrapolation

Richardson Extrapolation applied to the Central Difference Formulas

The error formulas given above can now be used to derive more accurate approximations. The central difference formulas may be written as

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f'''(x)h^2}{6} + O(h^4).$$

Or

$$f'(x) = \phi(h) + a_2h^2 + a_4h^4 + O(h^6), \tag{3.27}$$

$$f'(x) = \phi(2h) + 4a_2h^2 + 16a_4h^4 + O(h^6). \tag{3.28}$$

Subtracting Equation (3.28) from 4 × Equation (3.27) gives

$$f'(x) = \frac{1}{3}[4\phi(h) - \phi(2h) - 12a_4h^4] + O(h^6).$$

Example 17 (The derivative of $x\sin(x)$ at $x = 1$)**.** *Figure 3.18 shows a log-log plot of the error for the fourth order approximation found using Richardson extrapolation.*

Improving Accuracy with Extrapolation

This strategy is Richardson extrapolation, and in general can be applied whenever an unknown quantity ϕ_0 is approximated by a function $\phi(h)$ that can be evaluated, with error known to have an approximately power law behaviour

$$\phi_0 = \phi(h) + Ch^p + o(h^p).$$

Here p must be known, but C is typically unknown. Repeating the approach above gives

$$\phi_0 = \phi(2h) + 2^pCh^p + o(h^p)$$

and eliminating the "h^p" term and solving for ϕ_0 gives

$$\phi_0 = \frac{2^p\phi(h) - \phi(2h)}{2^p - 1} + o(h^p). \tag{3.29}$$

The process of Richardson extrapolation can be repeated to get successively more accurate formulas. For example, one can take the approximation derived above

$$C_4(h) := \frac{4}{3}\frac{f(x+h)-f(x-h)}{2h} - \frac{1}{3}\frac{f(x+2h)-f(x-2h)}{4h},$$

compute its error formula by the Taylor polynomial method, identify the *order* p of this approximation, and use (3.29) to get a new, still more accurate formula.

There are several ways to rearrange this process that reduce the work involved and make it more automatic: in the end, computation of the Taylor series expansions can be eliminated for any particular case, once the general pattern is known. Using a Taylor polynomial with more terms

$$\begin{aligned} f(x_0 \pm h) &= f(x_0) \pm hf'(x_0) + h^2 f''(x_0)/2 \pm h^3 f'''(x_0) \\ &\quad + \cdots + h^{2k} f^{(2k)}(x_0)/(2k)! \\ &\quad \pm h^{2k+1} f^{(2k+1)}(x_0)/(2k+1)! + o(h^{2k+1}) \end{aligned}$$

in the centred difference formula gives

$$\begin{aligned} \frac{f(x_0+h)-f(x_0-h)}{2h} &= f'(x_0) + h^2 f'''(x_0)/6 + h^4 f^{(5)}(x_0)/5! + \cdots \\ &\quad + h^{2k} f^{(2k+1)}(x_0)/(2k+1)! + o(h^{2k}) \end{aligned}$$

with only even powers of h occurring.

When Richardson extrapolation is applied, the first term, of $O(h^2)$ is eliminated so that the error in the new approximation $C_4(h)$ above is of the form Ch^4: the extrapolation can continue with $p = 4$. At each subsequent stage, one eliminates the lowest remaining power of h from the error formula for the new method.

It is worth putting this in a more general form, as its most important use will come later, and it should be emphasised that only the powers of h count, not the details of where the error formula comes from. Thus consider an approximation method $\phi(h)$ for a ϕ_0 as before, with

$$\phi_0 = \phi(h) + a_2 h^2 + a_4 h^4 + a_6 h^6 + \cdots$$

This has $p = 2$ and leads to

$$\phi_0 = \frac{4}{3}\phi(h) - \frac{1}{3}\phi(2h) - 4a_4 h^4 - 20a_6 h^6 + \cdots$$

from which the next extrapolation can be done with $p = 4$, and subsequent ones with $p = 6, 8, \ldots$.

Let

$$R(n, 0) = \phi(2^n h)$$

so that the initial Richardson extrapolation step gives

$$R(n, 1) = \frac{4R(n, 0) - R(n+1, 0)}{4-1}$$

for the approximation using points at intervals of $2^n h$. At each step $k = 1, 2, \ldots$, the quantity $R(n, k-1)$ will approximate ϕ_0 with an error term $O(h^{2k})$:

$$R(n, k-1) = \phi + O(h^{2k}).$$

Thus the Richardson extrapolation to eliminate this error term gives

$$R(n, k) = \frac{4^k R(n, k-1) - R(n+1, k-1)}{4^k - 1} \tag{3.30}$$

with $R(0,k)$ the most accurate approximation of all those achieved for k levels of extrapolation, while the other values $R(n,k)$ are useful mainly for doing the next extrapolation. This sequence of approximations contains a natural error estimate. For each level k of extrapolations, $R(0,k)$ should be substantially more accurate than either $R(1,k)$ or the best approximation at the previous level, $R(0,k-1)$. Thus the magnitude of either of the differences $R(0,k)-R(1,k)$ or $R(0,k)-R(1,k)$ is a plausible error estimate for $R(0,k)$.

Hence Richardson extrapolation can be applied for just as many levels as needed to achieve a specified level of accuracy, improving efficiency.

To reduce the amount of computation needed in this iterative process, it is also useful to work out exactly which of the intermediate values are needed in order to compute a particular $R(0,k)$.

The answer is that $R(0,k)$ is computed from $R(0,k-1)$ and $R(1,k-1)$, which in turn require $R(0,k-2)$, $R(1,k-2)$ and $R(2,k-2)$, and so on back to the values $R(0,0)\dots R(k,0)$. The intermediate values needed are exactly $R(m,j)$ with $m+j\le k$, giving the triangular array of results

$$\begin{array}{cccccc} R(0,0) & R(0,1) & R(0,2) & \cdots & \cdots & R(0,k) \\ R(1,0) & R(1,1) & R(1,2) & \cdots & R(1,k-1) & \\ \vdots & \vdots & \vdots & \iddots & & \\ R(k-2,0) & R(k-2,1) & R(k-2,2) & & & \\ R(k-1,0) & R(k-1,1) & & & & \\ R(k,0) & & & & & \end{array}$$

Here each element is given by Equation (3.30) in terms of the value to its left and the one immediately above that, and the approximations that will be used as the final values are the ones at the end of each row. What is more, the main computational cost is in evaluation of the members of the first column.

Thus, the rows can be computed from the top down, with the elements of each row computed from left to right, and once a row has been started, it is worth finishing it and using the last member as the new approximation of ϕ_0.

An error estimate such as $|R(0,k)-R(0,k-1)|$ can then be used to decide whether to continue to the next row.

Richardson Extrapolation Example

Example 18 (The derivative of $\sin x$ at $x=1$, $h=0.25$)**.**

$$R(0,0)=\frac{\sin(1.25)-\sin(0.75)}{0.5}=0.5346917$$

$$R(1,0)=\frac{\sin(1.5)-\sin(0.5)}{1}=0.5180694$$

$$R(0,1)=\frac{4R(0,0)-R(1,0)}{4-1}=0.5402325$$

$$R(2,0)=\frac{\sin(2)-\sin(0)}{2}=0.4546487$$

$$R(1,1)=\frac{4R(1,0)-R(2,0)}{4-1}=0.5392097$$

$$R(0,2)=\frac{4^2R(0,1)-R(1,1)}{4^2-1}=0.5403007$$

The true value is $\cos(1)=0.540302306$.

A PYTHON program of Richardson Extrapolation is posted on `wattle`. The output of calling the function with $f=\sin$, $x=1$, $h=0.25$,$M=10$ and $\epsilon=1.0e-06$ is:

```
[ 0.53469172  0.54023248  0.54030066  0.54030222  0.54030229]
[ 0.51806945  0.53920969  0.54020263  0.54028262  0.        ]
[ 0.45464871  0.5243157   0.53516305  0.          0.        ]
[ 0.24564775  0.36160551  0.          0.          0.        ]
[-0.10222553  0.          0.          0.          0.        ]
```

Once again, the true value is $\cos(1) = 0.540302306$.

Richardson Extrapolation Algorithm

Consider the case of approximating $f'(x)$ by combining the centred difference formula with Richardson extrapolation, seeking an estimated error of at most ϵ, with a maximum of M levels of extrapolation to avoid infinite loops. This leads to the algorithm, with L as the final result.

Algorithm 2. *Richardson Extrapolation*

$R(0,0) \leftarrow \frac{f(x+h)-f(x-h)}{2h}$.
for $n = 1$ *to* M ***do***
  $h \leftarrow 2h$.
  $R(n,0) \leftarrow \frac{f(x+h)-f(x-h)}{2h}$.
  for $k = 1$ *to* n ***do***
    $R(n-k,k) \leftarrow \frac{4^k R(n-k,k-1)-R(n-k+1,k-1)}{4^k-1}$
  end for
  $L \leftarrow R(0,n)$.
  if $|R(0,k) - R(1,k-1)| \leq \varepsilon$ ***then***
    stop
  end if
end for