Numerical Analysis II

M.R. O'Donohoe

General references:
S.D. Conte & C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach*, Third edition, 1981. McGraw-Hill.

L.F. Shampine, R.C. Allen, Jr & S. Pruess, *Fundamentals of Numerical Computing*, 1997. Wiley.

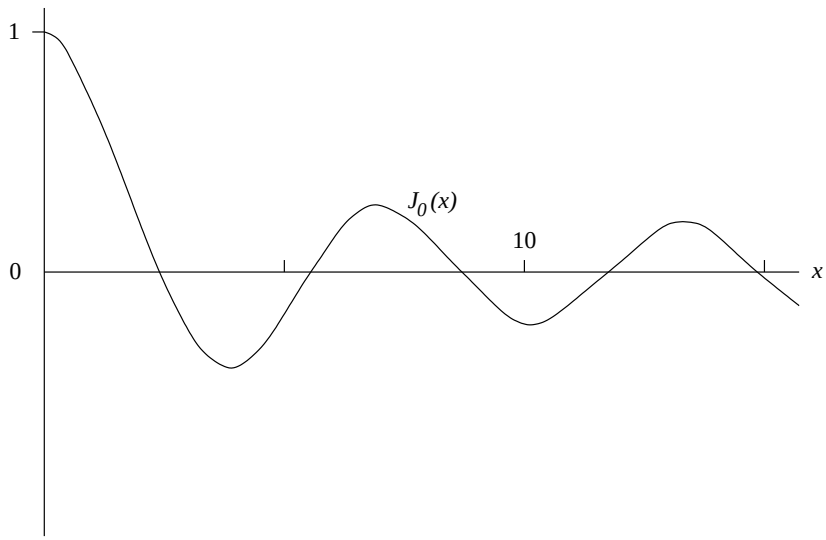
This course is concerned with the same problems as Numerical Analysis I, but the mathematical content is greater. In the interests of simplicity and brevity, the use of complex variables is avoided unless necessary. However, it should be noted that many real variable methods can be generalized to complex cases, although these are not always mentioned here. Familiarity with the notes for Numerical Analysis I is assumed.

1. Elementary Approximation Theory

Approximation theory is a major field in its own right, but it is also central to numerical analysis. The problems treated in this section are: (i) the approximation of a function of a real variable by a simpler function, and (ii) the approximation of a set of data points by a function of a real variable.

1.1 Preliminaries

To obtain a numerical approximation to a function, we first require accurate function values. These may be obtained in a variety of ways: Bessel's function $J_0(x)$ is used here as an example.



* A differential equation. $J_0(x)$ is defined by

$$xy'' + y' + xy = 0; \qquad y(0) = 1, y'(0) = 0.$$

This requires numerical solution of the differential equation – see Section 5.

* A Taylor series. The series

$$J_0(x) = 1 - \frac{x^2}{4} + \frac{x^4}{64} - \ldots + \frac{(-1)^k(\frac{1}{2}x)^{2k}}{(k!)^2} + \ldots$$

converges everywhere. However, the expansion is only useful near $x = 0$ and is particularly bad for $x \gg 0$.

* An asymptotic series.

$$J_0(x) \sim \sqrt{\frac{2}{\pi x}}\Big\{\cos(x-\frac{\pi}{4}).(1-\frac{9}{128x^2}+\ldots)+\sin(x-\frac{\pi}{4}).(\frac{1}{8x}-\frac{75}{1024x^3}+\ldots)\Big\}$$

  is useful for x large†.

* Other expansions about a point, e.g. infinite products, continued fractions.

* An integral representation.

$$J_0(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin\theta)\, d\theta$$

  is useful for small x but the integrand becomes highly oscillatory for large x.

* Chebyshev series – see Section 1.4.4.

1.1.1 Polynomial approximation

Consider the problem of approximating the value of some function $f(x)$ over some finite interval $[a, b]$. Since the operations of addition, subtraction and multiplication are the simplest and fastest real arithmetic operations available on a computer, it is sensible to consider a polynomial of degree n

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \tag{1.1}$$

as a possible approximating function.

Theorem 1.1: Weierstrass' Approximation Theorem (1885). *Given any function $f(x)$ that is continuous over a finite interval $x \in [a, b]$, and an arbitrary tolerance $\varepsilon > 0$, then there exists a finite n and a polynomial $P_n(x)$ such that*

$$|f(x) - P_n(x)| \leq \varepsilon.$$

Proof *omitted.*

In other words, any continuous function can be approximated over any finite interval to arbitrary accuracy by a polynomial. However, this theorem does not imply that all polynomial approximation methods work, nor does it tell us how to find the best set of polynomials for approximating a given function.

1.1.2 Taylor series

Taylor series are one way of obtaining polynomials of the form (1.1) and are well understood. Let

$$f(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n + \ldots \tag{1.2}$$

be the Taylor series expansion of some function $f(x)$ about $x = 0$, where $c_n = f^{(n)}(0)/n!$. It is necessary that $f(x)$ be analytic at $x = 0$, i.e. all the necessary derivatives exist for the coefficients $\{c_n\}$ to be found. By the ratio test, the series (1.2) converges for $|x| < r$ where

$$r = \lim_{n\to\infty}\left|\frac{c_n}{c_{n+1}}\right|.$$

However, the sequence of approximating functions $1,\ x,\ x^2,\ \ldots x^n,\ \ldots$ is unsatisfactory in general because the powers of x become closer to linear dependence as n increases.

† However, note that it is difficult to evaluate sin and cos for large arguments which contain rounding error.

Taylor series are only useful for analytic functions, whereas Weierstrass' theorem suggests that all continuous functions could be approximated by polynomials. In practice Taylor series are only used for approximation in special cases.

1.2 Interpolation

We consider the problem of finding a polynomial of degree n that is equal to a given function $f(x)$ at a prescribed set of points $x_0, x_1, \ldots x_n$. These interpolation points are sometimes called *abscissae*. The theory of interpolation is needed in order to investigate quadrature (numerical integration) – see Section 2.

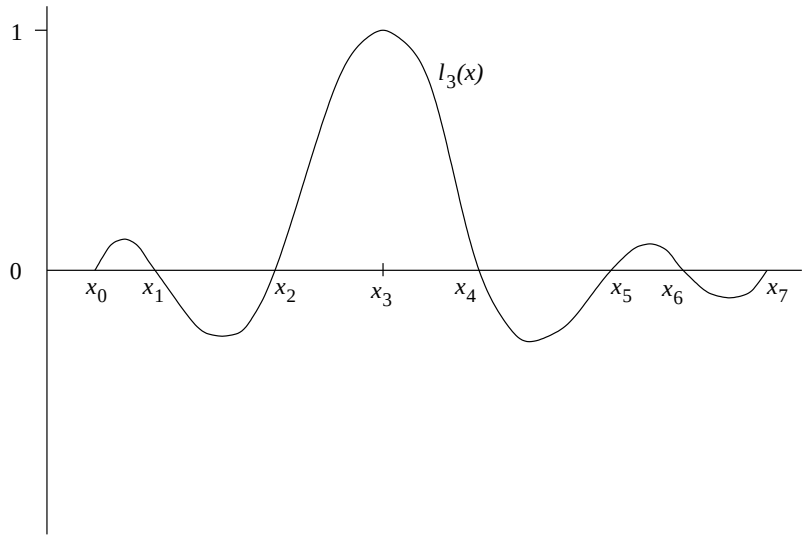
Of the various ways of expressing the formula for polynomial interpolation, the most useful for theoretical purposes is the *Lagrange* form:

$$L_n(x) = \sum_{i=0}^{n} f(x_i) l_i(x) \tag{1.3}$$

where each polynomial $l_i(x)$ depends only on the set of points $x_0, x_1, \ldots x_n$ and not on the function values. The polynomials $\{l_i(x)\}$ are called the *cardinal functions* and each has the form

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{1.4}$$

so that $l_i(x_i) = 1$ and $l_i(x_k) = 0$ for $k \neq i$, for example:



By choosing a particular interpolation point, say $x = x_k$, it is easy to verify that formula (1.3) is correct.

We define the error in $L_n(x)$ by

$$e_n(x) = f(x) - L_n(x)$$

so that $e_n(x_i) = 0$ for $i = 0, 1, \ldots n$. However we are more interested in the error at some general point x which is not one of the interpolation points. In order to treat x as a single point it is convenient to introduce another variable z and let $p_{n+1}(z)$ represent the polynomial of degree $n+1$ defined by

$$p_{n+1}(z) = (z - x_0)(z - x_1) \ldots (z - x_n) \tag{1.5}$$

where $p_{n+1}(x_i) = 0$ and $p_{n+1}(x) \neq 0$. Note that we can re-write (1.4) in the form

$$l_i(x) = \frac{p_{n+1}(x)}{p'_{n+1}(x_i)(x - x_i)} \tag{1.6}$$

Exercise 1a

By differentiating (1.5) show that formulae (1.4) and (1.6) are equivalent.

Let $L_n(x)$ represent the polynomial that interpolates the function $f(x)$ at the points x_0, x_1, ..., x_n. To establish the error term in polynomial interpolation, we construct the function

$$F(z) = f(z) - L_n(z) - [f(x) - L_n(x)]\frac{p_{n+1}(z)}{p_{n+1}(x)}$$

which has the property that

$$\begin{aligned} F(x_i) &= 0, \qquad i = 0, 1, \ldots n \\ F(x) &= 0, \end{aligned}$$

i.e. the function $F(z)$ has $n+2$ zeros. We now assume that $f(z)$ is $n+1$ times continuously differentiable, so that $F(z)$ is also $n+1$ times continuously differentiable. By *Rolle's Theorem*‡ $F'(z)$ has $n+1$ zeros, $F''(z)$ has n zeros, etc. It follows that $F^{(n+1)}(z)$ has at least 1 zero in the interval spanned by x_0, x_1, ..., x_n and x. Now $p_{n+1}^{(n+1)}(z) = (n+1)!$ so

$$F^{(n+1)}(z) = f^{(n+1)}(z) - L_n^{(n+1)}(z) - [f(x) - L_n(x)]\frac{(n+1)!}{p_{n+1}(x)} \tag{1.7}$$

Let $z = \xi$ be a point at which $F^{(n+1)}(z) = 0$. It follows from setting $z = \xi$ in (1.7) that

$$e_n(x) = \frac{p_{n+1}(x)}{(n+1)!} f^{(n+1)}(\xi). \tag{1.8}$$

The Lagrange formula (1.3) does not suggest an efficient algorithm for its evaluation so, in practical computation, polynomial interpolation is usually performed by *iterated linear interpolation* which is equivalent in exact arithmetic. Writing the formula for linear interpolation between the two points (x_i, u) and (x_j, v) as

$$L_{ij}[u, v](x) = \left(\frac{x - x_j}{x_i - x_j}\right)u + \left(\frac{x - x_i}{x_j - x_i}\right)v$$

we write $f_i = f(x_i)$ and define the array of interpolants $\{f_{ij}\}$ as follows.

linear	quadratic	cubic
$f_{01} = L_{01}[f_0, f_1]$	$f_{02} = L_{02}[f_{01}, f_{12}]$	$f_{03} = L_{03}[f_{02}, f_{13}]$
$f_{12} = L_{12}[f_1, f_2]$	$f_{13} = L_{13}[f_{12}, f_{23}]$	...
$f_{23} = L_{23}[f_2, f_3]$	...	
...		

In general, $f_{ij} = L_{ij}[f_{i,j-1}, f_{i+1,j}]$.

As a practical method of approximation, interpolation has drawbacks: because all data in floating point representations are potentially erroneous it is better to use methods such as least squares fitting, even if high accuracy is required.

‡ Recall that *Rolle's Theorem* essentially states that if $g(a) = 0$ and $g(b) = 0$ when $g(x)$ is continuously differentiable over the interval $[a, b]$ then $g'(x) = 0$ somewhere in the interval (a, b). This is fairly obvious, but is very useful in analysis.

1.3 Best approximations

A more practical approach to approximation is to consider some function $f(x)$ defined over an *interval* I.

The L_p *norms* on the interval I are defined by

$$\|f\|_p = \left\{ \int_I |f(x)|^p \, dx \right\}^{\frac{1}{p}}.$$

The *space of functions* for which this integral exists is usually denoted by L_p. [Recall that the corresponding *vector space* (or its norm) is usually denoted by l_p.]

Let $g(x)$ be an approximation to the function $f(x)$. We can choose $g(x)$ to minimize various *error norms* in order to approximate over an interval I of the real line.

* *Least first power* or L_1 approximation:

$$\min_g \int_I |f(x) - g(x)| \, dx.$$

* *Least squares* or L_2 approximation:

$$\min_g \int_I [f(x) - g(x)]^2 \, dx.$$

  Note that the L_2 norm itself is actually the square root of the integral.

* *Minimax* or *Chebyshev* or *uniform* or L_∞ approximation:

$$\min_g \max_{x \in I} |f(x) - g(x)|.$$

To accomplish this we frequently use the limiting properties of an approximation on a finite point set. The L_∞ approximation, for example, interpolates $f(x)$ at a set of points that cannot be predetermined.

Approximations of the three types defined above are called *best approximations.* We will look first at the theory of L_∞ approximation, which requires the interesting properties of *Chebyshev polynomials.*

Weight functions may be introduced into these norms, most usefully in the L_2 case – see Section 1.5.

1.4 Chebyshev methods

1.4.1 Chebyshev polynomials

Note that $\cos k\theta$ can be expressed as a polynomial in $\cos\theta$, e.g.

$$\begin{aligned} \cos 2\theta &= 2\cos^2\theta - 1 \\ \cos 3\theta &= 4\cos^3\theta - 3\cos\theta. \end{aligned}$$

Setting $x = \cos\theta$ we define the Chebyshev polynomial $T_k(x) = \cos(k\theta)$. The first few Chebyshev polynomials are

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

$T_k(x)$ is a polynomial of degree k in x which satisfies the recurrence relation

$$T_{k+1} = 2xT_k - T_{k-1}.$$

$T_k(x)$ has k real zeros in the interval $[-1,1]$ at

$$x = \cos\frac{(2j-1)\pi}{2k}; \quad j = 1,2,\ldots k.$$

$T_k(x)$ has $k+1$ maxima and minima (known collectively as *extrema*) in $[-1,1]$ at

$$x = \cos\frac{j\pi}{k}; \quad j = 0,1,2,\ldots k$$

such that $T_k(x) = \pm 1$ at these points. T_k may be described as an 'equal ripple' function in the range $[-1,1]$ and over the domain $[-1,1]$. The k zeros separate the $k+1$ extrema.

Chebyshev polynomials are orthogonal over $[-1,1]$ with respect to the weight function $1/\sqrt{1-x^2}$, i.e.

$$\begin{aligned} \int_{-1}^{1} \frac{T_r(x)T_s(x)}{\sqrt{1-x^2}}\,dx &= 0, \qquad r \neq s \\ \int_{-1}^{1} \frac{\{T_r(x)\}^2}{\sqrt{1-x^2}}\,dx &= \begin{cases} \pi, & r = 0 \\ \frac{\pi}{2}, & r \neq 0. \end{cases} \end{aligned} \tag{1.9}$$

Theorems involving Chebyshev polynomials typically apply to the interval $[-1,1]$. However this is not a restriction and Chebyshev methods can be applied over a general closed interval $[a,b]$ by means of the linear transformation

$$x = \frac{t - \frac{1}{2}(b+a)}{\frac{1}{2}(b-a)}.$$

Consequently we only consider $[-1,1]$ in Section 1.4.

In a polynomial of degree k the coefficient of the term in x^k is called the *leading coefficient*. Thus the leading coefficient of $T_k(x)$ is 2^{k-1}. It is often convenient to *normalize* the Chebyshev polynomials and consider instead the polynomials

$$\bar{T}_k(x) = T_k(x)/2^{k-1}$$

so that the leading coefficient of $\bar{T}_k(x)$ is 1. The extrema of $\bar{T}_k(x)$ are $\pm 1/2^{k-1}$.

1.4.2 L_∞ approximation

Theorem 1.2. *Let S_k be the set of all polynomials of degree k whose leading coefficient is 1, that is the set of all polynomials of the form*

$$P_k(x) = x^k + c_{k-1}x^{k-1} + \ldots + c_1 x + c_0.$$

Then $\bar{T}_k(x)$ is the polynomial in the set S_k with least absolute value over $[-1,1]$, i.e.

$$|\bar{T}_k(x)| \leq \frac{1}{2^{k-1}}$$

for all $x \in [-1,1]$ and

$$|P_k(x)| > \frac{1}{2^{k-1}}$$

for some $x \in [-1,1]$ for any $P_k(x) \in S_k$ such that $P_k(x) \not\equiv \bar{T}_k(x)$.

Proof omitted.

Theorem 1.3. *The best polynomial approximation over $[-1,1]$ to x^n of lower degree is $x^n - \bar{T}_n(x)$, which is of degree $n-2$.*

Proof omitted.

Theorem 1.4: The Chebyshev characterization theorem. *The best L_∞ approximation to $f(x) \in C[-1,1]$ by a polynomial $p_{n-1}(x)$ of degree $n-1$ has the property that*

$$\max_{x\in[-1,1]} |e(x)|$$

is attained at $n+1$ distinct points $-1 \le \xi_0 < \xi_1 < \ldots < \xi_n \le 1$ such that

$$e(\xi_j) = -e(\xi_{j-1})$$

for $j = 1, 2, \ldots n$ where

$$e(x) = f(x) - p_{n-1}(x).$$

The best L_∞ approximation is unique.

Proof. Assume that $e(x) \not\equiv 0$, otherwise the theorem is trivial. Let $\max_{x\in[-1,1]} |e(x)|$ be attained at the $r+1$ distinct points $-1 \le \eta_0 < \eta_1 < \ldots < \eta_r \le 1$. Consider the sign changes in the list $e(\eta_0), e(\eta_1), \ldots e(\eta_r)$. If the kth sign change is between $e(\eta_i)$ and $e(\eta_{i+1})$ then define

$$\mu_k = \tfrac{1}{2}(\eta_i + \eta_{i+1}).$$

If there are s sign changes then we have $-1 < \mu_1 < \mu_2 < \ldots < \mu_s < 1$. Let $q_s(x)$ be the unique polynomial of degree s that is zero at $\mu_1, \mu_2, \ldots \mu_s$ and such that $q_s(\eta_0) = e(\eta_0)$. Then $q_s(x)$ is non-zero and has the same sign as $e(x)$ whenever $|e(x)|$ attains its maximum. Therefore, if $p_{n-1}(x)$ is the best polynomial approximation of degree $n-1$ to $f(x)$ and ε is a sufficiently small positive number then

$$p_{n-1}(x) + \varepsilon q_s(x)$$

is a better approximation than $p_{n-1}(x)$. Hence we deduce that s must be greater than $n-1$. Hence r must be at least n and it only remains to prove uniqueness.

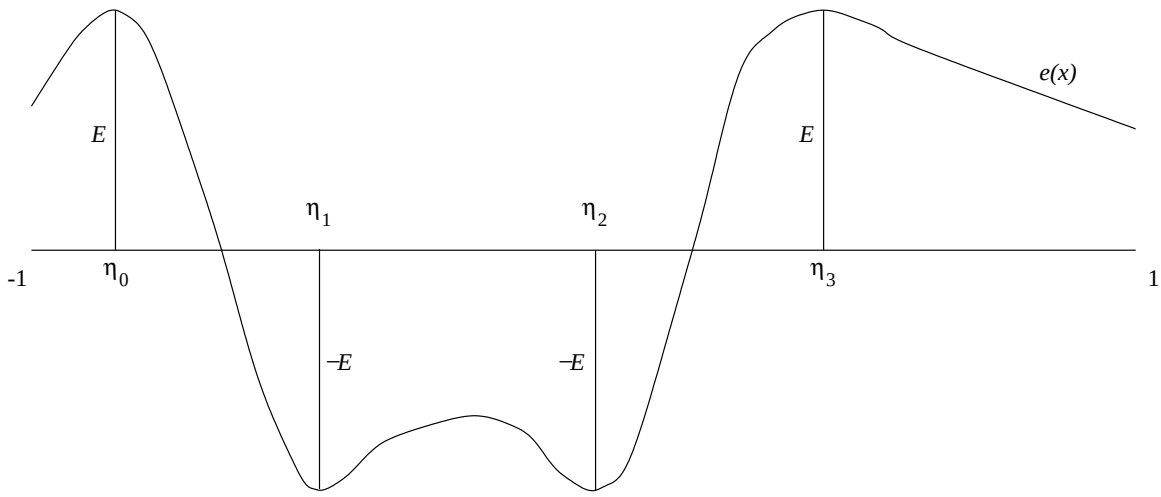
Assume that $q_{n-1}(x)$ is equally as good an approximation as $p_{n-1}(x)$, then so is $(p_{n-1} + q_{n-1})/2$ because

$$|f - \tfrac{1}{2}(p_{n-1} + q_{n-1})| \le \tfrac{1}{2}|f - p_{n-1}| + \tfrac{1}{2}|f - q_{n-1}| \le \tfrac{1}{2}E + \tfrac{1}{2}E = E$$

where E is the absolute value of the error incurred by p_{n-1} or q_{n-1} at each extremum. At an extremum of $f - \frac{1}{2}(p_{n-1} + q_{n-1})$ it is necessary for $p_{n-1} = q_{n-1}$ otherwise either $|f - p_{n-1}|$ or $|f - q_{n-1}|$ would exceed E at this point. But there are $n+1$ extrema of $f - \frac{1}{2}(p_{n-1} + q_{n-1})$ so p_{n-1} and q_{n-1} must be identical (because $p_{n-1} - q_{n-1}$ would have $n+1$ zeros if not).
¶

Note that the first part of the proof leads to a constructive algorithm.

Sketch graphs may be found useful to show how the proof works. The following diagram shows an example of the error $e(x)$. This diagram can also be used to show how a non-best approximation can be improved by adding a sufficiently small multiple of $q_s(x)$.



1.4.3 l_∞ approximation

The Chebyshev characterization theorem also holds for best l_∞ approximation if $[-1,1]$ is replaced by a set X_m of m discrete points, where $m \geq n+1$. In practice we cannot generally solve the approximation problem on an interval so we solve the approximation problem over a discrete point set in the required interval. From the characterization theorem the approximation problem over X_m reduces to finding a set of $n+1$ points in X_m which characterize the best approximation. The computation is usually performed using the 'exchange algorithm' or a linear programming method.

1.4.4 Chebyshev series

We can use the orthogonality property of Chebyshev polynomials (1.9) to expand a function $f \in L_2[-1,1]$ in the form

$$f(x) = \tfrac{1}{2}c_0 + \sum_{r=1}^{\infty} c_r T_r(x) \tag{1.10}$$

where

$$c_r = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_r(x)}{\sqrt{1-x^2}}\,dx. \tag{1.11}$$

Exercise 1b

By considering

$$\int_{-1}^{1} \frac{f(x)T_s(x)}{\sqrt{1-x^2}}\,dx$$

and assuming $f(x)$ has an expansion like (1.10), verify formula (1.11).

Note that, with the substitution $x = \cos\theta$, (1.10) becomes a Fourier cosine series. The error in truncating (1.10) after $n+1$ terms is smaller (in the Chebyshev sense) than the error in truncating a Taylor series after $n+1$ terms.

If the integrals (1.11) are evaluated by quadrature, various approximations to (1.10) result, e.g. using the

change of variable $x = \cos\theta$ and the repeated trapezium rule at intervals (of θ) of π/n we get

$$\tilde{f}_n(x) = \tfrac{1}{2}\tilde{c}_0 + \sum_{r=1}^{n-1} \tilde{c}_r T_r(x) + \tfrac{1}{2}\tilde{c}_n T_n(x)$$

where

$$\tilde{c}_r = \frac{2}{n} \sum_{j=0}^{n}{}'' T_r(x_j) f(x_j); \qquad x_j = \cos\frac{j\pi}{n}.$$

(The double prime indicates that the first and last terms of the summation are halved.) Such approximations are useful, particularly when the integrals (1.11) have to be evaluated numerically, but the coefficients $\{\tilde{c}_r\}$ now also depend on n so this is no longer a series expansion.

1.4.5 Lagrange interpolation with Chebyshev abscissae

From (1.8) the error in Lagrange interpolation over the points x_1, x_2, $\ldots x_n$ is given by

$$f(x) - L_{n-1}(x) = \frac{f^{(n)}(\xi)}{n!} \prod_{j=1}^{n} (x - x_j).$$

In practice we do not usually know $f^{(n)}(\xi)$ but we can attempt to reduce this error by minimizing $|\prod_{j=1}^{n}(x - x_j)|$ over the interval $[-1, 1]$. Choose the abscissae

$$x_j = \cos\frac{(2j-1)\pi}{2n} \qquad j = 1, 2, \ldots n \tag{1.12}$$

so that

$$\prod_{j=1}^{n} (x - x_j) \equiv \bar{T}_n(x).$$

Theorem 1.5. *The abscissae (1.12) minimize*

$$\max_{x\in[-1,1]} \left| \prod_{j=1}^{n} (x - x_j) \right|.$$

Proof similar to Theorem 1.4, and omitted.

We conclude that the abscissae (1.12) are a reasonable choice for Lagrange interpolation of an n-times differentiable function $f(x)$. In fact, if $f(x)$ is a polynomial of degree n then we obtain the best L_∞ approximation out of polynomials of degree $n-1$ over $[-1, 1]$ since $f^{(n)}(x)$ is a constant. It may be proved that Lagrange interpolation over the abscissae (1.12) converges uniformly as $n \to \infty$ for meromorphic§ functions $f(x)$. In contrast, using equally-spaced abscissae, Runge showed† that Lagrange interpolation diverges for the apparently well-behaved function $1/(1 + x^2)$ over $[-5, 5]$.

§ A *meromorphic function* is a function of a complex variable whose only singularities are *poles*.
† By means of complex variable theory.

1.4.6 Economization of Taylor series

We can approximate a Taylor series by truncating to a polynomial, say

$$P_n(x) = a_0 + a_1 x + a_2 x^2 \ldots + a_n x^n. \tag{1.13}$$

We have seen that the best L_∞ approximation to (1.13) over $[-1, 1]$ by a polynomial of lower degree is given by Lagrange interpolation at the zeros of $T_n(x)$. The error in approximating (1.13) is then

$$\begin{aligned} P_n(x) - L_{n-1}(x) &= \frac{P_n^{(n)}(\xi)}{n!} \bar{T}_n(x) \\ &= a_n \bar{T}_n(x). \end{aligned}$$

Hence we can *economize* $P_n(x)$ by forming

$$L_{n-1}(x) = P_n(x) - a_n \bar{T}_n(x),$$

i.e. forming a polynomial approximation of one degree lower‡. The maximum error due to economization is therefore $a_n/2^{n-1}$. The method is equivalent to truncating an approximate Chebyshev series after the term in T_{n-1}, so we may repeat the process if the leading coefficient is small e.g., rounding to 5 decimal places,

$$\begin{aligned} e^x &\simeq 1 + x + 0.5x^2 + 0.16667x^3 + 0.04167x^4 + 0.00833x^5 + 0.00139x^6 \\ &\simeq 1.26606T_0 + 1.13021T_1 + 0.27148T_2 + 0.04427T_3 + 0.00547T_4 + 0.00052T_5 + 0.00004T_6. \end{aligned}$$

Error in truncating Taylor series after term in $x^6 \simeq 0.00023$.
Economization error in neglecting terms in T_5 and $T_6 \simeq 0.00052 + 0.00004 = 0.00056$.
Total error after economization $\simeq 0.00023 + 0.00056 = 0.00079$.
This may be compared with the error in truncating the Taylor series after the term in x^4 ($\simeq 0.0099$).

1.5 Least-squares approximation over a finite interval

Suppose ϕ_0, ϕ_1, ϕ_2, ... are a *basis*, i.e. a set of approximating functions $\phi_n(x)$, over an interval $[a, b]$. Let $f(x)$ be a (suitable) function to be approximated by

$$f(x) = \sum_{j=0}^{n} a_j \phi_j. \tag{1.14}$$

Let $w(x)$ be a *weight function*, that is $w(x)$ must be positive in the interval $[a, b]$ except that it may be zero at either end-point. In order to obtain a least-squares fit to the function $f(x)$ we need to find the coefficients a_0, a_1, a_2, ... a_n which minimize

$$I = \int_a^b w(x) \left\{ f(x) - \sum_{j=0}^{n} a_j \phi_j \right\}^2 dx.$$

To find the minimum we take a particular a_i and form the equation $\partial I / \partial a_i = 0$, i.e.

$$\frac{\partial I}{\partial a_i} = -2 \int_a^b w(x) \left\{ f(x) - \sum_{j=0}^{n} a_j \phi_j \right\} \phi_i \, dx = 0.$$

‡ This approximation may also be derived from Theorem 1.3.

Exercise 1c

Verify the above formula for $\partial I/\partial a_i$.

(Note that the factor -2 can be ignored.) We get $n+1$ such equations and observe that they can be written in the form

$$\mathbf{Ca} = \mathbf{b}$$

where $\mathbf{a} = \{a_0, a_1, \ldots a_n\}$ and, writing $\mathbf{C} = (c_{ij})$, $\mathbf{b} = (b_i)$, we have

$$c_{ij} = \int_a^b w(x)\phi_i\phi_j \, dx, \tag{1.15}$$

$$b_i = \int_a^b w(x)f(x)\phi_i \, dx. \tag{1.16}$$

Assuming that all these integrals can be easily computed, we still have to solve an $(n+1) \times (n+1)$ set of simultaneous equations. This is by no means trivial and, in the interesting case $\phi_n \equiv x^n$, the solution is very difficult to obtain even to moderate accuracy§. However if we can arrange that $c_{ij} = 0$ for $i \neq j$ then we have the equations

$$\begin{bmatrix} c_{00} & & & & \\ & c_{11} & & & \\ & & c_{22} & & \\ & & & \cdots & \\ & & & & c_{nn} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

with the trivial solution

$$a_i = b_i/c_{ii} \qquad \text{for } \; i = 0, 1, 2, \ldots n.$$

Further, if we arrange for $c_{ii} = 1$ for all i then the solution is $a_i = b_i$ and we have reduced the problem to that of computing the integrals (1.16). If the sequence ϕ_0, ϕ_1, ϕ_2, ... has the property that

$$c_{ij} = \int_a^b w(x)\phi_i\phi_j \, dx = 0 \quad \text{for } \; i \neq j$$

then it is said to be an *orthogonal basis*. Also, if

$$\int_a^b w(x)[\phi_i(x)]^2 \, dx = 1 \quad \text{for all } i$$

then the sequence $\{\phi_i\}$ is said to be *orthonormalized*.

For example, the Chebyshev polynomials form an orthogonal basis over $[-1, 1]$ with $w(x) = 1/\sqrt{1-x^2}$, so a truncated Chebyshev series is itself a best L_2 approximation.

§ On IBM System/370 hardware, $n = 4$ gives just 1 significant figure in single precision, and $n = 12$ gives 1 significant figure in double precision.

1.6 The Gram-Schmidt process

The 'modified' (i.e. numerically stable) version of this algorithm is described for forming an *orthogonal basis* from a set of basis functions or vectors. Let $\{\phi_0^0, \phi_1^0, \phi_2^0, \ldots \phi_n^0\}$ be some basis to be orthogonalized. We compute the triangular array (column by column):

$$\begin{array}{lllll}
\phi_0^0 & & & & \\
\phi_1^0 & \phi_1^1 & & & \\
\phi_2^0 & \phi_2^1 & \phi_2^2 & & \\
\ldots & \ldots & \ldots & \ldots & \\
\phi_n^0 & \phi_n^1 & \phi_n^2 & \ldots & \phi_n^n
\end{array}$$

where $\{\phi_0^0, \phi_1^1, \phi_2^2, \ldots \phi_n^n\}$ is the computed orthogonal basis. The formula is

$$\phi_r^s = \phi_r^{s-1} - a_r^s \phi_{s-1}^{s-1}$$

where

$$a_r^s = \frac{< \phi_r^{s-1}, \phi_{s-1}^{s-1} >}{< \phi_{s-1}^{s-1}, \phi_{s-1}^{s-1} >}$$

and $< \lambda, \mu >$ denotes an inner product such as $\int_a^b w(x)\lambda(x)\mu(x)\,dx$. If an extra basis function ϕ_{n+1}^0 is added it is necessary to carry out $n+1$ further steps to find ϕ_{n+1}^{n+1}.

Exercise 1d

> Starting from 1, x, x^2 find the first three *Legendre polynomials* $P_0(x)$, $P_1(x)$, $P_2(x)$ that are orthogonal over $[-1, 1]$ with respect to the weight function $w(x) = 1$.

1.7 Tables and interpolation

A method often used for approximation of the common transcendental functions (exponential, trigonometric functions, etc) is to store large tables of function values and to use interpolation to determine intermediate values. If values are stored to a greater accuracy than required by library functions, and the intervals between the arguments of stored values is small, then low order interpolation is sufficient. The method has been found to be faster than a purely algorithmic approach, provided disc access is fast.

1.8 Domain reduction in computation

In a numerical library function to compute $f(x)$ for any representable value of x it is seldom efficient to use a single approximation over the whole interval $(-\infty, \infty)$. Depending on $f(x)$ the following devices may be useful.

* Periodic reduction, e.g. $\sin x$. Find

$$n = \text{int}\left(\frac{x}{\pi} + \frac{1}{2}\right)$$

  and compute $t = x - n\pi$ so that $t \in [-\frac{\pi}{2}, \frac{\pi}{2})$. Then

$$\sin x = (-1)^n \sin t.$$

* Binary reduction, e.g. e^x. Find

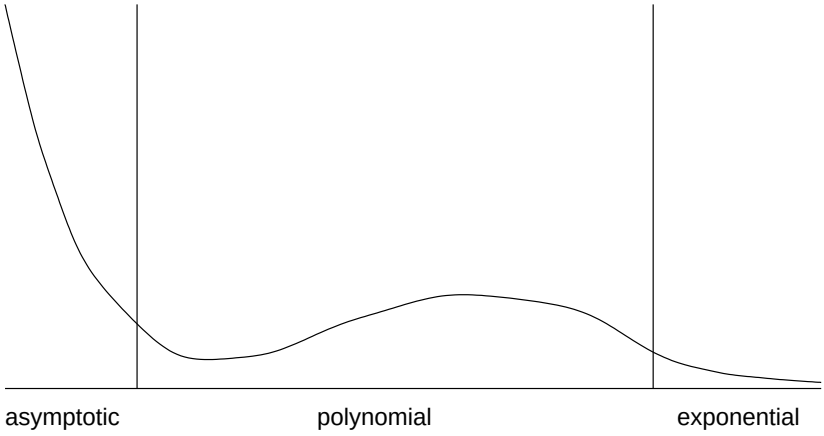
$$n = 1 \; + \; \text{int}(x \log_2 e)$$

and compute $t = n - x \log_2 e$ so that $t \in (0, 1]$. Then

$$e^x = 2^{x \log_2 e} = 2^{n-t} = 2^n.2^{-t}$$

and $2^{-t} \in [\frac{1}{2}, 1)$.

* Subdivision of the interval.



1.8.1 Example: approximation of square roots

To evaluate square roots on a computer Newton's method is normally used since it gives quadratic convergence near the root. Writing $t = \sqrt{x}$ Newton's method for $f(t) = t^2 - x$ is

$$t_{n+1} = \frac{1}{2}(t_n + \frac{x}{t_n}).$$

Writing $t_n = \sqrt{x} + e_n$ we have

$$\sqrt{x} + e_{n+1} = \frac{1}{2}(\sqrt{x} + e_n + \sqrt{x}\{1 - \frac{e_n}{\sqrt{x}} + \frac{e_n^2}{x} - \ldots\})$$

so

$$e_{n+1} = \frac{1}{2}\frac{e_n^2}{\sqrt{x}} + O(e_n^3). \tag{1.17}$$

It may be shown that the method converges for any $t_0 > 0$, so the problem reduces to finding an optimal starting value t_0.

Consider *normalized numbers* on a binary floating point implementation. We have $x = m \times 2^k$ where k is the exponent and $m \in [1, 2)$. If k is even $\sqrt{x} = \sqrt{m} \times 2^{k/2}$. If k is odd the correct exponent is given by

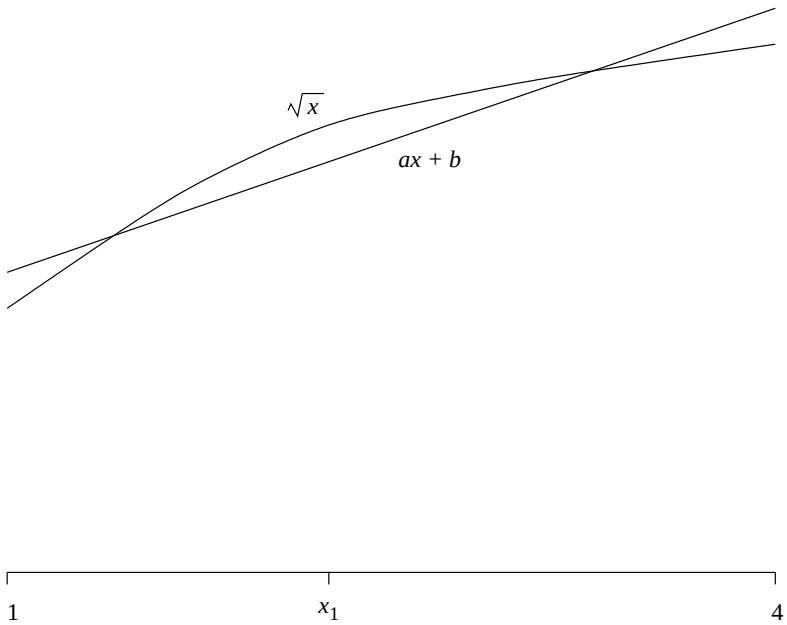
$$\sqrt{x} = \sqrt{2m} \times 2^{(k-1)/2}$$

since

$$\sqrt{2m} \in [\sqrt{2}, 2) \subset [1, 2).$$

Hence we need only use Newton's method for $\sqrt{m}$ or $\sqrt{2m}$. We require a starting approximation t_0 for $\sqrt{x}$

where $x \in [1, 4)$. The L_∞ linear approximation is suitable.



The error has three extrema at $1, x_1, 4$ by Theorem 1.4. Therefore the error

$$e(x) = ax + b - \sqrt{x}$$

has a turning point at x_1 so that, setting $de/dx = 0$ at $x = x_1$, we get

$$x_1 = \frac{1}{4a^2}.$$

From Theorem 1.4, $e(1) = -e(x_1) = e(4)$, i.e.

$$a + b - 1 = -\frac{1}{4a} - b + \frac{1}{2a} = 4a + b - 2$$

which gives $a = 1/3$, $b = 17/24$, and $\max |e_0| = 1/24$. From (1.17) $|e_1| \simeq 10^{-3}$.

Exercise 1e

> Repeat the derivation of the method of Section 1.8.1 for use on a calculator that uses base 10. Calculate the maximum relative error in the starting approximation and explain why the method works better in base 2 than in base 10. Assuming a precision of 9 decimal digits, and an absolute error $e_0 \simeq 1$ in the starting approximation, estimate how many iterations will be required in the worst case. (Assume $2^{10} \simeq 10^3$ in your calculation.)

1.9 Splines

This discussion is restricted to cubic splines, or splines of order 3. However, splines are well defined for all orders. In particular, a piecewise straight line (or *polyline*) is a spline of order 1.

1.9.1 Interpolating splines

A simplified treatment was given in Numerical Analysis I; this section provides the algebraic details. The *cubic spline function* $\phi(x)$ interpolating the values $\{y_j\}$ at the *knots* $\{x_j\}$ over the interval $[a, b]$ may be defined by the following conditions. Note that there is no requirement that interpolation should be at the knots, so this is only one illustrative case.

(i) ϕ is a polynomial of degree 3 in each interval $[x_1, x_2], [x_2, x_3], \ldots [x_{n-1}, x_n]$.

(ii) ϕ is a polynomial of degree 1 in each interval $[a, x_1], [x_n, b]$.

(iii) ϕ, ϕ' and ϕ'' are continuous in $[a, b]$.

(iv) $\phi(x_j) = y_j$, $j = 1, 2, \ldots n$.

The cubic spline is a piecewise cubic polynomial and has $4n$ coefficients. The conditions (i)–(iv) produce $4n$ linear equations which may be shown to be non-singular so that ϕ is uniquely determined.

The spline $\phi(x)$ has the following *minimum curvature property*†. Let $\psi(x)$ be any function, different from $\phi(x)$, such that ψ and ψ' are continuous in $[a, b]$ and ψ'' exists. Suppose also that condition (iv) holds for ψ, then

$$\begin{aligned}\int_a^b \{\psi''(x)\}^2\,dx - \int_a^b \{\phi''(x)\}^2\,dx &= \int_a^b \{\psi''(x) - \phi''(x)\}^2\,dx + 2\int_a^b \phi''(x)\{\psi''(x) - \phi''(x)\}\,dx \\ &> 2\int_a^b \phi''(x)\{\psi''(x) - \phi''(x)\}\,dx.\end{aligned}$$

Integrating by parts, the right-hand side of the inequality is zero so that

$$\int_a^b \{\psi''(x)\}^2\,dx > \int_a^b \{\phi''(x)\}^2\,dx,$$

i.e. $\phi(x)$ minimizes this integral out of all twice-differentiable interpolating functions $\psi(x)$ with continuous first derivative.

Exercise 1f

By integrating by parts, show that

$$\int_a^b \phi''(x)\{\psi''(x) - \phi''(x)\}dx = 0.$$

[Hint: use the fact that $\phi'''(x)$ is a piecewise constant.]

Defining $d_j = x_{j+1} - x_j$ and $\mu_j = \phi''(x_j)$, it may be verified that for $x \in [x_j, x_{j+1}]$

$$\phi(x) = \frac{(x - x_j)y_{j+1} + (x_{j+1} - x)y_j}{d_j} - \frac{(x - x_j)(x_{j+1} - x)\{(d_j + x_{j+1} - x)\mu_j + (d_j + x - x_j)\mu_{j+1}\}}{6d_j} \tag{1.18}$$

so that, in particular,

$$\begin{aligned}\phi'(x_j) &= \frac{y_{j+1} - y_j}{d_j} - \frac{(2\mu_j + \mu_{j+1})d_j}{6} \\ \phi'(x_{j+1}) &= \frac{y_{j+1} - y_j}{d_j} + \frac{(2\mu_{j+1} + \mu_j)d_j}{6}.\end{aligned} \tag{1.19}$$

Exercise 1g

Verify from (1.18) that $\phi(x_j) = y_j$, $\phi(x_{j+1}) = y_{j+1}$, $\phi''(x_j) = \mu_j$, $\phi''(x_{j+1}) = \mu_{j+1}$.

† The *minimum curvature property* is a misleading name: what we are talking about here is the *second derivative*, not *curvature* as such. In this case we require a scalar measurement over an interval, so we choose the L_2 norm of the second derivative.

From the continuity condition, and using (1.19), we can equate the values of $\phi'(x_j)$ in the pair of intervals $[x_{j-1}, x_j]$, $[x_j, x_{j+1}]$ to get

$$\frac{y_j - y_{j-1}}{d_{j-1}} + \frac{(2\mu_j + \mu_{j-1})d_{j-1}}{6} = \frac{y_{j+1} - y_j}{d_j} - \frac{(2\mu_j + \mu_{j+1})d_j}{6} \tag{1.20}$$

for $j = 2, 3, \ldots n-1$. We also know that ϕ is linear in $[a, x_1], [x_n, b]$ so that $\mu_1 = \mu_n = 0$. Let $\mathbf{m} = \{\mu_2, \mu_3, \ldots \mu_{n-1}\}$. From (1.20) we get the linear equations $\mathbf{Am} = \mathbf{z}$ where

$$z_j = \frac{6(y_{j+1} - y_j)}{d_j} - \frac{6(y_j - y_{j-1})}{d_{j-1}}$$

and

$$\mathbf{A} = \begin{pmatrix} 2(d_1 + d_2) & d_2 & & & \\ d_2 & 2(d_2 + d_3) & d_3 & & \\ & d_3 & \ldots & \ldots & \\ & & \ldots & d_{n-2} & \\ & & d_{n-2} & 2(d_{n-2} + d_{n-1}) & \end{pmatrix}$$

which is symmetric positive definite and tridiagonal, and hence well-conditioned for solution. The spline is easily evaluated in the form (1.18).

Exercise 1h

> Prove $\mathbf{A}$ is positive definite, i.e. $\mathbf{x}^T\mathbf{Ax} > 0$ for any $\mathbf{x} \neq \mathbf{0}$ given $d_j > 0$ for all j.
> [Hint: in the expansion of $\mathbf{x}^T\mathbf{Ax}$, prove that each d_j has a positive coefficient.]

As stated above, there is no requirement that interpolation should be at the knots. Indeed, interpolation midway between the knots has some theoretical advantage. There are several variations of the above procedure in order to produce special-purpose splines. For example, *periodic splines* have identical end conditions at each end, to enable approximation of simple closed curves using polar coordinates. Also, because of their adaptability to fairly arbitrary shapes, *parametric splines* are particularly useful in computer aided design.

1.9.2 Convergence of interpolating splines

Define the *mesh norm* $\Delta\mathbf{x}$ by

$$\Delta\mathbf{x} = \max(x_1 - a, x_2 - x_1, x_3 - x_2, \ldots x_n - x_{n-1}, b - x_n)$$

and let f, f' be continuous in $[a, b]$ and let f'' be *square integrable* in $[a, b]$. Schoenberg showed that

$$\lim_{\Delta\mathbf{x}\to 0} \|f - \phi\|_\infty = 0$$

$$\lim_{\Delta\mathbf{x}\to 0} \|f' - \phi'\|_\infty = 0$$

$$\lim_{\Delta\mathbf{x}\to 0} \|f'' - \phi''\|_2 = 0.$$

These results are independent of how $\Delta\mathbf{x} \to 0$.

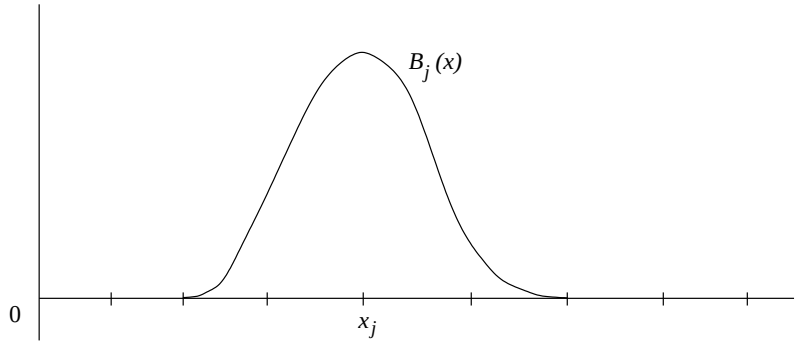
1.9.3 B-splines

It is possible to express the cubic spline $\phi(x)$ in terms of a linear combination of functions $\{B_j(x)\}$ which are themselves cubic splines which depend only on the set of knots x_1, x_2, $\ldots x_n$ and the end conditions. The general form of a spline over a set of n knots is then

$$\phi(x) = \sum_{j=-1}^{n+2} c_j B_j(x) \tag{1.21}$$

where $\{c_j\}$ is a set of coefficients to be determined. Note that $B_{-1}(x)$, $B_0(x)$, $B_{n+1}(x)$, $B_{n+2}(x)$ depend on the end conditions which need not concern us here.

The function $B_j(x)$ is called a *B-spline* and, for $j = 1, 2, \ldots n$, has the property that $B_j(x_k) = 0$ for $k \leq j-2$ and $k \geq j+2$, as shown below.



Note that the summation (1.21) for a particular value of x is over at most 4 terms. The study of B-splines leads to improved algorithms for spline approximation, and a clearer theoretical understanding of this topic.

2. Quadrature

Quadrature (or numerical integration) was introduced in Sections 2 and 3 of Numerical Analysis I. This Section takes a more theoretical approach.

2.1 Theory

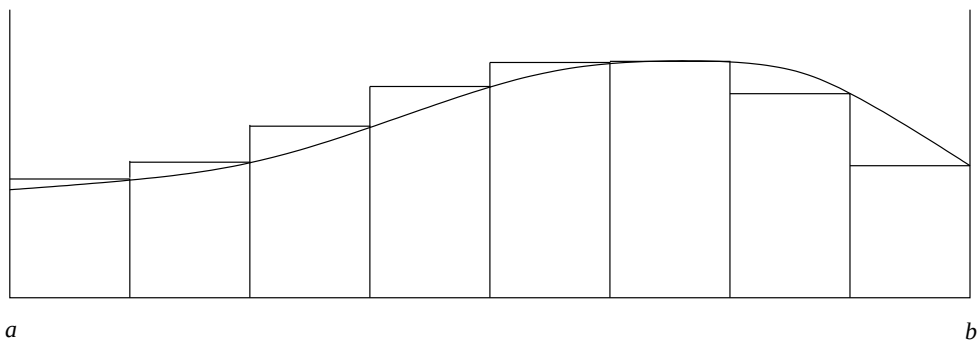
2.1.1 Riemann sum rules

Consider a set of numbers $a = \xi_0 < \xi_1 < \ldots < \xi_n = b$. A *Riemann integral* is defined by

$$\int_a^b f(x)\,dx = \lim_{\substack{n\to\infty \\ \Delta\xi\to 0}} \sum_{i=1}^{n} (\xi_i - \xi_{i-1}) f(x_i) \tag{2.1}$$

where $x_i \in [\xi_{i-1}, \xi_i]$, $\Delta\xi = \max_i |\xi_i - \xi_{i-1}|$. The summation on the right hand side of (2.1) is called a *Riemann sum*. Some primitive quadrature rules are clearly Riemann sums, e.g. the composite rectangular rule

$$\mathbf{R}_n f = h \sum_{i=1}^{n} f(a + ih)$$



with $h = (b-a)/n$. Other quadrature rules of the form

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

can be expressed as Riemann sums if there exist $\xi_1 < \xi_2 < \ldots < \xi_{n-1}$ such that

$$w_i = \xi_i - \xi_{i-1}, \qquad \xi_0 = a, \;\; \xi_n = b$$

and $x_i \in [\xi_{i-1}, \xi_i]$.

To show that a given sequence of quadrature rules $\{Q_n f\}$ converges to the integral of a function, it is necessary and sufficient to establish that each rule is a Riemann sum such that $\Delta\xi \to 0$ as $n \to \infty$.

2.1.2 Calculation of weights for quadrature rules

Interpolatory polynomial quadrature rules can be derived by integrating the Lagrange interpolation formula (1.3), i.e.

$$\begin{aligned}\int_a^b L_n(x)dx &= \int_a^b \sum_{i=0}^n f(x_i) l_i(x) dx \\ &= \sum_{i=0}^n f(x_i) \int_a^b l_i(x) dx \\ &= \sum_{i=0}^n w_i f(x_i)\end{aligned}$$

so that each w_i depends only on the choice of abscissae x_0, x_1, $\ldots x_n$.

Given a set of abscissae $x_0, x_1, \ldots x_n$ we wish to determine the unique weights $w_0, w_1, \ldots w_n$ such that the interpolating quadrature rule

$$\int_a^b W(x) f(x)\, dx = \sum_{i=0}^n w_i f(x_i) + E(f)$$

is exact for polynomials of degree n, where $W(x)$ is a weight function. Using equations (1.3) – (1.6) we can express the Lagrange polynomial interpolating to $f(x)$ at the abscissae as

$$L_n(x) = \sum_{i=0}^n f(x_i) \frac{p_{n+1}(x)}{p'_{n+1}(x_i)(x - x_i)}$$

so that

$$\begin{aligned}\sum_{i=0}^n w_i f(x_i) &= \int_a^b W(x) L_n(x)\, dx \\ &= \int_a^b W(x) \sum_{i=0}^n f(x_i) \frac{p_{n+1}(x)}{p'_{n+1}(x_i)(x - x_i)}\, dx \\ &= \sum_{i=0}^n f(x_i) \int_a^b W(x) \frac{p_{n+1}(x)}{p'_{n+1}(x_i)(x - x_i)}\, dx.\end{aligned}$$

Hence

$$w_i = \frac{1}{p'_{n+1}(x_i)} \int_a^b W(x) \frac{p_{n+1}(x)}{x - x_i}\, dx.$$

2.1.3 Error analysis

Theorem 2.1: Taylor's theorem. *If $x \in [a, b]$ and $f \in C^{N+1}[a, b]$,*

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!} f''(a) + \ldots + \frac{(x-a)^N}{N!} f^{(N)}(a) + \frac{1}{N!} \int_a^x f^{(N+1)}(t)(x-t)^N\, dt. \quad (2.2)$$

Proof. Let $T_N(a)$ represent the Taylor series up to the term in $(x-a)^N$ treated as a function of a, i.e.

$$T_N(a) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \ldots + \frac{(x-a)^N}{N!}f^{(N)}(a).$$

We can differentiate $T_N(a)$ once with respect to a,

$$\begin{aligned} T_N'(a) &= f'(a) - f'(a) + (x-a)f''(a) - (x-a)f''(a) + \ldots \\ &\qquad \ldots + \frac{(x-a)^{N-1}}{(N-1)!}f^{(N)}(a) - \frac{(x-a)^{N-1}}{(N-1)!}f^{(N)}(a) + \frac{(x-a)^N}{N!}f^{(N+1)}(a) \\ &= \frac{(x-a)^N}{N!}f^{(N+1)}(a). \end{aligned}$$

For any $g \in C^1[a,b]$

$$g(x) = g(a) + \int_a^x g'(t)\ dt$$

so

$$T_N(x) = T_N(a) + \int_a^x T_N'(t)\ dt$$

i.e.

$$f(x) = T_N(a) + \frac{1}{N!}\int_a^x (x-t)^N f^{(N+1)}(t)\ dt.$$

¶

Let $E(f)$ denote the discretization error in a quadrature rule, i.e.

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

Note that it is usual to write $E(f)$ as $E_x(f)$ to make it clear that integration is with respect to x. Suppose that the quadrature rule is exact for polynomials P_N of degree N, so that $E(P_N) = 0$. It is possible to obtain an expression for $E(f)$ by integrating the Lagrange interpolation formula (1.3).

Theorem 2.2: Peano's theorem. *If $E(P_N) = 0$ and $f \in C^{N+1}[a,b]$ then*

$$E(f) = \int_a^b f^{(N+1)}(t)K(t)\,dt$$

where

$$K(t) = \frac{1}{N!}E_x[(x-t)_+^N]$$

and

$$(x-t)_+^N = \begin{cases} (x-t)^N, & x > t \\ 0, & x \le t. \end{cases}$$

The function $K(t)$ is called the *Peano kernel* (or *influence function*) and is a spline function of order N.

Proof. We start from Taylor's theorem. Since $f \in C^{N+1}[a,b]$ and $E(P_r) = 0$ for $r \le N$ we get, directly from (2.2),

$$E(f) = \frac{1}{N!}E_x\Big\{\int_a^x f^{(N+1)}(t)(x-t)^N\ dt\Big\}.$$

It may be seen that it makes no difference to write

$$E(f) = \frac{1}{N!}E_x\Big\{\int_a^x f^{(N+1)}(t)(x-t)_+^N\ dt\Big\}$$

and further, because of the definition of $(x-t)_+^N$, we can replace the upper limit of the integral by b, thus

$$E(f) = \frac{1}{N!} E_x\Big\{ \int_a^b f^{(N+1)}(t)(x-t)_+^N \, dt \Big\}.$$

Now the limits of the integral no longer depend on x, so we can take E_x inside the integral to get

$$E(f) = \frac{1}{N!} \int_a^b f^{(N+1)}(t) E_x[(x-t)_+^N] \, dt.$$

¶

Peano's Theorem is quite elegant, but the algebraic details can get tedious, so we will restrict ourselves to the simplest quadrature rules as examples. Consider the simple trapezium rule for which

$$E(f) = \int_a^b f(x) \, dx - \tfrac{1}{2} h[f(a) + f(b)] \tag{2.3}$$

where $h = b - a$. The rule is exact for polynomials of degree 1, so taking $N = 1$ in Theorem 2.2 we get the Peano kernel

$$\begin{aligned} K(t) &= E_x[(x-t)_+] \\ &= \int_a^b (x-t)_+ \, dx - \tfrac{1}{2} h[(a-t)_+ + (b-t)_+] \end{aligned}$$

from (2.3). Since we require $K(t)$ only for $t \in [a, b]$ we can take $(a-t)_+ = 0$, $(b-t)_+ = b - t$ so that

$$\begin{aligned} K(t) &= \int_t^b (x-t) \, dx - \tfrac{1}{2} h(b-t) \\ &= \tfrac{1}{2}(a-t)(b-t). \end{aligned} \tag{2.4}$$

Using the kernel (2.4) in Peano's theorem we get

$$E(f) = \tfrac{1}{2} \int_a^b f''(t)(a-t)(b-t) \, dt. \tag{2.5}$$

To simplify (2.5) we need the following theorem.

Theorem 2.3. *If $K(t)$ does not change sign in $[a, b]$ then*

$$E(f) = \frac{f^{(N+1)}(\xi)}{(N+1)!} E(x^{N+1})$$

for some $\xi \in (a, b)$.

Proof. If $K(t)$ does not change sign the mean value theorem applied to Theorem 2.2 gives

$$E(f) = f^{(N+1)}(\xi) \int_a^b K(t) \, dt \tag{2.6}$$

and in particular

$$E(x^{N+1}) = (N+1)! \int_a^b K(t) \, dt. \tag{2.7}$$

The result follows from eliminating the integral from (2.6) and (2.7).
¶

To apply Theorem 2.3 to (2.5) we require

$$E(x^2) = \int_a^b x^2\,dx - \tfrac{1}{2}h(a^2+b^2) = -\tfrac{1}{6}h^3$$

so that

$$E(f) = -\tfrac{1}{12}h^3 f''(\xi)$$

for some $\xi \in (a,b)$.

Exercise 2a

> The mid-point rule $(b-a)f[\frac{1}{2}(a+b)]$ is exact for polynomials of degree 1. Use Peano's theorem to find a formula for $E(f)$. [Hint: this is similar to the trapezium rule example given above, except that it is harder to prove that $K(t)$ does not change sign in $[a,b]$.]

If $K(t)$ does change sign in $[a,b]$ then we can use the (sharp) inequality

$$|E(f)| \le \max_{x\in[a,b]} |f^{(N+1)}(x)|.\int_a^b |K(t)|\,dt.$$

2.2 Gaussian quadrature

In general, a polynomial quadrature rule with error $O(h^n)$ will integrate exactly a polynomial of degree $n-2$.

In *Gaussian quadrature* the abscissae $\{x_i\}$ of a quadrature rule are chosen such that the formula is accurate for the highest possible degree of polynomial. Consider

$$\int_a^b f(x)\,dx \simeq \sum_{i=1}^n w_i f(x_i).$$

At best the abscissae can be chosen such that an n-point rule is exact for polynomials of degree $2n-1$. Let $\{p_n\}$ be a sequence of *orthonormal polynomials* with respect to the weight function $W(x)$, i.e.

$$\int_a^b W(x)p_M(x)p_N(x)\,dx = \begin{cases} 0 & , M \ne N \\ 1 & , M = N. \end{cases}$$

Also, let $k_n > 0$ be the leading coefficient of $p_n(x)$, i.e.

$$p_n(x) = k_n x^n + \dots.$$

It is known that the zeros of real orthogonal polynomials are real, distinct and lie in (a,b). It may be shown that the zeros of $p_n(x)$ are the n abscissae $\{x_i\}$ which produce a quadrature rule that is exact for polynomials of degree $2n-1$. The weights are

$$w_i = -\frac{k_{n+1}}{k_n}\frac{1}{p_{n+1}(x_i)p_n'(x_i)}$$

and are always positive. The discretization error is given by

$$E(f) = \int_a^b W(x)f(x)\,dx - \sum_{i=1}^n w_i f(x_i) = \frac{f^{(2n)}(\xi)}{(2n)!\,k_n^2}$$

for some $\xi \in (a,b)$. Such Gauss rules are named after the orthogonal polynomials used, e.g. Gauss-Legendre, Gauss-Laguerre, etc.

The most commonly used Gauss rules are

$$\int_{-1}^{1} f(x)\,dx \qquad \text{Gauss} - \text{Legendre}$$

$$\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}\,dx \qquad \text{Gauss} - \text{Chebyshev}$$

$$\int_{-1}^{1} (1-x)^{\alpha}(1+x)^{\beta} f(x)\,dx \qquad \text{Gauss} - \text{Jacobi}$$

$$\int_{0}^{\infty} e^{-x} f(x)\,dx \qquad \text{Gauss} - \text{Laguerre}$$

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)\,dx \qquad \text{Gauss} - \text{Hermite}$$

Lobatto rules use both end points of the interval as abscissae, and the remaining abscissae are chosen optimally; these have degree $2n-3$. *Radau rules* use one end point as an abscissa and have degree $2n-2$; a Radau rule is often used instead of Gauss-Laguerre for the interval $[0,\infty)$. For integrals of the form $\int_{-\infty}^{\infty} e^{-x^2} f(x)\,dx$ the composite trapezium rule is superior to Gauss-Hermite – see Section 2.4.

2.3 Composite rules

Composite rules are also known as *compound*, *repeated*, *extended* or *iterated rules*. Let $\mathbf{R}$ be an m-point quadrature rule. We denote by $(n \times \mathbf{R})$ the rule $\mathbf{R}$ applied to n subintervals. Note that if $\mathbf{R}$ is an open rule (i.e. does not include end-points) then $(n \times \mathbf{R})$ uses mn points. If $\mathbf{R}$ is a closed rule (i.e. includes both end-points) then $(n \times \mathbf{R})$ uses only $(m-1)n+1$ points, which may be significantly less for small values of m. For $\mathbf{R}$ applied to $[-1,1]$ we write

$$\mathbf{R}f = \sum_{j=1}^{m} w_j f(x_j)$$

and for $(n \times \mathbf{R})$ applied to $[a,b]$ we write

$$(n \times \mathbf{R})f = \frac{b-a}{2n} \sum_{i=1}^{n} \sum_{j=1}^{m} w_j f(x_{ij}) \tag{2.8}$$

where x_{ij} is the jth abscissa of the ith subinterval. Note that the weights $\{w_j\}$ are independent of $[a,b]$. Also, for closed rules, some x_{ij}'s coincide.

Theorem 2.4. *Let $\mathbf{R}$ be a rule that integrates constants exactly, i.e. $\mathbf{R}.1 = \int_{-1}^{1} dx = 2$. If f is bounded in $[a,b]$ and is (Riemann) integrable then*

$$\lim_{n\to\infty} (n \times \mathbf{R})f = \int_a^b f(x)\,dx.$$

Proof. Reversing summations and taking the limit in (2.8),

$$\begin{aligned}\lim_{n\to\infty} (n \times \mathbf{R})f &= \tfrac{1}{2} \sum_{j=1}^{m} w_j \lim_{n\to\infty} \Big\{ \frac{b-a}{n} \sum_{i=1}^{n} f(x_{ij}) \Big\} \\ &= \tfrac{1}{2} \int_a^b f(x)\,dx. \sum_{j=1}^{m} w_j\end{aligned}$$

since the sum over i is a Riemann sum and therefore converges. Also, since $\mathbf{R}.1 = 2$, $\sum_{j=1}^{m} w_j = 2$.
¶

Theorem 2.4 essentially states that composite rules converge for bounded Riemann-integrable functions f.

The rule $\mathbf{R}$ is a *simplex rule* if the discretization error $E_{\mathbf{R}}(f)$ can be expressed in the form

$$E_{\mathbf{R}}(f) = c(b-a)^{k+1} f^{(k)}(\xi) \tag{2.9}$$

for $f \in C^k[a,b]$, $\xi \in (a,b)$, and constant c. The following theorem formalizes an idea that was explained in Numerical Analysis I, Section 2.

Theorem 2.5. *Let $\mathbf{R}$ be a simplex rule and let $E_{n\times\mathbf{R}}(f)$ denote the discretization error of $(n \times \mathbf{R})f$. Then*

$$\lim_{n\to\infty} n^k E_{n\times\mathbf{R}}(f) = c(b-a)^k [f^{(k-1)}(b) - f^{(k-1)}(a)].$$

That is to say, $(n \times \mathbf{R})f$ converges like n^{-k} for n sufficiently large.

Proof. From (2.9)

$$E_{n\times\mathbf{R}}(f) = \sum_{i=1}^{n} c\Big(\frac{b-a}{n}\Big)^{k+1} f^{(k)}(\xi_i)$$

where ξ_i lies inside the ith subinterval. Rearranging and taking limits,

$$\begin{aligned}
\lim_{n\to\infty} n^k E_{n\times\mathbf{R}}(f) &= c(b-a)^k \lim_{n\to\infty} \Big\{ \frac{b-a}{n} \sum_{i=1}^{n} f^{(k)}(\xi_i) \Big\} \\
&= c(b-a)^k \int_a^b f^{(k)}(x)\, dx \\
&= c(b-a)^k [f^{(k-1)}(b) - f^{(k-1)}(a)].
\end{aligned}$$

¶

For example, for Simpson's rule

$$E_{\mathbf{R}}(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi)$$

and

$$\lim_{n\to\infty} n^4 E_{n\times\mathbf{R}}(f) = -\frac{(b-a)^4}{2880} [f^{(3)}(b) - f^{(3)}(a)]$$

for $f \in C^4[a,b]$, i.e. Simpson's rule converges like h^5, so the composite rule converges like h^4.

2.4 Singular integrals

It is sometimes possible to integrate a function $f(x)$ over an interval in which it has a singularity. Typically such a singularity is a point where $f(x)$ or $f'(x)$ becomes infinite, but the integral remains finite. Although the value of the integral may be well-defined, quadrature methods based on polynomials are inappropriate as polynomials cannot display this behaviour.

It is necessary to know the location of the singularity or singularities, and to split the interval in such a way that each sub-integral has a singularity only at one end-point. Consequently we need only consider integrals with one singularity at an end-point of the interval.

Various analytic methods may be tried to remove a singularity before computation. For example, *integration by parts* may reduce the problem to the sum of a computable function and a non-singular integral. Also 'subtracting out' the singularity is simple in some cases, e.g.

$$\int_0^1 \frac{\cos x}{\sqrt{x}}\,dx = \int_0^1 \left\{ \frac{1}{\sqrt{x}} + \frac{\cos x - 1}{\sqrt{x}} \right\} dx = 2 + \int_0^1 \frac{\cos x - 1}{\sqrt{x}}\,dx.$$

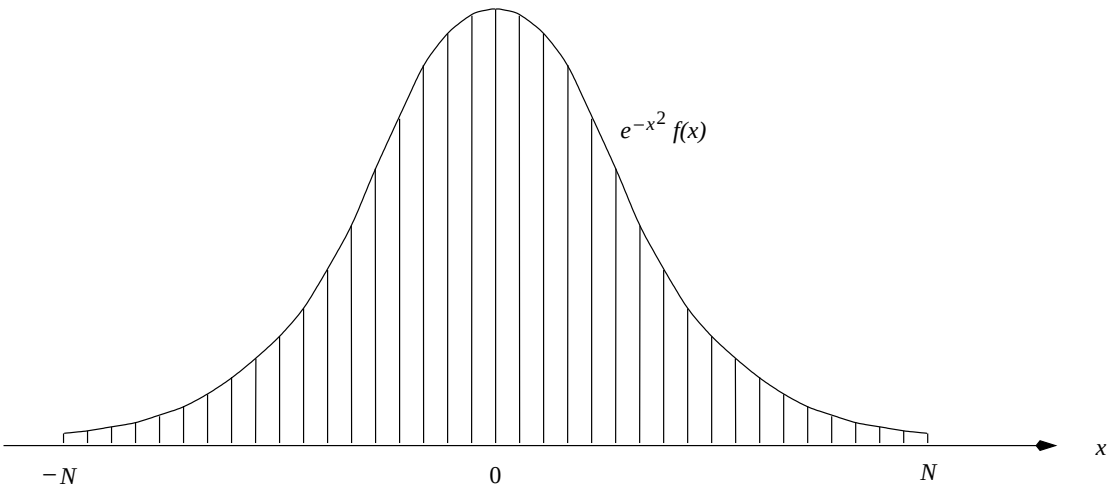
In this case $\cos x - 1 = O(x^2)$ so the integrand becomes $O(x^{3/2})$ which is less severe. This process can be repeated if necessary.

In general, as singularities are not polynomial-like, polynomial rules are unsuitable and indeed convergence is either not achieved or retarded.

A simple non-polynomial rule can be developed from a curious property of the trapezium rule. It turns out that, for a 'suitable' function $f(x)$, the best method for integrating

$$I = \int_{-\infty}^{\infty} e^{-x^2} f(x)dx \tag{2.10}$$

is to use the composite trapezium rule with $h = 1/N$, where N is a suitably large integer.



This can be truncated to $2N^2 + 1$ terms to give

$$I = h \sum_{i=-N^2}^{N^2} e^{-i^2h^2} f(ih) + O(e^{-A/h})$$

which has an exponential rate of convergence! This property holds provided that the function $f(x)$ has no singularities for any finite (real or complex) x. This is usually hard to establish, but if $f(x)$ does have

complex singularities (but none on the finite real line) then the rate of convergence is slower, but still of an exponential rate§.

The significance of this result is that we can transform the integral

$$I = \int_{-1}^{1} f(x)\ dx \tag{2.11}$$

to an infinite interval to which the composite trapezium rule can be applied. For example, using the transformation $x = \tanh u$ we get

$$dx = \operatorname{sech}^2 u\ du$$

so that (2.11) becomes

$$I = \int_{-\infty}^{\infty} \operatorname{sech}^2 u\ f(\tanh u)\ du.$$

Note that $sech^2 u \to e^{-u^2}$ as $u \to \pm\infty$. Choosing N sufficiently large that the integrand is negligible outside $[-N, N]$, setting $h = 1/N$ and applying the composite trapezium rule we get

$$h \sum_{i=-N^2}^{N^2} \operatorname{sech}^2(ih)\ f[\tanh(ih)]$$

which is a new (non-polynomial) quadrature rule with weights $w_i = h \operatorname{sech}^2(ih)$ and abscissae $x_i = \tanh(ih)$ and with an exponential rate of convergence. The great advantage of such a rule is that it can be used even when the integrand is singular at one or both end-points.

2.5 Multi-dimensional integration

2.5.1 Introduction

Consider the problem of evaluating

$$I = \int\int \ldots \int f(x_1, x_2, \ldots x_d)\ dx_1 dx_2 \ldots dx_d$$

where there are d integrals. Such problems often arise in practice, more commonly for 2 or 3 dimensions, but occasionally in 10 or 20 dimensions. As the problem becomes considerably more expensive to solve with each extra dimension, it is not surprising that different methods have been developed for different ranges of dimensions.

Product integration is useful for small dimensions, say 2 or 3 – see Section 2.5.3.

For circular (spherical) regions variable transformation methods, similar to that described in Section 2.4, have been used successfully.

The generalization of adaptive techniques to several dimensions leads to inefficiency; this is not too serious in 2 dimensions but increases with the dimension.

For irregular shaped regions, or for high numbers of dimensions, *Monte Carlo methods* are often used – see Section 2.5.5.

Gauss rules exist in more than one dimension, but they are rarely used. Unlike the one-dimensional case, Gauss rules in 2 dimensions do not always have positive weights. Also abscissae can lie outside the region of integration: this is highly undesirable because the integrand may not be defined.

§ There is a sense in which an exponential rate of convergence may be too fast, in that 'convergence testing' becomes more difficult.

2.5.2 Transformation to a hypercube

To devise integration rules it is convenient to consider standard regions, e.g. hypercube, simplex, hypersphere, surface of a hypersphere. Most regions can be transformed to standard regions. As an example, consider the d-dimensional integral

$$I = \int_{L_1}^{U_1} \int_{L_2(x_1)}^{U_2(x_1)} \int_{L_3(x_1,x_2)}^{U_3(x_1,x_2)} \cdots \int_{L_d(x_1,x_2,\ldots x_{d-1})}^{U_d(x_1,x_2,\ldots x_{d-1})} f(x_1, x_2, \ldots x_d)\ dx_1, dx_2, \ldots dx_d.$$

This fairly general region of integration is transformed to a hypercube by

$$x_i = \frac{U_i + L_i}{2} + y_i \frac{U_i - L_i}{2}, \quad i = 1, 2, \ldots d.$$

If $f(x_1, x_2, \ldots x_d) = g(y_1, y_2, \ldots y_d)$ then

$$I = \int_{-1}^{1} \int_{-1}^{1} \cdots \int_{-1}^{1} |\mathbf{J}|.g(y_1, y_2, \ldots y_d)\ dy_1, dy_2, \ldots dy_d$$

where the determinant of the Jacobian is

$$|\mathbf{J}| = \prod_{i=1}^{d} \frac{U_i - L_i}{2}.$$

2.5.3 Product rules

If $I_1 = [a, b]$, $I_2 = [c, d]$ are intervals then the Cartesian product $I_1 \times I_2$ denotes the rectangle $a \leq x \leq b$, $c \leq y \leq d$. Let $\mathbf{R}$, $\mathbf{S}$ denote the quadrature rules

$$\mathbf{R}f = \sum_{i=1}^{m} u_i f(x_i), \qquad \mathbf{S}f = \sum_{j=1}^{n} v_j f(y_j).$$

We define the product rule $\mathbf{R} \times \mathbf{S}$ by

$$(\mathbf{R} \times \mathbf{S})F = \sum_{i=1}^{m} \sum_{j=1}^{n} u_i v_j F(x_i, y_j).$$

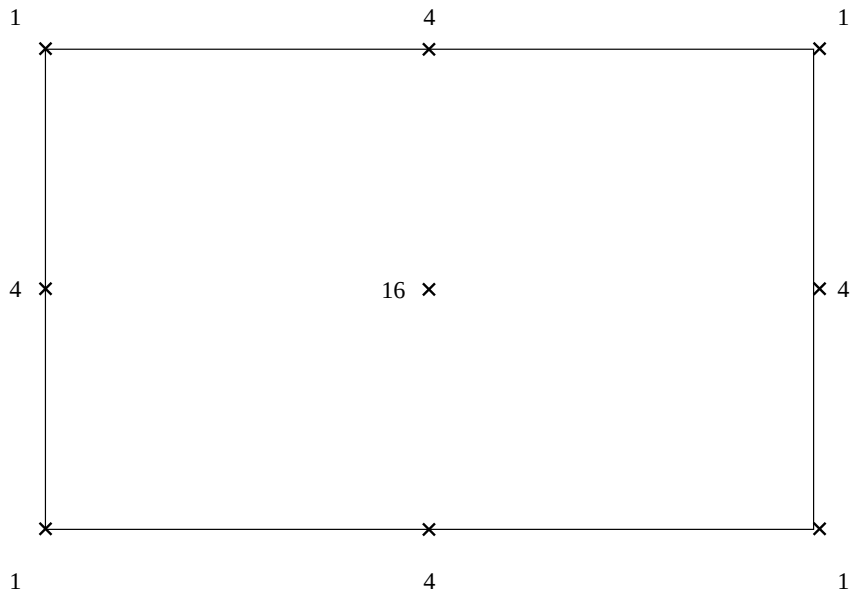
Theorem 2.6. *Let $\mathbf{R}$ integrate $f(x)$ exactly over the interval I_x and let $\mathbf{S}$ integrate $g(y)$ exactly over I_y, then $\mathbf{R} \times \mathbf{S}$ integrates $f(x)g(y)$ exactly over $I_x \times I_y$.*

Proof omitted.

Exercise 2b

Prove Theorem 2.6.

As an example of product integration, we can extend Simpson's rule to the rectangle using 9 abscissae with weights in the ratios shown.



The weights shown are multiplied by $hk/36$ where h, k are the step lengths in each dimension. The rule is exact for any linear combination of the monomials

$$\begin{array}{cccc} 1 & x & x^2 & x^3 \\ y & xy & x^2y & x^3y \\ y^2 & xy^2 & x^2y^2 & x^3y^2 \\ y^3 & xy^3 & x^2y^3 & x^3y^3 \end{array}$$

The idea generalizes to d dimensions but becomes increasingly inefficient and is rarely used for $d > 3$. Simple product integration is only suitable for integrals over rectangular (or hypercuboid) regions.

2.5.4 High dimensions

The problem with high dimensions is the *dimensional effect*: most methods behave like $n^{-k/d}$ as the number of dimensions d increases, and where k is a constant dependent on the method.

To improve the accuracy of a composite rule in one dimension we usually replace n subintervals by $2n$ subintervals. In d dimensions we replace n subintervals by $2^d n$ subintervals. It is especially advantageous to use the vertices, say of a hypercube, as abscissae because these points are re-used, e.g. the product trapezium rule or product Simpson rule. Consequently, a rule such as a Gauss rule is unsatisfactory when used in composite form because no abscissae lie on the boundary of the region.

Consider the product trapezium rule applied to a d-dimensional hypercube. This requires 2^d function evaluations. Halving the interval in each dimension requires a further $3^d - 2^d$ function evaluations. Note that $3^d - 2^d \simeq 3^d$ when d is large. Similarly, further halving of the interval in each dimension requires

approximately $5^d, 9^d, 17^d, \ldots$ extra function evaluations at each stage.

d	2^d	3^d	5^d	9^d	17^d
7	128	2187	8×10^4	5×10^6	10^8
10	1024	6×10^4	10^7	10^9	10^{12}
20	10^6	10^9	10^{14}	10^{19}	10^{25}

If one function evaluation takes 0.1 millisecond then 10^9 function evaluations take about 1 day of computation time on a sequential machine. Consequently for d large, special methods are required. High dimensional problems do exist, for example in physics, but fortunately high accuracy is not usually required: one significant figure or merely the order of magnitude is often sufficient. Serious consideration should be given to analytic methods such as changing the order of integration; even reducing the dimension by one can significantly affect the tractability of a problem. It should be noted that high dimensional integration is a problem well suited to highly-parallel computers.

2.5.5 Monte Carlo methods

While Monte Carlo methods cannot be used to obtain very high accuracy, they are not significantly less effective for high dimensional problems. The general idea of these methods is to choose pseudo-randomly generated abscissae and to use the average of the resulting function values multiplied by the volume of the integration region.

Note that pseudo-random numbers are not really random, but are deterministic sets of numbers which satisfy certain properties associated with 'randomness'.

Monte Carlo methods, and related number-theoretic methods, are probably the only practical methods for very high dimensional integration. The performance of Monte Carlo methods is very poor, even in one dimension, but the dimensional effect (i.e. the effect on efficiency of extending the method to increasing dimensions) is not so marked as with other methods.

Consider the one-dimensional case where $I = \int_0^1 f(x)\,dx$ and let $\{x_i\}$ be a set of pseudo-random variables, uniformly distributed on $[0,1]$. Then the *Strong Law of Large Numbers* states that

$$\text{Prob}\left\{ \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i) = I \right\} = 1,$$

i.e. using truly random abscissae and equal weights, convergence is *almost sure*.

Statistical error estimates can be found, but these depend on the variance σ^2 of the function $f(x)$ where

$$\sigma^2 = \int_0^1 \{f(x)\}^2\,dx - I^2.$$

As an example of the efficiency of Monte Carlo, suppose that $\sigma^2 = 1$. To obtain an estimate of I with error < 0.01 with 99% certainty requires 6.6×10^4 function evaluations. An extra decimal place with the same certainty requires 6.6×10^6 function evaluations. Clearly this is very inefficient in one dimension.

The advantage of Monte Carlo methods is that the statistical theory extends to d dimensions, and the error behaves like $n^{-1/2}$ rather than $n^{-k/d}$, i.e. it is (largely) independent of the number of dimensions. However, this does not mean that there is no dimensional effect: high dimensional functions tend to have larger variances (σ^2) than low dimensional ones. Nevertheless the dimensional effect is much less marked than for conventional interpolatory polynomial quadrature.

Because pseudo-random numbers are not really random, Monte Carlo methods have been refined by finding particularly advantageous sets of abscissae with appropriate properties. The Korobov-Conroy method uses

'optimal abscissae' derived from results in number theory and is often used instead of pseudo-random numbers for integration, although it is not always superior.

3. Non-linear Equations and Optimization

3.1 Non-linear equations in one variable

3.1.1 Simple iterative methods

Additional reference: E. Isaacson & H.B. Keller, *Analysis of numerical methods*, 1966. Wiley.

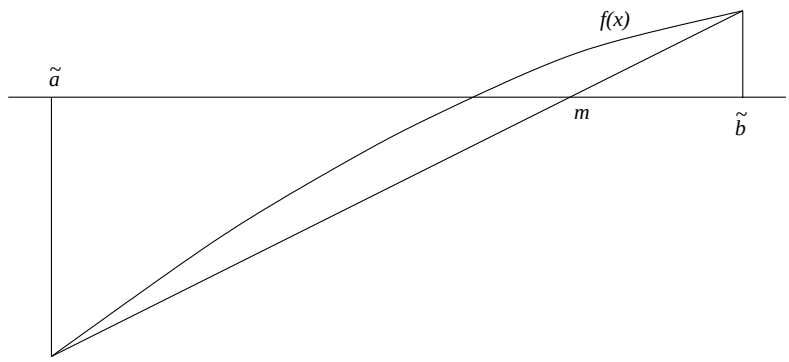
We consider the solution of the equation $f(x) = 0$ for a suitably smooth function f.

If $f(a)f(b) \leq 0$ then $f(x)$ must have a zero in the interval $[a, b]$ provided $f(x)$ is continuous. The *method of bisection* is a *robust* method for finding such a zero, i.e. a method guaranteed to converge although at a slow rate. The result of the algorithm is an interval $[\tilde{a}, \tilde{b}]$ containing the solution, and the algorithm terminates when $\tilde{b} - \tilde{a}$ is sufficiently small. The starting values are $\tilde{a} = a$, $\tilde{b} = b$ and the iteration can be summarized as follows

$$m = (\tilde{a} + \tilde{b})/2$$
$$\text{if } f(\tilde{a})f(m) \leq 0 \text{ then } \tilde{b} = m \text{ else } \tilde{a} = m$$

Suppose the calculation is performed in binary. The bisection method halves the length of the interval at each iteration. Therefore, at worst it will add one binary digit of accuracy at each step, so this iteration is only linearly convergent.

The method of bisection always chooses the mid-point of the current interval irrespective of the function values. An improvement can be achieved by using linear interpolation based on $f(\tilde{a})$ and $f(\tilde{b})$.



Thus the assignment of m in the above algorithm is replaced by

$$m = (f(\tilde{b})\tilde{a} - f(\tilde{a})\tilde{b})/(f(\tilde{b}) - f(\tilde{a}))$$

which is a weighted average of the function values and is unlikely to result in loss of significance since $f(\tilde{a})$, $f(\tilde{b})$ have opposite signs. This is sometimes called *regula falsi* or the *rule of false position.*

A further development is to abandon the use of intervals (and hence the guarantee of bracketting the zero of the function) by replacing the whole iteration with

$$x_{n+1} = (f(x_n)x_{n-1} - f(x_{n-1})x_n)/(f(x_n) - f(x_{n-1})) \tag{3.1}$$

which is known as the *secant method.* The rate of convergence is improved but there is now a possibility of loss of significance as $f(x_{n-1})$, $f(x_n)$ can have the same sign. Note that (3.1) can be written in the form

$$x_{n+1} = x_n - f(x_n)/\phi_n \tag{3.2}$$

where

$$\phi_n = (f(x_n) - f(x_{n-1}))/(x_n - x_{n-1})$$

and, since

$$\lim_{x_{n-1} \to x_n} \phi_n = f'(x_n)$$

we can derive the *Newton-Raphson* iteration

$$x_{n+1} = x_n - f(x_n)/f'(x_n) \tag{3.3}$$

from the secant method. Conversely, the secant method is merely a finite difference approximation to Newton-Raphson.

A practical algorithm, such as that of *Bus and Dekker* in the NAG library, may use a combination of methods: in this case linear interpolation, linear extrapolation and straightforward bisection. This method effectively achieves a compromise between efficiency and robustness.

Exercise 3a

> Devise an algorithm combining the *rule of false position* with the *method of bisection* to ensure that the bracketting interval is at worst halved for every two function evaluations. Decide if your algorithm is robust.

3.1.2 Fixed-point iteration theory

The Newton-Raphson formula (3.3) is an example of an iteration of the form

$$x_{n+1} = g(x_n) \tag{3.4}$$

where a fundamental requirement for convergence to the limit x is that

$$x = g(x),$$

i.e. that the limit x is a *fixed-point* of the function $g(x)$. Consequently methods of the form (3.4) are called *fixed-point iterative methods.*

We now consider the solution of $f(x) = 0$, where $f(x) = x - g(x)$ at the solution, using the iteration $x_{n+1} = g(x_n)$. We require that $g(x)$ is differentiable over some interval containing the fixed point x.

Theorem 3.1. *Suppose for the iteration*

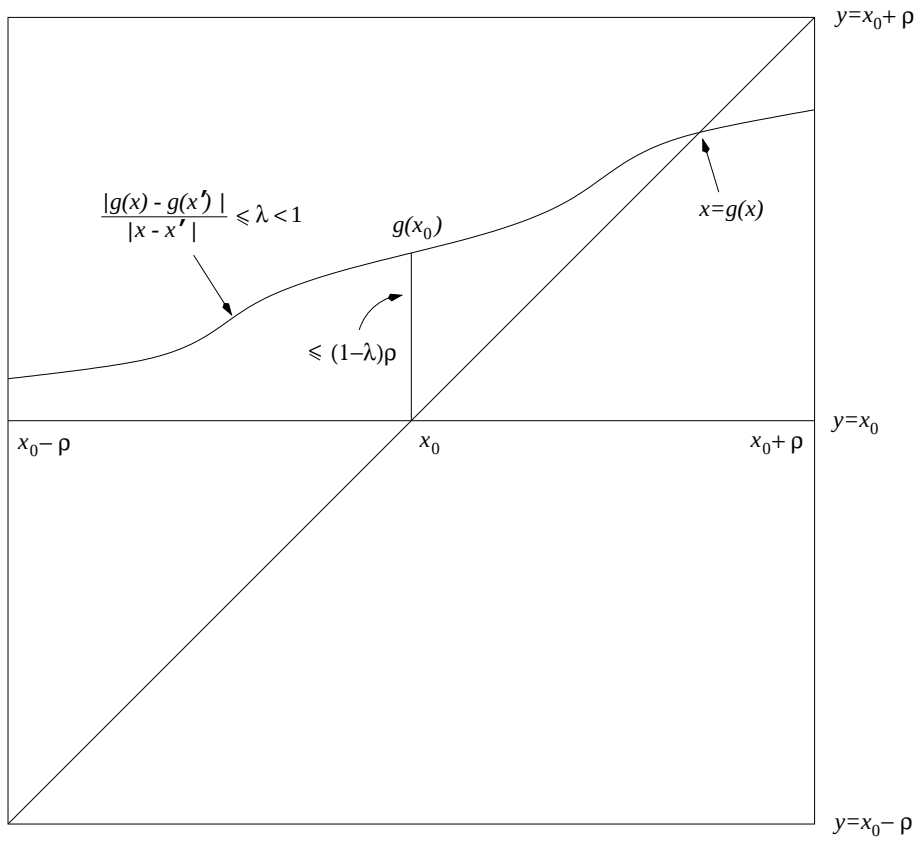
$$x_{n+1} = g(x_n)$$

that

$$|g(x) - g(x')| \leq \lambda |x - x'| \tag{3.5}$$

for all x and x' in an interval $I = [x_0 - \rho, x_0 + \rho]$, where the constant $\lambda < 1$ and $|x_0 - g(x_0)| \leq (1 - \lambda)\rho$. Under these conditions (1) all iterates lie in I, (2) the iterates converge, and (3) the solution is unique.

The proof is most simply explained graphically. Without loss of generality we can take $x_0 = 0$, which simplifies the diagram considerably: the line $y = x$ is shown as it is a convenient line with gradient 1.



Proof. (1)

$$\begin{aligned}|x_{n+1} - x_n| &= |g(x_n) - g(x_{n-1})| \\ &\leq \lambda |x_n - x_{n-1}| \\ &\leq \lambda^n |x_1 - x_0| \\ &\leq \lambda^n (1-\lambda)\rho\end{aligned}$$

because $x_1 - x_0 = g(x_0) - x_0$. Thus

$$\begin{aligned}|x_{n+1} - x_0| &\leq \sum_{m=0}^{n} |x_{m+1} - x_m| \\ &\leq \sum_{m=0}^{n} \lambda^m (1-\lambda)\rho \\ &\leq (1-\lambda^{n+1})\rho\end{aligned}$$

so x_{n+1} lies in the interval I.

(2)

$$\begin{aligned}|x_{n+p} - x_n| &\leq \sum_{m=n}^{n+p-1} |x_{m+1} - x_m| \\ &\leq \sum_{m=n}^{n+p-1} \lambda^m (1-\lambda)\rho \\ &\leq (1-\lambda^p)\lambda^n \rho\end{aligned}$$

which establishes convergence since the sequence $\{x_n\}$ is bounded, by part (1) of this proof.

(3) Suppose the root is not unique, i.e. suppose there exist roots x and y such that $x = g(x)$, $y = g(y)$ and $x \neq y$. Then

$$\begin{aligned} |x - y| &= |g(x) - g(y)| \\ &\leq \lambda |x - y| \\ &< |x - y| \end{aligned}$$

which is a contradiction.
¶

If in (3.5) x is close to x' and $g(x)$ is differentiable then it is reasonable to suppose that $\lambda \simeq |g'(x)|$. Indeed it may be shown that if $|g'(\xi)| < 1$ for a fixed point ξ then there exists an interval of convergence. Such a fixed point ξ is called a *point of attraction* for the iterative formula.

Exercise 3b

Show that, for Newton-Raphson, an interval of convergence always exists for a fixed point ξ.

3.1.3 Rate of convergence of Newton and secant methods

Let α be a root of $f(x) = 0$ and let $e_n = x_n - \alpha$.

From (3.3) for Newton-Raphson iteration we have

$$\begin{aligned} e_{n+1} &= e_n - f(\alpha + e_n)/f'(\alpha + e_n) \\ &= e_n - \frac{f(\alpha) + e_n f'(\alpha) + \frac{1}{2} e_n^2 f''(\alpha) + O(e_n^3)}{f'(\alpha) + e_n f''(\alpha) + O(e_n^2)} \\ &= e_n - e_n \frac{1 + \frac{1}{2} e_n f''(\alpha)/f'(\alpha) + O(e_n^2)}{1 + e_n f''(\alpha)/f'(\alpha) + O(e_n^2)} \\ &= e_n - e_n \{1 - \tfrac{1}{2} e_n f''(\alpha)/f'(\alpha) + O(e_n^2)\} \\ &= \tfrac{1}{2} e_n^2 f''(\alpha)/f'(\alpha) + O(e_n^3) \\ &= M e_n^2 + O(e_n^3) \end{aligned}$$

where $M = |f''(\alpha)/[2f'(\alpha)]|$, so there is quadratic convergence.

From (3.2) for the secant method, using the notation f_n for $f(\alpha + e_n)$, we have

$$\begin{aligned} e_{n+1} &= e_n - f_n \frac{e_n - e_{n-1}}{f_n - f_{n-1}} \\ &= \frac{e_n(f_n - f_{n-1}) - f_n(e_n - e_{n-1})}{f_n - f_{n-1}} \\ &= \frac{f_n e_{n-1} - f_{n-1} e_n}{f_n - f_{n-1}} \\ &= e_n e_{n-1} \frac{(\frac{f_n}{e_n} - \frac{f_{n-1}}{e_{n-1}})}{f_n - f_{n-1}}. \end{aligned}$$

Now

$$\frac{f_n}{e_n} = f'(\alpha) + \tfrac{1}{2} f''(\alpha) e_n + O(e_n^2)$$

so

$$e_{n+1} \simeq e_n e_{n-1} . \tfrac{1}{2} f''(\alpha) \frac{e_n - e_{n-1}}{f_n - f_{n-1}} \simeq e_n e_{n-1} \frac{f''(\alpha)}{2f'(\alpha)}$$

i.e.

$$|e_{n+1}| \simeq M |e_n| . |e_{n-1}|$$

where M is the same as for Newton-Raphson. Now the order of convergence p satisfies†

$$\left| \frac{e_{n+1}}{e_n^p} \right| = O(|e_n|^{1-p} |e_{n-1}|) = O\left(\left| \frac{e_n}{e_{n-1}^p} \right|^{1-p} \right)$$

provided $1 = -p(1-p)$ or $p^2 - p - 1 = 0$. The positive solution of this quadratic is $p = (\sqrt{5}+1)/2 \simeq 1.618$.

3.2 Zeros of polynomials

A polynomial

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \tag{3.6}$$

is of degree§ n if $a_n \neq 0$. If $p_n(x)$ has a factor $(x-\lambda)^r$ then we say that λ is a root of *multiplicity* r. The so-called 'fundamental theorem of algebra' states that $p_n(x)$ has precisely n real or complex roots if roots of multiplicity r are counted r times. We restrict the discussion to the problem of determining one or more roots of a polynomial with real coefficients.

If the coefficients of (3.6) are real then any complex roots appear in conjugate pairs, i.e. if $\mu + i\nu$ is a root then so is $\mu - i\nu$. The polynomial can be factorized into quadratic factors with real coefficients, except that if n is odd then one factor must be linear. It follows that any odd-degree polynomial with real coefficients must have at least one real root.

3.2.1 Condition of polynomial roots

Roots of polynomials of high degree are usually ill-conditioned. Assume an error δa_j in a particular coefficient a_j only. If $p_n(\alpha) = 0$ then the total derivative gives

$$\frac{\partial p_n}{\partial \alpha} . \delta\alpha + \frac{\partial p_n}{\partial a_j} . \delta a_j \simeq 0$$

from which we get a formula for relative error

$$\frac{\delta\alpha}{\alpha} \simeq -\frac{\alpha^{j-1}}{p_n'(\alpha)} . a_j . \frac{\delta a_j}{a_j} .$$

Multiple roots are ill-conditioned, and can be removed by finding the greatest common divisor (g.c.d.) of p_n and p_n', but this does not help with roots that are very close together and are also ill-conditioned.

Consider a *relatively* well-conditioned case where the roots are well separated, e.g.

$$\begin{aligned} p_n(x) &= (x+1)(x+2)\ldots(x+20) = 0 \\ &= x^{20} + 210x^{19} + \ldots + 20! = 0. \end{aligned}$$

Note that $n = 20$, and consider one root, say $\alpha = -16$. The relative error due to an error in, say, a_{19} is given by

$$\frac{\delta\alpha}{\alpha} \simeq \frac{16^{18}.210}{15!4!} . \frac{\delta a_j}{a_j} \simeq 3.2 \times 10^{10} . \frac{\delta a_j}{a_j} .$$

The relative error is magnified by about 10^{11}, although this is a relatively well-conditioned case.

† See Numerical Analysis I, Section 1.

§ The definition of *degree* is slightly different for the purposes of Section **3.2**.

3.2.2 Methods for finding zeros of polynomials

If only the real roots of a polynomial are required then search methods can be used, with the aid of various known properties, such as *Descartes' rule of signs.* Let n_+ be the number of positive real roots of $p_n(x)$. Let c be the number of changes in sign when the coefficients are taken in order, ignoring zero coefficients. Descartes' rule of signs states that

$$0 \leq c - n_+ = 2k$$

for some non-negative integer k. As an example, consider

$$p_4(x) = x^4 + 2x^2 - x - 1$$

for which $c = 1$. Descartes' rule gives $0 \leq 1 - n_+ = 2k$ which is only satisfied if $n_+ = 1$ and $k = 0$. Consequently there is precisely 1 positive real root.

Note that the sign changes in $p_n(-x)$ can be used to bound the number of negative real roots. By suitable shifts of origin, it is possible in principle to bracket real roots so that search methods can be used to locate them. Such methods are rarely used in practice, although various generalizations of search methods are often used as part of an algorithm for finding complex roots. For example, consider the equations

$$|a_n|x^n - |a_{n-1}|x^{n-1} - \ldots - |a_1|x - |a_0| = 0 \tag{3.7}$$

$$|a_n|x^n + |a_{n-1}|x^{n-1} + \ldots + |a_1|x - |a_0| = 0. \tag{3.8}$$

By Descartes' rule, as each equation has one sign change, there is exactly one real positive root, call these R for (3.7) and r for (3.8). A theorem due to Cauchy states that all roots of $p_n(x)$ lie in the annular region of the complex plane

$$r \leq |x| \leq R.$$

Other search methods could be used to subdivide this bounded region.

When a real root, or pair of complex conjugate roots of a polynomial are accurately determined, it is possible to factorize $p_n(x)$ to obtain a polynomial of lower degree that remains to be solved. This process is known as *deflation.* For example, consider the cubic

$$p_3(x) = x^3 - 6x^2 - 222x - 1160$$

which has precisely 1 real positive root, by Descartes' rule. We can search for this positive root by examining, say, $p_3(10^k)$ for various values of k until we have bounded it. Thus, since $p_3(0) < 0$, $p_3(1) < 0$, $p_3(10) < 0$, $p_3(100) > 0$ we deduce that this root lies in the interval $(10, 100)$ and can use any suitable method (e.g. bisection, Newton) to calculate it. This solution turns out to be $x = 20$. The factorization of $p_3(x)$ can then be performed numerically to obtain $(x - 20)(x^2 + 14x + 58)$. The *deflated polynomial* $x^2 + 14x + 58$ has the complex roots $-7 \pm 3i$.

Exercise 3c

> Devise a method for solving a cubic polynomial given that it has a double root. [Hint: Descartes' rule is not needed.]

Newton-Raphson iteration can be used to find real polynomial roots:

$$x_{n+1} = x_n - \frac{p_n(x_n)}{p_n'(x_n)}$$

and could in theory be used to find complex roots, but there is an overwhelming drawback. Suppose x_0 is a complex starting approximation, and that all roots are complex. If any iterate, x_k say, happens to be real then so will x_{k+1}, x_{k+2}, ... so convergence cannot occur. *Bairstow's method* overcomes this by using

a Newton-like iteration which converges to the coefficients of a quadratic factor, rather than to a solution itself. However, Bairstow's method is rarely used in modern software.

Müller's method is a generalization of the secant method which works well for polynomials, especially if complex roots are also required. The secant method is based on linear interpolation between two successive iterates. Müller' method uses quadratic interpolation between three iterates, say x_i, x_{i-1} and x_{i-2}. Suppose $q(x)$ is the quadratic that interpolates a function $f(x)$ at these points. The interpolating formula can be expressed in the form

$$q(x) = f(x_i) + c_i(x - x_i) + b_i(x - x_i)^2 \tag{3.9}$$

where

$$b_i = \frac{f(x_i)}{(x_i - x_{i-1})(x_i - x_{i-2})} + \frac{f(x_{i-1})}{(x_{i-1} - x_i)(x_{i-1} - x_{i-2})} + \frac{f(x_{i-2})}{(x_{i-2} - x_i)(x_{i-2} - x_{i-1})}$$

$$c_i = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} + b_i(x_i - x_{i-1}).$$

Applying one version of the quadratic formula‡ to (3.9) gives

$$x = x_i - \frac{2f(x_i)}{c_i \pm \sqrt{c_i^2 - 4b_i f(x_i)}}. \tag{3.10}$$

Müller's method uses the iteration

$$x_{i+1} = x_i + h_{i+1}$$

where h_{i+1} is the quotient on the right hand side of (3.10), with the sign chosen to maximize the absolute value of the denominator. The rate of convergence is better than the secant method, and almost quadratic, although no derivatives are required. For solution of polynomials, no starting approximation is needed since x_0, x_1 and x_2 may be almost arbitrarily chosen. If roots with small magnitude are found first then the build-up of rounding error is minimized, so it is reasonable to choose $x_0 = 0$. The square root in (3.10) ensures that complex values can arise naturally even if real starting values are used.

3.3 Unconstrained optimization

If $\mathbf{x}$ is a vector of n variables x_1, x_2,... x_n then the *unconstrained optimization problem* is usually expressed as the problem of determining

$$\min_{\mathbf{x}} F(\mathbf{x}) \tag{3.11}$$

which occurs at some point where

$$\nabla \mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{3.12}$$

where $\nabla \mathbf{F}$ is the vector of partial derivatives $\partial F / \partial x_i$. The maximization problem is conventionally treated as a minimization because

$$\max_{\mathbf{x}} F(\mathbf{x}) \equiv \min_{\mathbf{x}} \{-F(\mathbf{x})\}.$$

There is a close relationship between the unconstrained optimization problem and non-linear equations. Consider a single equation

$$f_j(\mathbf{x}) = 0.$$

We could equally well use the equation

$$[f_j(\mathbf{x})]^2 = 0$$

and, since squaring any non-zero quantity ensures that the result will be positive, any solution of the equation occurs at a minimum value of $[f_j(\mathbf{x})]^2$. A set of n non-linear equations can then be expressed as a

‡ See Numerical Analysis I, Section 1.

minimization problem of the form (3.11) where we take $F(\mathbf{x})$ to be a suitable combination of the f_j's such as

$$F(\mathbf{x}) = \sum_{j=1}^{n} [f_j(\mathbf{x})]^2. \tag{3.13}$$

Although special software is available for non-linear equations, as an alternative they can be solved using optimization software. Minimizing a function of the form (3.13) is referred to as the *non-linear least squares* problem.

Note that a *descent direction* is any direction $\mathbf{u}$ for which

$$\sum_{i=1}^{n} \frac{\partial F}{\partial x_i} u_i < 0.$$

3.3.1 Newton methods

Newton's method for solution of a system of n non-linear equations§ has been applied to the equations (3.12) and adapted for both non-linear least squares and general unconstrained minimization problems. For solution of the system of equations

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{3.14}$$

the basic Newton iteration is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}_k^{-1}\mathbf{f}(\mathbf{x}_k) \tag{3.15}$$

where $\mathbf{J}_k$ is the Jacobian matrix $(\partial f_i/\partial x_j)$. If Gaussian elimination is used to avoid inverting the matrix then the system of equations is

$$\mathbf{J}_k(\mathbf{x}_{k+1} - \mathbf{x}_k) = -\mathbf{f}(\mathbf{x}_k).$$

The method is rarely employed in practice without modification because it tends not to converge if the initial approximation is poor. Also, it can be very expensive to evaluate the Jacobian if the number of equations is large.

Continuation methods can be useful, e.g. instead of solving (3.14) solve a sequence of problems of the form

$$\mathbf{g}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}) - (1-t)\mathbf{f}(\mathbf{x}_0) = \mathbf{0}$$

where $\mathbf{x}_0$ is a starting approximation. The problem $\mathbf{g}(\mathbf{x}, 0) = \mathbf{0}$ has a trivial solution and $\mathbf{g}(\mathbf{x}, 1) = \mathbf{0}$ is the original problem. Solving a sequence of problems with gradually increasing values of t usually ensures a good starting approximation for the next problem.

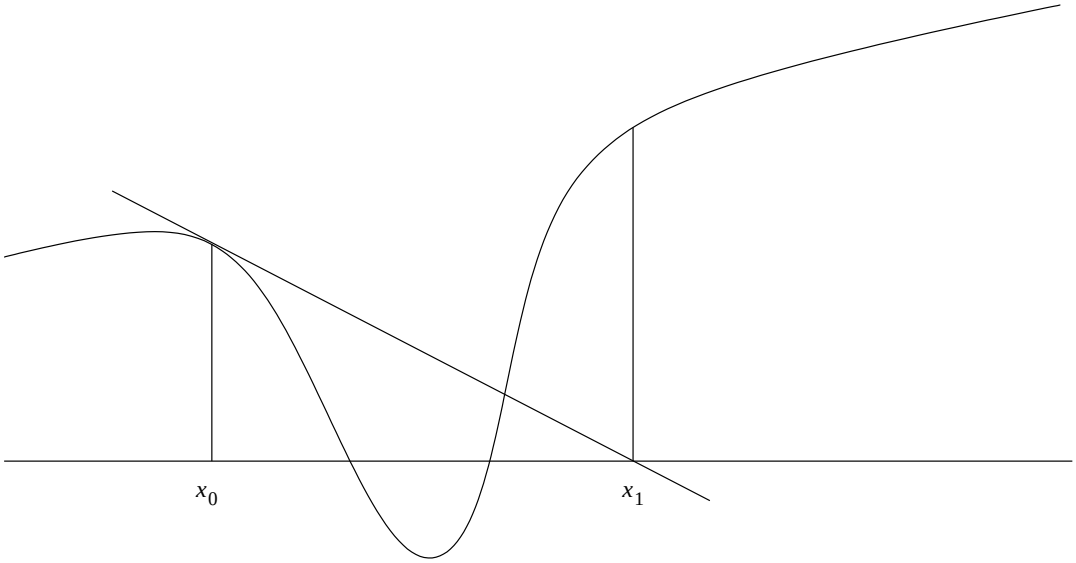
There are many variations on Newton's method, and we mention briefly the main differences between them. We write the Newton iteration in the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}_{k+1}$$

where $\mathbf{h}_{k+1}$ is the vector of increments.

In the *damped Newton method* the residuals are examined at each iteration to check that $\|\mathbf{f}(\mathbf{x})\|_2$ has decreased since the last iteration. If not, then $\mathbf{h}_{k+1}$ is rejected and $\frac{1}{2}\mathbf{h}_{k+1}$ is tried instead. This is repeated until a small enough increment vector is found so that the iteration produces an improvement. Note that the Newton direction is a descent direction so a small enough multiple of $\mathbf{h}_{k+1}$ can in principle be found but, in a pathological case, underflow might occur first, so it is necessary to have a fallback strategy if halving the increment a reasonable number of times does not work. The following diagram illustrates the principle by showing an example of the one-dimensional damped Newton method.

§ See Numerical Analysis I, Section 2.



Note that there are only two possible descent directions in one dimension, i.e. ± 1. Following the tangent produces a larger function value, but the Newton direction is downhill, so a sufficiently short increment in that direction must produce a reduction.

In the *modified Newton method* the cost of calculation is reduced by using the same matrix $\mathbf{J}$ for more than one iteration, on the grounds that successive Jacobians $\mathbf{J}_k$ and $\mathbf{J}_{k+1}$ will not differ very much close to the solution. If Gaussian elimination is used to solve the Newton equations then this costs $O(n^3)$ operations, plus the cost of recalculating the Jacobian. If the same Jacobian is used again then the cost of re-solving the same equations with a different right-hand side is only $O(n^2)$.

There is also a class of *matrix update methods* in which the iterations are calculated in the form (3.15) by first computing and storing $\mathbf{J}_0^{-1}$ but, at each subsequent iteration, performing only a *rank-one* or *rank-two update*† to the matrix such that the new inverse can be computed straightforwardly from the previous inverse without solving any equations.

Finally, *quasi-Newton methods* are those in which the iteration (3.15) is replaced by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{M}_k \mathbf{f}(\mathbf{x}_k)$$

where $\mathbf{M}_k$ is some other matrix corresponding to a descent direction.

3.3.2 Steepest descent methods

A different approach to optimization problems is offered by *steepest descent methods.* Let $\mathbf{u} = \{u_1, u_2, \ldots u_n\}$ be a vector representing a *search direction.* Consider the function

$$g(t) = F(\mathbf{x} + t\mathbf{u})$$

which is a function of the scalar t only. If $\mathbf{x}$ is an approximate solution then $g(0)$ is equal to the function value at the point $\mathbf{x}$. As t increases from zero then $g(t)$ represents the function value as the approximation moves from $\mathbf{x}$ in the direction $\mathbf{u}$. The *method of steepest descents* consists of minimizing $g(t)$ with repect to t along the steepest descent direction $\mathbf{u} = -\nabla\mathbf{F}$, i.e. if $\mathbf{x}_k$ is the kth approximation then

$$\mathbf{x}_{k+1} = \min_{t>0} g(t) = \min_{t>0} F(\mathbf{x}_k - t\nabla\mathbf{F}(\mathbf{x}_k)).$$

† The *rank* of a matrix is discussed in Section 4.

The minimization over the vector $\mathbf{x}$ is exchanged for a sequence of minimizations over the scalar t. The minimization over t is called a *line search*. This method is guaranteed to converge under suitable conditions, since all search directions are 'downhill' except at the solution, but may have a very slow rate of convergence. In 2 dimensions a minimum in a narrow valley is sufficient for the algorithm to perform poorly.

Practical methods use different *descent directions* and different methods for minimizing $g(t)$. If $\nabla\mathbf{F}$ can be calculated fairly cheaply then it is possible to solve $g'(t) = 0$ to find a minimum. If $\nabla\mathbf{F}$ is expensive to calculate then the line search can be performed using quadratic interpolation to obtain an approximate minimum.

4. Numerical Linear Algebra

4.1 Eigenproblems

Suppose an equation (of some sort) has a parameter λ and that this equation has a solution only for certain discrete values of λ, then the problem is called an *eigenproblem* and λ is said to be an *eigenvalue*. Eigenproblems are met in various fields, such as differential equations and matrix algebra.

The *linear matrix eigenvalue problem*

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0},$$

where $\mathbf{A}$ is an $n \times n$ matrix, is the most commonly encountered. The solution $\mathbf{x}$ is called an *eigenvector*. It may be shown that the eigenvalues are the zeros of $det(\mathbf{A} - \lambda\mathbf{I})$ which is a polynomial of degree n in λ and consequently has n roots. Therefore the matrix $\mathbf{A}$ has n eigenvalues, counting possible multiple roots. If the eigenvalues are distinct then there is a complete set of n linearly independent eigenvectors.

Strictly speaking, for each eigenvalue λ there is a corresponding *right eigenvector* $\mathbf{x}$ and a *left eigenvector* $\mathbf{y}^T$ such that

$$\mathbf{y}^T(\mathbf{A} - \lambda\mathbf{I}) = \mathbf{0}.$$

In practical problems, one or more eigenvalues are usually required, but eigenvectors are not always required. Consequently methods are designed so that eigenvectors need not be calculated unless needed.

4.1.1 Eigenvalues and eigenvectors

For a square matrix $\mathbf{A}$, we write

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{4.1}$$

$$\mathbf{y}^T\mathbf{A} = \lambda\mathbf{y}^T \tag{4.2}$$

where λ is an *eigenvalue* and $\mathbf{x}$ and $\mathbf{y}^T$ are eigenvectors.

Matrices $\mathbf{A}$ and $\mathbf{B}$ are said to be *similar* if there exists a non-singular matrix $\mathbf{R}$ such that $\mathbf{A} = \mathbf{R}^{-1}\mathbf{B}\mathbf{R}$. Writing

$$\mathbf{R}^{-1}\mathbf{B}\mathbf{R}\mathbf{x} = \lambda\mathbf{x}$$

and premultiplying by $\mathbf{R}$ we get

$$\mathbf{B}(\mathbf{R}\mathbf{x}) = \lambda(\mathbf{R}\mathbf{x})$$

or

$$\mathbf{B}\mathbf{z} = \lambda\mathbf{z}$$

where $\mathbf{z} = \mathbf{R}\mathbf{x}$. Therefore $\mathbf{A}$ and $\mathbf{B}$ have the same eigenvalues; the eigenvectors are transformed.

A matrix $\mathbf{Q}$ is *orthogonal* if $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, i.e. $\mathbf{Q}^{-1} = \mathbf{Q}^T$. If $\mathbf{A}$ is a symmetric matrix then the *orthogonal transformation* $\mathbf{Q}^T\mathbf{A}\mathbf{Q}$ is a similarity transformation that also preserves the symmetry. Note also that the product of orthogonal matrices is orthogonal, so orthogonal transformations can be built up from simpler orthogonal matrices.

Note that if λ_j and λ_k are two distinct eigenvalues,

$$\mathbf{y}_j^T \mathbf{A} \mathbf{x}_k = \lambda_k \mathbf{y}_j^T \mathbf{x}_k = \lambda_j \mathbf{y}_j^T \mathbf{x}_k$$

so that $\mathbf{y}_j^T \mathbf{x}_k = 0$, i.e. $\mathbf{y}_j^T$ and $\mathbf{x}_k$ are orthogonal.

Eigenvectors are usually normalized, e.g. such that $\|\mathbf{x}\| = 1$ if $\mathbf{x} \neq \mathbf{0}$.

We restrict the rest of this discussion to symmetric matrices because such problems are relatively well-conditioned and arise most often in practical situations. In this case all the eigenvalues are real, and the left and right eigenvectors are identical. Also, it follows that $\mathbf{x}_j^T \mathbf{x}_k = 0$ for $j \neq k$.

Exercise 4a

By considering both the transpose and the complex conjugate of (4.1) prove that, when $\mathbf{A}$ is a real symmetric matrix, the eigenvalues are real and the sets of left and right eigenvectors are identical.

4.1.2 Symmetric tridiagonal matrices

Problems with tridiagonal matrices are particularly easy to solve and lead to methods for solving general symmetric problems, since any symmetric matrix is *similar* to a symmetric tridiagonal matrix.

The eigenvalues of the *symmetric tridiagonal matrix*

$$\mathbf{T} = \begin{pmatrix} d_1 & e_1 & & & & \\ e_1 & d_2 & e_2 & & & \\ & e_2 & d_3 & e_3 & & \\ & & & \cdots & & \\ & & & e_{n-2} & d_{n-1} & e_{n-1} \\ & & & & e_{n-1} & d_n \end{pmatrix}$$

are the zeros of the polynomial $p_n(\lambda)$ resulting from the last determinant in the sequence

$$p_r(\lambda) = \det \begin{pmatrix} d_1 - \lambda & e_1 & & & & \\ e_1 & d_2 - \lambda & e_2 & & & \\ & e_2 & d_3 - \lambda & e_3 & & \\ & & & \cdots & & \\ & & & e_{r-2} & d_{r-1} - \lambda & e_{r-1} \\ & & & & e_{r-1} & d_r - \lambda \end{pmatrix}.$$

We also define $p_0(\lambda) = 1$. Note that $p_1(\lambda) = d_1 - \lambda$ and, using the rules for expanding determinants, there is a recurrence formula

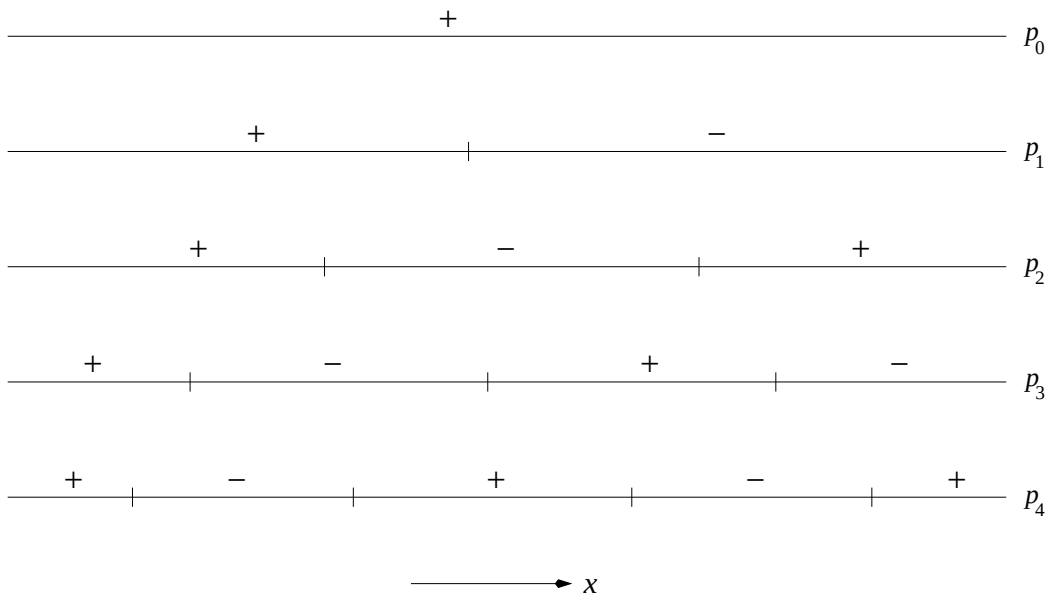
$$p_r(\lambda) = (d_r - \lambda) p_{r-1}(\lambda) - e_{r-1}^2 p_{r-2}(\lambda)$$

for $r = 2, 3, \ldots n$. Hence $p_n(\lambda)$ is easily computed.

For a given value of λ let $s(\lambda)$ denote the number of agreements in sign between successive members of the sequence $p_0(\lambda)$, $p_1(\lambda)$, $\ldots p_n(\lambda)$. If $p_r(\lambda) = 0$ then its sign is taken to be opposite to that of $p_{r-1}(\lambda)$, but the same as $p_{r+1}(\lambda)$.

Theorem 4.1: The Stürm sequence theorem. *The number of agreements in sign $s(\lambda)$ of successive members of the sequence $\{p_r(\lambda)\}$ is equal to the number of eigenvalues of $\mathbf{T}$ that are strictly greater than λ.*

The proof is easier to understand with the aid of a diagram.



Each line is a copy of the x axis representing a polynomial $p_r(x)$, with tick marks showing the zeros, and with the sign of the polynomial indicated.

Proof (in outline). By the method of induction, if $e_r \neq 0$ for $r = 1, 2, \ldots n-1$ then the zeros of $p_r(\lambda)$ strictly separate those of $p_{r+1}(\lambda)$, and the zeros of $p_n(\lambda)$ are distinct. Clearly $s(-\infty) = n$. The proof is completed by considering the effect of increasing λ away from $-\infty$, i.e. consider sliding a vertical line representing the value of λ across the diagram from left to right. The possibility of coincident λ values, for different r, is also easily explained. Note that if any $e_r = 0$ then the determinant can be factorized into two determinants of the same form but lower degree, so the same proof applies.
¶

If $s(\lambda_1) = k$ and $s(\lambda_2) = k + 1$ then one and only one eigenvalue of $\mathbf{T}$ lies in the interval $[\lambda_1, \lambda_2]$. This property forms the basis of algorithms that can be used (1) to find all the eigenvalues of $\mathbf{T}$, (2) to count how many eigenvalues lie in a certain interval, (3) to decide whether the matrix $\mathbf{T}$ is singular, or (4) to decide whether the matrix $\mathbf{T}$ is positive definite.

4.1.3 Calculation of eigenvectors

Let $\mathbf{T}$ be a tridiagonal matrix. The calculation of an eigenvector $\mathbf{x}$ requires solution of the linear equations

$$(\mathbf{T} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$$

except that $\mathbf{x}$ needs to be normalized. This means that the equations are over-determined. They can be solved by a method known as *inverse iteration* in which successive approximations are calculated by solving

$$(\mathbf{T} - \tilde{\lambda}\mathbf{I})\mathbf{x}_{n+1} = \mathbf{x}_n.$$

The important thing to notice is that λ has been replaced by an approximation $\tilde{\lambda}$ because otherwise the matrix $\mathbf{T} - \lambda\mathbf{I}$ would be singular. However, if $\tilde{\lambda}$ is a good approximation then the iteration converges very fast, but extra precision is normally needed.

The calculation of eigenvalues and eigenvectors each have computational complexity n^3.

4.1.4 Methods for reduction to tridiagonal form

As mentioned in Section 4.1.1, an orthogonal matrix $\mathbf{Q}$ can be built up as a product of simpler orthogonal matrices. One useful orthogonal matrix is called a *plane rotation* and is a variant of the unit matrix of the

form

$$\mathbf{R}(i,j) = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & \cos\theta & & & & \sin\theta & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & -\sin\theta & & & & \cos\theta & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix}$$

where the elements that are different from the unit matrix appear in the ith and jth rows and columns. Multiplying by the matrix $\mathbf{R}(i,j)$ is geometrically equivalent to a rotation through an angle θ in the plane (i,j). The advantage of a transformation based on $\mathbf{R}(i,j)$ is that it can be used to operate on selected elements of a matrix only.

Exercise 4b

Verify that $\mathbf{R}(i,j)$ is an orthogonal matrix.

Givens' method (1954) uses a succession of plane rotations to reduce a symmetric matrix to symmetric tridiagonal form. Consider the matrix $\mathbf{A}^{(m)} = (a_{ij}^{(m)})$ which represents the state of the transformation after m plane rotations. The effect on the (i,j)th element of forming the matrix $\mathbf{A}^{(m+1)} = \mathbf{R}(i+1,j)\mathbf{A}^{(m)}\mathbf{R}(i+1,j)^T$ is given by

$$a_{ij}^{(m+1)} = -a_{i,i+1}^{(m)} \sin\theta + a_{ij}^{(m)} \cos\theta.$$

We can choose to set $a_{ij}^{(m+1)}$ to zero, and the required value of θ is given by

$$\tan\theta = \frac{a_{ij}^{(m)}}{a_{i,i+1}^{(m)}}.$$

As symmetry is preserved, the calculations need only be performed on the lower triangle of the matrix. The method proceeds by annihilating elements by columns, starting with $a_{31}, \ldots a_{n1}$ by rotations in the planes $(2,3), \ldots (2,n)$. In the ith column the elements $a_{i+2,i}, \ldots a_{ni}$ are annihilated by rotations in the planes $(i+1, i+2), \ldots (i+1, n)$.

Householder's method (1958) improved on Givens' by annihilating a whole column at a time (and, by symmetry, a row also) by using a different orthogonal transformation.

4.1.5 Factorization methods

Rutishauser's qd algorithm (1954) was an entirely new but often unstable method which faded into obscurity. However, a development from this was his LR algorithm (1958) which uses the LR (or LU) factorization† and is based on the following property. Let

$$\mathbf{A} = \mathbf{LR}$$

where $\mathbf{L}$ and $\mathbf{R}$ are lower and upper (left and right) triangular factors, and then form the matrix

$$\mathbf{B} = \mathbf{RL}.$$

Now $\mathbf{R} = \mathbf{L}^{-1}\mathbf{A}$ so

$$\mathbf{B} = \mathbf{L}^{-1}\mathbf{AL}$$

so that $\mathbf{A}$ and $\mathbf{B}$ are similar. Under certain conditions, using the iteration

$$\begin{aligned} \mathbf{A}_m &= \mathbf{L}_m\mathbf{R}_m \\ \mathbf{A}_{m+1} &= \mathbf{R}_m\mathbf{L}_m \end{aligned}$$

the sequence $\{\mathbf{A}_m\}$ converges to an upper triangular matrix with the eigenvalues on the main diagonal. Unfortunately, the LR factorization can break down, causing the method to fail.

Francis' QR algorithm (1961) is a stable development of the LR method. The iteration is of the form

$$\begin{aligned} \mathbf{A}_m &= \mathbf{Q}_m\mathbf{R}_m \\ \mathbf{A}_{m+1} &= \mathbf{R}_m\mathbf{Q}_m \end{aligned}$$

but now the factor $\mathbf{Q}_m$ is an *orthogonal matrix* chosen such that the factorization does not break down. A further development of the algorithm, using shifts of origin‡, has a built-in convergence acceleration technique, and is generally regarded as the best method for symmetric problems (including tridiagonal problems)§.

4.2 Rank and generalized inverse matrices

Any matrix, of whatever shape, has a *rank*. In order to introduce this concept, and its significance in numerical linear algebra, we will examine an $n \times n$ diagonal matrix since we can write down its inverse, if it exists. We first assume that exact arithmetic is available, and consider the matrix

$$\mathbf{D} = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix}$$

which is non-singular if $d_i \neq 0$ for all i. The inverse is then $diag\{1/d_1, 1/d_2, \ldots 1/d_n\}$ and we would say that the matrix $\mathbf{D}$ has *rank* n. If, say, only the first r elements were non-zero then there would be no inverse matrix

† Sometimes used for solution of linear equations – see Numerical Analysis I, Section 2.

‡ Shifting the origin is a very old idea, applicable to most practical algorithms for eigenvalue problems, and is attributed to Schröder (1870).

§ However, recent work has focussed on the related problem of determining *singular values* of a *bidiagonal matrix* - see 4.2.1. Demmel & Kahan (1990) used a version of QR to show that these could be determined with maximum relative accuracy, even for very small singular values. Fernando & Parlett (1994) improved on Demmel & Kahan's algorithm by a simple development of Rutishauser's qd algorithm (1954) which does not use orthogonal transformations at all.

of order n. However, we could invert the sub-matrix of order r and form the inverse $diag\{1/d_1, 1/d_2, \ldots 1/d_r\}$. We would then say that the matrix $\mathbf{D}$ has rank r. The matrix

$$\mathbf{D}_r^+ = \begin{pmatrix} 1/d_1 & & & & & & \\ & 1/d_2 & & & & & \\ & & \ddots & & & & \\ & & & 1/d_r & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \end{pmatrix}$$

can then be regarded as an r-dimensional approximation to the inverse of $\mathbf{D}$.

Now consider the effect of using floating-point arithmetic. Suppose that each of the elements of $\mathbf{D}$ is being computed by some lengthy calculation, and that it is known that only the first r elements are non-zero but that d_r is very small and may underflow during the calculation. It is therefore uncertain whether the result will be a matrix effectively of rank r or $r-1$. To be on the safe side one could assume that $d_r = 0$ and form

$$\mathbf{D}_{r-1}^+ = \begin{pmatrix} 1/d_1 & & & & & & \\ & 1/d_2 & & & & & \\ & & \ddots & & & & \\ & & & 1/d_{r-1} & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \end{pmatrix}$$

In fact this $\mathbf{D}_{r-1}^+$ is the best l_2 approximation of order $r-1$ to the true inverse of order r, which is itself an approximate inverse of $\mathbf{D}_n$. Since $\mathbf{D}_{r-1}^+$ and $\mathbf{D}_r^+$ are 'approximations' to an n-dimensional inverse that does not actually exist, it is more correct to refer to these as *generalized inverses* instead†.

For a general $n \times n$ real matrix $\mathbf{A}$ the effective rank is less easy to determine but, if we can discover what it is, then we can form a generalized inverse of any matrix. For solution of the linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

we need to be able to determine rank and, in a doubtful case, judge whether or not to reduce the rank in order to improve numerical stability. The method of *singular value decomposition* provides not only a suitable way of measuring rank, but also a stable algorithm for solution of linear equations of all types.

4.2.1 Singular Value Decomposition (SVD)

Recall that, for any non-singular square matrix $\mathbf{A}$, the matrix $\mathbf{A}^T\mathbf{A}$ is symmetric and *positive definite*‡. This result generalizes as follows: for any $m \times n$ real matrix $\mathbf{A}$, the $n \times n$ matrix $\mathbf{A}^T\mathbf{A}$ is symmetric and *positive semi-definite*§.

Consider the eigenvalue problem

$$\mathbf{A}^T\mathbf{A}\mathbf{x}_j = \lambda_j \mathbf{x}_j$$

† The term *pseudo-inverse* is also used.

‡ A symmetric matrix $\mathbf{B}$ is *positive definite* if $\mathbf{x}^T\mathbf{B}\mathbf{x} > 0$ for any vector $\mathbf{x} \neq \mathbf{0}$ – see Numerical Analysis I, Section 2.

§ A symmetric matrix $\mathbf{B}$ is *positive semi-definite* if $\mathbf{x}^T\mathbf{B}\mathbf{x} \geq 0$ for any vector $\mathbf{x}$.

where the eigenvalues $\lambda_1,\ \lambda_2,\ \ldots\lambda_n$ are in decreasing order. As the matrix is positive semi-definite, it follows that the eigenvalues are non-negative, so we can write $\sigma_j = \sqrt{\lambda_j}$. Define the $n \times n$ matrix $\mathbf{W} = diag\{\sigma_1, \sigma_2, \ldots \sigma_n\}$. A set of orthonormal eigenvectors $\{\mathbf{x}_j\}$ can be found, and we can write these as an orthogonal matrix $\mathbf{V}$, i.e.

$$\mathbf{V} = (\,\mathbf{x}_1 \quad \mathbf{x}_2 \quad \ldots \quad \mathbf{x}_n\,)\,.$$

The values

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0 \tag{4.3}$$

are called the *singular values* of the $m \times n$ matrix $\mathbf{A}$†. Golub & Kahan (1965) showed that $\mathbf{A}$ may be factorized in the form

$$\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{V}^T \tag{4.4}$$

where $\mathbf{U}$ is an $m \times n$ matrix for which $\mathbf{U}^T\mathbf{U} = \mathbf{I}_{n\times n}$. The factorization (4.4) is called the *singular value decomposition (SVD)* of the matrix $\mathbf{A}$.

Stable algorithms for SVD have been developed, and utilize various eigenvalue methods such as Householder reduction and the QR algorithm. Computing the singular values (4.3) gives immediate information about the effective rank of a matrix since that is determined by the number of non-zero singular values. Further, if some singular values are 'small' then we can set them to zero in order to use a generalized inverse of lower dimension, provided we have some idea of what is meant by 'small' in this sense. The formula (4.4) provides a method for solving linear equations by a lower dimensional approximation without necessarily forming the generalized inverse explicitly.

4.3 Matrix norms

To any vector norm $\|\mathbf{x}\|_v$ there corresponds a matrix norm $\|\mathbf{A}\|_M$ such that

$$\|\mathbf{A}\|_M = \sup_{\text{all } \mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_v}{\|\mathbf{x}\|_v} = \sup_{\|\mathbf{x}\|_v=1} \|\mathbf{A}\mathbf{x}\|_v.$$

The norm $\|\mathbf{A}\|_M$ is *induced* by the norm $\|\mathbf{x}\|_v$. Induced matrix norms (also known as *compatible norms*) are usually given the same name as the corresponding vector norm, in particular the l_p norm of a matrix is that induced by the l_p norm of a vector. It is usual to write

$$\|\mathbf{A}\|_p = \sup_{\|\mathbf{x}\|_p=1} \|\mathbf{A}\mathbf{x}\|_p \tag{4.5}$$

where it is understood that the norms are different on each side of the equation. Note that (4.5) is the definition of a matrix norm, not an actual formula. The formulae derived from this definition, in the commonly used cases, are as follows.

$$\begin{array}{ccc} & \text{Vector norm} & \text{Matrix norm} \\ l_1 & \sum_{i=1}^n |x_i| & \max_j \sum_{i=1}^n |a_{ij}| \\ l_2 & \sqrt{\sum_{i=1}^n x_i^2} & \sqrt{\max \text{ eigenvalue of } \mathbf{A}^T\mathbf{A}} \\ l_\infty & \max_i |x_i| & \max_i \sum_{j=1}^n |a_{ij}| \end{array} \tag{4.6}$$

It follows that $\|\mathbf{A}\|_1 = \|\mathbf{A}^T\|_\infty$.

For any induced matrix norm, *Schwarz's inequality* holds:

$$\|\mathbf{A}\mathbf{B}\| \leq \|\mathbf{A}\|.\|\mathbf{B}\|.$$

† It follows that the *singular values* of a square *bidiagonal matrix* are the *eigenvalues* of a *symmetric positive definite tridiagonal matrix* - see the footnote at the end of 4.1.5.

This inequality also holds if $\mathbf{A}$ or $\mathbf{B}$ is replaced by a vector using the compatible vector norm.

4.4 Solution of linear equations

4.4.1 Condition of the problem

First consider the linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\mathbf{A}$ is a square matrix. We can check an approximate solution $\hat{\mathbf{x}}$ by forming the vector of *residuals*

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$$

and, if these residuals are small, it tells us that the equations are being satisfied, but this does not tell us anything about the size of the errors in $\hat{\mathbf{x}}$.

Define the error vector $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$. Note that $\mathbf{A}\mathbf{e} = \mathbf{A}\mathbf{x} - \mathbf{A}\hat{\mathbf{x}} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}} = \mathbf{r}$ so $\mathbf{e} = \mathbf{A}^{-1}\mathbf{r}$. Taking norms and using Schwarz's inequality, we have

$$\begin{aligned} \|\mathbf{e}\| &= \|\mathbf{A}^{-1}\mathbf{r}\| \\ &\leq \|\mathbf{A}^{-1}\|.\|\mathbf{r}\| \end{aligned}$$

so the relative error in $\mathbf{x}$ is represented by

$$\frac{\|\mathbf{e}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}^{-1}\| \frac{\|\mathbf{r}\|}{\|\mathbf{x}\|}.$$

But $\|\mathbf{b}\| = \|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\|.\|\mathbf{x}\|$ so we can replace $\|\mathbf{x}\|$ on the right-hand side and form an inequality in which the right-hand side is computable, i.e.

$$\frac{\|\mathbf{e}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}\|.\|\mathbf{A}^{-1}\| \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \tag{4.7}$$

where $K = \|\mathbf{A}\|.\|\mathbf{A}^{-1}\|$ is the *condition number* for the matrix $\mathbf{A}$. Note that K depends on the norm being used. If we choose the l_2 norm then, using (4.6), we see that SVD provides a simple formula for K, i.e.

$$K = \frac{\sigma_1}{\sigma_n}, \tag{4.8}$$

where σ_1 and σ_n are the largest and smallest singular values of $\mathbf{A}$. In the case of a singular, or nearly singular matrix, we can replace (4.8) by $K = \sigma_1/\sigma_r$ in order to examine the condition of an r-dimensional approximation.

Although (4.7) is a *sharp* inequality, it is nevertheless a worst case result, so the exact value of the condition number K cannot be taken too seriously. However, if $K > 1/macheps$ then accuracy is likely to increase if some singular values are discarded‡. In general some judgement, either human or algorithmic, is required to decide how many singular values, if any, to discard for a given matrix $\mathbf{A}$.

The ideas in this section can be generalized to $m \times n$ matrices and to complex matrices also.

4.4.2 Over-determined and under-determined linear equations

Consider linear equations with an $m \times n$ matrix $\mathbf{A}$ where $m \neq n$.

In the case $m > n$ there are more equations than unknowns, i.e. the solution is *over-determined*. Such a case may arise where a simple data model, e.g. a quadratic curve, is to be fitted to a large number of observed data points. Forming the SVD of $\mathbf{A}$ provides the opportunity to solve the equations directly. Typically the

‡ Recall that *macheps* or *machine epsilon* is the smallest $\varepsilon > 0$ such that $(1+\varepsilon)^* > 1$ for the floating-point precision in use – see Numerical Analysis I.

rank will be n, although a smaller value is possible. The solution obtained by SVD is in fact the best l_2 approximation, i.e. the linear least-squares solution.

In the case $m < n$ there are not enough equations to define a unique solution, i.e. the solution is *under-determined.* All possible solutions form a vector space of dimension r, where $r \leq n - m$. This problem does not often arise in practice, but it is interesting that even this case can be fully solved by SVD. The unique solution produced directly from the SVD method is the so-called *minimal* l_2 approximation, i.e. the r-dimensional solution vector $\mathbf{t}$ which minimizes $\|\mathbf{t}\|_2$ over the solution space. Furthermore, a set of r *basis vectors* can be extracted from the eigenvector matrix $\mathbf{V}$ such that any solution of the equations can be expressed as a linear combination of this basis.

5. Differential Equations

5.1 Preliminary theory

We begin by contrasting *linear differential equations* with *linear difference equations.* Consider first a linear *ordinary differential equation* (or *ODE*) with constant coefficients

$$a_n y^{(n)} + a_{n-1} y^{(n-1)} + \ldots + a_1 y' + a_0 y = 0. \tag{5.1}$$

This is termed *homogeneous* because the right-hand side is zero. Assuming a solution of the form $y = e^{\beta x}$ and substituting this in (5.1) gives the *characteristic equation*

$$a_n \beta^n + a_{n-1} \beta^{n-1} + \ldots + a_1 \beta + a_0 = 0 \tag{5.2}$$

which has n solutions, say β_1, β_2, $\ldots \beta_n$. If all the β_i are distinct then the general solution to (5.1) has the form

$$y(x) = \sum_{i=1}^{n} C_i e^{\beta_i x} \tag{5.3}$$

where each C_i is an arbitrary constant. In the case that, say, β_j is a triple root of the characteristic polynomial, then (5.3) is modified by including terms in $e^{\beta_j x}$, $xe^{\beta_j x}$ and $x^2 e^{\beta_j x}$. Note also that, as the roots can appear in complex conjugate pairs, sin and cos terms can appear in the general solution.

If the differential equation is non-homogeneous, i.e.

$$a_n y^{(n)} + a_{n-1} y^{(n-1)} + \ldots + a_1 y' + a_0 y = g(x) \tag{5.4}$$

and $p(x)$ is any particular solution of (5.4), then the general solution of (5.4) is

$$y(x) = p(x) + \sum_{i=1}^{n} C_i e^{\beta_i x}. \tag{5.5}$$

Now consider a homogeneous linear *difference equation* with constant coefficients

$$a_n y_{N+n} + a_{n-1} y_{N+n-1} + \ldots + a_1 y_{N+1} + a_0 y_N = 0. \tag{5.6}$$

Assuming a solution of the form $y_N = \beta^N$, substituting this in (5.6) and dividing through by β^N gives

$$a_n \beta^n + a_{n-1} \beta^{n-1} + \ldots + a_1 \beta + a_0 = 0. \tag{5.7}$$

Note that the *characteristic equation* (5.7) is identical to (5.2). If all the solutions β_i are distinct then the general solution to (5.6) can be shown to have the form

$$y_N = \sum_{i=1}^{n} c_i \beta_i^N \tag{5.8}$$

for all N, where each c_i is an arbitrary constant. In the case that, say, β_j is a triple root of the characteristic polynomial, then (5.8) is modified by including terms in β_j^N, $N\beta_j^N$ and $N^2\beta_j^N$. As with differential equations, sin and cos terms can appear in the general solution due to complex conjugate roots of (5.7).

If the difference equation is non-homogeneous, i.e.

$$a_n y_{N+n} + a_{n-1} y_{N+n-1} + \ldots + a_1 y_{N+1} + a_0 y_N = b_N \tag{5.9}$$

and p_N is any particular solution of (5.9), then the general solution of (5.9) is

$$y_N = p_N + \sum_{i=1}^{n} c_i \beta_i^N. \tag{5.10}$$

5.2 Initial value problems

We will consider the first order *initial value problem*

$$y' = f(x, y), \qquad y(x_0) = y_0 \tag{5.11}$$

in which the solution is known at an initial point $x = x_0$. Numerical methods generally progress the solution through a finite sequence of points x_0, x_1, $\ldots$ x_n. Note that the solution at $x = x_n$ can be written

$$y(x_n) = \int_{x_0}^{x_n} f(x, y(x))\ dx.$$

As with quadrature, the total number of evaluations of the function $f(x, y)$ is a good measure of the efficiency of a method.

By convention we write $y(x_n)$ for the true solution at the point x_n and let y_n denote a computed approximation to it.

Methods are characterized as *one step methods* in which the computed value y_{n+1} depends only on data evaluated at the point x_n, and *multistep methods* in which the computed y_{n+1} depends on more than one previous x_i value.

5.2.1 Euler's method

We begin with *one step methods*, of which Euler's is the simplest. It is of some practical value, but it is particularly useful in order to introduce the important concepts of *local error* and *global error*.

Methods for numerical solution, and error estimates, can typically be derived from the Taylor series

$$y(x_n + h) = y(x_n) + hy'(x_n) + \frac{h^2}{2!} y''(x_n) + \ldots + \frac{h^k}{k!} y^{(k)}(x_n) + \ldots \tag{5.12}$$

but note that the evaluation of $y^{(k)}(x_n)$ requires partial derivatives since it involves differentiation of $f(x, y(x))$. However, by ignoring terms $O(h^2)$ in (5.12), and using the equation (5.11), we obtain immediately Euler's formula

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{5.13}$$

with the error

$$E = \frac{h^2}{2} y''(\xi_n).$$

Because it relates only to a single step, this is termed the *local error* of the method. A method is said to be of *order* k if the local error is $O(h^{k+1})$, so Euler's method is of order 1.

Let e_n represent discretization error, i.e.

$$e_n = y(x_n) - y_n$$

for $n \geq 0$ and assume that $e_0 = 0$. By an analysis omitted here§ it may be shown that the error over n steps builds up in such a way that

$$|e_n| \leq \frac{hY}{2L}(e^{(x_n - x_0)L} - 1)$$

where Y and L are constants, i.e. for fixed x_n the rate of convergence is only linear. We say that the *global error* is $O(h)$, whereas the *local error* is $O(h^2)$. There is an analogy here with quadrature rules which have a lower order of convergence when used in composite form†.

Usually Euler's method requires a very small step size h to give reasonable accuracy, so is not often used in practice. Euler's method is the simplest of the family of *Taylor series methods*, but Taylor series methods of higher order are rarely used because of the partial derivatives required.

5.2.2 Runge-Kutta methods

Runge-Kutta methods, also derived from Taylor series, avoid the need for partial derivatives at the expense of extra function evaluations.

Consider again Euler's method

$$y_{n+1} = y_n + k_1, \qquad k_1 = hf(x_n, y_n) \tag{5.14}$$

where k_1 is an increment based on an estimate of y' at the point (x_n, y_n). Having computed k_1 it is possible to re-estimate y' based on the point $(x_{n+1}, y_n + k_1)$. This gives an increment, k_2 say, where

$$k_2 = hf(x_n + h, y_n + k_1). \tag{5.15}$$

The Runge-Kutta method of order 2 (sometimes called RK2) uses the average of these two increments, i.e.

$$y_{n+1} = y_n + \tfrac{1}{2}(k_1 + k_2) \tag{5.16}$$

where k_1, k_2 are defined by (5.14), (5.15) respectively. Formally, Runge-Kutta methods are derived from Taylor's theorem by finding the coefficients of a formula of a certain type in such a way that the error term is of as high an order as possible. The local error term for the formula (5.16) is

$$\frac{h^3}{12}(f_{xx} + 2ff_{xy} + f^2 f_{yy} - 2f_x f_y - 2ff_y^2) + O(h^4)$$

where f_x, f_{yy}, *etc* denote partial derivatives. The local error is of order h^3 compared with h^2 for Euler's method, but the increased complexity of the error term means that error estimates are difficult to obtain. Although RK2 requires two evaluations of $f(x, y)$ per step, larger steps can be taken than for Euler's method.

The most commonly used Runge-Kutta scheme is that of order 4 (called RK4) which is

$$y_{n+1} = y_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1) \\ k_3 &= hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2) \\ k_4 &= hf(x_n + h, y_n + k_3). \end{aligned}$$

§ See, for example, Conte & de Boor.
† See Theorem 2.5.

The local error is $O(h^5)$ and four function evaluations are required per step. Note that Runge-Kutta methods do not need a starting procedure.

As a calculation proceeds, it may be desirable to vary the size of the step length h. One way to check on the size of the Runge-Kutta step is to repeat part of a calculation using h and $h/2$ as step sizes. For a Runge-Kutta method of order p it may be shown that

$$\begin{aligned} y_{(h)}(x_{n+1}) &= y(x_{n+1}) + C_n h^p + O(h^{p+1}) \\ y_{(h/2)}(x_{n+1}) &= y(x_{n+1}) + C_n (h/2)^p + O(h^{p+1}). \end{aligned}$$

By subtraction, and taking absolute values,

$$E_n = |C_n|(h/2)^p \simeq \frac{|y_{(h/2)}(x_{n+1}) - y_{(h)}(x_{n+1})|}{2^p - 1}$$

where E_n provides a suitable error estimate.

Let ε be the *target error per unit step*, i.e. we will compare ε with E_n/h. Now for a Runge-Kutta method of order p, halving the step should reduce the error per unit step to approximately $\varepsilon' = \varepsilon/2^{p+1}$‡. Whenever the size of h is to be checked, there are three possibilities:

(1) $\varepsilon' < E_n/h < \varepsilon$. Accept $y_{(h/2)}(x_{n+1})$ and keep the same step h.

(2) $E_n/h > \varepsilon$. Go back to the point x_n, use $h/2$ and $h/4$ and repeat the test.

(3) $E_n/h < \varepsilon'$. Accept $y_{(h/2)}(x_{n+1})$ and change the step size to $2h$.

More sophisticated variations on this idea allow the step size to be varied more widely.

5.2.3 Higher order equations

The RK4 algorithm, and many other algorithms for initial value problems, can be generalized to the solution of a set of m simultaneous differential equations of the form

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad \mathbf{y}(x_0) = \mathbf{b} \tag{5.17}$$

where x is a scalar variable, $\mathbf{b}$ is the initial value of the vector of solutions $\mathbf{y}$, and $\mathbf{f}$ is a vector of functions. This is called a *first-order system* of differential equations. The generalization of the 4th order Runge-Kutta scheme is

$$\begin{aligned} \mathbf{y_{n+1}} &= \mathbf{y_n} + \tfrac{1}{6}(\mathbf{k_1} + 2\mathbf{k_2} + 2\mathbf{k_3} + \mathbf{k_4}) \\ \mathbf{k_1} &= h\mathbf{f}(x_n, \mathbf{y_n}) \\ \mathbf{k_2} &= h\mathbf{f}(x_n + \tfrac{1}{2}h, \mathbf{y_n} + \tfrac{1}{2}\mathbf{k_1}) \\ \mathbf{k_3} &= h\mathbf{f}(x_n + \tfrac{1}{2}h, \mathbf{y_n} + \tfrac{1}{2}\mathbf{k_2}) \\ \mathbf{k_4} &= h\mathbf{f}(x_n + h, \mathbf{y_n} + \mathbf{k_3}) \end{aligned}$$

where $\mathbf{k_1}$, $\mathbf{k_2}$, $\mathbf{k_3}$ and $\mathbf{k_4}$ are also now vectors. Because of the similarity of the scalar and vector formulae it is usual for software for initial-value problems to be written for first-order systems rather than just the scalar case.

In fact, this algorithm is quite generally applicable. Consider, for example, the solution of the second order initial-value differential equation

$$y'' + p(x)y'^2 = q(x); \quad y(0) = u, y'(0) = v \tag{5.18}$$

‡ The factor 2^p comes from the order of convergence. The extra 2 is because the meaning of 'per unit step' changes as a result of halving the step.

where p, q are given functions and u, v are given constants. By writing $z_1 = y$, $z_2 = y'$ we can write (5.18) as the pair of equations

$$z_1' = z_2$$
$$z_2' = -p(x)z_2^2 + q(x)$$

with initial conditions $z_1(0) = u$, $z_2(0) = v$. Note that this can be written in the vector form (5.17), and is now a first-order system. This formulation can easily be used for most initial-value problems.

5.2.4 Multistep methods

To solve the ODE

$$y' = f(x, y(x)) \tag{5.19}$$

we can integrate both sides

$$\int_{x_n}^{x_{n+1}} y'\,dx = \int_{x_n}^{x_{n+1}} f(x, y(x))\,dx$$

so

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x, y(x))\,dx \tag{5.20}$$

then approximate $f(x, y(x))$ by a polynomial that fits the calculated values of f at x_n, x_{n-1}, $\ldots x_{n-k}$ and integrate that polynomial over (x_n, x_{n+1}). This leads to an Adams-Bashforth method, which is an example of a *multistep method.* Note that it uses a quadrature rule incorporating abscissae outside the interval of integration. If $k = 3$ the resulting formula is most conveniently stated in the form

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \tag{5.21}$$

where f_n denotes $f(x_n, y_n)$, and the local error term is

$$E = \tfrac{251}{720}h^5 y^{(5)}(\xi)$$

for some ξ in the interval (x_{n-3}, x_{n+1}). Another class of formulae can be derived similarly by instead integrating (5.19) from x_{n-p} to x_{n+1}. In this case if p is odd then it can be shown that the coefficient of f_{n-p} is always zero, so the method is slightly more efficient. Interesting cases are $p = 1$, i.e.

$$y_{n+1} = y_{n-1} + 2hf_n \tag{5.22}$$

with the local error term

$$E = \frac{h^3}{3}y'''(\xi)$$

and $p = 3$, i.e.

$$y_{n+1} = y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \tag{5.23}$$

with the local error term

$$E = \tfrac{14}{45}h^5 y^{(5)}(\xi).$$

Multistep methods are more powerful that one-step methods but suffer from the disadvantage that starting procedures are required: the formulae always require one or more values that are not available at the start of the algorithm.

Exercise 5a

> Suppose formula (5.21) is used with $h = 0.1$ and Euler's method is used as a starting procedure with step h/N. Assuming a global error h/N in Euler's method, and that $|y^{(5)}| < 30$, estimate the number of steps N of Euler's method required. Suggest a more suitable starting procedure.

5.2.5 Predictor-corrector methods

All the ODE formulae described up to now may be classified as being of *open* type as they are formulae for computing y_{n+1} from data corresponding to previously evaluated points. Formulae of *closed* type are implicit, i.e. the formula for y_{n+1} will depend on data corresponding to x_{n+1}. In general, a closed formula tends to have a more favourable error term but is not necessarily useful on its own.

Using the multistep approach, we can integrate (5.19) from x_{n-1} to x_{n+1} and approximate the integral on the right hand side using an interpolating polynomial that fits the (undetermined) value f_{n+1}. This yields the formula

$$y_{n+1} = y_{n-1} + \frac{h}{3}(\tilde{f}_{n+1} + 4f_n + f_{n-1}) \tag{5.24}$$

where $\tilde{f}_{n+1}$ is the best available approximation to f_{n+1}. Assuming that f_{n+1} is exactly known, the local error term for (5.24) is

$$E = -\tfrac{1}{90}h^5 y^{(5)}(\zeta).$$

This is 28 times smaller than that for the comparable formula (5.23).

A *predictor-corrector method* uses a *predictor* formula to provide a starting approximation for an unknown quantity, typically f_{n+1}. A *corrector* is a closed formula that uses the best available estimate of f_{n+1} in order to estimate y_{n+1} and, possibly, further improve it by iteration.

Milne's method consists of using the formula (5.23) as a predictor, i.e. to provide a starting approximation to y_{n+1} and hence f_{n+1}, and the formula (5.24) as a corrector to be used iteratively. Generally the *predictor*, though less accurate, should be of similar order to the *corrector* to avoid inefficiency.

5.2.6 Runge-Kutta *versus* predictor-corrector methods

Runge-Kutta methods are self-starting and numerically stable for sufficiently small h. The user has no way of knowing whether the value of h is sufficiently small, except that step-size control can be used. However, step-size control can be expensive.

Predictor-corrector methods may require less computation time, e.g. RK4 requires four function evaluations per step, whereas fourth order predictor-corrector methods require only two. It is easier to estimate a suitable value of h for predictor-corrector methods, though starting methods are required and, hence, changing the value of h is expensive.

Predictor-corrector methods may be numerically unstable.

5.3 Stability theory

The local error of Euler's method (5.13) is $O(h^2)$, whereas the local error of, say, formula (5.22) is $O(h^3)$. When comparing numerical results from these two methods for a small value of h the accuracy of the latter method is typically better for the first few steps, but this may cease to be true as the number of steps increases.

The following table is taken from Conte & de Boor and illustrates numerical instability in practice. Two methods are applied to the initial value problem

$$y' = -2y + 1, \quad y(0) = 1$$

which has the exact solution

$$y = \tfrac{1}{2}e^{-2x} + \tfrac{1}{2}.$$

Methods (5.13) and (5.22) are compared using step length $h = 1/32$ in each case.

			Errors	
n	$x(n)$	$y(n)$	Euler(5.13)	Multistep(5.22)
0	0.0000000	1.0000000	0.0000000	0.0000000
1	0.0312500	0.9697065	-0.0009565	-0.0000000
16	0.5000000	0.6839397	-0.0059027	0.0001420
32	1.0000000	0.5676676	-0.0042733	0.0001571
48	1.5000000	0.5248935	-0.0023205	0.0002392
64	2.0000000	0.5091578	-0.0011202	0.0005429
72	2.2500000	0.5055545	-0.0007583	0.0008719
80	2.5000000	0.5033690	-0.0005070	0.0014214
96	3.0000000	0.5012394	-0.0002203	0.0038365
112	3.5000000	0.5004559	-0.0000931	0.0104110
120	3.7500000	0.5002765	-0.0000601	0.0171571
121	3.7812500	0.5002598	-0.0000568	-0.0182603
122	3.8125000	0.5002440	-0.0000538	0.0194397
123	3.8437500	0.5002293	-0.0000509	-0.0206902
124	3.8750000	0.5002154	-0.0000482	0.0220260
125	3.9062500	0.5002023	-0.0000456	-0.0234434
126	3.9375000	0.5001901	-0.0000431	0.0249564
127	3.9687500	0.5001785	-0.0000408	-0.0265630
128	4.0000000	0.5001677	-0.0000386	0.0282768

Note that the listed x values are selective. Although initially more accurate, the multistep method goes unstable eventually.

Any one-step method, such as Euler's, is unconditionally stable for sufficiently small h, whereas the stability of a multistep method must always be investigated in order to decide whether it is stable, and under what circumstances.

Note that the formula used for numerical solution of an ODE is typically a difference equation, and the key to understanding the stability of the method is to examine the general solution of this difference equation. Consider a general non-linear ODE $y' = f(x, y)$. If we use a multistep method then this will typically lead to a non-linear difference equation, which is as hard to solve as the original problem. Consequently it is usual to examine the stability of a method only for the linear ODE

$$y' = \lambda y \tag{5.25}$$

where λ is a constant. This is sometimes referred to as the *test equation*. The general solution is $y = Ce^{\lambda x}$ so we can consider increasing solutions, by taking $\lambda > 0$, or decreasing solutions, by taking $\lambda < 0$. The ODE (5.25) leads to a linear difference equation, so the theory in Section 5.1 can be applied. This can be made clear by means of examples. Also in the interests of simplicity, and without loss of generality, we take $x_0 = 0$ so that $x_n = nh$.

The analysis of Euler's method is straightforward when applied to (5.25). Substituting $f = \lambda y$ in (5.13) gives the difference equation

$$y_{n+1} = (1 + \lambda h)y_n$$

with the general solution $y_n = c(1 + \lambda h)^n$. Using the well-known result

$$\lim_{\varepsilon \to 0} (1 + \varepsilon)^{1/\varepsilon} = e \tag{5.26}$$

we get that

$$\begin{aligned} \lim_{h \to 0} y_n &= c. \lim_{h \to 0} (1 + \lambda h)^n \\ &= c. \lim_{h \to 0} (1 + \lambda h)^{(\lambda x_n)/(\lambda h)} . \\ &= ce^{\lambda x_n} \end{aligned} \tag{5.27}$$

The end result of this analysis is similar for all one-step methods, but the significance of it only becomes apparent when we compare it with the analysis for a multistep method.

Consider now the multistep formula (5.22) applied to the ODE (5.25). Substituting $f = \lambda y$ gives the difference equation

$$y_{n+1} - 2\lambda h y_n - y_{n-1} = 0$$

with the general solution $y_n = c_1\beta_1^n + c_2\beta_2^n$ where β_1, β_2 are the roots of the characteristic equation

$$\beta^2 - 2\lambda h\beta - 1 = 0,$$

i.e.

$$\begin{aligned}\beta_i &= \lambda h \pm \sqrt{1+\lambda^2 h^2} \\ &= \pm 1 + \lambda h + O(h^2).\end{aligned}$$

Again using (5.26) we have

$$\begin{aligned}\lim_{h\to 0} y_n &= \lim_{h\to 0}\{c_1(1+\lambda h)^n + c_2(-1+\lambda h)^n\} \\ &= \lim_{h\to 0}\{c_1(1+\lambda h)^{(\lambda x_n)/(\lambda h)} + (-1)^n c_2(1-\lambda h)^{(-\lambda x_n)/(-\lambda h)}\} \\ &= c_1 e^{\lambda x_n} + (-1)^n c_2 e^{-\lambda x_n}\end{aligned}$$

which being a second order difference equation has 2 solutions, only one of which corresponds to a solution of the ODE (5.25). Now the equation (5.25) must have an initial condition which, if applied to the formula (5.22) in exact arithmetic, should be equivalent to setting the constant $c_2 = 0$ so that the correct solution is obtained. However, if the calculation is performed in floating-point arithmetic then rounding error will cause a small component of the second solution to be added in as the calculation progresses. To see the effect of this, we consider two cases: (1) $\lambda > 0$, and (2) $\lambda < 0$.

(1) If $\lambda > 0$ then the solution $e^{\lambda x}$ of the ODE (5.25) is an increasing function of x. Adding in, due to rounding error, a small component of the second solution $e^{-\lambda x}$ does not matter since it is a decreasing function of x and is multiplied by a small value anyway.

(2) If $\lambda < 0$ then the solution $e^{\lambda x}$ of the ODE is a decreasing function of x. The second solution $e^{-\lambda x}$ of the difference equation is an increasing function of x and even if multiplied by a small value it will eventually swamp the other component of the solution.

A formula such as (5.22), in which the stability depends on the sign of λ in (5.25), is said to be *weakly stable*.

We will now consider the general case. When a multistep method is applied to the ODE (5.25) the characteristic equation for the resulting difference equation can be expressed in the form

$$\rho(\beta) + \lambda h\sigma(\beta) = 0 \tag{5.28}$$

where $\rho(\beta)$ and $\sigma(\beta)$ are polynomials. The left hand side of (5.28) is called the *stability polynomial*. As $h \to 0$ the roots of this polynomial approach the roots of $\rho(\beta) = 0$. We would expect that $\beta = 1$ would be one of these roots since it corresponds to the solution of the ODE (5.25). A method is said to be *strongly stable* if all the roots of $\rho(\beta) = 0$ are such that $|\beta_i| < 1$ except for the root $\beta = 1$. Then any extra solutions of the difference equation cannot dominate the solution that corresponds to the ODE.

For example, consider the Adams-Bashforth formula (5.21). The stability polynomial for this is

$$\beta^4 - \beta^3 - \frac{\lambda h}{24}(55\beta^3 - 59\beta^2 + 37\beta - 9)$$

with $\rho(\beta) = \beta^4 - \beta^3$. The roots of ρ are $\beta_1 = 1$, $\beta_i = 0$ for $i = 2$, 3 and 4. Consequently the Adams-Bashforth method is strongly stable.

Note that, trivially, all one-step methods are strongly stable simply because the resulting difference equation does not have any solutions other the one that corresponds to the solution of the ODE.

Weak and strong stability are concepts that do not take into account the size of h, merely that it has to be sufficiently small. In general we have to take into account the entire stability polynomial (5.28) in order to compare methods.

A method is said to be *relatively stable* if the extra roots of the stability polynomial are less than the principal root in magnitude. A method is said to be *absolutely stable* for those values of λh for which the roots of the stability polynomial are less than one in magnitude.

Finally, it is worth noting that the build-up of rounding error in an ODE solution can be significantly reduced by performing some of the calculations using extra precision. In particular, summations of the form $u + hv$ should be performed with extra precision when h is small compared with u and v, and the result stored using extra precision.

Exercise 5b

Consider the alternative formulae

$$y_{n+1} = y_n + hf(x_n, y_n) + O(h^2) \qquad (a)$$
$$y_{n+1} = y_{n-1} + 2hf(x_n, y_n) + O(h^3) \qquad (b)$$

applied to the problem

$$y' = -y, \qquad y(0) = 1$$

using $h = 0.2$ in each case. Given $y(0.2) = 0.82$, and using only 2 significant digits in your calculation, progress the solution as far as $x = 0.8$ using each method. The exact solution to this accuracy is

x	exact $y(x)$
0.4	0.67
0.6	0.55
0.8	0.45

Which method is more accurate? Why?

If the methods are applied as far as $x = 3.0$, the results at the last two steps are as follows:

x	method (a)	method (b)	exact $y(x)$
2.8	0.043	0.068	0.061
3.0	0.034	0.040	0.050
3.2			0.041
3.4			0.033
3.6			0.027
3.8			0.022

Progress the solution as far as $x = 3.8$ to supply the missing values from the table. Given that the low precision of your calculations contributes only slightly to the errors, explain the behaviour of each method in this region.

Despite the behaviour you have observed, formula (b) is often used as a predictor in a predictor-corrector method. Why is this justifiable?

State the properties that you would expect a compatible corrector formula to possess.

5.4 Stiff ODEs

In various fields, e.g. chemical reactions, control theory, and electronics, systems exist in which certain processes happen on a very short time-scale and others on a very long time-scale. For example, in modelling the depletion of the ozone layer in the upper atmosphere, it may take years for pollutants released on the earth's surface to rise to the ozone layer, but a resulting chemical reaction may take place over only a few seconds. It would not seem sensible to model the whole process by means of uniform time-steps. Such systems give rise to so-called *stiff differential equations*, in which the independent variable x often represents time, and for which there are *transient solutions* that require a short time-step for some part of the calculation.

We will consider only a simple linear example in order to study the special numerical problems that this poses. Consider the second order ODE

$$y'' + 1001y' + 1000y = 0, \qquad y(0) = 1, \quad y'(0) = -1 \tag{5.29}$$

which has the exact solution $y(x) = e^{-x}$. The general solution of the differential equation is

$$y(x) = Ae^{-x} + Be^{-1000x}$$

but the initial conditions ensure that $B = 0$. From the discussion of stability in Section 5.3 we would expect that, due to rounding error, a small proportion of the second component e^{-1000x} would creep into any numerical solution but, as this decays very rapidly, this in itself would not cause a problem.

Changing the notation slightly, we can write $y_1 = y$ and $y_2 = y'$ to convert (5.29) into the first-order system

$$\begin{aligned} y_1' &= y_2 & y_1(0) &= 1 \\ y_2' &= -1001y_2 - 1000y_1 & y_2(0) &= -1 \end{aligned}$$

so that the vector generalization of RK4 could be used. Analysis of this method, as applied to this problem, shows that stability is affected by both solutions of (5.29) even though one of them is absent for the given initial conditions. Stability is only achieved for $h < 0.0028$, which is a reasonable step size for calculating e^{-1000x} but rather small for e^{-x}.

The *trapezoidal method* is an example of a method which does not have such stability restrictions when applied to stiff problems. For a first-order system this may be expressed in vector notation as

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \tfrac{1}{2}h\{\mathbf{f}(x_n, \mathbf{y}_n) + \tilde{\mathbf{f}}(x_{n+1}, \mathbf{y}_{n+1})\}.$$

This is a closed formula and so needs to be used iteratively with, say, Euler's method to start it off. Using this formula on the problem (5.29) reasonable accuracy can be achieved for comparatively large step sizes, e.g. $h = 0.1$.

For a general non-linear first-order system

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$$

stiffness is determined by the eigenvalues of the Jacobian matrix $(\partial f_i/\partial y_j)$. In a region (of the x axis) where the eigenvalues of the Jacobian are negative and well-separated in magnitude the problem is stiff and special methods must be used. The work of C.W. Gear and others on *backward differentiation formulae* (or *BDF methods*) has led to much of the available modern numerical software for stiff equations.

M.R.O'D.
11.12.2003