Numerical Analysis MATH30029

Lecture Notes Autumn 2025 (Adapted from notes of Martin Sieber)

Contents

Course Introduction

Numerical Analysis concerns the development and study of computational methods for approximating solutions to problems in mathematics involving continuous variables (as opposed to the study of discrete mathematics which normally falls within the remit of Computer Science.)

In many application areas in the physical sciences problems arise which cannot be solved exactly using even the most sophisicated mathematical methods and so computational methods are devised to approximate their solutions.

There are various important considerations to when devising these methods/algorithms such as efficiency (speed), accuracy, robustness.

Numerical Analysis is largely concerned with answering these questions thereby allowing us to understand which algorithms are best suited to certain tasks.

We are going to study a subset of the most important problems in Numerical Analysis and the ones which are most relevant to practical applications. We cannot cover all the things we want and major omissions include a plethora of matrix methods (condition number, QR factorisation, Singular-Value Decomposition, Gauss-Seidel Interation, conjugate-gradient method). For those interested, books by Golub and Van Loan or Trefethen make excellent reading.

The type of questions that we will investigate are

- What kind of methods exist ?
- When does a method converge ?
- How rapidly does it converge ?
- Is a method stable ?
- How big is the error of an approximation ?
- What are the limitations of making calculations on a computer (*rounding error* effects) ?

This unit is not about writing computer code. The implementation of a numerical method into a computer program is an additional step requiring the development of algorithms.

If you want to read about a topic in more detail then a recommended course text is *Numerical Analysis* by Burden and Faires (available online via a link from the course webpage) and this covers most of the topics in the course. There are many other good books on Numerical Analysis which can be found in the Numerical Analysis section in the Queen's Library (books QA 297 ***).

1 Finite precision arithmetic and error

When making calculations on a calculator, or computer, numbers are not stored exactly (e.g. $\pi = 3.1416\ldots$ is irrational and $1/3 = 0.3333\ldots$ recurring both contain an infinitely long decimal representation). Most modern digital devices store numbers using 64-bit (or double precision; it used to be more common to use 32-bit or single precision) which is set by a universal standard: IEEE 754-2019.

1.1 Binary storage: approximation to numbers

Recall the conversion of a binary representation of a number to a decimal representation works according to this example,

$$101101 = 2^5 + 2^3 + 2^2 + 2^0 = 32 + 8 + 4 + 1 = 45$$

64-bit means that a number stored on a computer is represented by 64 bits of memory each of which can either be set to 0 or 1. E.g. a number x is represented by

$$x = \underbrace{0}_{s}\ \underbrace{10000000011}_{c}\ \underbrace{1011100100010000000000000000000000000000000000000000}_{f}.$$

The first bit (being 0 or 1) represents the sign of the number. The next 11 bits represent the (base 2) exponent and the final 52 bits are the binary fractional representation of the number. That is, the decimal representation of a number is given by

$$(-1)^s(1+f)2^{c-1023}.$$

In our example, this results in

$$x = (-1)^0 2^{(2^{10}+2^1+2^0-1023)}\left(1 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^5 + \left(\frac{1}{2}\right)^8 + \left(\frac{1}{2}\right)^{12}\right)$$

whose decimal representation is

$$x = 27.56640625.$$

The next *largest* number we can represent exactly is found by replacing the last digit by 1, or

$$x = 0\,10000000011\,1011100100010000000000000000000000000000000000000001$$

which is, in decimal,

$$x = 27.5664062500000017763568394002504646778106689453125$$

and the next *smallest* number is

$$x = 0\,10000000011\,1011100100001111111111111111111111111111111111111111$$

which is, in decimal,

$$x = 27.5664062499999982236431605997495353221893310546875.$$

It's not the number of decimal places represented by binary storage of numbers that is the immediate problem, it is that only some numbers can be represented exactly.

The relative difference between two consecutive numbers is evidently $1/2^{52} \approx 0.22 \times 10^{-16}$ and we commonly refer to a 64-bit systems as being accurate to roughly 16 digits. For 32-bit systems, the fraction (f) is 23 bits long, the exponent (c) is 8 bits so and the relative error is $1/2^{23} \approx 1 \times 10^{-7}$ and we commonly refer to 32-bit systems as being accurate to roughly 7 digits.

1.2 Limits on the size of numbers

As well as limited accuracy, the finite exponent means that computers have upper and lower limits on the size of numbers that can be stored. For 64-bit systems the smallest positive non-zero number is

$$x_{min} = (-1)^0 2^{1-1023}(1+0) \approx 0.22 \times 10^{-307}$$

and the largest number is

$$x_{max} = (-1)^0 2^{2046-1023}(1+1-2^{-52}) \approx 0.18 \times 10^{309}.$$

Attempts to store numbers smaller than x_{min} or larger than x_{max} result in what is referred to as **underflow** and **overflow**. The values of $c = 0$ (00000000000 in binary) and $c = 2047$ (1111111111) are reserved this. A computer will commonly represent numbers smaller than x_{min} as zero and continue. For numbers exceeding x_{max} a computer is likely to issue a "`NaN`" (not a number) warning, or 'crash'.

Q: Do we ever have to worry about numbers so big or so small ? Seems unlikely.

A: Not all the time, but certainly sometimes[1].

1.3 Rounding errors

Binary storage is not easy to work with since we are used to working in a decimal system. Based on above, we assume we have a system which stores N digits accurately ($N = 15$ for 64-bit systems).

E.g.: if $N = 5$ we would write

$$\pi = 3.1416, \qquad \mathrm{e}^{-1} = 3.6787 \times 10^{-1} \quad \text{(or, more simply, 0.36787)}$$

and we say we have 5-digit precision.

Remark: there are two ways of truncating the representation of a number: **chopping** and **rounding**. Throughout this course, we adopt rounding the last digit to the nearest whole number so that, for e.g., 3.1415926 is represented as 3.1416 (5 digits) and 3.14159 (6 digits).

A consequence of having to approximate numbers is that errors can be introduced into arithmetic. The operations $+, -, \times, /$ are called **floating point operations** or **flops**.

For e.g. with 5-digit precision, $x = \pi \times \mathrm{e}^{-1}$, which has an exact value $1.155727\ldots$ is represented numerically by

$$\tilde{x} = 3.1416 \times 0.36787 = 1.1557.$$

Here there is no loss of accuracy due to the truncated storage of π and of e^{-1} and through the arithmetic operation of multiplication. However, consider $x = \pi/\mathrm{e}^{-1} = 8.53973\ldots$ where the use of 5-digit precision results in

$$\tilde{x} = 3.1416/0.36787 = 8.5395,$$

and is *not* the exact value x rounded to 5 digits. This is a manifestation of **round-off error** and in a moment we show how these small errors can accumulate into larger errors.

Defn: There are two ways of quantifying error:

[1] The maiden flight of the Ariane 5 rocket in 1996 ended in disaster, exploding just 40 seconds after take off. The cause was found to be the conversion of a 64-bit decimal to a 16-bit integer which was too large to be represented.

(i) **Absolute error:** $e = x - \tilde{x}$.

(ii) **Relative error:** $e_r = \left|\dfrac{x - \tilde{x}}{x}\right|$. E.g. if the relative error is 0.02, this means the error is 2% of the exact value.

Defn: In a system with N-digit precision an exact number x is approximated by $\tilde{x}$ so that

$$\left|\frac{x - \tilde{x}}{x}\right| < \epsilon = 10^{-N}$$

and ϵ is the **machine accuracy**.

1.3.1 Example

Compute $f(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ at $x = 4.71$ using 3-digit arithmetic. In the table we work from left to right.

	x	x^2	x^3	$6.1x^2$	$3.2x$	$f(x)$
Exact	4.71	22.1841	104.487111	135.323301	15.072	-14.263899
3-digit	4.71	$4.71 \times 4.71 = 22.2$	$22.2 \times 4.71 = 105$	135	15.1	-13.4

The relative error is

$$e_r = \left|\frac{-14.263899 + 13.4}{-14.263899}\right| \approx 0.06$$

equivalent to 6% which is a lot higher than 0.1% associated with 3-digit accuracy.

Q: Can we do anything to improve the accuracy ?

A: In this case, consider the **nested** calculation: $f(x) = ((x - 6.1)x + 3.2)x + 1.5$. With this expression, using 3-digit accuracy we end up with a relative error of 0.0025 or 0.25%.

Why ? If we count the number of flops needed for the first calculation it is 9(×) plus 3(±). In the second calculation it less that half this: 2(×) plus 3(±) (i.e. fewer errors).

2 Linear systems of equations

In this section we consider solutions x_1, x_2, …, x_n to the system of n equations:

$$(1) \qquad \left.\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n & = & b_2 \\ \vdots \qquad\qquad\quad \vdots & \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n & = & b_n. \end{array}\right\}$$

We are already comfortable writing (1) as

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

where

$$A = \begin{bmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \ldots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}.$$

We can also express (1) as

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1, 2, \ldots, n$$

Provided the inverse, A^{-1}, exists the solution is represented by

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Remark 1: A^{-1} is a notational device: it doesn't tell you what A^{-1} is.

Remark 2: Indeed, the inverse exists iff $\det(A) \neq 0$ where $\det(A)$ is called the **determinant** of A. Then A is said to be **non-singular** and the solution is **unique**.

If $\det(A) = 0$ then A is said to be **singular**. In this case, there may either be **no solution** or **infinitely many solutions**. E.g.

$$\begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \alpha \end{pmatrix}$$

and $\det(A) = 0$.

- If $\alpha = 2$ then solution is $x_1 = 1 - x_2$ for any x_2. [2]
- If $\alpha \neq 2$ then there are no solutions.

E.g.: If $n = 2$ we know that there is an explicit formula:

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}$$

where $\det(A) = a_{11}a_{22} - a_{12}a_{21}$ is the determinant.

Remark: In fact, there is an explicit formula for (or algorithm) for calculating A^{-1} for $n > 2$, known as *Cramer's Rule*[3]. E.g. $n = 3$

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} \\ -\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} & -\begin{vmatrix} a_{12} & a_{11} \\ a_{32} & a_{31} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{pmatrix}$$

where

$$\det(A) = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} .$$

The process is recursive: for $n = 4$, each entry of A^{-1} is expressed as the determinant of a 3×3 matrix, which is itself expressed in terms of 2×2 determinants (as above). And so on.

Q: OK, so are we done ?

A: No, because a careful counting of the flops needed to invert an $n \times n$ matrix shows they scale like $(n+1)!$. This is bad since (Stirling's formula) $n!$ grows like $n^n/\sqrt{n}$. For example, imagine with $n = 10$ the calculation of an inverse took 10^{-5} seconds on a computer. For $n = 30$ it would take 4000 years !

[2] there is an overlap with eigenvectors and eigenvalues that we are not going to follow here

[3] after Gabriel Cramer (1704-1752), who published the rule for arbitrary n in 1750

2.1 Invertible matrices

A general class of matrices that are invertible are **orthogonal matrices** Q satisfying $QQ^T = Q^TQ = I$. Evidently $Q^{-1} = Q^T$. Otherwise examples of matrices with explicit inverses are rare.

2.1.1 Diagonal Matrices

If $a_{ij} = \alpha_i \delta_{ij}$ then clearly $a_{ij}^{-1} = \alpha_i^{-1}\delta_{ij}$. Simple.

2.1.2 Triangular systems

If the matrix U is **upper triangular** ($u_{ij} = 0$ if $i > j$) then the system $U\mathbf{x} = \mathbf{b}$ is easy to solve

$$\begin{array}{rcrcl}
u_{11}x_1 + u_{12}x_2 + u_{13}x_3 + & \dots & + u_{1n}x_n & = & b_1 \\
u_{22}x_2 + u_{23}x_3 + & \dots & + u_{2n}x_n & = & b_2 \\
\vdots & & \vdots & \vdots & \vdots \\
 & & u_{n-1\,n-1}x_{n-1} + u_{n-1\,n}x_n & = & b_{n-1} \\
 & & u_{nn}x_n & = & b_n.
\end{array}$$

If $u_{ii} \neq 0$, $i = 1, 2, \dots, n$, (this is equivalent to $\det U = \prod_{i=1}^n u_{ii} \neq 0$) then the unknowns x_j can be computed by **backward substitution**:

$$x_n = \frac{b_n}{u_{nn}};$$

$$x_j = \frac{b_j - \displaystyle\sum_{k=j+1}^{n} u_{jk}x_k}{u_{jj}}, \qquad j = n-1, n-2, \dots, 1.$$

Similarly if the matrix L is **lower triangular** ($l_{ij} = 0$ if $i < j$) then the system $L\mathbf{x} = \mathbf{b}$ can be solved by **forward substitution**:

$$x_1 = \frac{b_1}{l_{11}};$$

$$x_j = \frac{b_j - \displaystyle\sum_{k=1}^{j-1} l_{jk} x_k}{l_{jj}}, \qquad j = 1, 2, \ldots, n-1.$$

Sometimes it is possible to invert lower triangular matrices explicitly. Of use later on is the following (see homework):

$$L_k = \begin{bmatrix} 1 & 0 & \cdots & & & \cdots & 0 \\ 0 & \ddots & \ddots & & & & \vdots \\ \vdots & \ddots & 1 & & & & \\ & & -l_{k+1,k} & 1 & & & \\ & & -l_{k+2,k} & 0 & \ddots & \ddots & \vdots \\ \vdots & & \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & -l_{n,k} & 0 & \cdots & 0 & 1 \end{bmatrix} \quad \Longrightarrow \quad L_k^{-1} = \begin{bmatrix} 1 & 0 & \cdots & & & \cdots & 0 \\ 0 & \ddots & \ddots & & & & \vdots \\ \vdots & \ddots & 1 & & & & \\ & & l_{k+1,k} & 1 & & & \\ & & l_{k+2,k} & 0 & \ddots & \ddots & \vdots \\ \vdots & & \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & l_{n,k} & 0 & \cdots & 0 & 1 \end{bmatrix}.$$

2.2 Gaussian elimination

Although named after Carl Gauss, the method was used in China at least as far back as 179AD.

The idea is to apply **elementary row operations** to reduce the matrix A to upper triangular form allowing the resulting system to be solved using back substitution.

There are three row operations that do not change the solution vector $\mathbf{x}$:

(i) multiply row R_i by a constant λ: $\quad R_i \to \lambda\, R_i$

(ii) add λ times row R_j to row R_i: $\quad R_i \to R_i + \lambda R_j$

(iii) interchange rows R_i and R_j: $\quad R_j \leftrightarrow R_i$

We'll use (for now) the last two operations to bring the system into triangular form. This is the **Gaussian elimination** process. It is best described using an example.

2.2.1 Example

Solve the following for $\mathbf{x}$:

$$\begin{bmatrix} 3 & 6 & 9 \\ 2 & 5 & 2 \\ -3 & -4 & -11 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 3 \\ 4 \\ -5 \end{bmatrix}.$$

We perform the row operations

$$\begin{aligned} R_2 &\to R_2 - \frac{2}{3} R_1 \\ R_3 &\to R_3 + \ \ R_1 \end{aligned}$$

to obtain zeros below the diagonal in the first column:

$$\begin{bmatrix} 3 & 6 & 9 \\ 0 & 1 & -4 \\ 0 & 2 & -2 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 3 \\ 2 \\ -2 \end{bmatrix}.$$

Next, the row operation

$$R_3 \to R_3 - 2R_2$$

produces a zero below the diagonal in the second column:

$$\begin{bmatrix} 3 & 6 & 9 \\ 0 & 1 & -4 \\ 0 & 0 & 6 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 3 \\ 2 \\ -6 \end{bmatrix}.$$

Now the system is upper triangular it can be solved by **backward substitution**

$$\begin{aligned} 6x_3 = -6 \qquad &\Longrightarrow \qquad x_3 = -1 \\ x_2 - 4x_3 = 2 \qquad &\Longrightarrow \qquad x_2 = -2 \\ 3x_1 + 6x_2 + 9x_3 = 3 \qquad &\Longrightarrow \qquad x_1 = 8. \end{aligned}$$

Remark 1: At each step of the Gaussian elimination, the matrix A and the vector $\mathbf{b}$ change, but the solution vector $\mathbf{x}$ remains unaltered.

Remark 2: A compact space-saving device that helps simplify the presentation involves performing the row operations on the $n \times (n+1)$ **augmented matrix** $\tilde{A}$:

$$\tilde{A} = [A, \mathbf{b}] = \begin{bmatrix} a_{11} & \dots & a_{1n} & b_1 \\ \vdots & & \vdots & \vdots \\ a_{n1} & \dots & a_{nn} & b_n \end{bmatrix}.$$

2.2.2 General method

We apply the method to a general matrix A. In **step 1** of the Gaussian elimination process one applies row operations to obtain zeros below the diagonal in the first column.

Step 1:

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}$$

for $j = 1, \dots, n$ with

$$b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}} b_1$$

for $i = 2, \dots, n$ and results in an augmented matrix of the form

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} & b_n^{(1)} \end{bmatrix}.$$

Remark: This is possible only if $a_{11} \neq 0$; the matrix element a_{11} is called the **pivot element** of the operations in step 1.

Step k: We can apply the same set of row operations to each reduced matrix. I.e. for $1 \leq k \leq n-1$ we define

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)}$$

for $j = k, \dots, n$ with

$$b_i^{(k)} = b_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} b_k^{(k-1)}$$

for $i = k, \dots, n$ requiring pivot element $a_{kk}^{(k-1)} \neq 0$. E.g. After step 2 using $k = 2$ results in the augmented matrix

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_1 \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \dots & a_{2n}^{(1)} & b_2^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \dots & a_{3n}^{(2)} & b_3^{(2)} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \dots & a_{nn}^{(2)} & b_n^{(2)} \end{array}\right].$$

and the process continues until the matrix is in upper triangular form.

Remark: A simple calculation[4] of the flops needed shows it grows like n^3, with increasing n, significantly less rapdily than $(n+1)!$.

Q: What can go wrong?

A: A pivot element can be zero!

2.2.3 Example (Pivoting)

Consider slightly altering Example 2.2.1 to give the augmented matrix

$$\left[\begin{array}{ccc|c} 3 & 6 & 9 & 3 \\ 2 & 4 & 2 & 4 \\ -3 & -4 & -11 & -5 \end{array}\right].$$

We perform the row operations

$$\begin{aligned} R_2 &\to R_2 - \frac{2}{3} R_1 \\ R_3 &\to R_3 + R_1. \end{aligned}$$

[4] Specifically, we can deduce that there are a total of $n(n+1)/2$ divisions, $(2n^3 + 3n^2 - 5n)/6$ multiplications, and $(2n^3 + 3n^2 - 5n)/6$ additions for a total of approximately $2n^3/3$ operations.

This leads to

$$\begin{bmatrix} 3 & 6 & 9 & 3 \\ 0 & 0 & -4 & 2 \\ 0 & 2 & -2 & -2 \end{bmatrix}$$

and cannot continue with the usual Gaussian elimination process. The solution is to swap rows to obtain a non-vanishing pivot element and then continue. I.e.

$$\begin{bmatrix} 3 & 6 & 9 & 3 \\ 0 & 2 & -2 & -2 \\ 0 & 0 & -4 & 2 \end{bmatrix} .$$

In this example we obtain a matrix which is already in triangular form.

Remark 1: The process of swapping rows in order to obtain a non-vanishing pivot element is called **pivoting**.

Remark 2: In the case that the pivot element and all elements below it are zero then one can show that $\det(A) = 0$ (simple: you try it !).

2.3 Rounding errors

Rounding errors can affect Gaussian elimination. Here's a simple example.

2.3.1 Example

Consider the following system of equations with $\epsilon \ll 1$

$$\left.\begin{aligned} \epsilon x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}\right\}$$

The exact solution is

$$x_1 = \frac{1}{1-\epsilon}, \qquad x_2 = \frac{1-2\epsilon}{1-\epsilon}$$

and we can use the Binomial expansion to give $x_1 = 1+\epsilon+\epsilon^2+\ldots \approx 1+\epsilon$ when $\epsilon \ll 1$. Similarly, $x_2 \approx (1-2\epsilon)(1+\epsilon+\epsilon^2+\ldots) \approx 1-\epsilon$.

Take $\epsilon = 10^{-4}$ and let us perform calculations with 3-digit precision. Our augmented matrix is

$$\begin{bmatrix} 10^{-4} & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}. \tag{2}$$

The row operation $R_2 \to R_2 - 10^4 R_1$ leads to (exactly)

$$\begin{bmatrix} 10^{-4} & 1 & 1 \\ 0 & -9999 & -9998 \end{bmatrix}.$$

However, if we work with 3-digit precision this gets rounded (e.g. $9999 = 9.999 \times 10^4$ but since we can only store 3 digits we round to the nearest 3-digit number, 1.00×10^5) to

$$\begin{bmatrix} 10^{-4} & 1 & 1 \\ 0 & -10000 & -10000 \end{bmatrix}.$$

Then we have the upper triangular system

$$\begin{aligned} 10^{-4}x_1 + x_2 &= 1, \\ -10000x_2 &= -10000, \end{aligned}$$

with the solutions $x_2 = 1$ and $x_1 = 0$ instead of $x_2 \approx 1$ and $x_1 \approx 1$.

Q: Why did this happen ?

A: The pivot element (though not zero) is much smaller than the element below it. In fact one can formally show that the Gaussian elimination process is **numerically unstable**: the combined effect of multiple row operations propagate and inflate errors until they dominate solutions.

2.3.2 Partial pivoting

A solution is to interchange rows before Gaussian elimination

$$\begin{bmatrix} 1 & 1 & 2 \\ 10^{-4} & 1 & 1 \end{bmatrix}.$$

The row operation $R_2 - 10^{-4}R_1 \to R_2$ performed with 3-digit precision now leads to

$$\begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

whose solution is $x_2 = 1$ and $x_1 = 1$ and matches the exact solution expressed to 3-digit precision.

The method of **partial pivoting** formalises this process. Before step k of the Gaussian elimination process the row having the element with the largest magnitude amongst the leading column $(a_{kk}^{(k-1)}, \dots, a_{nk}^{(k-1)})$ is interchanged with the the row containing the pivot element.

2.3.3 Scaled partial pivoting

Partial pivoting works in many cases, but not always. Consider, for example, Example 2.3.1 but with the first row multiplied by 2×10^4. We obtain

$$\begin{aligned} 2x_1 + 2 \times 10^4 x_2 &= 2 \times 10^4 \\ x_1 + x_2 &= 2. \end{aligned} \tag{3}$$

The solution is same, $x_1 \approx 1$, $x_2 \approx 1$. However, this no longer requires partial pivoting and so Gaussian elimination leads to the same problem as before and we obtain $x_2 \approx 1$ and $x_1 \approx 0$ when performing calculations with 3-digit precision.

Q: Why ?

A: The largest elements in different rows have vastly different magnitude.

A possible solution (**Scaled partial pivoting**): before starting define a scale factor for each row:

$$s_i = \max_{1 \le j \le n} |a_{ij}|, \qquad i = 1, \dots, n$$

which is non-zero otherwise the matrix is singular. We now do $R_1 \leftrightarrow R_p$ where

$$\frac{|a_{p1}|}{s_p} = \max_{1 \le k \le n} \frac{|a_{k1}|}{s_k}$$

The effect of scaling is to ensure that the largest element in each row has a relative magnitude of 1. In the example above this reduces (3) to (2) and now partial pivoting works. For larger than 2×2 matrices subsequent row interchanges are performed using the same principle noting that the values of s_i used are computed only once before step 1 and move with the row interchanges.

Remark: Other pivoting strategies. Complete pivoting includes row and column interchanges, but comes with an additional computational cost that is only justified if high accuracy is critical.

2.4 LU-decomposition

Imagine that it is possible to write A as the product of two factors

$$A = LU,$$

where L is a lower triangular matrix and U an upper triangular matrix. It follows that

$$\mathbf{b} = A\mathbf{x} = LU\mathbf{x} = L\mathbf{y}$$

(say) where we have written $\mathbf{y} = U\mathbf{x}$. Now we can compute $\mathbf{y}$ from $L\mathbf{y} = \mathbf{b}$ using forward substitution, and then compute $\mathbf{x}$ from $U\mathbf{x} = \mathbf{y}$ using back substitution.

The row operations of Gaussian elimination at step 1 are expressed as

$$R_i \to R_i - l_{i1}R_1, \qquad i = 2, \dots, n, \qquad \text{where} \qquad l_{i1} = \frac{a_{i1}}{a_{11}}.$$

These row operations transform the matrix A into a new matrix $A^{(1)} = L_1 A$ where

$$L_1 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -l_{21} & 1 & 0 & \cdots & 0 \\ -l_{31} & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -l_{n1} & 0 & \cdots & 0 & 1 \end{bmatrix}.$$

It follows that $A = L_1^{-1} A^{(1)}$ (see Section 2.1.2). I.e.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & 0 & \cdots & 0 \\ l_{31} & 0 & 1 & 0 & \cdots & 0 \\ l_{41} & 0 & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ l_{n1} & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & \cdots & a_{nn}^{(1)} \end{bmatrix}$$

where $a_{ij}^{(1)} = a_{ij} - a_{i1}\, a_{1j}/a_{11}$ as in Gaussian elimination.

Remark: This process is **not** Gaussian elimination: we are not applying row operations to the vector $\mathbf{b}$. In fact this does not involve $\mathbf{b}$ at all.

Step 2: This process can be continued. Now the Gaussian elimination process is applied to $A^{(1)}$ and the new pivot element $a_{22}^{(1)}$ is used to eliminate the matrix elements in the second column below the diagonal. The row operations are

$$R_i \to R_i - l_{i2}R_2, \qquad i = 3, \ldots, n \qquad \text{where} \qquad l_{i2} = \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}$$

and we can encode this as $L_2 A^{(1)} = A^{(2)}$ where

$$L_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -l_{32} & 1 & 0 & \cdots & 0 \\ 0 & -l_{42} & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & -l_{n2} & 0 & \cdots & 0 & 1 \end{bmatrix}$$

in exactly the same manner as before. So $A^{(1)} = L_2^{-1}A^{(2)}$ and $A = L_1^{-1}L_2^{-1}A^{(2)}$ or

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & 0 & \cdots & 0 \\ l_{41} & l_{42} & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ l_{n1} & l_{n2} & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & & & \vdots \\ \vdots & \vdots & \vdots & & & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \cdots & \cdots & a_{nn}^{(2)} \end{bmatrix}.$$

One can continue this process until Gaussian elimination is complete. The final result is

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & 0 & \cdots & 0 \\ l_{41} & l_{42} & l_{43} & 1 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & \cdots & u_{3n} \\ \vdots & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 & u_{nn} \end{bmatrix}$$

where we have labelled the elements of the matrix obtained from Gaussian elimination by u_{ij}. The result is $A = LU$ as required.

2.4.1 Advantages of LU-decomposition

(i) Calculating the determinant of A, since

$$\det(A) = \det(L)\det(U) = \prod_{i=1}^{n} u_{ii}.$$

This is much quicker than the Laplace expansion along rows or columns if n is large.

(ii) Particularly useful for solving $A\mathbf{x}_i = \mathbf{b}_i$ for multiple RHS vectors $\mathbf{b}_i$, since $A = LU$ is done only once.

(iii) Can use (ii) to find A^{-1}. Consider $i = 1, 2, \dots, n$ and set $(\mathbf{b}_i)_j = \delta_{ij}$ and then we can organise the multiple systems in the following way:

$$A \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ \vdots & \vdots & & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n \\ \vdots & \vdots & & \vdots \end{bmatrix} = I.$$

Multiplying LHS and RHS by A^{-1} gives us

$$\begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ \vdots & \vdots & & \vdots \end{bmatrix} = A^{-1}$$

Therefore the n columns of A^{-1} are the n solutions, $\mathbf{x}_i$ of $A\mathbf{x}_i = \mathbf{b}_i$ for $i = 1, \dots, n$ with $(\mathbf{b}_i)_j = \delta_{ij}$.

Remark: The LU-decomposition is possible if the Gaussian elimination can be performed without row interchanges. What if it cannot ?

2.4.2 Partial Pivoting and Permutation Matrices

Consider, for e.g.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 2 & 1 \\ 6 & 3 & 6 \end{bmatrix}.$$

Let us partial pivot to promote the 6 to the top LH element. I.e. we have a pre-step 1 $R_1 \leftrightarrow R_3$. This row interchange can be encoded by premultiplying by a **permutation matrix**

$$P_{13} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad (= I \quad \text{under a row swap of } R_1 \leftrightarrow R_3.)$$

I.e.

$$P_{13}A = \begin{bmatrix} 6 & 3 & 6 \\ 4 & 2 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

Now step 1 is $R_2 \to R_2 - \frac{2}{3}R_1$ and $R_3 \to R_3 - \frac{1}{6}R_1$ to get

$$P_{13}A = \begin{bmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{1}{6} & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 & 3 & 6 \\ 0 & 0 & -1 \\ 0 & \frac{3}{2} & 2 \end{bmatrix} = L_1^{-1}A^{(1)}.$$

To continue we need to interchange R_2 and R_3 and this requires another permutation matrix

$$P_{23} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad (= I \quad \text{under a swap of } R_2 \leftrightarrow R_3.)$$

Remark: Permutation matrices belong to a class of **Orthogonal matrix** and satisfy $(P_{ij})^{-1} = (P_{ij})^T$. Since the swap needs to be applied to $A^{(1)}$ we write

$$P_{13}A = L_1^{-1}P_{23}^{-1}P_{23}A^{(1)}.$$

and it turns out (below) to be useful to premultiply by P_{23} so that

$$PA = P_{23}L_1^{-1}P_{23}^TU$$

where

$$U = P_{23}A^{(2)} = \begin{bmatrix} 6 & 3 & 6 \\ 0 & \frac{3}{2} & 2 \\ 0 & 0 & -1 \end{bmatrix}$$

is the final result of Gaussian elimination with partial pivoting,

$$P = P_{23}P_{13} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad (= I \quad \text{under } R_2 \leftrightarrow R_3 \textit{ and } R_1 \leftrightarrow R_3)$$

and

$$L = P_{23}L_1^{-1}P_{23}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{1}{6} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{6} & 1 & 0 \\ \frac{2}{3} & 0 & 1 \end{bmatrix}$$

where the premultiplication by P_{23} swaps rows and the post multiplication swaps columns !

Remarks: This is quite complicated but LU with pivoting can be summarised as $PA = LU$ in which:

- where P records all the row interchanges, applied to I;
- as you develop L through Gaussian elimination steps, any row changes need to be applied to L.
- Alternatively, do all necessary row interchanges first (i.e. do PA) and apply LU decomposition to PA.
- Note $P^{-1} = P^T$ is also an orthogonal matrix (see homework).
- When it comes to solving $A\mathbf{x} = \mathbf{b}$ write $PA\mathbf{x} = P\mathbf{b} = \mathbf{b}'$ first and then substitute $PA = LU$ and then forwards/backwards substitute.

3 Root finding

We consider methods for determining numerical solutions to

$$f(x) = 0$$

when they cannot be determined explicitly (e.g. $e^x + x = 0$). Normally root-finding methods are **iterative** and meaning the solution is not found in a finite number of steps.

I.e. we generate a sequence x_0, x_1, x_2 s.t. x_n is determined in terms of previous elements in the sequence with the goal that $x_n \to x^*$ as $n \to \infty$ where $f(x^*) = 0$.

3.1 Bisection method

Suppose we identify a and b such that $f(a)$ and $f(b)$ have opposite signs ($f(a) \cdot f(b) < 0$). It follows by the intermediate value theorem (IVT) that there exists at least one root $x = x^*$, $x^* \in (a, b)$ such that $f(x^*) = 0$. There can be any more than one root, but there must be an odd number of roots.

The bisection algorithm consists of these steps.

1. Set $n = 1$
2. Let $x_n = \frac{1}{2}(a + b)$.
3. Calculate $f(x_n)$.
4. If $f(x_n) = 0$ then $x^* = x_n$ and we are DONE.

   Else if $f(x_n) \cdot f(a) < 0$ then $\exists$ root in (a, x_n): redefine $b = x_n$ and continue to 5.

   Otherwise $f(x_n) \cdot f(a) > 0$ then $\exists$ root in (x_n, b): redefine $a = x_n$ and continue to 5.
5. Increase n by 1 and go back to step 2.

Remark: We need to force the iteration to stop after N steps, say.

3.1.1 Example

Find all roots of $f(x) = \mathrm{e}^x - 3x = 0$.

Draw graphs of e^x and $3x$ and identify roots of $f(x)$ as the values of x where curves intersect, which allows us to estimate intervals (a, b) which can be used to initiate the Bisection method.

Alternatively we can sample the function $f(x)$ at a few values of x. Here we have $f(0) > 0$, $f(1) < 0$ and $f(2) > 0$. There is one root in the interval $[0, 1]$ and another in the interval $[1, 2]$.

Here is the method with $a = 1$, $b = 2$ up to $n = 9$. The exact value of the root is $x^* = 1.51213\ldots$.

n	a	x_n	b	$f(a)$	$f(x_n)$	$f(b)$	max. err.
1	1.0	1.50000	2.0	-0.28172	-0.01831	1.38906	2^{-1}
2	1.50000	1.75000	2.0	-0.01831	0.50460	1.38906	2^{-2}
3	1.50000	1.62500	1.75000	-0.01831	0.20342	0.50460	2^{-3}
4	1.50000	1.56250	1.62500	-0.01831	0.08323	0.20342	2^{-4}
5	1.50000	1.53125	1.56250	-0.01831	0.03020	0.08323	2^{-5}
6	1.50000	1.51562	1.53125	-0.01831	0.00538	0.03020	2^{-6}
7	1.50000	1.50781	1.51562	-0.01831	-0.00660	0.00538	2^{-7}
8	1.50781	1.51172	1.51562	-0.00660	-0.00064	0.00538	2^{-8}
9	1.51172	1.51367	1.51562	-0.00064	0.00236	0.00538	2^{-9}

Remark 1: The maximum error after n iterations satisfies

$$|x_n - x^*| \leq \frac{|b-a|}{2^n}$$

and a, b are end points of the original interval. I.e. the maximum error is only halved by an iteration and although the method is very robust and convergence is guaranteed, it is considered to be slow.

Remark 2: If we want to compute a root to within a prescribed accuracy or tolerance $\epsilon_{tol} > 0$ then the number of steps should be the smallest N s.t. that

$$\frac{|b-a|}{2^N} < \epsilon_{tol}$$

which rearranges to

$$N > \log(|b-a|/\epsilon_{tol})/\log(2).$$

3.2 Fixed-point iteration

If we want to find x^* such that $f(x^*) = 0$ we can express the condition instead as $x^* = g(x^*)$. This is most simply done by defining $g(x) = x + f(x)$ but this is not the only way (e.g. $g(x) = x - f(x)$ also works).

Defn: If x^* satisfies

$$x^* = g(x^*)$$

it is said to be a **fixed point** of $g(x)$.

Defn: A **fixed-point iteration** is defined by

$$x_{n+1} = g(x_n), \qquad n = 0, 1, 2, 3, \ldots$$

for a given **initial guess** x_0 such that $x_n \to x^*$ as $n \to \infty$.

Remark: Fixed point iterations are often easier to analyse than root-finding methods.

Numerical experiment: Use a fixed-point iteration to find a root of $f(x) = \cos x - x = 0$.

Choose $g(x) = \cos(x)$ and $x_0 = 1$. I.e. we solve $x_{n+1} = \cos x_n$ with $x_0 = 1$.

Remark: The numerical iteration converges to the fixed point $x^* = 0.739085\ldots$ and the errors $e_n = x_n - x^*$ decrease by a factor that is approximately constant ($e_n/e_{n-1} \approx 0.673$ for large n). This is slower than bisection.

Q: Does the recasting of $f(x) = 0$ into $g(x) = x$ affect the convergence ?

A: Yes, as this example shows...

n	x_n	e_n	e_n/e_{n-1}
0	1.000000	0.260915	
1	0.540302	−0.198783	−.761869
2	0.857553	0.118468	−.595967
3	0.654290	−0.084795	−.715765
4	0.793480	0.054395	−.641488
5	0.701369	−0.037716	−.693376
6	0.763960	0.024875	−.659516
7	0.722102	−0.016983	−.682734
8	0.750418	0.011333	−.667304
9	0.731404	−0.007681	−.677785
10	0.744237	0.005152	−.670767
⋮	⋮	⋮	⋮
20	0.739184	0.000099	−.673558
⋮	⋮	⋮	⋮
30	0.739087	0.000002	−.673620

3.2.1 Example

Consider finding the roots of $f(x) = x^2 - x - 1$ (i.e. the solutions $x = x^*$ of $f(x) = 0$).

Here are four different fixed point iterations $x_{n+1} = g(x_n)$ whose fixed points x^* coincide with the roots of $f(x)$:

$$\begin{aligned} x_{n+1} &= g_1(x_n) \equiv x_n^2 - 1; \\ x_{n+1} &= g_2(x_n) \equiv 1 + \frac{1}{x_n}; \\ x_{n+1} &= g_3(x_n) \equiv \sqrt{1 + x_n}; \\ x_{n+1} &= g_4(x_n) \equiv x_n - \frac{x_n^2 - x_n - 1}{2x_n - 1}. \end{aligned} \tag{4}$$

(check: in each case assume $x_n \to x^*$ to show that $f(x^*) = 0$).

Let's calculate the iterates for each fixed-point iteration with $x_0 = 2$. In this case there are two roots which we know exactly since f is quadratic. The are $x^* = (1 \pm \sqrt{5})/2$. We see that g_1 does *not* converge; $g_{2,3,4}$ all converge to the positive root, $x^* = 1.618034\ldots$, but g_4 is much faster.

n	x_n from $g_1(x)$	x_n from $g_2(x)$	x_n from $g_3(x)$	x_n from $g_4(x)$
0	2	2	2	2
1	3	1.500000	1.732051	1.666667
2	8	1.666667	1.652892	1.619048
3	63	1.600000	1.628770	1.618034
4	3968	1.625000	1.621348	1.618034
5	15745023	1.615385	1.619058	
6		1.619048	1.618350	
7		1.617647	1.618132	
8		1.618182	1.618064	
9		1.617978	1.618043	
10		1.618056	1.618037	
11		1.618026	1.618035	
12		1.618037	1.618034	
13		1.618033	1.618034	
14		1.618034		
15		1.618034		

Qs: When does a scheme converge ? What determines the speed of convergence ?

3.2.2 Conditions for convergence

Fixed-Point Theorem (FPT): If $g(x) \in C[a,b]$ and $g(x) \in [a,b]$ for all $x \in [a,b]$ *there exists* at least one fixed point $x^* \in [a,b]$ such that $g(x^*) = x^*$.

Suppose, further, that $g(x)$ is differentiable on (a,b) and $\exists\, k < 1$ such that

$$|g'(x)| \leq k \qquad \forall x \in (a,b).$$

Then the fixed point is *unique* and, for any $x_0 \in [a, b]$, the sequence $x_{n+1} = g(x_n)$, $n = 0, 1, 2, \ldots$ *converges* to the fixed point.

Proof: (i) *Existence of fixed point:* If $g(a) = a$ or $g(b) = b$ then we are done. Assume otherwise so that

$$g(a) - a > 0 \qquad \text{and} \qquad g(b) - b < 0$$

since it is now assumed that $g(x) \in (a, b)$ if $x \in [a, b]$. It then follows from the intermediate value theorem that $\exists\, x^* \in (a, b)$ such that $g(x^*) - x^* = 0$.

(ii) *Uniqueness of fixed point:* Assume that $\exists$ two fixed points $x_1^*, x_2^* \in [a, b]$ such that $g(x_1^*) = x_1^*$, $g(x_2^*) = x_2^*$ and $x_2^* > x_1^*$ (w.l.o.g). Then

$$\frac{g(x_2^*) - g(x_1^*)}{x_2^* - x_1^*} = \frac{x_2^* - x_1^*}{x_2^* - x_1^*} = 1.$$

Using the mean-value theorem

$$\left| \frac{g(x_2^*) - g(x_1^*)}{x_2^* - x_1^*} \right| = |g'(\xi)| \leq k < 1$$

for some $\xi \in (x_1^*, x_2^*)$ which is a contradiction.

(iii) *Convergence of the iteration:* We use the mean-value theorem to obtain

$$|x_n - x^*| = |g(x_{n-1}) - g(x^*)| = |g'(\xi_n)|\, |x_{n-1} - x^*| \leq k\, |x_{n-1} - x^*|$$

where ξ_n lies between x_{n-1} and x^*. Using this relation iteratively we obtain

$$|x_n - x^*| \leq k\, |x_{n-1} - x^*| \leq k^2\, |x_{n-2} - x^*| \leq \ldots \leq k^n\, |x_0 - x^*|.$$

Taking the limit $n \to \infty$ then yields

$$\lim_{n\to\infty} |x_n - x^*| \leq \lim_{n\to\infty} k^n\, |x_0 - x^*| = 0.$$

Note: If x_n is sufficiently close to x^* and if we define the error at the nth step by $e_n = x_n - x^*$, it follows that

$$x_n - x^* = g(x_{n-1}) - g(x^*) = g'(\xi_n)\,(x_{n-1} - x^*) \approx g'(x^*)\,(x_{n-1} - x^*)$$

where $\xi_n \in (x_{n-1}, x^*)$. I.e. $e_n \approx g'(x^*) e_{n-1}$. Hence:

(i) The size of $g'(x^*)$ controls the speed of convergence;

(ii) The iteration will **not converge** to x^* if $|g'(x^*)| > 1$.

Example: In Example 3.2.1 with $x^* = 1.618034$ we find:

$$\begin{aligned} &g_1'(x^*) \approx 3.24 > 1, && \text{(does not converge);} \\ &g_2'(x^*) \approx -0.382, && \text{(converges);} \\ &g_3'(x^*) \approx 0.309, && \text{(converges slighter faster than } g_2\text{);} \\ &g_4'(x^*) = 0, && \text{(calculation above needs modifying...)} \end{aligned}$$

3.2.3 Order of convergence

Defn: Let $\{x_0, x_1, \ldots\}$ be a sequence that converges to x^*, meaning $\lim_{n\to\infty} x_n = x^*$. If positive constants α and λ exist such that

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^\alpha} = \lambda \tag{5}$$

then the sequence converges to x^* with **order of convergence** α and **asymptotic error constant** λ.

Remark: In particular, we distinguish the following important cases (with $e_n = x_n - x^*$):

- If $\alpha = 1$ (and if $\lambda < 1$) the sequence is **linearly convergent** ($|e_{n+1}| \approx \lambda |e_n|$ if n is large);
- If $\alpha = 2$ the sequence is **quadratically convergent** ($|e_{n+1}| \approx \lambda |e_n|^2$ if n is large);
- If $\alpha = 3$ the sequence is **cubically convergent** ($e_{n+1}| \approx \lambda |e_n|^3$ if n is large).

Theorem: If the fixed-point iteration $x_{n+1} = g(x_n)$ converges to $x^* \in [a, b]$ and g satisfies $g \in C^p[a, b]$ and

$$0 = g'(x^*) = g''(x^*) = \ldots = g^{(p-1)}(x^*), \qquad g^{(p)}(x^*) \neq 0$$

(i.e. the first $p-1$ derivatives vanish at $x = x^*$) then the **order of convergence** is p and the **asymptotic error constant** is

$$\lambda = \frac{|g^{(p)}(x^*)|}{p!}.$$

Proof: We use **Taylor's theorem** (1st year analysis), together with $x_{n+1} = g(x_n)$ and $x^* = g(x^*)$, to obtain

$$\begin{aligned} x_{n+1} - x^* &= g(x_n) - g(x^*) \\ &= \sum_{k=0}^{p-1} \frac{g^{(k)}(x^*)}{k!}(x_n - x^*)^k + \frac{g^{(p)}(\xi(x_n))}{p!}(x_n - x^*)^p - g(x^*) \\ &= \sum_{k=1}^{p-1} \frac{g^{(k)}(x^*)}{k!}(x_n - x^*)^k + \frac{g^{(p)}(\xi(x_n))}{p!}(x_n - x^*)^p \end{aligned}$$

where $\xi(x_n) \in (x_n, x^*)$. The first term on the right-hand side of the last line vanishes due to the assumption of the theorem and we obtain

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^p} = \lim_{n\to\infty} \frac{|g^{(p)}(\xi(x_n))|}{p!} = \frac{|g^{(p)}(x^*)|}{p!}$$

and comparison with (5) completes the proof.

Example: In our previous Example 3.2.1, $f(x) = x^2 - x - 1 = 0$ with a root $x^* = (1+\sqrt{5})/2$ we showed that

$$g_4(x) = x - \frac{x^2 - x - 1}{2x - 1}$$

is s.t. $g_4'(x^*) = 0$. One can further check that $g_4''(x^*) \neq 0$, and hence we conclude from the last theorem that the convergence is quadratic.

3.2.4 Example

Consider the fixed-point iteration for $g(x) = \sin(\pi x/2)$. I.e. we iterate $x_{n+1} = \sin(\pi x_n/2)$ with $x_0 = 1.5$ and there are 3 fixed points: $x^* = -1, 0, 1$ (easy to see graphically by drawing $y = x$ and $y = \sin(\pi x/2)$ and noting 3 intersections).

We note that $g'(1) = 0$ and $g''(1) = -\pi^2/4$, so order of convergence is 2 (quadratic) and asymptotic error constant is $\pi^2/8 \approx 1.233$. This is confirmed by numerical results:

n	x_n	$e_n = x_n - 1$	$\lvert e_n\rvert/\lvert e_{n-1}\rvert^2$
0	1.500000000	0.5	
1	0.707106769	−0.292893231	1.1715
2	0.896018863	−0.103981137	1.2120
3	0.986690760	−0.013309240	1.2309
4	0.999781489	−0.000218510	1.2335
5	0.999999940	−0.000000059	1.2483

Remark: The scheme cannot be used with a different x_0 to converge to the fixed point $x^* = 0$ since $g'(0) = \pi/2 > 1$. Using $x_0 < 1$, the scheme converges to $x^* = -1$.

3.3 Newton-Raphson method

The **Newton-Raphson method**[5], or simply **Newton's method**, is one of the most powerful and well-known methods for solving a root-finding problem of the form $f(x) = 0$.

Let x^* be a solution of $f(x) = 0$ and let x_n be a close approximation such that $|x_n - x^*|$ is small. We assume $f \in C^2[a,b]$ and x^*, $x_n \in [a,b]$ and use Taylor's theorem to obtain

$$f(x^*) = f(x_n) + f'(x_n)(x^* - x_n) + \frac{1}{2}f''(\xi(x_n))(x^* - x_n)^2$$

where $\xi(x_n)$ lies between x^* and x_n.

Now if $|x_n - x^*| \ll 1$ then $|x_n - x^*|^2 \ll |x_n - x^*|$ justifying the approximation

$$0 \approx f(x_n) + f'(x_n)(x^* - x_n).$$

Solving this equation for x^* results in

$$x^* \approx x_n - \frac{f(x_n)}{f'(x_n)}.$$

[5] developed in the 17th century by Isaac Newton (1641-1727) and Joseph Raphson (1648-1715)

This approximation suggests we define the RHS as a new approximation to x^*, leading to **Newton's method**:

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

It corresponds to a fixed-point iteration of the form $x_{n+1} = g(x_n)$ under the definition

$$g(x) = x - \frac{f(x)}{f'(x)}. \tag{6}$$

Remark: Newton's method can also be derived/motivated by graphical considerations.

3.3.1 Order of convergence of Newton's method

We know from a theorem in the last section that the order of convergence of a fixed-point iteration scheme is determined by how many of the first derivatives of the function $g(x)$ vanish at the fixed point x^*. For Newton's method we calculate from (6) that

$$g'(x) = 1 - \frac{f'^2 - ff''}{f'^2} = \frac{ff''}{f'^2}, \qquad g''(x) = \frac{f'^2(f'f'' + ff''') - ff''(2f'f'')}{f'^4}. \tag{7}$$

If we assume that x^* is a **simple zero** of $f(x)$, meaning that $f(x^*) = 0$ and $f'(x^*) \neq 0$ then

$$g'(x^*) = \frac{f(x^*)f''(x^*)}{[f'(x^*)]^2} = 0$$

whilst

$$g''(x^*) = \frac{f'(x^*)^3 f''(x^*)}{f'(x^*)^4} = \frac{f''(x^*)}{f'(x^*)}.$$

The second derivative $g''(x^*)$ is (in general) not zero and so of Newton's method is typically of **quadratic order** (for simple zeros). This is considered fast: $g_4(x)$ earlier was an example of the application of Newton's method to the function $f(x) = x^2 - x - 1$.

We now ask how robust the method is. In particular, how does the choice of x_0 affects the success of the method.

3.3.2 Region of convergence of Newton's method

Theorem: Let x^* be the fixed point of $g(x)$ with $g'(x^*) = 0$ whilst we assume $|g''(x)| < M$ on an open interval, I, containing x^*. Then $\exists\, \delta > 0$ s.t. for all $x_0 \in [x^* - \delta, x^* + \delta] \subset I$ the sequence $x_{n+1} = g(x_n)$ converges at least quadratically to x^*. Furthermore, for n sufficiently large,

$$|x_{n+1} - x^*| < \frac{M}{2}|x_n - x^*|^2.$$

Proof: Choose $k \in (0, 1)$ and $\delta > 0$ s.t. for $x \in [x^* - \delta, x^* + \delta] \subset I$, $|g'(x)| \leq k$ (this is possible since $g'(x^*) = 0$ and $g'(x)$ is assumed continuous). Since $k < 1$, $g(x) \in [x^* - \delta, x^* + \delta]$. I.e. the conditions of the FPT are met and hence the method converges for $x_0 \in [x^* - \delta, x^* + \delta]$.

We found previously using Taylor's theorem that when $g(x^*) = 0$,

$$|x_{n+1} - x^*| = \frac{1}{2}|g''(\xi_n)|\,|x_n - x^*|^2$$

where ξ_n lies between x_n and x^*. Using the bound for the second derivative we obtain

$$|e_{n+1}| = \frac{1}{2}|g''(\xi_n)|\,|e_n^2| < \frac{M}{2}|e_n|^2.$$

Provided $|e_n| < 2/M$, $|e_{n+1}| < |e_n|$ and and hence quadratic convergence is guaranteed.

3.3.3 Non-simple zeros

Defn: A solution x^* of $f(x) = 0$ is said to be a **zero of multiplicity** m if

$$0 = f(x^*) = f'(x^*) = \ldots = f^{(m-1)}(x^*), \qquad f^{(m)}(x^*) \neq 0.$$

Zeros with $m = 1$ (the usual case, where $f'(x^*) \neq 0$) are called **simple zeros**.

We consider now how zeros of multiplicity $m > 1$ (non-simple zeros) affect the convergence of Newton's method. By Taylor's theorem,

$$(8) \qquad f(x) = \sum_{k=0}^{m-1} \frac{f^{(k)}(x^*)}{k!}(x - x^*)^k + \frac{f^{(m)}(\xi(x))}{m!}(x - x^*)^m \equiv q(x)\,(x - x^*)^m$$

since all the terms in the sum now vanish and where $\xi(x) \in (x, x^*)$ and $q(x) = f^{(m)}(\xi(x))/m!$. such that $q(x^*) \neq 0$.

It follows that the derivatives of $f(x)$ are given by

$$\begin{aligned} f'(x) &= mq(x)(x-x^*)^{m-1} + q'(x)(x-x^*)^m, \\ f''(x) &= m(m-1)q(x)(x-x^*)^{m-2} + 2mq'(x)(x-x^*)^{m-1} + q''(x)(x-x^*)^m. \end{aligned}$$

Using (7) for $m > 1$ we have, with the expressions above for f, f' and f'' and taking the limits

$$g'(x^*) = \lim_{x \to x^*} \frac{f(x)f''(x)}{[f'(x)]^2} = \frac{m(m-1)}{m^2} = 1 - \frac{1}{m} \neq 0.$$

That is, if $m > 1$, and the root is non-simple, then Newton's method converges only linearly.

3.3.4 Example

Consider $f(x) = \mathrm{e}^x - x - 1$, which is zero at $x^* = 0$. We also note that

$$f'(x) = \mathrm{e}^x - 1 = 0, \qquad f''(x) = \mathrm{e}^x \neq 0, \qquad \text{at } x^* = 0$$

so $x^* = 0$ is zero of multiplicity two ($m = 2$). The table below shows how Newton's method converges only linearly for this function.

n	x_n	$\|e_n\|/\|e_{n-1}\|$
0	1.00000000	
1	0.581976771	0.581976771
2	0.319055110	0.548226535
3	0.167996019	0.526542306
4	0.086348965	0.513994098
5	0.043796084	0.507198691
6	0.022057412	0.503639042

3.3.5 Reinstating lost quadratic convergence

In the case a root x^* of $f(x)$ is not simple resulting in linear convergence of Newton's Method, quadratic convergence can be restored by applying Newton's method to

$$F(x) = f(x)/f'(x)$$

which implies the following iteration scheme

$$\text{(9)} \qquad x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)}, \qquad \text{where } F' = \frac{f'^2 - ff''}{f'^2}$$

and so

$$\text{(10)} \qquad x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{f'(x_n)f'(x_n) - f(x_n)f''(x_n)}$$

in terms of the original function.

Proof: If $f(x)$ has a zero of multiplicity m at $x = x^*$ then $f(x) = q(x)(x - x^*)^m$ with $q(x^*) \neq 0$, from (8). It follows that

$$F(x) = \frac{q(x)(x - x^*)^m}{mq(x)(x - x^*)^{m-1} + q'(x)(x - x^*)^m} = Q(x)(x - x^*)$$

where

$$Q(x) = \frac{q(x)}{mq(x) + q'(x)\,(x - x^*)} \to \frac{1}{m} \neq 0 \qquad \text{as } x \to x^*.$$

Thus $F(x)$ has a simple zero at $x = x^*$, the same as the zero of $f(x)$, and (9) therefore has quadratic convergence.

Remark: Since $f(x_n)$ and $f'(x_n)$ are very small as $x_n \to x^*$, (10) is a scheme in which the product of small numbers are divided by small numbers and this is susceptible to rounding errors.

3.4 Aitken's method

Aitken's method (or the Δ^2-method) can be used to accelerate the convergence of *any linearly convergent sequence*, regardless of its origin.

Suppose that $\{x_0, x_1, \ldots\}$ is a linearly convergent sequence with limit x^* and asymptotic error constant $\lambda < 1$. Then

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|} = \lambda < 1.$$

We therefore expect that

$$\frac{|x_{n+1} - x^*|}{|x_n - x^*|} \approx \lambda$$

for sufficiently large n. The approximation would therefore also hold when n is replaced by $n+1$, and so, dropping the modulus (with a certain amount of care), we claim that

$$\frac{x_{n+1} - x^*}{x_n - x^*} \approx \pm\lambda \approx \frac{x_{n+2} - x^*}{x_{n+1} - x^*}.$$

Multiplying across by the denominators gives

$$\begin{aligned} & (x_{n+1} - x^*)^2 \approx (x_{n+2} - x^*)\,(x_n - x^*) \\ \implies \quad & x_{n+1}^2 - 2x_{n+1}x^* + (x^*)^2 \approx x_{n+2}x_n - x^*(x_{n+2} + x_n) + (x^*)^2 \end{aligned}$$

and solving for x^* results in

$$(11) \qquad x^* \approx \frac{x_{n+2}x_n - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n} = x_n - \frac{x_{n+1}^2 - 2x_{n+1}x_n + x_n^2}{x_{n+2} - 2x_{n+1} + x_n} = x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}.$$

This can be written more compactly using the notation of a **forward difference** operator

$$\Delta x_n = x_{n+1} - x_n$$

since then we have

$$\Delta^2 x_n = \Delta(\Delta x_n) = \Delta(x_{n+1} - x_n) = \Delta x_{n+1} - \Delta x_n = x_{n+2} - 2x_{n+1} + x_n$$

and (11) becomes

$$x^* \approx x_n - \frac{(\Delta x_n)^2}{\Delta^2 x_n}.$$

Thus, we denote a new sequence $\{\hat{x}_n\}$ by defining

$$\hat{x}_n \approx x_n - \frac{(\Delta x_n)^2}{\Delta^2 x_n}$$

whose convergence can be shown to be faster than linear (see, for example, Burden & Faires).

3.4.1 Example

Use Aitken's Δ^2-method to improve the convergence to $x^* = 0$ of the linearly convergent sequence x_n for Newton's method applied to $f(x) = \mathrm{e}^x - x - 1$. The final column shows that $\hat{x}_n$ converges quadratically to zero compared to the original sequence. Note the onset of some round-off errors which are due dividing small numbers $(\Delta x_n)^2$ by small numbers $\Delta^2 x_n$ (also later in Section 5).

n	x_n	$\hat{x}_n$	$\lvert\hat{x}_n\rvert/\lvert x_{n+2}\rvert^2$
0	1.00000000	-0.126638770	1.2440
1	0.58197677	-0.035993993	1.2753
2	0.31905511	-0.009689718	1.2995
3	0.16799601	-0.002521470	1.3145
4	0.08634896	-0.000646777	1.3293
5	0.04379608	-0.000167448	1.3669
6	0.02205741		
7	0.01106787		

3.5 Solving systems of nonlinear equations: Multi-dimensional Newton's method

We consider now methods for solving *systems* of n nonlinear equations for n variables which we express as

$$f_i(x_1, x_2, \ldots, x_n) = 0 \qquad \text{for } i = 1, \ldots, n \tag{12}$$

or, in vector notation,

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \qquad \mathbf{f} = (f_1, f_2, \ldots, f_n)^{\mathrm{T}}, \quad \mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}.$$

This is generally a difficult problem and, unlike linear systems of equations, and there are no good general methods systems of nonlinear equations.

To understand this complexity consider the case in $(n = 2)$ two dimensions where (12) can be expressed ($f_1 = f$, $f_2 = g$, $x_1 = x$, $x_2 = y$) as

$$f(x, y) = 0 \qquad \text{and} \qquad g(x, y) = 0.$$

Both equations define curves in the xy-plane and the simulateous solution of both equations lie at the intersections of these curves. Thus, the simplicity in one dimension of a line crossing the axis is lost. The complexity of the picture described above increases further for $n > 2$.

However, once the neighbourhood of a root is located, there are methods of converging to the root. **Newton's method** is the most widely used of these[6]

Let the solution of (12) be given by $\mathbf{x}^*$, such that $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$. Assume that $\mathbf{x}^{(m)}$ is a good approximation to the solution such that $|\mathbf{x}^{(m)} - \mathbf{x}^*|$ is small. Then we use Taylor's expansion in $n > 1$ dimensions[7]

$$f_i(\mathbf{x}^*) = f_i(\mathbf{x}^{(m)}) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(m)})(x_j^* - x_j^{(m)}) + \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} \frac{\partial^2 f_i}{\partial x_j \partial x_k}(x_j^* - x_j^{(m)})(x_k^* - x_k^{(m)}) + \text{h.o.t}$$

(higher order terms) for $i = 1, 2, \ldots, n$. The double sum includes products of terms assumed to be sufficiently small that they can be neglected.

What remains can be expressed in matrix/vector notation as

$$\mathbf{f}(\mathbf{x}^*) \approx \mathbf{f}(\mathbf{x}^{(m)}) + J(\mathbf{x}^{(m)})(\mathbf{x}^* - \mathbf{x}^{(m)})$$

where $J(\mathbf{x})$ is the **Jacobian matrix**,

$$J(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1}(\mathbf{x}) & \ldots & \dfrac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & & \vdots \\ \dfrac{\partial f_n}{\partial x_1}(\mathbf{x}) & \ldots & \dfrac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix}.$$

[6] One can also generalise fixed-point methods to higher dimensions.

[7] (Multivariable Calculus course)

Since $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$ we have

$$\mathbf{x}^* \approx \mathbf{x}^{(m)} - J^{-1}(\mathbf{x}^{(m)})\mathbf{f}(\mathbf{x}^{(m)}).$$

This suggests the following iteration scheme (**multi-dimensional Newton's method**)

$$\mathbf{x}^{(m+1)} = \mathbf{x}^{(m)} - J^{-1}(\mathbf{x}^{(m)})\mathbf{f}(\mathbf{x}^{(m)}) \qquad m = 0, 1, 2, \ldots$$

subject to a sufficiently good initial guess $\mathbf{x}^{(0)}$.

Note that we can avoid evaluating the inverse matrix J^{-1} by setting

$$\mathbf{x}^{(m+1)} = \mathbf{x}^{(m)} - \mathbf{y}^{(m)}$$

where $\mathbf{y}^{(m)}$ can be determined as the solution of

$$J(\mathbf{x}^{(m)})\mathbf{y}^{(m)} = \mathbf{f}(\mathbf{x}^{(m)})$$

using, for e.g., Gaussian elimination.

Remarks: As in one dimension, Newton's method has, in general, quadratic convergence (no proof), but there are two disadvantages of the method:

(i) The method may not converge if the initial approximation is not good enough;

(ii) The Jacobian matrix of partial derivatives, J, must be evaluated at each step. Evaluating the Jacobian of $\mathbf{f}(\mathbf{x})$ may be difficult analytically, numerical expensive or impossible (if $\mathbf{f}(\mathbf{x})$ is not explicit).

3.5.1 Example

Consider

$$x^2 + y^2 = 4, \qquad xy = 1.$$

First, write $f_1(x, y) = x^2 + y^2 - 4$ and $f_2(x, y) = xy - 1$ so that system of equations is represented by $f_1 = 0$, $f_2 = 0$.

Note that the curves of $f_1 = 0$ and f_2 are represented by a circle of radius 2 and hyperbolae passing through $(1, 1)$ and $(-1, -1)$. In this way, we can see there are four roots placed symmetrically about the x- and y-axes.

We can find these roots exactly by eliminating between the two equations. Substituting $y = 1/x$ into the first equation gives

$$x^4 - 4x^2 + 1 = 0$$

which is a quadratic for x^2 and so

$$x^2 = (4 \pm \sqrt{16-4})/2 = 2 \pm \sqrt{3}$$

meaning that

$$x^* = \pm\sqrt{2 \pm \sqrt{3}} = \pm 1.9319,\ \pm 0.5176$$

and then $y^* = \pm 0.5176,\ \pm 1.9319$ are the four roots.

Now let's use Newton's method. We need

$$J = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix} \qquad \rightarrow \qquad J^{-1} = \frac{1}{2x^2 - 2y^2} \begin{bmatrix} x & -2y \\ -y & 2x \end{bmatrix}.$$

Note: this fails when $\det\{J\} = 0$ and this happens when $x = \pm y$. The vanishing of the determinant of the Jacobian indicates that $\mathbf{x}$ lies on a critical point and is the multi-dimensional equivalent to the vanishing of $f'(x)$, which results in the failure of the one-dimensional Newton's method.

Choose, for example, $\mathbf{x}^{(0)} = (2,0)^T$ which is close to one $(+/+)$ of the four roots. Then Newton's method gives us

$$\mathbf{x}^{(1)} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} - \frac{1}{8} \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix} \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

the last vector being $\mathbf{f}$ evaluated at $\mathbf{x}^{(0)}$. Then we find

$$\mathbf{x}^{(1)} = \begin{pmatrix} 2 \\ 0.5 \end{pmatrix}, \qquad \mathbf{x}^{(2)} = \begin{pmatrix} 1.9333 \\ 0.5167 \end{pmatrix}, \qquad \mathbf{x}^{(3)} = \begin{pmatrix} 1.9319 \\ 0.5176 \end{pmatrix}$$

and we have converged to 4 significant figures after 3 iterations !

3.6 The steepest descent method (the gradient method)

THIS SECTION IS OMITTED FROM THE COURSE IN 2025

This is a so-called **global method** that does not necessarily require a good initial approximation, but the convergence can be quite slow. The idea is to transform the root finding problem into a **minimisation problem**. Consider

$$g(\mathbf{x}) = \sum_{i=1}^{n} f_i^2(\mathbf{x}) \geq 0. \tag{13}$$

The function $g(\mathbf{x})$ has a minimum at $\mathbf{x}^*$ with value $g(\mathbf{x}^*) = 0$ if and only if $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$.

The method is described by the iterative step

$$\mathbf{x}^{(m+1)} = \mathbf{x}^{(m)} - \alpha_m \nabla g(\mathbf{x}^{(m)}), \qquad m \geq 0 \tag{14}$$

for a given initial guess $\mathbf{x}^{(0)}$ where

$$-\nabla g = \left(-\frac{\partial g}{\partial x_1}, \ldots, -\frac{\partial g}{\partial x_n}\right)^{\mathrm{T}}$$

is minus the gradient of g and *points in the direction of steepest descent*[8]

In (14), α_m determines how far we move from the position $\mathbf{x}^{(m)}$ along the straight path in the direction of steepest descent. This is determined by requiring the value of g on that path is minimised. In other words we should only move in the direction $-\nabla g$ for as long as we are descending.

I.e. $\alpha = \alpha_m$ is determined from

$$0 = \frac{\mathrm{d}h}{\mathrm{d}\alpha} \tag{15}$$

where

$$h(\alpha) = g[\mathbf{x}^{(m)} - \alpha \nabla g(\mathbf{x}^{(m)})].$$

Determining solutions of (15) are often complicated and crude methods are often sufficient as one need not be precise about the value of α_m to make the method work. For e.g. on might calculate $h(\alpha)$ for a few values of α and use the last value before $h(\alpha)$ starts to increase.

[8] (Multivariable Calculus) **E.g.:** if $\mathbf{x} = (x, y)$ then $g(x, y)$ can be interpreted as the height of a three-dimensional surface and, for any point (x, y), $-\nabla g$ points in the horizontal (compass) direction in which you will descend most rapidly. It is perpendicular to the contours of the surface on a map.

Remark: The method always converges to **a** minimum. This may be the **global minimum** where $g(\mathbf{x}) = 0$, or it may be a **local minimum** where $g(\mathbf{x}) \neq 0$. That is, it is possible for the method to fail to find the root $\mathbf{x}^*$ if convergence is not to the global miniumum.

3.6.1 Example

Apply the steepest descent method to the system of two equations

$$x_1^2 + x_2^2 - 4 = 0, \qquad x_1 x_2 - 1 = 0$$

with an initial guess $\mathbf{x}^{(0)} = (2, 0)$ (the same example as Section 3.5.1).

Using (13) we define

$$g(x_1, x_2) = f_1^2 + f_2^2 = (x_1^2 + x_2^2 - 4)^2 + (x_1 x_2 - 1)^2$$

so that g is minimised to zero at the solution. Now

$$\nabla g = \left(\begin{array}{c} 4x_1(x_1^2 + x_2^2 - 4) + 2x_2(x_1 x_2 - 1) \\ 4x_2(x_1^2 + x_2^2 - 4) + 2x_1(x_1 x_2 - 1) \end{array} \right).$$

So, the first iteration is

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \alpha_0 \nabla g(\mathbf{x}^{(0)}) = \left(\begin{array}{c} 2 \\ 0 \end{array} \right) + \alpha_0 \left(\begin{array}{c} 0 \\ 4 \end{array} \right)$$

after substitution of $\mathbf{x}^{(0)} = (2, 0)$. In order to find α_0 we consider

$$h(\alpha) = g[(2, 4\alpha)^T] = (16\alpha^2)^2 + (8\alpha - 1)^2 = 256\alpha^4 + 64\alpha^2 - 16\alpha + 1.$$

To find the minimum, set

$$0 = h'(\alpha) = 4 \times 256\alpha^3 + 2 \times 64\alpha - 16$$

or

$$0 = 64\alpha^3 + 8\alpha - 1$$

and we can determine numerically that is has a root $\alpha_0 \approx 0.1133$. Hence

$$\mathbf{x}^{(1)} \approx \left(\begin{array}{c} 2 \\ 0.4532 \end{array} \right)$$

as our first iteration towards the root. The next step is the same as above, but using $\mathbf{x}^{(1)}$ instead of $\mathbf{x}^{(0)}$. This is now a numerical process, not something that is feasible by hand. For completeness we find

$$\mathbf{x}^{(2)} \approx \begin{pmatrix} 1.946 \\ 0.501 \end{pmatrix}.$$

Recall the exact root was $\mathbf{x}^* = (1.9319, 0.5176)^T$.

Remark: Convergence is slow. Why ? gradients of g are zero at the solution, $\mathbf{x}^*$, which is when 1D Newton's method became linearly convergent.

4 Interpolation: approximation of curves by polynomials

A continuous function is an abstract idea and a computer only records information about a function $f(x)$ at discrete points $x = x_i$, $i = 0, \ldots, n$ (say). Interpolation addresses the fact that we often need to know about $f(x)$ at values $x \neq x_i$. Interpolation addresses how to join data points on a graph with a smooth curve.

4.1 Polynomial approximation

If there are $n+1$ points $(x_i, f(x_i))$ then we imagine we can fit a polynomial of degree n through the points.[9] I.e. we represent $f(x)$ by

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0.$$

And then determine a_i by setting $P_n(x_i) = f(x_i)$ at $x = x_i$. I.e.

$$a_n x_i^n + a_{n-1}x_i^{n-1} + \ldots + a_1 x_i + a_0 = f(x_i)$$

for $i = 0, 1, \ldots, n$. I.e. we have $n+1$ equations for $n+1$ unknowns. We can write this as

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ & & & & \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \\ a_n \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \\ f(x_n) \end{pmatrix}.$$

This matrix is a special type called a **Vandermonde matrix**. Suddenly this seems like a difficult task... but

4.2 Lagrange interpolating polynomials

Consider first $n = 1$ (motivation). Define

$$L_{1,0}(x) = \frac{x - x_1}{x_0 - x_1}, \qquad L_{1,1}(x) = \frac{x - x_0}{x_1 - x_0},$$

[9] This is not the only way of curve fitting: see Burden and Faires

s.t. $L_{1,0}(x_0) = 1$, $L_{1,0}(x_1) = 0$ and similarly for the other function. Then

$$P_1(x) = L_{1,0}(x)f(x_0) + L_{1,1}(x)f(x_1) \tag{16}$$

is indeed a degree 1 polynomial which passes through $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

Now consider

$$L_{n,k}(x) = \prod_{\substack{i=0\\i\neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)}$$

is a polynomial of degree n s.t.

$$L_{n,k}(x_j) = \left\{ \begin{array}{ll} 0, & \text{if } j \neq k \\ 1, & \text{if } j = k \end{array} \right\} = \delta_{jk}$$

(the **Kronecker delta**.) Then

$$P_n(x) = \sum_{k=0}^{n} L_{n,k}(x)f(x_k) \equiv \sum_{k=0}^{n} f(x_k) \prod_{\substack{i=0\\i\neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)} \tag{17}$$

is a polynomial of degree n s.t.

$$P_n(x_j) = \sum_{k=0}^{n} L_{n,k}(x_j)f(x_k) = \sum_{k=0}^{n} \delta_{jk} f(x_k) = f(x_j)$$

for $j = 0, 1, \ldots$. I.e. we have determined the interpolating polynomial without solving a matrix equation.[10]

4.2.1 Error

Q: How good is $P_n(x)$ for $x \neq x_k$?

Theorem: Let $f \in C^{n+1}[a, b]$ (f has $n+1$ cts derivatives) and $x_0, \ldots, x_n \in [a, b]$. Then

$$f(x) = P_n(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \tag{18}$$

[10] There are other ways of doing this: see divided differences in Burden and Faires

for some $\xi(x) \in [x_0, x_n] \subset [a, b]$ (ξ is generally unknown).

A: The final term in (18) represents the error in the approximation.

Proof: We first note that (18) is true if $x = x_k$, $k = 0, 1, \dots, n$. So consider $x \neq x_k$ and define g by

$$g(t) = f(t) - P_n(t) - (f(x) - P_n(x)) \prod_{i=0}^{n} \frac{(t - x_i)}{(x - x_i)}.$$

Note that since P_n is a polynomial and $f \in C^{n+1}[a, b]$, then $g \in C^{n+1}[a, b]$ also. First, note that when $t = x_k$,

$$g(x_k) = 0 - (f(x) - P_n(x)) \times 0 = 0$$

for $k = 0, 1, \dots, n$. Second, note that when $t = x$

$$g(x) = (f(x) - P_n(x))(1 - 1) = 0.$$

Therefore $g(t)$ is zero at $x, x_0, x_1, \dots, x_n$, i.e. at $n + 2$ distinct points. By the generalised Rolle's theorem, $\exists \xi \in (a, b)$ s.t.

$$0 = g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - P_n^{(n+1)}(\xi) - (f(x) - P_n(x)) \left. \frac{d^{(n+1)}}{dt^{(n+1)}} \prod_{i=0}^{n} \frac{(t - x_i)}{(x - x_i)} \right|_{t=\xi}.$$

First, since $P_n(x)$ is a polynomial of degree n, its $(n + 1)$th derivative is zero. Also, the final product is a polynomial in t of degree $n + 1$ and its leading term is $t^{(n+1)}$. So its $(n + 1)$th derivative is

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

We can now rearrange what is left to give (18).

4.2.2 Example

Consider fitting a polynomial of degree n to the curve $f(x) = 1/x$ at $x_i = 1 + i/n$, $i = 0, \dots, n$ (that is at equally spaced points between $x = 1$ and $x = 2$):

(i) Find $P_2(x)$; (ii) determine a bound on the maximum error $E = \max_{1\le x\le 2}\{|f(x) - P_2(x)|\}$; (iii) determine the actual maximum error, E.

(i) From the definition of the Lagrange interpolating polynomial

$$P_2(x) = f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

and with $x_0 = 1$, $x_1 = 3/2$, $x_2 = 2$ we have

$$P_2(x) = 1\frac{(x-3/2)(x-2)}{(1-3/2)(1-2)} + \frac{2}{3}\frac{(x-1)(x-2)}{(3/2-1)(3/2-2)} + \frac{1}{2}\frac{(x-1)(x-3/2)}{(2-1)(2-3/2)}.$$

We just need to tidy this up:

$$P_2(x) = \frac{13}{6} - \frac{3}{2}x + \frac{1}{3}x^2.$$

(ii)

$$E = \max_{1\le x\le 2}\{|f(x) - P_n(x)|\} \le \max_{1\le x\le 2}\{|f^{(n+1)}(x)|\}\frac{1}{(n+1)!}\max_{1\le x\le 2}\left|\prod_{i=0}^{n}(x-x_i)\right|.$$

So first we need $f'(x) = -1/x^2$, $f''(x) = 2/x^3$ and so on. We can see that $f^{(n+1)}(x) = (n+1)!(-1)^n/x^{n+2}$. Its maximum is when $x = 1$ and so now

$$E \le \max_{1\le x\le 2}\{|W(x)|\}, \qquad \text{where} \qquad W(x) = \prod_{i=0}^{n}(x-x_i).$$

To determine the maximum we differentiate (as usual).

For the case given: $n = 2$, $x_0 = 1$, $x_1 = 3/2$, $x_2 = 2$ and

$$W(x) = (x-1)(x-3/2)(x-2) = x^3 - \frac{9}{2}x^2 + \frac{13}{2}x - 3$$

so

$$W'(x) = 3x^2 - 9x + \frac{13}{2}.$$

We solve $W'(x) = 0$ to give $x = 3/2 \pm \sqrt{3}/6$ and both lie in $[1, 2]$. Now $W(3/2 \pm \sqrt{3}/6) = \mp\sqrt{3}/36$,

$$E \le \sqrt{3}/36 = 0.04811\ldots$$

(iii) Note, in (ii) this was the *bound* on the error, because we didn't know ξ. The actual maximum error is

$$E = \max_{1 \le x \le 2} \left\{ \left| \frac{1}{x} - \left(\frac{13}{6} - \frac{3x}{2} + \frac{x^2}{3} \right) \right| \right\}.$$

This requires us to solve

$$\frac{-1}{x^2} - \frac{2}{3}x + \frac{3}{2} = 0$$

which is a cubic (and so I find the answer numerically[11]: $x \approx 1.1890$ and $x \approx 1.7726$ are both roots in $1 \le x \le 2$.) This gives

$$E = \max\{0.01336, 0.009006\} = 0.01336.$$

This is (thankfully) less than the bound on the error !

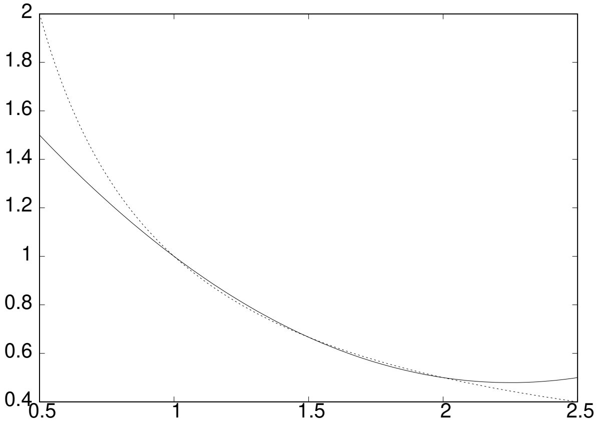


Figure 1: Plot of $1/x$ and $P_2(x)$

Remark: The method of interpolation can suffer from instabilties (not due to rounding errors). That is, as n increases and we add more interpolation points we want the resulting nth degree polynomial to fit the data better, but this doesn't always happen (see Burden & Faires). One practical method is to fit a lower degree polynomial to the data.

[11] How ? In Section 3

5 Differentiation

This section refers to the numerical approximation to the derivative of a function $f(x)$, say. This is required if f is not known explicitly and is known or can only be calculated at discrete points. The approximation of derivatives is also essential in the numerical solution of differential equations (later in the course).

Our starting point is the definition of the derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

and motivates what comes next. An equivalent approach would have been to use interpolating polynomials to approximate derivatives and results in the same approximations with $x_1 = x_0 \pm h$ and $x_2 = x_0 \mp h$.

5.1 Difference formulae

Defn: (i) The **forward difference approximation** at $x = x_0$ is

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h}$$

for $h > 0$ (h will be used throughout the remainder of the course, always as a *positive* **step size**).

(ii) The **backward difference approximation** is defined as

$$f'(x_0) \approx \frac{f(x_0) - f(x_0-h)}{h}.$$

Q: We expect that as $h \to 0$ the approximations improve. How ?

A: Assume that $f \in C^2[a,b]$, $x_0, x_0+h \in [a,b]$ and $h > 0$. Then from Taylor's theorem

$$f(x_0+h) = f(x_0) + h\, f'(x_0) + \frac{h^2}{2} f''(\xi)$$

for some $\xi \in (x_0, x_0+h)$. The last term is a remainder term and collects together all the terms in the Taylor series approximation.

Solving this equation for $f'(x_0)$ results in

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} - \frac{h}{2} f''(\xi).$$

and so the error associated with the forward difference approximation is

$$E \equiv E(h) = -\frac{h}{2} f''(\xi) = O(h)$$

and we say that the error is of **order** h. Note that ξ depends on h also.

Remark: We refer to E as the **truncation error** because it arises from the truncation of the Taylor series expansion.

Exercise: Find the truncation error in the backward difference approximation and show it is also $O(h)$.

Defn: Since $E = O(h)$ we say that the forward/backward difference approximation are are **first-order schemes**.

5.1.1 Central differences

If we use more than two evaluations of f we should get better approximations.

Defn: A **three-point central difference approximation** for $f'(x_0)$ is assumed of the form

$$f'(x_0) \approx \alpha f(x_0+h) + \beta f(x_0) + \gamma f(x_0-h).$$

The aim now is to determine α, β, γ to minimise the error (as a function of h) between the exact value of the derivative and its approximation:

$$E(h) = f'(x_0) - \left[\alpha f(x_0+h) + \beta f(x_0) + \gamma f(x_0-h)\right].$$

We assume $f \in C^3[a,b]$, $x_0, x_0 \pm h \in [a,b]$, and $h > 0$ and apply Taylor's theorem

$$\begin{aligned} E(h) &= f'(x_0) - \alpha\left[f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(\xi_1)\right] \\ &\quad -\beta f(x_0) - \gamma\left[f(x_0) - hf'(x_0) + \frac{h^2}{2}f''(x_0) - \frac{h^3}{6}f'''(\xi_2)\right] \end{aligned} \tag{19}$$

and $\xi_1 \in (x_0, x_0+h)$, $\xi_2 \in (x_0-h, x_0)$. There's some skill/luck in deciding where to truncate the Taylor series. You only know if you've done it right when you get to the end of the calculation.

Now we have

$$E(h) = f(x_0)(\alpha+\beta+\gamma) + f'(x_0)(1 - h\alpha + h\gamma) - \frac{h^2}{2}f''(x_0)(\alpha+\gamma) - \frac{h^3}{6}[\alpha f'''(\xi_1) - \gamma f'''(\xi_2)].$$

We set the coefficients of $f(x_0)$, $f'(x_0)$ and $f''(x_0)$ to zero to give three equations for our three unknowns

$$\alpha+\beta+\gamma = 0, \qquad h(\alpha-\gamma) = 1, \qquad \alpha+\gamma = 0$$

whose solution is

$$\alpha = \frac{1}{2h}, \qquad \beta = 0, \qquad \gamma = -\frac{1}{2h}.$$

Thus we have found that

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} + E(h) \tag{20}$$

where the truncation error is

$$E(h) = -\frac{h^2}{12}[f'''(\xi_1) + f'''(\xi_2)].$$

Using the intermediate value theorem[12] (IVT) we can conclude that there is a point $\xi \in [\xi_2, \xi_1]$ that takes the value

$$f'''(\xi) = \frac{1}{2}[f'''(\xi_1) + f'''(\xi_2)]$$

and thus $\xi \in (x_0 - h, x_0 + h)$.

[12] That is, if $f'''(x)$ is continuous and takes the values of $f'''(\xi_1)$ and $f'''(\xi_2)$ at $x = \xi_{1,2}$ then it must pass through the averages of the two values for some ξ between ξ_1 and ξ_2.

Therefore we find that the error of the approximation is

$$E(h) = -\frac{h^2}{6} f'''(\xi) = O(h^2),$$

and the approximation (20) is thus said to be a **second-order scheme**.

Remark: (20) is called the **central-difference approximation** to the first derivative and in spite of allowing ourselves three points to evaluate $f(x)$, the scheme only requires two.

5.1.2 Second derivatives

Defn: The **three-point central difference approximation for** $f''(x_0)$ follows in a similar way. Note that we certainly need at least three evaluations of f to determine curvature. We write

$$f''(x_0) \approx \alpha f(x_0 + h) + \beta f(x_0) + \gamma f(x_0 - h).$$

Then the error is

$$E = f''(x_0) - [\alpha f(x_0 + h) + \beta f(x_0) + \gamma f(x_0 - h)]$$

and we aim to choose α, β, γ to minimise E as a function of h.

Assuming $f \in C^4[a, b]$, $x_0, x_0 \pm h \in [a, b]$, and $h > 0$: we now have to include one more term in the Taylor expansion (for reasons that become clear)

$$\begin{aligned} E(h) &= f''(x_0) - \alpha \left[f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{6} f'''(x_0) + \frac{h^4}{24} f^{(iv)}(\xi_1) \right] \\ &\qquad -\beta f(x_0) - \gamma \left[f(x_0) - hf'(x_0) + \frac{h^2}{2} f''(x_0) - \frac{h^3}{6} f'''(x_0) + \frac{h^4}{24} f^{(iv)}(\xi_2) \right]. \end{aligned} \tag{21}$$

Now we have

$$\begin{aligned} E(h) &= f(x_0)(\alpha + \beta + \gamma) + hf'(x_0)(\alpha - \gamma) + \frac{1}{2} f''(x_0)(1 + h^2\alpha + h^2\gamma) - \frac{h^3}{6} f'''(x_0)(\alpha - \gamma) \\ &\qquad - \frac{h^4}{24} [\alpha f^{(iv)}(\xi_1) + \gamma f^{(iv)}(\xi_2)]. \end{aligned}$$

We eliminate coefficients of $f(x_0)$, $f'(x_0)$, $f''(x_0)$ to give three equations for the three unknowns:

$$\alpha+\beta+\gamma=0, \qquad \alpha-\gamma=0, \qquad \frac{h^2}{2}(\alpha+\gamma)=1.$$

The solution to this system of equations is given by

$$\alpha=\frac{1}{h^2}, \qquad \beta=-\frac{2}{h^2}, \qquad \gamma=\frac{1}{h^2}.$$

Note that the coefficient of $f'''(x_0)$ also vanishes in this case (this is why we went to an additional order in the expansion) and this means that

$$f''(x_0)=\frac{f(x_0+h)-2f(x_0)+f(x_0-h)}{h^2}+E(h)$$

where

$$E(h)=-\frac{h^2}{24}\left[f^{(iv)}(\xi_1)+f^{(iv)}(\xi_2)\right]=-\frac{h^2}{12}f^{(iv)}(\xi)=O(h^2)$$

with $\xi\in(x_0-h,x_0+h)$ after using the intermediate value theorem (IVT) as in the previous section. This approximation is a **second-order scheme.**

Remark: There are many possibilities to generalise these approximations including:

(i) Forward difference formulas (using for e.g. x_0, x_0+h, x_0+2h);

(ii) Backward difference formulas (using for e.g. x_0, x_0-h, x_0-2h);

(iii) Formulae which use more points or points distributed unevenly.

5.2 Round-off errors

According to the formulae we have developed, theoretical errors, $E(h)$, can be made arbitrarily small by choosing h sufficiently small. In practice, however, numerical calculations on a computer with finite-precision accuracy limits the size of h because of round-off errors. This is true in all such schemes we encounter whose accuracy is related to a step size, h, but numerical differentiation is particularly vulnerable to rounding errors because of the nature of the calculations involved. That is, it requires us to take differences of almost equal numbers to leave small numbers which are then divided by small numbers (h or h^2).

The issue is best illustrated by an example.

5.2.1 Example

Consider the central difference approximation to the derivative. I.e.

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} - \frac{h^2}{6} f'''(\xi),$$

for some $\xi \in (x_0 - h, x_0 + h)$. The final term is what we have called our truncation error, $E(h)$.

Consider now the effect of making small errors in the calculation of the approximation. That is, assume that the exact values are given by

$$f(x_0+h) = \tilde{f}(x_0+h) + e(x_0+h), \qquad f(x_0-h) = \tilde{f}(x_0-h) + e(x_0-h),$$

finite-precision calculations ($\hat{f}(x_0 \pm h)$) and round-off errors ($e(x_0 \pm h)$). These round-off errors are bounded by machine accuracy (see definition in Chapter 1):

$$|e(x_0 \pm h)| < \epsilon |f(x_0 \pm h)|$$

(and $\epsilon \approx 10^{-16}$ for devices with 64-bit storage).

The **total error** $E_t(h)$ is the sum of the **round-off errors** and the **truncation error**

$$E_t(h) = \frac{e(x_0+h) - e(x_0-h)}{2h} - \frac{h^2}{6} f'''(\xi).$$

Assuming that h is small enough for $f(x_0 \pm h) \approx f(x_0)$ and for $f'''(\xi) \approx f'''(x_0)$, standard inequalities give us

$$|E_t(h)| \leq \left| \frac{e(x_0+h)}{2h} \right| + \left| \frac{e(x_0-h)}{2h} \right| + \left| \frac{h^2}{6} f'''(\xi) \right| \lesssim \frac{\epsilon |f(x_0)|}{h} + \frac{h^2}{6} |f'''(x_0)|.$$

Miminising the total error as a function of h requires
(22)

$$0 = \frac{\mathrm{d}}{\mathrm{d}h} \left(\frac{\epsilon f(x_0)}{h} + \frac{h^2}{6} f'''(x_0) \right) = -\frac{\epsilon |f(x_0)|}{h^2} + \frac{h|f'''(x_0)|}{3}, \qquad \Longrightarrow \qquad h = h_{\text{opt}} = \left(\frac{3\epsilon |f(x_0)|}{|f'''(x_0)|} \right)^{1/3}.$$

This gives an estimate for the optimal choice of h which minimises the total error.

Remark: This result is not universal: it depends on the scheme being analysed.

h	E_t
0.01	-0.84×10^{-2}
0.001	-0.84×10^{-4}
0.0001	-0.84×10^{-6}
10^{-5}	-0.54×10^{-8}
10^{-6}	0.74×10^{-10}
10^{-7}	-0.25×10^{-9}
10^{-8}	0.21×10^{-7}

Numerical experiment: Use central difference formula to approximate the derivative of $f(x) = x^9$ at $x = 1$ with $h = 10^{-m}$, $m = 2, 3, \ldots, 8$ and deduce the error (the exact result is $f'(1) = 9$ of course).

We performed this calculation on a PC with 64-bit storage: so let's say $\epsilon = 10^{-16}$. We see that the error initially decreases with h, proportional to $O(h^2)$ as predicted by the truncation error. However, for values smaller than $h \approx 10^{-6}$ we notice that the errors start increasing. We now understand this to be the effect of round-off errors dominating the truncation error.

According to (22), we have $|f(1)| = 1$, $|f'''(1)| = 9 \times 8 \times 7 = 504$ and so

$$h_{opt} \approx \left(\frac{3 \times 10^{-16}}{504} \right)^{1/3} \approx 0.84 \times 10^{-6}$$

which agrees with the tabulated results.

Remark: The mimimum error of 10^{-10} in the table of approximations sounds OK, but remember that we started with calculations accurate to 10^{-16} and so we've lost 5 decimal places accuracy in just one calculation. And it's even worse for second derivatives.

Q: What can be done to to reduce the error?

A: Higher-order formulae (e.g. using more points). Or...

5.3 Richardson extrapolation

Richardson extrapolation[13] is a general method that is useful across many areas of Numerical Analysis. It can be used whenever we know *how* the error of an approximation depends on a parameter h of the approximation (usually h is a step size).

5.3.1 Illustration of the method

Assume that an exact quantity N is approximated by an expression $N_1(h)$ and it is known that the error has a power series expansion in h

$$N = N_1(h) + a_1 h + a_2 h^2 + a_3 h^3 + \dots \tag{23}$$

The coefficients a_i, $i = 1, 2, \dots$ are constants that may not be known analytically or even numerically.

We now halve the step size to obtain another approximation for N:

$$N = N_1\left(\frac{h}{2}\right) + a_1 \frac{h}{2} + a_2 \frac{h^2}{4} + a_3 \frac{h^3}{8} + \dots \tag{24}$$

Now we notice that the combination 2×(24)−(23) eliminates the leading order $O(h)$ term in the error, thus:

$$2N - N = \left[2N_1\left(\frac{h}{2}\right) - N_1(h)\right] - \frac{a_2}{2}h^2 - \frac{3a_3}{4}h^3 + \dots$$

In other words we have

$$N = N_2(h) + b_2 h^2 + b_3 h^3 + \dots \tag{25}$$

where

$$N_2(h) = 2N_1(h/2) - N_1(h) \tag{26}$$

[13] After Lewis Fry Richardson (1911) "The approximate arithmetical solution by finite differences of physical problems including differential equations, with an application to the stresses in a masonry dam" , though he was better known as a pioneer in the development of modern methods for weather forecasting.

and $b_2 = -a_2/2$, etc (since the values of a_i were unknown and unimportant, then redefining them as b_i's is OK).

I.e. we have obtained a new approximation, $N_2(h)$, to N which is $O(h^2)$ accurate by using calculations of N_1 accurate to $O(h)$. It's like magic.

This process can be repeated. Consider halving h again in equation (25):

$$N = N_2\left(\frac{h}{2}\right) + b_2\frac{h^2}{4} + b_3\frac{h^3}{8} + \dots \tag{27}$$

Now the linear combination 4×(27)−(25) eliminates the $O(h^2)$ term thus:

$$4N - N = 4N_2\left(\frac{h}{2}\right) - N_2(h) - \frac{1}{2}b_3h^3 + \dots$$

In other words we have a new approximation

$$N = N_3(h) - \frac{1}{6}b_3h^3 + \dots$$

where

$$N_3(h) = \frac{1}{3}\left[4N_2\left(\frac{h}{2}\right) - N_2(h)\right] \tag{28}$$

is accurate to $O(h^3)$. Using (26) in (28) results in

$$N_3(h) = \frac{1}{3}\left[8N_1\left(\frac{h}{4}\right) - 6N_1\left(\frac{h}{2}\right) + N_1(h)\right]$$

in terms of the original approximation scheme N_1. The advantage of this method is that one can get much higher accuracy with relatively large values of h and therefore avoid the onset of round-off errors.

Remark: Instead of halving h at each step we can also double it as we will see in later examples.

5.3.2 Example

Consider the central difference approximation for the derivative $f'(x_0)$

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} - \frac{h^2}{6} f'''(\xi).$$

Note: the value of $\xi \in (x_0 - h, x_0 + h)$ in this formula depends on h, so this formula is not a complete expansion in powers of h.

We can, however, obtain a complete expansion in h by returning to (21) and, instead of truncating the expansion of $f(x_0 \pm h)$, produce a full a Taylor series expansion. This results (check) in

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} - \sum_{k=1}^{\infty} \frac{h^{2k}}{(2k+1)!} f^{(2k+1)}(x_0).$$

Notice the sum is over even powers h^2, h^4 etc and so this relation may be expressed as

$$N = N_1(h) + a_2 h^2 + a_4 h^4 + a_6 h^6 + \ldots \tag{29}$$

since we are attempting to approximate $N = f'(x_0)$ with the scheme $N_1(h) = (f(x_0+h) - f(x_0 - h))/(2h)$ and we have no idea what all the higher derivatives of f are, hence the coefficients a_i are unknown.

We halve the step size to obtain another approximation for N

$$N = N_1\left(\frac{h}{2}\right) + a_2\left(\frac{h}{2}\right)^2 + a_4\left(\frac{h}{2}\right)^4 + a_6\left(\frac{h}{2}\right)^6 + \ldots \tag{30}$$

The combination (4×(30)−(29))/3 gives

$$N = N_2(h) + \frac{a_4}{3}\left(-\frac{3}{4}h^4\right) + \frac{a_6}{3}\left(-\frac{15}{16}h^3\right) + \ldots \tag{31}$$

where

$$N_2(h) = \frac{1}{3}\left[4N_1\left(\frac{h}{2}\right) - N_1(h)\right]$$

is now accurate to $O(h^4)$.

Exercise: repeat the process and halve h again in (31) to get $N = N_3(h) + O(h^6)$ where

$$N_3(h) \equiv \frac{1}{15}\left[16N_2\left(\frac{h}{2}\right) - N_2(h)\right].$$

and hence in terms of N_1 only

$$N_3(h) = \frac{1}{45}\left[64N_1\left(\frac{h}{4}\right) - 20N_1\left(\frac{h}{2}\right) + N_1(h)\right].$$

	E_t		
h	$f'(1) - N_1(h)$	$f'(1) - N_2(h)$	$f'(1) - N_3(h)$
0.01	-8.4×10^{-3}	3.1×10^{-7}	-3.9×10^{-13}
0.005	-2.1×10^{-3}	1.9×10^{-8}	
0.0025	-5.2×10^{-4}		

E.g.: Consider previous example $f(x) = x^9$ with $x_0 = 1$ and $h = 0.01$. We calculate $N_1(0.01)$, $N_1(0.01/2)$ and $N_1(0.01/4)$ and this allows us to compute the total error (recall, the best we could do without extrapolation was $E_t = 10^{-10}$ with $h_{opt} = 10^{-6}$; now we get E_t under 10^{-12} with a smallest value of $h = 0.0025$.

6 Integration

Many integrals cannot be evaluated by hand and require numerical methods. Numerical integration methods are often referred to as **quadrature**. They involve evaluating continuous functions at discrete points x_i, say, and weighting their contribution by c_i, say:

$$I = \int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} c_i f(x_i).$$

We start with methods that replace the integrand with a **Lagrange interpolating polynomial** which can be integrated explicitly.

6.1 The trapezoidal rule

Our simplest approximation to the integral I uses the linear Lagrange polynomial with $x_0 = a$, $x_1 = b$:

$$P_1(x) = \frac{(x-b)}{(a-b)} f(a) + \frac{(x-a)}{(b-a)} f(b).$$

Recall that

$$f(x) = P_1(x) + \frac{f''(\xi(x))}{2}(x-a)(x-b)$$

for some $\xi \in (a,b)$ includes the exact error term.

Thus we obtain

$$I = \int_a^b f(x)\,\mathrm{d}x = \int_a^b P_1(x)\,\mathrm{d}x + \frac{1}{2}\int_a^b f''(\xi(x))\,(x-a)\,(x-b)\,\mathrm{d}x.$$

The integral over the polynomial can be easily evaluated and results in

$$\int_a^b P_1(x)\,\mathrm{d}x = \left[\frac{1}{2}\frac{(x-b)^2}{(a-b)} f(a) + \frac{1}{2}\frac{(x-a)^2}{(b-a)} f(b)\right]_a^b = \frac{b-a}{2} f(a) + \frac{b-a}{2} f(b).$$

For the integration of the error term we need...

Theorem: The **weighted mean-value theorem for integrals**. Suppose $f \in C[a,b]$, g is integrable on $[a,b]$, and $g(x)$ does not change sign on $[a,b]$. Then there exists a number $c \in (a,b)$ such that

$$\int_a^b f(x)\, g(x)\, \mathrm{d}x = f(c) \int_a^b g(x)\, \mathrm{d}x.$$

Proof: Omitted (but a useful case is to consider $g(x) = 1$ where we can see how the result works graphically).

... Now assume that $f \in C^2[a,b]$ and define $g(x) = (x-a)(x-b)$, noting that $g(x)$ does not change sign in $[a,b]$. Then there exists a number $\xi \in (a,b)$ such that

$$\begin{aligned}
\frac{1}{2}\int_a^b f''(\xi(x))\,(x-a)\,(x-b)\,\mathrm{d}x &= \frac{1}{2}f''(\xi)\int_a^b (x-a)\,(x-b)\,\mathrm{d}x \\
&= \frac{1}{2}f''(\xi)(b-a)^3\int_0^1 t(t-1)\,\mathrm{d}t \\
&= -\frac{1}{12}(b-a)^3 f''(\xi)
\end{aligned}$$

(1st step is a change of variable $x - a = (b-a)t$). Let us express the result with $a = x_0$, $b = x_1$ and $h = x_1 - x_0$, where $x_1 > x_0$,

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = \frac{h}{2}\left[f(x_0) + f(x_1)\right] - \frac{h^3}{12}f''(\xi)$$

where $\xi \in (x_0, x_1)$. The name of the method arises from the fact that the area under the function $f(x)$ is estimated by a trapezium (the first term in the above, the second term is the error).

6.1.1 Composite trapezoidal rule

The trapezoidal rule does not, in general, give a good approximation unless h is small. The **composite trapezoidal rule** divides larger intervals into smaller subintervals to which the trapezium rule is applied.

Consider an interval $[a,b]$ that is divided into n equal subintervals of length $h = (b-a)/n$ and denote the borders of the subintervals by $x_i = a + ih$, $i = 0, 1, \ldots, n$.

Applying the trapezoidal rule to each subinterval gives

$$I = \int_a^b f(x)\,\mathrm{d}x = \sum_{i=1}^{n} \left[\frac{h}{2}\left(f(x_{i-1}) + f(x_i)\right) - \frac{h^3}{12} f''(\xi_i) \right]$$

where $x_{i-1} < \xi_i < x_i$. We can use the intermediate value theorem to simplify the sum over the error terms:

$$\frac{1}{n}\sum_{i=1}^{n} f''(\xi_i) = f''(\xi), \qquad n = \frac{(b-a)}{h}$$

for some $\xi \in (a,b)$. For the application of the intermediate value theorem we used the fact that the average of the error terms must lie between the minimum and the maximum of the error terms. Thus the final result can be written as

$$I = \int_a^b f(x)\,\mathrm{d}x = T_n + E_n$$

where

$$T_n = \frac{h}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + \ldots + 2f(x_{n-1}) + f(x_n)\right]$$

is the **composite trapezoidal rule** and

$$E_n = -\frac{h^2}{12}(b-a)\,f''(\xi)$$

where $\xi \in (a,b)$ is the **global truncation error**.

Remark 1: Although the local truncation error for each subinterval is $-h^3 f''(\xi_i)/12$, the global truncation error is $E_n = -h^2(b-a)f''(\xi)/12$ since it accumulates error from $n = (b-a)/h$ contributions.

Remark 2: In the global error the term ξ is a function of h. One can derive a complete expansion of the error term in powers of h similar to Section 5.3:

$$E_n = a_2 h^2 + a_4 h^4 + a_6 h^6 + \ldots$$

6.2 Simpson's rule

The trapezoidal rule can be improved upon by using a quadratic Lagrange polynomial to approximate $f(x)$, i.e.

$$f(x) = P_2(x) + \frac{1}{3!} f^{(3)}(\xi(x))\,(x - x_0)\,(x - x_1)\,(x - x_2).$$

where

$$P_2(x) = \frac{(x-x_1)\,(x-x_2)}{(x_0-x_1)\,(x_0-x_2)} f(x_0) + \frac{(x-x_0)\,(x-x_2)}{(x_1-x_0)\,(x_1-x_2)} f(x_1) + \frac{(x-x_0)\,(x-x_1)}{(x_2-x_0)\,(x_2-x_1)} f(x_2).$$

We consider now the interval $[a, b]$ and define $x_0 = a$, $x_2 = b$, and $x_1 = (x_0+x_2)/2$ (which divides the interval into two equal parts) to specify the Lagrange polynomial. Furthermore the step size is now $h = (b - a)/2$.

One can then prove the following

$$\int_{x_0}^{x_2} f(x)\,\mathrm{d}x = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{h^5}{90} f^{(iv)}(\xi)$$

where $\xi \in (x_0, x_2)$. This is **Simpson's rule**[14].

Remark 1: The first term on the right-hand side is obtained by integrating over the Lagrange polynomial $P_2(x)$; this is not difficult but lengthy. The derivation of the error term, on the other hand, is trickier. One expects the error term of order $O(h^4)$ whereas it turns out, due to certain cancellations, to be $O(h^5)$.

Remark 2: The error term is proportional to $f^{(iv)}(x)$. This means that Simpson's rule is exact for polynomials of degree 3 or less. In other words, even though we have approximated $f(x)$ using a quadratic interpolating polynomial, Simpson's rule exactly integrates all cubics polynomials !

Remark 3: One can continue using higher degree polynomials to approximate integrals: this results in the **Newton-Cotes formulas**[15]. This process is, however, problematic due to **Runge's**

[14] named after Thomas Simpson (1710-1761)

[15] named after Isaac Newton and Roger Cotes

phenomenon[16]. It occurs if one uses high-degree polynomials to interpolate a function at **equidistant points** since it can be shown to lead to oscillations at the edge of the interval similar to Gibb's phenomenon[17] when using Fourier series.

In conclusion: increasing the degree does not always improve the accuracy. It is better to use the trapezoidal rule or Simpson's rule over subdivisions of intervals.

6.2.1 Composite Simpson's rule

Similar to the composite Trapezium rule, we divide the integration range $[a, b]$ into smaller intervals and apply Simpson's rule to the subintervals. Note that we need an even number of subintervals, because Simpson's rule involves pairs of subintervals.

Divide the interval $[a, b]$ into n (even !) subintervals and set

$$h = \frac{b-a}{n}, \qquad x_i = a + ih, \qquad f_i \equiv f(x_i), \qquad i = 0, 1, \ldots, n.$$

Applying Simpson's rule to each pair of subintervals results in

$$\begin{aligned} I = \int_a^b f(x)\,\mathrm{d}x &= \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{h^5}{90} f^{(iv)}(\xi_1) \\ &\quad + \frac{h}{3}\left[f(x_2) + 4f(x_3) + f(x_4)\right] - \frac{h^5}{90} f^{(iv)}(\xi_2) \\ &\qquad \vdots \qquad\qquad \vdots \\ &\quad + \frac{h}{3}\left[f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right] - \frac{h^5}{90} f^{(iv)}(\xi_m) \end{aligned}$$

where $n = 2m$. The error term can again be simplified by using the IVT:

$$\frac{1}{m}\sum_{i=1}^{m} f^{(iv)}(\xi_i) = f^{(iv)}(\xi), \qquad \text{where } \xi \in (a, b).$$

Thus, the **composite Simpson rule** is

$$I = \int_a^b f(x)\,\mathrm{d}x = S_n + E_n$$

[16] discovered by Carl Runge in 1901

[17] attributed to J. Willard Gibbs (1899) though originally discovered by Henry Wilbraham (1848)

where

$$S_n = \frac{h}{3}\left[f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \ldots + 2f_{n-2} + 4f_{n-1} + f_n\right]$$

with $h = (b-a)/n$, n even, and

$$E_n = -\frac{h^4\,(b-a)}{180} f^{(iv)}(\xi)$$

where $\xi \in (a,b)$ is the **global truncation error** and is one order less than the local truncation error for the same reasons as before.

6.3 Romberg integration

Romberg integration[18] is simply Richardson's extrapolation applied to the composite trapezoidal rule.

At the end of Section 6.1.1 we asserted (without proof) that

$$I = \int_a^b f(x)\,\mathrm{d}x = T_n + a_2h^2 + a_4h^4 + a_6h^6 + \ldots \tag{32}$$

where T_n is given by the composite trapezoidal rule. That is, the error is an even power series in the step size h but the values of a_{2k} are unknown and unimportant.

IDEA: We decide that $n = 2^m$ for m sufficiently large (see Example 6.3.1), and instead of halving h we decide to double h (and so halve n) at each Romberg step. In this way, we do not need to make additional functional evaluations.

For the first Romberg iteration we do $h \to 2h$ (or $n \to n/2$) to give

$$I = T_{n/2} + a_2(2h)^2 + a_4(2h)^4 + a_6(2h)^6 + \ldots \tag{33}$$

so that $4\times(32) - (33)$ gives

$$I = \underbrace{\frac{4T_n - T_{n/2}}{3}}_{T_n^{(1)}} + b_4h^4 + b_6h^6 + \ldots \tag{34}$$

[18] Romberg, W. (1955), "Vereinfachte numerische Integration" (vereinfachte translates to simplified)

for some b_{2k} whose values are unimportant. Now $T_n^{(1)}$ is the first Romberg iterate and is accurate to $O(h^4)$.

Remark: Written out explicitly

$$\begin{aligned} T_n^{(1)} = \frac{1}{3}\left(4T_n - T_{n/2}\right) &= \frac{h}{3}\left[2f_0 + 4f_1 + 4f_2 + 4f_3 + 4f_4 + \ldots + 4f_{n-1} + 2f_n\right] \\ &\quad -\frac{2h}{6}\left[f_0 + 2f_2 + 2f_4 + \ldots + 2f_{n-2} + f_n\right] \\ &= S_n \end{aligned}$$

is exactly Simpson's Rule !

The process can be repeated (for as long as n can be sucessively halved). E.g. letting $h \to 2h$ and $n \to n/2$ again gives

$$I = T_{n/2}^{(1)} + b_4(2h)^4 + b_6(2h)^6 + \ldots \tag{35}$$

so that $16\times(34) - (35)$ gives

$$I = \underbrace{\frac{16T_n^{(1)} - T_{n/2}^{(1)}}{15}}_{T_n^{(2)}} + c_6h^6 + \ldots$$

and the second Romberg iterate $T_n^{(2)}$ is accurate to $O(h^6)$. Etc.

Remark: One can show that the second iteration of the Romberg integration also corresponds to a Newton-Cotes formula (i.e. derived from Lagrange interpolation with higher-degree polynomials), but higher ones don't. Importantly, Romberg iterates are more numerically stable than the Newton-Cotes formulas.

6.3.1 Example

Evaluate the integral $I = \int_0^1 x^7\, \mathrm{d}x = 0.125$.

We let $a = 0$, $b = 1$, $f(x) = x^7$ and write $f_i = f(x_i)$ where $x_i = ih$ where $h = 1/n$ and we choose $n = 1, 2, 4, 8, \ldots$. We use the following numerical methods:

(i) $T_n = \frac{1}{2}h[f_0 + 2f_1 + 2f_2 + 2f_3 + 2f_4 + \ldots + 2f_{n-1} + f_n]$ (composite Trapezoidal);

(ii) $S_n = \frac{1}{3}h[f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \ldots + 4f_{n-1} + f_n]$ (composite Simpson).

	Trapezoidal		Simpson	
n	T_n	E_n	S_n	E_n
1	0.5000000000	0.3750000000		
2	0.2539062500	0.1289062500	0.1718750000	0.0468750000
4	0.1603393555	0.0353393555	0.1291503906	0.0041503906
8	0.1340436935	0.0090436935	0.1252784729	0.0002784729
16	0.1272742003	0.0022742003	0.1250177026	0.0000177026
32	0.1255693834	0.0005693834	0.1250011111	0.0000011111
64	0.1251423980	0.0001423980	0.1250000695	0.0000000695
128	0.1250356028	0.0000356028	0.1250000043	0.0000000043

Remark: We see that trapezoidal error drops by a factor of 4 as the value of n is doubled, consistent with $O(h^2)$ error. The Simpson error drops by a factor of 16 on every doubling of n, consistent with $O(h^4)$ error.

We now use the values of T_n in the table above to compute successive Romberg iterates:

(iii) $T_n^{(1)} = (4T_n - T_{n/2})/3$ for $n \geq 2$ (and note that these are S_n);
$T_n^{(2)} = (16T_n^{(1)} - T_{n/2}^{(1)})/15$ for $n \geq 4$;
$T_n^{(3)} = (64T_n^{(2)} - T_{n/2}^{(2)})/63$ for $n \geq 8$.

6.4 Problems in the evaluation of integrals

In many cases quadrature methods yield very good results. However, we may encounter situations in which there are problems implementing methods which include the following:

- $f(x)$ is discontinuous;

n	T_n	$T_n^{(1)}$	$T_n^{(2)}$	$T_n^{(3)}$
1	0.5000000000			
2	0.2539062500	0.1718750000		
4	0.1603393555	0.1291503906	0.1263020833	
8	0.1340436935	0.1252784729	0.1250203451	0.1250000000

- $f(x)$ has discontinuous or singular derivatives;
- $f(x)$ has integrable singularities;
- $f(x)$ is highly oscillatory;
- range of integration is infinite.

One normally has to devise a bespoke techniques to overcome the particular difficulty. These often involve changing variables, dividing integrals into smaller parts, and explicitly removing singularities. We consider some examples.

6.4.1 Example 1

Consider

$$I = \int_0^1 \frac{\mathrm{e}^{-x}}{\sqrt{x}}\,\mathrm{d}x.$$

Here the singularity can be removed with a suitable transformation of variables: $x = t^2$, $\mathrm{d}x = 2t\mathrm{d}t$ gives

$$I = \int_0^1 2\mathrm{e}^{-t^2}\,\mathrm{d}t.$$

6.4.2 Example 2

$$I = \int_0^\infty \frac{\ln(x)}{1+x^4}\,\mathrm{d}x.$$

We divide the two problems up and write

$$I = \int_0^1 \frac{\ln(x)}{1+x^4} - \ln(x)\,\mathrm{d}x + \int_0^1 \ln(x)\,\mathrm{d}x + \int_1^\infty \frac{\ln(x)}{1+x^4}\,\mathrm{d}x.$$

We can do the middle integral by hand (it equates to $[x\ln(x) - x]_0^1 = -1$). For the last integral we make the change of variables $x = 1/t$ to get

$$I = -\int_0^1 \frac{x^4\ln(x)}{1+x^4}\,\mathrm{d}x - 1 + \int_1^0 \frac{\log(1/t)}{1+(1/t^4)}\left(\frac{-1}{t^2}\right)\,\mathrm{d}t$$

and the last integral simplifies to

$$-\int_0^1 \frac{t^2\ln(t)}{1+t^4}\,\mathrm{d}t.$$

Putting everything together gives

$$I = -1 - \int_0^1 \frac{(x^2+x^4)\ln(x)}{1+x^4}\,\mathrm{d}x$$

and the integrand is smooth and bounded and can be approximated accurately using quadrature.

6.5 Weighted integrals

Another angle of attack is to factorise the awkward component of an integrand, $w(x)$, say, from the well-behaved part, $f(x)$, say, by writing

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

For example, we will later consider $a = -1$, $b = 1$, $w(x) = 1/\sqrt{1-x^2}$.

In order to move onto this next topic – Gaussian quadrature – we first need to develop the theory of orthogonal polynomials, which will also be of use at the end of the course.

6.6 Orthogonal polynomials

Defn: An integrable function $w(x)$ is called a **weight function** on an interval $x \in (a,b)$ if $w(x) \geq 0$ for $x \in (a,b)$ but $w(x) \not\equiv 0$ on any subinterval of (a,b). Thus $w(x)$ can vanish at most at finitely many isolated points in (a,b).

Defn: A set of functions $\{\phi_i,\ i = 0,\dots,n\}$ is said to be **orthogonal** on an interval $[a,b]$ with respect to a continuous weight function $w(x)$ provided that the **inner product of ϕ_i and ϕ_j**,

$$\langle \phi_i, \phi_j \rangle \equiv \int_a^b w(x)\phi_i(x)\phi_j(x)\,\mathrm{d}x = \alpha_i\delta_{ij} = \begin{cases} 0, & \text{when } i \neq j, \\ \alpha_i, & \text{when } i = j. \end{cases} \tag{36}$$

Note that

$$\alpha_i = \langle \phi_i, \phi_i \rangle = \int_a^b w(x)\phi_i^2(x)\,\mathrm{d}x > 0$$

from the definition of $w(x)$.

Defn: If, additionally, $\alpha_i = 1$ for each $i = 0,1,\dots,n$ the set is said to be **orthonormal**.

Remark: In everything that follows we establish properties that hold when $\{\phi_i(x),\ i = 0,\dots,n\}$ is a set of orthogonal polynomials *such that ϕ_i is of degree i*.

6.6.1 Properties of orthogonal polynomials

Property 1: Any polynomial of degree $k \leq n$ can be written as

$$P_k(x) = \sum_{i=0}^{k} a_i\phi_i(x). \tag{37}$$

Proof: Choose a_k such that the coefficient of x^k on both sides of the equation are the same. It follows that $P_k(x) - a_k\phi_k(x)$ is a polynomial of degree $(k-1)$.

$$P_k(x) - a_k\phi_k(x) = \sum_{i=0}^{k-1} a_i\phi_i(x).$$

Now repeat the procedure to establish the result.

Remark: The coefficients can be calculated directly by using (36). Consider multiplying both sides of (37) by $w(x)\phi_j(x)$ and integrating over $a < x < b$. Then

$$\langle P_k, \phi_j \rangle = \sum_{i=0}^{k} a_i \langle \phi_i, \phi_j \rangle = \sum_{i=0}^{k} a_i \alpha_i \delta_{ij} = a_j \alpha_j .$$

It follows that

$$a_j = \frac{1}{\alpha_j} \int_a^b w(x) P_k(x) \phi_j(x) \, \mathrm{d}x .$$

Property 2: $\phi_k(x)$ is orthogonal to any polynomial $P_l(x)$ of lower degree, $l < k$.

Proof:

$$\langle P_l, \phi_k \rangle = \sum_{i=0}^{l} a_i \langle \phi_i, \phi_k \rangle = 0$$

because $i \leq l < k$.

Property 3: The polynomial $\phi_k(x)$ has k distinct zeros in (a, b).

Proof: Let $x = x_i$, $i = 1, \ldots, l$ denote all zeros of $\phi_k(x)$ of *odd multiplicity* in (a, b). The function

$$S(x) = \prod_{i=1}^{l} (x - x_i)$$

is a polynomial of degree $l \leq k$ since $\phi_k(x)$ is a polynomial of degree k and therefore cannot have more than k zeros.

The function $S(x)$ changes sign at the same positions as $\phi_k(x)$. This implies that the product $S(x)\phi_k(x)$ never changes sign on $a < x < b$ and consequently

$$\langle S, \phi_k \rangle \equiv \int_a^b w(x) S(x) \phi_k(x) \, \mathrm{d}x \neq 0 .$$

From property 2 above it must be that $S(x)$ is a polynomial of degree k and so $l = k$. This conclusion also implies that x_i are distinct.

6.6.2 Constructing a sequence of orthogonal polynomials

How do we construct ϕ_i for $i = 0, \dots, n$ for a given interval (a, b) and weight function $w(x)$?

That is, we want $\phi_i(x)$ to be polynomial of degree i such that $\langle \phi_i, \phi_j \rangle = 0$ for $0 \leq i, j, \leq n$.

Defn: The requirements above only specify ϕ_i to within a multiplicative constant. To ensure uniqueness one could make the set $\{\phi_i(x), i = 0, \dots, n\}$ orthonormal, but instead we often apply a **standardisation condition**. This is often, but not always, $\phi_i(1) = 1$ for all $i = 0, \dots, n$.

A manual approach

- First, $\phi_0(x) = 1$: this is a polynomial of degree 0 s.t. $\phi_0(1) = 1$.
- Next, $\phi_1(x) = A_1 x + B_1$ is a polynomial of degree 1. We require $\langle \phi_1, \phi_0 \rangle = 0$ and this means
$$A_1 \langle x, \phi_0 \rangle + B_1 \langle 1, \phi_0 \rangle = 0$$
such that
$$\phi_1(x) = A_1 \left(x - \frac{\langle x, 1 \rangle}{\langle 1, 1 \rangle} \right).$$
Imposing the standardisation $\phi_1(1) = 1$ determines A_1.
- Next, $\phi_2(x) = A_2 x^2 + B_2 x + C_2$ is a polynomial of degree 2. Now we require
$$0 = \langle \phi_2, \phi_0 \rangle = A_2 \langle x^2, 1 \rangle + B_2 \langle x, 1 \rangle + C_2 \langle 1, 1 \rangle$$
and
$$0 = \langle \phi_2, \phi_1 \rangle = A_2 \langle x^2, \phi_1 \rangle + B_2 \langle x, \phi_1 \rangle + C_2 \langle 1, \phi_1 \rangle$$
and, from standardisation used here $A_2 + B_2 + C_2 = 1$. That's 3 equations in 3 unknowns that can be solved for.

We can continue like this, but it gets complicated after the few terms.

6.6.3 The Gram-Schmidt process

But this can help...

Proposition: for $k \geq 0$

$$\phi_{k+1}(x) = A_{k+1}\left(x\phi_k(x) - \sum_{i=0}^{k} \frac{\langle x\phi_k, \phi_i\rangle}{\langle \phi_i, \phi_i\rangle}\phi_i(x)\right). \tag{38}$$

Proof: (Inductive) First, assuming $\phi_k(x)$ is a polynomial of degree k then $\phi_{k+1}(x)$ is a polynomial of degree $k+1$. Assuming the set $\{\phi_i(x),\ i = 0, \ldots, k\}$ is orthogonal, we have from (38) that

$$\langle \phi_{k+1}, \phi_j\rangle = A_{k+1}\left(\langle x\phi_k, \phi_j\rangle - \sum_{i=0}^{k} \frac{\langle x\phi_k, \phi_i\rangle}{\langle \phi_i, \phi_i\rangle}\langle \phi_i, \phi_j\rangle\right) = A_{k+1}\left(\langle x\phi_k, \phi_j\rangle - \frac{\langle x\phi_k, \phi_j\rangle}{\langle \phi_j, \phi_j\rangle}\langle \phi_j, \phi_j\rangle\right) = 0$$

by (36). Hence ϕ_{k+1} is orthogonal to all other $\phi_i(x)$, $i = 0, \ldots, k$.

We can simplify (38) by noting that

$$\langle x\phi_k, \phi_i\rangle = \int_a^b w(x) x\phi_k(x)\phi_i(x)\, \mathrm{d}x = \langle \phi_k, x\phi_i\rangle = 0, \qquad \text{if } i < k-1$$

since $x\phi_i$ is a polynomial of degree $i+1$ and using the earlier Property 2.

So, in fact, only the last two terms in the sum in (38) are non-zero and (38) can be reduced to a **3-term recurrence relation**

$$\phi_{k+1}(x) = A_{k+1}\left(x\phi_k(x) - \frac{\langle x\phi_k, \phi_k\rangle}{\langle \phi_k, \phi_k\rangle}\phi_k(x) - \frac{\langle x\phi_k, \phi_{k-1}\rangle}{\langle \phi_{k-1}, \phi_{k-1}\rangle}\phi_{k-1}(x)\right) \tag{39}$$

for $k \geq 1$ with

$$\phi_1(x) = A_1\left(x - \frac{\langle x, 1\rangle}{\langle 1, 1\rangle}\right)$$

and $\phi_0(x) = A_0$ such that A_k's satisfy the standardisation condition.

This is the **Gram-Schmidt process** for determining orthogonal polynomials.

6.6.4 Example: Legendre Polynomials

Choose $a = -1$, $b = 1$ and $w(x) = 1$. Also let $\phi_i \equiv P_i$ (standard notation in the literature for this choice) and use standardisation $P_n(1) = 1$.

- First $P_0(x) = 1$.
- Then
$$P_1(x) = A_1 \left(x - \frac{\int_{-1}^{1} x.1 \, \mathrm{d}x}{\int_{-1}^{1} 1.1 \, \mathrm{d}x} \right) = A_1 x$$
so $1 = P_1(x) = A_1.1$ means $A_1 = 1$ and $P_1(x) = x$.
- Next
$$P_2(x) = A_2 \left(x.x - \frac{\int_{-1}^{1} x.x.x \, \mathrm{d}x}{\int_{-1}^{1} x.x \, \mathrm{d}x} x - \frac{\int_{-1}^{1} x.x.1 \, \mathrm{d}x}{\int_{-1}^{1} 1.1 \, \mathrm{d}x} \right) = A_2(x^2 - (2/3)/2).$$
Applying $1 = P_2(x) = A_2(1 - (1/3))$ gives $A_2 = 3/2$ so
$$P_2(x) = \frac{3}{2}x^2 - \frac{1}{2}.$$
- And we can go on (it gets complicated). For e.g. $P_3(x) = \frac{5}{2}x^3 - \frac{3}{2}x$.

Remark: The set $\{P_k(x)\}$ are called the **Legendre polynomials**. They satisfy a 3-term recurrence relation
$$(k+1)P_{k+1}(x) = (2k+1)xP_k(x) - kP_{k-1}(x)$$
(no proof) but consistent with (39).

6.7 Gaussian quadrature

6.7.1 Introduction

Up to now the integral of a function has been approximated by a weighted sum of evaluations of the integrand at prescribed equally-spaced points. What about optimising the points at which the integrand is evaluated ?

Example: Consider the approximation

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx c_1 f(x_1).$$

Q: Can we choose c_1 and x_1 to ensure that the approximation is exact for any linear polynomial ?

A: Let $f(x) = a_0 + a_1 x$, then the LHS is $2a_0$ and the RHS is $c_1 a_0 + c_1 a_1 x_1$. If we want LHS = RHS for arbitrary a_0 and a_1 then we must set $c_1 = 2$ and $x_1 = 0$. I.e.

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

is exact for all linear polynomials $f(x)$.

Exercise: Consider the approximation

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx c_1 f(x_1) + c_2 f(x_2).$$

We have 4 parameters c_1, c_2, x_1 and x_2 to choose. Show that these can be defined to exactly integrate all cubic polynomials: $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$, with a_i arbitrary.

6.7.2 The main result

Theorem: Let $\{\phi_i, i = 0, \ldots, n\}$ be a set of orthogonal polynomials with respect to the weight function $w(x)$ on an interval (a, b) ($\phi_i(x)$ is of degree i). Then the integration formula

$$\int_a^b w(x) f(x)\,\mathrm{d}x \approx \sum_{j=1}^{n} w_j f(x_j), \tag{40}$$

where

$$w_j = \int_a^b w(x) \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)} \mathrm{d}x \tag{41}$$

and x_i, $i = 1, \ldots, n$ are the n zeros of $\phi_n(x)$, is **exact** if $f(x)$ is a polynomial of degree $(2n-1)$ or less.

Note: If $n = 1$ then $w_1 = \int_a^b w(x)\,\mathrm{d}x$.

Proof: Start by assuming that $f(x)$ is a polynomial of degree $(2n-1)$ or less. Then we can write

$$f(x) = q(x)\phi_n(x) + r(x) \tag{42}$$

where $q(x)$ and $r(x)$ are both polynomials of degree $(n-1)$ or less. The coefficients of the powers of $x^{n-1}, \ldots, x^0$ in $q(x)$ are chosen such that the coefficients of the powers of $x^{2n-1}, \ldots, x^n$ in the product $q(x)\phi_n(x)$ match the coefficients of the same powers of $f(x)$ and then the coefficients of the powers of $x^{n-1}, \ldots, x^0$ in $r(x)$ sort out the remaining powers of $x^{n-1}, \ldots, x^0$ of $f(x)$.

It follows that

$$\int_a^b w(x)f(x)\,\mathrm{d}x = \int_a^b w(x)q(x)\phi_n(x)\,\mathrm{d}x + \int_a^b w(x)r(x)\,\mathrm{d}x.$$

The first integral on the right-hand side vanishes by Property 2 of orthogonal polynomials.

Since $r(x)$ is a polynomial of degree $n-1$ we can exactly represent it by an interpolating Lagrange polynomial of degree $n-1$ that passes through the points $x_1, \ldots, x_n$ (noting that we've shifted indices from 0 to 1 from earlier). I.e.

$$r(x) = \sum_{j=1}^{n} r(x_j) \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}$$

and so

$$\int_a^b w(x)r(x)\,\mathrm{d}x = \sum_{j=1}^{n} r(x_j) \int_a^b w(x) \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}\,\mathrm{d}x = \sum_{j=1}^{n} w_j\, r(x_j)$$

using (41). But, from (42),

$$r(x_j) = f(x_j) - q(x_j)\phi_n(x_j) = f(x_j)$$

since x_j are the zeros of ϕ_n and the result follows.

6.7.3 Gauss-Legendre quadrature

This refers to Gaussian quadrature on the interval $(-1,1)$ with $w(x)=1$. We have already established that Legendre polynomials apply in this case.

For each n we develop a new approximation capable of exactly integrating polynomials of degree $(2n-1)$ or less (and thus increasing accuracy with increasing n when an approximation for $f(x)$ not a polynomial).

For each n first calculate the n zeros x_j, $j=1,\ldots,n$ of $\phi_n(x)\equiv P_n(x)$ and then calculate the weights w_j using (41).

For the first few Legendre polynomials we find:

n	Legendre polynomial: $P_n(x)$	roots: x_j	weights: w_j
1	$P_1(x)=x$	$x_1=0$	$w_1=2$
2	$P_2(x)=\frac{3}{2}x^2-\frac{1}{2}$	$x_1=-1/\sqrt{3}$, $x_2=1/\sqrt{3}$	$w_1=1$, $w_2=1$
3	$P_3(x)=\frac{5}{2}x^2-\frac{3}{2}x$	$x_1=-\sqrt{3/5}$, $x_2=0$, $x_3=\sqrt{3/5}$	$w_1=w_3=5/9$, $w_2=8/9$

E.g.: The **two-point Gauss-Legendre quadrature scheme** is

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

and is exact for any cubic polynomial $f(x)$. This answers the Exercise earlier.

6.7.4 Integrals defined on general intervals

The value of integrals defined over a general interval, (a,b), can be found by mapping (a,b) onto $(-1,1)$ with, for e.g., the simple linear transformation $t=(2x-a-b)/(b-a)$.

Example: Approximate the value of $I=\displaystyle\int_1^{3/2} \mathrm{e}^{-x^2}\,\mathrm{d}x$. using Gauss-Legendre quadrature.

First map $(1,\frac{3}{2})$ to $(-1,1)$ with $t=2(2x-\frac{5}{2})$ or $x=(t+5)/4$. Then

$$I=\frac{1}{4}\int_{-1}^{1}\exp(-(t+5)^2/16)\,\mathrm{d}t.$$

Using table above, with $n = 1$, $I \approx (2/4)\exp(-25/16) = 0.104805$

With $n = 2$, $I \approx (1/4)\exp(-(5-1/\sqrt{3})^2/16) + (1/4)\exp(-(5+1/\sqrt{3})^2/16) = 0.109400$.

The exact value is 0.109364 to 6 d.p. accuracy.

Remark: One of the problems of Gauss-Legendre is determining the zeros of polynomials for larger n. It would be nice if the orthogonal polynomials had explicit zeros...

6.7.5 Gauss-Chebyshev quadrature

Consider Gauss quadrature with $a = -1$, $b = 1$ and the weight function $w(x) = (1-x^2)^{-1/2}$. The standard notation is that $\phi_i(x) = T_i(x)$ and are known as **Chebychev polynomials of the first kind**. The standardisation condition $T_n(1) = 1$ applies.

Let's use the manual approach. It will help to establish the value of some integrals before we start:

$$\int_{-1}^{1} \frac{x}{\sqrt{1-x^2}}\,dx = \int_{-1}^{1} \frac{x^3}{\sqrt{1-x^2}}\,dx = 0 \tag{43}$$

since the integrands are odd functions; also, using the substitution $x = \cos\theta$, we have

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}\,dx = \int_{0}^{\pi} \mathrm{d}\theta = \pi \quad \text{and} \quad \int_{-1}^{1} \frac{x^2}{(1-x^2)^{1/2}}\,\mathrm{d}x = \int_{0}^{\pi} \cos^2\theta\,\mathrm{d}\theta = \pi/2 \tag{44}$$

- First $T_0(x) = 1$ is polynomial of degree 0 s.t. $T_0(1) = 1$.
- Next $T_1(x) = A_1 x + B_1$ is polynomial of degree 1 and we need

  $$\langle T_1, T_0\rangle = 0 = A_1\langle x, 1\rangle + B_1\langle 1, 1\rangle$$

  The first inner product is zero by (43) and the second is π by (44). Hence $B_1 = 0$ and $A_1 = 1$ is needed to satisfy $T_1(1) = 1$. Thus $T_1 = x$.
- Next let $T_2(x) = A_2x^2 + B_2x + C_2$. We need

  $$\langle T_2, T_0\rangle = 0 = A_2\langle x^2, 1\rangle + B_2\langle x, 1\rangle + C_2\langle 1, 1\rangle$$

and so $0 = A_2(\pi/2) + C_2\pi$ or $C_2 = -\frac{1}{2}A_2$. We also need

$$\langle T_2, T_0\rangle = 0 = A_2\langle x^2, x\rangle + B_2\langle x, x\rangle + C_2\langle 1, x\rangle = B_2(\pi/2)$$

Thus $T_2(x) = A_2(x^2 - \frac{1}{2})$ and $T_2(1) = 1$ means $A_2 = 2$ so that $T_2(x) = 2x^2 - 1$.

Exercise: Show that the $(n = 2)$ two-point Gaussian-Chebyshev quadrature rule is

$$\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}\,\mathrm{d}x \approx \frac{\pi}{2}f(-1/\sqrt{2}) + \frac{\pi}{2}f(1/\sqrt{2})$$

such that the approximation will be exact for polynomials $f(x)$ up to and including cubics.

Proposition: The Chebyshev polynomials are given by $T_n(x) = \cos(n\,\cos^{-1} x)$.

Proof: We first check the standardisation condition

$$T_n(1) = \cos(n \cos^{-1} 1) = \cos(0) = 1.$$

Next we show (by induction) that $T_n(x)$ are are indeed polynomials of degree n. The first two functions are

$$T_0(x) = \cos(0) = 1, \qquad T_1(x) = \cos(\cos^{-1} x) = x,$$

polynomials of degree 0 and 1, respectively. Now consider

$$\begin{aligned} T_{n+1}(x) + T_{n-1}(x) &= \cos((n+1)\cos^{-1} x) + \cos((n-1)\cos^{-1} x) \\ &= 2\cos(n\cos^{-1} x)\cos(\cos^{-1} x) = 2xT_n(x). \end{aligned}$$

Thus, we have obtained the following **recursion relation**

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

This can be used to show that if $T_n(x)$ is a polynomial of degree n (for $n \geq 1$), $T_{n+1}(x)$ will be a polynomial of degree $n+1$. Since it is true for $n = 0, 1$ it is true for all n.

Finally, we show that $T_n(x)$ are indeed orthogonal on the interval $(-1,1)$ with respect to $w(x) = (1-x^2)^{-1/2}$. In the following we make use again of the substitution $x = \cos\theta$, $\mathrm{d}x = -\sin\theta \mathrm{d}\theta$:

$$\begin{aligned}\int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}\,\mathrm{d}x &= \int_0^\pi \cos(n\theta)\cos(m\theta)\,\mathrm{d}\theta = \frac{1}{2}\int_0^\pi \cos((n+m)\theta) + \cos((n-m)\theta)\,\mathrm{d}\theta \\ &= \frac{1}{2}\left[\frac{\sin((n+m)\theta)}{(n+m)} + \frac{\sin((n-m)\theta)}{(n-m)}\right]_0^\pi \qquad \text{provided } n \neq m \\ &= 0.\end{aligned}$$

Remark: Gauss-Chebyshev quadrature is particularly simple since the zeros x_j and weights w_j are explicitly known. The zeros follow from

$$0 = T_n(x) = \cos(n\,\cos^{-1} x) \qquad \Longrightarrow \qquad n\,\cos^{-1} x = j\pi - \frac{\pi}{2}, \qquad j = 1,\ldots,n.$$

Hence the zeros are defined by

$$x_j = \cos\left(\frac{(2j-1)\pi}{2n}\right), \qquad j = 1,\ldots,n.$$

We state (without proof) that the weights are $w_j = \pi/n$, $j = 1,\ldots,n$ (see homework for $n = 1,2,3$.)

6.7.6 Other important orthogonal polynomials

The following two polynomials are defined on infinite intervals and can be useful for approximating integrals on $(0,\infty)$ and $(-\infty,\infty)$:

(i) **Laguerre polynomials** $L_n(x)$**:** Defined on the interval $(0,\infty)$ with $w(x) = \mathrm{e}^{-x}$. Standardisation: coefficient of x^n in $L_n(x)$ is $(-1)^n/n!$.

(ii) **Hermite polynomials** $H_n(x)$**:** Defined on the interval $(-\infty,\infty)$ with weight function $w(x) = \mathrm{e}^{-x^2}$. Standardisation: coefficient of x^n in $H_n(x)$ is 2^n.

7 Ordinary differential equations: initial value problems (IVPs)

7.1 Introduction

Differential equations play a central role in many scientific fields including mathematics, physics, biology, chemistry, engineering as well as finance. In spite of what you are taught, most differential equations cannot be solved in closed form. The numerical solution of differential equations is particularly useful for PDEs (e.g. computational fluid dynamics models). Here, we will deal only with ODEs.

Numerical methods date back to Euler in the mid-18th century. The primary motivation then was in approximating the solutions to Newton's differential equations describing the motion of planets and comets.

7.1.1 First-order ODEs

We consider the **initial-value problem (IVP)** consisting of the general 1st order ODE for $y = y(t)$:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y), \qquad a < t < b \tag{45}$$

subject to the **initial condition**

$$y(a) = \alpha. \tag{46}$$

Here, α, a, b are given constants and f is a given function of the independent variable, t, and the dependent variable, y: t is used to indicate time (although not all IVPs relate to time-varying problems).

7.1.2 Higher-order ODEs and their reduction to a system of first-order ODEs

Q: What about ODEs which are of higher order ? For e.g. $y''(t) = -g$ (the motion of a projectile under gravity) subject to $y(0) = h$, $y'(0) = u$. These are important too, surely ?

A: Sure. So let's consider n-th order ODE written in its most general form:

$$y^{(n)}(t) = f(t, y, y', y'', \ldots, y^{(n-1)}), \qquad a < t < b \tag{47}$$

with initial conditions (since this is an initial-value problem)

$$y(a) = \alpha_1, \qquad y'(a) = \alpha_2, \qquad \ldots \qquad y^{(n-1)}(a) = \alpha_n. \tag{48}$$

(say) where α_i's are all given as is f.

Now let $u_1(t) = y(t)$, $u_2(t) = y'(t)$ and so on up to $u_n(t) = y^{(n-1)}(t)$. Then we see that

$$u_1'(t) = u_2(t), \qquad u_2'(t) = u_3(t), \qquad \ldots \qquad u_{n-1}'(t) = u_n(t)$$

plus

$$u_n'(t) = y^{(n)}(t) = f(t, u_1, u_2, \ldots, u_n)$$

from (47). Also from (48)

$$u_1(a) = \alpha_1, \qquad u_2(a) = \alpha_2, \qquad \ldots \qquad u_n(a) = \alpha_n.$$

We can organise these relations into a first-order **system of ODEs**:

$$\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix}' = \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_n \\ f(t, u_1, u_2, \ldots, u_n) \end{bmatrix}$$

and writing $\mathbf{u}(t) = (u_1(t), u_2(t), \ldots, u_n(t))^T$ means the above is

$$\mathbf{u}' = \mathbf{f}(t, \mathbf{u}) \tag{49}$$

where $\mathbf{f} = (u_2, u_3, \ldots, f)^T$ and with $\mathbf{u}(a) = \boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^T$.

Remark: (49) is just a vector version of the first-order scalar ODE (45) and everything developed for first-order equations can be extended naturally to the higher-order system (see later).

7.2 Euler's method

Euler's method is the simplest numerical approximation method.

Consider again the initial-value problem (IVP)

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y), \qquad a < t < b, \qquad \text{with } y(a) = \alpha. \tag{50}$$

We first divide the interval $[a, b]$ into N equal subintervals. The solution is approximated at discrete times $t_i = a + ih$, $i = 0, \ldots, N$, called **mesh points** where $h = (b - a)/N$ is the **step size**.

Euler's method can be derived from Taylor expanding:

$$y(t_{i+1}) = y(t_i + h) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi_i) \tag{51}$$

for some $\xi_i \in (t_i, t_i + h)$. Assuming h is small justifies ignoring the h^2 term and we let the approximation, y_i, to $y(t_i)$ that follows from this satisfy

$$y_{i+1} = y_i + hf(t_i, y_i)$$

after using (50) to replace y' by f. This **iterative scheme** is called **Euler's method**. It is an example of a **first-order difference equation** and is used for $i = 0, 1, \ldots, N - 1$, whereby $y_1, y_2, \ldots$ up to y_N, the approximation to $y(b)$, is obtained. It requires $y_0 = \alpha$ to start the iteration.

Remark 1: Euler's method can also be derived by replacing the derivative in (50) by the forward difference approximation

$$y'(t_i) = f(t_i, y(t_i)), \qquad \text{and} \qquad y'(t_i) \approx \frac{y(t_i + h) - y(t_i)}{h}.$$

Remark 2: Another basis for developing numerical ODE methods is to start with the (second) Fundamental Theorem of Calculus

$$\int_{t_i}^{t_i+h} y'(t)\,\mathrm{d}t = y(t_i + h) - y(t_i)$$

from which

$$y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y(t))\, dt$$

follows immediately and is exact. Numerical integration of the ODE is now reduced to how to approximate integrals. The most basic method is the 'rectangle rule' in which the integral above is approximated by $(t_{i+1} - t_i) f(t_i, y(t_i))$ (base times height) and this implies Euler's method again. But immediately better is the trapezoidal rule and this already hints at a plethora of methods for numerical integration of ODEs.

7.2.1 Example

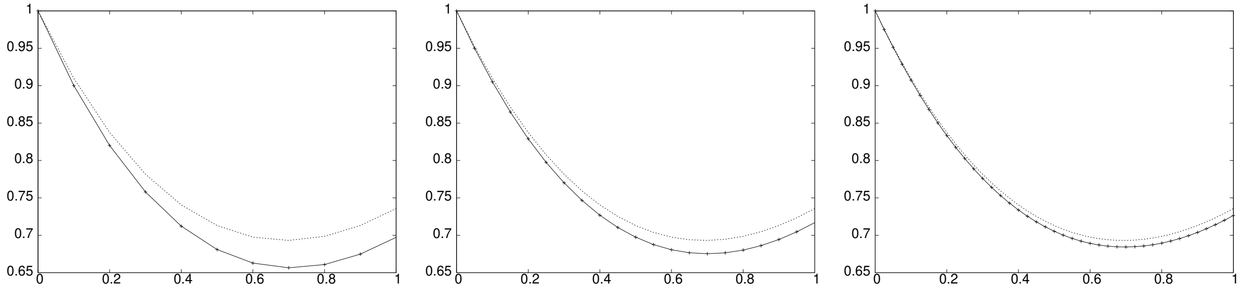
Consider the IVP

$$y' = -y + t, \qquad 0 < t < 1, \qquad \text{with } y(0) = 1.$$

Exact solution: We can find the exact solution using integrating factors and it gives

$$y(t) = 2\mathrm{e}^{-t} + t - 1.$$

Numerical experiment: In the plots below we show the output of computer code implementing the Euler method against the exact solution for step sizes $h = 0.1$, 0.05 and 0.025.



We observe that the error at $t = 1$ appears to half when h is halved. This suggests that the error is proportional to h.

Q: Can we confirm this error ?

A: Yes, but first we have to distinguish between different types of error.

7.3 Local truncation error

Defn: The **local truncation error** is the error introduced at each step *assuming that the solution at the beginning of each step is exact.* In going from step i to $i+1$, we denote this error by τ_{i+1}:

$$\tau_{i+1} = \underbrace{y(t_{i+1})}_{exact} - \underbrace{y_{i+1}}_{approx}$$

"Approx" means the numerical approximation to y_{i+1} assuming $y_i = y(t_i)$, the exact solution.

Example: For Euler's method it is

$$\begin{aligned} \tau_{i+1} &= y(t_i + h) - [y(t_i) + hf(t_i, y(t_i))] \\ &= y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi_i) - [y(t_i) + hy'(t_i)] = \frac{h^2}{2}y''(\xi_i) \end{aligned} \tag{52}$$

after using (51) and (50). Thus, the local truncation error for the Euler method is $O(h^2)$.

Remark: There are different definitions of the local truncation error and they can differ by a factor h from our definition (e.g. Burden and Faires).

7.4 Global error

In Euler's method the local truncation error of $O(h^2)$ isn't $O(h)$ as suggested by our numerical observations. Why ? Well, there we were observing the *cumulative error* after many steps. This gives rise to the notion of ...

Defn: The **global error** is the error at a fixed time; for example at the final time $t_N = b$. It is generally difficult to determine the global error precisely, because it depends on how errors accumulate and propagate during the iteration process. Indeed, this is a point we return to in a few pages. However, for now let us be satisfied with the following rough estimate:

Since the number of steps is $N = (b-a)/h = O(1/h)$, and the error made for a single step $\tau_{i+1} = O(h^{p+1})$, for some p, say, then the error at t_N is estimated to be $O(1/h) \times O(h^{p+1}) = O(h^p)$. To make this estimate, we have assumed that the errors accumulate linearly.

Defn: The **order of accuracy** of a method is defined to be p where the global error is $O(h^p)$.

E.g.: For Euler's method since $\tau_{i+1} = O(h^2)$ we assert that the global error is $O(h)$ so $p = 1$ and Euler's method is therefore a **first-order method**.

7.5 Solutions of linear difference equations

Sometimes one can solve the first-order difference equation defining Euler's method by hand (linearity is essential). This is useful as it allows us to compare y_i with the exact solution $y(t_i)$.

7.5.1 Example

Solve the difference equation from Euler's method applied to the ODE from Example 7.2.1:

$$y' = -y + t, \qquad 0 < t < 1, \qquad \text{with } y(0) = 1.$$

Euler's scheme for this IVP is

$$y_{i+1} = y_i + h(-y_i + t_i), \qquad i = 0, \dots, N-1,$$

with $y_0 = 1$ and $h = 1/N$, $t_i = ih$. This therefore reads as

$$y_{i+1} - (1-h)y_i = ih^2.$$

This is an **inhomogeneous difference equation** since there is a RHS term not related to y_i.

One can solve this similar to how one would solve an linear inhomogeneous ODE in A-level/1st year Calculus. That is, we start by writing

$$y_i = y_i^h + y_i^p$$

where y_i^h, y_i^p are solutions of the homogeneous problem and a particular solution (respectively). That is, we let y_i^h satisfy the **homogeneous** problem

$$y_{i+1}^h - (1-h)y_i^h = 0$$

and we look for $y_i^h = Az^i$ where A and z are constants[19], It follows that $z = (1-h)$ and so $y_i^h = A(1-h)^i$. Next we seek a **particular solution** y_i^p to

$$y_{i+1}^p - (1-h)y_i^p = ih^2.$$

Noting that i plays the part of x in continuous differential equations, we interpret the RHS term as being like a function proportional to x. Since we would look for particular solutions of the form $y(x) = Bx + C$ in such an instance, it makes sense here to look for a solution $y_i^p = iB + C$. Then it follows

$$(i+1)B + C - (1-h)(iB + C) = ih^2$$

and, since this equation holds for varying i, we equate coefficients of i to get $B - (1-h)B = h^2$ and the constants to get $B + C - (1-h)C = 0$. This gives $B = h$ and $C = -1$ and $y_i^p = ih - 1$. Thus, the **general solution** is

$$y_i = A(1-h)^i + ih - 1.$$

Finally, when $i = 0$ we have $y_0 = 1$ and this determines $A = 2$. In this way we have found the exact solution

$$y_i = 2(1-h)^i + ih - 1$$

to the difference equation resulting from Euler's method. That is, these are the values given explicitly for any given i that are computed from the iterative scheme.

7.5.2 Analysis of error for example above

Since this is explicit and we already have the exact solution $y(t) = 2\mathrm{e}^{-t} + t - 1$ we can compute the error for this example exactly, defined as

$$E(t_i) = y(t_i) - y_i = 2\mathrm{e}^{-t_i} + t_i - 1 - (2(1-h)^i + ih - 1).$$

Note, this is **not** the local truncation error, since it includes the accumulation of error. Nor is it the global error since t_i is not a fixed time (yet).

Now $t_i = ih$ and so

$$E(t_i) = 2\mathrm{e}^{-ih} - 2(1-h)^i. \tag{53}$$

[19] this is like using the ansatz $y(x) = A\mathrm{e}^{rx}$ for the Complementary Function when solving ODEs

Let us Taylor expand both terms to get

$$E(t_i) = 2(1 - ih + i^2h^2/2 + \ldots) - 2(1 - ih + i(i-1)h^2/2 + \ldots) = ih^2.$$

This increases with i and is $O(h^2)$, the same as the local truncation error. But note that we have assumed ih is small in throwing away higher order terms in the expansions above. If ih is small, this means it is only valid for small times At the final time when $i = N = 1/h$, $ih = 1$ is certainly **not** small.

This means the analysis above cannot be used to determine the global error.

Instead for $E(t_N)$ we use $ih = N$ and $N = 1/h$ directly in (53) to give

$$E(t_N) = 2\mathrm{e}^{-1} - 2(1-h)^{1/h}$$

which **is** now the global error since $t_N = 1$ is fixed. This expression is the basis for a different asymptotic calculation. And we expand

$$\begin{aligned}(1-h)^{1/h} = \exp\{(1/h)\ln(1-h)\} &= \exp\{(1/h)(-h - h^2/2 - h^3/3 - \ldots\} = \exp\{-1 - h/2 + \ldots\}\\ &= \mathrm{e}^{-1}(1 - h/2 - \ldots).\end{aligned}$$

Using this in the above gives

$$E(t_N) = 2\mathrm{e}^{-1} - 2\mathrm{e}^{-1}(1 - h/2) + O(h^2) = \mathrm{e}^{-1}h.$$

This confirms the global error in the example is $O(h)$.

7.6 Euler's method for higher-order ODEs.

We'll consider second order ODEs, but it is straightforward to generalise the discussion to higher-order ODEs (as suggested in Section 7.1.2). Consider

$$y'' = f(t, y, y'), \qquad a < t < b, \qquad \text{with } y(a) = \alpha,\ y'(a) = \beta.$$

This IVP can be transformed into a system of first-order ODEs by, say, setting $v(t) = y'(t)$ so that

$$\begin{aligned} y' &= v, & y(a) &= \alpha,\\ v' &= f(t, y, v), & v(a) &= \beta. \end{aligned}$$

Euler's method can be applied to each of these lines by letting v_i and y_i denote the approximations for $v(t_i)$ and $y(t_i)$, respectively. This results in

$$\begin{aligned} y_{i+1} &= y_i + hv_i, & y_0 &= \alpha, \\ v_{i+1} &= v_i + hf(t_i, y_i, v_i), & v_0 &= \beta. \end{aligned}$$

This is a system of first-order difference equations which can be solved by iteration.

Alternatively, we can rearrange the first line to

$$v_i = (y_{i+1} - y_i)/h$$

and substitute into the second line which leads to a **second-order difference equation** in the discrete variable y_i:

$$\frac{y_{i+2} - y_{i+1}}{h} = \frac{y_{i+1} - y_i}{h} + h\,f\left(t_i, y_i, \frac{y_{i+1} - y_i}{h}\right), \qquad y_0 = \alpha, \qquad \frac{y_1 - y_0}{h} = \beta$$

which reduces to

$$y_{i+2} = 2y_{i+1} - y_i + h^2 f\left(t_i, y_i, \frac{y_{i+1} - y_i}{h}\right), \qquad y_0 = \alpha, \qquad y_1 = \alpha + h\beta.$$

Remark 1: We have asserted (with some supporting evidence, athough there are theorems which really nail this down) that Euler's method is first-order accurate implying that its global error is $O(h)$.

Remark 2: $O(h)$ is not accurate enough for anything beyond the most basic applications. The focus from now will be in developing more sophisticated methods which improve the approximation by reducing the local truncation error.

7.7 Higher-order Taylor methods

One (obvious) way to improve the approximation is to include more terms in the Taylor expansion that gave rise to Euler's method. I.e.

$$y(t_i + h) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \frac{h^3}{6}y'''(t_i) + \ldots$$

7.7.1 Example: Taylor method of order 2

Let us include one extra term than for Euler's method.

We use the ODE to replace $y'(t) = f(t, y)$, but we also need an expression for the unknown second derivative $y''(t)$ (if we don't know $y(t)$, we can't say what $y''(t)$ is).

It is obtained by differentiating $y'(t) = f(t, y)$ with respect to t, noting that f is a multi-variable function and t appears twice. That is

$$y''(t) = \frac{\mathrm{d}}{\mathrm{d}t} f(t, y(t)) = \frac{\mathrm{d}t}{\mathrm{d}t}\frac{\partial f}{\partial t} + \frac{\mathrm{d}y}{\mathrm{d}t}\frac{\partial f}{\partial y} = f_t + f_y f$$

using the chain rule. We are given f so, in principle at least, we know its partial derivatives f_t and f_y.

We therefore obtain the following iteration scheme

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}\left[f_t(t_i, y_i) + f_y(t_i, y_i)\, f(t_i, y_i)\right], \qquad y_0 = \alpha. \tag{54}$$

where y_i is an approximation for $y(t_i)$. Since we neglected terms of $O(h^3)$ in the derivation of this scheme its local truncation error is $O(h^3)$. (54) is called the **Taylor method of order 2** and is an example of a higher-order Taylor method.

7.7.2 Taylor method of even higher order ?

For further improvement one needs to include the next term in the Taylor expansion $h^3 y'''(t_i)/6$ where

$$y'''(t) = \frac{\mathrm{d}}{\mathrm{d}t} y''(t) = \frac{\mathrm{d}}{\mathrm{d}t}[f_t + f_y f] = f_{tt} + f_{ty} f + f_{ty} f + f_{yy} f^2 + f_y f_t + f_y f_y f$$

(copious use of the chain rule again).

Remark: If it wasn't obvious before, it is now: higher-order Taylor methods have the advantage of higher order local truncation errors, but they require the evaluation of higher-order partial derivatives of the function $f(t, y)$. This can be complicated and time-consuming.

7.8 Runge-Kutta methods

Runge-Kutta methods aim to reduce the local truncation errors but by only using evaluations of $f(t, y)$ (and not its derivatives).

IDEA: To introduce the idea of the Runge-Kutta method we consider an iteration scheme of the form

$$y_{i+1} = y_i + af(t_i + b, y_i + c) \tag{55}$$

Note : If $a = h$ and $b = c = 0$ this agrees with Euler's method.

Q: Can we choose a, b, c "optimally" to reduce the truncation error ?

A: Yes, and we do so by matching this scheme to from Taylor's second order.

Let us apply a Taylor expansion in the two arguments of the function $f(t, y)$ in (55):

$$y_{i+1} = y_i + af(t_i, y_i) + af_t(t_i, y_i)b + af_y(t_i, y_i)c + \frac{a}{2}\left[f_{tt}b^2 + 2f_{ty}bc + f_{yy}c^2\right] + \ldots \tag{56}$$

Comparing (56) with equation (54) shows that we should choose

$$a = h, \quad ab = \frac{h^2}{2}, \quad ac = \frac{h^2}{2}f(t_i, y_i), \quad \Longrightarrow \quad a = h, \quad b = \frac{h}{2}, \quad c = \frac{h}{2}f(t_i, y_i).$$

Since a, b and c are all of order h it follows that the next term in the Taylor expansion in (54) are all $O(h^3)$.

Hence the Runge-Kutta iteration scheme has the following form

$$y_{i+1} = y_i + hf\left(t_i + \frac{h}{2}, y_i + \frac{hk}{2}\right) \qquad \text{where} \qquad k = f(t_i, y_i) \tag{57}$$

Note that $t_i + h/2$ is in the middle between t_i and t_{i+1}, and $y_i + hf(t_i, y_i)/2$ is in the middle between y_i and y_{i+1} in Euler's method. For this reason it is also called **the midpoint method**[20].

[20] we might also imagine that this scheme might emerge from the Fundamental Theorem of Calculus starting point outlined earlier in which the integral is approximated by the Trapezoidal rule – a midpoint rule – rather than a rectangle rule

It is classified as a **Runge-Kutta method of order 2** (or RK2) because its local truncation error is $O(h^3)$ and hence we expect a global error of $O(h^2)$.

Remark: Its application requires two functional evaluations per time step, $k = f(t_i, y_i)$ and $f(t_i + h/2, y_i + hk/2)$.

Numerical experiment: The following table shows how RK2 compares to the Euler method when applied to our previous example $y' = -y + t$ with $y(0) = 1$. In both cases the step size is $h = 0.1$. The second-order method is clearly more accurate.

	$y(t)$		
t	Euler	RK2	exact
0.2	0.82000	0.838050	0.83746
0.4	0.71220	0.741604	0.74064
0.6	0.66288	0.698807	0.69762
0.8	0.66093	0.699950	0.69866
1.0	0.69736	0.737082	0.73575

7.8.1 A family of RK2 methods

There are other Runge-Kutta methods of order 2. They can be obtained by starting from

$$y_{i+1} = y_i + af(t_i, y_i) + bf(t_i + c, y_i + d),$$

and by placing one condition on the four a, b, c, and d, before requiring use the order 2 Taylor method as before to place three further conditions. In this way one obtains a one-parameter family of Runge-Kutta methods of order 2.

7.8.2 Higher-order Runge-Kutta methods

If we allow more functional evaluations per time step then we can achieve higher order accuracy.

Of particular note, and very popular, is the **fourth-order Runge-Kutta method** known as **RK4**. It requires four functional evaluations per step. The derivation is somewhat involved and we cite the result here for interest only. The functional evaluations required are

$$\begin{aligned}
k_1 &= f\left(t_i, y_i\right) \\
k_2 &= f\left(t_i + \frac{h}{2}, y_i + \frac{hk_1}{2}\right) \\
k_3 &= f\left(t_i + \frac{h}{2}, y_i + \frac{hk_2}{2}\right) \\
k_4 &= f\left(t_i + h, y_i + hk_3\right)
\end{aligned}$$

and the iteration scheme has the form

$$y_{i+1} = y_i + \frac{h}{6}[k_1 + 2k_2 + 2k_3 + k_4].$$

7.9 Multistep methods

7.9.1 Introduction

The methods that we have considered so far are called **one-step methods** and can be written in the general form

$$y_{i+1} = y_i + h\,\Phi(t_i, y_i, h)$$

where Φ is more and more complicated for higher-order methods.

They use the approximation y_i from the previous mesh point t_i to evaluate y_{i+1}. After the calculation of y_{i+1} this information is discarded, and the next iteration uses information from the following mesh point.

Multistep methods employ a different philosophy. They use a number of previous approximations $y_i, y_{i-1}, y_{i-2}, \ldots$ and previous mesh points $t_i, t_{i-1}, t_{i-2}, \ldots$ to arrive at an estimate of the solution at t_{i+1}.

This is clearly efficient because one can reuse functional evaluations from previous steps without needing new calculations.

Defn: A **linear k-step multistep method** uses approximations from the previous k steps and can be written in its most general form as

$$y_{i+1} = \sum_{j=1}^{k} \alpha_j \, y_{i+1-j} + h \sum_{j=0}^{k} \beta_j \, f(t_{i+1-j}, y_{i+1-j}) \tag{58}$$

or perhaps more clearly as

$$\begin{aligned} y_{i+1} &= \alpha_1 y_i + \alpha_2 y_{i-1} + \ldots + \alpha_k y_{i+1-k} \\ &\quad + h\beta_0 f(t_{i+1}, y_{i+1}) + h\beta_1 f(t_i, y_i) + \ldots + h\beta_k f(t_{i+1-k}, y_{i+1-k}). \end{aligned} \tag{59}$$

Defn: If $\beta_0 \neq 0$ then y_{i+1} appears on both sides of the equation and (59) is then an implicit equation for y_{i+1} and the method is called **implicit**. If $\beta_0 = 0$ then the method is said to be **explicit**.

Remark: The reason for including implicit equations is that they are often more accurate than explicit methods as we will see later.

Example: Multistep formulas arise naturally by applying various approximation formulas for derivatives in Section 5 on Differentiation.

We have already explained that approximating the derivative in the ODE

$$y'(t_i) = f(t_i, y(t_i))$$

by the forward difference formula $y'(t_i) \approx [y(t_{i+1}) - y(t_i)]/h$ gives Euler's method. But using the more accurate central difference approximation $y'(t_i) \approx [y(t_{i+1}) - y(t_{i-1})]/2h$ instead gives

$$y_{i+1} = y_{i-1} + 2hf(t_i, y_i) \tag{60}$$

which is an example of an **explicit two-step method**.

Remark: The fact that a multistep method requires the solution at k mesh points to determine the solution at the next mesh point raises a new difficulty: how do we start the iteration ? Recall that the initial condition tells us only y_0.

The solution is often to use one-step methods of the required order of accuracy (such as Runge-Kutta) to approximate the solution at the first k mesh points, $y_0, y_1, \ldots y_{k-1}$ and use the multistep method thereafter.

7.9.2 Examples of multistep methods

The most popular k-step multistep formulas are (adopting the shorthand notation $f_j = f(t_j, y_j)$) listed below.

1. Adams-Bashforth (ABk):

$$y_{i+1} = y_i + h\beta_1 f_i + h\beta_2 f_{i-1} + \ldots + h\beta_k f_{i+1-k}.$$

   ($\alpha_1 = 1$, $\alpha_j = 0$ for $j = 2, \ldots, k$ and $\beta_0 = 0$, explicit)

2. Adams-Moulton (AMk):

$$y_{i+1} = y_i + h\beta_0 f_{i+1} + h\beta_1 f_i + \ldots + h\beta_k f_{i+1-k}.$$

   ($\alpha_1 = 1$ and $\alpha_j = 0$ for $j = 2, \ldots, k$, implicit).

3. Backward Differentiation formulas (BDk):

$$y_{i+1} = \alpha_1 y_i + \alpha_2 y_{i-1} + \ldots + \alpha_k y_{i+1-k} + h\beta_0 f_{i+1}$$

   ($\beta_j = 0$ for $j = 1, \ldots, k$, implicit).

Q: Many coefficients are set to zero (we will say why this is so later), but how do we choose the values of the remaining coefficients ?

A: Simples: we minimise the local truncation error, τ_{i+1}.

7.9.3 Example: Derivation of AB2

From Section 7.9.2 with $k = 2$, AB2 is given by

$$y_{i+1} = y_i + h\beta_1 f_i + h\beta_2 f_{i-1}.$$

Recall the definition of the local truncation error being the difference between the exact solution at t_{i+1} and the numerical approximation at t_{i+1} assuming the solution at t_i was exact.

Hence we obtain the local truncation error from

$$\begin{aligned}\tau_{i+1} &= y(t_i+h) - [y(t_i) + h\beta_1 y'(t_i) + h\beta_2 y'(t_i-h)] \\ &= \left[y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \frac{h^3}{6}y'''(t_i) + \ldots\right] \\ &\quad -y(t_i) - h\beta_1 y'(t_i) - h\beta_2\left[y'(t_i) - hy''(t_i) + \frac{h^2}{2}y'''(t_i) + \ldots\right]\end{aligned}$$

after using $y' = f$ and Taylor expanding everything around t_i. Collecting terms proportional to each power of h gives

$$\tau_{i+1} = hy'(t_i)\left[1-\beta_1-\beta_2\right] + h^2 y''(t_i)\left[\frac{1}{2}+\beta_2\right] + h^3 y'''(t_i)\left[\frac{1}{6}-\frac{\beta_2}{2}\right] + \ldots$$

Our aim is to eliminate as many terms possible and, since we have two parameters β_1 and β_2 we can take out terms proportional to h and h^2 by choosing

$$1-\beta_1-\beta_2 = 0, \qquad \text{and} \qquad \frac{1}{2}+\beta_2 = 0$$

which gives $\beta_1 = \frac{3}{2}$ and $\beta_2 = -\frac{1}{2}$ and we note that h^3 term is non-vanishing since $\frac{1}{2}\beta_2 - \frac{1}{6} = -\frac{1}{4}$.

Hence, the AB2 scheme is given by

$$y_{i+1} = y_i + \frac{h}{2}(3f_i - f_{i-1}).$$

Since the coefficient of h^3 does not vanish we conclude that $\tau_{i+1} = O(h^3)$. The order of accuracy is therefore 2 ($p = 2$).

Remark 1: AB1 is a one-step method and is easily seen to be Euler's method. One can show that ABk is kth-order accurate.

Remark 2: AMk can be shown to be $(k+1)$-th order accurate and this is because AMk has an additional free parameter β_0 which can be tuned to eliminate one more order of truncation error.

Exercises: Follow the methods in the example above to show that:

(i) BD1 is given by

$$y_{i+1} = y_i + hf_{i+1};$$

(ii) AM2 is given by

$$y_{i+1} = y_i + \frac{h}{12}(5f_{i+1} + 8f_i - f_{i-1}).$$

7.9.4 Numerical experiment

Consider

$$y'(t) = y(t), \qquad 0 < t < 2, \qquad y(0) = 1$$

which has the exact solution $y(t) = \mathrm{e}^t$. Three numerical methods are used to compute solutions: (i) Euler's method; (ii) (60) based on a central-difference (CD) approximation; (iii) AB4 ... which is given by

$$y_{i+1} = y_i + \frac{h}{24}(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3})!$$

The table below shows the (global) errors $E(t_N)$ at the end point $t_N = 2$ as a function of h. We see that errors reduce by different factors upon the halving of h, indicating the orders of accuracy of the schemes.

	$E(t_N)$			
Scheme	$h = 0.2$	$h = 0.1$	$h = 0.05$	ratio of last two
Euler	1.20	0.662	0.349	$1.9 \approx 2$
CD	0.0906	0.0238	0.00607	$3.92 \approx 4$
AB4	0.0042	0.00038	0.00003	$13.6 \approx 16$

7.9.5 Stability

Q: In a general k-step multistep method (59) one has $(2k+1)$ coefficients $\alpha_1, \ldots, \alpha_k$ and $\beta_0, \ldots \beta_k$. If none are set to zero, then it ought to be possible in the expansion of the local truncation error to eliminate all terms proportional to $y, y', \ldots, y^{(2k)}$. In this way one could orchestrate it such that $\tau_{i+1} = O(h^{2k+1})$ and $p = 2k$. So why not do this ?

A: It turns out that the resulting formulas are numerically **unstable** as we shall now see...

One must distinguish between two different definitions of stability:

Defn: If, for a fixed $T > 0$, the numerical approximation to $y(T)$ remains bounded as $h \to 0$ then we say that the numerical method is **stable** or **zero stable**.

Defn: If, for a fixed $h > 0$, the numerical approximation to $y(T)$ remains bounded as $T \to \infty$ then we say that the numerical method is **time-stable** or **A-stable** or **absolutely stable**.

7.10 Stability (or zero-stability)

Let us consider the first of these which is best described by an example.

7.10.1 Example

Consider the following IVP

$$\frac{\mathrm{d}y}{\mathrm{d}t} = y, \qquad 0 < t < 1, \qquad \text{with } y(0) = 1.$$

Exact solution: Easy to show $y(t) = \mathrm{e}^t$ and $y(1) = \mathrm{e} \approx 2.71828$.

Numerical experiment: Let's apply two different 2-step multistep methods to the problem above:

(i) **AB**2: $y_{i+1} = y_i + \frac{1}{2}h(3f_i - f_{i-1})$

(ii) **SB**2: $y_{i+1} = -4y_i + 5y_{i-1} + h(4f_i + 2f_{i-1})$

AB2 has order of accuracy 2, and SB2 is a scheme (SB = Special Brew) that I've invented but which I've designed to have order of accuracy 3.

Because these are two-step methods we need y_1 as well as $y_0 = 1$ and we've used the exact value $y_1 = y(h) = \mathrm{e}^h$ to avoid bias. We see that AB2 tends to the correct value as $h \to 0$, but SB2

h	0.2	0.1	0.05	0.025
AB2	2.68771	2.70881	2.71568	2.71760
SB2	2.73433	-0.12720	-1.62×10^6	-9.34×10^{18}

becomes increasingly unbounded in the same limit (even though the local truncation error is $O(h^4)$).

Q: What happened ?

A: The order of accuracy is based on the argument that the global error is $O(1/h) \times O(h^{p+1}) = O(h^p)$ for a local truncation error of $O(h^{p+1})$. That is, it assumes the error accumulates **linearly**.

But the local truncation error can "feed off itself" and accumulate errors exponentially and this is the signature of numerical instability.

7.10.2 Analysing stability

In order to investigate stability we consider the limit $h \to 0$.

In the $h \to 0$ limit the α_j terms in the multistep formula (59) remain important but the $h\beta_j$ terms can be neglected. This step simplifies the numerical scheme enough to allow us to analyse the global error.

Example: Consider scheme SB2 above under limit $h \to 0$. We obtain

$$y_{i+1} = -4y_i + 5y_{i-1}$$

or

$$y_{i+1} + 4y_i - 5y_{i-1} = 0.$$

This is a homogeneous **second-order difference equation** with constant coefficients. We can solve it using methods we outlined earlier in the Chapter:

We let $y_i = Az^i$ and, upon substitution, get

$$Az^{i+1} + 4Az^i - 5Az^{i-1} = 0.$$

Dividing through by Az^{i-1} (we are not interested in the trivial solution, $z = 0$) gives

$$0 = z^2 + 4z - 5 \quad \Longrightarrow \quad 0 = (z+5)(z-1) \quad \Longrightarrow \quad z = -5 \text{ or } z = 1.$$

Thus the **general solution** is given by the linear superposition of both solutions $y = Az^i$ with $z = 1$ and $z = -5$ as

$$y_i = A(-5)^i + B1^i = A(-1)^i 5^i + B.$$

Now $y_0 = 1$ implies $A + B = 1$ and $y_1 = \mathrm{e}^h = 1 + h + O(h^2)$. Neglecting $O(h^2)$ terms and using $i = 1$ gives $1 + h = A(-5) + B$. Solving gives

$$A = -h/6, \qquad B = 1 + h/6.$$

Thus our exact numerical solution is

$$y_i = -\frac{h}{6}(-1)^i 5^i + 1 + \frac{h}{6}.$$

We can see that the first term on the RHS grows exponentially as i increases. Indeed, at $i = N = 1/h$, we have

$$y_N = -\frac{h}{6}(-1)^N 5^{1/h} + 1 + \frac{h}{6} = -\frac{h}{6}(-1)^N \mathrm{e}^{(1/h)\ln(5)} + 1 + \frac{h}{6}$$

and the global error, $E(1) = y(1) - y_N$ is unbounded as $h \to 0$ and $N \to \infty$.

Defn: The component of the solution $A(-5)^i$ is known as a **parasitic solution**. It is introduced by the discretisation rather than by the ODE itself. If such a mode exists, it will always be excited by errors in the calculation such as round-off errors (as discussed in Chapter 1) or a small inaccuracy in the initial data. That is, the numerical instability is not a function of the particular ODE or initial condition in this problem, it is a property of the scheme.

Exercise: Show that the AB2 scheme is stable.

7.10.3 The root condition for stability

Let us now apply the same analysis to the general k-step multistep method (59). Consider

$$y_{i+1} = \sum_{j=1}^{k} \alpha_j y_{i+1-j} + h \sum_{j=0}^{k} \beta_j f_{i+1-j} \qquad \text{where} \qquad f_i = f(t_i, y_i).$$

Taking the $h \to 0$ limit reduces this to

$$y_{i+1} - \alpha_1 y_i - \alpha_2 y_{i-1} - \ldots - \alpha_k y_{i+1-k} = 0 \tag{61}$$

which is a homogeneous linear difference equation with constant coefficients. We look for solutions of the form $y_i = Az^i$ and thus obtain

$$z^{i+1} - \alpha_1 z^i - \alpha_2 z^{i-1} - \ldots - \alpha_k z^{i+1-k} = 0$$

since we are not interested in the trivial solution $z = 0$, dividing by z^{i+1-k} results in

$$z^k - \alpha_1 z^{k-1} - \alpha_2 z^{k-2} - \ldots - \alpha_{k-1} z - \alpha_k = 0.$$

This is called the **characteristic polynomial** and is of degree k. There are k roots of this polynomial which we label $z_j \in \mathbb{C}$, $j = 1, 2, \ldots, k$. There are two possibilities:

(i) The roots are distinct and then the general solution of (61) is

$$y_i = A_1 z_1^i + A_2 z_2^i + \ldots + A_k z_k^i.$$

(ii) If any of the roots are repeated, e.g. if $z_1 = z_2$, then $A_2 z_2^i$ is replaced with $A_2 i z_1^i$. This is just like repeated roots to the characteristic equation for 2nd order ODEs where you form a new linearly-independent second solution by multiplying by x.

Given the preceding definitions we can state the following theorem:

Theorem (root condition): A linear multistep formula is **stable** if and only if all roots z_j of its characteristic polynomial satisfy $|z_j| \leq 1$, and any root with $|z_j| = 1$ has multiplicity one.

7.10.4 Global error, consistency and convergence

If the method is stable then one can specify the order of the global error of the method.

Theorem (Dahlquist): If a linear multistep formula has local truncation error $O(h^{p+1})$ and is stable, then the global error is $O(h^p)$. If $p > 0$, the method will converge to the exact solution as $h \to 0$ and is called convergent with order of accuracy p.

Defn: A linear multistep method is said to be **consistent** if the local truncation error goes to zero faster than h: $\lim_{h\to 0} \tau_{i+1}/h = 0$. That is, $p > 0$.

Defn: From the definition of **consistent** it follows also that a method is **convergent** if it is both consistent and stable.

Example 1: Investigate stability and convergence of the following method

$$y_{i+1} = \frac{1}{2}y_i + \frac{1}{2}y_{i-1} + 2hf_i$$

Consider the $h \to 0$ limit:

$$y_{i+1} - \frac{1}{2}y_i - \frac{1}{2}y_{i-1} = 0$$

Setting $y_i = Az^i$ leads to the characteristic polynomial

$$0 = z^2 - \frac{1}{2}z - \frac{1}{2} = (z-1)\left(z+\frac{1}{2}\right)$$

Both roots satisfy $|z| \le 1$, and the root with $|z| = 1$ is simple. It follows from the root condition that the method is stable. Let us now investigate the order of accuracy. The local truncation error is

$$\begin{aligned}
\tau_{i+1} &= y(t_i+h) - \left[\frac{1}{2}y(t_i) - \frac{1}{2}y(t_i-h) - 2hy'(t_i)\right] \\
&= y(y_i) + hy'(t_i) + \ldots - \frac{1}{2}y(t_i) - \frac{1}{2}\left[y(t_i) - hy'(t_i) + \ldots\right] - 2hy'(t_i) \\
&= -\frac{1}{2}hy'(t_i) + \ldots
\end{aligned}$$

The local truncation error is $O(h)$, so the order of accuracy is $p = 0$. The method is not consistent and also not convergent.

Example 2: Investigate stability and convergence of the following multistep method

$$y_{i+1} = y_{i-3} + \frac{4}{3}h\,(f_i + f_{i-1} + f_{i-2})$$

Here the number of steps is $k = 4$. The characteristic polynomial is

$$z^4 - 1 = 0 \qquad \Longrightarrow \qquad z = 1, \quad z = -1, \quad z = i, \quad z = -i$$

All roots satisfy $|z| = 1$ and are simple. Hence the method is stable.

Additionally (exercise) one can show that the local truncation error is $O(h^3)$. Consequently, the order of accuracy $p = 2$ and therefore the method is convergent.

7.10.5 A comment on ABk and AMk methods

The Adams-Bashforth ($\beta_0 = 0$) and Adams-Moulton ($\beta_0 \neq 0$) formulae are given by

$$y_{i+1} = y_i + h\beta_0 f_{i+1} + h\beta_1 f_i + \ldots + h\beta_k f_{i+1-k}$$

The characteristic polynomial in all cases is $z - 1 = 0$, and from the root condition it follows that the methods are all stable.

As mentioned before, one can show that the order of accuracy of ABk is $p = k$, and the order of accuracy of AMk is $p = k + 1$. Hence the methods are convergent (since $k \geq 1$).

7.10.6 Maximal order of accuracy

The maximal order of accuracy that can be achieved is determined by the following theorem:

Theorem (first Dahlquist stability barrier): A **stable** k-step multistep formula can have order of accuracy at most:

- k: if explicit.
- $k + 1$: if implicit and k is odd.
- $k + 2$: if implicit and k is even.

(No proof)

Remark: It is for this reason many of the α_j and β_j are usually set equal to zero.

Remark: Implicit methods can be used practically in so-called predictor-corrector methods. For e.g. one may use ABk to step forwards using an explicit scheme to find $\tilde{y}_{i+1}$, say, (the predictor step) and then use this value in an implicit scheme, such as AMk find y_{i+1} (the corrector step).

7.11 Time stability (absolute stability or A-stabilty)

7.11.1 Introduction

In the previous section stability is defined by the behaviour in the limit $h \to 0$: a method is stable if the approximation for the solution $y(t)$ at a fixed time $t = T$ remains bounded as $h \to 0$.

Remark: Since neglecting $O(h)$ terms is the same as setting $f = 0$ in the ODE, stability is effectively determined by consider the canonical ODE $y' = 0$ in $0 < t < T$ with $y(0) = 1$.

A different question asks how small h needs to be in order to get good numerical results. We will see that this question is connected to a different notion of stability that is called **time stability** or **absolute stability**. It is determined by the behaviour of the approximate solution for fixed h as time $t \to \infty$.

7.11.2 The canonical time-stability problem

Time stability is determined by looking at another canonical IVP given by

$$y'(t) = \lambda y(t), \qquad t > 0, \qquad \text{with } y(0) = \alpha \tag{62}$$

where c is a real constant and $\lambda \in \mathbb{C}$, $\operatorname{Re}\{\lambda\} < 0$.

The exact solution is $y(t) = \alpha e^{\lambda t}$. Since $\operatorname{Re}\{\lambda\} < 0$ then $y(t) \to 0$ as $t \to \infty$.

Defn: A method is called **time stable** or **A-stable** if, for fixed values of $h > 0$ and $\lambda \in \mathbb{C}$, $\operatorname{Re}\{\lambda\} < 0$, it mimics this behaviour (i.e. the approximate numerical solution also approaches zero as $t \to \infty$.)

Remark: As for stability, the canonical time-stability problem is chosen so that an exact solution to the numerical difference equation that results can be determined exactly. But its motivation comes from

7.11.3 Example: Euler's method

If we apply Euler's method to the canonical time-stability problem we obtain

$$y_{i+1} = y_i + h\lambda y_i = (1 + h\lambda)y_i, \qquad y_0 = c.$$

The solution to this difference equation ($y = Az^i$ etc...) is

$$y_i = c(1 + h\lambda)^i.$$

We see that $y_i \to 0$ as $i \to \infty$ if and only if

$$|1 + h\lambda| < 1$$

or

$$|h\lambda - (-1)| < 1.$$

Although $h > 0$ is real, λ is considered to be complex and so we can describe this condition as a function of $\bar{h} = h\lambda$ in the complex $\bar{h}$-plane. Indeed, it says that the distance of $\bar{h}$ from the point -1 on the real axis should be less than 1. This is the interior of a circle of radius 1 centred on $\bar{h} = -1$ and falls entirely to the left of the imaginary axis.

This region is called the **time-stability region** for Euler's method.

For a particular value of λ it restricts the choice of h for which the method is time stable or absolutely stable.

For example, consider $\lambda \in \mathbb{R}$ and $\lambda < 0$ then the above condition leads to $-1 < 1 + h\lambda < 1$ or $-2 < h\lambda < 0$, and we obtain the following condition for h

$$0 < h < \frac{-2}{\lambda} = \left|\frac{2}{\lambda}\right|$$

We see that if $|\lambda|$ is very big then h has to be very small to get a reasonable result.

7.11.4 Example: Backward (implicit) Euler method

This scheme is new to us and given by

$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}).$$

We apply it to the canonical time-stability problem (62) to get

$$y_{i+1} = y_i + h\lambda y_{i+1} \qquad \text{or} \qquad y_{i+1} = \frac{1}{1-h\lambda} y_i.$$

This difference equation has the solution

$$y_i = c(1-h\lambda)^{-i}$$

and it satisfies $y_i \to 0$ as $i \to \infty$ if and only if

$$|1 - h\lambda| > 1.$$

This condition holds in the the exterior of a circle in the $\bar{h} = h\lambda$-plane with radius 1 and centred on $h\lambda = 1$.

Since we assumed that $\operatorname{Re}\{\lambda\} < 0$ and $h > 0$ we are only interested in the left half of the complex $h\lambda$-plane ($\operatorname{Re}\{h\lambda\} < 0$). Hence the condition $|1 - h\lambda| > 1$ does not lead to any restriction on h and the method is time stable for any value $h > 0$.

7.11.5 General result

We now apply a general k-step multistep method to our canonical problem $y' = \lambda y$, $\operatorname{Re}\lambda < 0$, $y(0) = c$ to get:

$$y_{i+1} = \alpha_1 y_i + \ldots + \alpha_k y_{i+1-k} + h\lambda\beta_0 y_{i+1} + h\lambda\beta_1 y_i + \ldots + h\lambda\beta_k y_{i+1-k}.$$

This is again a difference equation with constant coefficients, and we can find solutions of the form $y_i = Az^i$. After dividing by z^{i+1-k} we obtain the **stability polynomial**

$$(63) \qquad (1 - h\lambda\beta_0)\, z^k - (\alpha_1 + h\lambda\beta_1)\, z^{k-1} - \ldots - (\alpha_{k-1} + h\lambda\beta_{k-1})\, z - (\alpha_k + h\lambda\beta_k) = 0.$$

Since (63) is degree k is has roots $z_1, \ldots, z_k$ that depend on the value of $\bar{h} = h\lambda$.

Defn: A linear multistep method is **time stable (absolutely stable/A-stable)** for a given value of $h\lambda$ if all roots of the stability polynomial have modulus strictly less than one.

Defn: The **time-stability region** in the complex $\bar{h}$-plane is the set of those values of $\bar{h} = h\lambda$ for which all roots of the stability polynomial have modulus less than one.

Remark: The boundary of the time-stability region can often be obtained explicitly by solving equation (63) for $\bar{h} = h\lambda$ and setting $z = \mathrm{e}^{\mathrm{i}\theta}$ where $0 \leq \theta < 2\pi$.

Theorem: The second Dahlquist barrier.

No A-stable linear multistep method of order $p > 2$ can exist.

This means there is a practical trade off between order of accuracy and stability.

7.12 Stiff ODEs

The investigation of time stability is particularly important for so-called **stiff equations** for which small step sizes are often needed.

Defn: It is not easy to give a precise definition of a **stiff ODE**. Generally, it refers to differential equations which contain a term which is rapidly varying such as, for example, a rapidly decaying exponential term. The descriptor "stiff" originated from the numerical solution of mass-spring-damper systems with a large damping stiffness (see HW sheet).

Example: Consider the IVP

$$(64) \qquad \frac{\mathrm{d}y}{\mathrm{d}t} = -100\,(y - \cos t) - \sin t, \qquad 0 < t, \qquad \text{with } y(0) = 1.$$

Exact solution: One can readily find (integrating factors, or by making the substitution $x(t) = y(t) - \cos t$ which leads to $x' = -100x$) the general solution to be $y(t) = \cos t + c\mathrm{e}^{-100t}$ and the initial condition $y(0) = 1$ requires $c = 0$.

So, finally, $y(t) = \cos t$ and this is **not** a suspicious-looking solution.

Numerical experiment: Let us solve (64) numerically using the following two methods:

(i) AB2: $y_{i+1} = y_i + \frac{1}{2}h(3f_i - f_{i-1})$

(ii) BD2: $y_{i+1} = \frac{4}{3}y_i - \frac{1}{3}y_{i-1} + \frac{2}{3}hf_{i+1}$

(although the second method is implicit there is no problem solving for y_{i+1} in linear ODEs.)

We apply both methods to compute the value of $y(1)$ whose exact values it $\cos(1) \approx 0.5403023$. BD2 behaves well for all h, but AB2 works only if h is small enough.

h	AB2	BD2
0.2	14.40	0.5404
0.1	-5.7×10^4	0.54033
0.05	-1.91×10^9	0.540309
0.02	-5.77×10^{10}	0.5403034
0.01	0.5403020	0.5403026
0.005	0.5403022	0.5403024

As already noted, the substitution $x(t) = y(t) - \cos t$ transforms the ODE into our canonical problem $x'(t) = \lambda x(t)$ with $\lambda = -100$. So let's consider time-stability of both methods with $\lambda = -100$.

(i) AB2: Remembering that $f = \lambda y$ AB2 is written

$$y_{i+1} = y_i + \frac{3h}{2}\lambda y_i - \frac{h}{2}\lambda y_{i-1}$$

and $\lambda = -100$ so

$$y_{i+1} = (1 - 150h)y_i + 50hy_{i-1}.$$

It follows that the stability polynomial is $z^2 - (1 - 150h)z - 50h = 0$, a quadratic equation with two solutions. The method is time stable for those values of h for which the two roots have modulus $|z| < 1$. The boundary of the stability region is most easily obtained by rearranging the stability polynomial for h in terms of z and thereafter by setting $z = 1$ and $z = -1$ (these are the real values of the boundary $z = \mathrm{e}^{i\theta}$ respectively)

$$h = \frac{z^2 - z}{50 - 150z} \qquad \Longrightarrow \qquad h = \begin{cases} 0 & \text{if} \quad z = 1, \\ 0.01 & \text{if} \quad z = -1. \end{cases}$$

A more detailed investigation shows that $|z_j| < 1$, $j = 1, 2$ when $0 < h < 0.01$ and this is when the method is time stable. This result tallies with the numerical results discussed previously.

(ii) BD2: Again with $f = \lambda y$, BD2 is

$$y_{i+1} = \frac{4}{3}y_i - \frac{1}{3}y_{i-1} + \frac{2}{3}h\lambda y_{i+1}$$

and with $\lambda = -100$

$$(3 + 200h)y_{i+1} - 4y_i + y_{i-1} = 0.$$

The stability polynomial is therefore $(3 + 200h)z^2 - 4z + 1 = 0$ and rearranging for h in terms of z gives

$$h = \frac{4/z - 1/z^2 - 3}{200} \qquad \Longrightarrow \qquad h = \begin{cases} 0 & \text{if} \quad z = 1, \\ -1/25 & \text{if} \quad z = -1. \end{cases}$$

This is harder to untangle. Since $h > 0$ there is no upper bound on h and a more detailed investigation shows that the two roots z_1, z_2 satisfy $|z_j| < 1$ for all $h > 0$. Therefore the method is stable for all h as the table above confirms.

Remark: In general, implicit methods work better for stiff problems.

8 Ordinary differential equations: boundary value problems (BVPs)

8.1 Introduction

Let us consider canonical problems of the type

$$y'' = f(x, y, y') \qquad a < x < b,$$

with

$$y(a) = \alpha, \qquad y(b) = \beta.$$

Remark 1: Instead of two conditions at the initial point $x = a$ we now place one condition at either end point. Since we cannot simply propagate initial information from one end point to the other, new techniques are required for solving such problems.

Remark 2: These types of equations often occur in physical problems, for example for describing a loaded beam, a hanging chain or a vibrating string. These physical problems are often position dependent rather than time dependent, and for this reason we denote the independent variable in this chapter by x instead of t.

Remark 3: We need to consider at least 2nd order ODEs since we have information to supply to the ODE from two boundaries and thus need at least two constants of integration to appear in the solution.

Remark 4: One approach is to use **Shooting Methods** in which one transforms the BVP into an IVP by initially guessing a second initial condition at $x = a$ propagating the solution to $x = b$ and then iterating the initial condition in order to satisfy the condition at $x = b$. The name 'shooting' refers to attempts to find the range of a projectile fired at a target by adjusting the initial speed and angle of the projectile. It's a fairly natural and unsophisticated method to implement.

8.2 Finite difference methods for linear problems

We consider specifically **linear** BVPs, still of the canonical 2nd order type

$$y'' = f(x, y, y') = p(x)y' + q(x)y + r(x)$$

with Dirichlet[21] boundary conditions

$$y(a) = \alpha, \qquad y(b) = \beta.$$

IDEA: The idea of the finite difference methods is to replace derivatives by finite difference approximations, as we did in earlier. The method is best described by example:

8.2.1 Example:

Consider the BVP

$$y'' = 0, \qquad 0 < x < 6,$$

with

$$y(0) = -1, \qquad y(6) = 5.$$

Exact solution: The general solution to the ODE is found by integrating twice to get $y(x) = Ax + B$. From the boundary conditions follows $B = -1$ and $A = 1$. Thus the exact solution is $y(x) = x - 1$.

Finite difference solution: First divide the interval $[0, 6]$ into n equal subintervals, each of length $h = 6/n$. **Mesh points** are defined as $x_i = ih$, $i = 0, 1, \ldots, n$ where $h = 6/n$.

The term $y''(x)$ in the ODE is approximated by the **central difference approximation** at each of the mesh points (see earlier in course) using

$$y''(x_i) = \frac{y(x_i + h) - 2y(x_i) + y(x_i - h)}{h^2} - \frac{h^2}{12} y^{(iv)}(\xi_i)$$

where $x_i - h < \xi_i < x_i + h$. Letting $y_i \approx y(x_i)$ represent the approximate value of the solution at the mesh points, we note that the equal spacing of the central difference approximation allows the above to be approximated (i.e. we neglect the error term) by

$$0 = y''(x_i) \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}, \qquad \text{for } 1 \le i \le n - 1.$$

[21] implying the value of function is specified

The ODE only holds inside the interval $0 < x < 6$ and not on the boundaries $x = 0$ and $x = 6$ where boundary conditions apply instead. In terms of our discrete representation of the solution on the mesh points we have

$$y_0 = -1, \quad y_n = 5$$

Thus, for $i = 1$ we write

$$y_2 - 2y_1 = -y_0 = 1$$

and for $i = 2, \ldots, n-2$,

$$y_{i+1} - 2y_i + y_{i-1} = 0,$$

whilst for $i = n-1$ we have

$$-2y_{n-1} + y_{n-2} = -y_n = -5$$

In total we have $n-1$ equations for $n-1$ unknowns, $y_1, y_2, \ldots y_{n-1}$ which we can arrange into an $(n-1) \times (n-1)$ matrix system

$$\begin{bmatrix} -2 & 1 & 0 & \ldots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & 1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \ldots & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -5 \end{bmatrix}$$

E.g.: Let $n = 6$ so that $h = 1$ and $y_i \approx y(i)$. Our matrix equation is 5×5 and given by

$$\begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -5 \end{bmatrix}$$

Solutions can be found by Gaussian elimination and are given by

$$y_0 = -1, \qquad y_1 = 0, \qquad y_2 = 1, \qquad y_3 = 2, \qquad y_4 = 3, \qquad y_5 = 4, \qquad y_6 = 5.$$

That is, they have the form $y_i = i - 1 = y(i)$. That is, they agree with the exact solution.

Remark: One can show that the solution is exact also for other values of n. Why should this be ? Because the exact solution is $y(x) = x - 1$ for which $y^{(iv)}(x) = 0$ and hence no error was introduced in making the central difference approximation.

8.2.2 General problems

Let us now consider return to the general problem

$$y'' = f(x, y, y') = p(x)y' + q(x)y + r(x), \qquad a < x < b$$

with

$$y(a) = \alpha, \qquad y(b) = \beta.$$

We define mesh points to be

$$x_i = a + ih, \qquad i = 0, 1, \ldots, n, \qquad \text{where } h = (b - a)/n.$$

First and second derivatives of y are both approximated using central difference approximations

$$y'(x_i) = \frac{y(x_i + h) - y(x_i - h)}{2h} + O(h^2)$$

$$y''(x_i) = \frac{y(x_i + h) - 2y(x_i) + y(x_i - h)}{h^2} + O(h^2)$$

both with errors of $O(h^2)$, the former proportional to $y'''(x)$ and the latter proportional to $y^{(iv)}(x)$.

We insert these approximations into the ODE using $y_i \approx y(x_i)$. and set $p(x_i) = p_i$, $q(x_i) = q_i$ and $r(x_i) = r_i$ which gives

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = p_i \frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i + r_i, \qquad 1 \leq i \leq n - 1$$

for interior points and

$$y_0 = \alpha, \qquad y_n = \beta$$

on the boundaries. We use these two values in the $i = 1$ and $i = n - 1$ equations to give

$$y_2 \left(1 - \frac{h}{2} p_1\right) - y_1 \left(2 + h^2 q_1\right) = h^2 r_1 - \alpha \left(1 + \frac{h}{2} p_1\right)$$

and

$$-y_{n-1} \left(2 + h^2 q_{n-1}\right) + y_{n-2} \left(1 + \frac{h}{2} p_{n-1}\right) = h^2 r_{n-1} - \beta \left(1 - \frac{h}{2} p_{n-1}\right)$$

respectively, whilst for $i = 2, \ldots, n-2$ we have

$$y_{i+1}\left(1 - \frac{h}{2}p_i\right) - y_i\left(2 + h^2 q_i\right) + y_{i-1}\left(1 + \frac{h}{2}p_i\right) = h^2 r_i$$

Making the abbreviations

$$a_i = -2 - h^2 q_i, \qquad b_i = 1 + \frac{h}{2}p_i, \qquad c_i = 1 - \frac{h}{2}p_i$$

gives us the $n-1$ equations

$$\begin{aligned} y_2 c_1 + y_1 a_1 &= h^2 r_1 - \alpha b_1 \\ y_{i+1} c_i + y_i a_i + y_{i-1} b_i &= h^2 r_i \qquad\qquad i = 2, 3, \ldots, n-2 \\ y_{n-1} a_{n-1} + y_{n-2} b_{n-1} &= h^2 r_{n-1} - \beta c_{n-1} \end{aligned}$$

which can be written in matrix form as

$$\begin{bmatrix} a_1 & c_1 & 0 & \cdots & \cdots & 0 \\ b_2 & a_2 & c_2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & b_{n-2} & a_{n-2} & c_{n-2} \\ 0 & \cdots & \cdots & 0 & b_{n-1} & a_{n-1} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} h^2 r_1 - \alpha b_1 \\ h^2 r_2 \\ \vdots \\ \vdots \\ h^2 r_{n-2} \\ h^2 r_{n-1} - \beta c_{n-1} \end{bmatrix}$$

This is a **tridiagonal system** that can be solved by Gaussian elimination. In fact there is an LU decomposition which is defined iteratively for such systems.

8.2.3 Different types of boundary condition

In the above we have assumed **Dirichlet**-type boundary conditions in which $y(a)$ and $y(b)$ are given. But one often has either **Neumann**-type conditions, or mixed Dirichlet-Neumann (so-called **Robin**) conditions or even periodic boundary conditions. In each case, one has to deal with representing $y'(a)$ and/or $y'(b)$.

E.g. Return to Example 8.2.1 and replace the BCs with $y(0) = -1$, $y'(6) = 0$. Since we no longer have $y_n = y(6) = -5$ to use in the $i = n - 1$ equation, we have to do something different. We use a backwards difference approximation

$$y'(6) \approx \frac{y_n - y_{n-1}}{h}$$

which allows us to write $y_n \approx y_{n-1} + hy'(6)$ and we can use this in the $i = n - 1$ equation.

THIS SECTION IS OMITTED FROM THE COURSE IN 2025

8.3 Spectral methods for linear problems

We consider again our canonical boundary value problem (BVP) consisting of the 2nd order linear ODE

$$y''(x) = p(x)y'(x) + q(x)y(x) + r(x), \qquad a < x < b,$$

with (Dirichlet) BCs

(65)
$$y(a) = \alpha, \qquad y(b) = \beta.$$

It helps to write the ODE compactly as

(66)
$$(\mathcal{L}y)(x) = r(x), \qquad a < x < b$$

by defining the differential operator

$$\mathcal{L} = \frac{\mathrm{d}^2}{\mathrm{d}x^2} - p(x)\frac{\mathrm{d}}{\mathrm{d}x} - q(x).$$

IDEA: Approximate the unknown function as

(67)
$$y(x) \approx \tilde{y}(x) = \sum_{n=1}^{N} c_n \phi_{n-1}(x),$$

by expanding in terms of a known truncated set of **basis functions** $\phi_n(x)$ and unknown expansion coefficients c_n.

The step (67) classifies the method as a **spectral method** since it is also assumed that the set $\{\phi_n(x), n = 0, 1, 2, \ldots\}$ form a complete[22] set in interval $[a, b]$.

For example, one might expand $y(x)$ in a Fourier series on the interval $[a, b]$ and this the outcome of separation of variables methods in the APDE course, but here we assume the functions $\phi_n(x)$ can be chosen much more generally.

As well as being complete it is also convenient that the functions $\phi_n(x)$ are **orthogonal** on the interval $[a, b]$ with respect to a given **weight function** $w(x)$. I.e.

$$\langle \phi_m, \phi_n \rangle \equiv \int_a^b w(x)\phi_m(x)\phi_n(x)\,\mathrm{d}x = 0 \qquad \text{if} \qquad m \neq n$$

using the **inner product** notation from earlier in the course.

In (67) we decide N and $\phi_n(x)$ (or $w(x)$) and must subsequently devise a method for determining c_n, $n = 0, 1, \ldots, N$.

We imagine that increasing N will produce more accurate approximations to $y(x)$ (and this can be shown formally to be the case). We also note that, in practice, the choice of $\phi_n(x)$ may be influenced by anticipation of certain features of the solution, but this is beyond our scope.

Using (67) in (65) gives two equations:

$$\sum_{n=1}^{N} c_n \phi_{n-1}(a) = \alpha, \qquad \sum_{n=1}^{N} c_n \phi_{n-1}(b) = \beta$$

A further $N-2$ equations are now required to ensure a total of N equations for the N unknowns $c_1, \ldots, c_N$. These come from the ODE and we first substitite (67) into (66) to get

$$\mathcal{L}\left(\sum_{n=1}^{N} c_n \phi_{n-1}(x)\right) = \sum_{n=1}^{N} c_n \mathcal{L}\phi_{n-1}(x) \approx r(x), \qquad a < x < b. \tag{68}$$

(the last relationship is approximate because $\tilde{y}(x)$ is not exact in general).

Next, we make the error $(\mathcal{L}\tilde{y})(x) - r(x)$ orthogonal to as many functions, $\phi_m(x)$, as we can. In other words, we take the inner product of both sides of equation (68) with $\phi_{m-1}(x)$ where

[22] complete means that the functions span the space

$m = 1, \ldots, N-2$ to give the $N-2$ equations

$$\sum_{n=1}^{N} c_n \langle \phi_{m-1}, \mathcal{L}\phi_{n-1} \rangle = \langle \phi_{m-1}, r \rangle, \qquad m = 1, \ldots, N-2.$$

Such an approach can be referred to **Galerkin's method**.

Taken together we obtain a linear algebra problem consisting of N equations for N unknowns. It can be written in matrix/vector form:

$$\begin{bmatrix} \phi_0(a) & \phi_1(a) & \ldots & \phi_{N-1}(a) \\ \phi_0(b) & \phi_1(b) & \ldots & \phi_{N-1}(b) \\ \langle \phi_0, \mathcal{L}\,\phi_0 \rangle & \langle \phi_0, \mathcal{L}\,\phi_1 \rangle & \ldots & \langle \phi_0, \mathcal{L}\,\phi_{N-1} \rangle \\ \langle \phi_1, \mathcal{L}\,\phi_0 \rangle & \langle \phi_1, \mathcal{L}\,\phi_1 \rangle & \ldots & \langle \phi_1, \mathcal{L}\,\phi_{N-1} \rangle \\ \vdots & \vdots & & \vdots \\ \langle \phi_{N-3}, \mathcal{L}\,\phi_0 \rangle & \langle \phi_{N-3}, \mathcal{L}\,\phi_1 \rangle & \ldots & \langle \phi_{N-3}, \mathcal{L}\,\phi_{N-1} \rangle \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N-2} \\ c_{N-1} \\ c_N \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \langle \phi_0, r \rangle \\ \langle \phi_1, r \rangle \\ \vdots \\ \langle \phi_{N-3}, r \rangle \end{bmatrix}.$$

This linear system of equations can be solved with, for example, Gaussian elimination.

Remarks: There are two main advantages of this method

(i) It delivers a continuous representation of the solution $y(x)$ (see equation (67)) that can be evaluated anywhere, not just at the mesh points.

(ii) One can show that the method has a fast (exponential) convergence to the exact solution as N increases.

8.3.1 Example

Consider the ODE

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

with

$$y(-1) = \alpha, \qquad y(+1) = \beta.$$

Then $\mathcal{L} = d^2/dx^2$, $r(x) = 0$ and $a, b = -1, 1$.

Exact solution: For this simple BVP, exact solution is easily found:

$$y(x) = \frac{1}{2}(\beta + \alpha) + \frac{1}{2}(\beta - \alpha)x$$

Numerical solution: We now use Galerkin's method to approximate the solution numerically and make the choice:

$$y(x) \approx \tilde{y}(x) = \sum_{n=1}^{4} c_n T_{n-1}(x)$$

I.e. $N = 3$ and $\phi_n(x) = T_n(x)$ and Chebyshev polynomials. They are defined on $[-1, 1]$ and are orthogonal w.r.t. the weight function $w(x) = 1/\sqrt{1-x^2}$.

Recall: The Chebyshev polynomials are explicitly given by

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

and satisfy

$$\langle T_m, T_n \rangle = \begin{cases} 0 & m \neq n \\ \pi/2 & m = n \neq 0 \\ \pi & m = n = 0. \end{cases}$$

The first four polynomials have the form

$$T_0(x) = 1, \qquad T_1(x) = x, \qquad T_2(x) = 2x^2 - 1, \qquad T_3(x) = 4x^3 - 3x.$$

At the end points they take the values $T_n(1) = \cos(n \cdot 0) = 1$ and $T_n(-1) = \cos(n\,\pi) = (-1)^n$.

Forming a system of equations:

The two BCs demand that

$$\sum_{n=1}^{4} c_n T_{n-1}(-1) = \alpha, \qquad \sum_{n=1}^{4} c_n T_{n-1}(1) = \beta$$

in other words

$$\sum_{n=1}^{4} c_n (-1)^n = \alpha, \qquad \sum_{n=1}^{4} c_n = \beta.$$

The ODE reduces to

$$\langle T_{m-1}, \mathcal{L}\tilde{y}\rangle = \langle T_{m-1}, 0\rangle$$

for $m = 1, 2$ (since this gives us 4 equations in total). In other words

$$\sum_{n=1}^{4} c_n \langle T_{m-1}, T''_{n-1}\rangle = 0, \qquad m = 1, 2.$$

Putting everything into a single matrix system gives

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ \langle T_0, T''_0\rangle & \langle T_0, T''_1\rangle & \langle T_0, T''_2\rangle & \langle T_0, T''_3\rangle \\ \langle T_1, T''_0\rangle & \langle T_1, T''_1\rangle & \langle T_1, T''_2\rangle & \langle T_1, T''_3\rangle \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ 0 \\ 0 \end{bmatrix}.$$

From the definitions $T''_0 \equiv 0$, $T''_1 \equiv 0$. We further find that

$$\langle T_0, T''_3\rangle = \langle T_1, T''_2\rangle = 0$$

because these inner products are integrals of odd functions from $-1 < x < 1$. Finally, we note from the definitions that

$$T''_2(x) = 4 \quad \text{so } \langle T_0, T''_2\rangle = 4\pi, \qquad T''_3(x) = 24x \quad \text{so } \langle T_1, T''_3\rangle = 12\pi.$$

Thus we obtain the following system

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 4\pi & 0 \\ 0 & 0 & 0 & 12\pi \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ 0 \\ 0 \end{bmatrix}.$$

We do not need Gaussian elimination to solve this matrix equation. From the last two lines we find $c_3 = c_4 = 0$. Then the top two lines give

$$c_1 - c_2 = \alpha, \qquad c_1 + c_2 = \beta$$

from which we deduce

$$c_1 = \frac{\alpha + \beta}{2}, \qquad c_2 = \frac{\beta - \alpha}{2}.$$

The resulting approximation is

$$\tilde{y}(x) = \frac{\alpha + \beta}{2} + \frac{\beta - \alpha}{2} x$$

and agrees with the exact solution !

Remark: If the interval $[a, b]$ does not coincide with the interval $[-1, 1]$ but we wish to use Chebyshev polynomials as our basis function then we can perform a change of variables of the form $x' = 1 + 2(x - b)/(b - a)$ to map $x \in [a, b]$ onto the interval $x' \in [-1, 1]$. The variable change must be applied to the BVP, including derivatives using the chain rule (i.e. $\mathrm{d}/\mathrm{d}x = (\mathrm{d}x'/\mathrm{d}x)\mathrm{d}/\mathrm{d}x'$ and $\mathrm{d}x'/\mathrm{d}x = 2/(b - a)$).

Appendix: code used for producing results

This is purely for interest. I've written out some Fortran 77 code which I used to produce numerical results.

Bisection method

```
      program bisection

      integer n,nmax

      real a,b,x,fa,fb,fx
      real fun

      print *,'enter a'
      read *,a
      print *,'enter b'
      read *,b
      print *,'enter nmax'
      read *,nmax

      fa = fun(a)
      fb = fun(b)

      if (fa*fb.ge.0.0) stop

      do 10 n=1,nmax

        x = 0.5*(a+b)
        fx = fun(x)

        print *,n,a,x,b,fa,fx,fb,(b-a)/2.0**n

        if (fx*fa.ge.0.0) then
```

```
          a = x
          fa = fx
        else
          b = x
          fb = fx
        end if

 10   end do
      end

      real function fun(x)

      real x

      fun = exp(x)-3.0*x

      return
      end
```

Fixed point iteration

```
      program fixedpt

      integer n,nmax

      real x,xe,err
      real fun

      print *,'enter x0'
      read *,x
      print *,'enter x*'
      read *,xe
      print *,'enter nmax'
      read *,nmax

      err = x-xe

      print *,0,x,e

      do 10 n=1,nmax

        x = fun(x)

        print *,n,x,x-xe,(x-xe)/err

        err = x-xe

 10   end do
      end

      real function fun(x)

      real x

      fun = cos(x)

      return
      end
```

Newton-Raphson method

```
      program newton

      integer n,nmax

      real x
      real fun,fund
      print *,'enter x0'
      read *,x
      print *,'enter nmax'
      read *,nmax

      print *,0,x

      do 10 n=1,nmax

        x = x-fun(x)/fund(x)

        print *,n,x

10    end do
      end

      real function fun(x)

      real x

      fun = x**2-x-1

      return
      end

      real function fund(x)

      real x

      fund = 2*x-1

      return
      end
```

Euler’s method

```
     program euler

     integer i,n

     print *,'Enter N'
     read *,n

     real t,y,ye

     y = 1.0
     ye = 1.0
     t = 0.0
     h = 1.0/real(n)

     print *,t,y,ye

     do 10 i=1,n

       y = y+h*(-y+t)
       t = t+h

       ye = 2.0*exp(-t)+t-1.0
       print *,t,y,ye

10   end do
     end
```