MATH3018 & MATH6141
Numerical Methods

Dr. Ian Hawke

Contents

Chapter 1

Error analysis

1.1 Introduction

Numerical analysis is mathematics brought down to earth. In the world of mathematics numbers are exact: π is the ratio of the circumference to its radius and it is known exactly, each single digit after the other. Numerical analysis teaches humbleness: it tells us that in the real world we cannot know numbers exactly, we can only know 2, 20, a million of the digits that form the number π, but in no way we can know any non-rational number exactly. This may not seem a serious problem: does it really matter that we cannot represent more than 10 of the first digits of π? The answer to this specific question is that generally it does not matter. However, if we were to rephrase the question as: "Does it matter that we cannot represent a number with infinite precision?" then the answer would have to be: "Yes, it does matter". Small approximations caused by the fact the we represent numbers using "only" a finite set of digits can be amplified in the process of solving a given mathematical problem up to a point where they make the entire result meaningless.

It is in this sense that numerical analysis teaches us humbleness: it forces us to realise our limitations in representing numbers and to ask ourselves the question of whether the results obtained by applying exact mathematical procedures to these inexact representations can be considered accurate.

This chapter is an introduction to numerical methods. Its aims are to highlight the consequences of finite precision representation of numbers on numerical calculations.

1.2 Floating point representation

Real numbers are represented in digital computers using a *normalised scientific notation* or *floating point notation*: the decimal point is shifted and appropriate powers of 10 are supplied so that all the digits are to the right of the decimal point and the first digit displayed is not zero. For example,

$$\begin{aligned} 8491.2913 &= 0.84912913 \times 10^4, \\ -12.32 &= -0.1232 \times 10^2, \\ 0.00321 &= 0.321 \times 10^{-2}. \end{aligned}$$

The general notation for a floating point number (in base 10) is

$$\pm\, 0.a_1a_2a_3a_4 \ldots a_m \times 10^c,$$

where $0.a_1a_2a_3a_4 \ldots a_m$ is the *mantissa*, the integer c is the *exponent* and the integer 10 is called the *base* of this representation. The number m of digits in the mantissa is a measure of the precision of the representation: the bigger m the more digits are stored in the computer and the more accurately the real number is approximated. The digits a_i are called “significant digits”: a_1 is the most significant digit and a_m is the least significant digit. The size of the mantissa is also referred to as “the number of significant digits”.

1.3 Absolute and relative errors

Before discussing how the errors introduced in the representations of numbers affect numerical calculations we must define ways of measuring the size of the error. There are two standard definitions. We define the *absolute error* in representing the number x with a finite precision representation $\bar{x}$ as the quantity

$$E_a = x - \bar{x}.$$

We define the *relative error* as the quantity

$$E_r = \frac{x - \bar{x}}{x}.$$

The absolute error is independent of the size of the number x, while the relative error is essentially a measure of the size of the error in comparison with the quantity measured.

Example

Suppose that $x = 1322$ and $\bar{x} = 1000$. The absolute and relative errors are respectively:

$$E_a = 1322 - 1000 = 322,$$
$$E_r = \frac{1322 - 1000}{1322} = 0.24.$$

Now suppose that $x = 1000322$ and $\bar{x} = 1000000$. The two errors are:

$$E_a = 1000322 - 1000000 = 322,$$
$$E_r = \frac{1000322 - 1000000}{1000322} = 0.0032 = 0.32 \times 10^{-2}.$$

The absolute error is the same in both cases, but the relative errors are completely different. The choice between the two errors is problem dependent.

One of the aims of numerical analysis is to control the growth of the errors that occur naturally while solving a given mathematical problem. In other words, it is the *total error from all sources* which is of most interest. Normally, we try, when examining a method, to prove a theorem of the form

If we apply method X to solve problem Y then $|x - \bar{x}| \leq \varepsilon \quad \forall x.$

The crux of numerical analysis is that some problems are such that even a small amount of rounding or initial error causes a large inaccuracy in the answer: these are known as *ill-conditioned problems*.

1.4 Errors and simple mathematical operations

Before studying algorithms to solve typical numerical problems, like finding the roots of a nonlinear equation, we must understand how numerical errors affect and are affected by simple operations.

1.4.1 Addition

The error in computing the addition of two real numbers by adding up two floating point numbers is given by

$$x + y = \overline{x} + \overline{y} + e(x) + e(y),$$

where $e(x)$ and $e(y)$ are respectively the errors in the floating point representation of x and y.

The addition of two floating point numbers loses most of the properties of the standard addition. In the examples that follow assume that the size of the mantissa is $m = 3$:

1. In general, the sum of two floating point numbers is not the floating point representation of the sum:

$$\overline{x+y} \neq \bar{x} + \bar{y}.$$

   For example,

$$x = 1.007,\ y = 2.005 \implies \bar{x} = 1.00,\ \bar{y} = 2.00,$$

   whilst

$$\overline{x+y} = \overline{3.012} = 3.01 \neq \bar{x} + \bar{y} = 3.00$$

2. The sum of two numbers may be too big to be represented in the computer, i.e. the result of the addition is a numerical overflow.

3. Order matters, i.e. the addition of floating point numbers is not associative:

$$\overline{\overline{(\overline{x} + \overline{y})} + \overline{z}} \neq \overline{\overline{x} + \overline{(\overline{y} + \overline{z})}}.$$

   As a matter of fact, it is always better to add in ascending order of magnitude.

$$\begin{cases} \overline{x} = 1.00, \\ \overline{y} = 0.007, \\ \overline{z} = 0.006, \end{cases} \implies \begin{cases} \overline{\overline{(\overline{x} + \overline{y})} + \overline{z}} = \overline{\overline{(1.00 + 0.007)} + 0.006} = \overline{1.00 + 0.006} = 1.00, \\ \overline{\overline{x} + \overline{(\overline{y} + \overline{z})}} = \overline{1.00 + \overline{(0.007 + 0.006)}} = \overline{1.00 + 0.013} = 1.01 \end{cases}$$

   The second result is more accurate than the first. If each small number is added separately to a big number then they may be all “chopped off”. However, if all the small numbers are added together their sum may be sufficiently big not to suffer such fate.

1.4.2 Subtraction

The error in computing the difference of two real numbers by subtracting two floating point numbers is given by

$$x - y = \overline{x} - \overline{y} + e(x) - e(y),$$

where $e(x)$ and $e(y)$ are respectively the errors in the floating point representation of x and y.

The subtraction of two floating point numbers is a very delicate operation that can lead to an outstanding loss of significant digits especially when the numbers to be subtracted are nearly equal. If, for example, $x = 42.345$ and $y = 42.287$ then we have:

$$x - y = 0.058,$$
$$\overline{x} - \overline{y} = 42.3 - 42.2 = 0.100$$

1.4.3 Multiplication

The error in computing the product of two real numbers by multiplying two floating point numbers is given by

$$x\,y = \overline{x}\,\overline{y} + \overline{y}e(x) + \overline{x}e(y) + e(x)e(y),$$

where $e(x)$ and $e(y)$ are respectively the errors in the floating point representation of x and y. The last term in this expression is usually negligible.

Note that on an m digit computer, $\overline{x}\,\overline{y}$ is at most $2m$ digit long. This is the main reason why most computers use $2m$ digits for calculations on m-digit numbers. The result is cut back to m digits once the calculation is completed.

1.4.4 Division

The error in computing the ratio of two real numbers by dividing two floating point numbers is given by

$$\frac{x}{y} = \frac{\overline{x} + e(x)}{\overline{y} + e(y)} = \frac{\overline{x}}{\overline{y}} + \frac{e(x)}{\overline{y}} - \frac{\overline{x}e(y)}{\overline{y}^2} + O[e(x)e(y)],$$

where $e(x)$ and $e(y)$ are respectively the errors in the floating point representation of x and y. The last term in this expression is usually negligible. Clearly the error increases dramatically if $y \simeq 0$.

Remark - The errors introduced by the basic arithmetical operations can be combined when studying more complicated operations. For example, one can show that in the dot product of two vectors the error is roughly $n\varepsilon$, where n is the size of the two vectors and ε is an upper bound on the error of the representation of each component of the vectors.

1.5 Stable and unstable computations

The procedure to solve numerically a mathematical problem involves a series of operations that, quite often, must be repeated over and over again. It may happen that a small error gets amplified in the course of the iteration procedure until it dominates the numerical solution and makes it totally unreliable.

A numerical process is *unstable* if small errors made at one stage of the process are magnified in subsequent stages and seriously degrade the accuracy of the overall solution.

n	x_n	Abs. err.	Rel. err.
0	+1.0000000e+00	+0.0000000e-01	+0.0000000e-01
1	+3.3333333e-01	+0.0000000e-01	+0.0000000e-01
2	+1.1111110e-01	+1.0000000e-08	+9.0000000e-08
3	+3.7036990e-02	+4.7000000e-08	+1.2690000e-06
4	+1.2345490e-02	+1.8900000e-07	+1.5309000e-05
5	+4.1144710e-03	+7.5530000e-07	+1.8353790e-04
6	+1.3687210e-03	+3.0211000e-06	+2.2023819e-03
7	+4.4516310e-04	+1.2084270e-05	+2.6428298e-02
8	+1.0407880e-04	+4.8336990e-05	+3.1713899e-01
9	-1.4254266e-04	+1.9334792e-04	+3.8056671e+00
10	-7.5645659e-04	+7.7339168e-04	+4.5668005e+01
11	-3.0879216e-03	+3.0935666e-03	+5.4801604e+02
12	-1.2372384e-02	+1.2374266e-02	+6.5761923e+03
13	-4.9496435e-02	+4.9497062e-02	+7.8914304e+04
14	-1.9798804e-01	+1.9798825e-01	+9.4697166e+05
15	-7.9195293e-01	+7.9195300e-01	+1.1363660e+07

Table 1.1: *Iterations of the unstable map (1.1) with initial conditions $x_0 = 1$ and $x_1 = 1/3$. The correct value of the n-th iterate is $x_n = (1/3)^n$.*

Consider for example the sequence of real numbers defined by

$$\begin{cases} x_0 = 1, \\ x_1 = \dfrac{1}{3}, \\ x_{n+1} = \dfrac{13}{3}x_n - \dfrac{4}{3}x_{n-1}, \quad n \geq 1. \end{cases} \tag{1.1}$$

One can show by induction that this recurrence relation generates the sequence

$$x_n = \left(\frac{1}{3}\right)^n.$$

However, if we compute (1.1) numerically using 8 significant digits we obtain the results shown in Table 1.1. The absolute and relative error increase with the iterations until after the seventh iteration the result is completely unreliable. The algorithm is therefore *unstable*.

The instability of this algorithm is caused by the fact that an error present in x_n is multiplied by 13/3 in computing x_{n+1} (the proof of instability is more

n	x_n	Abs. err.	Rel. err.
0	+1.0000000e+00	+0.0000000e-01	+0.0000000e-01
1	+4.0000000e+00	+0.0000000e-01	+0.0000000e-01
2	+1.6000000e+01	+0.0000000e-01	+0.0000000e-01
3	+6.4000000e+01	+0.0000000e-01	+0.0000000e-01
4	+2.5600000e+02	+0.0000000e-01	+0.0000000e-01
$\vdots$	$\vdots$	$\vdots$	$\vdots$
15	+1.0737418e+09	+0.0000000e-01	+0.0000000e-01
16	+4.2949672e+09	+1.0000000e+02	+2.3283064e-08
17	+1.7179868e+10	+1.0000000e+03	+5.8207661e-08
18	+6.8719471e+10	+6.0000000e+03	+8.7311490e-08
19	+2.7487788e+11	+3.0000000e+04	+1.0913936e-07
20	+1.0995115e+12	+1.0000000e+05	+9.0949470e-08

Table 1.2: *Iterations of the unstable map (1.1) with initial conditions $x_0 = 1$ and $x_1 = 4$. The correct value of the n-th iterate is $x_n = 4^n$.*

complicated than this, but this statement is essentially correct). Hence there is a possibility that the error in x_1 will be propagated into x_{15}, for example, with a factor $(13/3)^{14}$. Since the absolute error in x_1 is approximately 10^{-8} and since $(13/4)^{14}$ is roughly 10^9, the error in x_{15} due solely to the error in x_1 could be as much as 10.

Whether a process is numerically stable or unstable should be decided on the basis of *relative* errors. Thus if there are large errors in the computations, that situation maybe quite acceptable if the answers are large. In the preceding example, let us start with initial values $x_0 = 1$ and $x_1 = 4$. The recurrence relation (1.1) is unchanged and therefore errors will still be propagated and magnified as before. But the correct solution is now $x_n = 4^n$ and the results of the computation are correct to seven significant figures (see Table 1.2). In this case the correct values are large enough to overwhelm the errors. The absolute errors are undoubtedly large (as before), but they are relatively negligible.

Remark - The errors we have discussed until now are independent of the mathematical problem we wish to solve and are inherent to the numerical procedure that we are using. However, there are mathematical problems that are intrinsically "difficult", in the sense that a small alteration of one of the parameters of the problem (for example, one of the coefficients of a system of linear equations) alters dramatically the value of the solution. Such problems are called *ill condi-*

tioned and we will discuss them as we encounter them.

Chapter 2

Linear equations

2.1 Introduction

The numerical solution of systems of linear equations can be a rather taxing problem. Solving a system like

$$\begin{cases} 2x + 3y = 7, \\ \quad x - y = 1, \end{cases}$$

is easy. Subtract twice the second equation from the first and obtain $y = 1$ and $x = 2$. This procedure is fine for a small system of linear equations. However, there are many numerical problems that require the solution of linear systems with many equations. For example, when integrating numerically a Partial Differential Equation it is typical to have to solve a 10000×10000 system of linear equations.

There are other, less obvious, snags in solving a linear system of equations. Consider the linear system

$$\begin{cases} \quad x + y = 1, \\ 2x + 2y = 3. \end{cases}$$

The second equation is incompatible with the first one and, therefore, the system has no solution. However, if the coefficients stored in the computer are ever so slightly altered, for example to

$$\begin{cases} 0.999999\,x + y = 0.999999, \\ \qquad 2x + 2y = 2.9999999, \end{cases}$$

then the system will have a solution. This is an example of an ill-conditioned system: a very small change in the matrix of the coefficients induces an enormous change in the solutions (from non-existence to existence in this case).

From these two examples, we can see the problems that have to be solved in devising algorithms for solving systems of linear equations. The algorithms should be fast, so that huge systems of equations can be solved in a reasonable length of time and they should be accurate, i.e. they should not introduce approximation errors that grow out of bounds. Moreover, if a linear system cannot be solved accurately (second example), we must be able to find functions of the matrix coefficients that can be used as an health warning against putting too much trust in the solutions.

If we want to know how close the numerical solution is to the correct solution we must first define the concept of "length" and "distance" for vectors and matrices. Once this is done we will be able to find appropriate functions of the matrix coefficients that we can use to check the accuracy of the numerical results. Only after this ground work has been carried out we will be able to discuss how to solve a linear system of equations.

2.2 Some definitions

2.2.1 Introduction

Unless otherwise stated, we plan to solve

$$A\boldsymbol{x} = \boldsymbol{b},$$

where A is a $n \times n$ real matrix, $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ is the vector of unknowns and $\boldsymbol{b} = (b_1, \ldots, b_n)^T$ is a given real vector. Moreover, we indicate with I the identity matrix and with 0 the zero matrix.

The basic theory of systems of linear equations states that the equations $A\boldsymbol{x} = \boldsymbol{b}$ have a unique solution if and only if $\det(A) \neq 0$, that solution being $\boldsymbol{x} = A^{-1}\boldsymbol{b}$. For the case $\det(A) = 0$, the equations either have no solutions (inconsistent set) or an infinite number of solutions (undetermined set).

Remark - Numerically, finding $\boldsymbol{x}$ using $\boldsymbol{x} = A^{-1}\boldsymbol{b}$ is invariably bad: there are faster and more accurate methods to solve a linear system of equations.

2.2.2 Vector and matrix norms

Vector norms

Let $\mathbb{R}^n$ be the set of all n-dimensional vectors with real components. A *norm* on $\mathbb{R}^n$ is a function $\|\cdot\|$ which assigns a real value to each vector in $\mathbb{R}^n$ (the "length" of the vector) and satisfies

1. $\|\boldsymbol{x}\| \geq 0 \ \forall \boldsymbol{x} \in \mathbb{R}^n$, with $\|\boldsymbol{x}\| = 0$ if and only if $\boldsymbol{x} = 0$.

2. $\|\alpha \boldsymbol{x}\| = |\alpha|\|\boldsymbol{x}\| \ \forall \boldsymbol{x} \in \mathbb{R}^n, \ \forall \alpha \in \mathbb{R}$.

3. $\|\boldsymbol{x}+\boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|, \ \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

Some common norms are:

$$\|\boldsymbol{x}\|_1 \equiv \sum_{j=1}^{n} |x_j| \,, \qquad \|\boldsymbol{x}\|_2 \equiv \left[\sum_{j=1}^{n} (x_j)^2\right]^{1/2},$$

$$\|\boldsymbol{x}\|_p \equiv \left[\sum_{j=1}^{n} (x_j)^p\right]^{1/p} \quad p > 0, \qquad \|\boldsymbol{x}\|_\infty = \max_j |x_j| \,.$$

Remark - $\|\cdot\|_2$ is called the *Euclidean norm*: it is the generalisation to n-dimensional spaces of the distance between two points in the coordinate plane.

Matrix norms

Let M_n denote the set of all real $n \times n$ matrices. A *norm* on M_n is a function $\|\cdot\|$ which assigns a real value to each matrix in M_n and that satisfies

1. $\|A\| \geq 0 \ \ \forall A \in M_n$. Moreover $\|A\| = 0$ if and only if $A = 0$.

2. $\|\alpha A\| = |\alpha|\|A\| \ \ \forall A \in M_n \ \ \forall \alpha \in \mathbb{R}$.

3. $\|A+B\| \leq \|A\| + \|B\| \ \ \forall A, B \in M_n$.

4. $\|AB\| \leq \|A\|\|B\|$

Vector norms can be used to define a family of matrix norms, called *induced norms*: these are the only ones used in this unit. The norm of a vector $\|\boldsymbol{x}\|$ measures its length; therefore, $\|A\boldsymbol{x}\|$ is the length of the vector $\boldsymbol{x}$ transformed by the matrix A. We can define a matrix norm as the maximum relative "stretching" induced by the matrix A. More formally, if $\|\cdot\|$ is a norm defined in $\mathbb{R}^n$ then we a define a function (norm) $\|\cdot\|$ on the matrix space M_n as

$$\|A\| \equiv \max_{\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\|.$$

These norms are said to be *induced* by the corresponding vector norm and we use the same symbol for the matrix and the vector norm. For example,

$$\|A\|_1 \equiv \max_{\|\boldsymbol{x}\|_1=1} \|A\boldsymbol{x}\|_1,$$

$$\|A\|_p \equiv \max_{\|\boldsymbol{x}\|_p=1} \|A\boldsymbol{x}\|_p,$$

$$\|A\|_\infty \equiv \max_{\|\boldsymbol{x}\|_\infty=1} \|A\boldsymbol{x}\|_\infty.$$

Remark 1 - The vector and matrix norms are compatible:

$$\|A\boldsymbol{x}\|_p \leq \|A\|_p \|\boldsymbol{x}\|_p.$$

However, we cannot mix norms. For example, it is **not** true in general that $\|A\boldsymbol{x}\|_1 \leq \|A\|_2 \|\boldsymbol{x}\|_2$.
Remark 2 - The value of the matrix 1-norm is the maximum of the 1-norms of the column vectors of the matrix:

$$\|A\|_1 = \max_{1\leq k\leq n} \sum_{j=1}^{n} |a_{jk}|.$$

Remark 3 - The value of the matrix infinity-norm is the maximum of the 1-norms of the row vectors of the matrix:

$$\|A\|_\infty = \max_{1\leq k\leq n} \sum_{j=1}^{n} |a_{kj}|.$$

2.2.3 The condition number

We now have the tools to assess the reliability of a numerical solution of a system of linear equations. Given a system $A\boldsymbol{x} = \boldsymbol{b}$ call $\boldsymbol{x}_c$ the computed solution and $\boldsymbol{x}_t$ the true solution. Define an additional vector $\boldsymbol{r}$ (*residual*) as,

$$\boldsymbol{r} = A\boldsymbol{x}_c - \boldsymbol{b}.$$

Any numerical method that attempts to solve the linear system of equations tries to minimise the vector $\boldsymbol{r}$: if $\boldsymbol{r} = 0$ the problem is solved exactly. We can rewrite $\boldsymbol{r}$ as

$$\boldsymbol{r} = A\boldsymbol{x}_c - A\boldsymbol{x}_t \implies \boldsymbol{x}_c - \boldsymbol{x}_t = A^{-1}\boldsymbol{r}.$$

We can see from this expression that if A^{-1} is "ill behaved" then even though $\boldsymbol{r}$ is very small the difference between the computed and the true solution can be very large. The *condition number of the matrix*, $K(A)$, is the quantity

$$K(A) = \|A\| \left\|A^{-1}\right\|,$$

and we call $\mathcal{E}$ the *weighted residual*,

$$\mathcal{E} = \frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|} = \frac{\|A\boldsymbol{x}_c - \boldsymbol{b}\|}{\|\boldsymbol{b}\|}.$$

One can show that

$$\frac{1}{K(A)}\mathcal{E} \leq \frac{\|\boldsymbol{x}_c - \boldsymbol{x}_t\|}{\|\boldsymbol{x}_t\|} \leq K(A)\mathcal{E}. \tag{2.1}$$

The condition number is always greater or equal to one, $K(A) \geq 1$. If $K(A) \simeq 1$ then the relative error is of the same order of the weighted residual: if the numerical method we have used has converged to a solution with small weighted residual we can be confident of the accuracy of the solution. However, if the condition number is big, $K(A) \gg 1$, then, even though the weighted residual may be very small, the relative error in the solution may be extremely large.

2.2.4 Ill-conditioned systems

The condition number gives us a warning that the solution may not be accurate. We must discuss a little bit more at length what this implies. Consider the linear system

$$\begin{cases} x + 2y = 3, \\ 0.499x + 1.001y = 1.5. \end{cases} \tag{2.2}$$

The solution is $x = y = 1$. Consider now a system with the same first equation and a slightly different second equation:

$$\begin{cases} x + 2y = 3, \\ 0.5x + 1.001y = 1.5. \end{cases} \tag{2.3}$$

The solution of this second system is $x = 3$ and $y = 0$. In other words, an extremely small change in one of the coefficients has produced an enormous change in the solution.

We can analyse this result in terms of the condition number. The matrix of the coefficients and its inverse are

$$A = \begin{pmatrix} 1 & 2 \\ \frac{499}{1000} & \frac{1001}{1000} \end{pmatrix} \qquad A^{-1} = \begin{pmatrix} \frac{1001}{3} & \frac{-2000}{3} \\ \frac{-499}{3} & \frac{1000}{3} \end{pmatrix}$$

with infinity norms (maximum of the 1-norm of the rows)

$$\|A\|_\infty = 3, \quad \left\|A^{-1}\right\|_\infty = \frac{3001}{3} \implies K(A) = 3001.$$

The condition number is big, as expected. To summarise this example, the following statements are roughly equivalent:

- The linear system is such that a small change in specification of the problem can lead to a large change in relative error;
- The condition number is large, $K(A) \gg 1$;
- The system is ill conditioned;
- As for all ill-conditioned problems, the relative error may be disastrous even if the residual is very small.

Being or not being ill-conditioned is a property of the linear system under study and cannot be eliminated by some cunning numerical algorithm. The best we can hope for is to have some warning that things may go horribly wrong: this is the purpose of the condition number.

While there are techniques based on the Singular Value Decomposition that try to extract as much information as possible from an ill-conditioned system, they are well beyond the scope of this unit and from now on we will assume that the linear system we intend to solve is perfectly well behaved, i.e. has a condition number of the order of unity.

2.3 Direct methods

2.3.1 Introduction

It is not possible to dwell on all the techniques that have been developed to solve linear systems of equations. The most recent methods can be very involved and their implementation may be considered skilled craftsmanship. Instead, we introduce the simplest examples of the two main categories of methods to solve linear systems. While it is true that no present day linear system solver uses these methods as we describe them, it is also true that most of the present day techniques are deeply rooted in these methods.

There are two big families of algorithms for solving linear systems: the first, the *direct methods*, consist of a finite list of transformations of the original matrix of the coefficients that reduce the linear systems to one that is easily solved. The second family, the *indirect or iterative methods*, consists of algorithms that specify a series of steps that lead closer and closer to the solution without, however, ever exactly reaching it. This may not seem a very desirable feature until we remember that we cannot in any case represent an exact solution: most iterative methods provide us with a highly accurate solution in relatively few iterations.

2.3.2 Gaussian elimination

The idea behind Gaussian elimination (and all direct methods) is that some linear systems of equations are very easy to solve. Suppose for example, that we wish to solve the problem $A\boldsymbol{x} = \boldsymbol{b}$ where A is an upper triangular matrix, i.e. $a_{ij} = 0$ if $j < i$. This equation can be solved by simple back-substitution. Consider, for example, the system

$$A\boldsymbol{x} = \begin{pmatrix} 3 & 2 & 1 \\ 0 & 5 & 4 \\ 0 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \boldsymbol{b}.$$

The last equation is trivial and we can easily obtain x_3. Once x_3 is known it can be substituted in the second equation (back-substitution) and so on:

$$\begin{aligned}
6x_3 = 3 \quad &\Longrightarrow \quad x_3 = \frac{1}{2}, \\
5x_2 + 4x_3 = 2 \quad &\Longrightarrow \quad 5x_2 + 2 = 2 \\
&\Longrightarrow \quad x_2 = 0, \\
3x_1 + 2x_2 + x_3 = 1 \quad &\Longrightarrow \quad 3x_1 + \frac{1}{2} = 1 \\
&\Longrightarrow \quad x_1 = \frac{1}{6}.
\end{aligned}$$

The Gauss elimination algorithm consists of a series of steps that transform a generic $n \times n$ matrix A with elements a_{ij} into an upper triangular matrix so that the ensuing linear system can be solved by back substitution. The algorithm is as follows:

1. Replace the j-th equation with

$$-\frac{a_{j1}}{a_{11}} \times (1^{\text{st}}\text{equation}) + (j\text{-th equation}),$$

   where j is an index that runs from 2 to n. In implementing this algorithm by hand it is convenient to write A and $\boldsymbol{b}$ as a single $n \times (n+1)$ matrix called the *augmented matrix*. Consider for example

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 10 \\ 11 \\ 12 \end{pmatrix},$$

   and write them as

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 4 & 5 & 6 & 11 \\ 7 & 8 & 0 & 12 \end{array}\right).$$

By applying the first step of the algorithm to this matrix we obtain

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 0 & 5-4\times 2 & 6-4\times 3 & 11-4\times 10 \\ 0 & 8-7\times 2 & 0-7\times 3 & 12-7\times 10 \end{array}\right) = \left(\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 0 & -3 & -6 & -29 \\ 0 & -6 & -21 & -58 \end{array}\right)$$

Remark - If $a_{11} = 0$ swap the first row with one whose first element is non zero.

2. Repeat the previous step, but starting with the next row down and with j greater than the row number. If the current row is row k then we must replace row j, $j > k$, with

$$-\frac{a_{jk}}{a_{kk}} \times (k^{\text{th}} \text{ equation}) + (j^{\text{th}} \text{ equation}),$$

where j is an index, $k < j \leq n$. The coefficient a_{kk} is called the *pivot*. In the case of our example system we have:

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 0 & -3 & -6 & -29 \\ 0 & -6-2(-3) & -21-2(-6) & -58-2(-29) \end{array}\right) = \left(\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 0 & -3 & -6 & -29 \\ 0 & 0 & -9 & 0 \end{array}\right).$$

3. When all the rows have been reduced, the matrix has been transformed in an upper triangular matrix and the linear system can be solved by back-substitution.

Remark - When designing an algorithm it is important to keep track of the number of operations required to solve the problem. For example, an algorithm which involves a number of operations that increases exponentially with the problem size becomes very quickly useless (such problems are called non-polynomial and are some of toughest nuts to crack in numerical analysis. The most famous example is the travelling salesman problem). The Gauss elimination algorithm is an n^3 algorithm, i.e. it requires a number of floating point operations that grows with the cube of the problem size.

2.3.3 Pivoting strategies

The Gauss elimination method just outlined suffers from poor accuracy if the matrix coefficients are of very different size. Consider, for example, the system

$$\begin{cases} 10^{-5}x_1 + x_2 & = 1, \\ x_1 + x_2 & = 2. \end{cases}$$

The 1-norm of the matrix of coefficients, A, is $\|A\|_1 = 2$. The inverse of A is

$$A^{-1} = \frac{1}{10^{-5}-1}\begin{pmatrix} 1 & -1 \\ -1 & 10^{-5} \end{pmatrix},$$

and has 1-norm $\|A^{-1}\|_1 = 2/(1-10^{-5}) \simeq 2$ so that the matrix condition number is a most benign $K(A) \simeq 4$. The problem is not ill-conditioned and we should not expect any problem in finding an accurate solution.

We can solve the problem exactly using Gauss elimination. By subtracting 10^5 times the first equation from the second we obtain the following augmented matrix and solution:

$$\left(\begin{array}{cc|c} 10^{-5} & 1 & 1 \\ 0 & -99999 & -99998 \end{array}\right) \Longrightarrow \begin{array}{lll} x_2 = \dfrac{-99998}{-99999} & = 0.9999899\ldots & \simeq 1, \\ x_1 = \dfrac{10^5}{99999} & = 1.00001000\ldots & \simeq 1. \end{array}$$

If, however, we now solve the same problem using floating point notation and only four significant figures, i.e. $m = 4$, we obtain a rather different result. The augmented matrix is now

$$\left(\begin{array}{cc|c} 0.1000\times10^{-4} & 0.1000\times10 & 0.1000\times10 \\ 0.1000\times10 & 0.1000\times10 & 0.2000\times10 \end{array}\right) \Longrightarrow \tag{2.4}$$

$$\left(\begin{array}{cc|c} 0.1000\times10^{-4} & 0.1000\times10 & 0.1000\times10 \\ 0.0000\times10^{0} & -0.1000\times10^{6} & -0.1000\times10^{6} \end{array}\right) \Longrightarrow \begin{cases} x_2 & = 1, \\ x_1 & = 0. \end{cases} \tag{2.5}$$

This is an example of *poor scaling*: whenever the coefficients of a linear system are of greatly varying sizes, we may expect that rounding errors may build up due to loss of significant figures. While ill-conditioning is an incurable “illness” because it is intrinsic to the matrix of coefficients, poor scaling is related to the algorithm and can be eliminated by suitably modifying the procedure to solve the linear system. In the case of this example, everything works fine if we first exchange the two rows so that the first row is the one with the biggest coefficient. In this case, the second row is transformed during Gaussian elimination by subtracting from it the first row multiplied by 10^{-5} and no rounding errors occur:

$$\left(\begin{array}{cc|c} 0.1000\times10 & 0.1000\times10 & 0.2000\times10 \\ 0.1000\times10^{-4} & 0.1000\times10 & 0.1000\times10 \end{array}\right) \Longrightarrow \tag{2.6}$$

$$\left(\begin{array}{cc|c} 0.1000\times10 & 0.1000\times10 & 0.2000\times10 \\ 0.0000\times10^{0} & 0.1000\times10 & 0.1000\times10 \end{array}\right) \Longrightarrow \begin{cases} x_2 & = 1, \\ x_1 & = 1. \end{cases} \tag{2.7}$$

The solutions are very close to the exact solution as we would have expected from the analysis of the condition number.

The procedure of exchanging rows (or columns) in order to have the biggest pivot (i.e. the first coefficient of the row that is to be subtracted from all the subsequent rows) is called *pivoting*. From the above example we can conclude that, in the k-th step of the Gauss elimination algorithm, we want to replace row j with

$$\mathrm{Row}_j \to \mathrm{Row}_j - \varepsilon\,\mathrm{Row}_k,$$

where ε is a small number, ideally as small as possible. This can be easily arranged: at each step of the algorithm we rearrange the set of equations so that we are eliminating using the largest pivot in modulus.

Notice that the factors that multiply the rows are always smaller than unity. This procedure is called *partial pivoting*.

An algorithm that is similar to the one above, but that exchanges both rows and columns in order that the biggest element in the matrix is the pivot is called *total pivoting*. However, total pivoting involves moving around huge chunks of computer memory and also doing an exhaustive search for the biggest matrix element. This practical problems are such that Gaussian elimination with partial pivoting is preferred.

2.3.4 Decomposition methods

Introduction

Decomposition methods attempt to rewrite the matrix of coefficients as the product of two matrices. The advantage of this approach is that the solution of a linear system of equations is split into smaller and considerably easier tasks. There are many decomposition techniques, but we are going to discuss only the simplest scheme, the LU decomposition, in order to understand the principles behind this approach to finding the solution of a linear system of equations.

Suppose that we are able to write the matrix A of coefficients of the linear system $A\boldsymbol{x} = \boldsymbol{b}$ as

$$A = LU,$$

where L is lower triangular and U is upper triangular. The solution of the linear system of equations becomes straightforward. We can write

$$A\boldsymbol{x} = \boldsymbol{b} \implies (L\,U)\,\boldsymbol{x} = \boldsymbol{b} \implies L\,(U\boldsymbol{x}) = \boldsymbol{b}.$$

If we call $\boldsymbol{y} = U\boldsymbol{x}$, then we have transformed the original linear system in two systems,

$$L\boldsymbol{y} = \boldsymbol{b}, \tag{2.8}$$

$$U\boldsymbol{x} = \boldsymbol{y}. \tag{2.9}$$

Each of these systems is triangular. The system (2.8) is lower triangular and can be solved by forward substitution while the system (2.9) is upper triangular and can be solved by back substitution. Therefore, even though we must solve two linear systems instead of only one, each of the two is very easy to solve. This is an advantage with respect to Gauss elimination especially if we have to solve $A\boldsymbol{x} = \boldsymbol{b}$ for many different values of the vector $\boldsymbol{b}$: we need to factorise the matrix A only once and then use the factorisation to find the solution vector $\boldsymbol{x}$ for all the different values of the known vectors $\boldsymbol{b}$.

There are two questions that we must now answer: "How can we factorise a matrix?" and "Is the factorisation possible?" We will answer the first question first and then discuss at more length the answer to the second.

Factorisation of a matrix

A first thing to note is that the LU factorisation as described above is not unique. A is a $n \times n$ matrix and has therefore n^2 coefficients. L and U are both triangular and thus have $n(n+1)/2$ entries each for a total of $n^2 + n$ entries. In other words, L and U have together n coefficients more than the original matrix A. Therefore we can choose n coefficients of L and U to our own liking.

To derive a formula for the LU factorisation we start by writing explicitly in terms of the components the decomposition $A = LU$. Call a_{ij}, ℓ_{ij} and u_{ij} the elements respectively of A, L and U. The following relation must hold:

$$a_{ij} = \sum_{s=1}^{n} \ell_{is} u_{sj} = \sum_{s=1}^{\min(i,j)} \ell_{is} u_{sj}, \tag{2.10}$$

where we have used the fact that $\ell_{is} = 0$ for $s > i$ and $u_{sj} = 0$ for $s > j$.

Let us start with $i = j = 1$. Equation (2.10) reduces to

$$a_{11} = \ell_{11} u_{11}.$$

We can now make use of the freedom of choosing n coefficients of L and U. The most common choices are:

- *Doolittle's factorisation* - Set $\ell_{ii} = 1$, i.e. the matrix L is *unit* lower triangular.
- *Crout's factorisation* - Set $u_{ii} = 1$, i.e. the matrix U is *unit* upper triangular.

Following the Doolittle's factorisation we set $\ell_{11} = 1$ and obtain

$$u_{11} = a_{11}.$$

```
for $k = 1, 2, \ldots, n$ do
    Specify a nonzero value for either $\ell_{kk}$ (Doolittle)
    or $u_{kk}$ (Crout) and compute the other from
        $$\ell_{kk} u_{kk} = a_{kk} - \sum_{s=1}^{k-1} \ell_{ks} u_{sk}.$$
    Build the $k$-th row of $U$:
    for $j = k+1, k+2, \ldots, n$ do
        $$u_{kj} = \left( a_{kj} - \sum_{s=1}^{k-1} \ell_{ks} u_{sj} \right) / \ell_{kk}$$
    end
    Build the $k$-th column of $L$:
    for $i = k+1, k+2, \ldots, n$ do
        $$\ell_{ik} = \left( a_{ik} - \sum_{s=1}^{k-1} \ell_{is} u_{sk} \right) / u_{kk}$$
    end
end
```

Table 2.1: *Algorithm for the Doolittle and Crout factorisation methods.*

We can now compute all the elements of the first row of U and of the first column of L, by setting either $i = 1$ or $j = 1$:

$$u_{1j} = a_{1j}, \qquad i = 1,\, j > 1 \tag{2.11}$$

$$\ell_{i1} = a_{i1}/u_{11}, \qquad j = 1,\, i > 1. \tag{2.12}$$

Consider now a_{22}. Equation (2.10) becomes:

$$a_{22} = \ell_{21}u_{12} + \ell_{22}u_{22}.$$

If, according to the Doolittle's scheme, we set $\ell_{22} = 1$ we have:

$$u_{22} = a_{22} - \ell_{21}u_{12},$$

where ℓ_{21} and u_{12} are known from the previous steps. We can now compute the second row of U and the second column of L by setting either $i = 2$ or $j = 2$:

$$u_{2j} = a_{2j} - \ell_{21}u_{1j}, \qquad i = 2,\, j > 2 \tag{2.13}$$

$$\ell_{i2} = \left(a_{i2} - \ell_{i1}u_{12}\right)/u_{22} \qquad j = 2,\, i > 2. \tag{2.14}$$

This procedure can be repeated for all the rows and columns of U and L. The compact version of the algorithm for LU decomposition using either Doolittle's or Crout's method is listed in Table 2.1.

Remark 1 - Notice that for the algorithm to work ℓ_{kk} and u_{kk} must be different from zero. However, this should not be interpreted as a condition for the matrix to have an LU decomposition. The following matrix,

$$\begin{pmatrix} 2 & 1 & -2 \\ 4 & 2 & -1 \\ 6 & 3 & 11 \end{pmatrix}$$

has $u_{22} = 0$, but it can be written as the product of

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 2 & 1 & -2 \\ 0 & 0 & 3 \\ 0 & 0 & 17 \end{pmatrix}.$$

Remark 2 - The LU decomposition with L unit triangular and the Gauss elimination algorithm are closely related. The matrix U is the matrix of the coefficients reduced to triangular form using Gaussian elimination. The matrix L can be obtained by writing each multiplier in the location corresponding to the zero entry in the matrix it was responsible for creating. Therefore one can use Gaussian elimination to find the LU decomposition of a matrix. The advantage of LU decomposition over Gaussian elimination is that if we have to solve many linear systems $A\boldsymbol{x} = \boldsymbol{b}$ for many different values of $\boldsymbol{b}$, we need to do only one LU decomposition.

Remark 3 - Pivoting is essential for the accuracy of this algorithm.

Conditions for factorisation

We must now specify under what conditions a matrix can be decomposed as the product of two triangular matrices. What follows is a list of sufficient conditions for the decomposition to be possible. We start with a theorem that involves the minors of A.

Theorem 2.3.1 *If all $n-1$ leading principal minors*[1] *of the $n \times n$ matrix A are nonsingular, then A has an LU decomposition. If the LU decomposition exists and A is non singular, then the LU decomposition is unique and* $\det(A) = u_{11}u_{22}\dots u_{nn}$ *(assuming that the matrix has been decomposed according the the Doolittle method).*

Proof: Golub & Van Loan, *Matrix computations*, page 97 (Third edition, 1996)

Remark 1 - This theorem is a consequence of the relation between Gaussian elimination and LU decomposition. It is possible to decompose the matrix in a lower and a upper triangular form if all the pivots in the Gaussian elimination are different from zero. The hypotheses in the above theorem ensure that this is the case.

Remark 2 - It is not sufficient for a matrix to be non-singular in order for it to have an LU decomposition. However, if the matrix is non singular then there exists a suitable permutation of its rows such that the permuted matrix has a (unique) LU decomposition.

A concept that is extremely useful in determining the behaviour and convergence of many algorithms to solve linear system is that of *diagonal dominance*. A matrix is strictly diagonally dominant if the modulus of each diagonal element is greater than the sum of the moduli of all the other elements in its row:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|, \qquad (1 \leq i \leq n).$$

This may seem a rather peculiar condition to require on a matrix. As a matter of fact, it is not so for many applications of practical interest. For example, most algorithms that solve numerically partial differential equations using finite

[1]A principal minor of order k is the minor

$$\begin{pmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{k1} & \cdots & a_{kk} \end{pmatrix}$$

```
**for** $k = 1, 2, \ldots, n$ **do**
    Compute the diagonal element
        $$\ell_{kk} = \left( a_{kk} - \sum_{s=1}^{k-1} \ell_{ks}^2 \right)^{1/2}.$$
    Build the $k$-th column of $L$:
    **for** $i = k+1, k+2, \ldots, n$ **do**
        $$\ell_{ik} = \left( a_{ik} - \sum_{s=1}^{k-1} \ell_{is}\ell_{ks} \right) / \ell_{kk}$$
    **end**
**end**
```

Table 2.2: *Algorithm for the Cholesky factorisation method*

difference methods involve the solutions of large linear systems whose matrix of coefficients is diagonally dominant.

The importance of diagonally dominance from the point of view of direct methods to solve linear systems of equations stems from the following theorems:

Theorem 2.3.2 *Every strictly diagonally dominant matrix is nonsingular and has an LU decomposition.*

Proof: Kincaid pages 190

Theorem 2.3.3 *Gaussian elimination of a diagonally dominant matrix does not require pivoting provided each row is first scaled with its maximum.*

Proof: Kincaid page 190

2.3.5 Cholesky factorisation

In the case of a symmetric, positive definite matrix (i.e. such that $\boldsymbol{x}^T A \boldsymbol{x} > 0 \ \forall \boldsymbol{x} \neq 0$) then it is possible to factorise the matrix as $A = LL^T$, in which L is lower triangular with a positive diagonal (*Cholesky factorisation*).

To find the Cholesky decomposition of the matrix A we can use an algorithm similar to that developed for the Doolittle and Crout methods and derived from it by assuming $U = L^T$. Its description is in Table 2.2.

2.3.6 Tridiagonal systems

Consider a 4×4 system

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 \\ a_1 & b_2 & c_2 & 0 \\ 0 & a_2 & b_3 & c_3 \\ 0 & 0 & a_3 & b_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}.$$

It is called a *tridiagonal system*. Tridiagonal systems give rise to particularly simple results when using Gaussian elimination. Forward elimination at each step yields a system

$$\begin{pmatrix} 1 & c_1/d_1 & 0 & 0 \\ 0 & 1 & c_2/d_2 & 0 \\ 0 & 0 & 1 & c_3/d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix},$$

where

$$\begin{array}{ll} d_1 = b_1, & y_1 = f_1/d_1, \\ d_2 = b_2 - a_1c_1/d_1, & y_2 = (f_2 - y_1a_1)/d_2, \\ d_3 = b_3 - a_2c_2/d_2, & y_3 = (f_3 - y_2a_2)/d_3, \\ d_4 = b_4 - a_3c_3/d_3, & y_4 = (f_4 - y_3a_3)/d_4. \end{array}$$

Finally, the backward substitution procedure gives the answer

$$\begin{cases} x_4 = y_4, \\ x_3 = y_3 - x_4c_3/d_3, \\ x_2 = y_2 - x_3c_2/d_2, \\ x_1 = y_1 - x_2c_1/d_1. \end{cases}$$

It is clear what will happen in general case of $n \times n$ tridiagonal system. The forward elimination procedure is

1. At the first step $d_1 = b_1$ and $y_1 = f_1/d_1$;
2. At the k-th step $d_k = b_k - a_{k-1}c_{k-1}/d_{k-1}$ and $y_k = (f_k - y_{k-1}a_{k-1})/d_k$.
3. Once all the y_k have been computed, the x_k can be determined using backward substitution: $x_n = y_n, \ldots, x_{k-1} = y_{k-1} - x_kc_{k-1}/d_{k-1}, \ldots$

2.4 Iterative methods

2.4.1 Introduction

The direct methods for solving linear systems of order n require about $n^3/3$ operations. In addition in practical computations with these methods, the errors which are necessarily introduced through rounding may become quite large for large n. Now we consider iterative methods in which an approximate solution is sought by using fewer operations per iteration. In general, these may be described as methods which proceed from some initial "guess", $\boldsymbol{x}^{(0)}$, and define a sequence of successive approximations $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$ which, in principle, converge to the exact solution. If the convergence is sufficiently rapid, the procedure may be terminated at an early stage in the sequence and will yield a good approximation. One of the intrinsic advantages of such methods is the fact that the errors, due to roundoff or even blunders, may be damped out as the procedure continues. In fact, special iterative methods are frequently used to improve "solutions" obtained by direct methods.

A large class of iterative methods may be defined as follows. Let the system to be solved be

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{2.15}$$

where $\det(A) \neq 0$. Then the coefficient matrix can be split, in an infinite number of ways, into the form

$$A = N - P,$$

where N and P are matrices of the same order as A. The system (2.15) is then written as

$$N\boldsymbol{x} = P\boldsymbol{x} + \boldsymbol{b}. \tag{2.16}$$

Starting from some *arbitrary* vector $\boldsymbol{x}^{(0)}$, we define a sequence of vectors $\{\boldsymbol{x}^{(n)}\}$, by the recursion

$$N\boldsymbol{x}^{(n)} = P\boldsymbol{x}^{(n-1)} + \boldsymbol{b}, \quad n = 1, 2, 3, \ldots \tag{2.17}$$

The different iterative methods are characterised by their choice of N and P. By looking at (2.17) we can already find some restrictions on the choice of N:

1. The matrix N should not be singular, i.e. $\det(N) \neq 0$.
2. The matrix N should be chosen such that a system of the form $N\boldsymbol{y} = \boldsymbol{z}$ is "easily" solved.

One can show that an iterative method converges if and only if the matrix $M = N^{-1}P$ exists and has all eigenvalues λ_i with modulus strictly less than 1:

$$\varrho(M) \equiv \max_i |\lambda_i| < 1 \Longleftrightarrow \text{Method converges}$$

where the symbol $\varrho(M)$ is called the *spectral radius* of M. It is sometimes hard to verify whether this condition is satisfied. A weaker, but more easily verifiable statement is that the method converges if at least one norm of the matrix M is strictly less than one:

$$\|M\| < 1 \implies \text{Method converges.}$$

However, the reverse is not true: a method may converge but some of the norms of M may be larger than or equal to one.

2.4.2 Iteration schemes

To simplify the notation we will assume in what follows that all the diagonal elements of the matrix A are one. This can be done by dividing each row by its diagonal element. If by chance, one row has a zero diagonal element we would just need to reorder the rows.

Jacobi's method

In Jacobi's method the matrix is split into diagonal and off-diagonal part:

$$A = I - (A_L + A_U), \implies \begin{cases} N & = I, \\ P & = A_L + A_U \end{cases}$$

where we have written the off-diagonal part as the sum of a lower and an upper triangular matrix, A_L and A_U respectively. The iteration scheme is

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{b} + (A_L + A_U)\boldsymbol{x}^{(n)}$$

and the convergence matrix is $M \equiv N^{-1}P = P$.

Gauss-Seidel's method

In Jacobi's method the old guess is used to estimate all the elements of the new guess. In Gauss-Seidel's method each new iterate is used as soon as it becomes available. The iteration scheme is as follows:

$$\begin{aligned} A &= I - (A_L + A_U), \\ \boldsymbol{x}^{(n+1)} &= \boldsymbol{b} + A_L\boldsymbol{x}^{(n+1)} + A_U\boldsymbol{x}^{(n)}, \implies \begin{cases} N & = I - A_L, \\ P & = A_U, \end{cases} \end{aligned} \tag{2.18}$$

where A_L and A_U are respectively the lower and upper triangular parts of A with diagonal elements set to zero. Notice that if we start computing the elements of $\boldsymbol{x}^{(n+1)}$ from the first, i.e. from $x_1^{(n+1)}$, then all the terms on the right hand side of (2.18) are known by the time they are used.

S.O.R. method

We can interpret the Gauss-Seidel algorithm as a method that applies at each guess $\boldsymbol{x}^{(n)}$ a correction term $\boldsymbol{c}$. We can rewrite the Gauss-Seidel iteration scheme as:

$$\begin{aligned} & \boldsymbol{x}^{(n+1)} = \boldsymbol{b} + A_L \boldsymbol{x}^{(n+1)} + A_U \boldsymbol{x}^{(n)} \\ \Longrightarrow \qquad & \boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \left[\boldsymbol{b} + A_L \boldsymbol{x}^{(n+1)} + (A_U - I)\boldsymbol{x}^{(n)}\right] \\ & \phantom{\boldsymbol{x}^{(n+1)}} = \boldsymbol{x}^{(n)} + \boldsymbol{c}, \end{aligned}$$

where the correction term is $\boldsymbol{c} = \boldsymbol{b} + A_L \boldsymbol{x}^{(n+1)} + (A_U - I)\boldsymbol{x}^{(n)}$. The idea of the S.O.R. (Successive Over-Relaxation) method is that the convergence may be pushed to go a bit faster by using a slightly larger correction. In other words, we introduce a parameter $\omega > 1$ (*SOR parameter*) that multiplies the correction term and write the iteration scheme as

$$\begin{aligned} \boldsymbol{x}^{(n+1)} &= \boldsymbol{x}^{(n)} + \omega \boldsymbol{c} \\ &= \boldsymbol{x}^{(n)} + \omega \left[\boldsymbol{b} + A_L \boldsymbol{x}^{(n+1)} + (A_U - I)\boldsymbol{x}^{(n)}\right]. \end{aligned}$$

Remark 1 - The introduction of the relaxation parameter ω can speed up convergence, but may also promote divergence. Moreover, it is not clear a priori what is the optimal value of ω to achieve convergence in the smallest number of iterations. A typical choice is to set ω quite close to one.

2.4.3 Analysis of convergence

A concept that is extremely useful in determining the behaviour and convergence of many algorithms to solve linear system is that of *diagonal dominance*. A matrix is strictly diagonally dominant if the modulus of each diagonal element is greater than the sum of the moduli of all the other elements in its row:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \qquad (1 \leq i \leq n).$$

This may seem a rather peculiar condition to require on a matrix. As a matter of fact, it is not so for many applications of practical interest. For example, most algorithms that solve numerically partial differential equations using finite difference methods involve the solutions of large linear systems whose matrix of coefficients is diagonally dominant.

The importance of diagonally dominance from the point of view of direct methods to solve linear systems of equations stems from the following theorems:

Theorem 2.4.1 *If the matrix of the coefficients A is strictly diagonally dominant then the Jacobi method converges.*

Theorem 2.4.2 *If the matrix of the coefficients A is strictly diagonally dominant then the Gauss-Seidel method converges.*

Remark - Note that diagonal dominance (i.e. the sum can be equal to the diagonal element) in itself does not ensure convergence.

Theorem 2.4.3 *If the matrix of the coefficients A is symmetric and positive definite then Gauss-Seidel's method converges.*

Remark 1 - This theorem does not hold for Jacobi's method.
Remark 2 - Convergence theorem for the SOR methods are much harder to prove.

2.5 Determinants and inverses

The entire gist of this section on linear systems has been that we may not wish to know the inverse or the determinant of the matrix of the coefficients in order to solve a linear system of equations. However, sometimes we need to know these two quantities. We can use the methods that we have listed to compute them accurately.

The LU decomposition offers the determinant as an easy to obtain side product. If $A = LU$ then

$$\det(A) = \det(L) \times \det(U).$$

In the case of Doolittle's factorisation we have $\det(L) = 1$ and therefore

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

If partial pivoting has been used for the Gaussian elimination then

$$\det(A) = \pm \prod_{i=1}^{n} u_{ii},$$

where the sign is positive or negative depending if the number of row permutations is even or odd.
Remark - To find the determinant using Gaussian elimination requires a number of operations of the order of n^3. The expansion in minors requires a number of operations of the order $n!$.

Finally, to find the inverse we can solve n systems of n linear equations. Let $\boldsymbol{e}_i$ denote the vector whose components are all zero except the i-th component that is, instead, equal to 1. Let $\boldsymbol{c}_i$ be the i-th column of A^{-1}. Then

$$AA^{-1} = I \implies A\boldsymbol{c}_i = \boldsymbol{e}_i, \quad i = 1, 2, \dots, n.$$

Therefore for each value of i we have to solve a system of n linear equations for the i-th column of the inverse of A. This method is much faster than Cramer's rule, that requires the evaluation of $n + 1$ determinants.

Further reading

Topics covered here are also covered in

- Chapters 3 and 8 of Linz & Wang, *Exploring Numerical Methods* (QA297 LIN),
- Chapter 4 (and extended through chapter 5) of Kincaid & Cheney, *Numerical Analysis* (QA297 KIN),
- Chapters 2 and 3 of Süli & Mayers, *An Introduction to Numerical Analysis* (not in library),
- throughout the extremely detailed Golub & Van Loan, *Matrix Computations* (QA263 GOL).

Chapter 3

Nonlinear equations

3.1 Introduction

Very few nonlinear equations can be solved analytically. For example, it is easy to solve the equation $x^2 + 2x + 1 = 0$: the left hand side is $(x + 1)^2$ and therefore there is a single root $x = -1$ with multiplicity two. There are other equations whose solution is not so easily found, like, for example,

$$e^x + x = 2.$$

From the graph of this equation it is clear that there is one and only one solution. However, there is no formula to obtain its exact value. The purpose of numerical methods for the solution of nonlinear equations is to fill in this gap: to give approximate, but accurate, solutions of nonlinear equations that cannot be solved analytically.

In this unit we give an introduction to this area of numerical analysis by discussing some simple algorithms. It should be made clear that the algorithms currently used in commercial packages and in research laboratories are far more advanced than anything that we will discuss. However, they are based on the same ideas and differ mainly in details of the implementation and in using various tricks to guarantee a fast and global convergence.

We will focus our attention mainly on the solution of a single nonlinear equation in one variable. However, the methods that we will discuss can be extended to systems of nonlinear equations in more than one variable: we will discuss briefly how to do this at the end of this chapter.

In all that follows we assume that we have to solve the nonlinear equation $f(x) = 0$, where x is a real number and $f(x)$ is a real differentiable function.

3.2 A simple example: The bisection method

The bisection method is by far the simplest method and, even though it is not used very much in practice, it is sturdy and reliable. Moreover, we can use it to introduce some general comments on the practical implementation of root finding algorithms.

Very briefly, the method assumes that by suitable inspired guesses two points $x_0^{(L)}$ and $x_0^{(R)}$ have been chosen such that

$$f(x_0^{(L)})f(x_0^{(R)}) \leq 0, \tag{3.1}$$

i.e. the function $f(x)$ changes sign in the interval $[x_0^{(L)}, x_0^{(R)}]$. If the product is zero then one of the two factors is zero and the problem is solved. We therefore assume that the inequality in (3.1) is strict. Since the function $f(x)$ is continuous, by the intermediate value theorem there exists a point $s \in (x_0^{(L)}, x_0^{(R)})$ such that $f(s) = 0$, i.e. s is a root of the equation $f(x) = 0$. The idea behind this method is that the mid point between $x_0^{(L)}$ and $x_0^{(R)}$, $x_0^{(M)}$, is an approximation of the root s. If a more accurate estimate is needed we can refine the approximation by checking in which half interval $(x_0^{(L)}, x_0^{(M)})$ or $(x_0^{(M)}, x_0^{(R)})$ the function $f(x)$ changes sign. We then discard the other half-interval and repeat the procedure.

More formally, the iteration procedure involves first constructing a new point, the centre of the interval and estimate of the root,

$$x_n^{(M)} = \frac{x_n^{(L)} + x_n^{(R)}}{2}, \quad n = 0, 1, 2, \ldots$$

and evaluating $f(x_n^{(M)})$. If this estimate of the root is not sufficiently accurate a new set of left and right points are chosen according to the sign of $f(x_n^{(M)})$:

$$\begin{cases} x_{n+1}^{(L)} &= x_n^{(L)}, \\ x_{n+1}^{(R)} &= x_n^{(M)} \end{cases} \quad \text{if } f(x_n^{(L)})f(x_n^{(M)}) < 0, \tag{3.2}$$

$$\begin{cases} x_{n+1}^{(L)} &= x_n^{(M)}, \\ x_{n+1}^{(R)} &= x_n^{(R)} \end{cases} \quad \text{if } f(x_n^{(L)})f(x_n^{(M)}) > 0. \tag{3.3}$$

The procedure is repeated until a stopping condition is reached. There are three conditions that must be checked by any iteration procedure: if any of them is satisfied the iteration must stop.

1. The number of iterations has exceeded a predetermined value: this is used to avoid cases where the convergence is exceedingly slow or, for any reason, the algorithm is going in an infinite loop.

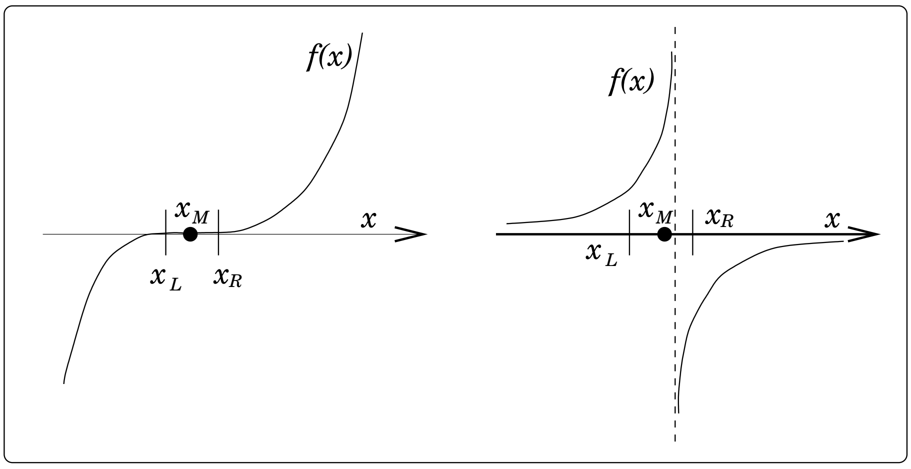


Figure 3.1: *In the left hand case the criterion* $\left|x_n^{(R)} - x_n^{(L)}\right| < \delta$ *fails, in the right hand case the criterion* $\left|f(x_n^{(M)})\right| < \varepsilon$ *fails.*

2. The absolute value of the function at the estimated root, $f(x_n^{(M)})$ is smaller than a predetermined number ε (usually fixed by the number of significant digits of the floating point representation).

3. The difference between two successive values of the estimated root (or the difference between $x_n^{(R)}$ and $x_n^{(L)}$ in the case of the bisection method) is smaller that a predetermined number δ, the requested accuracy of the roots.

Figure 3.1 shows two pathological cases where one of the last two criteria is satisfied, but not the other. In the left hand case there is a multiple root (a bane of root finding algorithms): the value of the function is very small, but the the left and right hand point are not close. For many numerical methods the speed of convergence is proportional to the slope of $f(x)$ at its root. Therefore their convergence is extremely slow at multiple roots where $f(x)$ is flat. This is not the case of the bisection method because its rate of convergence is independent of the slope of the function. However, if the value of the function is very close to zero then numerical errors may introduce spurious zeros and force the algorithm to converge to a spurious root.

In the right hand case the bisection interval is very small so that $\left|x^{(R)} - x^{(L)}\right| < \delta$ but the function is not small. While it is true that in this case the function is not continuous, it is also true that it may not be easy to determine whether the function whose zeros we wish to compute is continuous and so we must make a root finding algorithm capable of handling cases as pathological as these examples.

Remark 1 - The error in the location of the root at the n-th step is smaller than

$(x_0^{(R)} - x_0^{(L)})/2^n$.
Remark 2 - This method is easy to code and it always converges. However, it is rather slow.

3.3 Contraction mappings

3.3.1 Introduction

Many of the methods that we will discuss for solving nonlinear equations like

$$f(x) = 0 \tag{3.4}$$

are *iterative* and can be written in the form

$$x_{n+1} = g(x_n), \tag{3.5}$$

for some suitable function $g(x)$ and initial approximation x_0. The aim of the method is to find a suitable function $g(x)$ such that the sequence has a limit and that the limit is a root of $f(x)$:

$$\lim_{n\to\infty} x_n = s \quad \text{and} \quad f(s) = 0.$$

Note that if the limit exists then it is also a fixed point of the map $g(x)$:

$$s = \lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} g(x_n) = g\left(\lim_{n\to\infty} x_n\right) = g(s).$$

For example, in the case of the nonlinear problem $f(x) = 0$ we can define the function $g(x)$ to be

$$g(x) = x - f(x)$$

and use the mapping (3.5) to attempt finding the roots of $f(x)$. Methods of this kind are called *functional iterations methods* or *fixed point methods*.

Graphically, the solutions of (3.4) or the fixed points of (3.5) are the intersections between the graph of $g(x)$ and the line $y = x$ (see Figure 3.2). The iteration of the map (3.5) can be represented on the same graph (see Figure 3.2): in the case of the solid line path the method is converging to the fixed point, while for the dashed line path the method is diverging.

There are some general theorems that state under what condition the mapping (3.5) converges. Before studying them, however, it is worthwhile to make some general remarks.

- Usually iterative methods are valid for real and complex roots. However, in the latter case complex arithmetic must be incorporated into the appropriate computer codes and the initial estimate of the root must usually be complex.

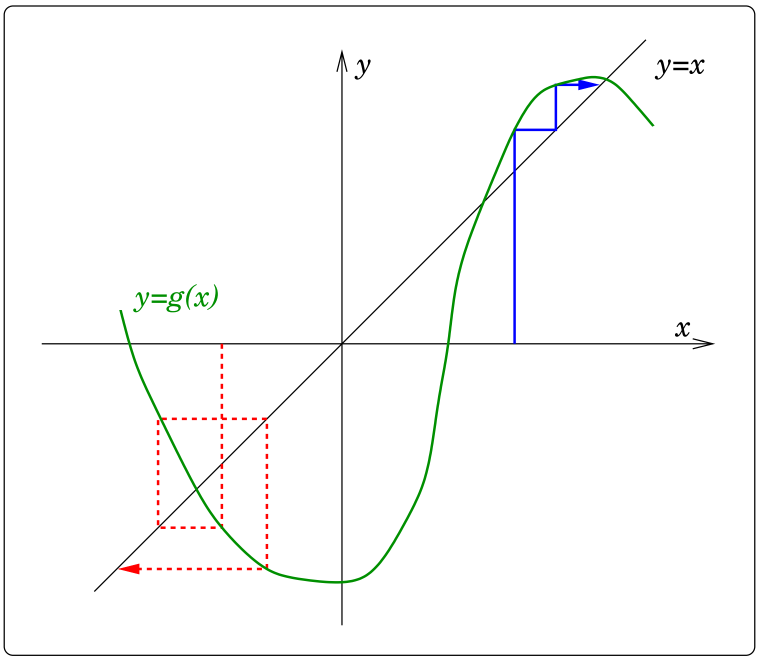


Figure 3.2: *Graphical representation of the functional iteration. The straight line paths are the graphical representation of the mapping (3.5).*

- The iterative methods require at least one initial estimate or guess at the location of the root being sought. If this initial estimate is “sufficiently close” to a root, then, in general, the procedure will converge. The problem of how to obtain such a “good” estimate is unsolved in general.
- As a general empirical rule, the schemes which converge more rapidly (i.e. higher order methods) require closer estimates. In practice, these higher order schemes may require the use of more significant digits in order that they converge as theoretically predicted. Thus, it is frequently a good idea to use a simple method to start with and then, when fairly close to the root, to use some higher order method for just a few iterations.

3.3.2 Geometrical interpretation of fixed point schemes

Before discussing formally what properties a map must have in order for the iteration scheme (3.5) to converge, we can obtain an approximate idea by considering the four maps in Figure 3.3. From the top two maps it is clear that in order to have a fixed point we must require $g(x)$ to be continuous (top left) and, moreover, that the range is contained in the domain (top right). These requirements are not enough to have a unique fixed point as it is shown in the bottom left corner: if the slope of the map is too high there may be two or more fixed points. It is only if the slope is smaller than unity that there can be only one fixed point (bottom right corner). Note than we do not require that map to be differentiable: it can have as many corners as it wish. The case of the two bottom maps is illustrated pictorially in Figure 3.4: at each iteration the image of the starting set I gets smaller and smaller until it reduces to a point (see Figure 3.4).

3.3.3 Definitions

We must now phrase these intuitive results in more formal terms. We start by defining a contracting map.
Definition - A continuous map $g(x)$ from an interval $I = [a, b] \subseteq \mathbb{R}$ into $\mathbb{R}$ is *contracting* if

1. the image of I is contained in I:
$$g(I) \subseteq I \quad \Leftrightarrow \quad g(x) \in I \, \forall x \in I.$$
2. the function $g(x)$ is Lipschitz continuous in I with Lipschitz constant $L < 1$:
$$|g(x) - g(y)| \leq L|x - y| \quad \forall x, y \in I.$$

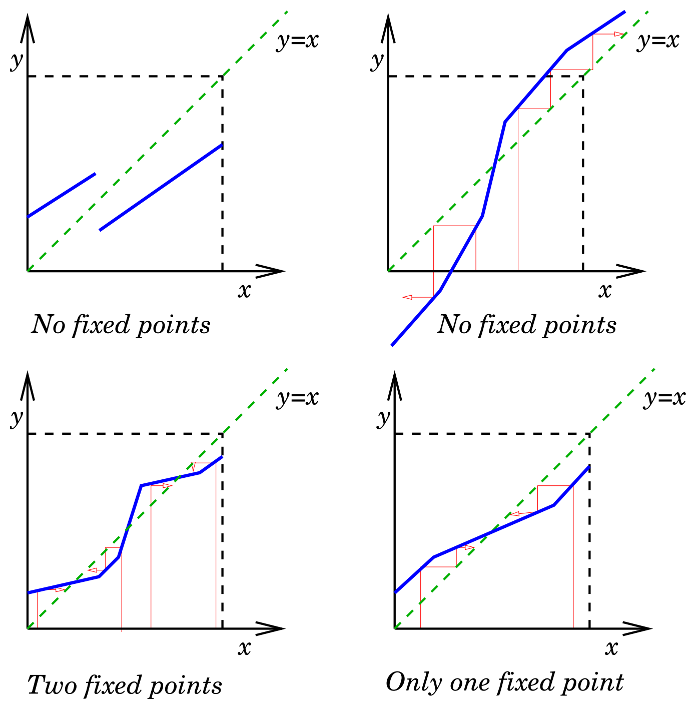


Figure 3.3: *Examples of maps and their convergence properties.*

In other words, the distance between the images is smaller than the distance between the two starting points.

Remark 1 - A function that is Lipschitz continuous is "more" than continuous, but "less" than differentiable. For example, the function $f_1(x) = \sqrt{x}$ is continuous in the interval $[0, 1]$, but it is not Lipschitz. On the other hand, the function $f_2(x) = |x|$ is Lipschitz in the interval $[-1, 1]$, but it is not differentiable at $x = 0$.
Remark 2 - If the function $g(x)$ is differentiable and Lipschitz continuous with constant L then

$$\left|\frac{\mathrm{d}g}{\mathrm{d}x}\right| \leq L.$$

3.3.4 Convergence theorems

A map that is contracting is also called a *contraction mapping*. From the definition and Figures 3.3 and 3.4 we can intuitively understand that if the map $g(x)$ in (3.5) is contracting, then we are guaranteed convergence. All this is expressed more formally (and more clearly) in the following sets of theorems.

First of all, we prove that if the image of the interval is contained in the interval (without any requirement of Lipschitz continuity) then there is at least one fixed point (but there may be many).

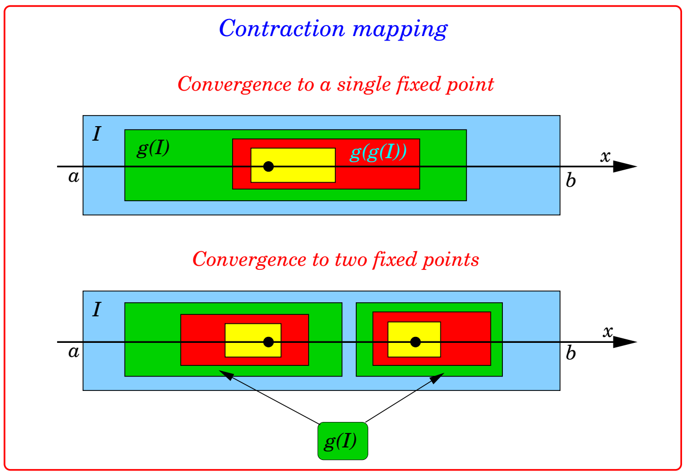

Contraction mapping
Convergence to a single fixed point
I
g(I)
g(g(I))
a
b
x
Convergence to two fixed points
I
a
b
x
g(I)

Figure 3.4: *Graphical representation of the contraction mapping principle. At each iteration the image of the interval gets smaller and the iterations of the contraction map converge towards its fixed point(s).*

Theorem 3.3.1 *If the function $g(x)$ is continuous in $I = [a, b]$ and $g(I) \subseteq I$, then $g(x)$ has at least one fixed point in I.*

Proof - Since $g(I) \subseteq I$ we must have

$$a \leq g(a) \leq b \quad \text{and} \quad a \leq g(b) \leq b.$$

If either $g(a) = a$ or $g(b) = b$ then there is one fixed point and the theorem is proved. Otherwise, the following inequalities must hold:

$$g(a) - a \geq 0 \quad \text{and} \quad g(b) - b \leq 0.$$

Define the function $F(x) = g(x) - x$. $F(x)$ is continuous and $F(a) \geq 0$, while $F(b) \leq 0$. Therefore, by the intermediate value theorem there exists a point $c \in [a, b]$ such that

$$F(c) = 0 \implies c = g(c).$$

■

Exercise - Give a graphical representation of this theorem. In particular show that the contraction map that would produce a graph similar to the bottom part of Figure 3.4 must have a jump discontinuity in $[a, b]$.

Theorem 3.3.1 provides us with an important result, but not with enough. Ideally we would like the zero to be unique. In order for this to be true, we must have more stringent requirements on $g(x)$: it must not vary too rapidly. This is assured if $g(x)$ is Lipschitz continuous with a sufficiently small Lipschitz constant.

Theorem 3.3.2 (Contraction mapping theorem) *If $g(x)$ is a contraction mapping in an interval $I = [a, b]$ then there exists one and only one fixed point of the map in $[a, b]$.*

We will prove a slightly less strong version of this theorem: we require that the map $g(x)$ is differentiable in I and that $|g'(x)| \leq L < 1$. Such a function is Lipschitz continuous of Lipschitz constant L: therefore it is a contraction mapping.

Theorem 3.3.3 *If $g(x)$ is a differentiable contraction mapping in an interval $I = [a, b]$, i.e.*

$$|g'(x)| \leq L < 1, \qquad \forall x \in [a, b],$$

then there exists one and only one fixed point of the map in $[a, b]$.

Proof - The existence of the derivative implies that the function $g(x)$ is continuous. Since, by hypothesis $g(I) \subseteq I$, then by Theorem 3.3.1 there is at least one fixed point, s_1. Suppose that there is another one $s_2 : s_2 = g(s_2)$ and $s_1 \neq s_2$. We can use the mean value theorem[1] to prove that this is impossible:

$$|s_2 - s_1| = |g(s_2) - g(s_1)| = |g'(\xi)(s_2 - s_1)| \leq L|s_2 - s_1| < ||s_2 - s_1|.$$

This inequality cannot be true and therefore there can be no other fixed point. (The proof assuming Lipschitz continuity only is essentially identical.) ■

The consequence of this theorem is that the algorithm represented by the mapping (3.5) is guaranteed to have a root in an interval $[a, b]$ if the map is a contraction mapping in this interval. The following theorem tells us how to find it.

Theorem 3.3.4 *Let $I = [a, b]$ and suppose that $g(x)$ is a contraction mapping in I. Then for arbitrary $x_0 \in I$, the sequence $x_n = g(x_{n-1})$, $n = 1, 2, \ldots$ converges to the unique fixed point, s, of the map. Moreover, if the error e_n at the n-th stage is defined by $e_n = x_n - s$ then*

$$|e_n| \leq \frac{L^n}{1 - L}|x_1 - x_0|.$$

[1]**Mean value theorem** - For any differentiable $F(x)$ in $I \subseteq \mathbb{R}$ and any $c, d \in I$ there exists a point $\xi \in [c, d]$ such that

$$F'(\xi) = \frac{F(d) - F(c)}{d - c}.$$

Proof - To prove the convergence to the fixed point, whatever the arbitrary guess $x_0 \in I$, we use the Lipschitz property of the map and the fact that $s = g(s)$ to bound $|e_n|$ from above with a bound that tends to zero as n tends to infinity. As a first step we have:

$$\begin{aligned} |e_n| &= |x_n - s| && (3.6)\\ &= |g(x_{n-1}) - g(s)| && (3.7)\\ &\le L|x_{n-1} - s| && (3.8)\\ &= L|e_{n-1}|. && (3.9) \end{aligned}$$

By applying this inequality over and over again we obtain

$$\begin{aligned} |e_n| &\le L|e_{n-1}| && (3.10)\\ &\le L^2|e_{n-2}| && (3.11)\\ &\le \dots && (3.12)\\ &\le L^n|e_0| && (3.13) \end{aligned}$$

By the definition of contraction mapping $L < 1$ and therefore

$$\lim_{n\to\infty} L^n = 0 \implies \lim_{n\to\infty} x_n = s.$$

Equation (3.13) provides a bound on the error of the n-th estimate in terms of the initial error. Unfortunately this quantity is not know, because we do not know the root s of the equation. We therefore must replace $|e_0|$ in (3.13) with an expression that we can compute, namely $|x_0 - x_1|$:

$$\begin{aligned} |x_0 - s| &= |x_0 - x_1 + x_1 - s| && (3.14)\\ &\le |x_0 - x_1| + |x_1 - s| && (3.15)\\ &\le |x_0 - x_1| + L|x_0 - s| && (3.16)\\ \implies \qquad |e_0| &= |x_0 - s| && (3.17)\\ &\le \frac{|x_0 - x_1|}{1 - L}. && (3.18) \end{aligned}$$

Using (3.13) we obtain

$$|e_n| \le L^n|e_0| \le \frac{L^n}{1-L}|x_0 - x_1|. \qquad (3.19)$$

■

These four theorems are represented graphically in Figure 3.5. Since the slope of the function $g(x)$ is smaller than unity successive iterations of the map get closer and closer to its fixed point.

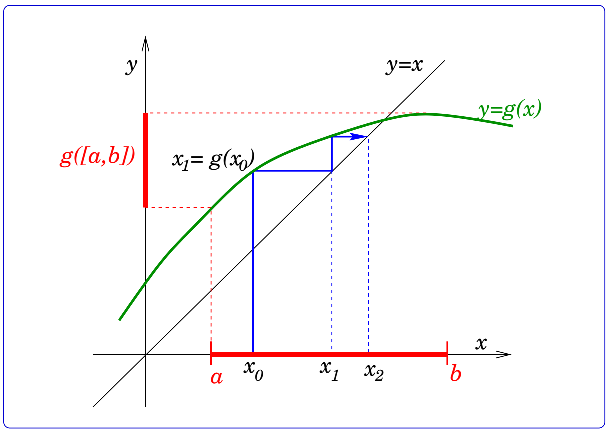


Figure 3.5: *Graphical representation of the contracting mapping theorems. The map $g(x)$ is a contraction mapping in $[a, b]$: the image of the interval is smaller than the original interval. Each iteration of a starting guess x_0 gets closer and closer to the fixed point.*

3.3.5 Speed of convergence

The bound (3.19) on the error provided by Theorem 3.3.4 depends, through the Lipschitz constant L, on the interval chosen to estimate the map. Moreover, it is reasonable to assume that the rate of convergence of the map, i.e. the rate of decrease of the error e_n depends only on the properties of the map in a neighbourhood of the root. We can show that this is indeed the case if the map is differentiable, so that it is possible to expand it in a Taylor polynomial: call $g(x)$ a suitably differentiable contraction mapping in $I = [a, b]$, s its fixed point, $x_0 \in I$ the starting point of the iteration and $e_n = x_n - s$ the error at the n-th iteration. Using the definition of the map and Taylor's expansion we can write:

$$\begin{aligned}
e_{n+1} &= x_{n+1} - s \\
&= g(x_n) - g(s) \\
&= g'(s)(x_n - s) + \frac{g''(s)}{2!}(x_n - s)^2 + \ldots + \frac{g^{(k)}(s)}{k!}(x_n - s)^k + R_{n,k} \\
&= g'(s)e_n + \frac{g''(s)}{2!}e_n^2 + \ldots + \frac{g^{(k)}(s)}{k!}e_n^k + R_{n,k},
\end{aligned}$$

where $R_{n,k}$ is the remainder of the expansion:

$$R_{n,k} = \frac{g^{(k)}(\xi)}{k!}(x_n - s)^{k+1}, \quad \xi \in [x_n, s].$$

Assuming that $g'(s) \neq 0$ then

$$e_{n+1} \sim g'(s)e_n,$$

i.e. the error decreases at a constant rate at each iteration: such a method is called *linear* or *first order*. If, instead, $g'(s) = 0$, but $g''(s) \neq 0$ then

$$e_{n+1} \sim g''(s)e_n^2,$$

i.e. the error at each iteration is proportional to the square of the previous error: such a method is called a *quadratic* or *second order* method. The more derivatives of $g(s)$ vanish the higher the order of the method and the faster the convergence. However, it may well be that the method will converge only if the starting point is very close to the root.

3.3.6 Error propagation

The final question that we must answer before discussing practical implementations of the theory we have just studied is "Are these theorems numerically stable?" In other words, what is the effect of the numerical error on the convergence properties of a contraction map? In actual computations it may not be possible, or practical, to evaluate the function $g(x)$ exactly (i.e. only a finite number of decimals may be retained after rounding or $g(x)$ may be given as the numerical solution of a differential equation, etc.). For any value of x we may then represent our approximation to $g(x)$ by $G(x) = g(x) + \delta(x)$ where $\delta(x)$ is the error committed in evaluating $g(x)$. Frequently we may know a bound for $\delta(x)$, i.e. $\delta(x) < \delta$. Thus the actual iteration scheme which is used may be represented as

$$X_{n+1} \equiv G(X_n) = g(X_n) + \delta_n, \quad n = 0, 1, 2, \dots, \tag{3.20}$$

where the X_n are the numbers obtained from the calculations and the $\delta_n \equiv \delta(X_n)$ satisfy

$$|\delta_n| \leq \delta, \quad n = 0, 1, 2, \dots \tag{3.21}$$

We cannot expect the computed iterates X_n of (3.20) to converge. However, under proper conditions, it should be possible to approximate a root to an accuracy determined essentially by the accuracy of the computations, δ.

For example, from Figure 3.6 we can see that for the special case of $g(x) = \alpha + L(x - \alpha)$, the uncertainty in the root α is bounded by $\pm\delta/(1 - L)$. We note that if the slope L is close to unity the problem is not "properly posed". The following theorem states quite generally that when the functional iteration scheme is convergent, the presence of errors in computing $g(x)$, of magnitudes bounded by δ, causes the scheme to estimate the root α with an uncertainty bounded by

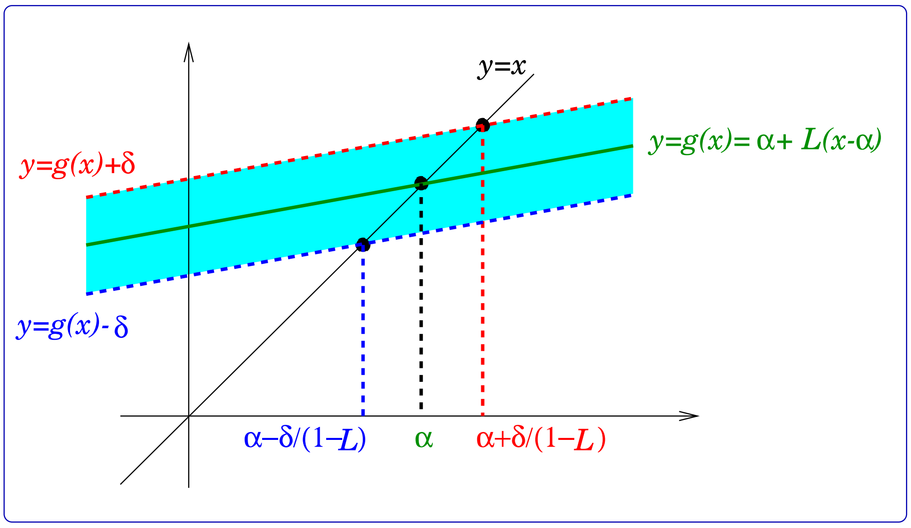


Figure 3.6: *The width of the numerical error band around the fixed point of the iteration map.*

$\pm\delta/(1-L)$, where L is the Lipschitz constant of the contraction mapping. The phrasing of this theorem is slightly different from the previous ones because the numerical error δ forces us to define quite strictly the interval we want to work in: we know that $g(I) \subseteq I$, but it is not generally true that $G(I) \subseteq I$.

Theorem 3.3.5 *Let $g(x)$ be a contraction mapping with fixed point s and let L be its Lipschitz constant in the interval $I(s, r_0) \equiv [s - r_0, s + r_0]$. Let δ be the bound on the numerical errors of the iterates of the numerical map (3.20) as defined in (3.21). Finally, assume that the starting point of the iteration of (3.20) is a point X_0 in the smaller interval*

$$X_0 \in I(s, R_0) \equiv [s - R_0, s + R_0], \tag{3.22}$$

where

$$0 < R_0 \leq r_0 - \frac{\delta}{1-L}.$$

Then the iterates X_n of (3.20) with the errors bounded by (3.21), lie in the interval $I(s, r_0)$, and

$$|s - X_n| \leq \frac{\delta}{1-L} + L^n \left(R_0 - \frac{\delta}{1-L} \right), \tag{3.23}$$

where $L^n \to 0$ as $n \to \infty$.

Proof - The proof of this theorem involves showing that the numerical error does not push the iteration outside the interval $I(s, r_0)$ where it is defined. In the process of doing so we also derive the bound (3.23) on the error of the numerical estimate. The proof that the iterations are always in the interval $I(s, r_0)$ is by induction.

The point X_0 is, by hypothesis, inside the interval $I(s, R_0) \subseteq I(s, r_0)$. We now suppose that the iterations $X_0, X_1, \ldots, X_{n-1}$ are in $I(s, r_0)$ and proceed to show that also $X_n \in I(s, r_0)$. By (3.20) and (3.21) we have

$$|s - X_n| \le |[g(s) - g(X_{n-1})] - \delta_{n-1}| \le |[g(s) - g(X_{n-1})]| + \delta.$$

Since $g(x)$ is a contraction mapping of Lipschitz constant L we can write

$$\begin{aligned} |s - X_n| &\le L|s - X_{n-1}| + \delta \\ &\le L^2|s - X_{n-2}| + L\delta + \delta \\ &\le L^n|s - X_0| + (L^{n-1} + \ldots 1)\delta \\ &= L^n|s - X_0| + \frac{1 - L^n}{1 - L}\delta. \end{aligned}$$

Hypothesis (3.22) implies that $|s - X_0| \le R_0$ so that

$$\begin{aligned} |s - X_n| &\le L^n R_0 + \frac{1 - L^n}{1 - L}\delta \\ &= L^n R_0 + \frac{\delta}{1 - L} - L^n \frac{\delta}{1 - L} \\ &\le R_0 + \frac{\delta}{1 - L} \\ &\le r_0. \end{aligned} \tag{3.24}$$

Thus all the iterates are in the interval $I(s, r_0)$ and the iteration process is defined. Moreover (3.24) can be rewritten as (3.23), thus completing the proof. ∎

Remark - Theorem 3.3.5 shows that the method is "as convergent as possible", that is, the computational errors which arise from the evaluation of $g(x)$ may cumulatively produce an error of magnitude at most $\delta/(1 - L)$. Moreover, such errors limit the size of the error bound *independently of the number of iterations*. Therefore, it is pointless to iterate the map until $L^n r_0 \ll \delta/(1 - L)$.

3.4 Examples of iteration procedures

3.4.1 Introduction

We have completed the theoretical introduction to the numerical solution of non-linear equations. We must now discuss some of the algorithms that apply the theory that we have developed.

3.4.2 The chord method (first order)

The chord method and Newton's methods are examples of the application of the contraction mapping theorems. Both these methods can be introduced using a general and elegant framework. As usual we suppose that we have to solve the nonlinear equation

$$f(x) = 0,$$

in some interval $a \leq x \leq b$. To define the different methods to solve this problem we introduce a function $\varphi(x)$ such that

$$0 < \varphi(x) < \infty, \quad x \in [a, b], \tag{3.25}$$

and we use it to construct a contraction mapping $x_{n+1} = g(x_n)$, where

$$g(x) = x - \varphi(x) f(x). \tag{3.26}$$

The fixed points of the map $g(x)$ are the roots of the function $f(x)$.

The simplest choice for $\varphi(x)$ in (3.26) is to take

$$\begin{aligned} & & \varphi(x) &\equiv m \neq 0, & (3.27)\\ &\Longrightarrow & g(x) &= x - mf(x) & (3.28)\\ &\Longrightarrow & x_{n+1} &= x_n - mf(x_n) & (3.29) \end{aligned}$$

where m is a number that we must choose appropriately in order for $g(x)$ to be a contraction mapping in $[a, b]$. The range of m depends on the slope of $f(x)$ in the interval $[a, b]$. From Theorem 3.3.3 we know that $g(x)$ is a contraction mapping if

$$\begin{aligned} & & |g'(x)| &< 1 \ \forall x \in [a, b] \\ &\Longrightarrow & |1 - mf'(x)| &< 1 \ \forall x \in [a, b] \\ &\Longrightarrow & 0 < mf'(x) &< 2 \ \forall x \in [a, b]. & (3.30) \end{aligned}$$

Thus m must have the same sign as $f'(x)$, while if $f'(x) = 0$ the inequality cannot be satisfied.

The iterates of (3.29) have a geometrical realisation in which the value x_{n+1} is the x intercept of the line with slope $1/m$ through $(x_n, f(x_n))$ (see Figure 3.7). The inequality (3.30) implies that this slope should be between ∞ (i.e. vertical) and $f'(x)/2$ (i.e. half the slope of the tangent to the curve $y = f(x)$). It is from this geometric description that the name chord method is derived - the next iterate is determined by a chord of constant slope joining a point on the curve to the x-axis.

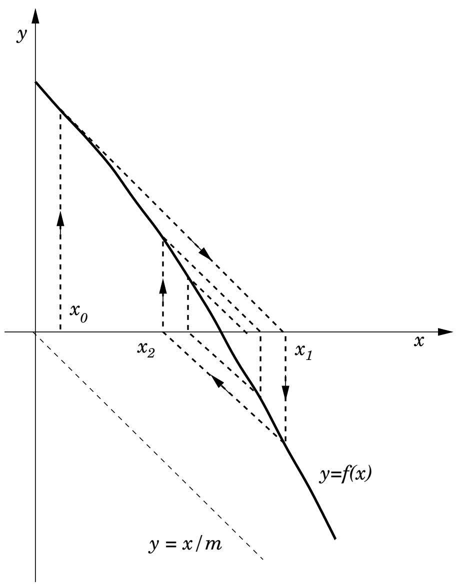


Figure 3.7: *Geometrical representation of the chord method: each new iterate is determined by a chord of constant slope joining a point on the curve to the x-axis.*

3.4.3 Newton's method (second order)

The idea behind Newton's method is to chose the function $\varphi(x)$ in order that the derivative of the iteration mapping $g(x)$ is zero at the root $x = s$. This is ensured by the choice

$$\varphi(x) = \frac{1}{f'(x)} \implies g(x) = x - \frac{f(x)}{f'(x)}, \tag{3.31}$$

so that the iteration procedure is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{3.32}$$

This root finding algorithm is called Newton's method. Theorem 3.3.3 guarantees that the method converges in an interval $[a, b]$ containing the root provided that $|g'(x)| < 1$ in the interval. It is at least second order at the root s of the equation $f(x) = 0$, if $f'(s) \neq 0$ and $f''(x)$ exists, since

$$g'(s) = \frac{f(s)f''(s)}{[f'(s)]^2} = 0. \tag{3.33}$$

The geometrical interpretation of this scheme simply replaces the chord in Figure 3.7 by the tangent to the line to $y = f(x)$ at $x_n, f(x_{n+1})$ (see Figure 3.8).

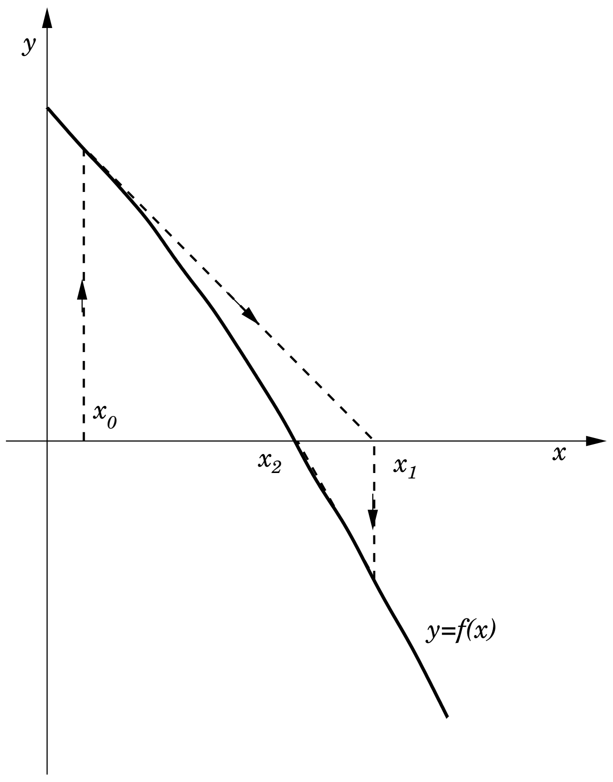


Figure 3.8: *Geometrical representation of Newton's method: each new iterate is intersection of the tangent to the graph of* $f(x)$ *with the* x*-axis.*

Remark 1 - It should be noted that Newton's method may be undefined and the condition (3.25) violated if $f'(x) = 0$ for some $x \in [a, b]$. In particular, if at the root $x = s$, $f'(s) = 0$, the procedure may no longer be of second order since the hypotheses that lead to (3.33) are not satisfied. To examine this case we assume that $f(x)$ has a root of multiplicity p at $x = s$. In other words we can write,

$$f(x) = (x - s)^p h(x), \quad p > 1,$$

where the function $h(x)$ has a second derivative and $h(s) \neq 0$. If we substitute this expression in the definition of $g(x)$ in (3.31) we find that

$$|g'(s)| = 1 - \frac{1}{p}.$$

So only in the case of a linear root, i.e. $p = 1$ is Newton's method second order, but it will converge as a first order method in the general case $p \neq 1$.
Remark 2 - Convergence is quadratic only if we are close to the root s.
Remark 3 - The advantage of Newton's method with respect to the bisection (and chord) method is the faster convergence. The main disadvantage with respect to the bisection method is that we need to start relatively close to the root in order to be sure that the method will converge.

3.4.4 Secant method (fractional order)

Newton's method requires the evaluation of the derivative of the function $f(x)$ whose root we want to find. To do this may be very complicated and time consuming: the secant method obviates this problem by approximating the derivative of $f'(x)$ with the quotient

$$f'(x_n) \simeq \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}. \tag{3.34}$$

This approximation comes directly from the definition of the derivative of $f(x)$ as the limit

$$f'(x) = \lim_{u \to x} \frac{f(u) - f(x)}{u - x}.$$

Substituting (3.34) into the algorithm (3.32) for Newton's method we obtain the secant method, namely:

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}. \tag{3.35}$$

The graphical interpretation of the secant method is similar to that of Newton's method. The tangent line to the curve is replaced by the secant line (see Figure 3.9).

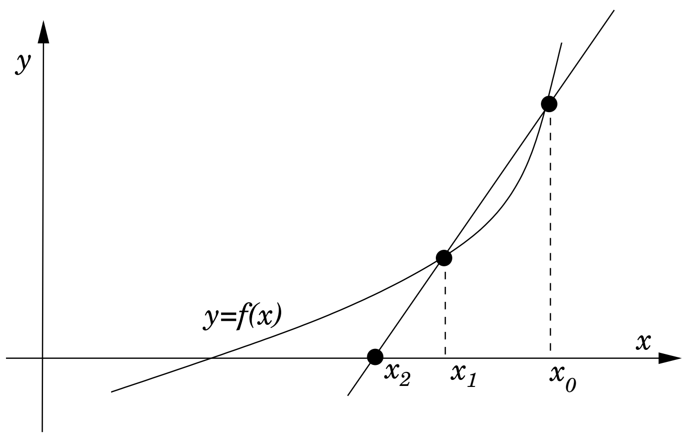


Figure 3.9: *Geometrical representation of the secant method: each new iterate is intersection of the secant to the graph of $f(x)$ with the x-axis.*

Remark 1 - This method requires two initial guesses: x_0 and x_1.
Remark 2 - The convergence of the secant method cannot be analysed using the contraction mapping theorems that we have studied because the method cannot be put in the form $x_{n+1} = g(x_n)$: each new iterate is a function of the previous **two**: $x_{n+1} = g(x_n, x_{n-1})$.
Remark 3 - It is possible to show that the error in the approximation decreases asymptotically as

$$|e_{n+1}| \sim |e_n|^{(1+\sqrt{5})/2}.$$

Since $(1 + \sqrt{5})/2 \simeq 1.62$ the secant method is a super-linear (i.e. faster than linear), but slower than quadratic convergence. This would suggests that the secant method is slower than Newton's method in reaching a given accuracy. However, Newton's method requires two function evaluations per iteration step, while the secant method requires only one. Therefore, when comparing the execution speed of the two methods we should/could compare *two* steps of the secant methods with one step of Newton's method. The rate of decrease of the error in two steps of the secant method is $1.62^2 \simeq 2.6$ which is better than the convergence rate of Newton's method.

3.5 Systems of nonlinear equations

3.5.1 Introduction

Finding the solutions of a systems of nonlinear equations,

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0, \\ f_2(x_1, x_2, \ldots, x_n) = 0, \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0, \end{cases} \qquad \Leftrightarrow \qquad \boldsymbol{f}(\boldsymbol{x}) = 0,$$

is considerably more difficult that solving a single nonlinear equation or solving a system of linear equations:

1. The system may have *no solution*, like for example

$$\begin{cases} x_1^2 + 2x_1x_2 + x_2^2 = 3, \\ x_1^3 + 3x_1^2x_2 + 3x_1x_2^2 + x_2^3 = 4, \end{cases} \Longleftrightarrow \begin{cases} (x_1 + x_2)^2 = 3, \\ (x_1 + x_2)^3 = 4, \end{cases}$$

   or have *no real solutions*, like for example,

$$\begin{cases} x_1^2 + x_2^2 = -5, \\ x_1^2 - 3\,x_2^2 = 11, \end{cases} \Longleftrightarrow \begin{cases} x_1 = \pm\mathrm{i}, \\ x_2 = \pm 2\mathrm{i}, \end{cases}$$

   or have a *unique solution*

$$\begin{cases} x_1^2 + x_2^2 = 0, \\ \cos(x_1x_2) = 1, \end{cases} \Longleftrightarrow \begin{cases} x_1 = 0, \\ x_2 = 0, \end{cases}$$

   or have *many solutions*

$$\begin{cases} x_1^2 + x_2^2 = 1, \\ \cos[\pi(x_1^2 + x_2^2)] + x_1^2 + x_2^2 = 0, \end{cases} \Longleftrightarrow \text{all } x_1 \text{ and } x_2 \text{ that belong to the circle } x_1^2 + x_2^2 = 1.$$

2. If the system is large, even the existence of a solution is quite unclear.

3. For many real world problems the methods used to find the solutions can be rather ad hoc.

3.5.2 Contraction mapping

One can extend to more than one dimension the theorems on contraction mapping that have been discussed so far. The contraction map is now a set of n nonlinear functions in n variables,

$$\boldsymbol{x}_{n+1} = \boldsymbol{g}(\boldsymbol{x}_n).$$

Essentially everything carries over by replacing scalars with vectors and absolute values with norms.

We define an *interval* in $\mathbb{R}^n$ as a set $I = \{\boldsymbol{x} \in \mathbb{R}^n | \, a_j < x_j < b_j, \, j = 1, 2, \dots, n\}$ where a_j and b_j are given constants. A *contraction map* in $\mathbb{R}^n$ is defined as:

Definition - A continuous map $\boldsymbol{g}(\boldsymbol{x})$ from an interval $I \subseteq \mathbb{R}^n$ into $\mathbb{R}^n$ is *contracting* if

1. the image of I is contained in I:

$$\boldsymbol{g}(I) \subseteq I \quad \Leftrightarrow \quad \boldsymbol{g}(x) \in I \quad \forall \boldsymbol{x} \in I.$$

2. the function $\boldsymbol{g}(\boldsymbol{x})$ is Lipschitz continuous in I with Lipschitz constant $L < 1$:

$$\|\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{y})\| \leq L \|\boldsymbol{x} - \boldsymbol{y}\| \quad \forall \boldsymbol{x}, \boldsymbol{y} \in I.$$

The convergence theorems are modified as follows:

Theorem 3.5.1 *If the function $\boldsymbol{g}(\boldsymbol{x})$ is continuous in $I \subseteq \mathbb{R}^n$ and $\boldsymbol{g}(I) \subseteq I$, then $\boldsymbol{g}(\boldsymbol{x})$ has at least one fixed point in I.*

Theorem 3.5.2 (Contraction mapping theorem in $\mathbb{R}^n$) *If $\boldsymbol{g}(\boldsymbol{x})$ is a contraction mapping in an interval $I \subseteq \mathbb{R}^n$ then there exists one and only one fixed point of the map in I.*

Theorem 3.5.3 *If $\boldsymbol{g}(\boldsymbol{x})$ is a differentiable contraction mapping in an interval $I \subseteq \mathbb{R}^n$, i.e.*

$$\left| \frac{\partial g_i}{\partial x_j} \right| \leq \frac{L}{n}, \quad \forall \boldsymbol{x} \in I, \quad \forall i, j \quad L < 1,$$

then there exists one and only one fixed point of the map in I.

Remark - The condition on the derivative can be relaxed somewhat. For example the theorem holds if $\|J(\boldsymbol{x})\|_\infty \leq L$, where $J(\boldsymbol{x})$ is the Jacobian matrix of the map $\boldsymbol{g}(\boldsymbol{x})$.

Theorem 3.5.4 *Let $I \subseteq \mathbb{R}^n$ be an interval in $\mathbb{R}^n$ and suppose that $\boldsymbol{g}(\boldsymbol{x})$ is a contraction mapping in I with Lipschitz constant $L < 1.$. Then for arbitrary $\boldsymbol{x}_0 \in I$, the sequence $\boldsymbol{x}_n = \boldsymbol{g}(\boldsymbol{x}_{n-1})$, $n = 1, 2, \ldots$ converges to the unique fixed point, $\boldsymbol{s}$, of the map. Moreover, if the error $\boldsymbol{e}_n$ at the n-th stage is defined by $\boldsymbol{e}_n = \boldsymbol{x}_n - \boldsymbol{s}$ then*

$$\|\boldsymbol{e}_n\|_\infty \leq \frac{L^n}{1-L}\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|_\infty.$$

Exercise - Show that

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{cases} g_1(x_1, x_2, x_3) = \dfrac{1}{3}\cos(x_2 x_3) + \dfrac{1}{6}, \\ g_2(x_1, x_2, x_3) = \dfrac{1}{9}\sqrt{x_1^2 + \sin(x_3) + 1.06} - 0.1, \\ g_3(x_1, x_2, x_3) = -\dfrac{1}{20}e^{-x_1 x_2} - \left(\dfrac{10\pi - 3}{60}\right), \end{cases} \tag{3.36}$$

satisfies all the conditions of theorem 3.5.3 in $-1 < x_i < 1$.

Remark 1 - In practice it may be rather tough to prove that $g(I) \subseteq I$.

Remark 2 - There is a “Gauss-Seidel” version of this method, where each iterate is used as soon as it becomes available.

Remark 3 - The analogy with linear systems extends to the S.O.R. method. It is often hard to get the direct iteration to converge. However, the convergence can be helped by using “relaxation” (the opposite of S.O.R.). This method introduces an “under-relaxation” parameter $\omega < 1$ in the iteration. Assuming that we know the iterate $\boldsymbol{x}_n$ we compute a first estimate of the iterate $n+1$, $\hat{\boldsymbol{x}}_{n+1}$ using the iteration map $\boldsymbol{g}(\boldsymbol{x})$:

$$\hat{\boldsymbol{x}}_{n+1} = \boldsymbol{g}(\boldsymbol{x}_n).$$

Use this estimate to obtain $\boldsymbol{x}_{n+1}$ according to

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \omega(\hat{\boldsymbol{x}}_{n+1} - \boldsymbol{x}_n).$$

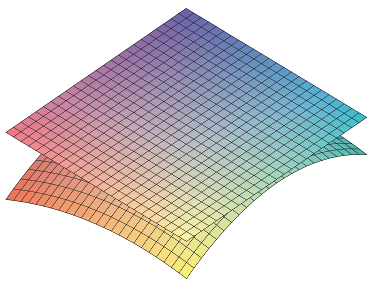
This procedure effectively multiplies the derivatives of the map $\boldsymbol{g}(\boldsymbol{x})$ by $\omega < 1$. If $\omega = 1$ this procedure reduces to the standard contraction mapping.

The drawback of this method is that sometimes ω has to be made so small that convergence is slow and too many iterations are needed.

3.5.3 Newton’s method

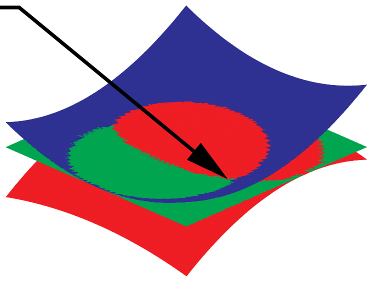
In discussing Newton’s method we shall use only 2×2 systems, but most of what we say generalises immediately to $n \times n$ systems. The model problem that we

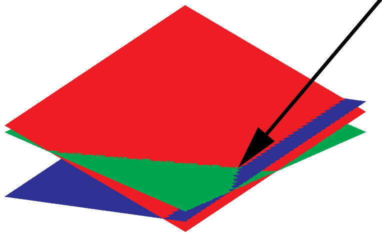
Geometrical interpretation of Newton's method in 2 dimensions



A surface can be approximated in a neighbourhood of a point by the plane tangent to the surface at that point.

The $\boxed{\text{intersection}}$ of the two surfaces with the xy-plane can be approximated by





the $\boxed{\text{intersection}}$ of the planes tangent to the surfaces with the xy-plane.

Figure 3.10: *Geometrical meaning of Newton's method to solve a system of two nonlinear equations in two variables. The solutions of the system are the intersections of the two surfaces with the xy-plane.*

wish to solve is

$$\begin{cases} f_1(x,y) = 0, \\ f_2(x,y) = 0, \end{cases} \qquad \Longleftrightarrow \qquad \boldsymbol{f}(\boldsymbol{x}) = 0.$$

The easiest way to obtain an expression for Newton's method in two dimensions is based on the its geometrical interpretation. We start by recapping the one dimension case: suppose that we wish to solve the equation $f(x) = 0$ and that we have evaluated the function at x_n. The next iterate, x_{n+1} is the intersection of the straight line tangent to the graph of $f(x)$ at $(x_n, f(x_n))$ with the x axis. The equation of this line is given by the first two terms of the Taylor expansion of the function $f(x)$ around x_n:

$$y = f(x_n) + f'(x_n)(x - x_n).$$

It intersects the x axis at the point x_{n+1} such that $y = 0$, i.e.

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n) \implies x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

The geometrical interpretation of Newton's method in two dimensions is that the iterate (x_{n+1}, y_{n+1}) is the common point where the planes tangent to the graph of $f_1(x,y)$ and $f_2(x,y)$ at (x_n, y_n) intersect the xy-plane, i.e. the plane $z = 0$ (see Figure 3.10). The plane tangent to the two surfaces are

$$\begin{aligned} z &= f_1(x_n, y_n) + (x - x_n)\frac{\partial f_1}{\partial x} + (y - y_n)\frac{\partial f_1}{\partial y}, \\ z &= f_2(x_n, y_n) + (x - x_n)\frac{\partial f_2}{\partial x} + (y - y_n)\frac{\partial f_2}{\partial y}. \end{aligned}$$

At the intersection with the xy-plane we have $z = 0$. Therefore the equations for the next iterate are

$$f_1(x_n, y_n) + (x_{n+1} - x_n)\frac{\partial f_1}{\partial x} + (y_{n+1} - y_n)\frac{\partial f_1}{\partial y} = 0, \tag{3.37}$$

$$f_2(x_n, y_n) + (x_{n+1} - x_n)\frac{\partial f_2}{\partial x} + (y_{n+1} - y_n)\frac{\partial f_2}{\partial y} = 0. \tag{3.38}$$

We introduce the Jacobian of the function $\boldsymbol{f}(\boldsymbol{x})$ at (x_n, y_n),

$$J(x_n, y_n) = \begin{pmatrix} \partial_x f_1(x_n, y_n) & \partial_y f_1(x_n, y_n) \\ \partial_x f_2(x_n, y_n) & \partial_y f_2(x_n, y_n) \end{pmatrix}$$

and write (3.37) and (3.38) in matrix notation as

$$J(x_n, y_n)\begin{pmatrix} x_{n+1} - x_n \\ y_{n+1} - y_n \end{pmatrix} = \begin{pmatrix} -f_1(x_n, y_n) \\ -f_2(x_n, y_n) \end{pmatrix},$$

so that the next iterate of Newton's method is

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - J^{-1} \begin{pmatrix} -f_1(x_n, y_n) \\ -f_2(x_n, y_n) \end{pmatrix}. \tag{3.39}$$

Equation (3.39) generalises to n-dimensions as

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - [J(\boldsymbol{x}_n)]^{-1} \boldsymbol{f}(\boldsymbol{x}_n). \tag{3.40}$$

The scheme (3.40) is numerically not convenient, because it requires the computation of the inverse of the Jacobian. Therefore, one normally defines an additional variable

$$\boldsymbol{z} = \boldsymbol{x}_{n+1} - \boldsymbol{x}_n \implies J\boldsymbol{z} = -\boldsymbol{f}(\boldsymbol{x}_n),$$

and solves the linear problem for $\boldsymbol{z}$ using, for example, Gauss elimination with pivoting. Once $\boldsymbol{z}$ is known we can obtain

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{z}.$$

Remark 1 - The initial guess is absolutely crucial and the method can be very "touchy". However, if we are close enough to the root and the Jacobian is non singular the convergence is quadratic.
Remark 2 - Newton's method is very computationally intensive: it requires the evaluation of n^2 derivatives and the solution of an $n \times n$ linear problem at each iteration. A multidimensional version of the secant method has been developed to reduce the number of computations per step (Broyden algorithm).

Further reading

Topics covered here are also covered in

- Chapter 7 of Linz & Wang, *Exploring Numerical Methods* (QA297 LIN),
- Chapter 3 of Kincaid & Cheney, *Numerical Analysis* (QA297 KIN),
- Chapters 1 and 4 of Süli & Mayers, *An Introduction to Numerical Analysis* (not in library – includes considerably more analysis of the convergence issues).

Chapter 4

Numerical Quadrature

4.1 Introduction

There are many circumstances in mathematical modelling when we need to evaluate an integral, like

$$\int_a^b f(x)\,\mathrm{d}x\,. \tag{4.1}$$

However, the integrals that can be evaluated analytically are few and many integrals of great interest in engineering, like, for example, the error function,

$$\mathrm{Erf}(x) = \int_{-\infty}^{x} e^{-\xi^2}\,\mathrm{d}\xi \tag{4.2}$$

can only be evaluated numerically.

We discuss two major techniques to evaluate integrals (*quadrature*): the first is based on *interpolation*, the second on *Gaussian quadrature*.

4.2 Polynomial interpolation

If we want to work with any function $f(x)$ numerically we must expect to represent it using a finite amount of information. Typically this is done by assuming that we know only the value of the function at a finite set of points, call *nodes*, $\{x_j\}$, with $j = 0, \dots, N$. We denote the function values $f_j \equiv f(x_j)$.

One standard approach is to approximate the function f by another function $g(x)$ which is easy to manipulate and *interpolates* f at the nodes, i.e.

$$g(x_j) = f_j = f(x_j), \quad j = 0, \dots, N. \tag{4.3}$$

We then use g in place of f wherever necessary, as we shall see below.

One standard, simple choice of *interpolating function* $g(x)$ is a polynomial. Given the $N+1$ values f_j at the nodes x_j there is a *unique* polynomial g of order N that interpolates f (as should be clear, as a polynomial of order N has $N+1$ coefficients to be fixed). There are two standard forms of writing this polynomial explicitly. Here we just show the Lagrange form to show the explicit construction is possible. Often the Newton form is used in numerical calculations instead.

First, given the nodes x_j, define the *fundamental polynomials* $\ell_j(x)$ by

$$\ell_j(x) = \prod_{\substack{i=0 \\ i\neq j}}^{N} \frac{x - x_i}{x_j - x_i}, \quad j = 0, 1, \ldots, N. \tag{4.4}$$

These have the property that $\ell_j(x_j) = 1$ and $\ell_j(x_k) = 0, j \neq k$. It follows that

$$g(x) = \sum_{j=0}^{N} f(x_j)\ell_j(x) \tag{4.5}$$

is the polynomial of degree N that interpolates the function $f(x)$ at the nodes x_j.

In what follows we shall see methods for solving quadrature and differential equations based on polynomial interpolation. For these the defining features are the locations of the nodes and the order of the interpolating polynomial. In many cases the formulas will simplify greatly from the general Lagrange form above, so each case is treated separately.

4.3 Numerical integration based on polynomial interpolation

4.3.1 Introduction

We assume that we can evaluate the function at a finite set of points, called *nodes*, $\{x_j\}$, with $j = 0, \ldots, N$ in the interval $[a, b]$ and we want to estimate (4.1). The idea behind all the methods based on polynomial interpolation is that we can replace the function $f(x)$ with a simpler function $g(x)$, i.e. a function whose integral we can evaluate analytically. We require that the function $g(x)$ *interpolates* the function $f(x)$, i.e. that it has the same value as $f(x)$ at the nodes:

$$g(x_j) = f(x_j), \qquad j = 0, 1, \ldots, N. \tag{4.6}$$

The simplest choice of class of functions $g(x)$ are polynomials and the different quadrature interpolation methods are differentiated by the order of the polynomial interpolation. Formulae based on polynomial interpolation with equally spaced nodes go under the generic name of *Newton-Cotes formulae*.

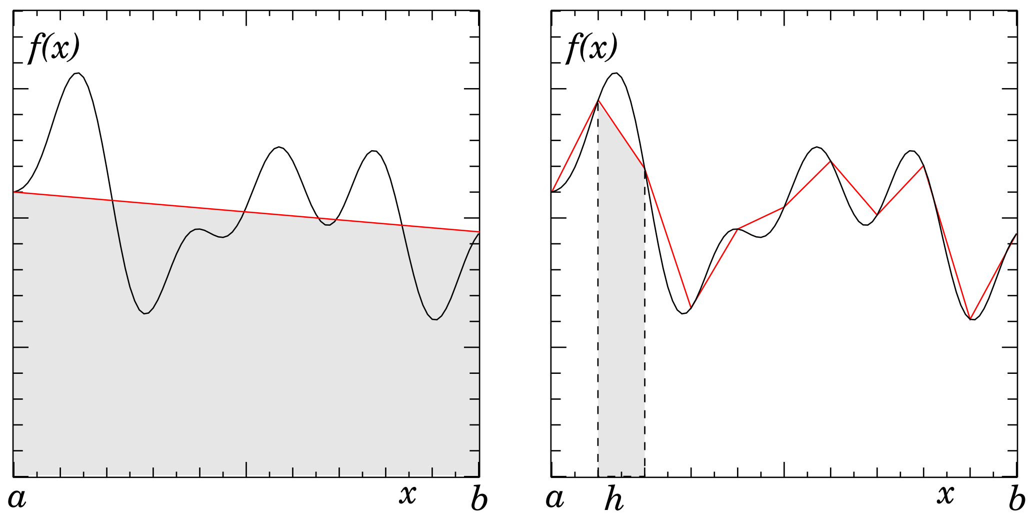


Figure 4.1: *The trapezoidal rule is based on the approximation of the given function $f(x)$ with an order one polynomial (left). In general one uses a composite trapezoidal rule: the integration range is divided in intervals of length h and the function is approximated by a different polynomial on each interval (right).*

4.3.2 Trapezoidal rule

The simplest case results if we choose an interpolating polynomial of order one, i.e. we replace the function with a straight line that interpolates the function at the end points of the integration range (see left panel of Figure 4.1). The area under the line is given by

$$A = \frac{1}{2}(b-a)[f(a)+f(b)] \tag{4.7}$$

and this is the approximate value of (4.1) according to the *trapezoidal rule*. It is clear that unless the range $[a,b]$ is very small the error in the estimate obtained using the trapezoidal rule is very large. The standard procedure is to make use of the *composite trapezoidal rule*: the integration range is divided into N intervals of length h and the function $f(x)$ is approximated on each interval by a straight line that interpolates the function at the nodes (see right panel of Figure 4.1). The area under each segment of the piecewise linear curve is

$$A_j = \frac{1}{2}(x_j - x_{j-1})[f(x_j)+f(x_{j-1})], \quad j = 1,2,\ldots,N. \tag{4.8}$$

By summing all the areas we obtain the composite trapezoidal rule:

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{j=1}^{N} A_j = \frac{1}{2}\sum_{j=1}^{N}(x_j - x_{j-1})[f(x_j)+f(x_{j-1})] \tag{4.9}$$

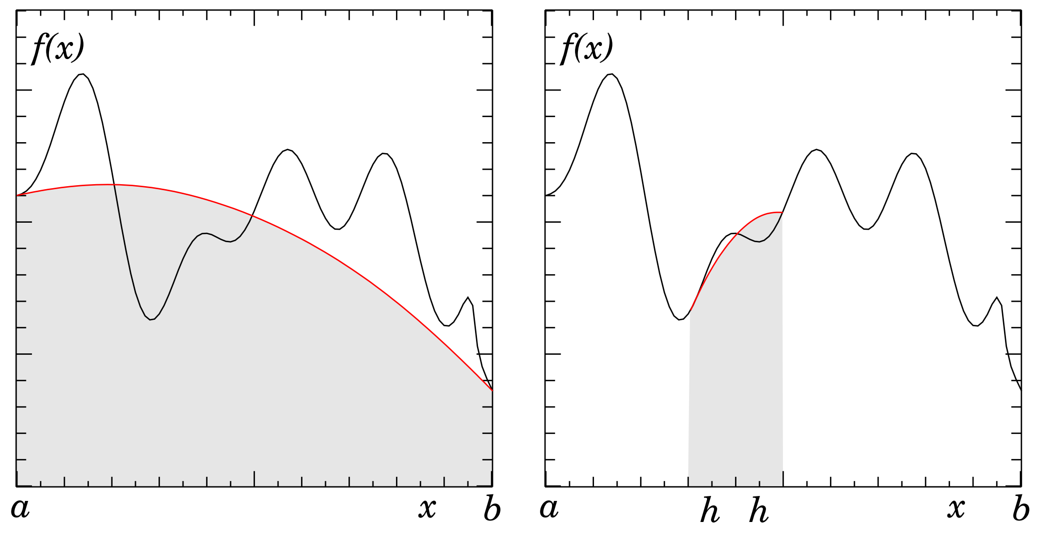


Figure 4.2: *Simpson's rule is based on the approximation of the given function $f(x)$ with an order two polynomial (left). In general one uses a composite Simpson's rule: the integration range is divided in intervals of length $2h$ and the function is approximated by different parabolas on each interval.*

If the nodes are equally spaced, i.e. if $(x_j - x_{j-1}) = h$ for all $j = 1, \ldots, N$, this formula reduces to:

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{2}\sum_{j=1}^{N}[f(x_j) + f(x_{j-1})] = \frac{h}{2}(f_0 + 2f_1 + 2f_2 + \ldots + 2f_{N-1} + f_N), \tag{4.10}$$

where $f_j \equiv f(x_j)$. The error in the approximation is given by

$$\text{Error} \leq \frac{1}{12}h^3 N M_2 = \frac{(b-a)h^2}{12}M_2, \qquad M_2 = \max_{x\in[a,b]}|f''(x)|. \tag{4.11}$$

4.3.3 Simpson's rule

In this case the function is approximated with a polynomial of order two (a parabola, see left panel of Figure 4.2). We can obtain the formula for the quadrature by writing the interpolating polynomial and integrating it. It turns out that the area under the parabola is given by

$$A = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right], \tag{4.12}$$

and this is the approximate value of (4.1) according to *Simpson's rule*. It is clear that unless the range $[a, b]$ is very small the error in the estimate obtained using the trapezoidal rule is very large. The standard procedure is to make use of the *composite Simpson's rule*: the integration range is divided into $N/2$ intervals of length $2h$, with N an even number. The function $f(x)$ is approximated on each interval by a parabola that interpolates the function at the nodes (see right panel of Figure 4.2). For simplicity we assume that the interpolation points $\{x_j\}$, $j = 0, 1, 2, \ldots, N$ are all equally spaced so that we can write:

$$x_j = x_0 + jh.$$

The area under the parabola joining the nodes x_{j-1}, x_j and x_{j+1} is

$$A_j = \frac{h}{3}\left[f(x_{j-1}) + 4f(x_j) + f(x_{j+1})\right] \tag{4.13}$$

where $j = 1, 3, 5, \ldots, N-1$. This result is known as ***Simpson's rule***. If we apply it to all the sub-intervals in $[a, b]$ we obtain the composite Simpson rule:

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{k=1}^{N/2} A_{2k-1} = \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{N/2-1} f(x_{2j}) + 4\sum_{j=1}^{N/2} f(x_{2j-1}) + f(b)\right] + R, \tag{4.14}$$

where the estimate for the error is given by

$$|R| \le \frac{M_4}{180}(b-a)h^4, \tag{4.15}$$

where $M_4 = \max|f^{(4)}(x)|$ with $a \le x \le b$. Note that Simpson's rule is exact for any cubic polynomial as $f^{(4)}(x) \equiv 0$ for all polynomials of order three.

We can obtain an estimate of the error by proceeding as follows. We compare the Taylor expansion of Simpson's rule for the integral in the range $[x_{j-1}, x_{j+1}]$, given by equation (4.13), to the Taylor expansion of the exact integral. The error is the first term in the difference of the two Taylor expansions. We start with the Taylor expansion of the Simpson's rule estimate of

$$\int_{x_{j-1}}^{x_{j+1}} f(x)\,\mathrm{d}x \simeq A_j = \frac{h}{3}[f(x_{j-1}) + 4f(x_j) + f(x_{j+1})], \tag{4.16}$$

where, to simplify the notation, we use the notation $f_j = f(x_j)$, $f_j' = f'(x_j)$, and, in general, $f_j^{(n)}$ for the n-th derivative of $f(x)$ evaluated at $x = x_j$. Using Taylor's theorem to expand $f(x)$ at $x = x_j$ we can write

$$A_j = 2hf_j + \frac{h^3}{3}f_j'' + \frac{h^5}{36}f_j^{(4)} + \mathcal{O}(h^6) \tag{4.17}$$

We now need to compute the Taylor expansion in powers of h of the exact result. In order to do this we introduce the function

$$F(t) = \int_{x_j-t}^{x_j+t} f(x)\,\mathrm{d}x\,. \tag{4.18}$$

Note that with this notation

$$\int_{x_{j-1}}^{x_{j+1}} f(x)\,\mathrm{d}x = F(h). \tag{4.19}$$

Therefore, in order to compute the Taylor expansion in powers of h of the exact result of the integral in equation (4.19), we need to expand $F(t)$ in Taylor series around $t = 0$:

$$F(h) = F(0) + h\,\frac{\mathrm{d}F}{\mathrm{d}t}\bigg|_{t=0} + \frac{h^2}{2}\,\frac{\mathrm{d}^2F}{\mathrm{d}t^2}\bigg|_{t=0} + \ldots + \frac{h^5}{5!}\,\frac{\mathrm{d}^5F}{\mathrm{d}t^5}\bigg|_{t=0} + \mathcal{O}\big(h^6\big) \tag{4.20}$$

We note that $F(0) = 0$. Moreover, by the fundamental theorem of calculus (with a little help from the chain rule)

$$\frac{\mathrm{d}F}{\mathrm{d}t} = f(x_j+t)+f(x_j-t) \implies \frac{\mathrm{d}^nF}{\mathrm{d}d^n} = f^{(n-1)}(x_j+t)+(-1)^{n+1}f^{(n-1)}(x_j-t).$$

Note that all the even derivatives of $F(t)$ are zero at $t = 0$ so that the Taylor expansion of (4.20) becomes

$$2hf_j + \frac{h^3}{3}f_j'' + \frac{h^5}{60}f_j^{(4)} + \mathcal{O}\big(h^6\big). \tag{4.21}$$

Combining these two expansions, we have that the local error in the interval $[x_{j-1}, x_{j+1}]$ is, up to order h^6,

$$E_j \equiv F(h) - A_j = -\frac{1}{90}h^5 f_j^{(4)} \implies |E_j| \le \frac{1}{90}h^5 M_4, \tag{4.22}$$

where M_4 is the same as in equation (4.15). Summing the local errors over all the intervals used in the composite Simpson's rule gives the total error, R,

$$|R| = \left|\sum_{k=1}^{N/2} E_{2k-1}\right| \le \sum_{k=1}^{N/2} |E_{2k-1}| \le \sum_{k=1}^{N/2} \frac{1}{90}h^5 M_4 = \frac{M_4}{180}(b-a)h^4. \tag{4.23}$$

4.3.4 Richardson extrapolation

This is a very general technique that uses the information on the rate of decrease of the error to get a better estimate of the numerical result required. As such it can be applied to many different algorithms, not just quadratures. However, here we illustrate it in the context of Simpson's rule.

Equation (4.15) tells us that the error on the estimate of the integral decreases with the fourth power of the integration step, h. However, in order to estimate the error using this formula we would also need the value of the coefficient M_4. We can get round this problem by using the following procedure. Indicate with I_n the estimate of the integral using n intervals and with I the exact value of the integral. From (4.15) we have that

$$I - I_{2n} \simeq C(2n)^{-4} = 2^{-4}\left(Cn^{-4}\right) \simeq 2^{-4}(I - I_n). \tag{4.24}$$

In this last step we have assumed that the constant C is the same whether we use n or $2n$ intervals to estimate the integral. This not strictly true, but it is a reasonable approximation if the integration range is small enough. From (4.24) we have that the exact value of the integral is approximately

$$I \simeq \frac{2^4 I_{2n} - I_n}{2^4 - 1}$$

and we use this as the new estimate of the integral (*Richardson extrapolated value*):

$$R_{2n} = \frac{2^4 I_{2n} - I_n}{2^4 - 1}. \tag{4.25}$$

The error in the approximation is roughly

$$E_{2n} \equiv |R_{2n} - I_{2n}| = \frac{|I_n - I_{2n}|}{2^4 - 1}. \tag{4.26}$$

This quantity is called the *computable estimate of the error*.

4.3.5 Adaptive quadrature

Adaptive quadrature methods are intended to compute definite integrals to a given precision by automatically choosing a set of nodes that takes into account the behaviour of the integrand by being denser where the integrand varies more rapidly. Ideally, the user supplies only the integrand f, the interval $[a, b]$, and the accuracy ϵ desired for computing the integral

$$\int_a^b f(x)\,\mathrm{d}x\,. \tag{4.27}$$

The program then divides the interval into sub-intervals of varying length so that numerical integration on these sub-intervals will produce results of acceptable precision. The main idea is that if Simpson's rule on a given sub-interval is not sufficiently accurate, that interval will be divided into two equal parts, and Simpson's rule will be used on each half. This procedure will be repeated in an effort to obtain an approximation to the integral with the same accuracy over all the sub-intervals involved. A rough algorithm is as follows:

1. Compute the integral (4.27) using 3 points and estimate the global error. If the error is smaller than the tolerance stop.
2. Divide the interval into two equal parts. Consider each part in turn.
3. Compute the integral on the sub-interval using 3 points, i.e. using a finer grid.
4. Estimate the error on the evaluation of the integral over the sub-interval. If it is larger than the maximum acceptable local error then divide the sub-interval in two and start again from point (2). Otherwise go to point (3) and work on the next sub-interval if there are any left.

In order to apply this idea we need to firstly to estimate the *local error*: we can do this using Richardson's extrapolation and equation (4.26). Secondly, we need to estimate the *global error* on the integration over the entire interval $[a, b]$ as a function of the local errors.

We indicate with e_i the local error on the integration over the segment $[x_{i-1}, x_i]$. If

$$|e_i| \leq \epsilon(x_i - x_{i-1})/(b-a), \tag{4.28}$$

then the total error will be bounded by

$$\left|\sum_{i=1}^{n} e_i\right| \leq \sum_{i=1}^{n} |e_i| \leq \frac{\epsilon}{b-a}\sum_{i=1}^{n}(x_i - x_{i-1}) = \epsilon. \tag{4.29}$$

Therefore in the adaptive quadrature algorithm outlined above we have to ensure that the local error satisfies (4.28).

4.4 Gaussian Quadrature

The aim of all quadrature techniques is to create formulae of the type

$$\int_a^b f(x)\,\mathrm{d}x \simeq \sum_{i=1}^{n} w_i f(x_i). \tag{4.30}$$

In the trapezoidal and Simpson's rule the points where the function is evaluated, x_i, are fixed *a priori* and we obtain that (4.30) is exact for all polynomial of degree less than or equal to n. In the case of the trapezoidal rule we have $n = 2$ and the rule is exact for all linear functions. In the case of Simpson's rule we have $n = 3$ and we would expect the formula to be exact for all quadratic polynomials. As a matter of fact, it is exact for polynomials of order three, but this is due to a happy cancellation.

However, there is no obligation to fix the nodes. As a matter of fact we could fix the coefficients (called *weights*) w_i, for example set them all to one, or we could not fix anything. The general criterion to find nodes and weights is to require that equation (4.30) is as accurate as possible, i.e. that it is exact for polynomials of degree as high as possible. In other words, we require that the nodes and weights are such that

$$\int_a^b x^s \,\mathrm{d}x = \sum_{i=1}^{n} w_i x_i^s, \quad s = 0, 1, ..., N, \tag{4.31}$$

with N as large as possible.

As above, we define the *fundamental polynomials* $\ell_i(x)$ given by

$$\ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{N} \frac{x - x_j}{x_i - x_j}, \quad i = 0, 1, \ldots, N, \tag{4.32}$$

have the property that $\ell_i(x_i) = 1$ and $\ell_i(x_k) = 0, i \neq k$. It follows that

$$p(x) = \sum_{i=0}^{N} f(x_i)\ell_i(x) \tag{4.33}$$

is the polynomial of degree N that interpolates the arbitrary function $f(x)$ at the given nodes x_i. If the nodes are known or fixed we can therefore immediately compute the weights as

$$w_i = \int_a^b \ell_i(x) \,\mathrm{d}x\,. \tag{4.34}$$

If neither the nodes nor the weights are fixed we have N nonlinear equations for $2n$ unknowns $x_1, ..., x_n$ and $w_1, ..., w_n$. One can show that this problem has a unique solution if $N = 2n - 1$, i.e. that it is possible to compute exactly the integrals of polynomials of degree up to $2N - 1$. The resulting algorithm is called a **Gaussian quadrature**. There are many formulae for Gaussian quadrature that depend mainly on the choice of integration interval and on the type of the integral. A generic Gauss quadrature formula is

$$\int_a^b W(x)f(x) \,\mathrm{d}x = \sum_{i=1}^{n} w_i f(x_i) \tag{4.35}$$

and the different formulae are summarised in the following table:

(a, b)	$W(x)$	Gauss-
$(-1, 1)$	1	Legendre
$(-1, 1)$	$(1-x^2)^{-1/2}$	Chebychev
$(0, \infty)$	$x^c e^{-x}$	Laguerre $c = 0, 1, \ldots$
$(-\infty, +\infty)$	e^{-x^2}	Hermite

Further reading

Topics covered here are also covered in

- Chapter 6 of Linz & Wang, *Exploring Numerical Methods* (QA297 LIN),
- Chapter 7 of Kincaid & Cheney, *Numerical Analysis* (QA297 KIN),
- Chapters 7 and 10 of Süli & Mayers, *An Introduction to Numerical Analysis* (not in library).

The issue of representing an arbitrary function by an interpolating function, which will recur in later chapters, is covered in considerable detail elsewhere; for example

- Chapters 4 and 5 of Linz & Wang, *Exploring Numerical Methods* (QA297 LIN),
- Chapter 6 of Kincaid & Cheney, *Numerical Analysis* (QA297 KIN),
- Chapters 6, 8, 9 and 11 of Süli & Mayers, *An Introduction to Numerical Analysis* (not in library).

Chapter 5

Initial value problems for Ordinary Differential Equations

5.1 Introduction

A system of first order differential equation (ODE) is a relationship between an unknown (vectorial) function $\boldsymbol{y}(x)$ and its derivative $\boldsymbol{y}'(x)$. The general system of first order differential equations has the form

$$\boldsymbol{y}'(x) = \boldsymbol{f}(x, \boldsymbol{y}(x)). \tag{5.1}$$

The solution of the differential equation is a function $\boldsymbol{y}(x)$ that satisfies (5.1). Analytic techniques produce a family of solutions and an initial condition of the form

$$\boldsymbol{y}(0) = \boldsymbol{y}_0 \tag{5.2}$$

can be used to determine a member of this family. The differential equations (5.1) and the initial condition (5.2) specify an *initial value problem*. We assume that all the differential equations that we are going to analyse satisfy the conditions of the theorem that guarantees the existence and uniqueness of the solution of an initial value problem.

Remark - If the original formulation of the problem is in the form of second or higher order differential equations, we can always recast it in the form (5.1) by introducing appropriate new variables.

5.2 Numerical differentiation

We can make use of Taylor's theorem to obtain numerically estimates of the derivative of a function $f(x)$ at a point x_0. Indicate with h a small step in the variable x. By Taylor's theorem we have that:

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) + \mathcal{O}\big(h^4\big). \tag{5.3}$$

Solving for $f'(x_0)$ we obtain the *forward difference estimate of the derivative*:

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} + \mathcal{O}(h). \tag{5.4}$$

Remark 1 - In other words the slope of the tangent to $f(x)$ at x_0 is approximated with that of the line through $[x_0, f(x_0)]$ and $[x_0+h, f(x_0+h)]$.
Remark 2 - The symbol $\mathcal{O}(h)$ on the far right of equation (5.4) indicates that the error of this estimate decreases linearly with the step size h.

Another estimate of the derivative can be obtained by taking a step backward:

$$f(x_0-h) = f(x_0) - hf'(x_0) + \frac{h^2}{2}f''(x_0) - \frac{h^3}{6}f'''(x_0) + \mathcal{O}\big(h^4\big). \tag{5.5}$$

Solving for $f'(x_0)$ we obtain the *backward difference estimate of the derivative*:

$$f'(x_0) = \frac{f(x_0) - f(x_0-h)}{h} + \mathcal{O}(h). \tag{5.6}$$

Both the forward and the backward estimate of the derivative are order one methods, in the sense that the error in the approximation decreases with the first power of the step size h. It is possible to obtain a higher order method by using a more symmetric formula: subtracting (5.5) from (5.3) we obtain the *central difference estimate of the derivative*:

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} + \mathcal{O}\big(h^2\big). \tag{5.7}$$

Remark 1 - The geometrical interpretation of this estimate is that the slope of the tangent to $f(x)$ at x_0 is well approximated by the slope of the secant through $[x_0-h, f(x_0-h)]$ and $[x_0+h, f(x_0+h)]$.
Remark 2 - An intuitive explanation of why the central difference is more accurate than either the forward and the backward difference is that it contains information on the function on <u>both</u> sides of the point where the derivative is to be estimated.
Exercise - Obtain an accurate estimate of $f''(x_0)$.

5.3 Errors

All procedures to solve numerically an initial value problem consist of transforming the continuous differential equation (5.1) into a discrete iteration procedure that starting from the initial condition (5.2) returns the values of the dependent variables $\boldsymbol{y}(x)$ at points $x_n = x_0 + nh$, with h a small number called the *discretisation step*.

In this discretisation and iteration procedure several types of errors arise. These are classified as follows:

1. Local truncation error
2. Local roundoff error
3. Global truncation error
4. Global roundoff error
5. Total error

The **roundoff error** is caused by the limited precision of computers. The **global roundoff error** is the accumulation of the local roundoff errors. The **total error** is the sum of the global truncation error and the global roundoff error.

The **local truncation error** is the error made in one step when we replace a continuous process (like a derivative) with a discrete one (like the numerical estimate of the derivative using forward differences). The local truncation error is inherent in any algorithm. The accumulation of all the local errors in the process of an iterative procedure (like those used to integrate a differential equation) give rise to the **global truncation** error. Again, this error will be present even if all calculations are performed using exact arithmetic. If the local truncation errors are $\mathcal{O}(h^{n+1})$, where h is the discretisation step used in evaluating the derivatives, then the global truncation error must be $\mathcal{O}(h^n)$ because the number of steps necessary to reach an arbitrary point x_f, having started at x_0, is $(x_f - x_0)/h$. We can proceed more formally to establish a better bound on the global truncation error and, more importantly, to understand better its relation with the original differential equation. For simplicity we consider only a single first order differential equation instead of a system like (5.1). Moreover, we assume that no roundoff error is involved.

Consider the initial value problem

$$y' = f(x, y), \qquad y(0) = s, \qquad 0 \leq x \leq X > 0. \tag{5.8}$$

The difference $y(x_n) - y_n$ is the global truncation error. This is not simply the sum of all local truncation errors that entered at the previous points. The key point is to understand how two solutions differ at any point if they are started with different initial conditions as each step in the numerical solution must use as its initial value the approximated ordinate computed at the preceding step.

We assume that $f_y = \frac{\partial f}{\partial y}$ is continuous and satisfies the condition $f_y \leq \lambda$ for $0 \leq x \leq X > 0$. The solution is $y = y(x, s)$. We would like to know how the solution depends on s. Define

$$u(x) = \frac{\partial y}{\partial s}\,. \tag{5.9}$$

We can obtain a differential equation - the variational equation - for u by differentiating with respect to s in the initial value problem (5.8) to get

$$u'(x) = \frac{\partial u}{\partial x} = \frac{\partial y'}{\partial s} = \frac{\partial f}{\partial s} = f_y(x, y)u, \qquad u(0) = 1, \qquad 0 \leq x \leq X > 0. \tag{5.10}$$

Note that if $f_y \leq \lambda$ for $0 \leq x \leq X > 0$, then the solution of the variational equation satisfies the inequality

$$|u(x)| \leq e^{\lambda x}, \qquad 0 \leq x \leq X > 0. \tag{5.11}$$

Proof: we have

$$u'/u = f_y = \lambda - \alpha(x), \qquad \alpha(x) \geq 0. \tag{5.12}$$

Integrating this inequality, we obtain

$$\log|u| = \lambda x - \int_0^x \alpha(\tau)\,\mathrm{d}\tau\,, \implies |u(x)| = e^{\lambda x} - \int_0^x \alpha(\tau)\,\mathrm{d}\tau \leq e^{\lambda x}. \tag{5.13}$$

The last inequality is justified because

$$\int_0^x \alpha(\tau)\,\mathrm{d}\tau \geq 0. \tag{5.14}$$

Using this inequality, it is easy to show that if the initial value problem is solved with two initial values s and $s + \delta$, the solutions differ at x by at most $|\delta|e^{\lambda x}$, as

$$|y(x, s) - y(x, s+\delta)| = |\delta| \left|\frac{\partial}{\partial s} y(x, s + \theta\delta)\right| = |u(x)||\delta| \leq e^{\lambda x}|\delta|, \qquad 0 < \theta < 1. \tag{5.15}$$

Global Error Theorem: if all local truncation errors $\delta_1, \delta_2, ..., \delta_n$ do not exceed δ in magnitude, then the global truncation error does not exceed

$$\delta \frac{1 - e^{n\lambda h}}{1 - e^{\lambda h}}. \tag{5.16}$$

Proof: In computing y_1 there was an error $|\delta_1|$. In computing y_2 the global error is

$$|\delta_1|e^{\lambda h} + |\delta_2|, \tag{5.17}$$

where the first term in the right-hand side is the error in the initial condition, and the second term is the new truncation error. In computing y_3 the global error is

$$(|\delta_1|e^{\lambda h} + |\delta_2|)e^{\lambda h} + |\delta_3|, \tag{5.18}$$

and so on. Finally, if $|\delta_i| \le \delta$, $i = 1, 2, ..., n$, we obtain for the global truncation error

$$|\delta_1|e^{n\lambda h} + |\delta_2|e^{(n-1)\lambda h} + \ldots + |\delta_n| \le \delta \frac{1 - e^{n\lambda h}}{1 - e^{\lambda h}}. \tag{5.19}$$

As a consequence of this theorem, if all local truncation errors $\delta_i = \mathcal{O}(h^{n+1}))$, then the global truncation error is $\mathcal{O}(h^n)$. In fact the numerator of the fraction is either order one or of the order of $\exp[\lambda(b - a)]$. The denominator is of the order of h, assuming that $\lambda h \ll 1$. If the global truncation error is $\mathcal{O}(h^n)$ the method is said to be of **order n**.

5.4 Euler's methods

In what follows, we assume that equation (5.1) refers to a single variable, $y(x)$. This makes the notation lighter: however, all the results obtained can be extended to systems of first ordinary differential equations.

We assume that we want to solve the initial value problem

$$y' = f(x, y), \qquad y(x_0) = y_0. \tag{5.20}$$

We introduce a step h and we first obtain an estimate of $y(x)$ at $x_1 = x_0 + h$ using Taylor's theorem, exactly as we had done for the forward difference estimate of the derivative. We obtain

$$y(x_1) \equiv y(x_0+h) = y(x_0)+y'(x_0)h+\mathcal{O}(h^2) = y(x_0)+hf(x_0, y(x_0))+\mathcal{O}(h^2). \tag{5.21}$$

By analogy we obtain that the value y_n of the function at the point $x_n = x_0 + nh$ is given by

$$y_{n+1} \equiv y(x_{n+1}) = y_n + hf(x_n, y_n) + \mathcal{O}(h^2). \tag{5.22}$$

This iteration scheme to estimate the solution of the initial value problem (5.20) at the points x_n is called *Euler's method*. This method is extremely simple, but very inaccurate and potentially unstable. It is reliable only if an extremely small step h is used. Its geometrical interpretation is that we use the slope of the function

$y(x_n)$ at the beginning of the interval $[x_n, x_{n+1}]$ to estimate the value of $y(x_{n+1})$. This suggests that we can obtain a better estimate using an “average” of the slope over the same interval, i.e. if we could compute

$$y_{n+1} = y_n + h\frac{f(x_n, y_n) + f(x_{n+1}, y_{n+1})}{2}. \tag{5.23}$$

However, we do not know y_{n+1} and so we cannot use this relation as it stands. However, we can use equation (5.21) to estimate the value of $y(x)$ at x_{n+1} and use this value in equation (5.23) to obtain a refined estimate. This procedure is called the Euler predictor-corrector method and can be summarised as:

$$\begin{aligned} y_{n+1}^{(p)} &= y_n + hf(x_n, y_n) && \text{Predictor step} \\ y_{n+1} &= y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(p)})\right] && \text{Corrector step.} \end{aligned} \tag{5.24}$$

We can show that the predictor-corrector method is of order $\mathcal{O}(h^3)$ by comparing the Taylor expansion of (5.20) with that of (5.24). However, we will not do this here as it is exactly the same procedure that is used to find the coefficients of the second order Runge-Kutta method (shown below).

On the other hand, we can find the error for the modified Euler method by writing the Taylor expansion of equation (5.20):

$$y_{n+1} = y_n + y_n'h + \frac{1}{2}y_n''h^2 + \mathcal{O}(h^3). \tag{5.25}$$

Replacing the second derivative by the forward-difference approximation, we obtain

$$y_{n+1} = y_n + y_n'h + \frac{1}{2}h^2\frac{y_{n+1}' - y_n'}{h} + \mathcal{O}(h^3), \tag{5.26}$$

and hence,

$$y_{n+1} = y_n + \frac{h}{2}(y_{n+1}' + y_n') + \mathcal{O}(h^3). \tag{5.27}$$

This shows that the error of one step of the modified Euler method is $\mathcal{O}(h^3)$. This is the *local error*. There is an accumulation of local errors from step to step, so that the error over the whole range of application, the *global error*, is $\mathcal{O}(h^2)$.

5.5 Runge-Kutta Methods

The Euler method is not very accurate. Much greater accuracy can be obtained more efficiently using a group of methods named after two German mathematicians, Runge and Kutta. The idea behind these methods is to match the Taylor expansion of $y(x)$ at $x = x_n$ up to the highest possible and/or convenient order.

As an example, let us consider the derivation of the second order method. Here, the increment to y is a weighted average of two estimates which we call k_1 and k_2. Thus for the equation

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x,y), \tag{5.28}$$

we have

$$y_{n+1} = y_n + ak_1 + bk_2, \tag{5.29}$$
$$k_1 = hf(x_n, y_n), \tag{5.30}$$
$$k_2 = hf(x_n + \alpha h, y_n + \beta k_1). \tag{5.31}$$

We fix the four parameter a, b, α and β so that (5.29) agrees as well as possible with the Taylor series expansion of the differential equation (5.28)

$$\begin{aligned} y_{n+1} &= y_n + hy'_n + \frac{h^2}{2}y''_n + \dots \\ &= y_n + hf(x_n,y_n) + \frac{h^2}{2}\frac{\mathrm{d}}{\mathrm{d}x}f(x_n,y_n) + \dots, \\ &= y_n + hf_n + h^2\left(\frac{1}{2}f_x + \frac{1}{2}f_y f_n\right) + \dots \end{aligned} \tag{5.32}$$

where $f_n \equiv f(x_n, y_n)$. On the other hand, using (5.29) we have

$$y_{n+1} = y_n + ahf_n + bhf[x_n + \alpha h, y_n + \beta hf_n]. \tag{5.33}$$

Expand the right-hand side of (5.33) in a Taylor series in terms of x_n, y_n

$$y_{n+1} = y_n + ahf_n + bh\left[f_n + f_x(x_n,y_n)\alpha h + f_y(x_n,y_n)f(x_n,y_n)\beta h\right], \tag{5.34}$$

or, rearranging,

$$y_{n+1} = y_n + h(a+b)f_n + h^2\left[f_x(x_n,y_n)\alpha b + f_y(x_n,y_n)f(x_n,y_n)\beta b\right]. \tag{5.35}$$

This result is identical to the Taylor series expansion (5.32) if

$$a + b = 1, \quad \alpha b = \frac{1}{2}, \quad \beta b = \frac{1}{2}. \tag{5.36}$$

Note that there are only three equations to be satisfied by the four unknowns. We can therefore assign an arbitrary value to one of the unknowns. For example, if we take $a = b = \frac{1}{2}$, and $\alpha = \beta = 1$, we obtain the Euler predictor-corrector method.

Fourth-order Runge-Kutta methods are the most widely used and are derived in a similar way. The most commonly used set of values leads to the algorithm

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{5.37}$$

with

$$\begin{array}{ll} k_1 = hf(x_n, y_n), & k_2 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1), \\ k_3 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2), & k_4 = hf(x_n + h, y_n + k_3). \end{array} \tag{5.38}$$

The local error term for the fourth-order Runge-Kutta methods is $\mathcal{O}(h^5)$; the global error is $\mathcal{O}(h^4)$.

5.6 Error in Runge-Kutta methods

Whatever the method used one needs to check whether the results of the integration are reliable, i.e. one would like an estimate of the local truncation error. Moreover, the knowledge of this quantity would allow us to use a variable step in integrating the differential equation: if the local error is much smaller than a predefined threshold then the step size can be increased. If it is larger the step size should be decreased.

Call $y_e(x_0+h)$ the exact value of the solution $y(x)$ of the initial value problem

$$y'(x) = f(x, y), \qquad y(x_0) = y_0. \tag{5.39}$$

and indicate with $y_h(x_0 + h)$ the solution obtained using a fourth order Runge-Kutta method with step h. By construction we have that

$$|y_e(x_0 + h) - y_h(x_0 + h)| = Ch^5. \tag{5.40}$$

Here C is a number independent of h but dependent on x_0 and on the function y_e. To estimate Ch^5 and, hence, the local error we assume that C does not change as x changes from x_0 to $x_0 + h$ (a similar procedure is used in Richardson's extrapolation). Let $y_{h/2}(x_0 + h)$ be the solution obtained using two steps of length $h/2$ of the fourth order Runge-Kutta method. By assumption we have that

$$\begin{aligned} y_e(x_0 + h) &= y_h(x_0 + h) + Ch^5, \\ y_e(x_0 + h) &= y_{h/2}(x_0 + h) + 2C(h/2)^5. \end{aligned} \tag{5.41}$$

By subtraction we obtain from these two equations that

$$\text{Local truncation error} = Ch^5 = \frac{y_h - y_{h/2}}{1 - 2^{-4}}. \tag{5.42}$$

Thus the local truncation error is approximately $y_h - y_{h/2}$.

This estimate of the local error is rather expensive if used in a variable step algorithm, because it requires two integrations to be run at the same time for a total of 12 function evaluations.

A second and more efficient method is to compare the result of the fourth order with those of a fifth order Runge-Kutta method. As we have seen in the derivation of the Runge-Kutta method of order 2, a number of parameters must be selected. A similar selection process occurs in establishing higher order Runge-Kutta methods. Consequently, there is not just one Runge-Kutta method of each order, but a family of methods. As shown in the following table, the number of required function evaluations increases more rapidly than the order of the Runge-Kutta methods:

Number of function evaluations	1	2	3	4	5	6	7	8
Maximum order of Runge-Kutta method	1	2	3	4	4	5	6	6

This makes the higher-order Runge-Kutta methods less attractive than the fourth order method, since they are more expensive to use. However, Fehlberg (1969) devised a fourth order Runge-Kutta method that required five function evaluation and a fifth order Runge-Kutta method that required six function evaluations, five of which were the same as those used for the fourth-order method. Therefore, with only six function evaluations we have a fourth-order Runge-Kutta method with estimate of the local error.

5.7 Multistep Methods

5.7.1 Introduction

The modified Euler method and Runge-Kutta methods for solving initial value problem are single-step methods since they do not use any knowledge of prior values of $y(x)$ when the solution is being advanced from x and $x+h$. If $x_0, x_1, ..., x_n$ are steps along the $x-$axis, then y_{n+1} depends only on y_n, and knowledge of $y_{n-1}, ..., y_0$ is not used.

It is reasonable to expect that more accurate results may be obtained if previous values of $y(x)$ were used when estimating y_{n+1}. This idea is at the heart of multi-step methods. This principle is transformed into an algorithm by writing the solution of the initial value problem,

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y), \quad y(x_0) = y_0, \tag{5.43}$$

as

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x, y(x))\,\mathrm{d}x\,. \tag{5.44}$$

The integral on the right can be approximated by a numerical quadrature formula that depends on $\{x_{n-j}, y_{n-j}\}$ and the result will be a formula for generating the approximate solution step by step.

The general form of a multistep method to solve an initial value problem

$$y' = f(x, y), \qquad y(x_0) = y_0, \tag{5.45}$$

is

$$a_k y_{n+1} + a_{k-1} y_n + \ldots + a_0 y_{n+1-k} = h[b_k f_{n+1} + b_{k-1} f_n + \ldots + b_0 f_{n+1-k}] \tag{5.46}$$

Such an algorithm is called a *k-step method*. The coefficients a_i, b_i are given. As before, y_i denotes an approximation to the solution at $x_i = x_0 + ih$, and $f_i = f(x_i, y_i)$. This formula is used to compute y_{n+1}, assuming that $y_{n+1-k}, y_{n-k}, ..., y_n$ are known. We assume that $a_k \neq 0$. If $b_k = 0$, the method is said to be *explicit*, and y_{n+1} can be computed directly. In the opposite case the method is said to be *implicit*.

5.7.2 Adams-Bashforth formula

An explicit multistep method of the type

$$y_{n+1} = y_n + a f_n + b f_{n-1} + c f_{n-2} + ..., \tag{5.47}$$

where $f_i = f(x_i, y_i)$ belongs to the class of Adam-Bashforth methods. The Adam-Bashforth formula of order 5, based on the equally spaced points $x_i = x_0 + ih$, $i = 0, 1, \ldots, n$ is:

$$y_{n+1} = y_n + \frac{h}{720}[1901 f_n - 2774 f_{n-1} + 2616 f_{n-2} - 1274 f_{n-3} + 251 f_{n-4}]. \tag{5.48}$$

To obtain the coefficients that appear in this equation we start by observing that we wish to approximate the integral

$$\int_{x_n}^{x_{n+1}} f[x, y(x)]\,\mathrm{d}x \approx h[A f_n + B f_{n-1} + C f_{n-2} + D f_{n-3} + E f_{n-4}]. \tag{5.49}$$

The coefficients A, B, C, D, E are determined by requiring that this equation is exact whenever the integrand is a polynomial of degree less than 5. For simplicity we assume that $x_n = 0$, $x_{n-1} = -1$, $x_{n-2} = -2$, $x_{n-3} = -3$, $x_{n-4} = -4$, and $h = 1$. We take as a basis the following five polynomials:

$$\begin{aligned} p_0(x) &= 1, \\ p_1(x) &= x, \\ p_2(x) &= x(x+1), \\ p_3(x) &= x(x+1)(x+2), \\ p_4(x) &= x(x+1)(x+2)(x+3) \end{aligned} \tag{5.50}$$

When these are substituted in the equation

$$\int_0^1 p_m(x)\,\mathrm{d}x \approx Ap_m(0)+Bp_m(-1)+Cp_m(-2)+Dp_m(-3)+Ep_m(-4) \tag{5.51}$$

for $m = 0, 1, 2, 3, 4$, we obtain the system for determination of A, B, C, D, E

$$\begin{cases} A+B+C+D+E = 1, \\ -B-2C-3D-4E = 1/2, \\ 2C+6D+12E = 5/6, \\ -6D-24E = 9/4, \\ 24E = 251/30. \end{cases} \tag{5.52}$$

The coefficients of the Adam-Bashforth formula are obtained by back substitution.

Remark 1 - A special procedure must be employed to start the method since initially only $y_0 \equiv y(x_0)$ is known. A Runge-Kutta method is ideal for obtaining y_1, y_2, y_3 and y_4.

Remark 2 - The method is order five, i.e. the local error is $\mathcal{O}(h^6)$ and the global error is $\mathcal{O}(h^5)$.

5.7.3 Adams-Moulton formula

The Adam-Bashforth formulae are most often used in conjunction with other formulae to enhance their precision. One such scheme can be set up by making use of the implicit version of equation (5.47):

$$y_{n+1} = y_n + af_{n+1} + bf_n + cf_{n-1} + \dots \tag{5.53}$$

Following the same steps as in the derivation of the coefficients of the Adams-Bashforth formula we obtain the *Adams-Moulton formula* of order 5:

$$y_{n+1} = y_n + \frac{h}{720}[251f_{n+1} + 646f_n - 264f_{n-1} + 106f_{n-2} - 19f_{n-3}]. \tag{5.54}$$

This cannot be used directly as y_{n+1} occurs on both sides of the equation. However, we can set up a predictor-corrector algorithm that uses the Adam-Bashforth formula to predict a tentative value for $y_{n+1}^{(p)}$, and then the Adams-Moulton formula to compute a corrected value of y_{n+1} using $y_{n+1}^{(p)}$ on the right hand side of (5.54). In other words, in (5.54) we evaluate f_{n+1} as $f_{n+1}(x_{n+1}, y_{n+1}^{(p)})$ using the predicted value $y_{n+1}^{(p)}$ obtained from the Adam-Bashforth formula.

The Adams-Moulton method is extremely efficient: only two function evaluations are needed per step for the former method, whereas six are needed for the Runge-Kutta-Fehlberg method. All have similar error terms. On the other hand changing the step size with the multistep methods is considerably more awkward than with single step methods.

5.7.4 Order of multistep methods

The *order* of a multistep method is the number of terms in the Taylor expansion of the solution that are correctly represented by the method. The accuracy of a numerical solution is largely determined by the order of the algorithm used to integrate the initial value problems. To determine the order of a given multistep method we introduce the linear functional:

$$L(y) = \sum_{p=0}^{k} [a_p y(x_{n-k} + ph) - h b_p y'(x_{n-k} + ph)]. \tag{5.55}$$

This is a direct representation of (5.46) once we take into account that $y' = f(x, y)$. If the method (5.46) were exact then we should have $L(y) = 0$. The order of the lowest non-zero term in the Taylor expansion of (5.55) is the order of the method.
Using the Taylor series for $y(ph)$ and $y'(ph)$ with $p = 0, 1, ..., k$ we obtain:

$$y(x_0+ph) = \sum_{j=0}^{+\infty} \frac{(ph)^j}{j!} y^{(j)}(x_0), \qquad y'(x_0+ph) = \sum_{j=0}^{+\infty} \frac{(ph)^j}{j!} y^{(j+1)}(x_0), \tag{5.56}$$

so that

$$L(y) = d_0 y(x_0) + d_1 h y'(x_0) + d_2 h^2 y''(x_0) + \dots , \tag{5.57}$$

where

$$d_0 = \sum_{p=0}^{k} a_p, \quad d_1 = \sum_{p=0}^{k} (p a_p - b_p), \quad d_2 = \sum_{p=0}^{k} \left(\frac{p^2}{2} a_p - p b_p \right), \dots \tag{5.58}$$

and, in general,

$$d_j = \sum_{p=0}^{k} \left(\frac{p^j}{j!} a_p - \frac{p^{j-1}}{(j-1)!} b_p \right), \qquad j \geq 2. \tag{5.59}$$

If $d_0 = d_1 = ... = d_m = 0$, then

$$L(y) = d_{m+1} h^{m+1} y^{(m+1)}(x_0) + \mathcal{O}\big(h^{m+2}\big) \tag{5.60}$$

represents the local truncation error and the method has order m.

Remark - For the Adam-Bashforth method $d_0 = \ldots = d_5 = 0$ and $d_6 = 95/288$. Hence the method if of order five and the local error is $\mathcal{O}(h^6)$.

5.7.5 Convergence, stability and consistency of multistep methods

A method to solve numerically initial value problems is *convergent* if in the limit of infinitely small step size h the numerical solution converges to the exact solution:

$$\lim_{h \to 0} y(x; h) = y(x) \tag{5.61}$$

where $y(x)$ is the exact solution of

$$y' = f(x, y), \qquad y(x_0) = y_0 \tag{5.62}$$

and $y(h; x)$ is the numerical solution obtained using an integration step h.

A method is called *stable* if the numerical solutions are bounded at all iteration steps over a finite interval. It is called *consistent* if at lowest order it is a faithful representation of the differential equation we wish to integrate.

Convergence, stability and consistency are related one to the other. In fact, one can show that *a multistep method is convergent if and only if it is stable and consistent.*

It is fairly straightforward to determine the stability and consistency of a multistep method like (5.46). Construct the two polynomials:

$$p(z) = a_k z^k + a_{k-1} z^{k-1} + \ldots ... + a_0, \tag{5.63}$$

$$q(z) = b_k z^k + b_{k-1} z^{k-1} + \ldots + b_0. \tag{5.64}$$

The first polynomial is called the *stability polynomial*. One can show that a multistep method is consistent if $p(1) = 0$ and $p'(1) = q(1)$. The question of stability is more complex.

A multi-step method is to all intents and purposes a difference equation that is (hopefully) based on the differential equation that we wish to solve, in the sense that as the integration step tends to zero the difference equation tends to the differential equation (i.e. that the difference equation is *consistent* with the differential equation). We expect that a consistent difference equation has a solution that is close to the solution of the original differential equation. However, it is also possible that it has other solutions and that these may grow as the number of integration steps increases so that they swamp the "right" solution. We can, therefore, consider two cases of stability: a method may be stable in the sense that it represents faithfully the solution of the differential equation over a finite interval. However, as the size of the integration region increases the spurious solutions of the difference equation grow and the numerical solution no longer has much to do with the exact solution of the differential equation. We can also require a stronger stability: we can, in fact, require that the spurious solutions tend to zero as the number of integration steps increases. In this case we can use the method to integrate a given differential equation for as long as we wish (within the limits of the growth of the global integration error).

Call $\{r_p\}, = 0, 1, \dots, k$ the roots of $p(z)$. The stability polynomial satisfies the *root condition* if

$$|r_p| \leq 1, \qquad 0 \leq p \leq k,$$

and all roots that satisfy $|r_j| = 1$ are simple. The stability polynomial satisfies the *strong root condition* if

$$r_0 = 1,\ |r_p| < 1, \qquad 1 \leq p \leq k.$$

The root conditions are related to the stability of the multi-step method.

1. If the stability polynomial satisfies the root condition, then the method is stable, in the sense that for h sufficiently small it will deliver accurate results over a small interval.

2. If the stability polynomial satisfies the strong root condition, then the method is *relatively stable*, meaning that, for h sufficiently small, the spurious solutions of the difference equation go to zero.

3. A method that is stable, but not relatively stable is called *weakly stable* and may exhibit diverging solutions for long integrations.

5.8 Summary

Single-step (multi-stage) and multi-step methods have advantages and disadvantages. The following table tries to summarise the main features of these methods.

	Multi-Stage	**Multi-Step**
Self-starting	Yes	No
Easy for variable steps	Yes	No
Computationally efficient	No	Yes
Theory “intuitive”	No	Yes

The following table, summarises the main numerical features of some of the algorithms that we have described.

Method	**Type**	**Local Error**	**Global Error**	**F.E. / Step**[1]	**Stability**	**Ease of changing step size**	**Recommended?**
Modified Euler	Single-step	$\mathcal{O}(h^3)$	$\mathcal{O}(h^2)$	2	Good	Good	No
Fourth-order Runge-Kutta	Single-step	$\mathcal{O}(h^5)$	$\mathcal{O}(h^4)$	4	Good	Good	Yes
Runge-Kutta-Fehlberg	Single-step	$\mathcal{O}(h^6)$	$\mathcal{O}(h^5)$	6	Good	Good	Yes
Milne	Multistep	$\mathcal{O}(h^5)$	$\mathcal{O}(h^4)$	2	Poor	Poor	No
Adams-Moulton	Multistep	$\mathcal{O}(h^5)$	$\mathcal{O}(h^4)$	2	Good	Poor	Yes

Further reading

Topics covered here are also covered in

- Chapter 10 of Linz & Wang, *Exploring Numerical Methods* (QA297 LIN),
- Chapter 8 of Kincaid & Cheney, *Numerical Analysis* (QA297 KIN),
- Chapter 12 of Süli & Mayers, *An Introduction to Numerical Analysis* (not in library),

[1]Function Evaluations per Step

- Part I (especially chapters 1–3, but chapters 4 and 5 are also useful) of Iserles, *A First Course in the Numerical Analysis of Differential Equations* (QA297 ISE).

Note that notation and implied motivation, particularly around the multistep methods, can be inconsistent with the presentation here.

Chapter 6

Boundary value problems for Ordinary Differential Equations

6.1 Introduction

A boundary value problem consists in finding a solution of a differential equation in an interval $[a, b]$ that satisfies constraints at both ends (boundary conditions). A typical example of a boundary value problem is

$$y'' = f(x, y, y'), \quad y(a) = A, \quad y(b) = B, \quad a \leq x \leq b. \tag{6.1}$$

The question of the existence of solutions of problems like (6.1) is non trivial. However, here we assume that we are always trying to find a solution to a well posed problem, that is a problem that has one and only one solution.

Boundary value problems describe many important physical and engineering problems, from the sagging of a beam to the shape of the electron orbitals in atoms.

6.2 Shooting method

We describe this method to solve boundary value problems using equation (6.1) as an example. One natural way to attack this problem is to solve the related initial-value problem, with a guess of the appropriate initial value $y'(a)$. Then we can integrate the equation to obtain the approximate solution, hoping that $y(b) = B$. If not, then the guessed value of $y'(a)$ can be altered and we can try again. The process is called **shooting**, and there are ways of doing it systematically.

Denote the guessed value of $y'(a)$ by z, so that the corresponding initial value problem is

$$y'' = f(x, y, y'), \quad y(a) = A, \quad y'(a) = z, \quad a \le x \le b. \tag{6.2}$$

The solution of this problem is $y = y(x, z)$. The objective is to select z so that $y(b, z) = B$. We call

$$\phi(z) \equiv y(b, z) - B, \tag{6.3}$$

so that our objective is simply to solve for z the equation

$$\phi(z) = 0\,. \tag{6.4}$$

In general equation (6.4) is nonlinear. We can use any numerical root finding method to solve it. One of the most effective is the secant method:

$$z_{n+1} = z_n - \phi(z_n)\frac{z_n - z_{n-1}}{\phi(z_n) - \phi(z_{n-1})}, \qquad n > 1.$$

The advantage of this method is that we need only to know $\phi(z)$ in order to apply it and that it is fairly accurate. The disadvantage is that we need two initial guesses z_0 and z_1 in order to start the method.

In the case of Newton's method, the iteration formula for z is

$$z_{n+1} = z_n - \frac{\phi(z_n)}{\phi'(z_n)}. \tag{6.5}$$

To determine $\phi'(z)$, we can proceed in two ways. Firstly, we could evaluate the derivative numerically by writing

$$\phi'(z_n) \simeq \frac{\phi(z_n + h) - \phi(z_n)}{h}, \qquad h \ll 1.$$

A second option is to introduce

$$u(x, z) = \frac{\partial y}{\partial z}$$

and we differentiate with respect to z all the equations in (6.2). This becomes

$$u'' = f_y(x, y, y')u + f_{y'}(x, y, y')u', \quad u(a) = 0, \quad u'(a) = 1.$$

The last differential equation is called the *first variational equation*. It can integrated numerically using for y and y' the values obtained by the numerical integration of equation (6.2). Finally, by differentiating (6.4) we obtain

$$\phi'(z) = u(b, z)$$

that enables us to use the Newton's method to find a root of ϕ.

To summarise, the steps of the algorithm are as follows:

1. Guess a value for the missing initial condition, z_0, and set the iteration counter n to zero, $n = 0$.
2. Compute $y = y(x, z_n)$ by integrating the initial value problem
$$y'' = f(x, y, y'), \quad y(a) = A, \quad y'(a) = z_n, \quad a \leq x \leq b.$$
3. Compute $\phi(z_n) = y(b, z_n) - B$. Use a nonlinear solver (e.g. Newton's or the secant method) to find a new value z_{n+1} for the missing initial condition.
4. If z_{n+1} is not sufficiently accurate increase the iteration counter by one, $n \to n + 1$ and repeat steps (2-4) until $\phi(z_{n+1})$ is sufficiently small.

6.3 Finite-difference method

Another approach to solving boundary value problems is to discretise the derivatives that appear in the differential equation and transform the problem in a linear system. As an example, consider the equation

$$y'' + p(x)y' + q(x)y = f(x), \quad y(a) = A, \quad y(b) = B, \quad a \leq x \leq b. \tag{6.6}$$

We can represent the derivatives using their finite difference approximations:

$$\begin{aligned} y'(x_i) &= \frac{y_{i+1} - y_{i-1}}{2h} - \frac{h^2}{6} y'''(\xi_1), \\ y''(x_i) &= \frac{y_{i+1} + y_{i-1} - 2y_i}{h^2} - \frac{h^2}{12} y^{(4)}(\xi_2), \end{aligned} \tag{6.7}$$

where $x_i = a + ih$, $i = 0, 1, 2, ..., n + 1$, $h = (b - a)/(n + 1)$. Thus, the discrete version of (6.6) is the system of $n + 2$ linear algebraic equations

$$\begin{aligned} &y_0 = A, \\ &y_{i-1}\left(1 - \frac{h}{2}p_i\right) - y_i\left(2 - h^2 q_i\right) + y_{i+1}\left(1 + \frac{h}{2}p_i\right) = h^2 f_i, \\ &y_{n+1} = B, \end{aligned}$$

where $p_i = p(x_i)$, $q_i = q(x_i)$, $f_i = f(x_i)$. This can be solved as such or reduced to an $n \times n$ system. For example, if $n = 4$, the corresponding matrix form is given by

$$\begin{pmatrix} -2 + h^2 q_1 & 1 + \frac{h}{2}p_1 & 0 & 0 \\ 1 - \frac{h}{2}p_2 & -2 + h^2 q_2 & 1 + \frac{h}{2}p_2 & 0 \\ 0 & 1 - \frac{h}{2}p_3 & -2 + h^2 q_3 & 1 + \frac{h}{2}p_3 \\ 0 & 0 & 1 - \frac{h}{2}p_4 & -2 + h^2 q_4 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{pmatrix}$$

with

$$F_1 = h^2 f_1 - A(1 - \frac{h}{2}p_1), \qquad F_2 = h^2 f_2,$$
$$F_3 = h^2 f_3, \qquad F_4 = h^2 f_4 - B(1 + \frac{h}{2}p_4).$$

This system is tridiagonal and can be solved by a special form of the Gaussian elimination algorithm. In the general case, we have an $n \times n$ system

$$T\boldsymbol{y} = \boldsymbol{F}, \tag{6.8}$$

where T is a tridiagonal $n \times n$ matrix, $\boldsymbol{y} = (y_1, ..., y_n)^T$ and $\boldsymbol{F} = (F_1, ..., F_n)^T$.

To analyse the error induced by the representation (6.7) of the derivatives we introduce an error vector $(e_1, ..., e_n)$, where $e_i = y(x_i) - y_i$, $i = 1, \dots, n$ and $y(x)$ is assumed to be the exact solution. Substituting into the finite difference representation (6.8) of equation (6.6), we obtain the system

$$T\boldsymbol{e} = h^4\boldsymbol{G}$$

where $\boldsymbol{G}$ is a constant vector. If $\det T \neq 0$, we have

$$\boldsymbol{e} = h^4 T^{-1}\boldsymbol{G}.$$

We can use this relation to show that the error is $O(h^2)$ [sic] as $h \to 0$.

6.4 The Ritz method

The Ritz, Galerkin, Square Least methods are used widely on problems in which it is required to determine an unknown function. Of course, boundary value problems for differential equations are in this category. Suppose we are confronted with a problem of the form

$$\mathcal{L}u(x) = f(x) \tag{6.9}$$

in which $\mathcal{L}$ is the linear operator

$$\mathcal{L}u = -\frac{\mathrm{d}}{\mathrm{d}x}\left[p(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right] + q(x)u = f(x). \tag{6.10}$$

Here $f(x)$ is a given function and $u(x)$ is is the function to be determined from the equation and boundary conditions

$$u(a) = 0, \qquad u(b) = 0. \tag{6.11}$$

We assume that $p(x) \geq p_0 > 0$, $q(x) \geq 0$ for $x \in [a,b]$. This means that the operator $\mathcal{L}$ is symmetric and positive definite in the real Hilbert space $L_2[a,b]$ since, using the integration by parts, we have

$$\begin{aligned}
< \mathcal{L}u, u > &= \int_a^b \mathcal{L}u \cdot u \, \mathrm{d}x \\
&= -\int_a^b (pu')'u \, \mathrm{d}x + \int_a^b qu^2 \, \mathrm{d}x \\
&= \int_a^b p(u')^2 \, \mathrm{d}x + \int_a^b qu^2 \, \mathrm{d}x \geq 0
\end{aligned}$$

for arbitrary u. The quantity $< \mathcal{L}u, u >$ is called the energy of the element u relative the operator $\mathcal{L}$.
Consider a linear quadratic functional

$$\begin{aligned}
J(u) &=< \mathcal{L}u, u > -2 < f, u > \qquad (6.12a) \\
&= \int_a^b p(u')^2 \, \mathrm{d}x + \int_a^b qu^2 \, \mathrm{d}x - 2\int_a^b fu \, \mathrm{d}x \, . \qquad (6.12b)
\end{aligned}$$

A theorem states that if the equation $\mathcal{L}u(x) = f(x)$ has a solution u_0, this solution minimises the functional $J(u)$. Conversely, if there exists an element u_0 that minimises the functional $J(u)$, this element satisfies the equation $\mathcal{L}u(x) = f(x)$. The proof is based on the ability to reduce the functional $J(u)$ to the form

$$J(u) =< \mathcal{L}(u - u_0), u - u_0 > - < \mathcal{L}u_0, u_0 >,$$

and then analysing the function

$$\begin{aligned}
g(t) &= J[u_0(x) + tv(x)] \\
&=< \mathcal{L}u_0, u_0 > +2t < \mathcal{L}u_0, v > +t^2 < \mathcal{L}v, v > -2 < f, u_0 > -2t < f, v > .
\end{aligned}$$

The basic idea of the Ritz method consists in replacing the boundary value problem for $\mathcal{L}u(x) = f(x)$ by the problem of minimising the functional $J(u)$. Suppose we select basis functions u_1, u_2,...,u_n,... in $L_2[a,b]$. Hence, we seek a solution of the form

$$u_n(x) = \sum_{m=1}^{n} c_m u_m(x), \qquad (6.13)$$

where c_1, c_2,...,c_n are unknown constants. Substituting this sum into the functional $J(u) =< \mathcal{L}u, u > -2 < f, u >$, we obtain

$$J(u_n) = J(c_1, c_2, ..., c_n) = \sum_{m,k=1}^{n} c_m c_k < \mathcal{L}u_m, u_k > -2\sum_{m=1}^{n} c_m < f, u_m >, \qquad (6.14)$$

which is a quadratic form with respect to the unknown constants $c_1, c_2, ..., c_n$. Looking for a minimum element of $J(u_n) = J(c_1, c_2, ..., c_n)$, we require that

$$\frac{\partial}{\partial c_m} J(c_1, c_2, ..., c_n) = 0, \qquad m = 1, 2, ..., n. \tag{6.15}$$

Thus, we come to the linear $n \times n$ system of algebraic equations for the unknown constants c_1, c_2,...,c_n

$$\sum_{m=1}^{n} c_m < \mathcal{L} u_m, u_k >=< f, u_k >, \qquad k = 1, 2, ..., n, \tag{6.16}$$

which is called the Ritz system. After we have found the unknown constants c_1, c_2,...,c_n, the expression (6.13) gives us an approximate solution.

6.5 The collocation method

6.5.1 Introduction

The method of collocation can be used to tackle many problems in the numerical analysis of ordinary and partial differential equations. Here we give a general description of this method that can be adapted easily to the solution of a boundary value problem for an ordinary differential equation.

Suppose that we have a linear operator $\mathcal{L}$ (for example, a linear differential equation) that acts on a space of functions. We wish to solve the equation

$$\mathcal{L} u(x) = w(x), \tag{6.17}$$

where $w(x)$ is a known function and $u(x)$ is the solution we are looking for. We indicate with

$$\{v_1(x), v_2(x), \ldots, v_n(x)\}$$

a set of n known functions (*basis functions*) and we write the (unknown) solution of equation (6.17) as

$$u(x) = \sum_{j=1}^{n} c_j v_j(x), \tag{6.18}$$

where the coefficients c_j are (at this stage) unknown. In general $u(x)$ written as in equation (6.18) cannot be an exact solution of equation (6.17). However, we can find a set of coefficients $\{c_j\}$ such that (6.18) is a good approximation of the solution of the equation (6.17).

Remark - Note that the series solution of a differential equation falls into this class of methods. In that case the functions $v_j(x) = x^j$, with $j = 0, 1, 2, \ldots, n$ and the coefficients c_j of the expansion are determined by requiring that equation (6.17) is satisfied at each order in x.

6.5.2 The norm method

There are various ways of defining "good approximation". As a first example, consider the boundary value problem

$$u''(x) - u(x) = 0\,, \qquad u(0) = 1, \quad u(1) = e. \tag{6.19}$$

We wish to find an approximate solution to this problem using as basis the functions

$$v_0(x) = 1, \quad v_1(x) = x \quad \text{and} \quad v_2(x) = x^2,$$

and write the approximate solution of equation (6.19) as a polynomial of order two:

$$u^{(a)}(x) = c_0 v_0(x) + c_1 v_1(x) + c_2 v_2(x) = c_0 + c_1 x + c_2 x^2. \tag{6.20}$$

It is clear that there are no values of the constants c_j that can make $u^{(a)}(x)$ as defined in equation (6.20) equal to the exact solution of equation (6.19),

$$u^{(e)}(x) = e^x \tag{6.21}$$

for all values of x. However, we can attempt to find some values of these parameters that minimise the error in approximating $u^{(e)}(x)$ with $u^{(a)}(x)$. There are many different measures of the error of the approximation. For example, we could require that the coefficients c_j are such that

1. $u^{(a)}(x)$ satisfies the boundary conditions, i.e. $u^{(a)}(0) = 1$ and $u^{(a)}(1) = e$.
2. $u^{(a)}(x)$ satisfies as well as possible equation (6.19) in the sense that
$$F(c_j) \equiv \left\| \frac{\mathrm{d}^2}{\mathrm{d}x^2} u^{(a)}(x) - u^{(a)}(x) \right\|_2^2 = \int_0^1 \left[\frac{\mathrm{d}^2}{\mathrm{d}x^2} u^{(a)}(x) - u^{(a)}(x) \right]^2 \mathrm{d}x \tag{6.22}$$
is as small as possible. A criterion (distantly?) related to this is used in the finite element method to solve partial differential equations.

The left boundary condition fixes c_0:

$$u^{(a)}(0) = c_0 = 1. \tag{6.23}$$

The right boundary condition gives a relation between c_1 and c_2:

$$u^{(a)}(1) = c_0 + c_1 + c_2 = e \implies c_1 = e - 1 - c_2. \tag{6.24}$$

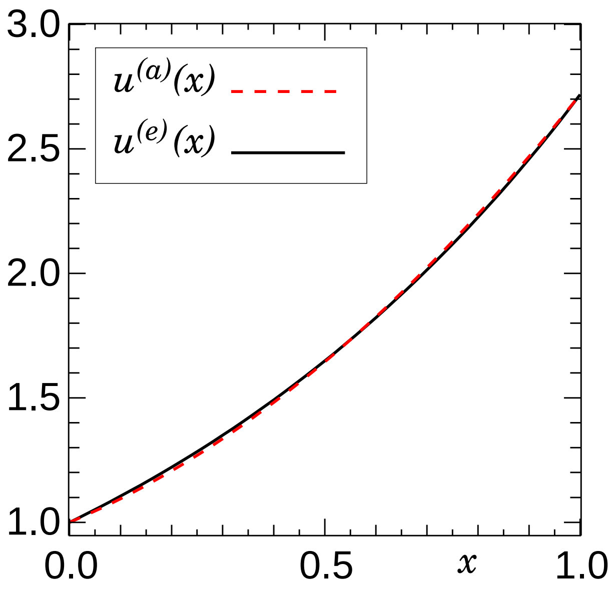


Figure 6.1: *Graphs of the exact [Eq. (6.21)] and approximate [Eq. (6.26)] solution of the boundary value problem (6.19).*

Substituting (6.23) and (6.24) into (6.19) gives that $F(c_j)$ defined by (6.22) is

$$F(c_2) = \frac{47}{10}c_2^2 - \frac{13}{6}(1+e)c_2 + \frac{1+e+e^2}{3}. \tag{6.25}$$

We solve for c_2 by requiring that $F(c_2)$ is as small as possible, i.e. that the derivative of $F(c_2)$ with respect to c_2 is zero. Differentiating (6.25) we obtain

$$\frac{\mathrm{d}F}{\mathrm{d}c_2} = \frac{47}{5}c_2 - \frac{13}{6}(1+e) = 0 \implies c_2 = \frac{65}{282}(1+e),$$

so that from equation (6.24) we have

$$c_1 = \frac{217e - 347}{282}$$

and the approximate solution of the boundary value problem (6.19) is

$$u^{(a)}(x) = 1 + \frac{217e-347}{282}x + \frac{65}{282}(1+e)x^2. \tag{6.26}$$

The graphs of the approximate and exact solutions are shown in Figure 6.1: the match is pretty impressive for a three node approximation.

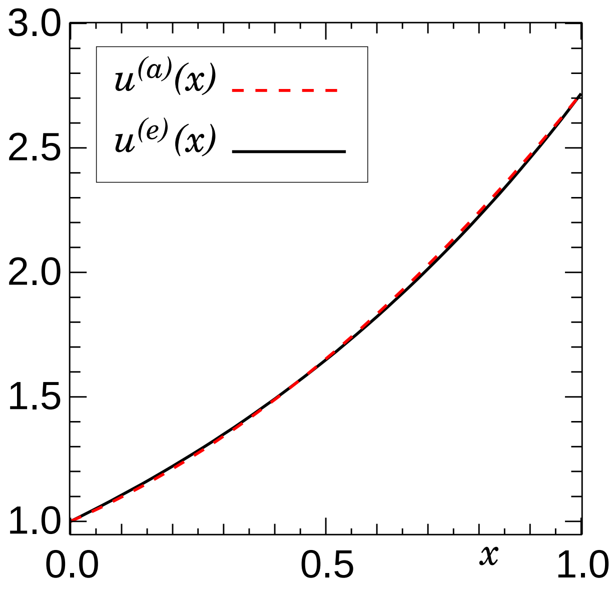


Figure 6.2: *Graphs of the exact [Eq. (6.21)] and approximate [Eq. (6.27)] solution of the boundary value problem (6.19).*

6.5.3 The collocation method

In the *collocation* method we substitute the approximate solution (6.18) in (6.17),

$$\mathcal{L}\sum_{j=1}^{n} c_j v_j(x) = w(x) \implies \sum_{j=1}^{n} c_j \mathcal{L} v_j(x) = w(x),$$

and we require that this equation should be satisfied at a set of n *collocation points* $\{x_j\}$, i.e. that the coefficients c_j are the solution of the system of linear equations

$$\sum_{j=1}^{n} c_j \mathcal{L} v_j(x_k) = w(x_k), \qquad k = 1, 2, \ldots, n.$$

Note that the values of the basis functions $v_j(x)$ at the collocation points x_k should be such that the matrix of the coefficients of this system is non-singular.

Once again consider as an example the boundary value problem (6.19) and write the approximate solution as in equation (6.20). We require that it should satisfy the boundary conditions as in equations (6.23) and (6.24), so that the approximate solution is now given by

$$u^{(a)}(x) = 1 + (e - 1 - c_2)x + c_2 x^2.$$

Finally, we require that this function should satisfy the differential equation (6.19) at $x = 1/2$. Taking into account that

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2} u^{(a)}(x) = 2c_2$$

we have that this requirement is equivalent to

$$2c_2 - \left[1 + (e - 1 - c_2)\frac{1}{2} + c_2\left(\frac{1}{2}\right)^2\right] = 0 \implies c_2 = \frac{2}{9}(1+e) \implies c_1 = \frac{7e-11}{9}$$

so that the approximate solution is given by

$$u^{(a)}(x) = 1 + \frac{7e-11}{9}x + \frac{2}{9}(1+e)x^2. \tag{6.27}$$

The graphs of the approximate and exact solutions are shown in Figure 6.2: the match is pretty impressive for a three node approximation and is comparable, but different, from that in Figure 6.1.

Remark - In general it is not advisable to use as basis functions the powers of x or a uniformly spaced set of nodes. Orthogonal polynomials (like the Legendre and the Chebyschev polynomials) and non-uniform grids (e.g. Gauss-Lobatto grids) give more accurate and numerically stable results.

Further reading

Topics covered here are also covered in

- Chapter 11 of Linz & Wang, *Exploring Numerical Methods* (QA297 LIN),
- Chapter 8 of Kincaid & Cheney, *Numerical Analysis* (QA297 KIN),
- Chapter 13 (and to some extent 14) of Süli & Mayers, *An Introduction to Numerical Analysis* (not in library).

Chapter 7

Finite difference methods for Partial Differential Equations

7.1 Revision of Partial Differential Equations

A *partial differential equation* is a relation between the partial derivatives of an unknown function and the independent variables. The order of the highest derivative is called the *order* of the equation.

Just as in the case of an ordinary differential equation, we say that a partial differential equation is *linear* if it is of the first degree in the dependent variable (the unknown function) and its partial derivatives. If each term of such equation contains either the dependent variable or one of its derivatives, the equation is called *homogeneous*; otherwise it is said to be *non homogeneous*.

Example - The equations

$$\begin{array}{ll} (a) & \dfrac{\partial \phi}{\partial t} - \dfrac{\partial^2 \phi}{\partial x^2} = 0, \\ (b) & \phi(x,t)\dfrac{\partial^2 \phi}{\partial t^2} + \dfrac{\partial \phi}{\partial x} = f(x,t), \\ (c) & \dfrac{\partial^4 \phi}{\partial x^4} + \dfrac{\partial^4 \phi}{\partial y^4} = g(x,y), \end{array}$$

are, respectively, (a) Linear, homogeneous, second order, (b) nonlinear, non-homogeneous, second order, (c) linear, non-homogeneous, fourth order.

In this unit we consider only second order linear partial differential equations with constant coefficients:

$$A\frac{\partial^2 u}{\partial x^2} + 2B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} = f(x,y).$$

These equations are classified in three groups[1]:

Type	Coefficients	Typical form	Name
Hyperbolic	$B^2 - 4AC > 0$	$u_{tt} - c^2 u_{xx} = 0$	Wave eq.
Parabolic	$B^2 - 4AC = 0$	$u_t - c^2 u_{xx} = 0$	Heat eq.
Elliptic	$B^2 - 4AC < 0$	$u_{xx} + u_{yy} = 0$	Laplace eq.

The names arise by analogy with the curve

$$ax^2 + 2bxy + cy^2 = f,$$

which represents an hyperbola, parabola and ellipse according as $b^2 - 4ac$ is positive, zero or negative, respectively.

The classification of the equations is very important for the study of their solutions. A *solution* of a partial differential equation in some region R of the space of the independent variables is a function that has all the partial derivatives that appear in the equation and that satisfies the equation everywhere in R. Like ordinary differential equations, Partial differential equation have many solutions. For example, the functions

$$u(x,y) = x^2 - y^2, \quad u = e^x \cos(y), \quad u = \ln\bigl(x^2 + y^2\bigr),$$

are all solutions of the equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0.$$

Exactly like for ordinary differential equations we must specify something more if we wish to obtain a unique solution: for example we must specify the value of the function on the boundary of the region R (*boundary conditions*), and/or we must specify the value of the function at the start (*initial conditions*). What type of boundary or initial condition to use depends on the type of Partial differential equation.

Hyperbolic

Example: $u_{tt} - c^2 u_{xx} = 0$.
Physical meaning: Wave motion
Initial conditions: The value of u and its time derivative at $t = 0$: $u(x,0) = f(x)$ and $u_t(x,0) = g(x)$.
Boundary conditions: Either u or its normal derivative at the boundary of the integration region: e.g. $u(0,t) = \phi_1(t)$ and $u(L,t) = \phi_2(t)$. The former type of

[1]Note that the coefficients A, B, etc. need not be constant

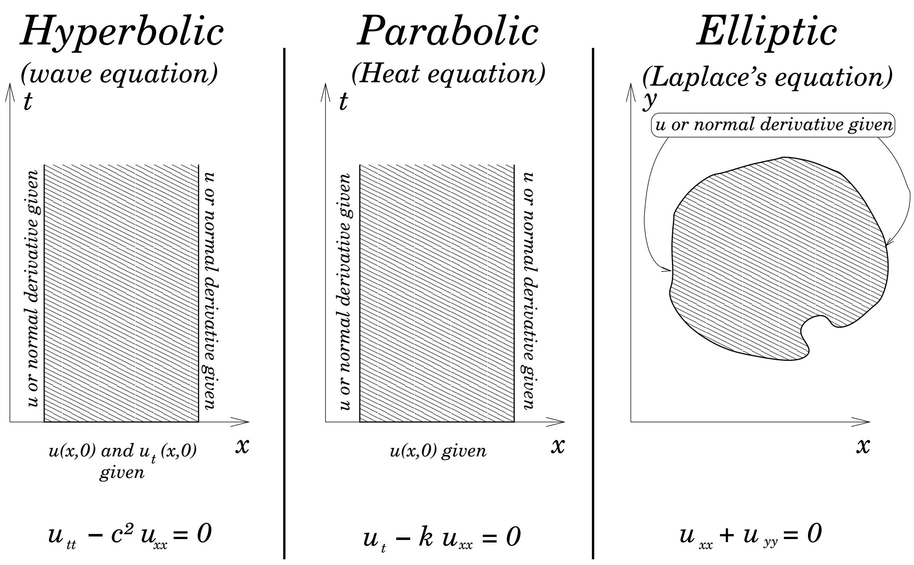

Hyperbolic
(wave equation)
t
u or normal derivative given
u or normal derivative given
u(x,0) and u_t (x,0) given
x
u_tt − c² u_xx = 0
Parabolic
(Heat equation)
t
u or normal derivative given
u or normal derivative given
u(x,0) given
x
u_t − k u_xx = 0
Elliptic
(Laplace's equation)
y
u or normal derivative given
x
u_xx + u_yy = 0

Figure 7.1: *Types of partial differential equations and their boundary and initial conditions.*

boundary condition is called a *Dirichlet* boundary condition; the latter is called a *Neumann* boundary condition. It is also possible to mix the two types of boundary conditions (*mixed boundary conditions*).

Parabolic

Example: $u_t - c^2 u_{xx} = 0$.
Physical meaning: Heat diffusion
Initial conditions: The value of u at $t = 0$: $u(x, 0) = f(x)$.
Boundary conditions: Either u or its normal derivative at the boundary of the integration region: e.g. $u(0, t) = \phi_1(t)$ and $u(L, t) = \phi_2(t)$.

Elliptic

Example: $u_{xx} + u_{yy} = 0$ (Laplace's equation).
Physical meaning: Stationary profile of a membrane, stationary temperature profile of a metal plate, electrostatic potential in the absence of charges.
Initial conditions: They do not apply to these equations.
Boundary conditions: Either u or its normal derivative at the boundary of the integration region: e.g. if the region is the rectangle $0 \le x \le a$, $0 \le y \le b$: $u(0, y) = \phi_1(y)$, $u(a, y) = \phi_2(y)$, $u(x, 0) = g_1(x)$ and, finally, $u(x, b) = g_2(x)$.

7.2 Numerical methods for PDEs

There are many classes of methods to solve numerically partial differential equations. Three main groups are:

1. *Finite difference methods*

   The differential operators are approximated using their finite difference representation on a given grid. In this way the Partial Differential Equation is transformed to a (nonlinear) algebraic equation for the values of the solution on the grid points. These are the only methods that we discuss at any length in this unit.

2. *Finite elements methods*

   The domain of the solution is divided into cells. The solution is represented as a simple function (e.g. a linear function) on each cell and the partial differential equation is transformed to an algebraic problem for the matching conditions of the simple solutions at the boundaries of the cells.

3. *Spectral methods*

   The solution is represented by a superposition of known functions (e.g. trigonometric functions or special polynomials). The partial differential equation is transformed to a set of algebraic equations or ordinary differential equations for the amplitudes of the component functions. A subclass of these methods are the *collocation methods*: the solution is represented on a grid and the decomposition of the solution in known functions is used to estimate to a high degree of accuracy the partial derivatives of the solution on the grid points. These methods are highly accurate and fast, but in general require the domain of the solution to be fairly simple, e.g. a rectangle.

7.3 Elliptic Equations

7.3.1 Introduction

A "standard form" of elliptic partial differential equation with Dirichlet boundary conditions is

$$\begin{cases} u_{xx} + u_{yy} = f(x,y), & \text{in } \Omega, \\ u\big|_{\partial\Omega} = \phi(x,y), & \text{on } \partial\Omega. \end{cases} \tag{7.1}$$

This problem has a unique solution if the domain Ω has a smooth boundary and f and ϕ are continuous functions.

7.3.2 A simple finite difference method

We are going to solve equation (7.1) numerically using the method of finite differences. At the heart of this method is the finite difference approximation of the second derivative of a function $y(x)$:

$$y''(x) = \frac{1}{h^2}[y(x+h) + y(x-h) - 2y(x)] - \frac{h^2}{12}y^{(4)}(\xi), \tag{7.2}$$

where h is a fixed step size. We illustrate this method using a simple case of equation (7.1), namely the case of a Dirichlet problem on a rectangle $0 < x < a$, $0 < y < b$. First, a network of grid points is established on the rectangle:

$$(x_i, y_j) = (ih_x, jh_y), \qquad 0 \le i \le n+1, \quad 0 \le j \le m+1,$$

where the step sizes in the x and y directions are given by

$$h_x = \frac{a}{n+1}, \qquad h_y = \frac{b}{m+1}.$$

Next, the differential equation in (7.1) at the mesh point (x_i, y_j) is replaced by its finite-difference analogue at that point, which for $1 \le i \le n$ and $1 \le j \le m$ is:

$$\frac{1}{h_x^2}[u_{i-1,j} + u_{i+1,j} - 2u_{i,j}] + \frac{1}{h_y^2}[u_{i,j-1} + u_{i,j+1} - 2u_{i,j}] = f_{ij},$$
$$\Longrightarrow \qquad u_{i-1,j} + u_{i+1,j} - 2u_{i,j} + \alpha[u_{i,j-1} + u_{i,j+1} - 2u_{i,j}] = h^2 f_{i,j}, \tag{7.3}$$

where $u_{i,j} = u(x_i, x_j)$, $\alpha = (h_x/h_y)^2$ and $h = h_x$. The values of $u_{i,j}$ are known when $i = 0$ or $n+1$ and when $j = 0$ or $m+1$, since these are the prescribed boundary values in the problem:

$$u_{0,j} = \phi(x_0, y_j) = \phi(0, y_j), \qquad u_{i,0} = \phi(x_i, y_0) = \phi(x_i, 0),$$
$$u_{n+1,j} = \phi(x_{n+1}, y_j) = \phi(a, y_j), \qquad u_{i,m+1} = \phi(x_i, y_{m+1}) = \phi(x_i, b).$$

This means that equation (7.1) has been transformed to a non-homogeneous system of linear equations, given by (7.3), with unknowns $u_{i,j}$ and $1 \le i \le n$, $1 \le j \le m$.

As an example, consider the simple case $n = 2$ and $m = 3$. The approximate solution of equation (7.1) at the grid points are the solution of the 6×6 system

$$\begin{aligned}
[u_{01} - 2u_{11} + u_{21}] + \alpha[u_{10} - 2u_{11} + u_{12}] &= h^2 f_{11},\\
[u_{02} - 2u_{12} + u_{22}] + \alpha[u_{11} - 2u_{12} + u_{13}] &= h^2 f_{12},\\
[u_{03} - 2u_{13} + u_{23}] + \alpha[u_{12} - 2u_{13} + u_{14}] &= h^2 f_{13},\\
[u_{11} - 2u_{21} + u_{31}] + \alpha[u_{20} - 2u_{21} + u_{22}] &= h^2 f_{21},\\
[u_{12} - 2u_{22} + u_{32}] + \alpha[u_{21} - 2u_{22} + u_{23}] &= h^2 f_{22},\\
[u_{13} - 2u_{23} + u_{33}] + \alpha[u_{22} - 2u_{23} + u_{24}] &= h^2 f_{23}.
\end{aligned}$$

The unknown quantities in this problem can be ordered in many ways. We select the one known as the natural ordering

$$u = [u_{11}, u_{12}, u_{13}, u_{21}, u_{22}, u_{23}]^T.$$

The system has the form $A\boldsymbol{u} = \boldsymbol{F}$ with

$$A = \begin{pmatrix} -2(1+\alpha) & \alpha & 0 & 1 & 0 & 0 \\ \alpha & -2(1+\alpha) & \alpha & 0 & 1 & 0 \\ 0 & \alpha & -2(1+\alpha) & 0 & 0 & 1 \\ 1 & 0 & 0 & -2(1+\alpha) & \alpha & 0 \\ 0 & 1 & 0 & \alpha & -2(1+\alpha) & \alpha \\ 0 & 0 & 1 & 0 & \alpha & -2(1+\alpha) \end{pmatrix} \tag{7.4}$$

and

$$\boldsymbol{F} = \begin{pmatrix} h^2 f_{11} - u_{01} - \alpha u_{10} \\ h^2 f_{12} - u_{02} \\ h^2 f_{13} - u_{03} - \alpha u_{14} \\ h^2 f_{21} - u_{31} - \alpha u_{20} \\ h^2 f_{22} - u_{32} \\ h^2 f_{23} - u_{33} - \alpha u_{24} \end{pmatrix},$$

where in the expressions of the components of the vector $\boldsymbol{F}$ only the boundary values of $u_{i,j}$ are present.

In general, the $n \times m$ system is sparse because each equation contains at most five unknowns. Iterative procedures such as the Gauss-Seidel iterative method can be quite effective in this situation. Moreover, in setting up such a procedure there is no need to store the matrix A of the coefficients defined in equation (7.4). In fact, we can rewrite equation (7.3) as

$$u_{i,j} = \frac{1}{2(1+\alpha)} \left[u_{i-1,j} + u_{i+1,j} + \alpha(u_{i,j-1} + u_{i,j+1}) - h^2 f_{i,j} \right], \quad 1 \le i \le n,\ 1 \le j \le m. \tag{7.5}$$

This equation is a Gauss-Seidel formula to update $u_{i,j}$. When this equation is used, the value obtained from the right-hand side replaces the old value of $u_{i,j}$. It can also be seen as an implementation of Jacobi's method, if we assume that the variable on the left-hand side are stored separately from those on the right hand side. As initial guess of the solution we can assume that $u_{i,j} \equiv 0$ (in general we should use a guess that satisfies the boundary conditions). Of course an initial guess close to the solution will reduce the number of iterations needed for convergence.

7.3.3 Error analysis

How accurate is the solution obtained using (7.3) or, equivalently (7.5)? To answer this question we introduce the error

$$e_{i,j} = u_{i,j} - \hat{u}_{i,j},$$

where $\hat{u}_{i,j} = u(x_i, y_j)$ are the values of the exact solution of equation (7.1). Substituting

$$u_{i,j} = e_{i,j} + \hat{u}_{i,j}$$

into the difference equation (7.3), we obtain

$$\frac{1}{h_x^2}[e_{i-1,j} + e_{i+1,j} - 2e_{i,j}] + \frac{1}{h_y^2}[e_{i,j-1} + e_{i,j+1} - 2e_{i,j}] =$$
$$f_{i,j} - \frac{1}{h_x^2}[\hat{u}_{i-1,j} + \hat{u}_{i+1,j} - 2\hat{u}_{i,j}] - \frac{1}{h_y^2}[\hat{u}_{i,j-1} + \hat{u}_{i,j+1} - 2\hat{u}_{i,j}].$$

Using the finite-difference approximation formula (7.2) to eliminate the finite differences we obtain

$$\begin{aligned}\frac{1}{h_x^2}[e_{i-1,j} + e_{i+1,j} - 2e_{i,j}] + \frac{1}{h_y^2}[e_{i,j-1} + e_{i,j+1} - 2e_{i,j}] = f_{i,j} -& \\ &[u_{xx}(x_i, y_j) + \frac{h_x^2}{12}u_{xxxx}(\xi_i, y_i)] - \\ &[u_{yy}(x_i, y_j) + \frac{h_2^2}{12}u_{yyyy}(x_i, \zeta_j)] \\ &= -\frac{h_x^2}{12}u_{xxxx}(\xi_i, y_i) - \frac{h_y^2}{12}u_{yyyy}(x_i, \zeta_j)\end{aligned}$$

where we have used

$$u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j) = f_{i.j}.$$

Hence, the error satisfies the difference equation

$$e_{i-1,j} + e_{i+1,j} - 2e_{i,j} + \alpha[e_{i,j-1} + e_{i,j+1} - 2e_{i,j}] = -\frac{h_x^4}{12}u_{xxxx}(\xi_i, y_i) - \frac{h_x^2 h_y^2}{12}u_{yyyy}(x_i, \zeta_j),$$

and zero boundary conditions. Analysing this equality, it is possible to prove that

$$||e_{i,j}|| \simeq O(h_x h_y).$$

This means that in the limit $n \to +\infty$ and $m \to +\infty$ the norm of the error tends to zero.

7.4 Parabolic Equations

7.4.1 Introduction

Throughout this section we assume that the parabolic equation to be solved is

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \qquad 0 \le x \le 1,\ 0 \le t \tag{7.6}$$

with boundary conditions

$$u(0,t) = 0, \qquad u(1,t) = 0,$$

and initial condition $u(x,0) = g(x)$. There are many finite difference methods to solve equation (7.6). Here we consider only a few that are typical examples of their respective classes. Moreover, it should be noted that equation (7.6) is linear. The methods that we discuss become much harder to implement in the case of nonlinear equations.

7.4.2 An explicit method - Forward-Time, Centred-Space (FTCS)

The method

We fix a time discretisation step δ and a space discretisation step h so that the solution of equation (7.6) is represented on the grid

$$(x_i, t^n) = (ih, n\delta), \qquad 0 \le i \le N+1,\ n \ge 0$$

where the number of grid points between $x = 0$ and $x = 1$ is $N+2$. We use the notation u_i^n to indicate $u(x_i, t^n)$. We discretise the time derivative in equation (7.6) using a forward difference approximation

$$\frac{\partial u}{\partial t} = \frac{u_i^{n+1} - u_i^n}{\delta} + O(\delta) \tag{7.7}$$

and the spatial derivative using

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1}^n + u_{i-1}^n - 2u_i^n}{h^2} + O(h^2) \tag{7.8}$$

so that equation (7.6) is represented by the set of (linear) algebraic equations

$$u_i^{n+1} = u_i^n + \frac{\delta}{h^2}(u_{i+1}^n + u_{i-1}^n - 2u_i^n)$$

or, equivalently,

$$u_i^{n+1} = (1-2s)u_i^n + s(u_{i+1}^n + u_{i-1}^n) \tag{7.9}$$

where $s = \delta/h^2$. Equation (7.9) is a finite difference representation of Equation (7.6): since this equation gives the new values of u_i^{n+1} explicitly in terms of previous values of u_{i+1}^n, u_i^n and u_{i-1}^n the method based on this equation is called an *explicit method*. As it involves a forward difference approximation of the time derivative and a centred approximation of the spatial derivative it is called a *forward-time centred-space* (FTCS) method.

Equation (7.9) must be complemented by the discretized version of the initial and boundary conditions, namely

$$u_0^n = u_{N+1}^n = 0 \quad \text{and} \quad u_i^0 = g(x_i).$$

Consistency, Stability and Convergence

We have claimed that equation (7.9) is a finite difference representation of equation (7.6) and that, therefore, can be used to find an accurate numerical solution to this equation. Given any algorithm to solve numerically a partial differential equation we must clearly determine if it is "good", in the sense that it can be used to obtain efficiently an accurate solution of the problem we aim to solve. In order to do this we must define clearly what we mean by a "good method".

1. A finite difference equation is **consistent** with a partial differential equation if the difference between the Finite Difference Equation (FDE) and the PDE (i.e. the truncation error) vanishes as the sizes of the grid spacings go to zero independently.

   When the truncation error of the finite difference approximations of the individual exact partial derivatives are known, proof of consistency is straightforward. When the truncation errors of the individual finite difference approximations are not known, the complete finite difference equation must be analysed for consistency. That is accomplished by expressing each term in the finite difference equation by a Taylor series about a particular grid point. The resulting equation, which is called the modified differential equation (MDE), can be simplified to yield the exact form of the truncation error of the complete finite difference equation. We will not develop this concept further in this unit.

2. The **order** of a finite difference approximation of a partial differential equation is the rate at which the global error of the finite difference solution approaches zero as the size of the grid spacings approach zero.

   The global error of a finite difference equation is the order of the truncation error terms in the finite difference approximations of the individual exact partial derivatives in the Partial Differential Equation.

3. When applied to a partial differential equation that has a bounded solution, a finite difference equation is **stable** if it produces a bounded solution and is unstable otherwise.

   If the solution of the FDE is bounded for all values of the grid spacings, then the FDE is *unconditionally stable*. If the solution of the FDE is bounded only for certain values of the grid spacings, then the FDE is *conditionally stable*. If the solution of the FDE is unbounded for all values of the grid spacings, then the FDE is *unconditionally unstable*. The most used method to prove the stability of a (linear) finite difference scheme is the Von Neumann method (see below).

4. A finite difference method is **convergent** if the solution of the finite difference equation approaches the exact solution of the partial differential equation as the sizes of the grid spacings go to zero.

   Convergence is the most desirable property of an integration scheme. However it is quite hard to prove directly that a method is convergent.

Consistency, stability and convergence are not independent concepts. They are related by the following theorem due to Lax (1954):

> Given a properly posed linear initial-value problem and a finite difference approximation to it that is consistent, stability is the necessary and sufficient condition for convergence.

Thus, the question of convergence of a finite difference method is answered by a study of the consistency and stability of the finite difference equation. If the finite difference equation is consistent and stable, then the finite difference method is convergent.

The Lax equivalence theorem applies to well-posed linear initial-value problems. May problems in engineering and science are not linear and nearly all problems involve boundary conditions in addition to initial conditions. There is no equivalence theorem for such problems. Nonlinear PDEs must be linearised locally and the FDE that approximates the linearised PDE is analysed for stability. Experience has shown that the stability criteria obtained of the linearised FDE also apply to the nonlinear FDE and that FDEs that are consistent and whose linearised equivalent is stable generally converge even for nonlinear initial-boundary-value problems.

Consistency, stability and convergence of a FTCS method

Equation (7.9) is first order in time and second order in space: in fact the error involved in the discretisation of the time derivative, equation (7.7), is $O(\delta)$, while the error in the discretisation of the space derivative, equation (7.8), is $O(h^2)$.

Equation (7.9) is clearly consistent. As $\delta \to 0$ and $h \to 0$ the finite difference approximations of the time and space derivatives, equations (7.7, 7.8), converge to the respective derivatives.

To verify whether the method given by equation (7.9) is stable, we start by observing that the solution of equation (7.6) with boundary conditions $u(0, t) = u(1, t) = 0$ tends to zero in the long time limit whatever the initial condition $g(x)$. Therefore, also the solution of the finite difference approximation, equation (7.9), must tend to zero as the time discretisation index n tends to infinity. To verify whether this is the case we use the *Von Neumann method.* The solution of equation (7.9) is of the form

$$u_\ell^k = e^{\mathrm{i}\alpha\ell h} q^k \tag{7.10}$$

where $\mathrm{i} = \sqrt{-1}$ while α and q are real parameters that have to be determined by requiring that (7.10) satisfies (7.9). Substituting (7.10) into (7.9) we obtain that q is given by

$$q = s\left(e^{\mathrm{i}\alpha h} + e^{-\mathrm{i}\alpha h}\right) + (1 - 2s) = 1 - 4s\sin^2\left(\frac{\alpha h}{2}\right).$$

In order for $|u_\ell^k|$ to decrease to zero as k tends to infinity we must have

$$|q| < 1 \implies s < \frac{1}{2} \implies \frac{\delta}{h^2} < \frac{1}{2}. \tag{7.11}$$

In other words, the FTCS method (7.9) is only conditionally stable and the time and space discretisation steps, δ and h respectively, must satisfy the constraint (7.11). Using the Lax theorem we can therefore conclude that, under these conditions, the FTCS method is also convergent. However, equation (7.11) is a rather stringent constraint as the time step must be reduce by a factor of four if the space step is halved. Therefore, methods like (7.9) tend to be rather slow.

7.4.3 An Implicit method - Backward-Time, Centred-Space (BTCS)

The method

In the explicit finite difference approximation of the diffusion equation the right hand side of equation (7.6) is computed at the current time step and used to compute the value of the solution at the next time step. In the implicit finite difference approximation the right hand side is "computed" at the next time step, so that the finite difference scheme is an implicit equation for the new value of the solution. In the case of equation (7.6) this is accomplished by using a backward difference approximation of the time derivative at $t^{n+1} = t + \delta$ so that equation (7.6) is discretised as

$$\begin{aligned} \frac{u_i^{n+1}-u_i^n}{\delta} &= \frac{u_{i+1}^{n+1}+u_{i-1}^{n+1}-2u_i^{n+1}}{h^2} \\ \Longrightarrow \qquad (1+2s)u_i^{n+1} &= u_i^n + s(u_{i+1}^{n+1}+u_{i-1}^{n+1}) \qquad (7.12) \end{aligned}$$

where $s=\delta/h^2$ and the unknowns are all the u_i^{n+1}. Equation (7.12) constitutes of a tri-diagonal linear system for the unknowns u_i^n with row

$$-su_{i-1}^{n+1}+(1+2s)u_i^{n+1}-su_{i+1}^{n+1}=u_i^n, \qquad 1\le i\le N.$$

This system can be solved by Gaussian elimination. Note, however, that if the partial differential equation is nonlinear then equation (7.12) becomes a nonlinear algebraic equation for the unknowns u_i^{n+1} and is, as a consequence, very hard to solve.

Consistency, stability and convergence

Like the FTCS method, the BTCS is first order in time and second order in space. Moreover, it is clearly consistent.

The stability analysis is very similar to that of the explicit method. We look for a solution of equation (7.12) of the form (7.10) and substitute it in equation (7.12). We obtain that q must satisfy

$$q = 1+2s\left(e^{\mathrm{i}\alpha h}+e^{-\mathrm{i}\alpha h}-2\right)q \implies q=\frac{1}{1+4s\sin^2\left(\frac{\alpha h}{2}\right)}.$$

In other words $|q|<1$ for all values of s and the backward-time, centred space method is unconditionally stable and, hence, unconditionally convergent.

7.4.4 A second order method in both time and space: Crank-Nicolson

The backward-time centred-space approximation of the diffusion equation (7.6) has a major advantage over explicit methods: it is unconditionally stable. However, in one respect it is not an improvement on the explicit method: it is only first order in time, even though it is second order in space. Using a second order finite difference approximation of the time derivative would be an obvious improvement.

Crank and Nicholson (1947) proposed approximating the partial derivative u_t using a finite difference centred at the time value $t^n+\delta/2\equiv t^{n+1/2}$:

$$\frac{\partial u}{\partial t}=\frac{u_i^{n+1}-u_i^n}{\delta}+O(\delta^2)$$

This choice of grid point requires that also the spatial derivative should be evaluated at $t^{n+1/2}$. Crank and Nicholson suggested that the second derivative at this grid point could be approximated with the average of the second derivatives at t^n and at t^{n+1}:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{t+\delta/2} \simeq \frac{1}{2}\left(\left.\frac{\partial^2 u}{\partial x^2}\right|_{t} + \left.\frac{\partial^2 u}{\partial x^2}\right|_{t+\delta}\right).$$

With this choice of discretisation the finite difference representation of equation (7.6) is:

$$\frac{u_i^{n+1}-u_i^n}{\delta} = \frac{1}{2}\left(\frac{u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}}{h^2}+\frac{u_{i+1}^{n}-2u_i^{n}+u_{i-1}^{n}}{h^2}\right)$$

$$\Longrightarrow \quad -su_{i-1}^{n+1}+2(1+s)u_i^{n+1}-su_{i+1}^{n+1} = su_{i-1}^{n}+2(1-s)u_i^{n}-su_{i+1}^{n} \quad (7.13)$$

The system of linear equations (7.13) is once again tri-diagonal and can be solved efficiently using a Gaussian elimination algorithm.

By deriving the Modified Differential Equation (MDE) corresponding to the finite difference scheme (7.13) it is possible to show that the algorithm is indeed second order both in time and in space and that it is consistent. Using the Von Neumann method, i.e. by looking for a solution of (7.13) of the form (7.10), we obtain that

$$q = \frac{1-s\sin^2(\alpha h/2)}{1+s\sin^2(\alpha h/2)}.$$

We have $|q| < 1$ for all values of $s = \delta/h^2$ and, hence, the Crank-Nicolson method is unconditionally stable and convergent.

7.4.5 The Hopscotch method - A method for nonlinear parabolic equations

The Hopscotch method (Gourlay 1970) is an interesting combination of the forward-time centred-space (FTCS) method and the backward-time centred-space (BTCS) method. The basic idea of the hopscotch method is to make two sweeps through the solution domain at each time step. On the first sweep, the explicit FTCS method, equation (7.9), re-written here for convenience,

$$u_i^{n+1} = (1-2s)u_i^n + s(u_{i+1}^n + u_{i-1}^n) \quad (7.14)$$

is applied at every other grid point. On the second sweep, the implicit BTCS method, equation (7.12),

$$u_i^{n+1} = u_i^n + s(u_{i+1}^{n+1} + u_{i-1}^{n+1} - 2u_i^{n+1}) \quad (7.15)$$

is applied at the remaining points. Equation (7.15) appears to be implicit, but the values of $u_{i\pm1}^{n+1}$ are known from the first sweep through the grid with the explicit FTCS method. Consequently, equation (7.15) can be solved explicitly for u_i^{n+1} to give

$$(1+2s)u_i^{n+1} = u_i^n + s(u_{i+1}^{n+1} + u_{i-1}^{n+1}).$$

The pattern of explicit and implicit points is alternated at each time level. Moreover, the explicit and implicit steps can be combined in the more compact

$$u_i^{n+2} = 2u_i^{n+1} - u_i^n \tag{7.16}$$

where u_i^{n+1} has been computed using (7.15). This simplification further increases the efficiency of the method. Equation (7.14) must be used on the first time step, but equation (7.16) can be used at all subsequent time steps. Note, however, that a drawback of this approach is that the solution is known only on half the grid points at each time step. If the complete solution is needed, for example, to produce a graph, then the implicit step (7.15) should follow the combined step (7.16). To restart the iteration, the explicit step (7.14) is used once more followed by (7.16).

The hopscotch method, applied to the diffusion equation, is stable and consistent. However, in order to guarantee an accurate result as δ and h tend to zero we have to require that the mesh ratio δ/h^2 is constant. In other words, in this respect the Hopscotch method is no better than the BTCS method (but is simpler to implement, especially if the equation is non linear, see below).

If the equation is nonlinear then the implicit step becomes a nonlinear algebraic equation for u_i^n. However, we can approximate the nonlinear terms by computing the nonlinearity using an average value for u_i^n:

$$\bar{u}_i^n = \frac{1}{4}(u_{i+1}^n + u_{i-1}^n + u_{i+1}^{n+1} + u_{i-1}^{n+1}) \tag{7.17}$$

7.5 Hyperbolic Equations

7.5.1 Introduction

The theory of hyperbolic equations and their solutions is quite involved and is not studied in this unit. However, it is essential to know it and understand it if any serious work with hyperbolic partial differential equations is to be attempted. Here we consider a very simple case and use it to discuss a few methods to integrate hyperbolic partial differential equations. The methods discussed apply also to more complex cases, but care should be taken to ensure that the solutions obtained are acceptable.

As an example of a hyperbolic equation we consider the advection equation

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0, \qquad a \le x, \quad v > 0. \tag{7.18}$$

This equation represents a function propagating in the positive x-direction with speed v. In order for the equation to have a unique solution we must specify an initial condition, $u(x,0) = F(x)$ and a boundary condition at $x = a$, $u(a,t) = G(t)$. Note that the domain is unbounded in the positive x-direction: numerically this implies that the numerical solution of equation (7.18) is acceptable only if the effects of the right boundary are negligible, i.e. only for the time taken for the signal to propagate across the integration region and reach the right boundary.

7.5.2 The forward-time centred-space method

The most straightforward finite difference method for solving hyperbolic partial differential equations would appear to be the forward-time centred-space (FTCS) method. Applied to the diffusion equation this method is conditionally stable; however, when applied to the convection equation this method is unconditionally unstable. The algorithm consists in replacing the time derivative in equation (7.18) with its forward difference approximation and the space derivative with the centred-difference approximation. We obtain:

$$\frac{u_i^{n+1} - u_i^n}{\delta} + v\frac{u_{i+1}^n - u_{i-1}^n}{2h} = 0$$

where δ is the time step, h is the space step, the spatial grid is $x_i = ih$, with $i = 0, 1, \ldots, N+1$, the time grid is $t^n = n\delta$ and $u_i^n = u(x_i, t^n)$. Solving for u_i^n we obtain

$$u_i^{n+1} = u_i^n - \frac{c}{2}(u_{i+1}^n - u_{i-1}^n)\,, \tag{7.19}$$

where $c = u\delta/h$ is the *convection number*. The stability analysis of this equation shows that the solution of (7.19) is $u_\ell^k = \exp(\mathrm{i}\alpha\ell h)q^k$, with

$$q = 1 - \mathrm{i}c\sin(\alpha h),$$

where $\mathrm{i} = \sqrt{-1}$. From this we obtain that

$$|q| = \sqrt{1 + c^2\sin^2(\alpha h)} > 1 \;\forall c \in \mathbb{R}.$$

Hence the FTCS method is unconditionally unstable.

7.5.3 The Lax method

Lax (1954) proposed a modification to the FTCS method for the convection equation that yields a conditionally stable method. In that modification, the value u_i^n in the finite difference approximation of the time derivative is replaced by the average of its values at the neighbouring points:

$$u_i^{n+1} = \frac{1}{2}(u_{i+1}^n + u_{i-1}^n) - \frac{c}{2}(u_{i+1}^n - u_{i-1}^n)\,, \tag{7.20}$$

This method is not consistent in general. It can be shown that the partial differential equation that is represented by equation (7.20) is

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = \frac{1}{2}\left(\frac{h^2}{\delta} - v^2\delta\right)\frac{\partial^2 u}{\partial x^2} + \frac{1}{3}\left(vh^2 - v^3\delta^2\right)\frac{\partial^3 u}{\partial x^3} + \dots\,, \tag{7.21}$$

where h and δ are the space and time discretisation steps. From this equation we can see that as h and δ tend to zero the first term on the right hand side does not vanish: on the contrary, it is undetermined. However, if the two discretisation steps tend to zero so that their ratio is constant, for example at the value $\beta = h/\delta$, then equation (7.20) approaches the equation

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = \frac{1}{2}\left(\beta h - v^2\delta\right)\frac{\partial^2 u}{\partial x^2} \tag{7.22}$$

as h and δ tend to zero. The equation (7.22) is a parabolic convection-diffusion equation. Substituting the convection number $c = v\delta/h$ into equation (7.22) we obtain

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = \frac{1}{2}vh\left(\frac{1}{c} - c\right)\frac{\partial^2 u}{\partial x^2} \tag{7.23}$$

In other words if $c \neq 1$ the Lax approximation introduces some numerical diffusion in the model that is to be integrated. This effect is common to most finite difference schemes: it is normally not important if the original equation includes a diffusive term. It may, however, give completely false results if no diffusion terms are present in the original equation. In the case of the advection equation the effect of the numerical diffusion is to smooth the signals as they propagate and decrease their amplitude.

Another effect of the numerical diffusion of the Lax algorithm is to make it conditionally stable. The Von Neumann stability analysis shows that the method is stable if

$$c = \frac{v\delta}{h} \leq 1. \tag{7.24}$$

Comparing this result with equation (7.23) we see that the scheme is stable if numerical diffusion is present. Equation (7.24) is called the Courant-Friedrichs-Lewy stability criterion. It states that the numerical speed of propagation $v_{num} = h/\delta$ must be greater than or equal to the physical speed of propagation v.

Derivation of Equation (7.21)

To obtain equation (7.21) we Taylor expand u_i^{n+1} and $u_{i\pm1}^n$ around u_i^n (indicated with u in what follows):

$$\begin{aligned} u_i^{n+1} &= u + \delta\partial_t u + \frac{\delta^2}{2}\partial_{tt}u + \frac{\delta^3}{3!}\partial_{ttt}u + O(\delta^4), \\ u_{i\pm1}^n &= u + h\partial_x u \pm \frac{h^2}{2}\partial_{xx}u \pm \frac{h^3}{3!}\partial_{xxx}u + O(h^4). \end{aligned}$$

Substituting into equation (7.20) and dividing through by δ we obtain

$$\partial_t u + v\partial_x u = -\frac{\delta}{2}\partial_{tt}u - \frac{\delta^2}{6}\partial_{ttt}u + \frac{h^2}{2\delta}\partial_{xx}u - \frac{vh^2}{6}\partial_{xxx}u + O(\delta^3) + O(h^3) \quad (7.25)$$

We now wish to replace the derivatives with respect to time with derivatives respect to space using this relation. At first order in δ we have that [we indicate with $O(\delta^n)$ terms that are of the order of n in either δ or h]:

$$\partial_t u = -v\partial_x u + O(\delta) \quad (7.26)$$

so that

$$\partial_{tt}u = -v\partial_x\partial_t u + O(\delta) = v^2\partial_{xx}u + O(\delta) \quad (7.27)$$

This is not accurate enough to be replaced into equation (7.25): we need to obtain an expression correct up to second order in the space and time steps. Starting once again from equation (7.25) we have

$$\partial_t u = -v\partial_x u - \frac{\delta}{2}\partial_{tt}u + O(\delta^2).$$

Differentiating with respect to time on both sides and making use of (7.26) and (7.27) we obtain:

$$\begin{aligned} \partial_{tt}u &= -v\partial_x\partial_t u - \frac{\delta}{2}\partial_t\partial_{tt}u + O(\delta^2) \\ &= -v\partial_x\left(-v\partial_x u - \frac{\delta}{2}\partial_{tt}u\right) - \frac{\delta}{2}\partial_t v^2\partial_{xx}u + O(\delta^2) \\ &= v^2\partial_{xx}u + v\partial_x\left(\frac{\delta}{2}v^2\partial_{xx}u\right) + v^3\frac{\delta}{2}\partial_{xxx}u + O(\delta^2) \\ &= v^2\partial_{xx}u + \delta v^3\partial_{xxx}u + O(\delta^2). \end{aligned}$$

Differentiating equation (7.27) with respect to time we obtain

$$\partial_{ttt}u = -v^3\partial_{xxx}u + O(\delta),$$

and substituing both these expression into equation (7.25) we (finally) obtain equation (7.21).

7.5.4 Upwind methods

A salient feature of hyperbolic equations is that they describe the propagation of information. In the case of the advection equation (7.18) the information propagates from negative to positive x with speed v. This type of information propagation is referred to as *upwind* propagation, since the information comes from the direction from which the convection velocity comes, that is, the upwind direction. Finite difference methods that account for the upwind influence are called *upwind* methods.

The simplest procedure for developing an upwind finite difference equation is to replace the time derivative by the first-order forward-difference approximation and the space derivative by the first-order one-sided-difference approximation in the upwind direction. For $v > 0$ the finite difference upwind approximation of equation (7.18) is

$$\frac{u_i^{n+1} - u_i^n}{\delta} + v\frac{u_i^n - u_{i-1}^n}{h} = 0 \implies u_i^{n+1} = u_i^n - c(u_i^n - u_{i-1}^n), \tag{7.28}$$

where $c = v\delta/h$ is the convection number. The upwind method is consistent, first order in time and space and stable if $c \leq 1$, i.e. if the convection number satisfies the Courant-Friedrichs-Lewy condition. As in the case of the Lax scheme numerical diffusion (and dispersion) is present unless $c = 1$.

Further reading

Topics covered here are also covered in

- Chapters 12 and 13 of Linz & Wang, *Exploring Numerical Methods* (QA297 LIN),
- Chapter 9 of Kincaid & Cheney, *Numerical Analysis* (QA297 KIN),
- Parts II and III (especially chapters 13 and 14) of Iserles, *A First Course in the Numerical Analysis of Differential Equations* (QA297 ISE).

Chapter 8

Eigenvalues

8.1 Introduction

The eigenvalues and eigenvectors of a matrix are of fundamental importance in many matrix problems: for example, in matrix descriptions of physical models they often represent fundamental physical quantities, like the energy levels of an atom or the frequency of vibrations of a string. In numerical analysis problems, the eigenvalues determine the convergence conditions of many iterations schemes, like Jacobi and Gauss-Seidel.

It is, therefore, important to be able to compute quickly and accurately the eigenvalues and, eventually, the eigenvectors of a square matrix. This is an apparently simple problem that could be tackled using standard nonlinear equations techniques. In fact, the eigenvalues λ of a $n \times n$ square matrix A are the n roots of the characteristic polynomial of A,

$$\det(A - \lambda \mathrm{Id}) = 0.$$

We could envisage using a root finding algorithm to detect all the roots of the polynomial and, hence, all the eigenvalues of the matrix A. This is the procedure that is used in analytical calculations when finding the eigenvalues of small (2×2 or 3×3) matrices. However, roots of high order polynomials may be very sensitive to the values of the coefficients: therefore, all the unavoidable numerical errors involved in the lengthy calculation of the coefficients of the characteristic polynomial of A may well be amplified at the stage of computing the roots and produce estimates that bear very little relation to the eigenvalues of A.

Example 8.1 *Polynomials may be badly conditioned, in the sense that a small change in the values of just one coefficient changes considerably the values of the roots.*

Consider the polynomial

$$\begin{aligned} p(z) &= (z-1)(z-2)(z-3)(z-4)(z-5) \\ &= -120 + 274z - 225z^2 + 85z^3 - 15z^4 + z^5. \end{aligned}$$

The roots of this fifth order polynomial are quite obviously $\{1, 2, 3, 4, 5\}$. The roots of

$$p(z) = -120 + 274z - 225z^2 + 85z^3 - 15z^4 + 1.01z^5$$

are $\{0.999, 2.067, 2.624, 4.580 \pm 0.966\, i\}$. ■

It is therefore necessary to develop techniques to identify the eigenvalues of a matrix A that do not rely on finding the roots of the characteristic polynomial.

If we are interested in just a few eigenvalues then we can use one of the many variations of the *power method*. Obtaining the full spectrum is a much harder problem; an efficient algorithm to do so, the QR-algorithm, was developed by J.G.F. Francis in 1961. We describe the power method first and give a relatively simple account of the QR-algorithm after. The chapter is based in its entirety on the book by Kincaid and Cheney (chapter 5).

8.2 The power method

8.2.1 The method

The power method is an iterative procedure designed to compute the dominant eigenvalue of a matrix and its corresponding eigenvector. The method is based on two assumptions:

1. There is a single eigenvalue of maximum modulus.
2. There is a linearly independent set of n eigenvectors, where n is the size of the matrix.

According to the first assumption the eigenvalues of A can be labelled so that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots |\lambda_n|$$

According to the second assumption there is a basis of eigenvectors $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n\}$ in $\mathbb{C}^n$ such that:

$$A\boldsymbol{u}_j = \lambda_j \boldsymbol{u}_j, \qquad 1 \leq j \leq n.$$

To start the iteration we need to define an initial vector $\boldsymbol{x}^{(0)}$ and we have to require that its component along the eigenvector $\boldsymbol{u}_1$ is not zero:

$$\boldsymbol{x}^{(0)} = \sum_{j=1}^{n} a_j \boldsymbol{u}_j, \qquad a_1 \neq 0.$$

Consider now the vector

$$\boldsymbol{x}^{(k)} = A\boldsymbol{x}^{(k-1)} = \ldots = A^k \boldsymbol{x}^{(0)}.$$

Its decomposition on the basis of the eigenvectors is

$$\boldsymbol{x}^{(k)} = \sum_{j=1}^{n} \lambda_j^k \boldsymbol{u}_j = \lambda_1^k \left[a_1 \boldsymbol{u}_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^k a_2 \boldsymbol{u}_2 + \left(\frac{\lambda_3}{\lambda_1}\right)^k a_3 \boldsymbol{u}_3 + \ldots \left(\frac{\lambda_n}{\lambda_1}\right)^k a_n \boldsymbol{u}_n \right].$$

We note that all the terms λ_j/λ_1, $j = 2, 3, \ldots, n$, have modulus strictly smaller than one. We can write this expression as

$$\boldsymbol{x}^{(k)} = \lambda_1^k (a_1 \boldsymbol{u}_1 + \boldsymbol{\varepsilon}^{(k)}) \tag{8.1}$$

where

$$\boldsymbol{\varepsilon}^{(k)} \equiv \sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_1}\right)^k a_j \boldsymbol{u}_j \tag{8.2}$$

is such that

$$\|\boldsymbol{\varepsilon}^{(k)}\| = O\left(\left|\frac{\lambda_j}{\lambda_1}\right|^k\right) \implies \lim_{k\to\infty} \|\boldsymbol{\varepsilon}^{(k)}\| = 0. \tag{8.3}$$

As we have assumed that $a_1 \neq 0$ we can write

$$\boldsymbol{x}^{(k)} = \lambda_1^k a_1 \boldsymbol{u}_1 + O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right). \tag{8.4}$$

This expression is at the heart of the power method: at each iteration the vector $\boldsymbol{x}^{(k)}$ aligns itself more and more along the eigenvector $\boldsymbol{u}_1$ and is multiplied by a (complex) number that gets closer and closer to the eigenvalue λ_1. In order to obtain a quantitative estimate of the eigenvalue we need to pass from vectors to numbers. As a simple example, we define the ratio

$$r_k = \frac{\|\boldsymbol{x}^{(k+1)}\|}{\|\boldsymbol{x}^{(k)}\|} = |\lambda_1| \frac{\|a_1 \boldsymbol{u}_1 + \boldsymbol{\varepsilon}^{(k+1)}\|}{\|a_1 \boldsymbol{u}_1 + \boldsymbol{\varepsilon}^{(k)}\|}.$$

Using equation (8.3) we then have that

$$\lim_{k\to\infty} r_k = |\lambda_1|.$$

Therefore, a first approximate algorithm to implement the power method is as follows:

1. Chose an initial vector $\boldsymbol{x}^{(0)}$ such that its component along $\boldsymbol{u}_1$ is not zero.

2. From the vector at iteration k, $\boldsymbol{x}^{(k)}$, compute the vector at iteration $k+1$, $\boldsymbol{x}^{(k+1)} = A\boldsymbol{x}^{(k)}$.
3. Compute the ratio $r_k = \|\boldsymbol{x}^{(k+1)}\|/\|\boldsymbol{x}^{(k)}\|$.
4. If the ratio r_k has converged to $|\lambda_1|$ stop; otherwise repeat from step 2.

Remark 1 - In practice, it is impossible to know a priori that the vector $\boldsymbol{x}_0$ satisfies the condition that its component along $\boldsymbol{u}_1$ is not zero. However, a random choice of vector will, with great probability, satisfy it. In general, however, this is a moot point: numerical errors will always induce a non-zero coefficient a_1 that will be amplified at each iteration and the method will ultimately converge to the dominant eigenvalue.

Remark 2 - From equation (8.1) we see that $\|\boldsymbol{x}^{(k)}\| \simeq |\lambda_1|^k$. Therefore if $|\lambda_1| > 1$ we have that $\|\boldsymbol{x}^{(k)}\| \to \infty$ as $k \to \infty$. This may induce an overflow problem for sufficiently large k. If, on the other hand, $|\lambda_1| < 1$, then $\|\boldsymbol{x}^{(k)}\| \to 0$ as $k \to \infty$. This may induce an underflow problem for sufficiently large k. To avoid either pitfall it is convenient to normalise the vectors $\boldsymbol{x}^{(k)}$ at each iteration (or after a small number of iterations). A suitable algorithm is:

1. Chose an initial vector $\boldsymbol{x}^{(0)}$ such that its component along $\boldsymbol{u}_1$ is not zero and $\|\boldsymbol{x}^{(0)}\| = 1$.
2. From the vector at iteration k, $\boldsymbol{x}^{(k)}$, compute the vector at iteration $k+1$, $\boldsymbol{x}^{(k+1)} = A\boldsymbol{x}^{(k)}$.
3. Compute $r_k = \|\boldsymbol{x}^{(k+1)}\|$ and assign $\|\boldsymbol{x}^{(k+1)}\|/r_k \to \|\boldsymbol{x}^{(k+1)}\|$.
4. If the ratio r_k has converged to $|\lambda_1|$ stop; otherwise repeat from step 2.

Remark 3 - The method as described so far returns only the modulus of the eigenvalue. However, it is fairly simple to modify it so that it returns the eigenvalue itself. We lose information on the phase of the eigenvalue when we compute the ratio r_k using the norm of $\|\boldsymbol{x}^{(k+1)}\|$ and $\|\boldsymbol{x}^{(k)}\|$. We could use, instead, a linear function ϕ from $\mathbb{C}^n \to \mathbb{C}$, that is, $\phi(\alpha\boldsymbol{x} + \beta\boldsymbol{y}) = \alpha\phi(\boldsymbol{x}) + \beta\phi(\boldsymbol{y})$. (Note that $\|\boldsymbol{x}\|$ is not linear!) A possible choice for $\phi(\boldsymbol{x})$ is, for example, the sum of the components of $\boldsymbol{x}$. We can define the ratio r_k in terms of the functional $\phi(\boldsymbol{x})$ as

$$r_k = \frac{\phi(\boldsymbol{x}^{(k+1)})}{\phi(\boldsymbol{x}^{(k)})} = \lambda_1 \frac{a_1\phi(\boldsymbol{u}_1) + \phi(\boldsymbol{\varepsilon}^{(k+1)})}{a_1\phi(\boldsymbol{u}_1) + \phi(\boldsymbol{\varepsilon}^{(k)})}. \tag{8.5}$$

Using equation (8.3) and the fact that $\lim_{\|\boldsymbol{x}\|\to 0}\phi(\boldsymbol{x}) = 0$ we then have that

$$\lim_{k\to\infty} r_k = \lambda_k.$$

The algorithm that implements this scheme is as follows:

1. Chose an initial vector $\boldsymbol{x}^{(0)}$ such that its component along $\boldsymbol{u}_1$ is not zero.
2. At each iteration $k = 0, 1, 2, \ldots$ compute $d_k = \|\boldsymbol{x}^{(k)}\|$ and assign $\|\boldsymbol{x}^{(k)}\|/d_k \rightarrow \|\boldsymbol{x}^{(k)}\|$.
3. From the vector at iteration k, $\boldsymbol{x}^{(k)}$, compute the vector at iteration $k+1$, $\boldsymbol{x}^{(k+1)} = A\boldsymbol{x}^{(k)}$.
4. Compute $r_k = \phi(\boldsymbol{x}^{(k+1)})/\phi(\boldsymbol{x}^{(k)})$.
5. If the ratio r_k has converged to λ_1 stop; otherwise repeat from step 2.

8.2.2 Rate of convergence

We now show that the power method converges linearly. We define $\mu = \lambda_2/\lambda_1$. This parameter controls the convergence rate of the method. We apply Taylor's theorem to equation (8.5) and use equation (8.3) to obtain

$$r_k = \lambda_1 \frac{a_1\phi(\boldsymbol{u}_1) + \phi(\boldsymbol{\varepsilon}^{(k+1)})}{a_1\phi(\boldsymbol{u}_1) + \phi(\boldsymbol{\varepsilon}^{(k)})} = \lambda_1 \left[1 - \frac{\phi(\boldsymbol{\varepsilon}^{(k)})}{a_1\phi(\boldsymbol{u}_1)}\right] + O(\mu^{k+1}).$$

Therefore the relative error of the approximation, defined as

$$\epsilon^{(k)} \equiv \left|\frac{r_k - \lambda_1}{\lambda_1}\right|, \tag{8.6}$$

is given by

$$\epsilon^{(k)} = \left|\frac{\phi(\boldsymbol{\varepsilon}^{(k)})}{a_1\phi(\boldsymbol{u}_1)}\right| + O(\mu^{k+1}) = c_k\mu^k \tag{8.7}$$

where $\{c_k\}$ is a bounded sequence. Note that this equation implies that $\epsilon^{(k)} \propto \mu\epsilon^{(k+1)}$, i.e. that the rate of decrease of the error is linear. In fact one can show that

$$\frac{r^{(k+1)} - \lambda_1}{\lambda_1} = (c + \delta_k)\frac{r^{(k)} - \lambda_1}{\lambda_1}$$

where $|c| < 1$ and $\lim_{k\rightarrow\infty} \delta_k = 0$.

Remark 1 - The condition that there should be a single eigenvalue of maximum modulus should be qualified. If the eigenvalue is multiple then the power method will converge to it, but it will not be able to return the eigenvectors associated to the (multiple) eigenvalue; it will just return a random vector in the eigenspace of the eigenvalue of maximum modulus. If, on the other hand, the matrix has distinct eigenvalues all with the same maximum modulus (a rather common situation in problems with symmetry), the method may fail to converge.

Remark 2 - The convergence can be improved using Aitken's acceleration, a technique similar to Richardson's extrapolation that uses the fact that the error decreases linearly to improve the convergence.

8.2.3 Variations on the power method

The inverse power method

We can use a variation of the power method, called the *inverse power method*, to compute the smallest eigenvalue of a non singular (and well conditioned) matrix. The algorithm is based on the result of linear algebra that if λ is an eigenvalue of a nonsingular matrix A, then λ^{-1} is an eigenvalue of A^{-1}.

We assume as before that the eigenvectors of A form a basis for $\mathbb{C}^n$, but we now require that the eigenvalue of smallest modulus is isolated, i.e. that we can label the eigenvalues of A so that

$$|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_{n-1}| > |\lambda_n| > 0.$$

Therefore the eigenvalues of A^{-1} are ordered as

$$|\lambda_n^{-1}| > |\lambda_{n-1}^{-1}| \geq \ldots \geq |\lambda_1^{-1}| > 0.$$

We are now in the position to compute λ_n^{-1} using the power method applied to A^{-1}.

Remark - It is <u>not</u> a good idea to compute A^{-1} first and then use $\boldsymbol{x}^{(k+1)} = A^{-1}\boldsymbol{x}^{(k)}$. It is better to obtain $\boldsymbol{x}^{(k+1)}$ by solving the linear system

$$A\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)}$$

using an efficient (and accurate) linear solver. The LU decomposition is an excellent candidate for this problem.

8.2.4 The shifted inverse power method

The shifted inverse power method is used to compute the eigenvalue of A closest to a given complex number ν. Suppose that there is a real number ϵ such that there is only one eigenvalue of A, λ_j, such that

$$|\lambda_j - \nu| \leq \epsilon \quad \text{and} \quad |\lambda_i - \nu| > \epsilon \; \forall\, i \neq j. \tag{8.8}$$

Consider now the "shifted" matrix $A - \nu\text{Id}$: its eigenvalues are given by $\lambda_i - \nu$. Equation (8.8) implies that we can find $\lambda_j - \nu$ by applying the inverse power method to the matrix $A - \nu\text{Id}$. This algorithm is called the shifted inverse power method.

8.3 The QR-algorithm

8.3.1 Introduction

The methods to compute the full spectrum of a matrix are based on the idea that finding the eigenvalues is an easy problem for certain classes of matrices, e.g. diagonal and triangular matrices: their eigenvalues are the elements along the diagonal. The trick is to find operations that transform a matrix in an "easy" form while not altering the spectrum. The first step in doing so is to identify classes of matrices that have the same spectrum and classes of operations that do not alter the spectrum.

Definition 8.1 *Two square matrices are similar if there exists a nonsingular matrix P such that $B = PAP^{-1}$. We say that B is obtained from A using a similarity transformation. If P is unitary, i.e. $P = P^{\dagger}$, the two matrices are unitary similar.*

Theorem 8.3.1 *Similar matrices have the same eigenvalues.*

Proof

$$A\boldsymbol{x} = \lambda\boldsymbol{x} \quad \Leftrightarrow \quad (PAP^{-1})(P\boldsymbol{x}) = PA\boldsymbol{x} = \lambda(P\boldsymbol{x}).$$

So if $\boldsymbol{x}$ is an eigenvector of with eigenvalue λ then $A\ P\boldsymbol{x}$ is an eigenvector of PAP^{-1} with the same eigenvalue. Moreover, P provides a bijection between the eigenvectors of A and the eigenvectors of PAP^{-1}. ■

In other words, similar matrices have the same spectrum and similarity transformations do not change the spectrum of a matrix.

8.3.2 Schur's theorem

Theorem 8.3.1 suggests a strategy to find the eigenvalues of matrix: find a similarity transformation $B = PAP^{-1}$ such that B is triangular. The following theorem states that this strategy is always (theoretically) possible.

Theorem 8.3.2 (Schur) *Every square matrix is unitary similar to a triangular matrix.*

In order to prove this theorem we need the following two lemmas. In the remainder of this section, we always use the inner product $(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^{\dagger}\boldsymbol{v}$ on $\mathbb{C}^n$ and the associated norm $\|\boldsymbol{u}\| = \sqrt{\boldsymbol{u}^{\dagger}\boldsymbol{u}}$.

Lemma 8.1 *The matrix $Id - \boldsymbol{u}\boldsymbol{u}^{\dagger}$ is unitary if and only if $\|\boldsymbol{u}\| = \sqrt{2}$ or $\boldsymbol{u} = 0$.*

Proof

$$(\mathrm{Id}-\boldsymbol{u}\boldsymbol{u}^\dagger)(\mathrm{Id}-\boldsymbol{u}\boldsymbol{u}^\dagger)^\dagger = (\mathrm{Id}-\boldsymbol{u}\boldsymbol{u}^\dagger)^2 = \mathrm{Id}-2\boldsymbol{u}\boldsymbol{u}^\dagger+\boldsymbol{u}\boldsymbol{u}^\dagger\boldsymbol{u}\boldsymbol{u}^\dagger = \mathrm{Id}+(\boldsymbol{u}^\dagger\boldsymbol{u}-2)\boldsymbol{u}\boldsymbol{u}^\dagger$$

■

Lemma 8.2 *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors such that $\|\boldsymbol{x}\| = \|\boldsymbol{y}\|$ and $(\boldsymbol{x}, \boldsymbol{y})$ is real. Then there exists an unitary matrix $U = Id - \boldsymbol{u}\boldsymbol{u}^\dagger$ such that $U\boldsymbol{x} = \boldsymbol{y}$. The vector $\boldsymbol{u}$ is given by:*

$$\boldsymbol{u} = \sqrt{2}\frac{\boldsymbol{x}-\boldsymbol{y}}{\|\boldsymbol{x}-\boldsymbol{y}\|}.$$

Proof

$$\begin{aligned} U\boldsymbol{x} &= \left(\mathrm{Id} - \frac{2}{\|\boldsymbol{x}-\boldsymbol{y}\|^2}(\boldsymbol{x}-\boldsymbol{y})(\boldsymbol{x}-\boldsymbol{y})^\dagger\right)\boldsymbol{x} \\ &= \boldsymbol{x} - \frac{2(\|\boldsymbol{x}\|^2 - (\boldsymbol{y},\boldsymbol{x}))}{\|\boldsymbol{x}\|^2 - (\boldsymbol{x},\boldsymbol{y}) - (\boldsymbol{y},\boldsymbol{x}) + \|\boldsymbol{y}\|^2}(\boldsymbol{x}-\boldsymbol{y}) = \boldsymbol{y} \end{aligned}$$

■

Remark - The matrix U is called a *Householder reflection* and its action can be interpreted geometrically as: "the vector $\boldsymbol{y}$ can be obtained by reflecting the vector $\boldsymbol{x}$ with respect to a plane orthogonal to the vector $\boldsymbol{u}$".

Proof of Schur's theorem - We prove this theorem for two reasons: firstly, the proof is an example of the techniques used to transform a given matrix in an "easy" matrix. Secondly, the proof clearly shows why this theorem is of little practical use.

In a nutshell the proof consists in constructing a succession of unitary transformations that change the given $n \times n$ matrix A in upper triangular form, one row and column at a time.

The proof of the theorem is by induction on the size n of the matrix. The theorem is clearly true for $n = 1$. We now suppose that it is true for all matrices of order $n-1$ and prove that it must also hold for any matrix A of order n. The idea of the proof is to use Lemma 8.2 to find a unitary transformation U that acts on the first column of A and transforms it in a scalar multiple of the vector $\boldsymbol{e}_1 = (1, 0, 0, \ldots, 0)$, i.e.

$$U\left(\begin{array}{c|ccc} a_{11} & a_{12} & \ldots & a_{1n} \\ \hline a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & & \ddots & \\ a_{n1} & a_n2 & \ldots & a_{nn} \end{array}\right)U^\dagger = \left(\begin{array}{c|c} \mu & \boldsymbol{w} \\ \hline 0 & \\ \vdots & A_{n-1} \\ 0 & \end{array}\right) \tag{8.9}$$

where A_{n-1} is an $(n-1) \times (n-1)$ matrix, $\boldsymbol{w}$ is a row vector of dimension $n-1$ and μ is a number. Unfortunately, in order to build the transformation matrix U we need one eigenvalue λ of A with the corresponding eigenvector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$, chosen such that $\|\boldsymbol{v}\| = 1$. Following Lemma 8.2 we define the (complex) number

$$\beta = \begin{cases} \dfrac{v_1}{|v_1|} & v_1 \neq 0 \\ 1 & v_1 = 0 \end{cases}$$

and the vector

$$\boldsymbol{u} = \frac{\sqrt{2}}{\|\boldsymbol{v} - \beta\boldsymbol{e}_1\|}(\boldsymbol{v} - \beta\boldsymbol{e}_1).$$

Note that $(\boldsymbol{v}, \beta\boldsymbol{e}_1) = v_1^* v_1/|v_1| = |v_1| \in \mathbb{R}$ and that $\|\boldsymbol{v}\| = \|\beta\boldsymbol{e}_1\| = 1$ so that both the hypotheses of Lemma 8.2 are satisfied. Therefore, the matrix

$$U = \mathrm{Id} - \boldsymbol{u}\boldsymbol{u}^\dagger \tag{8.10}$$

is unitary and such that

$$U\boldsymbol{v} = \beta\boldsymbol{e}_1 \implies U^\dagger\boldsymbol{e}_1 = \frac{1}{\beta}\boldsymbol{v}. \tag{8.11}$$

The matrix defined by this equation is the transformation required for (8.9) to be valid. To verify that this is the case we note that the product of any $n \times n$ matrix with $\boldsymbol{e}_1$ is its first column vector. We therefore proceed to verify that the first column vector of $UAU^\dagger$ is a scalar multiple of $\boldsymbol{e}_1$:

$$UAU^\dagger\,\boldsymbol{e}_1 = UA\frac{1}{\beta}\boldsymbol{v} = \frac{1}{\beta}U\lambda\boldsymbol{v} = \frac{\lambda}{\beta}U\boldsymbol{v} = \lambda\boldsymbol{e}_1.$$

Hence the application of U defined in (8.10) to A reduces this matrix to the form (8.9) with $\mu = \lambda$.

To complete the proof we note that by the induction hypothesis there is a unitary matrix Q_{n-1} such that $Q_{n-1}A_{n-1}Q_{n-1}^\dagger$ is triangular. Therefore, the unitary transformation that reduces A to triangular form is

$$V = \left(\begin{array}{c|c} 1 & 0\ldots0 \\ \hline 0 & \\ \vdots & Q_{n-1} \\ 0 & \end{array}\right) U$$

■

At each step of the proof we assume that the eigenvalues exist and use them, but we do not address the question of how to compute them. In other words, if we know the eigenvalues then we can apply the method described in the proof to find the unitary transformation required to put A in upper triangular form. However, if we do not know them we cannot apply the procedure and we are no nearer to solving the original eigenvalue problem.
Remark - One could think of using the power method to find one eigenvalue of A and thus build U and A_{n-1}, and then apply the same trick to A_{n-1} and so on. This method is called *deflation* and it works, but is rather sensitive to numerical errors.

Corollary 8.1 *Every Hermitian matrix is unitary similar to a diagonal matrix.*

Proof - If A is Hermitian then $A = A^\dagger$. Let U be the unitary matrix such that $UAU^\dagger$ is upper triangular. Then $(UAU^\dagger)^\dagger$ is lower triangular. But

$$(UAU^\dagger)^\dagger = (U^\dagger)^\dagger A^\dagger U^\dagger = UAU^\dagger.$$

Thus, the matrix $UAU^\dagger$ is both upper and lower triangular; hence, it must be a diagonal matrix.

8.3.3 Householder's QR-factorisation

In order to introduce the QR-algorithm of Francis to find the eigenvalues of a matrix, we must first take a detour and discuss *orthogonal factorisations* of a matrix. An orthogonal factorisation consists in writing a given matrix A as the product of other matrices, some of which are orthogonal. The most useful orthogonal factorisation if the QR-factorisation developed by Alston Householder: its objective is to factor an $m \times n$ matrix A into a product

$$A = Q\,R \tag{8.12}$$

where Q is an $m \times m$ unitary matrix and R is an $m \times n$ upper triangular matrix.

The QR-factorisation is used not only to find the eigenvalues of a matrix, but also to solve least-squares problems and ill-posed problems. The algorithm produces the factorisation

$$Q^\dagger A = R \tag{8.13}$$

where $Q^\dagger$ is built step by step as the product of unitary matrices that at step $k = 0, 1, \ldots, m-1$ have the form

$$U_k = \begin{pmatrix} \mathrm{Id}_{k-1} & 0 \\ 0 & \mathrm{Id}_{m-k+1} - \boldsymbol{u}\boldsymbol{u}^\dagger \end{pmatrix}, \quad k = 1, 2, \ldots, n-1, \tag{8.14}$$

with Id_k the $k \times k$ identity matrix and $\boldsymbol{u} \in \mathbb{C}^{m-k+1}$. These matrices are called *reflections* or *Householder transformations*: the aim of each of them is to transform the matrix A into a matrix that looks more and more like R, in a similar way that a triangular matrix is constructed in the proof of Schur's theorem.

At the first step we wish to determine a vector $\boldsymbol{u} \in \mathbb{C}^m$ such that $\mathrm{Id}_m - \boldsymbol{u}^\dagger \boldsymbol{u}$ is unitary and $(\mathrm{Id}_m - \boldsymbol{u}^\dagger \boldsymbol{u})A$ has first column of the form suitable for an upper triangular matrix, i.e. of the type $(\beta, 0, \ldots, 0)^T$. We indicate with $\boldsymbol{a}_1$ the first column of A and with $\boldsymbol{e}_1$ the vector $(1, 0, \ldots, 0)$. Following Lemma 8.2 we define the (complex) number

$$\beta = \begin{cases} -\dfrac{a_{11}}{|a_{11}|}\|\boldsymbol{a}_1\|_2 & a_{11} \neq 0 \\ \|\boldsymbol{a}_1\|_2 & a_{11} = 0 \end{cases}$$

and the vector

$$\boldsymbol{u} = \frac{\sqrt{2}}{\|\boldsymbol{a}_1 - \beta \boldsymbol{e}_1\|}(\boldsymbol{a}_1 - \beta \boldsymbol{e}_1).$$

By Lemma 8.2 the matrix

$$U_1 = \mathrm{Id} - \boldsymbol{u}\boldsymbol{u}^\dagger \tag{8.15}$$

is unitary and such that

$$U_1 \boldsymbol{a}_1 = \beta \boldsymbol{e}_1. \tag{8.16}$$

Using the unitary matrix U defined in equation (8.15) we have that the matrix $U_0 A$ has form:

$$U_1 A = \left(\begin{array}{c|c} \beta & \boldsymbol{w}_1 \\ \hline 0 & \\ \vdots & \tilde{A}_1 \\ 0 & \end{array}\right) \equiv R_1$$

We can now repeat the same procedure for the first column of the matrix A_1 and build the unitary operator U_k defined in equation (8.14) with $k = 2$ and so on. At stage k in the process, we shall have multiplied A on the left by k unitary matrices and the result will be a matrix having its first k columns of the correct form, i.e. with 0's below the diagonal:

$$U_k\, U_{k-1}\, U_{k-2}\, \ldots\, U_1 A = \left(\begin{array}{c|c} J & H \\ \hline 0 & W \end{array}\right)$$

where J is an upper triangular $k \times k$ matrix, 0 is an $(m-k) \times k$ null matrix, H is $k \times (n-k)$ and W is $(m-k) \times (m-k)$.

The process terminates at $k = (n-1)$ when R_{n-1} is in the correct form for equation (8.12), i.e. $R = R_{n-1}$. Moreover,

$$Q^\dagger = U_{n-1} \ldots U_1 \implies Q = U_1 \ldots U_{n-1}.$$

8.3.4 The QR-Algorithm of Francis

The basic algorithm

The QR-Algorithm of Francis is an iterative procedure that uses a variant of the QR-factorisation to find the eigenvalues of a $n \times n$ matrix A by reducing to triangular form.

We can interpret the QR-algorithm as an extension of the power method - from Epperson (2002):

> One of the primary drawbacks to the power methods is that they work on a single vector at a time; if we want all the eigenpairs of a matrix, then we spend a lot of computation time in deflation [...]. Why not do the power method on several vectors at once? Let Z_0 be an initial guess matrix and do the computation
>
> $$Y_{k+1} = AZ_k, \qquad Z_{k+1} = Y_{k+1}D_{k+1}$$
>
> where D_{k+1} is some properly chosen diagonal scaling matrix. The problem is that this won't work; the repeated multiplications by A will simply cause all of the columns of Y_{k+1} to line up on the direction of the dominant eigenvalue and so we will converge to n copies of the dominant eigenpair - hardly an improvement on the power method.
>
> This algorithm does have something to recommend it, though. If the Z_k matrices are all *orthogonal* matrices, then we get something that might work, because the orthogonality of the columns of Z_k would prevent the columns of Y_{k+1} from all lining up in the same direction.

The QR-algorithm implements this idea using the QR factorisation. Additional information on its relation to the power method and deflation techniques can be found in Burden & Faires (2005, 8th ed.).

The QR-algorithm consists of the following steps:

1. Start the iteration - Iteration index $k = 1$, working matrix $A_k = A$.

2. Compute the QR factorisation of $A_k = Q_k R_k$, where Q_k is unitary and R is upper triangular with non-negative diagonal.

3. Compute $A_{k+1} = R_k Q_k$. Under appropriate hypotheses, the matrix A_{k+1} tends, as $k \to \infty$, to an upper triangular matrix whose diagonal elements are the eigenvalues of A . The procedure is repeated from step (2) until convergence has been attained.

Remark 1 - All the matrices A_k are similar to A:

$$A_k = Q_k R_k = (Q_k R_k)\left(Q_k Q_k^\dagger\right) = Q_k\left(R_k Q_k\right) Q_k^\dagger = Q_k A_{k+1} Q_k^\dagger.$$

Remark 2 - If the matrix A is real, then all the matrices A_k are real. Therefore, if A has non-real eigenvalues then at most A_k will converge to a “triangular” matrix with 2×2 minors on its diagonal.
Remark 3 - In order to reduce the burden of computation and increase the speed of convergence the QR-algorithm is coupled with other techniques, namely the reduction to upper Hessenberg form and an origin shift (*Shifted QR-factorisation*).

Further reading

Topics covered here are also covered in

- Chapter 9 of Linz & Wang, *Exploring Numerical Methods* (QA297 LIN),
- Chapter 5 of Süli & Mayers, *An Introduction to Numerical Analysis* (not in library),
- Chapters 7 and 8 of the extremely detailed Golub & Van Loan, *Matrix Computations* (QA263 GOL).

Appendix A

Key Mathematical Results

There are a number of key results that you are expected to know. These will have been covered in previous courses. For rigorous statements and proofs you should consult books on the reading list or basic calculus or linear algebra texts.

A.1 Linear Algebra

A vector $\boldsymbol{v}$ of length N is a set of N numbers ordered either as a row or column. In either case the notation v_i is used to denote the coefficients.

The dot product of a row vector $\boldsymbol{x}$ with a column vector $\boldsymbol{y}$ is given by

$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i=1}^{n} x_i \cdot y_i. \tag{A.1}$$

A matrix A of size N is a set of N^2 numbers ordered into N rows and N columns. These numbers are called the coefficients, denoted a_{ij}, i indicating the row and j the column.

Matrix-matrix multiplication written $AB = C$ defines the coefficients of the matrix C in terms of the (given) matrix coefficients of A and B by

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}. \tag{A.2}$$

In other words, the c_{ij} coefficient is given by the dot product of the i^{th} row of A with the j^{th} column of B.

Matrix-vector multiplication $A\boldsymbol{x} = \boldsymbol{b}$ defines the coefficients of the column vector $\boldsymbol{b}$ in terms of the gives matrix A and column vector $\boldsymbol{x}$; as expected this is given by

$$b_i = \sum_{l=1}^{n} a_{il} x_l. \tag{A.3}$$

The system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$ where $A, \boldsymbol{b}$ are known and $\boldsymbol{x}$ is unknown, has a unique solution if, and only if, the determinant of the matrix A is non-zero.

Eigenvectors of a matrix A are any vector $\boldsymbol{x}$ such that $A\boldsymbol{x} = \lambda\boldsymbol{x}$. The value λ is called an eigenvalue. Eigenvalues can be found from the characteristic polynomial $\det(A - \lambda I) = 0$, where I is the $N \times N$ identity matrix (this is not a practical way of computing eigenvalues when N is large). Eigenvectors are only defined up to a multiplicative constant (i.e., if $\boldsymbol{x}$ is an eigenvector then $\alpha\boldsymbol{x}$ is an eigenvector with the same eigenvalue, provided $\alpha \neq 0$).

A *basis* is a set of N independent vectors $\boldsymbol{e}_i$ such that any vector $\boldsymbol{x}$ can be expressed as a linear combination

$$\boldsymbol{x} = \alpha_1 \boldsymbol{e}_1 + \ldots \alpha_N \boldsymbol{e}_N. \tag{A.4}$$

A particularly useful case is when the eigenvectors of a matrix A form a basis.

A.2 Taylor's theorem

If f is a continuous function on $x \in [a, b]$ and the $(n+1)^{\text{th}}$ derivative exists on (a, b), then for any point $c \in [a, b]$, we can express $f(x)$ (for any point $x \in [a, b]$) as

$$f(x) = \left\{ \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(c)\,(x-c)^k \right\} + E_n(x), \tag{A.5}$$

where, for some point ξ between c and x, the *error* or *remainder* E_n can be expressed as

$$E_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi)\,(x-c)^{n+1}. \tag{A.6}$$

Here the notation $f^{(n)}$ means the n^{th} derivative of f.

For our purposes, the most useful way of writing this will be in the form of a Maclaurin series where $c = 0$. We shall also often consider cases where $x \to h$, $|h| \ll 1$, leading to

$$f(h) = f(0) + h f^{(1)}(0) + \frac{h^2}{2} f^{(2)}(0) + \frac{h^3}{3!} f^{(3)}(0) + \cdots + \mathcal{O}(h^n). \tag{A.7}$$

The notation $\mathcal{O}(h^n)$ means that the remaining terms contain powers of h at least as big as n; when h is small it is usually reasonable to neglect these terms.

A.3 Triangle inequality

For any normed vector space V with elements $\boldsymbol{x}, \boldsymbol{y} \in V$, the triangle inequality states that

$$\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|. \tag{A.8}$$

For our purposes we will mostly be concerned with real numbers where $\|\boldsymbol{x}\|$ has the simple interpretation as the absolute value of the number x. In this case the triangle inequality simply says that the sum of the lengths of any two sides of a triangle must be greater than or equal to the length of the remaining side.

A.4 Difference equations

For linear constant coefficient ODEs

$$a_n y^{(n)}(x) + \cdots + a_1 y^{(1)}(x) + a_0 y(x) = 0 \tag{A.9}$$

there is a simple solution method in terms of the *characteristic polynomial*

$$a_n \lambda^n + \cdots + a_1 \lambda + a_0 = 0. \tag{A.10}$$

The characteristic polynomial follows from substituting the assumption $y(x) = \exp[\lambda x]$ into equation (A.9); λ is then a root of the characteristic polynomial. The general solution is the general linear combination of all solutions found from the characteristic polynomial. Note that repeated roots complicate the analysis slightly.

The same approach can be used for *difference equations*. These are equations that define sequences (of e.g. real numbers or vectors), usually written in the form

$$a_k y_n + \cdots + a_1 y_{n-k+1} + a_0 y_{n-k} = 0. \tag{A.11}$$

Substituting the assumption $y_n = \lambda^n$ gives the characteristic polynomial

$$a_n \lambda^n + \cdots + a_1 \lambda + a_0 = 0. \tag{A.12}$$

Assuming all roots are distinct, the solution is a linear combination of the simple solutions λ^n. If μ is a root with multiplicity k then the sequences

$$y_n^{\{l\}} = \frac{\mathrm{d}^{(l-1)}}{\mathrm{d}\lambda^{(l-1)}} \left[\lambda^n\right], \quad l = 0, \ldots, k \tag{A.13}$$

are all solutions.

For our purposes the key is that for the solution of a difference equation to be *bounded* (i.e., y_n is bounded as $n \to \infty$) we need all roots λ to satisfy $|\lambda| \leq 1$ (and if the root is repeated we need $|\lambda| < 1$).