Manchester
Metropolitan
University

6G6Z3002 Computational Methods for Ordinary Differential Equations

Part B

Lecture Notes

Dr Zahir Hussain

2019–2020

School of Computing, Mathematics & Digital Technology

Contents

Unit information

The purpose of these lecture notes is to support the Runge-Kutta methods and computational linear algebra half of the unit 6G6Z3002 Computational Methods in Ordinary Differential Equations which will be taught by Zahir Hussain. These notes are incomplete and some examples have been left for the student to complete in the lectures. It is vital that students attend the lectures and tutorials in order to complete the these examples and to help fully understand the material.

The original notes have been constructed as part of a team effort involving Dr Nicolette Rattenbury, Dr Jon Shiach and Dr Zahir Hussain. Whilst every attempt has been made to ensure the accuracy of these notes, there may be the occasional typographical error. If you do find an error, give Zahir the details so that the appropriate changes can be made.

The teaching team

This unit will be taught by Dr Lida Nejad and Dr Zahir Hussain. Lida and Zahir will be taking the lectures and tutorials in alternate weeks. The contact details for the teaching team are given in Table 1, and can be found on the Moodle page for this unit.

Table 1: Teaching team contact details

Tutor	Office	Email	Telephone no.
Dr Lida Nejad (unit leader)	E116c	l.nejad@mmu.ac.uk	0161 247 3583
Dr Zahir Hussain	E115b	z.hussain@mmu.ac.uk	0161 247 3555

As the teaching team, we will do our best to support you in this unit and during your final year. If you have any problems regarding the unit please come and speak to us as soon as possible. Our office hours will be available on the front of our office doors and on Moodle. If there are any other issues that are effecting your studies, you are advised to see your personal tutor as soon as possible.

Delivery pattern

This unit is delivered by two one-hour lectures and a one hour tutorial each week. As you will have experienced in levels 4 and 5, the lectures are the main delivery method used to present information to a large group at one time. Students learn best when they are an active participant in lectures, therefore we encourage you to ask questions and will try to minimise the times when you are not active by going through worked examples with you.

For the tutorials you will split into groups and the tutorials will take place in one of the computer laboratories in the central block of the John Dalton Building. The tutorials are your chance to receive more one-to-one support and formative feedback.

Learning and teaching

Having got to this stage in your studies you should already be familiar with the learning and teaching practices employed at undergraduate level. However, since this is a level 6 (final year) unit there will be certain expectations of the students from the teaching team. The pace at which the content will be delivered, especially in the first term, may be quicker than you are used to. Our advice would be to try and stay on top of the material as much as possible and make sure you are completing the required number of hours of independent study outside of class time.

In order to successfully complete this unit students should:

- read through the sections of the lecture notes that will be covered in a lecture prior to attending the lecture–don't worry if you don't understand it, just by familiarising yourself with the material will help you;
- attempt the tutorial exercises for a tutorial before attending the tutorial–it may not be possible to cover all of the exercises in the tutorials so it is important that we concentrate on the areas that are causing you difficulties;
- attend all lectures and tutorials–of course there may be instances where this is not possible, if you do miss a session, make sure you minimise the disruption to your studies by going through the material that you missed.

In return the teaching team will:

- provide students with detailed lecture notes and other supporting material;
- answer student queries regarding the material (may not always be possible in the lecture);
- provide solutions/answers to worked examples and tutorial exercises;
- provide formative feedback based on the tutorial exercises and summative feedback based on the coursework assignment;
- provide exam style questions for use as revision materials.

Most students find the step up to level 6 challenging and there will be times during the lectures that you may not fully understand all of the concepts being taught. In these situations we would advise you to read over the material again, follow the examples given and most importantly ask questions in lectures and tutorials. If there are gaps in your knowledge from levels 4 and 5, you must make sure that you address these as there will not be much time to go over previous level material again.

Chapters 1 and 6 briefly covers some essential topics from levels 4 and 5 that will be required for the Runge-Kutta methods and computational linear algebra parts of this unit. Please read through these chapters before the material is covered in the lectures.

MATLAB

This unit makes extensive use of MATLAB to perform the computations. Some of you will had the pleasure of working with MATLAB more than others during level 5 and may be a bit rusty on some of the commands. We encourage you to make sure you have your level 4 (first year) MATLAB lecture notes to hand to act as a reference (an electronic copy is available *via* Moodle). Fortunately we will be providing numerous examples of MATLAB programs and of course the teaching team will be available during tutorials if you need some help.

Teaching schedule

The teaching schedule for the Runge-Kutta methods and computational linear algebra half of the unit is given in Table 2. We will try to follow to this schedule as closely as possible although students must be prepared to be flexible in case there is a reason to deviate from this.

Table 2: Runge-Kutta methods and computational linear algebra teaching schedule

Week	Lecture date	Material
2	30/09/19	**ODEs Preliminaries**: definition of ODEs, initial value problems, Taylor series, solving ODEs using single step methods, solving systems of ODEs.
4	14/10/19	**Explicit Runge-Kutta Methods**: Butcher tableau, derivation of 2nd, 3rd and 4th order Runge-Kutta methods.
6	28/10/19	**Implicit Runge-Kutta Methods**: stiffness, order conditions for implicit methods, derivation of Gauss-Legendre, Radau and DIRK methods. **Coursework handed out**
8	11/11/19	**Stability**: the model problem, absolute stability, A-stability, stability functions for explicit and implicit Runge-Kutta methods, plotting stability regions.
10	25/11/19	**Boundary Value Problems**: existence and uniqueness of solutions, shooting methods, finite difference methods.
12	09/12/19	**Consolidation and revision of term 1 material.**
Christmas Vacation (3 weeks)		
13	06/01/20	**Coursework feedback**
15	20/01/20	**LU/LUP Factorisation**: LU factorisation, solving linear systems using LU factorisation, LUP Factorisation, **Cholesky Factorisation**: positive definite matrices, Cholesky factorisation.
17	03/02/20	**Indirect Methods**: Gauss-Seidel methods, convergence behaviour of iterative methods, the SOR method.
19	17/02/20	**QR Factorisation**: Linear least squares problem, orthogonal matrices, QR factorisation using the Gram-Schmidt process, solving linear systems using QR factorisation.
21	02/03/20	**Exam revision**
23	16/03/20	**Exam revision**
Easter Vacation (3 weeks)		

Chapter 1

Ordinary Differential Equations Preliminaries

1.1 Ordinary Differential Equations

An *Ordinary Differential Equation* (ODE) is an equation that contains a function of one independent variable and the derivatives of this variable. For example, let t be the independent variable then we can write an ODE of the form

$$y^{(n)}(t) = f(t, y'(t), y''(t), \ldots, y^{(n-1)}(t)) \tag{1.1}$$

where $y'(t) = \dfrac{d}{dt}y(t)$, $y''(t) = \dfrac{d^2}{dt^2}y(t)$ and $y^{(n)}(t) = \dfrac{d^n}{dt^n}y(t)$ denotes the first, second and nth derivatives of $y(t)$ with respect to t. Note that it is common practice to express the function $y(t)$ as simply y, therefore the ODE in Eq. (1.1) can be written as

$$y^{(n)} = f(t, y', y'', \ldots, y^{(n-1)}).$$

Definition 1.1.1 (Order of an ODE). The *order* of an ODE is the highest order of the derivatives that define the ODE. For example, the ODE in Eq. (1.1) is an nth order ODE.

Definition 1.1.2 (Linear ODE). An ODE of order n is said to be *linear* if it can be written as a linear combination of the derivatives, i.e.,

$$y^{(n)} = a_0(t)y + a_1(t)y' + a_2(t)y'' + \ldots + a_{n-1}(t)y^{(n-1)}.$$

1.2 Initial Value Problems

An *Initial Value Problem* (IVP) is an ODE where the solution for an initial value of the independent variable is known, for example, consider the following first-order ODE

$$y' = f(t, y), \qquad y(a) = \alpha.$$

Here a is the initial value of the independent variable t and α is the solution to the ODE at this initial value. An IVP has a unique solution if the function $f(t, y)$ is continuous in the region containing a and α.

Example 1.2.1. Solve the following ODE

$$y' = y. \tag{1.2}$$

Since $y' = \dfrac{dy}{dt}$ then

$$\frac{dy}{dt} = y$$
$$\frac{1}{y}dy = dt.$$

Integrating both sides

$$\int \frac{1}{y}dy = \int 1dt$$
$$\ln|y| = t + c$$
$$\therefore y = e^{t+c}.$$

The solution is a function of t and the constant of integration c so we have a family of solutions that will depend on the value of c (Fig. 1.1).

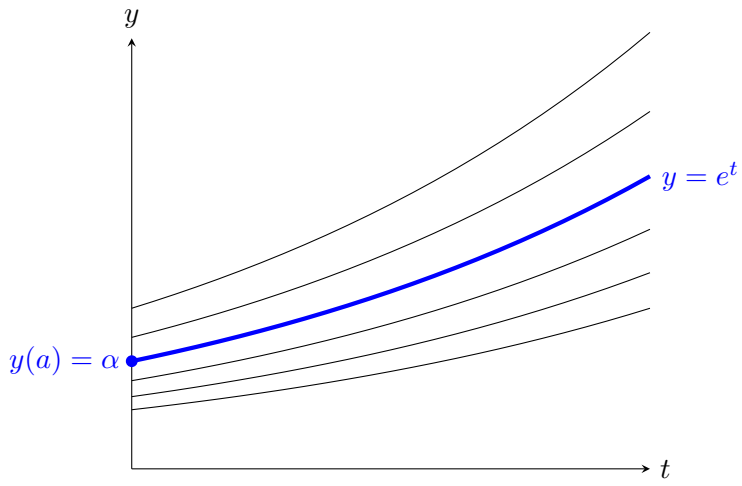


Figure 1.1: The family of solutions to the ODE $y'(t) = y$.

Let's say that the solution to the ODE in Eq. (1.2) when $t = 0$ is $y(0) = 1$. This allows us to calculate the value of c,

$$1 = e^{0+c}$$
$$\therefore c = 0,$$

so the solution to the IVP is $y = e^t$ which is a function of only one variable and a unique solution exists for a given value of t, i.e., we need initial values to find the unique solution to an ODE. □

1.3 The Taylor series

The Taylor series, named after English mathematician Brook Taylor (1685–1731), is a series expansion of a function $f(t)$ about a point a and is used as the basis of most numerical ODE solvers. The derivation of the Taylor series proceeds as follows: given an infinitely differentiable function $f(t)$, we want to express this as a polynomial in terms of an arbitrary point a such that

$$f(t) = \alpha_0 + \alpha_1(t-a) + \alpha_2(t-a)^2 + \alpha_3(t-a)^3 + \ldots + \alpha_n(t-a)^n + \ldots, \tag{1.3}$$

where the coefficients α_i $(i = 0, 1, \ldots, n)$ are to be determined. If we let $t = a$ then Eq. (1.3) becomes

$$\alpha_0 = f(a).$$

To find α_1 we differentiate Eq. (1.3)

$$f'(t) = \alpha_1 + 2\alpha_2(t-a) + 3\alpha_3(t-a)^2 + 4\alpha_4(t-a)^3 + \ldots, \tag{1.4}$$

and again letting $t = a$ we have

$$\alpha_1 = f'(a).$$

To find α_2 we differentiate Eq. (1.4) again to give

$$f''(t) = 2\alpha_2 + 3 \cdot 2\alpha_3(t-a) + 4 \cdot 3\alpha_4(t-a)^2 + \ldots,$$

and once more if $t = a$ we have

$$\alpha_2 = \frac{1}{2}f''(a).$$

If we continue in this way then the ith coefficient is

$$\alpha_i = \frac{1}{i!}f^{(i)}(a),$$

therefore substituting the coefficients α_n into Eq. (1.3) gives

$$f(t) = f(a) + (t-a)f'(a) + \frac{(t-a)^2}{2!}f''(a) + \ldots + \frac{(t-a)^n}{n!}f^{(n)}(a) + \ldots. \tag{1.5}$$

Equation (1.5) gives the value of the function $f(t)$ at an arbitrary point $t = a$. It is useful to know how the function behaves a small distance from this arbitrary point. Let $t = a + h$ so $h = t - a$ then

$$f(a+h) = f(a) + hf'(a) + \frac{h^2}{2!}f''(a) + \ldots + \frac{h^n}{n!}f^{(n)}(a) + \ldots = \sum_{n=0}^{\infty} \frac{h^n}{n!}f^{(n)}(a),$$

replacing a with t we can write

$$f(t+h) = f(t) + hf'(t) + \frac{h^2}{2!}f''(t) + \ldots + \frac{h^n}{n!}f^{(n)}(t) + \ldots = \sum_{n=0}^{\infty} \frac{h^n}{n!}f^{(n)}(t). \tag{1.6}$$

Equation (1.6), herein referred to in these notes as *the Taylor series*, is used extensively in the derivation of numerical methods used to approximate ODEs and finite-difference methods used to approximate derivatives.

1.3.1 Truncating the Taylor series

Equation (1.6) gives an exact value of the function $f(t+h)$ when summing an infinite number of terms, however, this is not possible to calculate in practice. Instead, we approximate the function by summing only the first few terms of the Taylor series and in doing so we say that we are *truncating* the Taylor series. Note that the nth derivative of $f(t+h)$ is multiplied by $h^n/n!$, so as n gets larger, the nth term has a minimal effect on the approximation of $f(t+h)$.

To account for the terms that are being ignored, when we truncate the Taylor series we use 'big oh'* notation to denote the error due to the higher order terms not being present in the

*Not to be confused with similar notation used to denote computational complexity of an algorithm which we will encounter in the linear algebra part of this unit.

approximation. For example, if we truncate the Taylor series so we only use the sum of the first n terms, then we append $O(h^{n+1})$ to the truncated series to denote the *truncation error* and we refer to the series as the *nth-order Taylor series expansion* (for most practical applications, and ones that we will use in this unit, expansion up to fourth-order will suffice), i.e.,

$$\begin{aligned}
\text{1st order :} \quad & f(t+h) = f(t) + hf'(t) + O(h^2), \\
\text{2nd order :} \quad & f(t+h) = f(t) + hf'(t) + \frac{h}{2}f''(t) + O(h^3), \\
\text{3rd order :} \quad & f(t+h) = f(t) + hf'(t) + \frac{h^2}{2}f''(t) + \frac{h^3}{6}f^{(3)}(t) + O(h^4), \\
\text{4th order :} \quad & f(t+h) = f(t) + hf'(t) + \frac{h^2}{2}f''(t) + \frac{h^3}{6}f^{(3)}(t) + \frac{h^4}{24}f^{(4)}(t) + O(h^5).
\end{aligned}$$

The $O(h^n)$ terms means that the rate of which the error tends to zero as $h \to 0$ is proportional to a polynomial function in terms of h of degree n. Note that in general, the higher the order of the Taylor series, the more accurate the approximation. Since the approximation depends on the size of the step length h, a higher-order method will allow a larger value of h to be used to gain comparable accuracy to a lower-order method thus saving computational effort.

Example 1.3.1. Consider the behaviour of the errors when using the Taylor series to approximate e^{t+h} for decreasing values of h. Since $\frac{d}{dt}e^t = e^t$ then the Taylor series expansions for this function to fourth-order is

$$e^{t+h} \approx e^t + he^t + \frac{h^2}{2}e^t + \frac{h^3}{6}e^t + \ldots + \frac{h^n}{n!}e^t + \ldots.$$

Let's consider the approximation case when $t = 0$ and $h = 1$ (when using $t = 0$, the Taylor series is known as the *Maclaurin series*) then

$$\begin{aligned}
\text{1st order:} \quad & e^1 \approx e^0 + e^0 = 1 + 1 = 2, \\
\text{2nd order:} \quad & e^1 \approx e^0 + e^0 + \frac{e^0}{2} = 1 + 1 + \frac{1}{2} = 2.5, \\
\text{3rd order:} \quad & e^1 \approx e^0 + e^0 + \frac{e^0}{2} + \frac{e^0}{6} = 1 + 1 + \frac{1}{2} + \frac{1}{6} = 2.6667, \\
\text{4th order:} \quad & e^1 \approx e^0 + e^0 + \frac{e^0}{2} + \frac{e^0}{6} + \frac{e^0}{24} = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} = 2.7083,
\end{aligned}$$

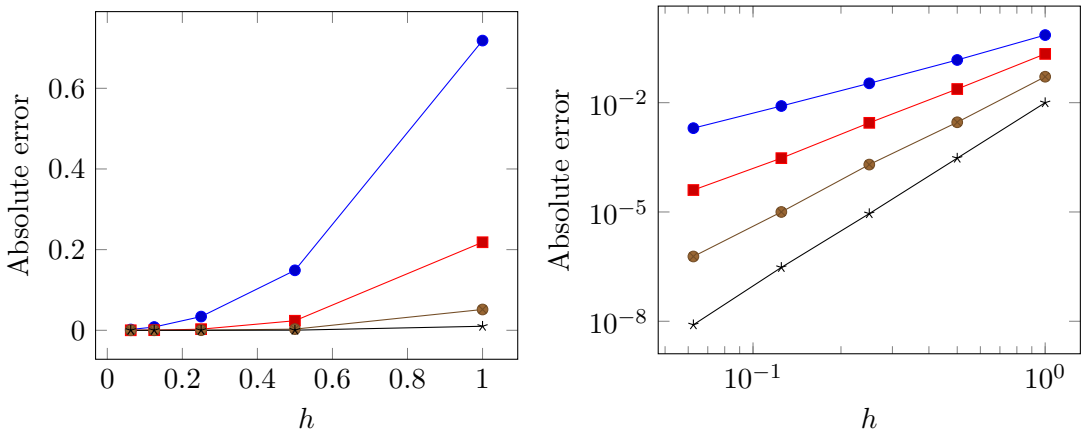
We can see that as the order of the Taylor series increases, the approximation gets closer the actual value of $e^1 = 2.7183$ (correct to four decimal places). The absolute error for the first-order approximation is $|2.7183 - 2| = 0.7183$ whereas the error for the fourth-order approximation is $|2.7183 - 2.7083| = 0.01$ which shows that as the order of the approximation increases, the absolute error decreases. The errors between the Taylor series approximations of e^h using step lengths $h = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$ and $\frac{1}{16}$ for expansions of order 1 through to 4 are shown in Table 1.1.

Table 1.1: Truncation errors for the Taylor series approximations of e^h.

h	$O(h^2)$	$O(h^3)$	$O(h^4)$	$O(h^5)$
1.0000	0.7183	0.2183	0.0516	0.0100
0.5000	0.1487	0.0237	0.0029	0.0003
0.2500	0.0340	0.0028	0.0002	$8.5e{-6}$
0.1250	0.0081	0.0003	1.0e−5	2.6e−7
0.0625	0.0020	4.1e−5	6.4e−7	8.0e−9

The behaviour of the truncation errors can be better seen when plotted against the step length h as shown in Fig. 1.2. Its is clear that the higher order approximations have a much smaller error than the lower order approximations. What isn't so clear from Fig. 1.2(a) is how the behaviour of the errors changes as $h \to 0$. To make a better comparison of the convergence of the truncation errors we can use a logarithmic scale as shown in Fig. 1.2(b). This causes the points to lie on a straight line for each order expansion and the gradient of the line is an estimate of n in $O(h^n)$. For example, using the values from Table 1.1

$$
\begin{aligned}
O(h^2): \qquad & n \approx \frac{\log(0.7183) - \log(0.0020)}{\log(1) - \log(0.0625)} = 2.1221, \\
O(h^3): \qquad & n \approx \frac{\log(0.2183) - \log(4.1 \times 10^{-5})}{\log(1) - \log(0.0625)} = 3.0946, \\
O(h^4): \qquad & n \approx \frac{\log(0.0516) - \log(6.4 \times 10^{-7})}{\log(1) - \log(0.0625)} = 4.0747, \\
O(h^5): \qquad & n \approx \frac{\log(0.01) - \log(8 \times 10^{-9})}{\log(1) - \log(0.0625)} = 5.0634.
\end{aligned}
$$



(a) Standard scale **(b)** Logarithmic scale

Figure 1.2: Truncation errors for the Taylor series approximations of e^h.

□

1.3.2 Bivariate Taylor series

The Taylor series shown in Eq. (1.6) can be extended to represent a *bivariate* function (function of two variables) by using the total derivative of a bivariate function.

Definition 1.3.1 (Total derivative). Let $f(t, y_1, y_2, \ldots)$ be a multivariable function. The derivative of f with respect to one of its variables, t say, is determined by assuming that the other variables in the function depend on t. Therefore, using the chain rule we have

$$\frac{df}{dt} = \frac{\partial f}{\partial t}\frac{dt}{dt} + \frac{\partial f}{\partial y_1}\frac{dy_1}{dt} + \frac{\partial f}{\partial y_2}\frac{dy_2}{dt} + \ldots \tag{1.7}$$

Multiplying both sides by the differential dt gives

$$
\begin{aligned}
df &= \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial y_1}dy_1 + \frac{\partial f}{\partial y_2}dy_2 + \ldots \\
&= \left(dt\frac{\partial}{\partial t} + dy_1\frac{\partial}{\partial y_1} + dy_2\frac{\partial}{\partial y_2} + \ldots \right) f(t, y_1, y_2, \ldots)
\end{aligned} \tag{1.8}
$$

which is the *total derivative* of the function $f(t, y_1, y_2, \ldots)$.

The derivation of the bivariate Taylor series requires the evaluation of the nth derivative of the bivariate function $f(t, y)$. Applying Eq. (1.8) gives

$$f'(t,y) = \left(dt\frac{\partial}{\partial t} + dy\frac{\partial}{\partial y} \right) f(t,y),$$

and applying Eq. (1.8) to $f'(t, y)$ gives the second derivative

$$\begin{aligned} f''(t,y) &= \left(dt\frac{\partial}{\partial t} + dy\frac{\partial}{\partial y} \right) f'(t,y) \\ &= \left(dt\frac{\partial}{\partial t} + dy\frac{\partial}{\partial y} \right) \left(dt\frac{\partial}{\partial t} + dy\frac{\partial}{\partial y} \right) f(t,y) \\ &= \left(dt\frac{\partial}{\partial t} + dy\frac{\partial}{\partial y} \right)^2 f(t,y). \end{aligned}$$

It can be clearly seen that

$$f^{(n)}(t,y) = \left(dt\frac{\partial}{\partial t} + dy\frac{\partial}{\partial y} \right)^n f(t,y).$$

Substituting the derivatives of the bivariate function $f(t, y)$ into the Taylor series, Eq. (1.6), and letting h and k be the change in the t and y variables respectively gives

$$\begin{aligned} f(t+h, y+k) &= f(y,t) + \left(h\frac{\partial}{\partial t} + k\frac{\partial}{\partial y} \right) f(t,y) + \frac{1}{2!}\left(h\frac{\partial}{\partial t} + k\frac{\partial}{\partial y} \right)^2 f(t,y) + \dots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!}\left(h\frac{\partial}{\partial t} + k\frac{\partial}{\partial y} \right)^n f(t,y), \end{aligned} \tag{1.9}$$

which is the *bivariate Taylor series*. For example, the first and second-order bivariate Taylor series expansions are:

$$\begin{aligned} f(t+h, y+k) &= f(t,y) + hf_t(t,y) + kf_y(t,y) + O(h^2, k^2), \\ f(t+h, y+k) &= f(t,y) + hf_t(t,y) + kf_y(t,y) + \frac{h^2}{2}f_{tt}(t,y) + hkf_{ty}(t,y) \\ &\quad + \frac{k^2}{2}f_{yy}(t,y) + O(h^3, k^3). \end{aligned}$$

1.4 Single step methods for solving IVPs

There are two main types of computational methods for solving IVPs, *single step methods* and *multistep* methods. Both types of methods calculate the solutions at discrete steps in terms of the independent variable. The change in the independent variable from one step to the next is called the *step length*. Single step methods are so called because only information from the current step is required to calculate the solution at the next step whereas multistep methods require information from the current step and multiple previous steps. These notes will focus on single step methods.

1.4.1 The Euler method

The simplest single step method is the *Euler method* named after Swiss mathematician Leonard Euler (1707–1783). Consider the general first-order IVP

$$y' = f(t,y), \qquad y(a) = \alpha, \qquad a \le t \le b.$$

The values of the function $y(t)$ that satisfies the IVP is calculated at various values of the independent variable t in the domain $a \leq t \leq b$. The values of t are determined at equally spaced points in the domain. If the solution is calculated at $N+1$ equally spaced points in the domain and h is the distance between two successive points then the values of t can be calculated using

$$t_i = a + ih, \qquad i = 0, 1, \ldots, N,$$

Note that it is common practice to denote the first element in an array with the subscript 0 and the last element with subscript N. Hence the number of elements in the array is $N+1$. This is useful for calculating the differences between two successive values but care must be taken if using a programming language which uses 1-indexing (where arrays are assumed to start at 1), e.g., MATLAB and FORTRAN.

Since we want $t_0 = a$ and $t_N = a + Nh = b$ then N is related to h by

$$N = \frac{b-a}{h}.$$

The Euler method is derived by truncating the Taylor series so that all terms involving second-order or higher are ignored, i.e.,

$$y(t+h) = y(t) + hy'(t) + O(h^2).$$

Since $y' = f(t, y)$ then we have

$$y(t+h) = y(t) + hf(t, y) + O(h^2).$$

Utilising subscript notation where n denotes the values at the current known step and $n+1$ denotes the values at the next step we have

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{1.10}$$

which is known as the *Euler method*. The solution process begins with calculating the values of t for the step length h and assigning the initial condition $y_0 = \alpha$. The solution at the next step given by Eq. (1.10) is repeated N times until the solution to the IVP has been calculated for all values of t (see Algorithm 1)

Algorithm 1 The Euler method

Require: An IVP of the form $y' = f(t, y)$, $a \leq t \leq b$, $y(a) = \alpha$ and a step length h.

$N \leftarrow \dfrac{b-a}{h}$

$y_0 \leftarrow \alpha$ and $t_0 \leftarrow a$

for $n = 1, \ldots, N$ **do**

$y_{n+1} \leftarrow y_n + hf(t_n, y_n)$

$t_{n+1} \leftarrow t_n + h$

end for

return $\mathbf{t} = (t_0, \ldots, t_N)$ and $\mathbf{y} = (y_0, \ldots, y_N)$

Example 1.4.1. Solve the following IVP using the Euler method with a step length $h = 0.2$.

$$y' = y + t, \qquad 0 \leq t \leq 1, \qquad y(0) = 1.$$

The independent variable is in the domain $[0, 1]$ so using a step length of $h = 0.2$ we can calculate the number of steps required

$$N = \frac{b-a}{h} = \frac{1-0}{0.2} = 5.$$

Next we initialise $t_0 = 0$, $y_0 = 1$ and perform $N = 5$ steps of the Euler method Eq. (1.10). In this case $f(t, y) = y + t$.

$$
\begin{aligned}
n = 0, \qquad & y_1 = 1 + 0.2f(0, 1) = 1 + 0.2(0 + 1) = 1.2, \\
& t_1 = t_0 + h = 0 + 0.2 = 0.2, \\
n = 1, \qquad & y_2 = 1.2 + 0.2f(0.2, 1.2) = 1.2 + 0.2(0.2 + 1.2) = 1.48, \\
& t_2 = t_1 + h = 0.2 + 0.2 = 0.4, \\
n = 2, \qquad & y_3 = 1.48 + 0.2f(0.4, 1.48) = 1.48 + 0.2(0.4 + 1.48) = 1.856, \\
& t_3 = t_2 + h = 0.4 + 0.2 = 0.6, \\
n = 3, \qquad & y_4 = 1.856 + 0.2f(0.6, 1.856) = 1.856 + 0.2(0.6 + 1.856) = 2.3472, \\
& t_4 = t_3 + h = 0.6 + 0.2 = 0.8, \\
n = 4, \qquad & y_5 = 2.3472 + 0.2f(0.8, 2.3472) = 2.3472 + 0.2(0.8 + 2.3472) = 2.97664, \\
& t_5 = t_4 + h = 0.8 + 0.2 = 1.
\end{aligned}
$$

The analytical solution to this ODE is $y(t) = 2e^t - t - 1$. A computed solutions using the Euler method have been compared to the analytical solutions in Table 1.2 and Fig. 1.3. There are clear differences between the computed and analytical solutions and the absolute error between the two gets larger as more steps are calculated. This is because the Euler method is only first-order accurate since the terms involving second-order derivatives and higher in the Taylor series were ignored. The accuracy of the computed solutions can be improved by decreasing the value of the step length h, this will of course increase the number of steps N and therefore increase the computational effort required to solve the IVP. □

Table 1.2: Solutions for the IVP $y' = y + t$, $0 \leq t \leq 1$, $y(0) = 1$ calculated using the Euler method.

t	y_{euler}	y_{exact}	$\lvert y_{exact} - y_{euler} \rvert$
0.00	1.000000	1.000000	0.000000
0.20	1.200000	1.242806	0.042806
0.40	1.480000	1.583649	0.103649
0.60	1.856000	2.044238	0.188238
0.80	2.347200	2.651082	0.303882
1.00	2.976640	3.436564	0.459924

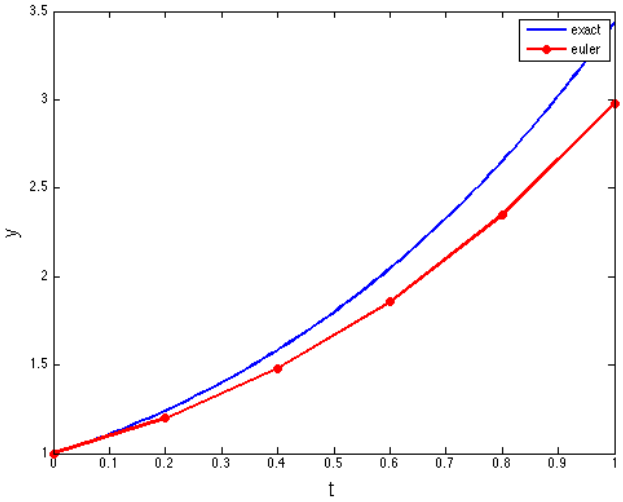


Figure 1.3: Comparison between the solutions computed using the Euler method and the exact solutions for the IVP $y' = y + t$, $0 \leq t \leq 1$, $y(0) = 1$.

1.5 Using MATLAB to perform calculations

Computing the solution to an IVP such as in Example 1.4.1 can be a tedious exercise even for a fairly large step length of $h = 0.2$. In practice we write computer programs to perform the calculations for us. The MATLAB function shown in 1.1 computes the solution to the general IVP

$$y' = f(t, y), \qquad a \leq t \leq b, \qquad y(a) = \alpha,$$

using a step length h.

Listing 1.1: MATLAB function for the Euler method.

```
function [t,y] = euler(f, a, b, y0, h)

% euler.m by Jon Shiach
%
% This function solves the IVP f' = f(t,y), a <= t <= b, y(0) = y0 using
% the Euler method with step length h

% calculate number of steps
nsteps = (b - a)/h;

% calculate t array and initialise y
t = a : h : b;
y(:,1) = y0;

% loop through steps and calculate solution
for n = 1 : nsteps
    y(:,n+1) = y(:,n) + h*f(t(n),y(:,n));
end
```

To invoke the function shown in 1.1 another program is written that sets up the IVP to be solved 1.2.

Listing 1.2: MATLAB program to define an IVP and invoke the Euler method to solve it.

```
% euler_ex1.m by Jon Shiach
%
% This program solves the IVP y' = t + y, 0 <= t <= 1, y(0) = ya using the
% Euler method
```

```
% define solution variables
a = 0;          % lower bound for t
b = 1;          % upper bound for t
ya = 1;         % initial value of y
h = 0.2;        % step length

% define ODE
f = @(t,y) y + t;

% call Euler function to solve IVP
[t,y] = euler(f, a, b, ya, h);

% calculate the absolute error between the exact and computed solutions
y_ex = 2*exp(t) - t - 1;
err = abs(y_ex - y);

% output solution
fprintf('\n---------------------------------------\n')
fprintf('  t    |   RK2    |   Exact  |   Error  \n')
fprintf('---------------------------------------\n')
for i = 1 : length(t)
    fprintf('%5.2f | %8.6f | %8.6f | %8.6f\n',t(i),y(i),y_ex(i),err(i))
end
fprintf('---------------------------------------\n\n')

% plot solution
plot(t, y_ex, 'linewidth', 2)
hold on
plot(t, y, 'ro-','linewidth', 2, 'markerfacecolor', 'r')
hold off
xlabel('t', 'fontsize', 16)
ylabel('y', 'fontsize', 16)
legend('exact', 'euler')
```

1.6 Writing higher order ODEs as a system of first order ODEs

Higher order ODEs can be transformed into a system of first-order ODEs. This is useful since it allows us to use our first-order ODE solvers (e.g., Euler, Runge-Kutta, Adams methods etc.) to solve higher order ODEs. For example, consider the nth order ODE

$$y^{(n)} = f(t, y', y'', \ldots, y^{(n-1)}).$$

If we define n new functions $y_1, y_2, \ldots, y_n$ such that

$$\begin{aligned} y_1 &= y, \\ y_2 &= y', \\ y_3 &= y'', \\ &\vdots \\ y_n &= y^{(n-1)}, \end{aligned}$$

then we can write the original ODE as a system of n first-order ODEs

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= y_3, \\ y_3' &= y_4, \\ &\vdots \\ y_n' &= f(t, y_1, y_2, \ldots, y_{n-1}). \end{aligned}$$

Each of these first-order ODEs can be solved at the same time using an ODE solver. For convenience, we write the above system in vector notation such that

$$\mathbf{y}' = F(t, \mathbf{y}), \tag{1.11}$$

where $\mathbf{y}$ and $F(t, \mathbf{y})$ are the vectors

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}, \qquad F(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ y_3 \\ y_4 \\ \vdots \\ f(t, y_1, y_2, \ldots, y_{n-1}) \end{pmatrix}.$$

The system of ODEs in Eq. (1.11) can be solved using ODE solvers. For example, the Euler method becomes

$$\mathbf{y}_{n+1} = \mathbf{y}_n + hF(t_n, \mathbf{y}_n).$$

Example 1.6.1. The height, y, of an object acting under gravity can be modelled by the ODE

$$y'' = -g,$$

where $g = 9.81\text{ms}^{-2}$ is the acceleration due to gravity. An object on the ground is launched into the air with a vertical velocity of 20ms^{-1}, calculate the height of the object over the domain $t \in [0, 5]$ using step length $h = 1$.

The velocity is the change in the vertical height y with respect to time, i.e., y', and the acceleration is the change in velocity with respect to time, i.e., y''. Therefore, this problem can be formulated as the IVP

$$y'' = -g, \qquad y(0) = 0, \qquad y'(0) = 20.$$

Let $y_1 = y$ and $y_2 = y_1'$ then this second-order ODE can be written as the following system of first-order ODEs

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= -g, \end{aligned}$$

or in vector form

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \qquad F(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ -g \end{pmatrix},$$

such that $\mathbf{y}' = F(t, \mathbf{y})$. For this problem, $y_1 = 0$ and $y_2 = 20$ so the initial values of the vectors $\mathbf{y}$ and $F(t, \mathbf{y})$ are

$$\mathbf{y}_0 = \begin{pmatrix} 0 \\ 20 \end{pmatrix}, \qquad F(t_0, \mathbf{y}_0) = \begin{pmatrix} 20 \\ -g \end{pmatrix}.$$

Using a step length $h = 1$ we have $N = 5$. Using the Euler method to solve the IVP gives

$$
\begin{aligned}
n = 0, \qquad & \mathbf{y}_1 = \mathbf{y}_0 + hF(t_0, \mathbf{y}_0) = \begin{pmatrix} 0 \\ 20 \end{pmatrix} + \begin{pmatrix} 20 \\ -9.81 \end{pmatrix} = \begin{pmatrix} 20 \\ 10.19 \end{pmatrix}, \\
& t_1 = t_0 + h = 0 + 1 = 1, \\
n = 1, \qquad & \mathbf{y}_2 = \mathbf{y}_1 + hF(t_1, \mathbf{y}_1) = \begin{pmatrix} 20 \\ 10.19 \end{pmatrix} + \begin{pmatrix} 10.19 \\ -9.81 \end{pmatrix} = \begin{pmatrix} 30.19 \\ 0.38 \end{pmatrix}, \\
& t_2 = t_1 + h = 1 + 1 = 2, \\
n = 2, \qquad & \mathbf{y}_3 = \mathbf{y}_2 + hF(t_2, \mathbf{y}_2) = \begin{pmatrix} 30.19 \\ 0.38 \end{pmatrix} + \begin{pmatrix} 0.38 \\ -9.81 \end{pmatrix} = \begin{pmatrix} 30.57 \\ -9.43 \end{pmatrix}, \\
& t_3 = t_2 + h = 2 + 1 = 3, \\
n = 3, \qquad & \mathbf{y}_4 = \mathbf{y}_3 + hF(t_3, \mathbf{y}_3) = \begin{pmatrix} 30.57 \\ -9.43 \end{pmatrix} + \begin{pmatrix} -9.43 \\ -9.81 \end{pmatrix} = \begin{pmatrix} 21.14 \\ -19.24 \end{pmatrix}, \\
& t_4 = t_3 + h = 3 + 1 = 4, \\
n = 4, \qquad & \mathbf{y}_5 = \mathbf{y}_4 + hF(t_4, \mathbf{y}_4) = \begin{pmatrix} 21.14 \\ -19.24 \end{pmatrix} + \begin{pmatrix} -19.24 \\ -9.81 \end{pmatrix} = \begin{pmatrix} 1.9 \\ -29.05 \end{pmatrix}, \\
& t_5 = t_4 + h = 4 + 1 = 5.
\end{aligned}
$$

The analytical solution to this IVP can be found by integrating the ODE to give $y(t) = 20t - \frac{1}{2}gt^2$. A comparison between the analytical solution and the solutions computed using the Euler method can be seen in Fig. 1.4. Here there is a clear difference between the two solutions and the Euler method is clearly unsuitable for solving this problem. □

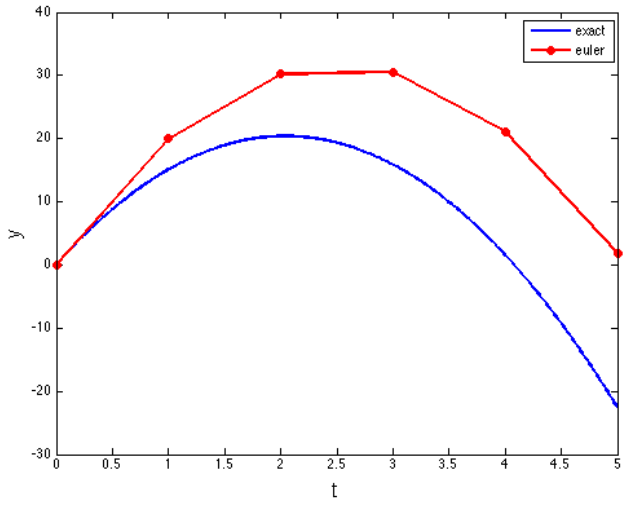


Figure 1.4: Comparison between the solutions computed using the Euler method and the exact solutions for the second-order IVP $y'' = -g$, $0 \leq t \leq 5$, $y(0) = 0$, $y'(0) = 20$.

1.7 ODEs Preliminaries Tutorial Exercises

For these questions you should make use of MATLAB or Excel to help you with the calculations.

1. Consider the following IVP

$$y' = \frac{t}{y}, \qquad 0 \leq t \leq 1, \qquad y(0) = 1.$$

   (a) Solve the IVP using the Euler method with a step length of $h = 0.1$

   (b) Solve the IVP using the second-order Runge-Kutta method using the same step length.

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{2}(k_1 + k_2), \\ k_1 &= hf(t_n, y_n), \\ k_2 &= hf(t_n + h, y_n + k_1). \end{aligned}$$

2. Consider the second-order IVP

$$y'' = y' + ty, \qquad 1 \leq t \leq 3, \qquad y(1) = 1, \qquad y'(1) = 2.$$

   Express this ODE as a system of two first-order ODEs and solve using a step length $h = 0.2$ with

   (a) the Euler method;

   (b) the second-order Runge-Kutta method.

The solutions to these tutorial exercises are given on page 157

Chapter 2

Explicit Runge-Kutta methods

The explicit Runge-Kutta methods (more commonly known simply as the Runge-Kutta methods) named after German mathematicians Carl Runge (1856–1927) and Martin Kutta (1867–1944), are a family of ODE solvers that are derived from the Taylor series expansion. Runge-Kutta methods are an example of a *single step method* because the calculation of value of the solution at the next step can be done in a single step using the current known value of the solution (although multiple intermediate steps are used but the values of which are not needed for the next step). Multistep methods where multiple steps are required to calculate the next value of the solutions (e.g., Adams methods) will be covered by Dr Nejad. This chapter will introduce the second, third and fourth-order explicit Runge-Kutta methods.

2.1 General form of an explicit Runge-Kutta method

The solution to the first-order ODE

$$y' = f(t, y),$$

can be approximated using an iterative scheme of the form

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i = y_n + b_1 k_1 + b_2 k_2 + \ldots + b_n k_n, \tag{2.1}$$

where

$$\begin{aligned}
k_1 &= hf(t_n, y_n),\\
k_2 &= hf(t_n + c_2 h, y_n + a_{21} k_1),\\
k_3 &= hf(t_n + c_3 h, y_n + a_{31} k_1 + a_{32} k_2),\\
&\vdots\\
k_s &= hf(t_n + c_s h, y_n + a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1}).
\end{aligned}$$

This is the general form of an explicit Runge-Kutta method. The values k_i are known as *stages* of the Runge-Kutta method which depend on coefficient values a_{ij}, b_i and c_i where $i, j = 1, \ldots, s$ and s is the number of stages. The values of these coefficients are derived using the Taylor series. Note that the values of the ith stage, k_i, is explicitly stated in terms of the previous stages, k_1, k_2, …, k_{i-1}, so a method of this type is called an *explicit Runge-Kutta method*.

2.2 Butcher tableau

The coefficients a_{ij}, b_i and c_i are commonly expressed in a table called a *Butcher tableau* after John Butcher (1933–present). Representing a Runge-Kutta method in this way is more

convenient for deriving the methods and performing stability analysis (stability is discussed in Chapter 4). The Butcher tableau takes the following form

$$\begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b} \end{array}$$

where A is the matrix of a_{ij} values, $\mathbf{c} = (c_1, c_2, \ldots, c_s)^T$ and $\mathbf{b} = (b_1, b_2, \ldots, b_s)$, i.e.,

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

The coefficients c_i and a_{ij} are connected by the condition

$$c_i = \sum_{j=1}^{s} a_{ij}, \qquad i = 1, \ldots, s, \tag{2.2}$$

i.e., the coefficient c_i is equal to the sum of the elements on the ith row of the matrix $[A]_{ij} = a_{ij}$. This is known as the *row sum condition*.

Example 2.2.1. Write the following explicit Runge-Kutta method as a Butcher tableau.

$$\begin{aligned} k_1 &= hf(t_n, y_n), \\ k_2 &= hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right), \\ k_3 &= hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right), \\ k_4 &= hf\left(t_n + h, y_n + k_3\right), \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned}$$

This is a four-stage explicit Runge-Kutta method. The values of $\mathbf{b}$ are the coefficients of k_i therefore $\mathbf{b} = \left(\frac{1}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{6}\right)$. The values of $\mathbf{c}$ are the coefficients of h in the function $f(t, y)$ therefore $\mathbf{c} = \left(0, \frac{1}{2}, \frac{1}{2}, 1\right)$. The values of A are the coefficients of k_i in the function $f(t, y)$, therefore we have the matrix

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

so the Butcher tableau for this method is

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Note that for an explicit Runge-Kutta method, the A matrix will always be a lower triangular matrix (i.e., $[A]_{ij} = 0$ when $i \leq j$) so it is standard practice to ignore the zeros on the main diagonal and upper triangular region. Also, due to the row sum condition Eq. (2.2), c_1 is always equal to zero for an explicit Runge-Kutta method. □

2.3 Second-order Runge-Kutta methods

To demonstrate the derivation of explicit Runge-Kutta methods we will consider the derivation of the second-order Runge-Kutta method (or RK2 for short). I have skipped over the derivation of first-order Runge-Kutta method as it is simply the same as the *Euler method* discussed in Section 1.4.1 (page 6).

Consider the general first-order ODE system

$$y' = f(t, y). \tag{2.3}$$

Since we want to construct a second-order method, we start with the Taylor series expansion truncated to second-order, i.e.,

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2}y''(t) + O(h^3). \tag{2.4}$$

We can replace the first derivative with $f(t, y)$ and the second derivative is obtained by differentiating Eq. (2.3) using the chain rule on $f(t, y)$, i.e.,

$$\begin{aligned} y''(t) &= \frac{\partial f}{\partial t}\frac{dt}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} \\ &= f_t(t, y) + f_y(t, y)y'(t) \\ &= f_t(t, y) + f_y(t, y)f(t, y) \qquad (\text{since } y' = f(t, y)), \end{aligned}$$

where $f_t(t, y)$ and $f_y(t, y)$ denote the partial derivates of $f(t, y)$ with respect to t and y respectively. Substituting this expression into the second-order Taylor expansion, Eq. (2.4), we have

$$\begin{aligned} y(t+h) &= y(t) + hf(t, y) + \frac{h^2}{2}\left[f_t(t, y) + f_y(t, y)f(t, y)\right] + O(h^3). \\ &= y(t) + \frac{h}{2}f(t, y) + \frac{h}{2}\left[f(t, y) + hf_t(t, y) + hf_y(t, y)f(t, y)\right] + O(h^3). \end{aligned} \tag{2.5}$$

Using Eq. (1.9), the bivariate Taylor series expansion truncated to first-order is

$$f(t+h, y+k) = f(t, y) + hf_t(t, y) + kf_y(t, y) + O(h^2), \tag{2.6}$$

then the expression in the square brackets of Eq. (2.5) can be written as

$$f(t+h, y+hf(t, y)) = f(t, y) + hf_t(t, y) + hf(t, y)f_y(t, y) + O(h^2).$$

Therefore Eq. (2.5) can be written as

$$y(t+h) = y(t) + \frac{h}{2}f(t, y) + \frac{h}{2}f(t+h, y+hf(t, y)) + O(h^3),$$

or alternatively, as the iterative scheme

$$k_1 = hf(t_n, y_n), \tag{2.7a}$$

$$k_2 = hf(t_n + h, y_n + k_1), \tag{2.7b}$$

$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2) \tag{2.7c}$$

Equations (2.7a) to (2.7c) are known collectively as the *classical second-order Runge-Kutta method* (RK2) or *Heun's method*. Since $a_{21} = 1$, $b_1 = b_2 = \frac{1}{2}$ and $c_2 = 1$ the Butcher tableau for the classical second-order Runge-Kutta method is

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Note that the row sum condition is satisfied, i.e., $c_2 = a_{21}$.

2.3.1 General form of the second-order Runge-Kutta methods

There are a variety of different Runge-Kutta methods that satisfy the conditions for second-order accuracy. Using Eq. (2.1), the general form of a second-order Runge-Kutta method is

$$y_{n+1} = y_n + b_1 k_1 + b_2 k_2 + O(h^3), \tag{2.8}$$

where

$$\begin{aligned} k_1 &= hf(t, y), \\ k_2 &= hf(t + c_2 h, y + a_{21} k_1). \end{aligned}$$

To obtain other second-order Runge-Kutta methods we need to find combinations of b_1, b_2, c_2 and a_{21} which match the second-order Taylor series expansion in Eq. (2.5). To do this we use the bivariate Taylor series expansion, Eq. (1.9)

$$\begin{aligned} f(t + c_2 h, y + a_{21} k_1) &= f(t, y) + c_2 h f_t(t, y) + a_{21} f_y(t, y) k_1 + O(h^2) \\ &= f(t, y) + c_2 h f_t(t, y) + h a_{21} f_y(t, y) f(t, y) + O(h^2). \end{aligned}$$

Therefore the second-order Runge-Kutta method, Eq. (2.8), can be written as

$$\begin{aligned} y(t + h) &= y(t) + b_1 h f(t, y) + b_2 h f(t + c_2 h, y + a_{21} k_1) + O(h^3) \\ &= y(t) + b_1 h f(t, y) + b_2 h [f(t, y) + c_2 h f_t(t, y) + h a_{21} f_y(t, y) f(t, y)] + O(h^3) \\ &= y(t) + (b_1 + b_2) h f(t, y) + b_2 h^2 [c_2 f_t(t, y) + a_{21} f_y(t, y) f(t, y)] + O(h^3). \end{aligned}$$

We need this expression to match that the second-order Taylor series in Eq. (2.5), therefore

$$b_1 + b_2 = 1, \tag{2.9a}$$

$$c_2 b_2 = \frac{1}{2}, \tag{2.9b}$$

$$a_{21} b_2 = \frac{1}{2}. \tag{2.9c}$$

Thus we have a system of three nonlinear equations in four unknowns. Equations (2.9a) to (2.9c) are known as the *order conditions for a second-order explicit Runge-Kutta method*. To derive a second-order explicit Runge-Kutta method we simply need to determine a values of b_1, b_2, c_2 and a_{21} that satisfy these order conditions. Since this is an overdetermined system (there are more unknowns than equations) we need to fix a value for one of the variables and solve to find the rest.

Example 2.3.1. Derive a second-order explicit Runge-Kutta method where $c_2 = \frac{1}{2}$.

Substituting c_2 into Eqs. (2.9a) to (2.9c) gives

$$\begin{aligned} b_1 + b_2 &= 1, \\ \frac{1}{2} b_2 &= \frac{1}{2}, \\ a_{21} b_2 &= \frac{1}{2}. \end{aligned}$$

This is easily solved to give $b_2 = 1$, $b_1 = 0$ and $a_{21} = \frac{1}{2}$ so the method is

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

This scheme is also known as the *midpoint rule*. □

2.3.2 Solving an IVP using a Runge-Kutta method

Consider a first-order IVP of the form

$$y' = f(t, y), \qquad a \leq t \leq b, \qquad y(a) = \alpha.$$

To this to solve this IVP using a second-order explicit Runge-Kutta method with the Butcher tableau

$$\begin{array}{c|cc} 0 & 0 & \\ c_2 & a_{21} & \\ \hline & b_1 & b_2 \end{array}$$

we follow the steps outlined in Algorithm 2.

Algorithm 2 Second-order explicit Runge-Kutta method

Require: An IVP of the form $y' = f(t, y)$, $a \leq t \leq b$, $y(a) = \alpha$ and a step length h.

$N \leftarrow \dfrac{b-a}{h}$
$y_0 \leftarrow \alpha$ and $t_0 \leftarrow a$
for $n = 0, \ldots, N-1$ **do**
  $k_1 \leftarrow hf(t_n, y_n)$
  $k_2 \leftarrow hf(t_n + c_2 h, y_n + a_{21}k_1)$
  $y_{n+1} \leftarrow y_n + b_1 k_1 + b_2 k_2$
  $t_{n+1} \leftarrow t_n + h$
end for
return $\mathbf{t} = (t_0, \ldots, t_N)$ and $\mathbf{y} = (y_0, \ldots, y_N)$

Example 2.3.2. Use the classical second-order Runge-Kutta method to calculate the solution to the following IVP using a step length of $h = 0.2$

$$y' = t + y, \qquad 0 \leq t \leq 1, \qquad y(0) = 1.$$

Since $a = 0$ and $b = 1$ we have

$$N = \frac{1-0}{0.2} = 5,$$

and $t_0 = 0$ and $y_0 = 1$.
$n = 0$:

$$\begin{aligned} k_1 &= hf(t_0, y_0) = 0.2(0+1) = 0.2, \\ k_2 &= hf\left(t_0 + h, y_0 + k_1\right) = 0.2(0.2 + 1.2) = 0.28, \\ y_1 &= y_0 + \frac{1}{2}(k_1 + k_2) = 1 + \frac{1}{2}(0.2 + 0.28) = 1.24, \\ t_1 &= t_0 + h = 0 + 0.2 = 0.2. \end{aligned}$$

$n = 1$:

$$\begin{aligned} k_1 &= hf(t_1, y_1) = 0.2(0.2 + 1.24) = 0.288, \\ k_2 &= hf(t_1 + h, y_1 + k_1) = 0.2(0.4 + 1.528) = 0.3856, \\ y_2 &= y_1 + \frac{1}{2}(k_1 + k_2) = 1.24 + \frac{1}{2}(0.288 + 0.3856) = 1.5768, \\ t_2 &= t_1 + h = 0.2 + 0.2 = 0.4. \end{aligned}$$

$n = 2$:

$$
\begin{aligned}
k_1 &= hf(t_2, y_2) = 0.2(0.4 + 1.5768) = 0.39536, \\
k_2 &= hf(t_2 + h, y_2 + k_1) = 0.2(0.6 + 1.97216) = 0.514432, \\
y_3 &= y_2 + \frac{1}{2}(k_1 + k_2) = 1.5768 + \frac{1}{2}(0.39536 + 0.514432) = 2.031696, \\
t_3 &= t_2 + h = 0.4 + 0.2 = 0.6.
\end{aligned}
$$

$n = 3$:

$$
\begin{aligned}
k_1 &= hf(t_3, y_3) = 0.2(0.6 + 2.031696) = 0.526339, \\
k_2 &= hf(t_3 + h, y_3 + k_1) = 0.2(0.8 + 2.558035) = 0.671607, \\
y_4 &= y_3 + \frac{1}{2}(k_1 + k_2) = 2.031696 + \frac{1}{2}(0.526339 + 0.671607) = 2.630669, \\
t_4 &= t_3 + h = 0.6 + 0.2 = 0.8.
\end{aligned}
$$

$n = 4$:

$$
\begin{aligned}
k_1 &= hf(t_4, y_4) = 0.2(0.8 + 2.630669) = 0.686134, \\
k_2 &= hf(t_4 + h, y_4 + k_1) = 0.2(1 + 3.316803) = 0.863361, \\
y_5 &= y_4 + \frac{1}{2}(k_1 + k_2) = 2.630669 + \frac{1}{2}(0.686134 + 0.863361) = 3.405416, \\
t_5 &= t_4 + h = 0.8 + 0.2 = 1.
\end{aligned}
$$

The exact solution to this IVP is $y = 2e^t - t - 1$ so we can calculate the exact solution corresponding to each value of t and the absolute error between the computed solution and the exact solution (Table 2.1).

Table 2.1: Exact and computed solutions using the second-order Runge-Kutta method for the IVP example.

t	RK2	Exact	$\lvert y_{exact} - y_{computed} \rvert$
0.00	1.000000	1.000000	0.000000
0.20	1.240000	1.242806	0.002806
0.40	1.576800	1.583649	0.006849
0.60	2.031696	2.044238	0.012542
0.80	2.630669	2.651082	0.020413
1.00	3.405416	3.436564	0.031147

□

2.3.3 MATLAB code

A MATLAB function for calculating the solution of an IVP using the second-order Runge-Kutta method is shown in Listing 2.1.

Listing 2.1: MATLAB function for the second-order Runge-Kutta method.

```
function [t,y] = rk2(f, a, b, ya, h)

% rk2.m by Jon Shiach
%
% This function solves the IVP f' = f(t,y), a <= t <= b, y(a) = ya using
% the classical 2nd order explicit Runge-Kutta method with step length h
```

```
% calculate number of steps
nsteps = (b - a)/h;

% calculate the steps
y(1) = ya;
t(1) = a;
for n = 1 : nsteps
    k1 = h*f(t(n),y(n));
    k2 = h*f(t(n) + h,y(n) + k1);
    y(n+1) = y(n) + 0.5*(k1 + k2);
    t(n+1) = t(n) + h;
end
```

Example 2.3.3. Use the MATLAB function given in Listing 2.1 to calculate the solution of the IVP shown in Example 2.3.2 using a step length $h = 0.2$.

The MATLAB program shown in Listing 2.2 defines the IVP and invokes the `rk2` function from Listing 2.1 to calculate the solution.

Listing 2.2: MATLAB program used to solve the IVP given in Example 2.3.2.

```
% rk2_ex1.m by Jon Shiach
%
% This program solves the IVP y' = t + y, 0 <= t <= 1, y(0) = ya using the
% classical second-order Runge-Kutta method

% clear workspaces (should always to be done at the beginning of an m-file)
clear
clc

% define solution variables
a = 0;              % lower bound for t
b = 1;              % upper bound for t
ya = 1;             % initial value of y
h = 0.2;            % step length

% define ODE function
f = @(t,y) t + y;

% call the RK2 function to solve IVP
[t,y] = rk2(f, a, b, ya, h);

% calculate the absolute error between the exact and computed solutions
y_ex = 2*exp(t) - t - 1;
err = abs(y_ex - y);

% output solution
fprintf('\n---------------------------------------\n')
fprintf('  t    |   RK2    |   Exact   |   Error  \n')
fprintf('---------------------------------------\n')
for i = 1 : length(t)
    fprintf('%5.2f | %8.6f | %8.6f | %8.6f\n',t(i),y(i),y_ex(i),err(i))
end
fprintf('---------------------------------------\n\n')
```

Running the m-file in Listing 2.2 produces the following output which are the same values as shown in Table 2.1 (as expected).

```
---------------------------------------
  t    |    RK2     |    Exact   |    Error
---------------------------------------
 0.00 | 1.000000 | 1.000000 | 0.000000
 0.20 | 1.240000 | 1.242806 | 0.002806
 0.40 | 1.576800 | 1.583649 | 0.006849
 0.60 | 2.031696 | 2.044238 | 0.012542
 0.80 | 2.630669 | 2.651082 | 0.020413
 1.00 | 3.405416 | 3.436564 | 0.031147
---------------------------------------
```

It is always a good idea to check that the results outputted by a program match those calculated using a pen and paper (known as a *hand calculation*) and that no mistakes or 'bugs' in the program are causing errors in the solution. This process is known as *verifying* the code. □

2.4 Third-order Runge Kutta methods

The general form of the third-order Runge-Kutta method is

$$\begin{aligned} k_1 &= hf(t_n, y_n), \\ k_2 &= hf(t_n + c_2 h, y_n + a_{21}k_1), \\ k_3 &= hf(t_n + c_3 h, y_n + a_{31}k_1 + a_{32}k_2), \\ y_{n+1} &= y_n + b_1 k_1 + b_2 k_2 + b_3 k_3, \end{aligned}$$

which can be written as the Butcher tableau

$$\begin{array}{c|ccc} 0 & & & \\ c_2 & a_{21} & & \\ c_3 & a_{31} & a_{32} & \\ \hline & b_1 & b_2 & b_3 \end{array}$$

If we do a Taylor series expansion of both the approximated solution and the exact solution as we did for the second-order methods, then in order for the two solutions to be equivalent up to third-order the following conditions must be satisfied (the derivation of these is outside the scope of this unit)

$$b_1 + b_2 + b_3 = 1, \tag{2.10a}$$

$$c_2 b_2 + c_3 b_3 = \frac{1}{2}, \tag{2.10b}$$

$$b_2 a_{21} + b_3(a_{21} + a_{32}) = \frac{1}{2}, \tag{2.10c}$$

$$b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3}, \tag{2.10d}$$

$$b_2 c_2 a_{21} + b_3 c_3 (a_{31} + a_{32}) = \frac{1}{3}, \tag{2.10e}$$

$$b_2 a_{21}^2 + b_3 (a_{31} + a_{32})^2 = \frac{1}{3}, \tag{2.10f}$$

$$b_3 c_2 a_{32} = \frac{1}{6}, \tag{2.10g}$$

$$b_3 a_{21} a_{32} = \frac{1}{6}. \tag{2.10h}$$

This gives eight equations in eight unknowns, but these equations are not independent. From Eqs. (2.10g) and (2.10h) we have $c_2 = a_{21}$ and from Eqs. (2.10d) and (2.10e) we have $c_3 = a_{31} + a_{32}$ (i.e., the row sum condition) so this reduces the system of equations to

$$b_1 + b_2 + b_3 = 1, \tag{2.11a}$$
$$b_2c_2 + b_3c_3 = \frac{1}{2}, \tag{2.11b}$$
$$b_2c_2^2 + b_3c_3^2 = \frac{1}{3}, \tag{2.11c}$$
$$b_3a_{32}c_2 = \frac{1}{6}. \tag{2.11d}$$

Now we have four equations and six unknowns so there are potentially and infinite number of third-order Runge-Kutta methods. To derive a third-order explicit Runge-Kutta method we do the following:

1. let $c_1 = 0$, $c_s = 1$ and specify one value of c_2, b_1, b_2 or b_3;
2. substitute the known values of b_i and c_i into Eqs. (2.11a) to (2.11c) and solve for the remaining values of b_i and c_i;
3. substitute b_3 and c_2 into Eq. (2.11d) and solve for a_{32}.
4. use the row sum condition to find a_{21} and a_{31}.

Example 2.4.1. Derive a third-order Runge-Kutta method with $c_2 = \frac{1}{2}$.

Since $c_1 = 0$ and $c_3 = 1$ then $\mathbf{c} = (0, \frac{1}{2}, 1)$. We now need to solve for b_1, b_2 and b_3. Substituting the values of c_i into Eqs. (2.11a) to (2.11c) gives the following linear system

$$\begin{aligned} b_1 + b_2 + b_3 &= 1, \\ \frac{1}{2}b_2 + b_3 &= \frac{1}{2}, \\ \frac{1}{4}b_2 + b_3 &= \frac{1}{3}. \end{aligned}$$

Solving gives $b_1 = \frac{1}{6}$, $b_2 = \frac{2}{3}$ and $b_3 = \frac{1}{6}$. We now need to find the values of a_{ij}. Substituting b_3 and c_2 into Eq. (2.11d) gives

$$\frac{1}{6}a_{32}\frac{1}{2} = \frac{1}{6},$$

so $a_{32} = 2$. For the remaining values of a_{ij} we use the row sum condition, i.e.,

$$\begin{aligned} a_{21} &= c_2 = \frac{1}{2}, \\ a_{31} &= c_3 - a_{32} = -1. \end{aligned}$$

The Butcher tableau for this method is

0			
$\frac{1}{2}$	$\frac{1}{2}$		
1	-1	2	
	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$

which corresponds to the iterative scheme

$$k_1 = hf(t_n, y_n), \tag{2.12a}$$

$$k_2 = hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right), \tag{2.12b}$$

$$k_3 = hf\left(t_n + h, y_n - k_1 + 2k_2\right), \tag{2.12c}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 4k_2 + k_3). \tag{2.12d}$$

The scheme given in Eqs. (2.12a) to (2.12d) is known as *Kutta's third-order method* or more commonly the *classical third-order Runge-Kutta method* (or RK3 for short). □

2.5 Fourth-order Runge-Kutta methods

We will now go on to look at fourth-order Runge-Kutta methods. These methods are the most common Runge-Kutta methods used in practice since they offer a good compromise between accuracy and computational complexity. The general form of the fourth-order explicit Runge-Kutta method is

$$\begin{aligned}
k_1 &= hf(t_n, y_n),\\
k_2 &= hf(t_n + c_2 h, y_n + a_{21}k_1),\\
k_3 &= hf(t_n + c_3 h, y_n + a_{31}k_1 + a_{32}k_2),\\
k_4 &= hf(t_n + c_4 h, y_n + a_{41}k_1 + a_{42}k_2 + a_{43}k_3),\\
y_{n+1} &= y_n + b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4,
\end{aligned}$$

which can be written as the Butcher tableau

0				
c_2	a_{21}			
c_3	a_{31}	a_{32}		
c_4	a_{41}	a_{42}	a_{43}	
	b_1	b_2	b_3	b_4

If we do a Taylor series expansion of both the approximated solution and the exact solution as we did for the second and third-order methods, then for the two solutions to be equivalent up to fourth-order the following conditions must be satisfied (the full derivation of these conditions is outside the scope of this unit):

$$\begin{aligned}
b_1 + b_2 + b_3 + b_4 &= 1,\\
b_2 c_2 + b_3 c_3 + b_4 c_4 &= \frac{1}{2},\\
b_2 c_2^2 + b_3 c_3^2 + b_4 c_4^2 &= \frac{1}{3},\\
b_2 c_2^3 + b_3 c_3^3 + b_4 c_4^3 &= \frac{1}{4},\\
b_3 c_3 a_{32} c_2 + b_4 c_4 (a_{42} c_2 + a_{43} c_3) &= \frac{1}{8},\\
b_3 a_{32} c_2 + b_4 (a_{42} c_2 + a_{43} c_3) &= \frac{1}{6}, \qquad (2.13a)\\
b_3 a_{32} c_2^2 + b_4 (a_{42} c_2^2 + a_{43} c_3^2) &= \frac{1}{12}, \qquad (2.13b)\\
b_4 a_{43} a_{32} c_2 &= \frac{1}{24}. \qquad (2.13c)
\end{aligned}$$

Although it is possible to derive methods directly from these equations it is usual to apply a simplifying assumption. The following condition is known as the $D(1)$ condition (Butcher, 1987)

$$\sum_{i=j+1}^{s} b_i a_{ij} = b_j(1 - c_j), \qquad j = 1, \ldots, s,$$

which can be used to replace Eqs. (2.13a) to (2.13c). Imposing this condition the conditions for a fourth-order method reduce to*

$$b_1 + b_2 + b_3 + b_4 = 1, \tag{2.14a}$$
$$b_2c_2 + b_3c_3 + b_4c_4 = \frac{1}{2}, \tag{2.14b}$$
$$b_2c_2^2 + b_3c_3^2 + b_4c_4^2 = \frac{1}{3}, \tag{2.14c}$$
$$b_2c_2^3 + b_3c_3^3 + b_4c_4^3 = \frac{1}{4}, \tag{2.14d}$$
$$b_3c_3a_{32}c_2 + b_4c_4(a_{42}c_2 + a_{43}c_3) = \frac{1}{8}, \tag{2.14e}$$
$$b_3a_{32} + b_4a_{42} = b_2(1 - c_2), \tag{2.14f}$$
$$b_4a_{43} = b_3(1 - c_3), \tag{2.14g}$$
$$0 = b_4(1 - c_4). \tag{2.14h}$$

Equation (2.14h) gives us $c_4 = 1$ and for a four-stage method $b_4 \neq 0$. Substituting $c_4 = 1$ into Eqs. (2.14a) to (2.14d) gives a system of four equations in terms of the six unknowns b_1, b_2, b_3, b_4, c_2 and c_3 (remember that $c_1 = 0$). In order to solve this system we choose two of these unknowns to be free parameters and we solve for the remaining unknowns.

To derive a fourth order explicit Runge-Kutta method do the following:

1. let $c_1 = 0$ and $c_4 = 1$ and specify values for any two from c_2, c_3, b_1, b_2, b_3 and b_4. Substitute the known values into Eqs. (2.14a) to (2.14d) and solve for the remaining values of b_i and c_i;
2. substitute b_3, b_4 and c_3 into Eq. (2.14g) and solve to give a_{43};
3. substitute b_i and c_i into Eqs. (2.14e) and (2.14f) and solve for a_{32} and a_{42};
4. determine the values of a_{21}, a_{31} and a_{41} using the row sum condition.

Example 2.5.1. Derive a fourth-order Runge-Kutta method with $c_2 = \frac{1}{2}$ and $c_3 = \frac{1}{2}$.

Since $c_1 = 0$ and $c_4 = 1$ then we have the vector $\mathbf{c} = (0, \frac{1}{2}, \frac{1}{2}, 1)^T$. We need to solve for b_1, b_2, b_3 and b_4. Substituting the values of $\mathbf{c}$ into Eqs. (2.14a) to (2.14d) gives the following linear system

$$\begin{aligned}
b_1 + b_2 + b_3 + b_4 &= 1, \\
\frac{1}{2}b_2 + \frac{1}{2}b_3 + b_4 &= \frac{1}{2}, \\
\frac{1}{4}b_2 + \frac{1}{4}b_3 + b_4 &= \frac{1}{3}, \\
\frac{1}{8}b_2 + \frac{1}{8}b_3 + b_4 &= \frac{1}{4}.
\end{aligned}$$

We can use MATLAB's `solve` function to solve this system

*Students should note that any order conditions that you may need will be provided in the exam. You do not need to memorise these.

```
>> [b1,b2,b3,b4] = solve('b1+b2+b3+b4=1','b2*1/2+b3*1/2+b4=1/2',...
'b2*1/4+b3*1/4+b4=1/3','b2*1/8+b3*1/8+b4=1/4')
Warning: The solutions are parametrized by the symbols:
z = C_

> In solve at 94

b1 =

1/6


b2 =

2/3 - z


b3 =

z


b4 =

1/6
```

Solving gives $b_1 = \frac{1}{6}$, $b_2 = \frac{2}{3} - z$, $b_3 = z$ and $b_4 = \frac{1}{6}$. Here z is a free parameter, if we let $z = \frac{1}{3}$ then $b_2 = b_3 = \frac{1}{3}$. Now we need to solve for a_{ij}. Substituting $b_3 = \frac{1}{3}$, $b_4 = \frac{1}{6}$ and $c_3 = 1$ into Eq. (2.14g) gives

$$\frac{1}{6}a_{43} = \frac{1}{3}\left(1 - \frac{1}{2}\right),$$

which simplifies to $a_{43} = 1$. Substituting known values of b_i, c_i and a_{43} into Eqs. (2.14e) and (2.14f) gives the linear system

$$2a_{32} + 2a_{42} = 1,$$
$$2a_{32} + a_{42} = 1.$$

This is easily solved to give $a_{42} = 0$ and $a_{32} = \frac{1}{2}$. Substituting the known values of a_{ij}, b_i and c_i into a Butcher tableau

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & a_{21} & & & \\ \frac{1}{2} & a_{31} & \frac{1}{2} & & \\ 1 & a_{41} & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

The remaining values of a_{ij} are calculated using the condition that c_i is equal to the sum of the ith row of A, therefore $a_{21} = \frac{1}{2}$, $a_{31} = 0$ and $a_{41} = 0$ and the completed Butcher tableau is

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

which corresponds to the iterative scheme

$$k_1 = hf(t_n, y_n), \tag{2.15a}$$

$$k_2 = hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right), \tag{2.15b}$$

$$k_3 = hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right), \tag{2.15c}$$

$$k_4 = hf\left(t_n + h, y_n + k_3\right), \tag{2.15d}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{2.15e}$$

Equations (2.15a) to (2.15e) are collectively known as the *classical fourth-order Runge-Kutta method* or simply *the Runge-Kutta method* (RK4 for short). □

2.6 Higher order Runge-Kutta methods

Higher order methods can be derived in a similar way, by comparing the Taylor expansion of the exact solution with the Taylor expansion of the solution given by the method, however the algebra involved in solving the order conditions becomes very tricky. An easier way of deriving the order conditions was derived by Butcher (1987). This introduces the concept of trees, which allows derivation of higher order methods in a sytematic way. This is outside the scope of this unit.

As the order of the method increases, the number of conditions that need to be satisfied and hence the number of stages required to achieve that order increases even faster. As we have seen, we can obtain a fourth order method with only four stages. In order to obtain a fifth order method, we require at least five stages and seventh order method requires at least 9 stages. This is shown in Table 2.2.

Table 2.2: Number of stages required for a particular order Runge-Kutta method.

Order	1	2	3	4		5	6		7		8
No. conditions	1	2	4	8		17	37		85		200
No. stages	1	2	3	4	5	6	7	8	9	10	11
No. parameters	1	3	6	10	15	21	28	36	45	55	66

2.7 Explicit Runge-Kutta methods tutorial exercises

1. Write the Butcher tableau for the following Runge-Kutta method

$$\begin{aligned} k_1 &= hf(t_n, y_n), \\ k_2 &= hf\left(t_n + \frac{1}{4}h, y_n + \frac{1}{4}k_1\right), \\ k_3 &= hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right), \\ k_4 &= hf\left(t_n + h, y_n + k_1 - 2k_2 + 2k_3\right), \\ y_{n+1} &= y_n + \frac{1}{6}k_1 + \frac{2}{3}k_3 + \frac{1}{6}k_4. \end{aligned}$$

2. Write out the equations for the following Runge-Kutta method

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{4} & \frac{1}{4} & & & \\ \frac{1}{2} & -\frac{1}{2} & 1 & & \\ 1 & \frac{1}{4} & 0 & \frac{3}{4} & \\ \hline & 0 & \frac{4}{9} & \frac{1}{3} & \frac{2}{9} \end{array}$$

3. Derive an explicit third-order Runge-Kutta method such that $c_2 = \frac{1}{3}$. Write your answer in the form of a Butcher tableau.

4. Derive an explicit fouth-order Runge-Kutta method such that $b_1 = 0$ and $c_2 = \frac{1}{5}$. Write your answer in the form of a Butcher tableau.

5. Use the classical second-order Runge-Kutta method to calculate the solution to the following IVP

$$y'(t) = y - t, \qquad y(0) = 2,$$

at $y(0.5)$ using a step length of $h = 0.1$. Repeat the calculations for step lengths $h = 0.05$, $h = 0.0125$ and $h = 0.00625$. Compare your solutions at $y(0.5)$ to the exact solution $y(t) = t + e^t + 1$ and calculate the absolute error. What can you say about the behaviour of the error as the step length decreases? Hint: you may use the MATLAB code in Section 2.3.3 to help you.

The solutions to these exercises are given on page 158.

Chapter 3

Implicit Runge-Kutta methods

It was seen in the previous chapter, as the order of an explicit Runge-Kutta method increases, the number of stages and variables increases to the point where it is impractical to use such methods. Fortunately there is another class of Runge-Kutta methods that allow us to increase the order accuracy without incurring this additional complication. These are called *implicit Runge-Kutta methods* (IRK) where an s stage implicit method can have an accuracy order of up to $2s$. In addition to achieving a higher order than we can with explicit methods, implicit methods allow for the solution of stiff systems which cannot be solved using explicit methods.

3.1 Stiffness

The phenomenon of stiffness is not precisely defined in the literature although some attempts at describing a stiff problem are:

- A problem is stiff if it contains widely varying times scales, i.e., some components of the solution decay much more rapidly than others.
- A problem is stiff if the step length is dictated by stability requirements rather than the accuracy requirements.
- A problem is stiff if explicit methods don't work, or work only with a very small step length.
- A linear problem is stiff if all of its eigenvalues of the Jacobian of f differ greatly in magnitude.

Stiffness will be discussed in more depth in Dr. Nejad's part of the unit.

3.2 General form of an implicit Runge-Kutta method

The general form of an implicit Runge-Kutta method is

$$\begin{aligned} k_1 &= hf(t_n + c_1 h, y_n + a_{11}k_1 + a_{12}k_2 + \ldots + a_{1s}k_s), \\ k_2 &= hf(t_n + c_2 h, y_n + a_{21}k_1 + a_{22}k_2 + \ldots + a_{2s}k_s), \\ &\vdots \\ k_s &= hf(t_n + c_s h, y_n + a_{s1}k_1 + a_{s2}k_2 + \ldots + a_{ss}k_s), \\ y_{n+1} &= y_n + b_1 k_1 + b_2 k_2 + \ldots + b_s k_s. \end{aligned}$$

which can be written in a Butcher tableau

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

Notice that the calculation of the stage values k_i require the value of k_i, i.e., these are implicit relationships hence why this type of method is called implicit methods.

3.2.1 Differences between implicit and explicit Runge-Kutta methods

It is easy to determine whether a Runge-Kutta method is implicit or explicit by looking at the A matrix in the Butcher tableau. An explicit method only has non-zero elements in the lower triangular part of A (Fig. 3.1(a)). An implicit method has non-zero elements on the main diagonal and in the upper triangular part of A (Fig. 3.1(b)). Note that due to the row sum condition, c_1 is always zero for an explicit method.

$$\begin{array}{c|cccc} 0 & 0 & 0 & \cdots & 0 \\ c_2 & a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{s,s-1} & 0 \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

(a) Explicit Runge-Kutta method.

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

(b) Implicit Runge-Kutta method.

Figure 3.1: Butcher tableaux for explicit and implicit Runge-Kutta methods.

We saw in Chapter 2 that the stage values for an explicit Runge-Kutta method are

$$\begin{aligned} k_1 &= hf(t_n, y_n), \\ k_2 &= hf(t_n + c_2 h, y_n + a_{21}k_1), \\ &\vdots \\ k_s &= hf(t_n + c_s h, y_n + a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{s,s-1}k_{s-1}). \end{aligned}$$

These stage values can be calculated sequentially since the stage value k_i only depends on the previous stage values $k_{i-1}, k_{i-2}, \ldots, k_1$. Consider the stage values for an implicit Runge-Kutta method

$$\begin{aligned} k_1 &= hf(t_n + c_1 h, y_n + a_{11}k_1 + a_{12}k_2 + \ldots + a_{1s}k_s), \\ k_2 &= hf(t_n + c_2 h, y_n + a_{21}k_1 + a_{22}k_2 + \ldots + a_{2s}k_s), \\ &\vdots \\ k_s &= hf(t_n + c_s h, y_n + a_{s1}k_1 + a_{s2}k_2 + \ldots + a_{ss}k_s). \end{aligned}$$

Here the calculation of a stage value k_i requires the values of all other stage values and not just those preceding it. This means that the calculation of k_i requires solving a system of non-linear equations (usually using Newton's method) so an s stage implicit method requires more computational resources than an s stage explicit method. However, explicit methods cannot be applied to stiff problems and an s stage implicit method can have order up to $2s$ whereas the order of an s stage explicit method is less than or equal to s.

3.3 Determining the order of an IRK method

Recall in the derivation of a 4th order explicit Runge-Kutta method we made use of the $D(1)$ simplifying assumption. There are several other such assumptions that we can use, these are (Butcher, 2008)

$$B(k): \qquad \sum_{i=1}^{s} b_i c_i^{j-1} = \frac{1}{j}, \qquad j = 1, 2, \ldots, k, \tag{3.1a}$$

$$C(k): \qquad \sum_{j=1}^{s} a_{ij} c_j^{l-1} = \frac{1}{l} c_i^l, \qquad i = 1, 2, \ldots, s, \qquad l = 1, 2, \ldots k, \tag{3.1b}$$

$$D(k): \qquad \sum_{i=1}^{s} b_i c_i^{l-1} a_{ij} = \frac{1}{l} b_j (1 - c_j^l), \qquad j = 1, 2, \ldots, s, \qquad l = 1, 2, \ldots, k, \tag{3.1c}$$

If we let $G(p)$ represent the fact that a given implicit Runge-Kutta method has order p, then it can be shown that

$$B(2s) \text{ and } C(s) \text{ and } D(s) \implies G(2s).$$

In other words, to determine the order of a Runge-Kutta method, we need to find the highest value of k for which the $B(k)$, $C\left(\lfloor \frac{1}{2}k \rfloor\right)$ and $D\left(\lfloor \frac{1}{2}k \rfloor\right)$ conditions in Eqs. (3.1a) to (3.1c) are satisfied*.

To determine the order of an implicit Runge-Kutta method we find the highest value of k for which the $B(k)$ condition Eq. (3.1a) is satisfied. Halve this value of k and round down to the nearest whole number and check that the $C(\lfloor \frac{1}{2}k \rfloor)$ and $D(\lfloor \frac{1}{2}k \rfloor)$ conditions, Eqs. (3.1b) and (3.1c), are also satisfied for this value. If they are then the method is of order k, if not then the order of the method is the highest value of k that is satisfied by all of the $B(k)$, $C(\lfloor \frac{1}{2}k \rfloor)$ and $D(\lfloor \frac{1}{2}k \rfloor)$ conditions. If the $C(\lfloor \frac{1}{2}k \rfloor)$ or $D(\lfloor \frac{1}{2}k \rfloor)$ condition is not satisfied for any value of k then the method is first order.

Example 3.3.1. Determine the order of the following implicit Runge-Kutta method.

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{4} & \frac{1}{8} & \frac{1}{8} & & \\ \frac{7}{10} & -\frac{1}{100} & \frac{14}{25} & \frac{3}{20} & \\ 1 & \frac{2}{7} & 0 & \frac{5}{7} & \\ \hline & \frac{1}{14} & \frac{32}{81} & \frac{250}{567} & \frac{5}{54} \end{array}$$

*$\lfloor x \rfloor$ is the floor operator which is the value of x rounded down to the nearest integer.

This is a 4 stage IRK method so it may have an order of 8. First check the $B(k)$ condition

$$
\begin{aligned}
j=1: &\qquad \sum_{i=1}^{4} b_i c_i^0 = \frac{1}{14}+\frac{32}{81}+\frac{250}{567}+\frac{5}{54}=1,\\
j=2: &\qquad \sum_{i=1}^{4} b_i c_i^1 = \frac{1}{14}\cdot 0+\frac{32}{81}\cdot\frac{1}{4}+\frac{250}{567}\cdot\frac{7}{10}+\frac{5}{54}\cdot 1=\frac{1}{2},\\
j=3: &\qquad \sum_{i=1}^{4} b_i c_i^2 = \frac{1}{14}\cdot 0^2+\frac{32}{81}\left(\frac{1}{4}\right)^2+\frac{250}{567}\left(\frac{7}{10}\right)^2+\frac{5}{54}\cdot 1^2=\frac{1}{3},\\
j=4: &\qquad \sum_{i=1}^{4} b_i c_i^3 = \frac{1}{14}\cdot 0^3+\frac{32}{81}\left(\frac{1}{4}\right)^3+\frac{250}{567}\left(\frac{7}{10}\right)^3+\frac{5}{54}\cdot 1^3=\frac{1}{4},\\
j=5: &\qquad \sum_{i=1}^{4} b_i c_i^4 = \frac{1}{14}\cdot 0^4+\frac{32}{81}\left(\frac{1}{4}\right)^4+\frac{250}{567}\left(\frac{7}{10}\right)^4+\frac{5}{54}\cdot 1^4=\frac{1}{5},\\
j=6: &\qquad \sum_{i=1}^{4} b_i c_i^5 = \frac{1}{14}\cdot 0^5+\frac{32}{81}\left(\frac{1}{4}\right)^5+\frac{250}{567}\left(\frac{7}{10}\right)^5+\frac{5}{54}\cdot 1^5=\frac{401}{2400}\neq\frac{1}{6}.
\end{aligned}
$$

Since all of the $B(k)$ conditions are satisfied upto $k=5$ then this method has a maximum order of 5. Recall that $B(2s)$ and $C(s)$ and $D(s) \implies G(2s)$, then in order for the method to have an order of 5, $C(2)$ and $D(2)$ must both be satisfied (we don't need to check $C(3)$ and $D(3)$ since we know that this method cannot have order 6). Checking the $C(2)$ condition we need to check all combinations of $i=1,2,3,4$ and $l=1,2$. First, check for the i values when $l=1$

$$
\begin{aligned}
i=1: &\qquad \sum_{j=1}^{4} a_{1j} c_j^0 = 0+0+0+0=0=c_1^1,\\
i=2: &\qquad \sum_{j=1}^{4} a_{2j} c_j^0 = \frac{1}{8}+\frac{1}{8}+0+0=\frac{1}{4}=c_2^1,\\
i=3: &\qquad \sum_{j=1}^{4} a_{3j} c_j^0 = -\frac{1}{100}+\frac{14}{25}+\frac{3}{20}+0=\frac{7}{10}=c_3^1,\\
i=4: &\qquad \sum_{j=1}^{4} a_{4j} c_j^0 = \frac{2}{7}+0+\frac{5}{7}+0=1=c_4^1
\end{aligned}
$$

Next we check the i values when $l=2$

$$
\begin{aligned}
i=1: &\qquad \sum_{j=1}^{4} a_{1j} c_j^1 = 0\cdot 0+0\cdot\frac{1}{4}+0\cdot\frac{7}{10}+0\cdot 1=0=\frac{1}{2}0^2=\frac{1}{2}c_1^2,\\
i=2: &\qquad \sum_{j=1}^{4} a_{2j} c_j^1 = \frac{1}{8}\cdot 0+\frac{1}{8}\cdot\frac{1}{4}+0\cdot\frac{7}{10}+0\cdot 1=\frac{1}{32}=\frac{1}{2}\left(\frac{1}{4}\right)^2=\frac{1}{2}c_2^2,\\
i=3: &\qquad \sum_{j=1}^{4} a_{3j} c_j^1 = -\frac{1}{100}\cdot 0+\frac{14}{25}\cdot\frac{1}{4}+\frac{3}{20}\cdot\frac{7}{10}+0\cdot 1=\frac{49}{200}=\frac{1}{2}\left(\frac{7}{10}\right)^2=\frac{1}{2}c_3^2,\\
i=4: &\qquad \sum_{j=1}^{4} a_{4j} c_j^1 = \frac{2}{7}\cdot 0+0\cdot\frac{1}{4}+\frac{5}{7}\cdot\frac{7}{10}+0\cdot 1=\frac{1}{2}=\frac{1}{2}\cdot 1^2=\frac{1}{2}c_4^2.
\end{aligned}
$$

So the $C(2)$ condition is satisfied. Checking $D(2)$ with $l = 1$

$$
\begin{aligned}
j = 1: \qquad \sum_{i=1}^{4} b_i c_i^0 a_{i1} &= \frac{1}{14} \cdot 0^0 \cdot 0 + \frac{32}{81}\left(\frac{1}{4}\right)^0 \frac{1}{8} - \frac{250}{567}\left(\frac{7}{10}\right)^0 \frac{1}{100} + \frac{5}{54} \cdot 1^0 \cdot \frac{2}{7} \\
&= \frac{1}{14} = \frac{1}{14}(1-0) = b_1(1-c_1^1), \\
j = 2: \qquad \sum_{i=1}^{4} b_i c_i^0 a_{i2} &= \frac{1}{14} \cdot 0^0 \cdot 0 + \frac{32}{81}\left(\frac{1}{4}\right)^0 \frac{1}{8} + \frac{250}{367}\left(\frac{7}{10}\right)^0 \frac{14}{25} + \frac{5}{54} \cdot 1^0 \cdot 0 \\
&= \frac{8}{27} = \frac{32}{81}\left(1 - \frac{1}{4}\right) = b_2(1-c_2^1), \\
j = 3: \qquad \sum_{i=1}^{4} b_i c_i^0 a_{i3} &= \frac{1}{14} \cdot 0^0 \cdot 0 + \frac{32}{81}\left(\frac{1}{4}\right)^0 (0) + \frac{250}{367}\left(\frac{7}{10}\right)^0 \frac{3}{20} + \frac{5}{54} \cdot 1^0 \cdot \frac{5}{7} \\
&= \frac{25}{189} = \frac{250}{567}\left(1 - \frac{7}{10}\right) = b_3(1-c_3^1), \\
j = 4: \qquad \sum_{i=1}^{4} b_i c_i^0 a_{i4} &= \frac{1}{14} \cdot 0^0 \cdot 0 + \frac{32}{81}\left(\frac{1}{4}\right)^0 \cdot 0 + \frac{250}{567}\left(\frac{7}{10}\right)^0 \cdot 0 + \frac{5}{54} \cdot 1^0 \cdot 0 \\
&= 0 = \frac{5}{54}(1-1) = b_4(1-c_4^1).
\end{aligned}
$$

Finally, we check the j values when $l = 2$

$$
\begin{aligned}
j = 1: \qquad \sum_{i=1}^{4} b_i c_i^1 a_{i1} &= \frac{1}{14} \cdot 0 \cdot 0 + \frac{32}{81} \cdot \frac{1}{4} \cdot \frac{1}{8} + \frac{250}{567} \cdot \frac{7}{10} \cdot -\frac{1}{100} + \frac{5}{54} \cdot 1 \cdot \frac{2}{7} \\
&= \frac{1}{28} = \frac{1}{2} \cdot \frac{1}{14}(1-0^2) = \frac{1}{2} b_1(1-c_1^2), \\
j = 2: \qquad \sum_{i=1}^{4} b_i c_i^1 a_{i2} &= \frac{1}{14} \cdot 0 \cdot 0 + \frac{32}{81} \cdot \frac{1}{4} \cdot \frac{1}{8} + \frac{250}{567} \cdot \frac{7}{10} \cdot \frac{14}{25} + \frac{5}{54} \cdot 1 \cdot 0 \\
&= \frac{5}{27} = \frac{1}{2} \cdot \frac{32}{81}\left[1 - \left(\frac{1}{4}\right)^2\right] = \frac{1}{2} b_2(1-c_2^2), \\
j = 3: \qquad \sum_{i=1}^{4} b_i c_i^1 a_{i3} &= \frac{1}{14} \cdot 0 \cdot 0 + \frac{32}{81} \cdot \frac{1}{4} \cdot 0 + \frac{250}{567} \cdot \frac{7}{10} \cdot \frac{3}{20} + \frac{5}{54} \cdot 1 \cdot \frac{5}{7} \\
&= \frac{86}{756} = \frac{1}{2} \cdot \frac{7}{10}\left[1 - \left(\frac{7}{10}\right)^2\right] = \frac{1}{2} b_3(1-c_3^2), \\
j = 4: \qquad \sum_{i=1}^{4} b_i c_i^1 a_{i4} &= \frac{1}{14} \cdot 0 \cdot 0 + \frac{32}{81} \cdot \frac{1}{4} \cdot 0 + \frac{250}{567} \cdot \frac{7}{10} \cdot 0 + \frac{5}{54} \cdot 1 \cdot 0 \\
&= 0 = \frac{1}{2} \cdot \frac{5}{54}(1-1^2) = \frac{1}{2} b_4(1-c_4^2).
\end{aligned}
$$

So the $B(5)$, $C(2)$ and $D(2)$ conditions are all satisfied so this implicit Runge-Kutta method is an order 5 method. Note that if either the $C(2)$ or $D(2)$ condition were not satisfied but both were satisfied for $k = 1$ then this method would be an order 2 method since $B(2)$, $C(1)$ and $D(1)$ would all be satisfied. □

3.4 Gauss-Legendre methods

Gauss-Legendre methods are a family of methods that are derived using Gauss-Legendre quadrature. The solution to the Legendre differential equation can take the form of a number of polynomial functions. Using the values of the roots of these polynomials as the values of c_i in an implicit Runge-Kutta method results in a Gauss-Legendre method. All Gauss-Legendre methods are A-stable (see Chapter 4) and an s stage Gauss-Legendre method has order $2s$

Legendre polynomials are the solutions to the Legendre differential equation

$$(1-t^2)y'' - 2ty' + l(l+1)y = 0,$$

over the interval $[-1, 1]$ and expressed explicitly by

$$P_n(t) = 2^n \sum_{k=0}^{n} t^k \binom{n}{k} \binom{\frac{n+k-1}{2}}{n},$$

where $\binom{n}{k}$ is the Binomial coefficient. The first four Legendre polynomials are

$$P_0(t) = 1, \tag{3.2a}$$
$$P_1(t) = t, \tag{3.2b}$$
$$P_2(t) = \frac{1}{2}(3t^2 - 1), \tag{3.2c}$$
$$P_3(t) = \frac{1}{2}(5t^3 - 3t). \tag{3.2d}$$

For a single step method, we are only interested in taking steps from t to $t + h$ so we want the Legendre polynomials in the interval $[0, 1]$. To do this we can make use of shifted Legendre polynomials where t is shifted using the mapping $t \mapsto 2t - 1$ in Eqs. (3.2a) to (3.2d). The first four shifted Legendre polynomials are therefore

$$P_0^*(t) = 1, \tag{3.3a}$$
$$P_1^*(t) = 2t - 1, \tag{3.3b}$$
$$P_2^*(t) = 6t^2 - 6t + 1, \tag{3.3c}$$
$$P_3^*(t) = 20t^3 - 30t^2 + 12t - 1. \tag{3.3d}$$

If we let $c_1, c_2, \ldots, c_s$ denote the roots of $P_n^*(t)$ then it can be shown that there exists positive numbers $b_1, b_2, \ldots, b_s$ such that

$$\int_0^1 \phi(t)dt = \sum_{i=1}^{s} b_i \phi(c_i),$$

for any polynomial $\phi(t)$ of degree $2s$ or less.

A Gauss-Legendre method can be derived using the following steps

1. Choose $c_1, c_2, \ldots, c_s$ as the roots of $P_s^*(t)$;
2. Choose $b_1, b_2, \ldots, b_s$ to satisfy the $B(2s)$ condition;
3. Choose a_{ij} to satisfy the $C(s)$ condition

Example 3.4.1. Derive a fourth order Gauss-Legendre method.

Since a Gauss-Legendre method has order $2s$, then for a fourth order method $s = 2$. The values of c_i are the roots of the polynomial $P_2^*(t) = 6t^2 - 6t + 1$ (Eq. (3.3c)). Using the quadratic formula gives $c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}$ and $c_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}$.

The values of b_i need to satisfy the $B(4)$ condition. Recall Eq. (3.1a)

$$B(k): \qquad \sum_{i=1}^{s} b_i c_i^{j-1} = \frac{1}{j}, \qquad j = 1, 2, \ldots, k,$$

then b_1 and b_2 must satisfy

$$\begin{aligned}
b_1 + b_2 &= 1, \\
\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right) b_1 + \left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right) b_2 &= \frac{1}{2}, \\
\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right)^2 b_1 + \left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right)^2 b_2 &= \frac{1}{3}, \\
\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right)^3 b_1 + \left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right)^3 b_2 &= \frac{1}{4}.
\end{aligned}$$

Solving gives $b_1 = b_2 = \frac{1}{2}$. Finally we need to satisfy the $C(2)$ condition. Recall Eq. (3.1b)

$$C(k): \qquad \sum_{j=1}^{s} a_{ij} c_j^{l-1} = \frac{1}{l} c_i^l, \qquad i = 1, 2, \ldots, s, \qquad l = 1, 2, \ldots, k.$$

For $i = 1$, a_{11} and a_{12} must satisfy

$$\begin{aligned}
a_{11} + a_{12} &= \frac{1}{2} - \frac{\sqrt{3}}{6}, \\
\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right) a_{11} + \left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right) a_{12} &= \frac{1}{2}\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right)^2.
\end{aligned}$$

Solving gives $a_{11} = \frac{1}{4}$ and $a_{12} = \frac{1}{4} - \frac{\sqrt{3}}{6}$. For $i = 2$, a_{21} and a_{22} must satisfy

$$\begin{aligned}
a_{21} + a_{22} &= \frac{1}{2} + \frac{\sqrt{3}}{6}, \\
\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right) a_{11} + \left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right) a_{22} &= \frac{1}{2}\left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right)^2.
\end{aligned}$$

Solving gives $a_{21} = \frac{1}{4} + \frac{\sqrt{3}}{6}$ and $a_{22} = \frac{1}{4}$. Now we have the values of c_i, b_i and a_{ij}, the Butcher tableau for the 4th order Gauss-Legendre method is

$$\begin{array}{c|cc}
\frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6} \\
\frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4} \\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}$$

□

3.5 Radau methods

Gauss-Legendre methods give us maximal order for the number of stages, however sometimes it is better the sacrifice order to gain better stability properties (stability is discussed in Chapter 4). An s-stage Radau method have order $2s-1$.

Instead of choosing c_i to obtain as high a degree as possible for polynomials $\phi(c_i)$ such that

$$\int_0^1 \phi(t)dt = \sum_{i=1}^{s} b_i\phi(c_i).$$

we choose either $c_1 = 0$ or $c_s = 1$ and we can still obtain order $2s-1$. There are two types of Radau methods:

Radau I: for a Radau I method we choose $c_1 = 0$ and the remaining c_i values are chosen as the roots of

$$0 = P_s^*(t) + P_{s-1}^*(t).$$

The values of b_i and a_{ij} are calculated using the order conditions $B(k)$ and $C(k)$ respectively.

Radau II: for a Radau II method we choose $c_s = 1$ and the remaining c_i values are chosen as the roots of

$$0 = P_s^*(t) - P_{s-1}^*(t).$$

The values of b_i and a_{ij} are calculated using the order conditions $B(k)$ and $C(k)$ respectively.

Example 3.5.1. Derive the third order Radau I method.

Choosing $c_1 = 0$, the value of c_2 is the root of the polynomial $P_2^*(t) + P_1^*(t)$, i.e.,

$$0 = 6t^2 - 6t + 1 + 2t - 1 = 6t^2 - 4t,$$

which is easily solved to give $c_2 = \frac{2}{3}$. The values of b_i need to satisfy the $B(4)$ condition. The first two equations of the $B(4)$ system are

$$\begin{aligned} b_1 + b_2 &= 1, \\ \frac{2}{3}b_2 &= \frac{1}{2}, \end{aligned}$$

which is solved to give $b_1 = \frac{1}{4}$ and $b_2 = \frac{3}{4}$. The values of a_{ij} need to satisfy the $C(2)$ condition. For $i = 1$ and $l = 2$ we have

$$0a_{11} + \frac{2}{3}a_{12} = 0,$$

and since the row sum of A equals the c value we have $a_{11} = 0$ and $a_{12} = 0$. For $i = 2$ and $l = 2$ we have

$$\frac{2}{3}a_{22} = \frac{1}{2}\left(\frac{2}{3}\right)^2,$$

and $a_{12} = \frac{1}{3}$ and $a_{22} = \frac{1}{3}$. So the Butcher tableau for the third order Radau I method

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{2}{3} & \frac{1}{3} & \frac{1}{3} \\ \hline & \frac{1}{4} & \frac{1}{4} \end{array}$$

□

3.6 DIRK methods

Diagonally Implicit Runge-Kutta (DIRK) methods are implicit methods where the A matrix in the Butcher tableau is lower triangular (as with explicit Runge-Kutta methods) but with at least one non-zero element on the main diagonal, i.e.,

$$\begin{array}{c|cccc} c_1 & a_{11} & & & \\ c_2 & a_{21} & a_{22} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

DIRK methods have the advantage that the stage values k_i can be calculated sequentially, i.e., once k_i is calculated then we can calculate k_{i+1} without having to go back and re-calculate k_i as we have to with other implicit methods.

The derivation of kth-order DIRK method is done by considering the $B(k)$ and $C(\lfloor \frac{1}{2}k \rfloor)$ condition along with $\mathbf{b}^T A\mathbf{c} = \frac{1}{k!}$ (this order condition is derived in Chapter 4, page 45). For example, the order conditions for a third-order DIRK method are

$$b_1 + b_2 = 1, \tag{3.4a}$$

$$b_1 c_1 + b_2 c_2 = \frac{1}{2}, \tag{3.4b}$$

$$b_1 c_1^2 + b_2 c_2^2 = \frac{1}{3}, \tag{3.4c}$$

$$a_{11} = c_1, \tag{3.4d}$$

$$a_{21} + a_{22} = c_2, \tag{3.4e}$$

$$b_1 a_{11} c_1 + b_2(a_{21} c_1 + a_{22} c_2) = \frac{1}{6}. \tag{3.4f}$$

The system of equations Eqs. (3.4a) to (3.4f) can be solved to give

$$b_1 = \frac{c_2 - \frac{1}{2}}{c_2 - c_1}, \qquad b_2 = \frac{\frac{1}{2} - c_1}{c_2 - c_1}, \qquad c_2 = \frac{2 - 3c_1}{3 - 6c_1},$$

$$a_{11} = c_1, \qquad a_{21} = c_2 - a_{22}, \qquad a_{22} = \frac{\frac{1}{6} - b_1 c_1^2}{b_2(c_2 - c_1)},$$

where c_1 is a free parameter. For example, choosing $c_1 = \frac{1}{3}$ gives

$$\begin{array}{c|cc} \frac{1}{3} & \frac{1}{3} & \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

3.6.1 SDIRK methods

Singly Diagonal Implicit Runge-Kutta (SDIRK) methods are a variation on DIRK methods with the additional constraint that the elements on the main diagonal are all the same ($\lambda = a_{11} = a_{22} = \ldots = a_{ss}$), i.e.,

$$\begin{array}{c|cccc} c_1 & \lambda & & & \\ c_2 & a_{21} & \lambda & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & \lambda \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

We can derive a third-order SDIRK method by considering the $D(1)$ order condition in addition to Eqs. (3.4a) to (3.4f). Using Eq. (3.1c) with $k = 1$ gives

$$\begin{aligned} b_1 a_{11} + b_2 a_{21} &= b_1, \\ b_2 a_{22} &= b_2(1 - c_2). \end{aligned}$$

Since $a_{22} = \lambda$ then using using the second equation

$$\begin{aligned} b_2\lambda &= b_2(1 - c_2) \\ \therefore c_2 &= 1 - \lambda. \end{aligned}$$

Substituting $c_2 = 1 - \lambda$ into the solutions for b_1 and b_2 from before

$$\begin{aligned} b_1 &= \frac{\frac{1}{2} - \lambda}{1 - 2\lambda} = \frac{1}{2}, \\ \therefore b_2 &= \frac{1}{2} \end{aligned}$$

Invoking the row sum condition produces the following Butcher tableau

$$\begin{array}{c|cc} \lambda & \lambda & \\ 1-\lambda & 1-2\lambda & \lambda \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

The value of λ needs to satisfy Eq. (3.4f), substituting in the known values

$$\begin{aligned} \frac{1}{2}\lambda^2 + \frac{1}{2}[(1 - 2\lambda)\lambda + \lambda(1 - \lambda)] &= \frac{1}{6} \\ 3\lambda^2 + 3(2\lambda - 3\lambda^2) &= 1, \\ -6\lambda^2 + 6\lambda - 1 &= 0. \end{aligned}$$

Solving the quadratic gives $\lambda = \frac{1}{2} \pm \frac{1}{\sqrt{3}}$. Using the third-order SDIRK method with $\lambda = \frac{1}{2} \pm \frac{1}{\sqrt{3}}$ is A-stable (meaning that we can use any value of the step length - see Chapter 4) whereas the third-order DIRK method derived above is not. SDIRK methods are often preferred to DIRK methods for this reason.

3.7 Implicit Runge-Kutta methods tutorial exercises

1. Determine the order of the following implicit Runge-Kutta method

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

2. Derive a second-order Gauss-Legendre method.
3. Derive the third-order Radau II method.
4. Derive an SDIRK method with $\lambda = 1 - \frac{1}{2}\sqrt{2}$ and $c_2 = 3\lambda$.
5. What are the advantages of using DIRK methods?

The solutions to these tutorial exercises are given on page 160

Chapter 4

Stability

The issue of stability and understanding when a computational method is stable is essential when using numerical methods solve differential equations. If a method used to solve a system of differential equations is unstable, the computed solutions will diverge and it will not be possible to solve the system. The stability of a single step method will depend upon the step length h. Since it is desirable to solve a system of differential equations as quickly as possible, we choose the maximum value of h for which the method remains stable. For some systems, the stability properties of an explicit method will mean that the value of h is small enough to make the application of the method impractical. Such systems are known to as *stiff systems* and require an A-stable implicit method.

4.1 The model problem

The stability of a ODE solver can be explored by solving a model linear problem of the form:

$$y' = \lambda y. \tag{4.1}$$

If $z = h\lambda$, then since the exact solution to this model ODE is $y = \exp(\lambda t)$, for a single step of length h the exact solution will be multiplied by $\exp(h\lambda)$. In the same interval, the approximate solution computed using a single step method will be multiplied by a function of z which is specific to the method being used. This function of z is called the *stability function*. The region in the complex plane such that the computed solution remains bounded after many steps of the method is called the *stability region*.

For the model problem above, it is the left-hand side of the complex plane that is of more interest since the exact solution, $y = \exp(\lambda t)$, is bounded for negative real values of λ (and therefore z). Since we want to compute a close approximation of the exact solution, we want the stability function for a method to behave in a similar way to the exact solution. Hence, our analysis of the stability of single step methods to based on the comparisons between the stability function and $\exp(h\lambda)$.

Definition 4.1.1 (Stability function). A single step method for the model ODE given in Eq. (4.1) can be written in the form

$$y_{n+1} = R(z)y_n, \tag{4.2}$$

where $z = h\lambda$. The function $R(z)$ is the amplification factor from one step to the next and is known as the stability function for that particular single step method.

It can clearly be seen that the iterative scheme in Eq. (4.2) is divergent if $|R(z)| > 1$ and convergent otherwise. This leads to the definition of the *region of absolute stability* for a numerical method.

Definition 4.1.2 (Region of absolute stability). A single step method is stable if its stability function, $R(z)$, satisfies

$$\{z \in \mathbb{C} : |R(z)| \leq 1\},$$

where $\mathbb{C}$ is the set of complex numbers. The region of absolute stability is the region on the complex plane where z satisfies this condition.

Example 4.1.1. Consider the Euler method

$$y_{n+1} = y_n + hf(t_n, y_n),$$

when applied to the model problem $y' = \lambda y$ we have

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

So the incrementing function for this method is

$$R(z) = 1 + z,$$

therefore the region of absolute stability is

$$\{z \in \mathbb{C} : |1 + z| \leq 1\}.$$

The region of absolute stability for the Euler method is plotted in Fig. 4.1. For the method to remain stable, $z = h\lambda$ must lie within this region.

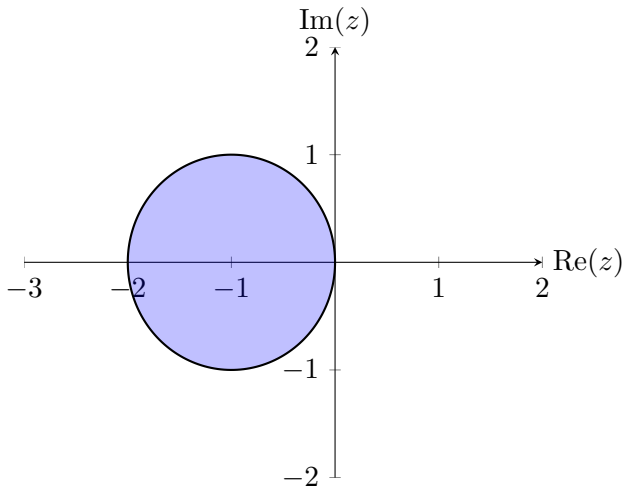


Figure 4.1: The region of absolute stability for the Euler method.

Let's see what happens when the Euler method is used to solve the IVP

$$y' = -2.3y, \qquad y(0) = 1$$

using step sizes $h = 1$ and $h = 0.7$. Comparing the ODE to the model ODE in Eq. (4.1) then $\lambda = -2.3$. When $h = 1$, $z = -2.3$ which lies outside of the stability region and the solution will diverge. When $h = 0.7$, $z = 0.7(-2.3) = -1.61$ which lies within the stability region.

The solutions to this IVP over the range $0 \leq t \leq 5$ using $h = 1$ and $h = 0.7$ is shown in Figure. 4.2. It can be clearly seen that using a step size of $h = 1$ the solution diverges and the method is unstable whereas using a step size of $h = 0.7$ the method will converge to the solution. □

Definition 4.1.3 (A-stable). A method is considered to be A-stable if for the model problem in Eq. (4.1), the stability region includes all of the left hand side of the complex plane $\mathrm{Re}(z) < 0$. For stiff problems, it is considered necessary for a method to be A-stable. An explicit method cannot be A-stable since the stability function of an explicit method is a polynomial function.

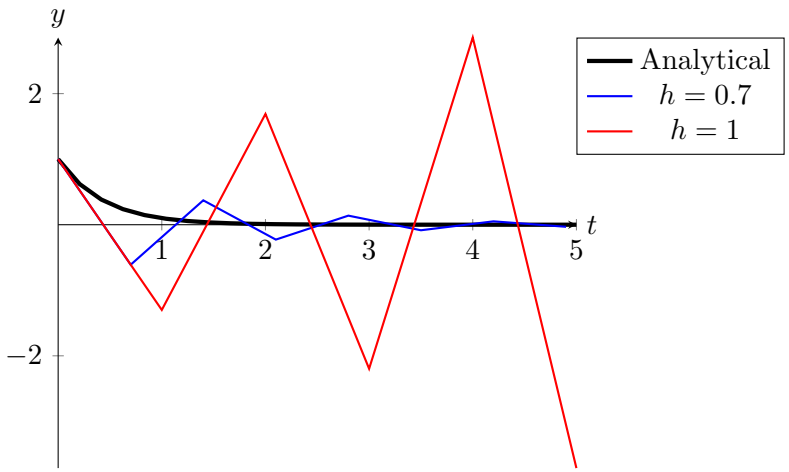


Figure 4.2: Solutions of the IVP $y' = -2.3y$, $y(0) = 0$ using the Euler method with step lengths $h = 0.7$ and $h = 1$.

4.2 Stability function of a Runge-Kutta method

To determine the stability function of an s stage Runge-Kutta method we rewrite the method using vector notation. Consider the general form of the Runge-Kutta method

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i,$$

where

$$k_i = hf\left(t_n + c_i h, y_n + \sum_{j=1}^{s} a_{ij} k_i\right).$$

Let $Y_i = y_n + \sum_{j=1}^{s} a_{ij} k_i$ then the method can be written as

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i f(t_n + c_i h, Y_i).$$

Using the model equation $y' = \lambda y$ we have $f(t, y) = \lambda y$ so we can ignore the $t_n + c_i h$ term and the method becomes

$$y_{n+1} = y_n + h\lambda \sum_{i=1}^{s} b_i Y_i, \tag{4.3}$$

and

$$Y_i = y_n + h\lambda \sum_{j=1}^{s} a_{ij} Y_j. \tag{4.4}$$

Therefore, for an s-stage method we have

$$\begin{aligned} Y_1 &= y_n + h\lambda(a_{11}Y_1 + a_{12}Y_2 + \ldots + a_{1s}Y_s), \\ Y_2 &= y_n + h\lambda(a_{21}Y_1 + a_{22}Y_2 + \ldots + a_{2s}Y_s), \\ &\vdots \\ Y_s &= y_n + h\lambda(a_{s1}Y_1 + a_{s2}Y_2 + \ldots + a_{ss}Y_s). \end{aligned}$$

Let $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_s)^T$ and $\mathbf{1} = (1, 1, \ldots, 1)^T$ then we can write Eq. (4.4) in vector form

$$\mathbf{Y} = y_n\mathbf{1} + h\lambda A\mathbf{Y}, \tag{4.5}$$

which can be rearranged to make $\mathbf{Y}$ the subject

$$\mathbf{Y} = (I - zA)^{-1}\mathbf{1}y_n.$$

Therefore the vector form of the general Runge-Kutta method is

$$y_{n+1} = y_n + z\mathbf{b}^T(I - zA)^{-1}\mathbf{1}y_n,$$

and the stability function for a Runge-Kutta method is

$$R(z) = 1 + z\mathbf{b}^T(I - zA)^{-1}\mathbf{1}. \tag{4.6}$$

4.3 Stability function of an explicit Runge-Kutta method

If the Runge-Kutta method is explicit then we can evaluate the stability function Eq. (4.6) using the geometric series of matrices (see Appendix A on page 155). We know that over one time step the exact solution will be multiplied by $\exp(z)$, so we want our Runge-Kutta method to do the same, i.e.,

$$\exp(z) \approx 1 + z\mathbf{b}^T(I - zA)^{-1}\mathbf{1}.$$

Since the series expansion of $\exp(z)$ is

$$\exp(z) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \ldots + \frac{z^k}{k!} + \ldots, \tag{4.7}$$

and applying the geometric series of matrices to Eq. (4.6) gives

$$\begin{aligned} R(z) &= 1 + z\mathbf{b}^T(I + zA + z^2A^2 + z^3A^3 + \ldots + z^kA^k + \ldots)\mathbf{1}, \\ &= 1 + z\mathbf{b}^T\mathbf{1} + z^2\mathbf{b}^TA\mathbf{1} + z^3\mathbf{b}^TA^2\mathbf{1} \ldots + z^k\mathbf{b}^TA^{k-1}\mathbf{1} + \ldots, \end{aligned} \tag{4.8}$$

then equating Eq. (4.7) and Eq. (4.8) results in the following condition on the coefficients of z^k.

$$\frac{1}{k!} = \mathbf{b}^TA^{k-1}\mathbf{1}, \qquad k = 1, 2, \ldots, p. \tag{4.9}$$

This means that for an order k method to be stable Eq. (4.9) must be satisfied for values up to and including k. When $k = 1$, Eq. (4.9) is

$$1 = \mathbf{b}^TA^0\mathbf{1} = \sum_{i=1}^{s} b_i$$

which is equivalent to the $B(1)$ condition (Eq. (3.1a) on page 31). If we let $\mathbf{c} = (c_1, c_2, \ldots, c_s)^T$ then the vector form of the $C(1)$ condition (Eq. (3.1b)) is

$$\mathbf{c} = A\mathbf{1}.$$

When $k = 2$, Eq. (4.9) is

$$\frac{1}{2} = \mathbf{b}^T A\mathbf{1} = \mathbf{b}^T \mathbf{c}.$$

For all other values of $k > 2$ we can substitute $\mathbf{c} = A\mathbf{1}$ in Eq. (4.9) so that

$$\frac{1}{k!} = \mathbf{b}^T A^{k-1}\mathbf{1} = \mathbf{b}^T A^{k-2}\mathbf{c}, \qquad k = 3, 4, \ldots,$$

Combining these results gives

$$1 = \mathbf{b}^T \mathbf{1}, \qquad k = 1, \tag{4.10a}$$

$$\frac{1}{k!} = \mathbf{b}^T A^{k-2}\mathbf{c}, \qquad k \geq 2. \tag{4.10b}$$

Example 4.3.1. Determine the stability function for the following Runge-Kutta method and give the order of the method.

$$\begin{aligned}
k_1 &= hf(t_n, y_n), \\
k_2 &= hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right), \\
k_3 &= hf\left(t_n + \frac{3}{4}h, y_n + \frac{3}{4}k_2\right), \\
k_4 &= hf\left(t_n + h, y_n + \frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3\right), \\
y_{n+1} &= y_n + \frac{7}{24}k_1 + \frac{1}{4}k_2 + \frac{1}{3}k_3 + \frac{1}{8}k_4.
\end{aligned}$$

Here we have an explicit method with

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{3}{4} & 0 & 0 \\ \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & 0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} \frac{7}{24} \\ \frac{1}{4} \\ \frac{1}{3} \\ \frac{1}{8} \end{pmatrix}, \qquad \mathbf{c} = \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{3}{4} \\ 1 \end{pmatrix}.$$

Using MATLAB to calculate the coefficients in Eqs. (4.10a) and (4.10a)

```
>> a = [0 0 0 0 ; 1/2 0 0 0 ; 0 3/4 0 0 ; 2/9 1/3 4/9 0];
>> b = [7/24 1/4 1/3 1/8]';
>> c = [0 1/2 3/4 1]';
>> rats(b'*ones(4,1)) % coefficient of z

ans =

       1

>> rats(b'*c) % coefficient of z^2

ans =

      1/2
```

```
>> rats(b'*A*c) % coefficient of z^3

ans =

      3/16

>> rats(b'*A^2*c) % coefficient of z^4

ans =

      1/48
```

So the stability function is

$$R(z) = 1 + z + \frac{1}{2}z^2 + \frac{3}{16}z^3 + \frac{1}{48}z^4.$$

Since the stability function only agrees with the expansion of $\exp(z)$ (Eq. (4.7)) up to the $p = 2$ term then we can say that this method is a second order method. □

4.3.1 Plotting stability regions

Stability regions where the stability function is bounded can be plotted on the complex plane using MATLAB. To plot the region of absolute stability where $|R(z)| < 1$ we generate a matrix of z values and then calculate the value of the stability function for each of these. Using the `contour` plot function we can plot a single contour line that corresponds with the value $R(z) = 1$ that represents the boundary of the stability region.

The MATLAB code for plotting the stability region of an explicit fourth-order Runge-Kutta method is given below.

Listing 4.1: MATLAB code used to plot the stability regions of explicit Runge-Kutta methods

```
% plot_erk_stability.m by Jon Shiach
%
% This program plots the stability regions of explicit Runge-Kutta methods

% clear workspaces
clear
clc

% generate matrix of z values
x = -10 : 0.1 : 5;
y = -10 : 0.1 : 10;
[x, y] = meshgrid(x, y);
z = x + 1i*y;

% define RK stability functions
R1 = 1 + z;
R2 = 1 + z + 1/2*z.^2;
R3 = 1 + z + 1/2*z.^2 + 1/6*z.^3;
R4 = 1 + z + 1/2*z.^2 + 1/6*z.^3 + 1/24*z.^4;

% plot regions
hold on
contour(x, y, abs(R1), [1, 1], 'g-', 'linewidth', 2)
contour(x, y, abs(R2), [1, 1], 'k-', 'linewidth', 2)
contour(x, y, abs(R3), [1, 1], 'b-', 'linewidth', 2)
contour(x, y, abs(R4), [1, 1], 'r-', 'linewidth', 2)

% plot axes lines
plot([-10 10], [0 0], 'k-')
plot([0 0], [-10 10], 'k-')
```

```
hold off

% format plot
axis([-5 2 -3.5 3.5])
xlabel('Re(z)', 'fontsize', 14)
ylabel('Im(z)', 'fontsize', 14)
text(-0.5, 3.25, 'p=4', 'fontsize', 14)
text(-0.5, 2.5, 'p=3', 'fontsize', 14)
text(-0.75, 1.9, 'p=2', 'fontsize', 14)
text(-1, 1.2, 'p=1', 'fontsize', 14)
box on
grid on
```

The stability regions for the explicit Runge-Kutta methods of order $p = 1, 2, 3, 4$ have been plotted on the same set of axes in Fig. 4.3.

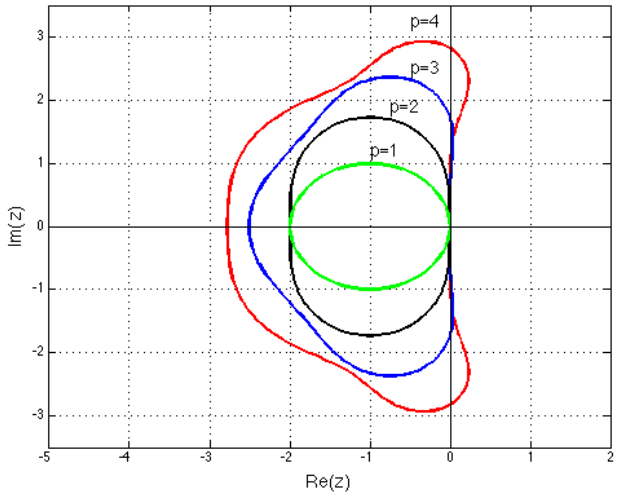


Figure 4.3: Regions of absolute stability for the explicit Runge-Kutta methods up to order 4.

4.3.2 Implicit methods

The stability function of an implicit Runge-Kutta method can be calculated using Cramer's rule on Eq. (4.6) to give

$$R(z) = \frac{\det(I + z(\mathbf{1}\mathbf{b}^T - A))}{\det(I - zA)}. \tag{4.11}$$

Note that this expression assumes that $\mathbf{1}\mathbf{b}^T$ is a diagonal matrix where the values of $\mathbf{b}$ are the elements on the main diagonal. For example if $\mathbf{b} = (b_1, b_2)^T$ then

$$\mathbf{1}\mathbf{b}^T = \begin{pmatrix} b_1 & 0 \\ 0 & b_2 \end{pmatrix}.$$

We can use Eq. (4.11) to test whether an implicit method is A-stable using the following theorem:

Theorem 4.3.1. *Given an implicit Runge-Kutta method with the stability function of the form*

$$R(z) = \frac{N(z)}{D(z)},$$

and define the polynomial $E(y)$ by

$$E(y) = D(iy)D(-iy) - N(iy)N(-iy). \tag{4.12}$$

The Runge-Kutta method is A-stable iff the following are satisfied:

1. *All poles of $R(z)$, that is all roots of $D(z)$, are in the right-hand plane;*
2. *$E(y) \geq 0$ for all $y \in \mathbb{R}$.*

Example 4.3.2. Determine the stability function of the following implicit Runge-Kutta method and test whether it is A-stable or not.

$$\begin{array}{c|cc} \frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Using Eq. (4.11)

$$\begin{aligned} R(z) &= \frac{\det\left[\begin{pmatrix}1&0\\0&1\end{pmatrix} + z\left(\begin{pmatrix}\frac{1}{2}&0\\0&\frac{1}{2}\end{pmatrix} - \begin{pmatrix}\frac{1}{4}&\frac{1}{4}-\frac{\sqrt{3}}{6}\\ \frac{1}{4}+\frac{\sqrt{3}}{6}&\frac{1}{4}\end{pmatrix}\right)\right]}{\det\left[\begin{pmatrix}1&0\\0&1\end{pmatrix} - z\begin{pmatrix}\frac{1}{4}&\frac{1}{4}-\frac{\sqrt{3}}{6}\\ \frac{1}{4}+\frac{\sqrt{3}}{6}&\frac{1}{4}\end{pmatrix}\right]} \\ &= \frac{\det\begin{pmatrix}1+\frac{1}{4}z & z(\frac{\sqrt{3}}{6}-\frac{1}{4}) \\ z(-\frac{1}{4}-\frac{\sqrt{3}}{6}) & 1+\frac{1}{4}z\end{pmatrix}}{\det\begin{pmatrix}1-\frac{1}{4}z & z(\frac{1}{4}-\frac{\sqrt{3}}{6}) \\ z(-\frac{1}{4}-\frac{\sqrt{3}}{6}) & 1-\frac{1}{4}z\end{pmatrix}} \\ &= \frac{(1+\frac{1}{4}z)(1+\frac{1}{4}z) - z^2(-\frac{1}{4}-\frac{\sqrt{3}}{6})(\frac{\sqrt{3}}{6}-\frac{1}{4})}{(1-\frac{1}{4}z)(1-\frac{1}{4}) - z^2(-\frac{1}{4}-\frac{\sqrt{3}}{6})(\frac{1}{4}-\frac{\sqrt{3}}{6})} \\ &= \frac{1+\frac{1}{2}z+\frac{1}{12}z^2}{1-\frac{1}{2}z+\frac{1}{12}z^2} \end{aligned}$$

Here we have

$$\begin{aligned} N(z) &= 1 + \frac{1}{2}z + \frac{1}{12}z^2, \\ D(z) &= 1 - \frac{1}{2}z + \frac{1}{12}z^2, \end{aligned}$$

The roots of $D(z)$ are $3 + i\sqrt{3}$ and $3 - i\sqrt{3}$ therefore all roots of $D(z)$ are in the right-hand plane so the first condition is satisfied. Using Eq. (4.12)

$$\begin{aligned} E(y) &= D(iy)D(-iy) - N(iy)N(-iy) \\ &= \left(1 - \frac{1}{2}iy - \frac{1}{12}y^2\right)\left(1 + \frac{1}{2}iy - \frac{1}{12}y^2\right) \\ &\quad - \left(1 + \frac{1}{2}iy - \frac{1}{12}y^2\right)\left(1 - \frac{1}{2}iy - \frac{1}{12}y^2\right) \\ &= 1 + \frac{1}{2}iy - \frac{1}{12}y^2 - \frac{1}{2}iy + \frac{1}{4}y^2 + \frac{1}{24}iy^3 - \frac{1}{12}y^2 - \frac{1}{24}iy^3 + \frac{1}{144}y^4 \\ &\quad - 1 + \frac{1}{2}iy + \frac{1}{12}y^2 - \frac{1}{2}iy - \frac{1}{4}y^2 + \frac{1}{24}iy^3 + \frac{1}{12}y^2 - \frac{1}{24}iy^3 - \frac{1}{144}y^4 \\ &= 0 \end{aligned}$$

Since $E(y) \geq 0$ and the roots of $D(z)$ are in the right-hand plane then we can say that this method is A-stable. □

4.3.3 Using MATLAB to determine stability of IRK methods

The algebra used in determining the stability function of an IRK and $E(y)$ can be performed using MATLAB. Consider the following IRK method

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

The MATLAB commands to determine $N(z)$, $D(z)$ and hence whether this IRK is A-stable or not are given below.

```
>> syms z y
>> A = [5/12 -1/12 ; 3/4 1/4];
>> b = [1/2 0 ; 0 1/2];
>> N = det(eye(2) + z*(b - A))

N =

z^2/12 + z/3 + 1

>> D = det(eye(2) - z*A)

D =

z^2/6 - (2*z)/3 + 1
```

So $N(z) = \frac{1}{12}z^2 + \frac{1}{3}z + 1$ and $D(z) = \frac{1}{6}z^2 - \frac{2}{3}z + 1$. The roots of $D(z)$ can be determined using the `roots` command

```
>> roots([1/6 -2/3 1])

ans =

   2.0000 + 1.4142i
   2.0000 - 1.4142i
```

So $D(z)$ has roots at $z = 2 \pm \sqrt{2}i$ which are both in the right-hand plane. The function $E(y)$ can be determined using inline functions

```
>> N = @(z) z^2/12 + z/2 + 1;
>> D = @(z) z^2/6 - 2*z/3 + 1;
>> E = D(1i*y)*D(-1i*y) - N(1i*y)*N(-1i*y)

E =

- (- y^2/6 + y*((2*i)/3) + 1)*(y^2/6 + y*((2*i)/3) - 1) + (- y^2/12 + y*(i/2) +
     1)*(y^2/12 + y*(i/2) - 1)

>> simplify(E)

ans =

(y^2*(3*y^2 + 4))/144
```

So $E(y) = \frac{1}{48}y^4 + \frac{1}{36}y^2$, therefore $E(y) \geq 0$ for all y so this IRK is A-stable.

Note that MATLAB is primarily a computational mathematics package as opposed to a symbolic package such as MathCAD and Mathematica. It can perform symbolic maths but may return strange results, especially when dealing with square roots.

4.4 Stability tutorial exercises

1. Determine the stability function of the following Runge-Kutta method

$$\begin{array}{c|ccccc} 0 & & & & & \\ \frac{1}{4} & \frac{1}{4} & & & & \\ \frac{1}{2} & \frac{1}{2} & 0 & & & \\ \frac{3}{4} & 0 & \frac{1}{2} & \frac{1}{4} & & \\ 0 & 0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} & \\ \hline & -1 & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & 1 \end{array}$$

2. Determine the stability function of the following Runge-Kutta method and show that it is A-stable.

$$\begin{array}{c|cc} \frac{1}{4} & \frac{7}{24} & -\frac{1}{24} \\ \frac{3}{4} & \frac{13}{24} & \frac{5}{24} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

3. Plot the region of absolute stability for the following Runge-Kutta method.

$$\begin{array}{c|cc} \frac{1}{3} & \frac{1}{3} & 0 \\ 1 & 1 & 0 \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

4. Determine the stability function of the following Runge-Kutta method.

$$\begin{array}{c|ccccc} 0 & & & & & \\ \frac{2}{7} & \frac{2}{7} & & & & \\ \frac{4}{7} & -\frac{8}{35} & \frac{4}{5} & & & \\ \frac{6}{7} & \frac{29}{42} & -\frac{2}{3} & \frac{5}{6} & & \\ 1 & \frac{1}{6} & \frac{1}{6} & \frac{5}{12} & \frac{1}{4} & \\ \hline & \frac{11}{96} & \frac{7}{24} & \frac{35}{96} & \frac{7}{48} & \frac{1}{12} \end{array}$$

   What is the maximum order of the method?

The solutions to these tutorial exercises are on page 161

Chapter 5

Boundary Value Problems

A *Boundary Value Problem* (BVP) is a type of ODE where the values of the independent variables are specified at the boundaries of the domain in question. Unlike IVPs, the solution to the ODE at the initial values of y', y'' etc. are unknown. BVPs are used in a wide range of different applications for modelling physical and chemical phenomena.

5.1 General two-point Boundary Value Problems

A two-point BVP refers to an ODE where the values of the independent boundaries are specified at two points either end of a one-dimensional domain and can be written in the form

$$y'' = f(t, y, y'), \qquad y(a) = \alpha, \qquad y(b) = \beta,$$

where $a \leq t \leq b$ is the domain of the independent variable, a and b are the boundaries of the domain and α and β are the values of y at these boundaries. The solution to a BVP is the function, y, of the independent variable t that satisfies the ODE (Fig. 5.1).

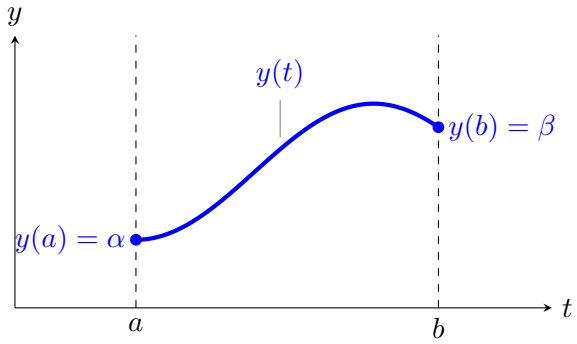


Figure 5.1: The solution to the BVP $y'' = f(t, y, y')$, $y(a) = \alpha$, $y(b) = \beta$.

5.2 Existence and uniqueness

In general, IVPs have a unique solution but this is not always true for BVPs. A BVP may have a unique solution, an infinite number of solutions or not solutions shown by the following examples.

Example 5.2.1. Consider the following BVP:

$$y' + y = 0, \qquad y(0) = 0, \qquad y\left(\frac{\pi}{6}\right) = 1. \tag{5.1}$$

The general solution to this differential equation is

$$y(t) = c_1 \cos(t) + c_2 \sin(t). \tag{5.2}$$

Substituting the boundary conditions gives

$$\begin{aligned} 0 = y(0) &= c_1 \cos(0) + c_2 \sin(0) = c_1, \\ 1 = y\left(\frac{\pi}{6}\right) &= c_1 \cos\left(\frac{\pi}{6}\right) + c_2 \sin\left(\frac{\pi}{6}\right) = \frac{\sqrt{3}}{2} c_1 + \frac{1}{2} c_2 \end{aligned}$$

From the first equation we have $c_1 = 0$ and substituting into the second equation gives

$$1 = \frac{1}{2} c_2$$

so $c_2 = 2$ and the BVP given in Eq. (5.1) has a unique solution which is

$$y(t) = 2 \sin(t)$$

□

Example 5.2.2. Consider the following BVP:

$$y' + y = 0, \qquad y(0) = 0, \qquad y(\pi) = 0. \tag{5.3}$$

Using the general solution given in Eq. (5.2) results in

$$\begin{aligned} 0 = y(0) &= c_1 \cos(0) + c_2 \sin(0) = c_1, \\ 0 = y(\pi) &= c_1 \cos(\pi) + c_2 \sin(\pi) = -c_1. \end{aligned}$$

Here we have no information on the value of c_2, so there are an infinite number of solutions.

$$y(t) = c_2 \sin(t).$$

□

Example 5.2.3. Consider the following BVP:

$$y' + y = 0, \qquad y(0) = 0, \qquad y(2\pi) = 1. \tag{5.4}$$

Using the general solution given in Eq. (5.2) results in

$$\begin{aligned} 0 = y(0) &= c_1 \cos(0) + c_2 \sin(0) = c_1, \\ 1 = y(2\pi) &= c_1 \cos(2\pi) + c_2 \sin(2\pi) = c_1. \end{aligned}$$

Here we have a contradiction since c_1 cannot be equal to both 0 and 1. Therefore there are no solutions that satisfy the BVP given in Eq. (5.4). □

For the general case, the theory of existence and uniqueness for BVPs is quite complicated and its in-depth study is outside of the scope of this unit. However, the following theorem is given here (without proof).

Theorem 5.2.1. *For linear boundary value problems of the general form*

$$y'' = p(t)y' + q(t)y + r(t), \qquad a \le t \le b, \qquad y(a) = \alpha, \qquad y(b) = \beta \tag{5.5}$$

where $y'' = f(t, y, y')$ is a linear function and $p(t)$, $q(t)$ and $r(t)$ are some function of t then the BVP has a unique solution if the following conditions are satisfied:

1. *$p(t)$, $q(t)$ and $r(t)$ are continuous on $[a, b]$;*
2. *$q(t) > 0$ for all $t \in [a, b]$.*

Example 5.2.4. Show that the following BVP has a unique solution

$$y'' = (t^3 + 5)y + \sin(t), \qquad 0 \leq t \leq 1, \qquad y(0) = 0, \qquad y(1) = 1.$$

Comparing this BVP to Eq. (5.5) gives

$$\begin{aligned} p(t) &= 0, \\ q(t) &= t^3 + 5, \\ r(t) &= \sin(t). \end{aligned}$$

Here $p(t)$, $q(t)$ and $r(t)$ are all continuous on $[0, 1]$ and $q(t) \geq 5$ on $[0, 1]$. Therefore this BVP has a unique solution. □

5.3 Shooting methods

The *shooting method* for solving a BVP reduces the problem to an IVP which can then be solved using an ODE solver (Runge-Kutta, Adams etc.). However, only the initial value of the function y is known and we do not know what the initial value of $y'(a)$ which is required to solve the IVP. To overcome this we simply guess the value of $y'(a)$ and solve the IVP using our ODE solvers. The computed solution at the upper boundary $y(b)$ is compared to the known solution β and we adjust our initial guess of $y'(a)$ guess accordingly. This process of adjusting the guess value continues until we have a solution at the upper boundary that is close enough to β, i.e., we've hit the 'target' (Fig. 5.2).

For example, consider the following two-point BVP

$$y'' = f(t, y, y'), \qquad y(a) = \alpha, \qquad y(b) = \beta. \tag{5.6}$$

Here the boundary conditions $y(a) = \alpha$ and $y(b) = \beta$ are known but the initial condition $y'(a)$ (which is needed to solve an IVP) is not given. If $y'(a)$ was known then the BVP could be written as an IVP such that

$$y'' = f(t, y, y'), \qquad y(a) = \alpha, \qquad y'(a) = s,$$

where s is the initial value for $y'(a)$. First we need to reduce this second-order ODE into a system of two first-order ODEs so we can solve it. Let $y_1 = y$ and $y_2 = y'$ then we have

$$\begin{aligned} y_1' &= y_2 & y(a), &= \alpha, \\ y_2' &= f(t, y_1, y_2), & y'(a) &= s. \end{aligned}$$

The solution of y_1 will coincide with the solution of the BVP given in Eq. (5.6) if we can find a value of s where the value of y_1 at the boundary b is equation close to β.

Example 5.3.1. Solve the following BVP using the shooting method (Johnson, 2008)

$$y'' = 6t, \qquad y(0) = 0, \qquad y(1) = 1.$$

First, transform the second-order ODE into a system of two first-order ODEs. Let $y_1 = y$ and $y_2 = y_1'$ then

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 0, \\ y_2' &= 6t, & y_2(0) &= s. \end{aligned}$$

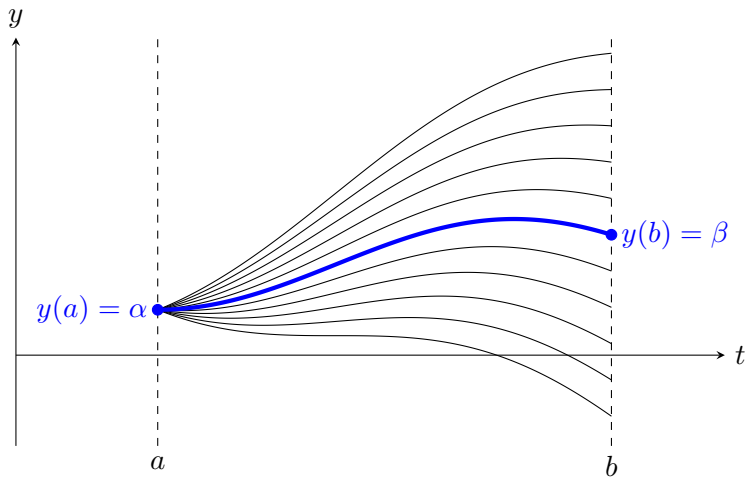


Figure 5.2: The shooting method calculates the solution to an BVP by solving an IVP with guess values for $y'(a)$.

where s is a guess of the value of $y'(0)$. When solving systems of ODEs we write them in vector form such that $\mathbf{y}' = F(t, \mathbf{y})$ where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \qquad F(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ f(t, y_1, y_2) \end{pmatrix}.$$

For this example $y(0) = 0$, $y'(0) = s$ and $f(t, y_1, y_2) = 6t$ so

$$\mathbf{y}_0 = \begin{pmatrix} 0 \\ s \end{pmatrix}, \qquad F(t_0, \mathbf{y}_0) = \begin{pmatrix} s \\ 6t_0 \end{pmatrix}.$$

The classical fourth-order explicit Runge-Kutta method has been chosen to solve this system using a step length of $h = 0.5$. Let's begin with a guess value of $s = 1$, for the first step $t_0 = 0$:

$$\begin{aligned}
\mathbf{k}_1 &= hf(t_0, \mathbf{y}_0) = 0.5 \begin{pmatrix} 1 \\ 6(0) \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0 \end{pmatrix}, \\
\mathbf{k}_2 &= hf(t_0 + \tfrac{1}{2}h, \mathbf{y}_0 + \tfrac{1}{2}\mathbf{k}_1) = 0.5 \begin{pmatrix} 1 + \frac{1}{2} \cdot 0 \\ 6(0 + \frac{1}{2} \cdot 0.5) \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.75 \end{pmatrix}, \\
\mathbf{k}_3 &= hf(t_0 + \tfrac{1}{2}h, \mathbf{y}_0 + \tfrac{1}{2}\mathbf{k}_2) = 0.5 \begin{pmatrix} 1 + \frac{1}{2} \cdot 0.75 \\ 6(0 + \frac{1}{2} \cdot 0.5) \end{pmatrix} = \begin{pmatrix} 0.6875 \\ 0.75 \end{pmatrix}, \\
\mathbf{k}_4 &= hf(t_0 + h, \mathbf{y}_0 + \mathbf{k}_3) = 0.5 \begin{pmatrix} 1 + 0.75 \\ 6(0 + 0.5) \end{pmatrix} = \begin{pmatrix} 0.875 \\ 1.5 \end{pmatrix}, \\
\mathbf{y}_1 &= \mathbf{y}_0 + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \\
&= \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \frac{1}{6}\left[\begin{pmatrix} 0.5 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} 0.5 \\ 0.75 \end{pmatrix} + 2 \begin{pmatrix} 0.6875 \\ 0.75 \end{pmatrix} + \begin{pmatrix} 0.875 \\ 1.5 \end{pmatrix} \right] = \begin{pmatrix} 0.625 \\ 1.75 \end{pmatrix}.
\end{aligned}$$

Now $t_1 = t_0 + h = 0.5$:

$$\mathbf{k}_1 = hf(t_1, \mathbf{y}_1) = 0.5 \begin{pmatrix} 1.75 \\ 6 \cdot 0.5 \end{pmatrix} = \begin{pmatrix} 0.875 \\ 1.5 \end{pmatrix},$$

$$\mathbf{k}_2 = hf(t_1 + \tfrac{1}{2}h, \mathbf{y}_1 + \tfrac{1}{2}\mathbf{k}_1) = 0.5 \begin{pmatrix} 1.75 + \frac{1}{2} \cdot 1.5 \\ 6(0.5 + \frac{1}{2} \cdot 0.5) \end{pmatrix} = \begin{pmatrix} 1.25 \\ 2.25 \end{pmatrix},$$

$$\mathbf{k}_3 = hf(t_1 + \tfrac{1}{2}h, \mathbf{y}_1 + \tfrac{1}{2}\mathbf{k}_2) = 0.5 \begin{pmatrix} 1.75 + \frac{1}{2} \cdot 2.25 \\ 6(0.5 + \frac{1}{2} \cdot 0.5) \end{pmatrix} = \begin{pmatrix} 1.4375 \\ 2.25 \end{pmatrix},$$

$$\mathbf{k}_4 = hf(t_1 + h, \mathbf{y}_1 + \mathbf{k}_4) = 0.5 \begin{pmatrix} 1.75 + 2.25 \\ 6(0.5 + 0.5) \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix},$$

$$\begin{aligned} \mathbf{y}_2 &= \mathbf{y}_1 + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \\ &= \begin{pmatrix} 0.625 \\ 1.75 \end{pmatrix} + \frac{1}{6}\left[\begin{pmatrix} 0.875 \\ 1.5 \end{pmatrix} + 2 \begin{pmatrix} 1.25 \\ 2.25 \end{pmatrix} + 2 \begin{pmatrix} 1.4375 \\ 2.25 \end{pmatrix} + \begin{pmatrix} 2 \\ 3 \end{pmatrix} \right] = \begin{pmatrix} 2 \\ 4 \end{pmatrix}. \end{aligned}$$

Here, the solution at the right-hand boundary is $y(1) = 2$ which is an overshoot of the target value $\beta = 1$. Let's try again but this time using a guess value of $s = -1$ (the calculations of $\mathbf{k}_i$ have been omitted for brevity).

$$\mathbf{y}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \frac{1}{6}\left[\begin{pmatrix} -0.5 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} -0.5 \\ 0.75 \end{pmatrix} + 2 \begin{pmatrix} -0.3125 \\ 0.75 \end{pmatrix} + \begin{pmatrix} -0.125 \\ 1.5 \end{pmatrix} \right] = \begin{pmatrix} -0.375 \\ -0.25 \end{pmatrix},$$

$$\mathbf{y}_2 = \begin{pmatrix} -0.375 \\ -0.25 \end{pmatrix} + \frac{1}{6}\left[\begin{pmatrix} -0.125 \\ 1.5 \end{pmatrix} + 2 \begin{pmatrix} 0.25 \\ 2.25 \end{pmatrix} + 2 \begin{pmatrix} 0.4375 \\ 2.25 \end{pmatrix} + \begin{pmatrix} 1 \\ 3 \end{pmatrix} \right] = \begin{pmatrix} 0 \\ 2 \end{pmatrix}.$$

Now the solution at the right-hand boundary is $y(0) = 0$ which is an undershoot of the target value $\beta = 1$. Let's try once more but this time using a guess value of $s = 0$.

$$\mathbf{y}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \frac{1}{6}\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} 0 \\ 0.75 \end{pmatrix} + 2 \begin{pmatrix} 0.1875 \\ 0.75 \end{pmatrix} + \begin{pmatrix} 0.375 \\ 1.5 \end{pmatrix} \right] = \begin{pmatrix} 0.125 \\ 0.75 \end{pmatrix},$$

$$\mathbf{y}2 = \begin{pmatrix} 0.125 \\ 0.75 \end{pmatrix} + \frac{1}{6}\left[\begin{pmatrix} 0.375 \\ 1.5 \end{pmatrix} + 2 \begin{pmatrix} 0.75 \\ 2.25 \end{pmatrix} + 2 \begin{pmatrix} 0.9375 \\ 2.25 \end{pmatrix} + \begin{pmatrix} 1.5 \\ 3 \end{pmatrix} \right] = \begin{pmatrix} 1 \\ 3 \end{pmatrix}.$$

Now the solution at the right-hand boundary is $y(0) = 1 = \beta$. Therefore the solution to the BVP using the Euler method with step length $h = 0.5$ is

$$\begin{array}{lll} t_0 = 0, & t_1 = 0.5, & t_2 = 1, \\ y_0 = 0, & y_1 = 0.125, & y_2 = 1. \end{array}$$

The solutions to this BVP using the three guess values $s = 1$, $s = -1$ and $s = 0$ have been plotted in Fig. 5.3.

□

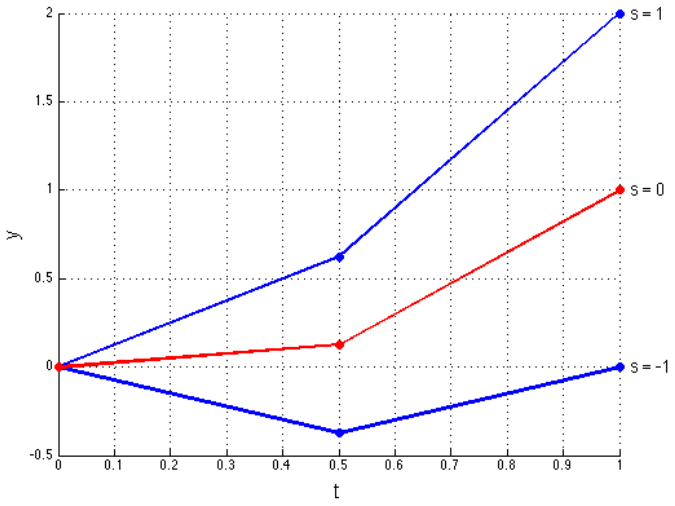


Figure 5.3: Solutions of the BVP $y'' = 6t$, $y(0) = 0$, $y(1) = 1$, using the shooting method.

Example 5.3.2. A firework is launched from a launch tube and is set to explode after 5 seconds. For the maximum effect, the firework needs to explode when it is 40 metres above the ground. What is the launch velocity of the firework? (Heckbert, 2000)

This problem can be formulated as the following BVP

$$y'' = -g, \qquad y(0) = 0, \qquad y(5) = 40,$$

where $g = 9.81\text{ms}^{-2}$ is the acceleration due to gravity. Note that since we require the launch velocity, it is that value of $y'(0)$ that we are looking for (velocity is the change in distance with respect to time). Let $y_1 = y$ and $y_2 = y_1'$ then we have the equivalent IVP

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 0, \\ y_2' &= -g, & y_2(0) &= s. \end{aligned}$$

The solutions calculated using the Euler method to solve the ODEs with a step length of $h = 1$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + hF(t_n, \mathbf{y}_n),$$

where

$$\mathbf{y}_0 = \begin{pmatrix} 0 \\ s \end{pmatrix}, \qquad F(t, \mathbf{y}) = \begin{pmatrix} y_2 \\ -g \end{pmatrix}.$$

The MATLAB program in Listing 5.1 below calculates the solutions using guess values $s = 20$, 30, 40 and 50. This uses the function to calculate the Euler method solution to an IVP shown in Listing 1.1 on page 9.

Listing 5.1: MATLAB program to solve a BVP using the shooting method.

```
% bvp_ex1.m by Jon Shiach
%
% This program solves the following BVP using the shooting method
%
%       y'' = -g, y(0)=0, y(1)=1

% clear workspaces
clear
clc
```

```
% solution variables
a = 0;                          % lower boundary
b = 5;                          % upper boundary
ya = 0;                         % lower boundary value
yb = 40;                        % upper boundary value
h = 1;                          % step length
s = [20 30 40 50];              % vector of guess values for y'(a)

% define ODE function
f = @(t,y) [y(2) ; -9.81];

% solve BVP using different guess values
for i = 1 : length(s)

    % initialise y
    y0 = [ ya ; s(i) ];

    % solve ODE using Euler method
    [t,y] = euler(f, a, b, y0, h);

    % plot current solution
    plot(t, y(1,:), 'bo-', 'markerfacecolor', 'b', 'linewidth', 2)
    text(b+0.05, y(1,end), sprintf('s_%1i = %1.0f',i,s(i)), 'fontsize', 14)
    hold on
end

% format plot
plot(b, yb, 'ro', 'markerfacecolor','r')   % plot upper boundary value
hold off
text(b+0.05, yb, sprintf('y_b = %1.0f', yb), 'fontsize', 14)
xlabel('t (s)', 'fontsize',16)
ylabel('y (m)', 'fontsize',16)
grid on
shg
```

A plot of the solutions for each of these guess values is shown in Fig. 5.4. The values of $y(5)$ calculated using these guess values were $y(5, 20) = 1.9$, $y(5, 30) = 51.9$, $y(5, 40) = 101.9$ and $y(5, 50) = 151.9$. This means the actual value of s will be in the range $20 < s < 30$.

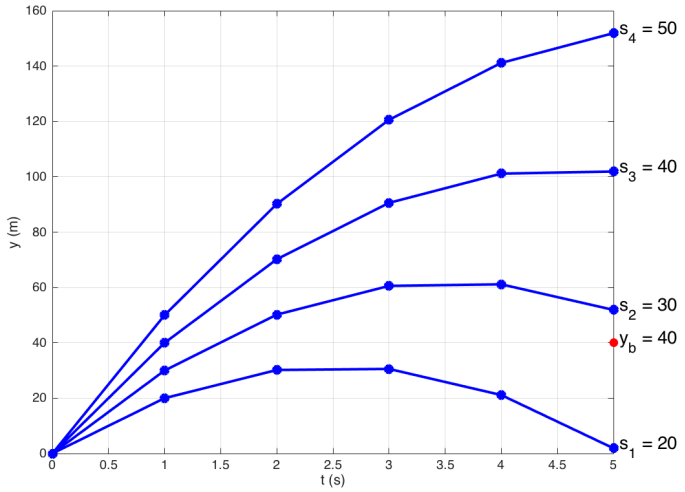


Figure 5.4: Solutions of the BVP $y'' = -g$, $y(0) = 0$, $y(5) = 40$ using Euler method with step length $h = 1$.

□

5.3.1 The secant method

We have seen that the shooting method uses a guess for the value of $y'(a)$ and invokes an ODE solver to solve an IVP. In order to solve the BVP accurately, a method is needed to improve on the guess value. Perhaps that most obvious method to improve the guess value would be to choose a point halfway between two guesses that are known to be under and over estimates (known as the *bisection method*). This will eventually lead to an optimal value of the guess value for the solution method, however, there is a more efficient method known as the *secant method*.

The secant method is a numerical method that is used to calculate the root of function. To use this in order to improve on the guess values in the shooting method, we need to write an expression for the error between the upper boundary value β and the computed upper boundary value for a given guess of $y'(a)$. When this error is zero we have the optimal guess value. Let $g(s)$ be a function defined as the difference between the value at the boundary b calculated using a guess value of s, $y_{computed}(b, s)$, and the actual boundary condition $y(b) = \beta$, i.e.,

$$g(s) = y_{computed}(b, s) - \beta.$$

The value of s that gives $y_{computed}(b, s) = \beta$ is the root of the function $g(s)$, therefore we can use the secant root finding method to improve on the guess value. If s_i and s_{i-1} are the last two guess values, then the next guess value given by the secant method is

$$s_{i+1} = s_i - g(s_i)\frac{s_i - s_{i-1}}{g(s_i) - g(s_{i-1})}. \tag{5.7}$$

The BVP is solved using these guess values until two successive values of s agree to a given tolerance (see Algorithm 3).

Algorithm 3 Shooting method for solving a two-point BVP.

Require: A BVP expressed as a system of two first-order ODEs of the form $\mathbf{y}' = F(t, \mathbf{y})$, $a \le t \le b$, $y(a) = \alpha$, $y(b) = \beta$, step length h and two guess values s_1 and s_2.
  $err \leftarrow 1$ ▷ initialise error variable
  $i \leftarrow 1$ ▷ initialise count variable
  while $err > tol$ **do**
    $\mathbf{y}_0 \leftarrow (\alpha, s_i)^T$ ▷ initialise $\mathbf{y}_0$ using guess value s_i
    Invoke IVP solver on $\mathbf{y}_0$ using step length h to return t and $Y = (\mathbf{y}_1, \mathbf{y}_2)^T$
    $g_i \leftarrow Y_{1,N} - \beta$
    if $i > 1$ **then**
      $s_{i+1} \leftarrow s_i - g_i \left(\dfrac{s_i - s_{i-1}}{g_i - g_{i-1}} \right)$ ▷ use secant method to calculate improved guess value
    end if
    $err \leftarrow |s_{i+1} - s_i|$
    $i \leftarrow i + 1$
  end while
  return t and Y

Example 5.3.3. Solve the BVP given in example 5.3.2 using the secant method to calculate the values of s.

We need the solutions to the ODEs using two values of s. Let's choose $s_1 = 20$ and $s_2 = 40$ as the initial guess values. Invoking the Euler method with $h = 1$ for gives $y_1(5, 20) = 1.9$ and

$y_1(5, 40) = 101.9$. Using the secant method Eq. (5.7) to calculate the next value gives

$$\begin{aligned} g(s_1) &= y_{computed}(5, 20) - \beta = 1.9 - 40 = -38.1, \\ g(s_2) &= y_{computed}(5, 40) - \beta = 101.9 - 40 = 61.9, \\ s_3 &= s_2 - g(s_2)\frac{s_2 - s_1}{g(s_2) - g(s_1)} = 40 - 61.9\left(\frac{40 - 20}{61.9 + 38.1}\right) = 27.62. \end{aligned}$$

Invoking the Euler method again to calculate the solution for $s_3 = 27.62$ gives $y_1(5, 27.62) = 40$. Therefore, using the secant method to calculate the guess value only required three guesses to solve the BVP.

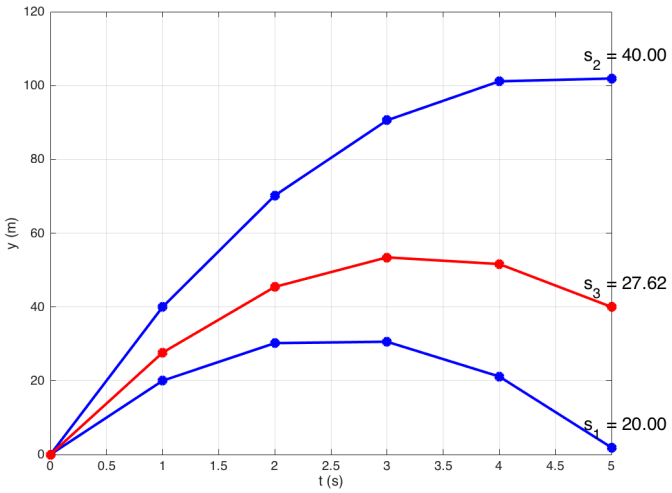


Figure 5.5: Solutions of the BVP $y'' = -g$, $y(0) = 0$, $y(5) = 40$ using the Euler method with step length $h = 1$.

The analytical solution for this BVP is $y = 32.525t - \frac{1}{2}gt^2$ so the initial launch velocity of the rocket is $y'(0) = 32.53$. We can improve on the accuracy of our solution by reducing the step length used in the Euler method. The plot in Fig. 5.6 shows the solution to the BVP using the Euler method with a step length of $h = 0.1$ and using the secant method to calculate the guess values. Here we have a solution of $y'(0) = 32.04$ so it is the lower order accuracy of the Euler method that is causing this difference between the computed solution and the analytical solution. Solving this BVP using the fourth-order Runge-Kutta method with a step length $h = 1$ gives $y'(0) = 32.525$.

□

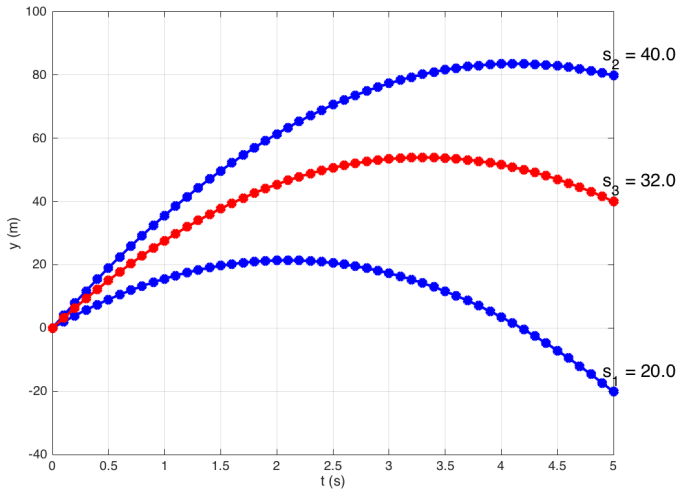


Figure 5.6: Solutions of the BVP $y'' = -g$, $y(0) = 0$, $y(5) = 40$ using the Euler method with step length $h = 0.1$.

5.4 Finite-difference methods

The finite-difference method for solving BVPs converts the BVP into a system of algebraic equations by replacing the derivatives with finite-difference approximations derived from the Taylor series.

The finite-difference method solves the dependent variable, y, at various discrete points in the domain. These points are known as *finite-difference nodes*. For simplicity, it is common to locate the finite-difference nodes at equally spaced intervals across the domain creating what is known as a *uniform mesh grid*. Consider Fig. 5.7 where the domain $t \in [a, b]$ has been discretised using a uniform mesh grid. The node at the lower boundary is labelled node 0 and the node at the upper boundary is labelled node N, therefore there are $N + 1$ nodes used in this mesh.

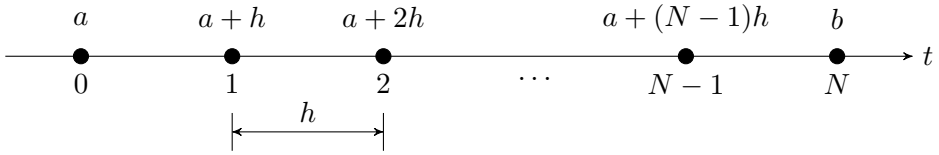


Figure 5.7: Discretising the domain using a uniform mesh.

If h is the distance between two neighbouring nodes, then the value of the ith node in the uniform mesh grid can be calculated using

$$t_i = a + ih, \qquad i = 0, \ldots, N. \tag{5.8}$$

Since we want $a + Nh = b$ then the step length h is related to N by

$$h = \frac{b - a}{N}, \tag{5.9}$$

or alternatively

$$N = \frac{b - a}{h}. \tag{5.10}$$

5.4.1 Derivation of finite-difference approximations

Consider the Taylor series (the derivation of which is given in Section 1.3)

$$y(t + h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \frac{^2}{3!}y'''(t) + \ldots + \frac{h^n}{n!}y^{(n)}(t). \tag{5.11}$$

Using subscript notation where $y_{i+1} = y(t + h)$ and $y_i = y(t)$ this can be written as

$$y_{i+1} = y_i + hy'_i + \frac{h^2}{2!}y''_i + \frac{h^3}{3!}y'''_i + \ldots + \frac{h^n}{n!}y^{(n)}_i. \tag{5.12}$$

Replacing h by $-h$ in Eq. (5.11) and using subscript notation gives

$$y_{i-1} = y_i - hy'_i + \frac{h^2}{2!}y''_i - \frac{h^3}{3!}y'''_i + \ldots + (-1)^n\frac{h^n}{n!}y^{(n)}_i. \tag{5.13}$$

We can use Eqs. (5.12) and (5.13) to derive approximations of derivatives. For example, truncating Eq. (5.12) so that all derivatives second-order or higher are omitted gives

$$y_{i+1} = y_i + hy'_i + O(h^2),$$

where $O(h^2)$ represents the truncation error attributed to the terms that have been rejected. Rearranging to make y_i' the subject gives (note that $O(h^n)/h^m = O(h^{n-m})$).

$$y_i' = \frac{y_{i+1} - y_i}{h} + O(h). \tag{5.14}$$

This is known as the *forward* difference approximation of a first-order derivative. Alternatively, doing similar with Eq. (5.13)

$$y_i' = \frac{y_i - y_{i-1}}{h} + O(h), \tag{5.15}$$

which is the *backward* difference approximation of a first-order derivative.

Equations (5.14) and (5.15) are both first-order accurate finite-difference approximations since only first order derivatives are included in the derivation. Obviously, a more accurate approximation would include the second-order derivatives. Subtracting Eq. (5.13) from Eq. (5.12) and truncating to second-order gives

$$y_{i+1} - y_{i-1} = 2hy_i' + O(h^3),$$

therefore

$$y_i' = \frac{y_{i+1} - y_{i-1}}{2h} + O(h^2). \tag{5.16}$$

This is known as the *central* difference approximation of a first-order derivative. The central difference approximation of a second-order derivative can be derived by summing Eqs. (5.12) and (5.13) and truncating to third-order

$$y_{i+1} + y_{i-1} = 2y_i + h^2 y_i'' + O(h^4),$$

therefore

$$y_i'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + O(h^2). \tag{5.17}$$

This is known as the second-order symmetric difference approximation of a second-order derivative.

5.4.2 The accuracy of a finite-difference approximation

The error attributed to the truncation of the higher order terms in the Taylor series when deriving a finite-difference approximation is given as a function of the step length h. For example, the central difference formula in Eq. (5.16) has an error term $O(h^2)$. This means that the error between the value of the derivative y' calculated using this finite-difference approximation and the exact value is proportional to h^2 (see Section 1.3.1). Therefore the condition $h < 1$ applies when using finite-differences.

The exponent of the step length h in the error term is known as the *order* of the finite-difference approximation. For example, Eq. (5.16) is a second-order finite-difference approximation since the exponent of h is 2. Note that care must be taken not to confuse the order of a finite-difference approximation with the order of a derivative. For example, Eq. (5.16) is a second-order accurate finite-difference approximation of a first-order derivative, y'.

5.4.3 Using finite-difference methods to solve BVP

To solve a BVP using finite-difference methods we first need to discretise the independent variable across the domain using Eq. (5.8). The derivatives in the ODE are replaced by finite-difference approximations and rearranged into a linear system of equations. The linear system is solved to give the solution to the BVP.

For example, consider the general two-point BVP below

$$y'' = f(t, y), \qquad y(a) = \alpha, \qquad y(b) = \beta.$$

The independent variable t is discretised across the interior of the domain using N finite-difference nodes, therefore

$$h = \frac{b-a}{N},$$

and $t_i = a + ih$ for $i = 0, \ldots, N$. The derivatives in the ODE are replaced by finite-difference approximations and factorised in terms of the y terms, so in case we have

$$\begin{aligned} \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} &= f(t_i, y_i) \\ y_{i-1} - 2y_i + y_{i+1} &= h^2 f(t_i, y_i), \qquad i = 1, \ldots, N. \end{aligned}$$

Writing this system out in full

$$\begin{aligned} i = 0 : &\qquad y_0 = \alpha, \\ i = 1 : &\qquad y_0 - 2y_2 + y_3 = h^2 f(t_1, y_1), \\ i = 2 : &\qquad y_1 - 2y_3 + y_4 = h^2 f(t_2, y_2), \\ &\vdots \qquad\qquad \vdots \\ i = N-1 : &\qquad y_{N-2} - 2y_{N-1} + y_N = h^2 f(t_{N-1}, y_{N-1}), \\ i = N : &\qquad y_N = \beta. \end{aligned}$$

The values of the first and last nodes, $i = 0$ and $i = N$, are known for a BVP (α and β) so we can reduce the number of equations in the system from $N + 1$ to $N - 1$.

$$\begin{aligned} i = 1 : &\qquad -2y_1 + y_2 = f(t_1, y_1) - \alpha, \\ i = 2 : &\qquad y_1 - 2y_2 + y_3 = f(t_2, y_2), \\ i = 3 : &\qquad y_2 - 2y_3 + y_4 = f(t_3, y_3), \\ &\vdots \\ i = N-2 : &\qquad y_{N-3} - 2y_{N-2} + y_{N-1} = f(t_{N-2}, y_{N-2}), \\ i = N-1 : &\qquad y_{N-2} - 2y_{N-1} = f(t_{N-1}, y_{N-1}) - \beta. \end{aligned}$$

Expressing this system in matrix form gives

$$\begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} f(t_1, y_1) - \alpha \\ f(t_2, y_2) \\ f(t_3, y_3) \\ \vdots \\ f(t_{N-2}, y_{N-2}) \\ f(t_{N-1}, y_{N-1}) - \beta \end{pmatrix}.$$

This is a nonsingular tridiagonal system that can be solved to give the solutions of the BVP at the finite-difference nodes.

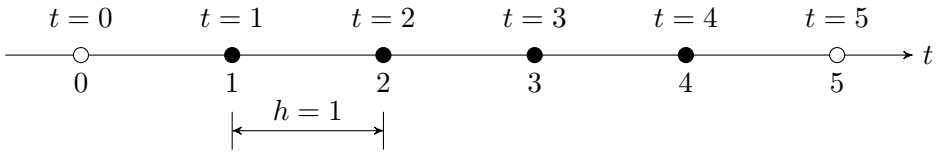
Example 5.4.1. Solve the following BVP using the finite-difference method with 6 nodes.

$$y'' = -g, \qquad y(0) = 0, \qquad y(5) = 40.$$

First we discretise the independent variable. Using Eq. (5.9) with $a = 0$, $b = 5$ and $N = 6-1 = 5$ (N is always 1 less than the number of nodes) the step length is

$$h = \frac{b-a}{N} = \frac{5-0}{5} = 1.$$

So the finite-difference mesh looks like



and the finite-difference nodes are at $\mathbf{t} = (0, 1, 2, 3, 4, 5)$. The nodes $t_i = 1, \ldots, N-1$ are the interior nodes that we are going to calculate the value of the solution. Next we replace the second-order derivative with the second-order central difference to give

$$y_{i-1} - 2y_i + y_{i+1} = -gh^2.$$

Substituting values for i into this expression provides us with the following linear system

$$\begin{aligned} y_0 - 2y_1 + y_2 &= -gh^2, \\ y_1 - 2y_2 + y_3 &= -gh^2, \\ y_2 - 2y_3 + y_4 &= -gh^2, \\ y_3 - 2y_4 + y_5 &= -gh^2. \end{aligned}$$

This is a system of 4 equations in 6 unknowns. Since the values at the the boundary nodes are known ($y_0 = 0$ and $y_5 = 40$) we can reduce this to the following 4 equations in 4 unknowns

$$\begin{aligned} -2y_1 + y_2 &= -g, \\ y_1 - 2y_2 + y_3 &= -g, \\ y_2 - 2y_3 + y_4 &= -g, \\ y_3 - 2y_4 &= -g - 40. \end{aligned}$$

or in matrix form

$$\begin{pmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 0 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} -g \\ -g \\ -g \\ -g - 40 \end{pmatrix}.$$

Solving this system gives $y_1 = 27.62$, $y_2 = 45.43$, $y_3 = 53.43$ and $y_4 = 51.62$. The computed solutions have been compared to the analytical solution $y(t) = 32.525t - \frac{1}{2}gt^2$ in Fig. 5.8. The computed solutions show good agreement with the analytical solutions at the finite-difference nodes, however, linear interpolation between nodes deviates from the analytical solution. Using more finite-difference nodes would eleviate this, but of course at an additional computational cost.

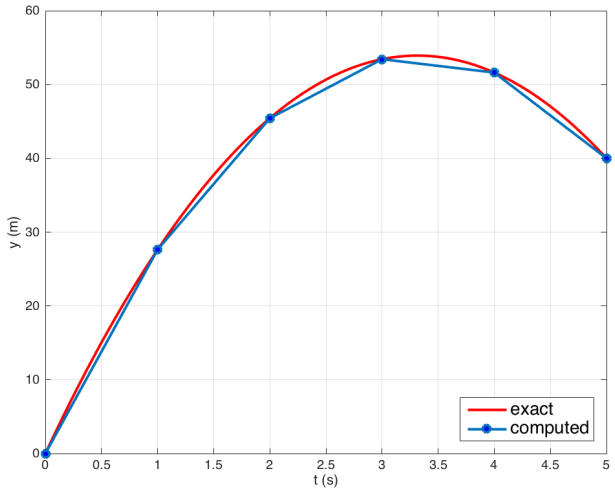


Figure 5.8: Solutions to the BVP $y'' = -g$, $y(0) = 0$, $y(5) = 40$ using the second-order finite-difference method with 4 nodes.

□

5.4.4 MATLAB code

The MATLAB code used to compute the solution shown in Fig. 5.8 is given in Listing 5.2. Here the MATLAB command `A\d` has been used to solve the linear system $A\mathbf{y} = \mathbf{d}$.

Listing 5.2: MATLAB code for solving the BVP given in Example 5.4.1.

```
% bvp_ex4.m by Jon Shiach
%
% This program solves the following BVP using a first-order
% finite-difference method.
%
%           y'' = -g, y(0) = 0, y(5) = 40

% clear workspaces
clear
clc

% solution variables
N = 5;                          % no. finite-difference nodes - 1
a = 0;                          % lower boundary
b = 5;                          % upper boundary
ya = 0;                         % lower boundary value
yb = 40;                        % upper boundary value
g = 9.81;                       % acceleration due to gravity

% calculate t array
h = (b - a)/N;
t = [a : h : b]';

% calculate the coefficient matrix A and RHS vector d
for i = 1 : N - 1
    A(i,i) = -2;                % main diagonal
    if i > 1
        A(i,i-1) = 1;           % lower diagonal
    end
    if i < N - 1
        A(i,i+1) = 1;           % upper diagonal
```

```
    end
    d(i,1) = -g;            % RHS vector
end
d(1) = d(1) - ya;
d(N-1) = d(N-1) - yb;

% solve linear system and append boundary values
y = [ ya ; A\d; yb];

% calculate exact solution
t_ex = a : 0.01 : b;
y_ex = 32.525*t_ex - 0.5*g*t_ex.^2;

% plot solution
plot(t_ex, y_ex, 'r-', 'linewidth', 2)
hold on
plot(t, y, 'o-', 'markerfacecolor', 'b', 'linewidth', 2)
hold off
xlabel('t (s)', 'fontsize', 16)
ylabel('y (m)', 'fontsize', 16)
mylegend = legend('exact', 'computed')
set(mylegend, 'fontsize', 14, 'location', 'southeast')
grid on
box on
shg
```

Example 5.4.2 (Worked example). Solve the following BVP using the finite-difference method with 6 nodes

$$y'' + \frac{1}{1+t^2}y = 7t, \qquad y(0) = 0, \qquad y(1) = 2.$$

□

5.4.5 Order accuracy versus number of nodes

We have seen that the order accuracy of a finite difference method is given as a function of the step length, i.e., for an nth order method we have $O(h^n)$. Where multiple finite-difference approximations of different orders are used to approximate derivatives, it is the error of the lowest order finite-difference that will dominate. Therefore we say that a finite-difference method is nth order where n is the order of the lower finite-difference approximation.

The higher the order of a finite-difference method the more accurate it is, in addition, the smaller the value of h, the more accurate the approximate but more nodes are required to discretise the domain. It is advisable to use a higher order finite-difference approximation where possible but the same accuracy of a higher order method can be achieved by a lower order method using more finite-difference nodes.

Example 5.4.3. Solve the following BVP using a first-order finite-difference method:

$$y'' + 3ty' + 7y = \cos(2t), \qquad y(0) = 1, \qquad y(\pi) = 0.$$

Calculate the solution using $N = 50$ and $N = 500$.

Replace the y' with a first-order forward difference and y'' with a second-order central difference.

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + 3t_i \frac{y_{i+1} - y_i}{h} + 7y_i = \cos(2t_i).$$

Factorising the y_i terms gives

$$y_{i-1} + (7h^2 - 3ht_i - 2)y_i + (1 + 3ht_i)y_{i+1} = h^2 \cos(2t_i).$$

For the nodes adjacent to the boundaries ($i = 1$ and $i = N - 1$)

$$\begin{aligned}(7h^2 - 3ht_1 - 2)y_1 + (1 + 3ht_1)y_2 &= h^2 \cos(2t_1) - y(0),\\ y_{N-2} + (7h^2 - 3ht_{N-1} - 2)y_{N-1} &= h^2 \cos(2t_{N-1}) - (1 + 3ht_N)y(\pi).\end{aligned}$$

Let $a_{ii} = 7h^2 - 3ht_i - 2$, $a_{i,i+1} = 1 + 3ht_i$ and $d_i = h^2 \cos(2t_i)$ then this linear system can be written as the matrix equation

$$\begin{pmatrix} a_{11} & a_{12} & & & & \\ 1 & a_{22} & a_{23} & & & \\ & 1 & a_{33} & a_{34} & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & a_{N-2,N-2} & a_{N-2,N-1} \\ & & & & 1 & a_{N-1,N-1} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} d_1 - y(0) \\ d_2 \\ d_3 \\ \vdots \\ d_{N-2} \\ d_{N-1} - (1 + 3ht_N)y(\pi) \end{pmatrix}.$$

The MATLAB code in Listing 5.3 is used to perform the calculations. Note that since arrays in MATLAB start at an index of 1, the indices for t need to be incremented by 1 when calculating the coefficient matrix and right-hand side vector.

Listing 5.3: MATLAB code for solving the BVP given in Example 5.4.3.

```
% bvp_ex5.m by Jon Shiach
%
% This program solves the following BVP using a first-order
% finite-difference method.
%
%           y'' + 3xy' + 7y = cos(2x),  y(0) = 1,  y(pi) = 0

% clear workspaces
clear
```

```
clc

% solution variables
N = 50;                         % no. finite-difference nodes - 1
a = 0;                          % lower boundary
b = pi;                         % upper boundary
ya = 1;                         % lower boundary value
yb = 0;                         % upper boundary value

% calculate t array
h = (b - a)/N;
t = [a : h : b]';

% calculate the coefficient matrix A and RHS vector d
for i = 1 : N - 1
    A(i, i) = 7*h^2 - 3*h*t(i+1) - 2;         % main diagonal
    if i > 1
        A(i, i-1) = 1;                        % lower diagonal
    end
    if i < N - 1
        A(i, i+1) = 1 + 3*h*t(i+1);           % upper diagonal
    end
    d(i, 1) = h^2*cos(2*t(i+1));              % RHS vector
end
d(1) = d(1) - ya;
d(N-1) = d(N-1) - (1 + 3*h*t(N))*yb;

% solve linear system and append boundary values
y = [ya ; A\d ; yb];

% plot solution
plot(t, y, 'linewidth', 2)
axis([ a b -2 1])
xlabel('t', 'fontsize', 16)
ylabel('y', 'fontsize', 16)
grid on
box on
```

The solutions calculated using 51 and 501 nodes are plotted in Fig. 5.9(a). Note that there is a significant difference between the two solutions. Since a first-order method has been used the error term is $O(h)$ which means that we have linear convergence to the exact solution as the step length decreases. Therefore the solution using 501 nodes is much more accurate than the solution using just 51 nodes.

□

Example 5.4.4. Solve the following BVP using a second-order finite-difference method:

$$y'' + 3ty' + 7y = \cos(2t), \qquad y(0) = 1, \qquad y(\pi) = 0.$$

Calculate the solution using 51 and 501 nodes.

Replace the y' with a first-order forward difference and y'' with a second-order central difference.

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + 3t_i \frac{y_{i+1} - y_{i-1}}{2h} + 7y_i = \cos(2t_i).$$

Factorising the y_i terms gives

$$\left(1 - \frac{3}{2}ht_i\right) y_{i-1} + (7h^2 - 2)y_i + \left(1 + \frac{3}{2}ht_i\right) y_{i+1} = h^2 \cos(2t_i).$$

and for the nodes adjacent to the boundaries ($i = 1$ and $i = N - 1$)

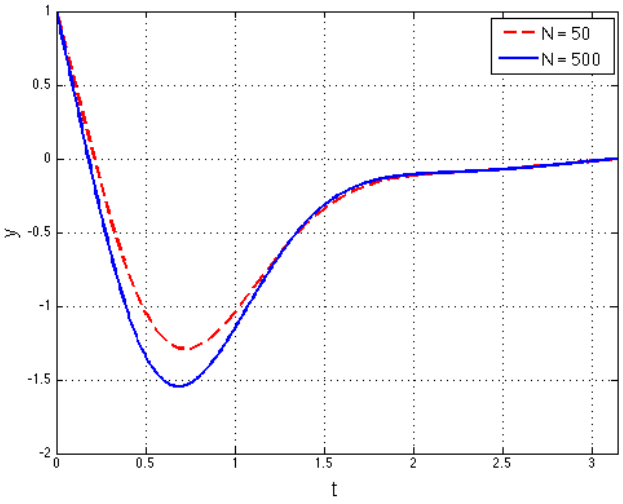
$$(7h^2 - 2)y_1 + \left(1 + \frac{3}{2}ht_1\right) y_2 = h^2 \cos(2t_1) - \left(1 - \frac{3}{2}ht_0\right) y(0),$$
$$\left(1 - \frac{3}{2}ht_{N-1}\right) y_{N-2} + (7h^2 - 2)y_{N-1} = h^2 \cos(2t_{N-1}) - \left(1 + \frac{3}{2}ht_N\right) y(\pi).$$

Let $a_{i,i-1} = 1 - \frac{3}{2}ht_i$, $a_{ii} = 7h^2 - 2$, $a_{i,i+1} = 1 + \frac{3}{2}ht_i$ and $d_i = h^2 \cos(2t_i)$ the matrix equation is

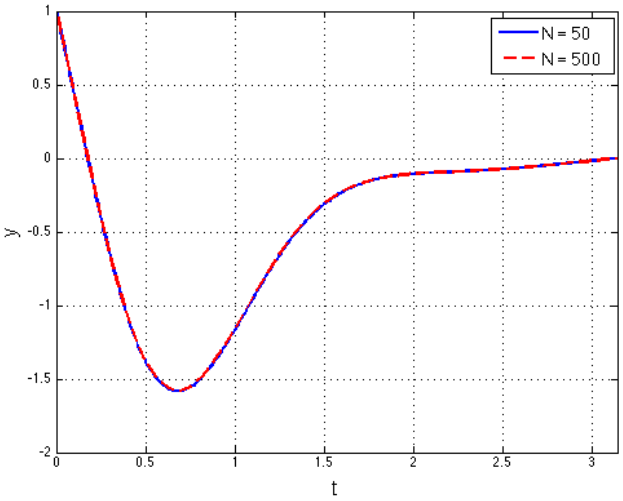
$$\begin{pmatrix} a_{11} & a_{12} & & & & & \\ a_{21} & a_{22} & a_{23} & & & & \\ & a_{32} & a_{33} & a_{34} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & a_{N-2,N-3} & a_{N-2,N-2} & a_{N-2,N-1} \\ & & & & a_{N-1,N-2} & a_{N-1,N-1} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} d_1 - y(0) \\ d_2 \\ d_3 \\ \vdots \\ d_{N-2} \\ d_{N-1} - \left(1 + \frac{3}{2}h\pi\right) y(\pi) \end{pmatrix}.$$

The solutions calculated using 51 and 501 nodes are plotted in Fig. 5.9(b). Unlike the first-order solution, there is little difference between the solutions when using 51 and 501 nodes and using 10 times the number of nodes does not significantly increase the accuracy of the solution. □

It is important to mention that when using a much larger number of mesh points, the coefficient matrix resulting from the finite-difference method treatment of BVPs will be large and sparse (with many non-zero elements). One of the biggest tasks is the solution of the resulting linear system of equations. Most solver packages implement sophisticated sparse solution methods for the treatment of these matrices.



(a) First-order



(b) Second-order

Figure 5.9: Solutions of the BVP $y'' + 3ty' + 7y = \cos(2t)$, $y(0) = 1$ and $y(\pi) = 0$ using first and second order finite-difference methods.

5.5 BVP tutorial exercises

1. Which of the following BVPs have a unique solution?

   (a)
   $$y'' = -\frac{4}{t}y' + \frac{2}{t^2}y - \frac{2\ln(t)}{t^3}, \qquad 1 \leq t \leq 2, \qquad y(1) = \frac{1}{2}, \qquad y(2) = \ln(2)$$

   (b)
   $$y'' = e^t + y\cos(t) - (t+1)y', \qquad 0 \leq t \leq 2, \qquad y(0) = 1, \qquad y(2) = \ln(3)$$

   (c)
   $$y'' = (t^3 + 5)y + sin(t), \qquad 0 \leq t \leq 1, \qquad y(0) = 0, \qquad y(1) = 1$$

   (d)
   $$y'' = [5y + \sin(3t)]e^t, \qquad 0 \leq t \leq 1, \qquad y(0) = 0, \qquad y(1) = 0.$$

2. Consider the following BVP
   $$y'' = 2t, \qquad 0 \leq t \leq 2, \qquad y(0) = 1, \qquad y(2) = 3.$$
   Using the shooting method and Euler's method, calculate the solution for $h = 0.5$ using guess values of:

   (a) $s = 1$;

   (b) $s = -1$;

3. Use the secant method to calculate the next value of s for your solutions to the BVP in question 2. Calculate the solutions using this value of s and compare them to the exact solution of $y = \frac{1}{3}t^3 - \frac{1}{3}t + 1$.

4. Use the finite-difference method with step length $h = 0.5$ to solve the BVP in question 2.

The solutions to these tutorial exercises are given on page 162

Chapter 6

Linear Systems Preliminaries

The methods used to solve linear systems rely heavily on matrices and the properties of matrices. This chapter will cover some of the basics of linear algebra and matrices and acts as a revision for previous studies in the subject.

6.1 Linear systems

A *linear system* is a set of linear equations that are related, for example,

$$\begin{array}{ccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1n}x_n & = & b_1, \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + & \cdots & + & a_{2n}x_n & = & b_2, \\ a_{31}x_1 & + & a_{32}x_2 & + & a_{33}x_3 & + & \cdots & + & a_{3n}x_n & = & b_3, \\ \vdots & & \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & a_{m3}x_3 & + & \cdots & + & a_{mn}x_n & = & b_n. \end{array} \tag{6.1}$$

In this system there are n unknown *variables* x_j where $j = 1, \ldots, n$ related by m equations such that the value of x_j is shared by each of the m equations. Each unknown variable x_j is multiplied by a *coefficient* a_{ij} where $i = 1, \ldots, m$. A linear combination of the coefficients and variables is equal to a known quantity b_i.

The solution of a system of linear equations is the values of the variables x_j that satisfy all of the equations in the system. A solution to a linear system can behave in one of three ways:

i. If the number of equations m is less than the number of unknowns n then the system is said to be *underdetermined* and there can be an infinite number of solutions.

ii. If the number of equations m is equal to the number of unknowns n then if a solution exists it is unique.

iii. If the number of equations m is more than the number of unknowns n then the system is said to be *overdetermined*. If none of the equations are a linear combination of another then an overdetermined system has no solution.

Note that in case ii above, even if $m = n$ there may not be a solution. For example, consider the system of two equations and two unknowns below

$$\begin{array}{ccccc} x_1 & + & 2x_2 & = & 3, \\ x_1 & + & 2x_2 & = & 4. \end{array}$$

Here there is a clear contradiction between the two equations. Attempting to solve this system by subtracting the first equation from the second gives $0 = 1$. A linear system is said to be *consistent* if it has a solution and *inconsistent* if it does not.

6.1.1 Linear systems written in matrix form

The linear system like the one seen in Eq. (6.1) can be written as a *matrix equation* of the form

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

where A is the *coefficient matrix* containing the coefficients a_{ij}, $\mathbf{x}$ is an $n \times 1$ vector containing the unknown variables x_j and $\mathbf{b}$ is the $n \times 1$ *right-hand side vector* containing the known values b_i, i.e.,

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

6.1.2 Solving linear systems using the matrix inverse

Given a linear system of the form $A\mathbf{x} = \mathbf{b}$ then if A has an inverse, we can premultiply both sides by A^{-1} to give

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

since $A^{-1}A = I$ then

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

Note that the calculation of the matrix inverse is not a trivial calculation for matrices where $n \geq 3$ so solving linear systems in this way is only really applicable to smaller systems.

Example 6.1.1. Solve the following linear system using the inverse of the coefficient matrix

$$\begin{aligned} x_1 \quad\quad\quad + 3x_3 &= 5, \\ 2x_1 - x_2 + 4x_3 &= 9, \\ -2x_1 + 3x_2 + x_3 &= -6. \end{aligned}$$

Using MATLAB to perform the calculations

```
>> a = [1 0 3 ; 2 -1 4 ; -2 3 1];
>> b = [5 9 -6]';
>> x = inv(A)*b

x =

     2
    -1
     1
```

Therefore the solution is $x_1 = 2$, $x_2 = -1$ and $x_3 = 1$. □

Note that the solution to a linear system of the form $A\mathbf{x} = \mathbf{b}$ can also be calculated using the command `A\b`. For example

```
>> a = [1 0 3 ; 2 -1 4 ; -2 3 1];
>> b = [5 9 -6]';
>> x = A\b

x =

     2
    -1
     1
```

6.2 Eigenvectors and eigenvalues

Definition 6.2.1. If A is a square matrix, $\mathbf{x}$ is an *eigenvector* that satisfies

$$A\mathbf{x} = \lambda\mathbf{x}, \tag{6.2}$$

where λ is a scalar known as the *eigenvalue*.

Rearranging Eq. (6.2) gives

$$\begin{aligned} A\mathbf{x} - \lambda\mathbf{x} &= 0 \\ \therefore (A - \lambda I)\mathbf{x} &= 0, \end{aligned}$$

therefore the determinant of the matrix $(A - \lambda I)$ must be equal to zero. So to calculate the eigenvalues we find the values of λ that satisfy

$$\det(A - \lambda I) = 0.$$

Once the values of λ have been determined, an eigenvector is the solution to the homogeneous system $(A - \lambda I) = 0$.

Example 6.2.1. Calculate the eigenvectors of the matrix

$$A = \begin{pmatrix} 1 & 1 \\ -2 & 4 \end{pmatrix}.$$

Using the relation $\det(A - \lambda I) = 0$ then

$$\begin{aligned} 0 &= \begin{vmatrix} 1-\lambda & 1 \\ -2 & 4-\lambda \end{vmatrix} \\ &= (1-\lambda)(4-\lambda) + 2 \\ &= \lambda^2 - 5\lambda + 6 \\ &= (\lambda - 2)(\lambda - 3). \end{aligned}$$

Therefore A has two eigenvalues $\lambda_1 = 2$ and $\lambda_2 = 3$. The eigenvector associated with λ_1 is the solution to the following system:

$$\begin{aligned} -x_1 &+ x_2 &= 0, \\ -2x_1 &+ 2x_2 &= 0. \end{aligned}$$

The first equation gives $x_1 = x_2$, therefore

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_1 \end{pmatrix} = x_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

This means that an eigenvector of A associated to the eigenvalue $\lambda = 2$ is

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Note that any scalar multiple of an eigenvector is also an eigenvector associated with the same eigenvalue. The eigenvector associated with λ_2 is the solution to the following linear system:

$$\begin{aligned} -2x_1 &+ x_2 &= 0, \\ -2x_1 &+ x_2 &= 0. \end{aligned}$$

Either equation gives the solution $x_2 = 2x_1$, therefore

$$\mathbf{x} = \begin{pmatrix} x_1 \\ 2x_1 \end{pmatrix} = x_1 \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

So the eigenvector of A associated with the eigenvalue $\lambda = 3$ is

$$\mathbf{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

□

Geometrically speaking an eigenvector of a matrix A does not change direction when multiplied by A. If $\mathbf{x}$ is an eigenvector of A associated with the eigenvalue λ, then $\|A\mathbf{x}\| = \lambda|\mathbf{x}|$. This can be seen in Fig. 6.1 where the vector $A\mathbf{x}$ is a scalar multiple of the vector $\mathbf{x}$. If $\mathbf{v}$ is any non-eigenvector of A then $A\mathbf{v}$ will point in a different the direction to $\mathbf{v}$. This means that eigenvectors are very useful when dealing with differential equations since they can be used to 'fix' the rate of change of a variable.

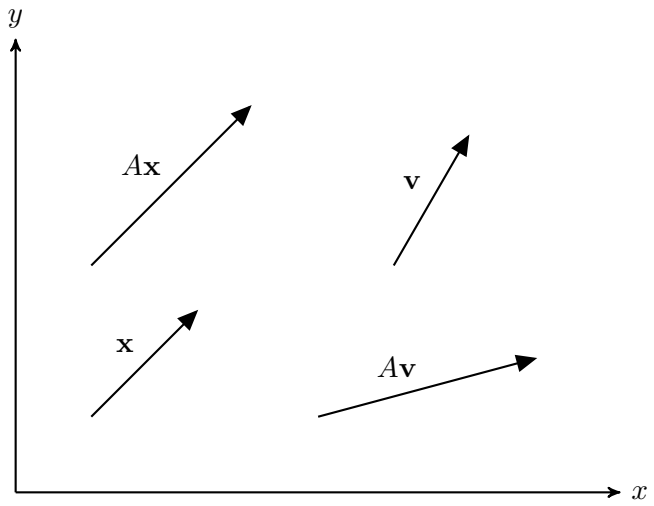


Figure 6.1: The eigenvectors of A do not change direction when multiplied by A.

6.2.1 Calculating eigenvectors and eigenvalues using MATLAB

The MATLAB command for calculating the eigenvalues of a matrix `A` is

```
eig(A)
```

To calculate the eigenvectors we can use

```
[v,e] = eig(A)
```

where `v` is a matrix where each column is an eigenvector and `e` is a matrix where the elements on the main diagonal are the eigenvalues.

For example, to calculate the eigenvectors and eigenvalues of

$$A = \begin{pmatrix} 1 & 1 \\ -2 & 4 \end{pmatrix},$$

we use

```
>> a = [1 1 ; -2 4];
>> [v,e] = eig(A)

v =

   -0.7071   -0.4472
   -0.7071   -0.8944


e =

     2     0
     0     3
```

Note that the eigenvectors have been given as unit vectors. To avoid this use the `'nobalance'` option, e.g.,

```
>> [v,e] = eig(A,'nobalance')

v =

   -1.0000   -0.5000
   -1.0000   -1.0000


e =

     2     0
     0     3
```

6.3 Matrix norms

The analysis of solution methods for linear systems often require a means of comparison between two matrices in terms of their 'size' or 'distance' from each other and this can achieved through the use of *matrix norms*. Matrix norms are defined in terms of a vector norm so we refer to matrix norms as *subordinate* to the vector norm (Stensby, 2013).

Definition 6.3.1. The *vector norm* of some vector $\mathbf{x}$ is denoted by $|\mathbf{x}|$ and is a measure of the *size* of the vector. For example, if $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a vector then

$$|\mathbf{x}| = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p},$$

is the *p-norm*. When $p = 2$ we have the well known *Euclidean norm* or L^2 *norm*. Vector norms satisfy the following properties:

1. $|\mathbf{x}| > 0$ when $\mathbf{x} \neq 0$ and $|\mathbf{x}| = 0$ if and only if $\mathbf{x} = \mathbf{0}$.
2. $|k\mathbf{x}| = |k||\mathbf{x}|$ for any scalar k.
3. $|\mathbf{x}+\mathbf{y}| \leq |\mathbf{x}| + |\mathbf{y}|$.

The matrix norm of an $n \times n$ matrix A is denoted by $\|A\|$ and defined in terms of the vector norm as

$$\|A\| = \max \left(\frac{|A\mathbf{x}|}{|\mathbf{x}|} \right),$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ is an $n \times 1$ column vector and $|\mathbf{x}| \neq \mathbf{0}$. Matrix norms satisfy the following properties:

$$\begin{aligned} \|A\| &> 0 \qquad \text{when } A \neq 0. \\ \|kA\| &= |k|\|A\| \qquad \text{for any scalar } k. \\ \|A + B\| &\leq \|A\| + \|B\|. \\ \|AB\| &\leq \|A\|\|B\|. \end{aligned}$$

Three of the most common matrix norms are the 1-norm, the ∞-norm and the 2-norm:

- *matrix 1-norm*:

$$\|A\|_1 = \max_j \left(\sum_{i=1}^{n} |a_{ij}| \right),$$

  i.e., the maximum column sum using absolute values.

- *matrix ∞-norm*:

$$\|A\|_\infty = \max_i \left(\sum_{j=1}^{n} |a_{ij}| \right),$$

  i.e., the maximum row sum using absolute values.

- *matrix 2-norm*

$$\|A\|_2 = \sqrt{\text{largest eigenvalue of } A'A}.$$

Example 6.3.1. Calculate the (i) 1-norm, (ii) ∞-norm and (iii) 2-norm of the following matrix:

$$A = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 2 \\ 2 & 1 & 1 \end{pmatrix}.$$

(i) $\|A\|_1 = \max(1 + 0 + 2, 0 + 1 + 1, 2 + 2 + 1) = \max(3, 2, 5) = 5$

(ii) $\|A\|_\infty = \max(1 + 0 + 2, 0 + 1 + 2, 2 + 1 + 1) = \max(3, 3, 4) = 4$

(iii)

$$A'A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ -2 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 2 \\ 2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 2 & 0 \\ 2 & 2 & 3 \\ 0 & 3 & 9 \end{pmatrix},$$

using MATLAB to calculate the eigenvalues

```
>> a = [5 2 0 ; 2 2 3 ; 0 3 9];
>> eig(A)

ans =

    0.1565
    5.6341
   10.2094
```

Therefore $\|A\|_2 = \sqrt{10.2904} = 3.1952$.

□

6.3.1 Calculating matrix norms using MATLAB

The MATLAB commands for calculating the matrix 1-norm, ∞-norm and 2-norm respectively are:

```
norm(A,1)
norm(A,inf)
norm(A,2)
```

For example, to calculate the 1-norm, ∞-norm and 2-norm of

$$A = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 2 \\ 2 & 1 & 1 \end{pmatrix},$$

we use

```
>> a = [1 0 -2 ; 0 1 2 ; 2 1 1];
>> norm(A,1)

ans =

     5

>> norm(A,inf)

ans =

     4

>> norm(A,2)

ans =

    3.1952
```

6.4 Spectral radius

Definition 6.4.1. The *spectral radius* of a square matrix A is denoted $\rho(A)$ is defined as

$$\rho(A) = \max_i(|\lambda_i|), \tag{6.3}$$

where λ_i are the eigenvalues of A, i.e., the spectral radius is the largest absolute eigenvalue.

Theorem 6.4.1. *$\rho(A) \leq \|A\|$ for all subordinate matrix norms.*

Proof. Let $\mathbf{x}$ be an eigenvector of A where $A\mathbf{x} = \lambda\mathbf{x}$ then

$$|A\mathbf{x}| = |\lambda\mathbf{x}| = |\lambda||\mathbf{x}|.$$

A property of matrix norms states that

$$|A\mathbf{x}| \leq \|A\||\mathbf{x}|,$$

therefore

$$\begin{aligned} |\lambda||\mathbf{x}| &\leq \|A\||\mathbf{x}| \\ |\lambda| &\leq \|A\|. \end{aligned}$$

Since $\rho(A) = \max_i |\lambda_i|$ then $\rho(A) \leq \|A\|$. □

Note that the spectral radius is not a matrix norm. For example, consider the spectral radius of the following matrices:

$$A = \begin{pmatrix} 1 & 0 \\ 2 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 0 & 2 \\ 0 & 1 \end{pmatrix}.$$

Here $\rho(A) = 1$ and $\rho(B) = 1$. Now consider the spectral radius of $A + B$:

$$A + B = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix},$$

$\rho(A + B) = 3$. This means $\rho(A + B) \not\leq \rho(A) + \rho(B)$ which contradicts the properties of norms given on page 81, therefore the spectral radius is not a matrix norm.

Example 6.4.1. Calculate the spectral radii of the following matrices

$$A = \begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix}, \qquad B = \begin{pmatrix} 4 & -1 & 2 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}.$$

Using MATLAB to perform the calculations

```
>> a = [1 -1 ; 2 4];
>> B = [4 -1 2 ; 0 2 1 ; 0 1 2];
>> max(abs(eig(A)))

ans =

     3

>> max(abs(eig(B)))

ans =

     4
```

Therefore $\rho(A) = 3$ and $\rho(B) = 4$. □

6.5 Ill-conditioned systems

An *ill-conditioned system* is a system where small variations in the value of the coefficient matrix or right-hand side vector cause large variations in the solution to $\mathbf{x}$. For example, consider the following system

$$\begin{pmatrix} 1 & 1 \\ 1 & 1.001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}.$$

The solution to this system is $x_1 = 2$ and $x_2 = 0$. Now see what happens when a very slight variation is applied to the right-hand side vector

$$\begin{pmatrix} 1 & 1 \\ 1 & 1.001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2.001 \end{pmatrix}.$$

The solution is now $x_1 = 1$ and $x_2 = 1$. Ill-conditioned systems are sensitive to rounding errors during intermediate calculations.

Definition 6.5.1. The *condition number* of an $n \times n$ matrix A is denoted by $\kappa(A)$ and defined by

$$\kappa(A) = \|A\|\|A^{-1}\|,$$

The larger the value of $\kappa(A)$ the more sensitive the system $A\mathbf{x} = \mathbf{b}$ is to small changes in the values of A and $\mathbf{b}$.

Proof. Let $\delta\mathbf{x}$ be the variation in the solution of the system $A\mathbf{x} = \mathbf{b}$ and $\delta\mathbf{b}$ be the variation in the right-hand side vector $\mathbf{b}$. The approximate solution $\mathbf{x} + \delta\mathbf{x}$ corresponds to the right-hand side vector $\mathbf{b} + \delta\mathbf{b}$, i.e.,

$$A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b},$$

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

$$\begin{aligned} A\delta\mathbf{x} &= \delta\mathbf{b} \\ \therefore \delta\mathbf{x} &= A^{-1}\delta\mathbf{b}. \end{aligned} \tag{6.4}$$

The condition number is the relative change in $\delta\mathbf{x}$ compared to $\delta\mathbf{b}$. Using the properties of vector norms on Eq. (6.4)

$$|\delta\mathbf{x}| = |A^{-1}\delta\mathbf{b}| \leq \|A^{-1}\||\delta\mathbf{b}|, \tag{6.5}$$

and for $A\mathbf{x} = \mathbf{b}$:

$$|\mathbf{b}| = |A\mathbf{x}| \leq \|A\||\mathbf{x}|,$$

or to put it another way

$$\|A\||\mathbf{x}| \geq |\mathbf{b}|. \tag{6.6}$$

Dividing Eq. (6.5) by Eq. (6.6) gives the change in $\delta\mathbf{x}$ relative to the change in $\delta\mathbf{b}$

$$\begin{aligned} \frac{|\delta\mathbf{x}|}{\|A\||\mathbf{x}|} &\leq \frac{\|A^{-1}\||\delta\mathbf{b}|}{|\mathbf{b}|} \\ \frac{|\delta\mathbf{x}|}{|\mathbf{x}|} &\leq \|A\|\|A^{-1}\|\frac{|\delta\mathbf{b}|}{|\mathbf{b}|}. \end{aligned}$$

Therefore the relative change in $\delta\mathbf{x}$ is less than or equal to the relative change in $\delta\mathbf{b}$ multiplied by $\|A\|\|A^{-1}\|$ which is the condition number $\kappa(A)$. □

Example 6.5.1. Calculate the condition number of the matrices (i) $A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$ and (ii) $B = \begin{pmatrix} 1 & 1 \\ 1 & 1.1 \end{pmatrix}$.

(i) Calculate A^{-1}

$$A^{-1} = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}.$$

Using the 1-norm, $\|A\|_1 = 3$ and $\|A^{-1}\|_1 = 3$ so $\kappa(A) = 9$.

(ii) Calculate B^{-1}

$$B^{-1} = \frac{1}{0.1}\begin{pmatrix} 1.1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 11 & -10 \\ -10 & 10 \end{pmatrix}.$$

Using the 1-norm, $\|B\|_1 = 2.1$ and $\|B^{-1}\|_1 = 21$ so $\kappa(B) = 44.1$

□

6.5.1 Calculating the condition number using MATLAB

The MATLAB command for calculated the condition number is

```
cond(A,p)
```

where `p` defines the matrix norm used (`1`, `2` or `inf`). For example, to calculate the condition numbers of the systems (i) and (ii) from Example 6.5.1

```
>> a = [1 1 ; 1 2];
>> cond(A,1)

ans =

     9

>> a = [1 1 ; 1 1.1];
>> cond(A,1)

ans =

   44.1000
```

6.6 Elementary row operations

Linear systems of the form $A\mathbf{x} = \mathbf{b}$ can be manipulated by performing *elementary row operations* (EROs) on the matrices A and $\mathbf{b}$ in order to make solutions more easily obtainable. The three types of EROs are listed below.

Type I – Permute (swap) row i and row j:

$$R_i \leftrightarrow R_j, \qquad i \neq j.$$

Type II – Multiply row i by a non-zero scalar k:

$$R_i \to kR_i, \qquad k \neq 0.$$

Type III – Add k times row j to row i:

$$R_i \to R_i + kR_j, \qquad i \neq j.$$

Note that as long as the EROs are applied to both the coefficient matrix A and the right-hand side vector $\mathbf{b}$ then the application of an ERO does not alter the solution to a linear system, i.e., the solution to a system with an ERO applied to it is the same as the solution to the original system.

6.7 Gaussian elimination

Elementary row operations can be applied to help solve a linear system using a process called *Gaussian elimination* or *row reduction*. To begin with, an *augmented matrix* is formed by

combining the coefficient matrix A and the right-hand side vector $\mathbf{b}$ to form an $m \times (n+1)$ matrix $[A|\mathbf{b}]$, e.g.,

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_n \end{array}\right).$$

Note that a partition has been used here to distinguish between the elements of the coefficient matrix and the right-hand side vector.

Definition 6.7.1. A matrix A is said to be in *row echelon form* if it satisfies the following:

- All non-zero rows are above all zero rows;
- The first non-zero element in a row (called the *pivot element*) is to the right of the pivot element of the row above;
- All elements in the column below the pivot element are zeroes.

The aim of Gaussian elimination is to manipulate the augmented matrix $[A|\mathbf{b}]$ using EROs so that the matrix A is in row echelon form. This is done by adding a multiple of the pivot row to the rows beneath the pivot row (Type III ERO). Let a_{ij} be an element in the augmented matrix and a_{pq} be the pivot element in row p and column q then

$$a_{ij} = a_{ij} - \frac{a_{iq}}{a_{pq}} a_{pj},$$

where $i = p, \ldots, n$ and $j = 1, \ldots, n+1$. Once the augmented matrix is in row echelon form the solution to the system is found using back substitution.

6.7.1 Back and forward substitution

Once a coefficient matrix has been reduced to row echelon form the solution for $\mathbf{x}$ can be obtained using a method called *back substitution*. Consider the following matrix equation that has been reduced to row echelon form:

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix},$$

this corresponds to the linear system

$$\begin{array}{rcrcrcrcl} u_{11}x_1 & + & u_{12}x_2 & + & u_{13}x_3 & + & \cdots & + & u_{1n}x_n = b_1, \\ & & u_{22}x_2 & + & u_{23}x_3 & + & \cdots & + & u_{2n}x_n = b_2, \\ & & & & u_{33}x_3 & + & \cdots & + & u_{3n}x_n = b_3, \\ & & & & & & & & \vdots \quad\quad \vdots \\ & & & & & & & & u_{mn}x_n = b_n. \end{array}$$

Rearranging of the final equation gives

$$x_n = \frac{b_n}{u_{mn}}, \tag{6.7}$$

and for all other equations

$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right), \qquad i = n-1, \ldots, 1. \tag{6.8}$$

A matrix reduced to row echelon form is called an *upper triangular matrix* where all elements below the main diagonal are all zeros. Some solution methods require the solution using a *lower triangular matrix* where elements above the main diagonal are all zeros, e.g.,

$$\begin{pmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{m1} & l_{m2} & l_{m3} & \cdots & l_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix}.$$

The forward substitution method begins with the solution of x_1 using

$$x_1 = \frac{b_1}{l_{11}}, \tag{6.9}$$

all then for all other values of x_i

$$x_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right), \qquad i = 2, 3, \ldots, n. \tag{6.10}$$

Example 6.7.1. Solve the following linear system using Gaussian elimination

$$\begin{pmatrix} 1 & 3 & 0 \\ 2 & -4 & -1 \\ -3 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 5 \\ -1 \end{pmatrix}.$$

Using row operations to reduce the augmented matrix to row echelon form:

$$\left(\begin{array}{ccc|c} 1 & 3 & 0 & -1 \\ 2 & -4 & -1 & 5 \\ -3 & 1 & 2 & -1 \end{array} \right) \begin{array}{l} \\ R_2 - 2R_1 \\ R_2 + 3R_1 \end{array} \sim \left(\begin{array}{ccc|c} 1 & 3 & 0 & -1 \\ 0 & -10 & -1 & 7 \\ 0 & 10 & 2 & -4 \end{array} \right) \begin{array}{l} \\ \\ R_3 + R_2 \end{array}$$

$$\sim \left(\begin{array}{ccc|c} 1 & 3 & 0 & -1 \\ 0 & -10 & -1 & 7 \\ 0 & 0 & 1 & 3 \end{array} \right).$$

Using back substitution to find the solution

$$\begin{aligned} x_3 &= 3, \\ x_2 &= -\frac{1}{10}(7+3) = -1, \\ x_1 &= -1 + 3 = 2. \end{aligned}$$

□

6.7.2 MATLAB code for Gaussian elimination and back substitution

The MATLAB code for solving the linear system $A\mathbf{x} = \mathbf{b}$ using Gaussian elimination is given in Listing 6.1.

Listing 6.1: MATLAB function to solve the linear system $A\mathbf{x} = \mathbf{b}$ using Gaussian elimination.

```
function x = gelim(A,b)

% gelim.m by Jon Shiach
%
% This function calculates the solution to the linear system Ax=b using
% Gaussian elimination

% calculate the size of the coefficient matrix
[m,n] = size(A);

% loop through the columns of the coefficient matrix
for j = 1 : m

    % row reduce rows below row j
    for i = j + 1 : n
        ratio = A(i,j)/A(j,j);
        for k = j : m
            A(i,k) = A(i,k) - ratio*A(j,k);
        end
        b(i) = b(i) - ratio*b(j);
    end
end

% perform back substitution to calculate solution
for i = m : -1 : 1
    x(i) = b(i);
    for j = i + 1 : m
        x(i) = x(i) - A(i,j)*x(j);
    end
    x(i) = x(i)/A(i,i);
end
```

6.8 Partial pivoting

Sometimes using only type III EROs will result in divide by zero errors or causing inaccuracies in the solution. Consider the following linear system.

$$\begin{aligned} 0.001x_1 &+ x_2 &= 1, \\ x_1 &+ x_2 &= 2. \end{aligned}$$

Basic algebra shows that the solution to this system is $x_1 = 1.001$ and $x_2 = 0.999$. Using Gaussian elimination to solve this system working to 4 decimal place accuracy we have:

$$\left(\begin{array}{cc|c} 0.001 & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \begin{array}{l} \\ R_2 - \frac{1}{0.001}R_1 \end{array} \sim \left(\begin{array}{cc|c} 0.001 & 1 & 1 \\ 0 & -999 & -998 \end{array}\right).$$

Therefore the solution is

$$\begin{aligned} x_2 &= \frac{998}{999} = 0.9990, \\ x_1 &= \frac{1}{0.001}\left(1 - 0.999\right) = 1.0000. \end{aligned}$$

So the solution of x_1 differs from the actual solution by 0.001. This is because the value of the pivot element (0.001) was small compared to the element below it (1) and this produces large numbers elsewhere in the augmented matrix when EROs are applied. Now consider the same system with the exception that the equations are written in a different order, i.e.,

$$\begin{aligned} x_1 &+ x_2 &= 2,\\ 0.001x_1 &+ x_2 &= 1. \end{aligned}$$

Now applying Gaussian elimination working to 4 decimal place accuracy we have:

$$\left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0.001 & 1 & 1 \end{array}\right) \begin{array}{l} \\ R_2 - 0.001R_1 \end{array} \quad \sim \quad \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 0.9990 & 0.9980 \end{array}\right).$$

Therefore the solution is

$$\begin{aligned} x_2 &= \frac{0.9980}{0.9990} = 0.999,\\ x_1 &= 2 - 0.999 = 1.001. \end{aligned}$$

By swapping the two equations around the value of the pivot element was large compared to the element below it so we do not have large numbers in the augmented matrix.

This is an example of *partial pivoting* where we swap the pivot row with the row with the largest absolute value in the column beneath the pivot element. Partial pivoting has two main advantages: some systems cannot be solved without partial pivoting, e.g.,

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix},$$

also Gaussian elimination is more stable when using partial pivoting.

Example 6.8.1. Solve the following linear system using Gaussian elimination with partial pivoting:

$$\begin{aligned} 2x_1 &+ 2x_2 &+ x_3 &= 6,\\ 6x_1 &+ 3x_2 &- x_3 &= 13,\\ -4x_1 &+ 2x_2 &+ 2x_3 &= -10. \end{aligned}$$

Row reduce to row echelon form using partial pivoting

$$\left(\begin{array}{ccc|c} 2 & 2 & 1 & 6 \\ 6 & 3 & -1 & 13 \\ -4 & 2 & 2 & -10 \end{array}\right) \begin{array}{l} R_1 \leftrightarrow R_2 \\ \\ \\ \end{array} \quad \sim \quad \left(\begin{array}{ccc|c} 6 & 3 & -1 & 13 \\ 2 & 2 & 1 & 6 \\ -4 & 2 & 2 & -10 \end{array}\right) \begin{array}{l} \\ R_2 - \frac{1}{3}R_1 \\ R_3 + \frac{2}{3}R_1 \end{array}$$

$$\sim \quad \left(\begin{array}{ccc|c} 6 & 3 & -1 & 13 \\ 0 & 1 & \frac{4}{3} & \frac{5}{3} \\ 0 & 4 & \frac{4}{3} & -\frac{4}{3} \end{array}\right) \begin{array}{l} \\ R_2 \leftrightarrow R_3 \\ \\ \end{array} \quad \sim \quad \left(\begin{array}{ccc|c} 6 & 3 & -1 & 13 \\ 0 & 4 & \frac{4}{3} & -\frac{4}{3} \\ 0 & 1 & \frac{4}{3} & \frac{5}{3} \end{array}\right) \begin{array}{l} \\ \\ \\ R_3 - \frac{1}{4}R_2 \end{array}$$

$$\sim \quad \left(\begin{array}{ccc|c} 6 & 3 & -1 & 13 \\ 0 & 4 & \frac{4}{3} & -\frac{4}{3} \\ 0 & 0 & 1 & 2 \end{array}\right).$$

Using back-substitution to calculate the solution

$$\begin{aligned} x_3 &= 2,\\ x_2 &= \frac{1}{4}\left(-\frac{4}{3} - \frac{8}{3}\right) = -1,\\ x_1 &= \frac{1}{6}(13 + 2 + 3) = 3. \end{aligned}$$

□

Chapter 7

LU factorisation

You should have seen through previous studies in the subject that the solution to a linear system can be obtained using a process called Gaussian elimination (see Section 6.7). Gaussian elimination uses Elementary Row Operations (EROs) to row reduce the coefficient matrix to an upper triangular matrix so that a solution can be found using back substitution. Another method commonly used to solve linear systems is to factor the coefficient matrix into the product of two triangular matrices. The purpose of doing this is that forward and back substitution can be directly applied to these matrices to obtain a solution to the linear system. Furthermore, the values of the right-hand side vector can change without having to recalculate the factorisation.

7.1 Elementary matrices

Definition 7.1.1. An *elementary matrix* is a matrix that is produced by performing a single elementary row operation (ERO) on an identity matrix. There are three EROs and therefore there are three types of elementary matrices.

Type I – Permute (swap) rows i and j of I

$$P_{ij} : R_i \leftrightarrow R_j, \qquad i \neq j.$$

Type II – Multiply row i of I by the non-zero scalar k

$$M_i(k) : R_i \to kR_i, \qquad k \neq 0.$$

Type III – Add k times row j of I to row i of I

$$A_{ij}(k) : R_i \to R_i + kR_j, \qquad i \neq j. \tag{7.1}$$

Example 7.1.1. Write down all of the elementary matrices that can be applied to a 2×2 matrix.

$$P_{12} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad M_1(k) = \begin{pmatrix} k & 0 \\ 0 & 1 \end{pmatrix}, \qquad M_2(k) = \begin{pmatrix} 1 & 0 \\ 0 & k \end{pmatrix},$$
$$A_{12}(k) = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix}, \qquad A_{21}(k) = \begin{pmatrix} 1 & 0 \\ k & 1 \end{pmatrix},$$

where $k \in \mathbb{R}$. □

Remark 1. Premultiplying an $n \times m$ matrix A by an $n \times n$ elementary matrix E has the same effect of performing the corresponding elementary row operation on A. For example, given the following matrix

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix},$$

then

$$P_{12}A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 1 & 2 \end{pmatrix}, \qquad \text{(permute rows 1 and 2)},$$

$$M_1(k)A = \begin{pmatrix} k & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} k & 2k \\ 3 & 4 \end{pmatrix}, \qquad \text{(multiply row 1 by k)},$$

$$M_2(k)A = \begin{pmatrix} 1 & 0 \\ 0 & k \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3k & 4k \end{pmatrix}, \qquad \text{(multiply row 2 by k)},$$

$$A_{12}(k)A = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1+3k & 2+4k \\ 3 & 4 \end{pmatrix}, \qquad \text{(add } k \text{ times row 2 to row 1)},$$

$$A_{21}(k)A = \begin{pmatrix} 1 & 0 \\ k & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3+k & 4+2k \end{pmatrix}, \qquad \text{(add } k \text{ times row 1 to row 2)}.$$

7.1.1 Inverse of an elementary matrix

Since an elementary matrix is the results of applying a single ERO, the calculation of the inverse elementary matrix is a simple matter of undoing the ERO that was applied. It can be easily seen that

$$\begin{aligned} P_{ij}^{-1} &= P_{ij}, \\ M_i^{-1}(k) &= M_i\left(\tfrac{1}{k}\right), \\ A_{ij}^{-1}(k) &= A_{ij}(-k). \end{aligned}$$

Example 7.1.2. Write down all of the inverse elementary matrices that can be applied to a 2×2 matrix.

$$P_{12}^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad M_1^{-1}(k) = \begin{pmatrix} \frac{1}{k} & 0 \\ 0 & 1 \end{pmatrix}, \qquad M_2^{-1}(k) = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{k} \end{pmatrix},$$

$$A_{12}^{-1}(k) = \begin{pmatrix} 1 & -k \\ 0 & 1 \end{pmatrix}, \qquad A_{21}^{-1}(k) = \begin{pmatrix} 1 & 0 \\ -k & 1 \end{pmatrix},$$

where $k \in \mathbb{R}$. □

7.2 LU factoristation

LU factorisation (also known as *LU decomposition*) is a procedure for factorising a matrix A into the product of a *lower triangular matrix* L and an *upper triangular matrix* U, i.e.,

$$A = LU.$$

The advantage of writing a matrix as a product of L and U is that the solution to a triangular set of equations is easy to calculate using forward and back substitution.

If A is an $m \times n$ matrix then L and U have the following properties:

- L is an $m \times m$ lower triangular matrix where $[L]_{ij} = 0$ when $i < j$, e.g.,

$$L = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}.$$

- U is an $m \times n$ upper triangular matrix where $[U]_{ij} = 0$ when $i > j$, e.g.,

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix}.$$

If A is a 2×2 matrix then using LU factorisation we have

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{pmatrix}.$$

Using matrix multiplication gives the following system of equations

$$\begin{aligned} a_{11} &= l_{11}u_{11}, \\ a_{12} &= l_{11}u_{12}, \\ a_{21} &= l_{21}u_{11}, \\ a_{22} &= l_{21}u_{12} + l_{22}u_{22}. \end{aligned}$$

This system is underdetermined since we have four equations and six unknowns (l_{11}, l_{21}, l_{22}, u_{11}, u_{12} and u_{22}). However, if the elements on the main diagonal of L have a value of 1 (i.e., $l_{11} = 1$ and $l_{22} = 1$) then we have the same number of equations as unknowns and a unique solution can be found.

Theorem 7.2.1. *Given a set of k elementary matrices $E_1, E_2, \ldots, E_k$ applied to a matrix A to row reduce into row echelon form without permuting rows, then A can be written as the product of two matrices L and U such that*

$$A = LU, \tag{7.2}$$

where

$$U = E_k \ldots E_2 E_1 A, \tag{7.3}$$

$$L = E_1^{-1} E_2^{-1} \ldots E_k^{-1}. \tag{7.4}$$

Proof. Let A be a non-singular square matrix A that has been reduced to row echelon form without permuting rows and E_i are the elementary matrices corresponding to the elementary row operations. Therefore

$$U = E_k \dots E_2 E_1 A,$$

which is the same as Eq. (7.3). Inverting the elementary matrices and premultiplying both sides gives

$$E_1^{-1} E_2^{-1} \dots E_k^{-1} U = A,$$

since $A = LU$ then

$$L = E_1^{-1} E_2^{-1} \dots E_k^{-1},$$

which is the same as Eq. (7.4). □

Example 7.2.1. Determine the LU factorisation of the following matrix using elementary matrices:

$$A = \begin{pmatrix} 1 & 3 & 0 \\ 2 & -4 & -1 \\ -3 & 1 & 2 \end{pmatrix}.$$

Row reduce A to row echelon form

$$\begin{pmatrix} 1 & 3 & 0 \\ 2 & -4 & -1 \\ -3 & 1 & 2 \end{pmatrix} \begin{matrix} \\ R_2 - 2R_1 \\ R_3 + 3R_1 \end{matrix} \sim \begin{pmatrix} 1 & 3 & 0 \\ 0 & -10 & -1 \\ 0 & 10 & 2 \end{pmatrix} \begin{matrix} \\ \\ R_3 + R_2 \end{matrix} \sim \begin{pmatrix} 1 & 3 & 0 \\ 0 & -10 & -1 \\ 0 & 0 & 1 \end{pmatrix}.$$

The elementary matrices used to row reduce A are

$$E_1 = A_{21}(-2), \qquad E_2 = A_{31}(3), \qquad E_3 = A_{32}(1),$$

hence the inverse elementary matrices are

$$E_1^{-1} = A_{21}(2), \qquad E_2^{-1} = A_{31}(-3), \qquad E_3^{-1} = A_{32}(-1).$$

Therefore

$$U = A_{32}(1) A_{31}(3) A_{21}(-2) A = \begin{pmatrix} 1 & 3 & 0 \\ 0 & -10 & -1 \\ 0 & 0 & 1 \end{pmatrix},$$

$$L = A_{21}(2) A_{31}(-3) A_{32}(-1) = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -1 & 1 \end{pmatrix}.$$

The LU factorisation of A is

$$A = \begin{pmatrix} 1 & 3 & 0 \\ 2 & -4 & -1 \\ -3 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 3 & 0 \\ 0 & -10 & -1 \\ 0 & 0 & 1 \end{pmatrix}.$$

□

Example 7.2.2 (Worked example). Determine the LU factorisation of the following matrices using elementary matrices:

(i) $A = \begin{pmatrix} 2 & 3 \\ 5 & -1 \end{pmatrix}$

(ii) $B = \begin{pmatrix} -1 & 3 & -2 \\ 1 & -2 & 1 \\ 4 & 4 & -3 \end{pmatrix}$

□

7.3 Solving a linear system using LU factorisation

Consider the solution of the linear system $A\mathbf{x} = \mathbf{b}$. Using LU factorisation $A = LU$ so

$$LU\mathbf{x} = \mathbf{b}.$$

Let $\mathbf{y} = U\mathbf{x}$ then

$$L\mathbf{y} = \mathbf{b}.$$

To solve a linear system using LU factorisation we do the following

1. Determine the LU factorisation of the coefficient matrix A;
2. Solve $L\mathbf{y} = \mathbf{b}$ using forward substitution;
3. Solve $U\mathbf{x} = \mathbf{y}$ using back substitution.

Example 7.3.1. Solve the following linear system using LU factorisation

$$\begin{aligned} x_1 + 3x_2 &= -1, \\ 2x_1 - 4x_2 - x_3 &= 5, \\ -3x_1 + x_2 + x_3 &= -1. \end{aligned}$$

Writing the linear system in the form $A\mathbf{x} = \mathbf{b}$

$$\begin{pmatrix} 1 & 3 & 0 \\ 2 & -4 & -1 \\ -3 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 5 \\ -1 \end{pmatrix}.$$

The matrices L and U where $LU = A$ are (shown in Example 7.2.1)

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -1 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 1 & 3 & 0 \\ 0 & -10 & -1 \\ 0 & 0 & 1 \end{pmatrix}.$$

Therefore $L\mathbf{y} = \mathbf{b}$ is

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 5 \\ -1 \end{pmatrix}.$$

Apply forward subsitution to solve $\mathbf{y}$:

$$\begin{aligned} y_1 &= -1, \\ y_2 &= 5 - 2y_1 = 5 + 2 = 7, \\ y_3 &= -1 + 3y_1 + y_2 = -1 - 3 + 7 = 3. \end{aligned}$$

$U\mathbf{x} = \mathbf{y}$ is

$$\begin{pmatrix} 1 & 3 & 0 \\ 0 & -10 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 7 \\ 3 \end{pmatrix}.$$

Apply back subsitution to solve $\mathbf{x}$:

$$\begin{aligned} x_3 &= 3, \\ x_2 &= \frac{7 + x_3}{-10} = \frac{7 + 3}{-10} = -1, \\ x_1 &= -1 - 3x_2 = 2. \end{aligned}$$

□

Example 7.3.2 (Worked example). Solve the following system of equations using LU factorisation

$$\begin{aligned} x_1 + x_2 + 3x_3 &= 14, \\ 2x_1 + x_2 + 3x_3 + x_4 &= 16, \\ -x_1 + 3x_2 + x_3 &= 6, \\ x_1 + 3x_3 + 2x_4 &= 20. \end{aligned}$$

□

7.4 LUP factorisation

The method of LU factorisation only allows for elementary row operations of type III to be applied to row reduce a matrix into row echelon form. It was seen in Section 6.8 that partial pivoting improves the stability of Gaussian elimination and may be necessary if the value of a pivot element is zero. For this reason it is necessary to consider factorising using type I elementary row operations, i.e., permuting two rows.

Theorem 7.4.1 (LUP factorisation). *If A is an $n \times n$ non-singular matrix that is expressed in row echelon form by use of elementary matrices of type I (permutation of two rows P_i) and type III (addition of a multiple of another row A_i), then A can be expressed in the form*

$$PA = LU, \tag{7.5}$$

where

$$U = A_k P_k \dots A_2 P_2 A_1 P_1 A, \tag{7.6a}$$
$$P = P_k \dots P_2 P_1, \tag{7.6b}$$
$$L = L_1 L_2 \dots L_k, \tag{7.6c}$$

and

$$L_i = P_k \dots P_{i+2} P_{i+1} A_i^{-1} P_{i+1} P_{i+2} \dots P_k. \tag{7.7}$$

Proof. Let A be a non-singular square matrix that has been reduced to row echelon form by Gaussian elimination with partial pivoting. Without loss of generality, let P_1, P_2, P_3 be elementary matrices of type I (permuting of two rows) and $A_1 A_2 A_3$ be elementary matrices of type III (adding a multiple of one two to another), the upper triangular matrix U is

$$U = A_3 P_3 A_2 P_2 A_1 P_1 A. \tag{7.8}$$

which is the same form as Eq. (7.6a). Rearranging $PA = LU$ gives

$$U = L^{-1} P A. \tag{7.9}$$

Let L_i^{-1} be a permutation of A_i, then writing Eq. (7.8) into the form similar to Eq. (7.9) gives

$$U = L_3^{-1} L_2^{-1} L_1^{-1} P_3 P_2 P_1 A, \tag{7.10}$$

Therefore

$$P = P_3 P_2 P_1,$$

which is the same form as Eq. (7.6b). To determine L define

$$L_3^{-1} = A_3, \tag{7.11a}$$
$$L_2^{-1} = P_3 A_2 P_3, \tag{7.11b}$$
$$L_1^{-1} = P_3 P_2 A_1 P_2 P_3, \tag{7.11c}$$

then Eq. (7.10) can be written as

$$U = \underbrace{A_3}_{L_3^{-1}} \underbrace{P_3 A_2 P_3}_{L_2^{-1}} \underbrace{P_3 P_2 A_1 P_2 P_3}_{L_1^{-1}} P_3 P_2 P_1 A,$$

since $P_i P_i = I$, pairs of permutation matrices can be cancelled out to give

$$U = A_3 P_3 A_3 P_2 A_1 P_1 A,$$

which is the same as Eq. (7.8). Finally from Eq. (7.10)

$$L = (L_3^{-1}L_2^{-1}L_1^{-1})^{-1} = L_1L_2L_3,$$

which is the same form of Eq. (7.6c). The values of L_i come from rearranging Eq. (7.11) to give

$$\begin{aligned} L_1 &= P_3P_2A_1^{-1}P_2P_3, \\ L_2 &= P_3A_2^{-1}P_3, \\ L_3 &= A_3^{-1}. \end{aligned}$$

which is the same form as Eq. (7.7).

□

Example 7.4.1. Determine the LUP factorisation of the matrix

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

Applying ERO with partial pivoting:

$$\begin{pmatrix} 2 & 2 & 1 \\ 6 & 3 & -1 \\ -4 & 2 & 2 \end{pmatrix} \begin{matrix} R_1 \leftrightarrow R_2 \end{matrix} \sim \begin{pmatrix} 6 & 3 & -1 \\ 2 & 2 & 1 \\ -4 & 2 & 2 \end{pmatrix} \begin{matrix} R_2 - \frac{1}{3}R_1 \\ R_3 + \frac{2}{3}R_1 \end{matrix}$$

$$\sim \begin{pmatrix} 6 & 3 & -1 \\ 0 & 1 & \frac{4}{3} \\ 0 & 4 & \frac{4}{3} \end{pmatrix} \begin{matrix} R_2 \leftrightarrow R_3 \end{matrix} \sim \begin{pmatrix} 6 & 3 & -1 \\ 0 & 4 & \frac{4}{3} \\ 0 & 1 & \frac{4}{3} \end{pmatrix} \begin{matrix} \\ R_3 - \frac{1}{4}R_2 \end{matrix}$$

$$\sim \begin{pmatrix} 6 & 3 & -1 \\ 0 & 4 & \frac{4}{3} \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore

$$U = A_{32}(-\tfrac{1}{4})P_{23}A_{31}(\tfrac{2}{3})IA_{21}(-\tfrac{1}{3})P_{12}A = \begin{pmatrix} 6 & 3 & -1 \\ 0 & 4 & \frac{4}{3} \\ 0 & 0 & 1 \end{pmatrix}$$

Note that the method for LUP factorisation assumes rows are permuted for every step of the Gaussian elimination process. Where no permutations are needed we use $P_i = I$ as the elementary matrix.

Let $P_1 = P_{12}$, $P_2 = I$, $P_3 = P_{23}$, $A_1 = A_{12}(-\frac{1}{3})$, $A_2 = A_{13}(\frac{2}{3})$ and $A_3 = A_{23}(-\frac{1}{4})$ then

$$P = P_3P_2P_1 = P_{23}IP_{12} = P_{23}P_{12} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

$$L_1 = P_3P_2A_1^{-1}P_2P_3 = P_{23}A_{21}(\tfrac{1}{3})P_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{3} & 0 & 1 \end{pmatrix}$$

$$L_2 = P_3A_2^{-1}P_3 = P_{23}A_{31}(\tfrac{2}{3})P_{23} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$L_3 = A_3^{-1} = A_{32}(\tfrac{1}{4}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{4} & 1 \end{pmatrix}$$

$$\therefore L = L_1L_2L_3 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & \frac{1}{4} & 1 \end{pmatrix}$$

Therefore the LUP factorisation of A is

$$PA = LU$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 2 & 2 & 1 \\ 6 & 3 & -1 \\ -4 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & \frac{1}{4} & 1 \end{pmatrix} \begin{pmatrix} 6 & 3 & -1 \\ 0 & 4 & \frac{4}{3} \\ 0 & 0 & 1 \end{pmatrix}.$$

□

Example 7.4.2 (Worked example). Determine the LUP factorisation of the following matrix using elementary matrices:

$$A = \begin{pmatrix} 0 & 1 & -2 \\ 1 & 0 & 2 \\ 3 & -2 & 2 \end{pmatrix}.$$

□

7.5 Solving linear systems using LUP factorisation

Consider the solution of the linear system $A\mathbf{x} = \mathbf{b}$. Any permutation of A must also be applied to the right-hand side vector $\mathbf{b}$, therefore

$$PA\mathbf{x} = P\mathbf{b}.$$

Since for LUP factorisation $PA = LU$ then

$$LU\mathbf{x} = P\mathbf{b}.$$

Let $\mathbf{y} = U\mathbf{x}$ then

$$L\mathbf{y} = P\mathbf{b},$$

and

$$U\mathbf{x} = \mathbf{y}.$$

To solve a system using LUP factorisation we do the following:

1. Determine the LUP factorisation of the coefficient matrix A;
2. Solve $L\mathbf{y} = P\mathbf{b}$ using forward substitution;
3. Solve $U\mathbf{x} = \mathbf{y}$ using back substitution.

Example 7.5.1. Solve the following linear system using LUP factorisation:

$$\begin{array}{rcrcrcl} 2x_1 & + & 2x_2 & + & x_3 & = & 6, \\ 6x_1 & + & 3x_2 & - & x_3 & = & 4, \\ -4x_1 & + & 2x_2 & + & 2x_3 & = & 8. \end{array}$$

It was seen in example 7.4.1 that $AP = LU$ where

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & \frac{1}{4} & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 6 & 3 & -1 \\ 0 & 4 & \frac{4}{3} \\ 0 & 0 & 1 \end{pmatrix}, \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

Solve $L\mathbf{y} = P\mathbf{b}$

$$\begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & \frac{1}{4} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 6 \\ 4 \\ 8 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & \frac{1}{4} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \\ 6 \end{pmatrix},$$

using forward substitution

$$\begin{aligned} y_1 &= 4, \\ y_2 &= 8 + \frac{2}{3}(4) = \frac{32}{3}, \\ y_3 &= 6 - \frac{1}{3}(4) - \frac{1}{4}\left(\frac{32}{3}\right) = 2. \end{aligned}$$

Solve $U\mathbf{x} = \mathbf{y}$

$$\begin{pmatrix} 6 & 3 & -1 \\ 0 & 4 & \frac{4}{3} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ \frac{32}{3} \\ 2 \end{pmatrix},$$

using back substitution

$$\begin{aligned} x_3 &= 2, \\ x_2 &= \frac{1}{4}\left[\frac{32}{3} - \frac{4}{3}(2)\right] = 2, \\ x_1 &= \frac{1}{6}[4 + 2 - 3(2)] = 0. \end{aligned}$$

□

Example 7.5.2 (Worked example). Solve the following linear system using LUP factorisation:

$$\begin{aligned} x_2 - 2x_3 &= -4, \\ x_1 + 2x_3 &= 6, \\ 3x_1 - 2x_2 + 2x_3 &= 18. \end{aligned}$$

□

7.6 Tutorial exercises: LU factorisation

1. Consider the following matrix
$$A = \begin{pmatrix} 2 & 3 & -1 \\ 4 & 9 & -1 \\ 0 & 3 & 2 \end{pmatrix}.$$
Calculate the LU factorisation of A using elementary matrices.

2. Solve the following linear system using LU factorisation:
$$\begin{aligned} 2x_1 &+ 3x_2 - x_3 &= 4, \\ 4x_1 &+ 9x_2 - x_3 &= 18, \\ &3x_2 + 2x_3 &= 11. \end{aligned}$$

3. Consider the following matrix
$$A = \begin{pmatrix} 3 & 9 & 5 \\ 1 & 2 & 2 \\ 2 & 4 & 5 \end{pmatrix}.$$
Calculate the LUP factorisation of A using elementary matrices.

4. Solve the following linear system using LUP factorisation:
$$\begin{aligned} 3x_1 &+ 9x_2 + 5x_3 &= 20, \\ x_1 &+ 2x_2 + 2x_3 &= 3, \\ 2x_1 &+ 4x_2 + 5x_3 &= 4. \end{aligned}$$

The solutions can be found on page 163.

Chapter 8

Cholesky factorisation

We saw in Chapter 7 that solving a linear system using LU or LUP factorisation uses two thirds of the computational effort when solving the same system using Gaussian elimination. *Cholesky factorisation* (also known as *Cholesky decomposition*) is a method of factorising a particular type of coefficient matrix (called a positive definite matrix) so that the solution of the linear system only uses one third of the computational effort when using Gaussian elimination.

8.1 Positive definite matrices

Definition 8.1.1. An $n \times n$ matrix A is said to be *positive definite* if it is symmetric and $\mathbf{x}^T A\mathbf{x}$ is positive for any non-zero column vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$.

Theorem 8.1.1. *All of the eigenvalues of a positive definite matrix are positive.*

Proof. Let A be a positive definite matrix and λ is an eigenvalue of A. Given a non-zero eigenvector $\mathbf{x}$ of A then

$$\mathbf{x}^T A\mathbf{x} > 0.$$

The definition of an eigenvector, Eq. (6.2), is $A\mathbf{x} = \lambda\mathbf{x}$ therefore

$$\mathbf{x}^T \lambda\mathbf{x} = \lambda(\mathbf{x}^T\mathbf{x}) > 0.$$

Since $\mathbf{x}^T\mathbf{x} > 0$ then $\lambda > 0$. □

Theorem 8.1.2. *Given a non-singular square matrix A, the matrix $A^T A$ is positive definite.*

Proof. Using the definition of a symmetric matrix, $A = A^T$ then

$$A^T A = (A^T A)^T = (A^T)^T A^T = AA^T = A^T A,$$

therefore $A^T A$ is symmetric. Let $\mathbf{x}$ be a non-zero column vector and using the definition of a positive definite matrix $\mathbf{x}^T A\mathbf{x} > 0$ then

$$\begin{aligned} \mathbf{x}^T(A^T A)\mathbf{x} &> 0 \\ (\mathbf{x}A)^T A\mathbf{x} &> 0 \\ A\mathbf{x} \cdot A\mathbf{x} &> 0. \end{aligned}$$

Since the dot product of the vector $A\mathbf{x}$ with itself is always positive then $A^T A$ is positive definite. □

Definition 8.1.2. A *leading principal minor* of an $n \times n$ matrix A is the determinant of an $k \times k$ matrix formed by taking the first k rows and columns of A where $k = 1, \ldots, n$.

Example 8.1.1. Calculate the leading minors of the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

A is a 3×3 matrix so it has three leading principal minors. The first leading principal minor is the determinant of the matrix formed by taking the first row and column of A:

$$\begin{vmatrix} 1 \end{vmatrix} = 1.$$

Note that the first leading principal minor of a matrix A is simply the element a_{11}. The second leading minor is the determinant of the matrix formed by taking the first two rows and columns of A:

$$\begin{vmatrix} 1 & 2 \\ 4 & 5 \end{vmatrix} = 5 - 8 = -3.$$

The third principal minor is the determinant of A itself:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 2\begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} + 3\begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} = -3 - 2(-6) + 3(-3) = 0.$$

□

Theorem 8.1.3 (Sylvestor's criterion). *An $n \times n$ matrix A is positive definite if and only if all of its leading principal minors are all positive.*

Example 8.1.2. Show that the following matrix is positive definite:

$$A = \begin{pmatrix} 4 & 3 & 1 \\ 3 & 5 & 3 \\ 1 & 3 & 3 \end{pmatrix}.$$

Using Sylvestor's criterion we need to show that the principal minors are all positive

$$\begin{aligned} \begin{vmatrix} 4 \end{vmatrix} &= 4 > 0, \\ \begin{vmatrix} 4 & 3 \\ 3 & 5 \end{vmatrix} &= 20 - 9 = 11 > 0, \\ \begin{vmatrix} 4 & 3 & 1 \\ 3 & 5 & 3 \\ 1 & 3 & 3 \end{vmatrix} &= 4\begin{vmatrix} 5 & 3 \\ 3 & 3 \end{vmatrix} - 3\begin{vmatrix} 3 & 3 \\ 1 & 3 \end{vmatrix} + \begin{vmatrix} 3 & 5 \\ 1 & 3 \end{vmatrix} = 4(6) - 3(6) + 4 = 10 > 0. \end{aligned}$$

Since all of the leading principal minors are positive, we can say that A is a positive definite matrix using Sylvestor's criterion. □

8.2 Cholesky factorisation

Cholesky factorisation, named after French mathematician André-Louis Cholesky (1875–1918), is a procedure for factorising a matrix A into the product of a lower triangular matrix L and its transpose, i.e., $A = LL^T$. or

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} & l_{41} \\ 0 & l_{22} & l_{32} & l_{42} \\ 0 & 0 & l_{33} & l_{43} \\ 0 & 0 & 0 & l_{44} \end{pmatrix}.$$

Cholesky factorisation is only possible if A is positive definite. The computational complexity of calculating the Cholesky factorisation is $O(n/3)$. LU factorisation has a computational complexity of $O(2n/3)$ so solving using Cholesky factorisation incurs half the computational cost of LU factorisation.

To factorise a small matrix ($n \leq 3$) using Cholesky factorisation we can multiply out the individual elements and solve. For example, consider the matrix A

$$A = \begin{pmatrix} 4 & 6 \\ 6 & 10 \end{pmatrix}.$$

The leading principal minors are 4 and 4 which are both positive, therefore this matrix is positive definite by Sylvestor's criterion. Writing $A = LL^T$ we have

$$\begin{pmatrix} 4 & 6 \\ 6 & 10 \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{pmatrix},$$

therefore

$$\begin{aligned} 4 &= l_{11}^2, \\ 6 &= l_{11} l_{21}, \\ 10 &= l_{21}^2 + l_{22}^2. \end{aligned}$$

This is easily solved to give $l_{11} = 2$, $l_{21} = 3$ and $l_{22} = 1$ so the Cholesky factorisation of A is

$$L = \begin{pmatrix} 2 & 0 \\ 3 & 1 \end{pmatrix}.$$

Example 8.2.1 (Worked example). Calculate the Cholesky factorisation of the following matrix

$$A = \begin{pmatrix} 9 & 6 & 6 \\ 6 & 5 & 4 \\ 6 & 4 & 13 \end{pmatrix}.$$

□

8.3 The Cholesky-Crout algorithm

For larger matrices, the calculation of the Cholesky factorisation can be done using a version of Crout's algorithm, hereafter known as the Cholesky-Crout algorithm. Similar to Crout's algorithm for LU factorisation, the elements of L are calculated in column-by-column order.

Theorem 8.3.1 (Cholesky-Crout algorithm). *If A is a positive definite matrix then it can be expressed using Cholesky factorisation as a product of a lower triangular matrix L and its transpose such that*

$$A = LL^T,$$

where

$$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}, \tag{8.1a}$$

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right). \tag{8.1b}$$

Proof. Applying matrix multiplication to a lower diagonal matrix L and its transpose L^T and equating to a positive definite matrix A gives

$$\begin{aligned}
\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots \\ a_{21} & a_{22} & a_{23} & \cdots \\ a_{31} & a_{32} & a_{33} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} &= \begin{pmatrix} l_{11} & 0 & 0 & \cdots \\ l_{21} & l_{22} & 0 & \cdots \\ l_{31} & l_{32} & l_{33} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} & \cdots \\ 0 & l_{22} & l_{32} & \cdots \\ 0 & 0 & l_{33} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \\
&= \begin{pmatrix} l_{11}^2 & l_{11}l_{21} & l_{11}l_{31} & \cdots \\ l_{11}l_{21} & l_{21}^2 + l_{22}^2 & l_{21}l_{31} + l_{22}l_{33} & \cdots \\ l_{11}l_{31} & l_{21}l_{31} + l_{22}l_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.
\end{aligned}$$

The elements on the main diagonal of A are

$$\begin{aligned}
a_{jj} &= \sum_{k=1}^{j} l_{jk}^2 \\
&= l_{jj}^2 + \sum_{k=1}^{j-1} l_{jk}^2,
\end{aligned}$$

therefore

$$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2},$$

which is Eq. (8.1a). Note that since A is positive definite then the expression under the square root is always real and positive. The elements off the main diagonal are

$$\begin{aligned}
a_{ij} &= \sum_{k=1}^{j} l_{jk} l_{ik} \\
&= l_{jj} l_{ij} + \sum_{k=1}^{j-1} l_{ik} l_{jk},
\end{aligned}$$

therefore

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right),$$

which is Eq. (8.1b). □

Algorithm 4 Cholesky-Crout algorithm

Require: An $n \times n$ positive definite matrix A.
  for $j = 1, \ldots, n$ **do** ▷ loop through columns of A
    $l_{jj} \leftarrow \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$ ▷ calculate main diagonal element
    for $i = j+1, \ldots, n$ **do** ▷ loop through lower triangular elements
      $l_{ij} \leftarrow \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right)$ ▷ calculate lower triangular elements
    end for
  end for
  return L.

Example 8.3.1. Calculate the Cholesky factorisation of the matrix

$$A = \begin{pmatrix} 16 & 12 & -4 \\ 12 & 13 & 1 \\ -4 & 1 & 21 \end{pmatrix}.$$

Using the Cholesky-Crout algorithm:

$$\begin{aligned}
l_{11} &= \sqrt{a_{11}} = 4, \\
l_{21} &= \frac{1}{l_{11}}(a_{21}) = \frac{1}{4}(12) = 3, \\
l_{31} &= \frac{1}{l_{11}}(a_{31}) = \frac{1}{4}(-4) = -1, \\
l_{22} &= \sqrt{a_{22} - l_{21}^2} = \sqrt{13 - 9} = 2, \\
l_{32} &= \frac{1}{l_{22}}(a_{32} - l_{31} l_{21}) = \frac{1}{2}(1 + 3) = 2, \\
l_{33} &= \sqrt{a_{33} - l_{31}^2 - l_{32}^2} = \sqrt{21 - 1 - 4} = 4.
\end{aligned}$$

Therefore the Cholesky factorisation of A is

$$L = \begin{pmatrix} 4 & 0 & 0 \\ 3 & 2 & 0 \\ -1 & 2 & 4 \end{pmatrix}.$$

We can check whether this is correct by calculating $LL^T = A$

$$\begin{pmatrix} 4 & 0 & 0 \\ 3 & 2 & 0 \\ -1 & 2 & 4 \end{pmatrix} \begin{pmatrix} 4 & 3 & -1 \\ 0 & 2 & 2 \\ 0 & 0 & 4 \end{pmatrix} = \begin{pmatrix} 16 & 12 & -4 \\ 12 & 13 & 1 \\ -4 & 1 & 21 \end{pmatrix} = A.$$

□

Example 8.3.2 (Worked example). Calculate the Cholesky factorisation of the matrix using the Cholesky-Crout algorithm

$$A = \begin{pmatrix} 4 & -6 & 4 & 6 \\ -6 & 10 & -2 & -11 \\ 4 & -2 & 29 & -5 \\ 6 & -11 & -5 & 30 \end{pmatrix}.$$

□

8.3.1 Cholesky factorisation MATLAB code

The MATLAB code for calculating the Cholesky factorisation of a matrix A is given in Listing 8.1.

Listing 8.1: MATLAB function for calculating the Cholesky factorisation of the matrix A.

```
function L = cholesky(A)

% cholesky.m by Jon Shiach
%
% This function factorises the matrix A into the product of a lower
% diagonal matrix L and its transpose such that A = LL' using the
% Cholesky-Crout algorithm.

% calculate size of the system
A = [4 -6 4 6; -6 10 -2 -11; 4 -2 29 -5; 6 -11 -5 30];
n = size(A,1);

% check that the matrix A is positive definite
for i = 1 : n
    if det(A(1:i,1:i)) < 0
        error('Matrix A is not positive definite.');
    end
end

% initialise L
L = zeros(n);

% loop through the columns of A
for j = 1 : n

    % calculate diagonal element
    L(j,j) = A(j,j);
    for k = 1 : j - 1
        L(j,j) = L(j,j) - L(j,k)^2;
    end
    L(j,j) = sqrt(L(j,j));

    % loop through the elements under the main diagonal
    for i = j + 1 : n

        % calculate non-diagonal element
        L(i,j) = A(i,j);
        for k = 1 : j - 1
            L(i,j) = L(i,j) - L(j,k)*L(i,k);
        end
        L(i,j) = L(i,j)/L(j,j);
    end
end
```

8.4 Solving linear systems using Cholesky factorisation

Consider the solution of the linear system $A\mathbf{x} = \mathbf{b}$. Since $LL^T = A$ then

$$LL^T\mathbf{x} = \mathbf{b}.$$

Let $\mathbf{y} = L^T\mathbf{x}$ then

$$L\mathbf{y} = \mathbf{b},$$

and

$$L^T\mathbf{x} = \mathbf{y}.$$

To solve a system using Cholesky factorisation we do the following:

1. Determine the Cholesky factorisation of the coefficient matrix A;
2. Solve $L\mathbf{y} = \mathbf{b}$ using forward substitution;
3. Solve $L^T\mathbf{x} = \mathbf{y}$ using back substitution.

Example 8.4.1. Solve the following linear system using Cholesky factorisation

$$\begin{array}{rcrcrcr} 16x_1 & + & 12x_2 & - & 4x_3 & = & 72, \\ 12x_1 & + & 13x_2 & & x_3 & = & 62, \\ -4x_1 & & x_2 & + & 21x_3 & = & -26. \end{array}$$

We saw in example 8.3.1 that the Cholesky factorisation of this coefficient matrix is

$$L = \begin{pmatrix} 4 & 0 & 0 \\ 3 & 2 & 0 \\ -1 & 2 & 4 \end{pmatrix}.$$

Solve $L\mathbf{y} = \mathbf{b}$

$$\begin{pmatrix} 4 & 0 & 0 \\ 3 & 2 & 0 \\ -1 & 2 & 4 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 72 \\ 62 \\ -26 \end{pmatrix},$$

using forward substitution

$$\begin{aligned} y_1 &= \frac{72}{4} = 18, \\ y_2 &= \frac{1}{2}[62 - 3(18)] = 4, \\ y_3 &= \frac{1}{4}[-26 + 18 - 2(4)] = -4. \end{aligned}$$

Solve $L^T\mathbf{x} = \mathbf{y}$

$$\begin{pmatrix} 4 & 3 & -1 \\ 0 & 2 & 2 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 18 \\ 4 \\ -4 \end{pmatrix},$$

using back substitution

$$\begin{aligned} x_3 &= \frac{-4}{4} = -1, \\ x_2 &= \frac{1}{2}[4 - 2(-1)] = 3, \\ x_1 &= \frac{1}{4}[18 - 3(3) - 1] = 2. \end{aligned}$$

□

Example 8.4.2 (Worked example). Solve the following linear system using Cholesky factorisation

$$\begin{aligned} 4x_1 - 6x_2 + 4x_3 + 6x_4 &= 40, \\ -6x_1 + 10x_2 - 2x_3 - 11x_4 &= -52, \\ 4x_1 - 2x_2 + 29x_3 - 5x_4 &= 93, \\ 6x_1 - 11x_2 - 5x_3 + 30x_4 &= 69. \end{aligned}$$

□

8.5 Tutorial exercises: Cholesky factorisation

1. Determine whether the following matrices are positive definite?

   (a) $A = \begin{pmatrix} 5 & 3 & 1 \\ 3 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix}$;

   (b) $B = \begin{pmatrix} 2 & 0 & -2 & 1 \\ 0 & 3 & 0 & 1 \\ -2 & 0 & 3 & 0 \\ 1 & 1 & 0 & 2 \end{pmatrix}$.

2. Find the range of values of α so that the following matrix is positive definite:

$$A = \begin{pmatrix} 2 & \alpha & -1 \\ \alpha & 2 & 1 \\ -1 & 1 & 4 \end{pmatrix}.$$

3. Calculate the Cholesky factorisation of the following matrices

   (a) $A = \begin{pmatrix} 4 & -6 \\ -6 & 10 \end{pmatrix}$;

   (b) $B = \begin{pmatrix} 16 & 12 & 20 \\ 12 & 18 & 21 \\ 20 & 21 & 33 \end{pmatrix}$;

   (c) $C = \begin{pmatrix} 4 & -4 & 2 & 8 \\ -4 & 29 & 8 & 2 \\ 2 & 8 & 6 & 11 \\ 8 & 2 & 11 & 30 \end{pmatrix}$;

4. Solve the following linear system using Cholesky factorisation:

$$\begin{aligned} x_1 + 3x_2 - 2x_3 &= -5, \\ 3x_1 + 13x_2 &= -3, \\ -2x_1 + 17x_3 &= 36. \end{aligned}$$

5. Download the MATLAB file `cholesky.m` from moodle (also given in Section 8.3.1 on page 114) and save to your own area on the network. Use this function to calculate the Cholesky factorisation of the coefficient matrix for the following linear system and hence calculate the solution.

$$\begin{aligned} x_1 + 2x_2 + 4x_3 + 7x_4 + 11x_5 &= 50, \\ 2x_1 + 13x_2 + 23x_3 + 38x_4 + 58x_5 &= 265, \\ 4x_1 + 23x_2 + 77x_3 + 122x_4 + 182x_5 &= 769, \\ 7x_1 + 38x_2 + 122x_3 + 294x_4 + 430x_5 &= 1571, \\ 11x_1 + 58x_2 + 182x_3 + 430x_4 + 855x_5 &= 2548. \end{aligned}$$

The solutions can be found on page 164.

Chapter 9

Indirect methods for solving linear systems

The methods for solving linear systems seen in the previous chapters are all known as *direct methods* because a single application of the method is needed to calculate the solution. If the solution obtained by a direct method deviates from the actual solution, there is no way in which to refine the solution to improve its accuracy. Numerical methods for solving linear systems commonly use iterative methods where a solution is refined over a series of iterations to obtain a solution at a required accuracy. Iterative methods are known as *indirect methods*. Indirect methods are rarely used for small systems since the direct methods are more computationally efficient, however, for larger systems indirect methods are much more computationally efficient and require less memory.

9.1 The Jacobi method

9.1.1 Matrix form of the Jacobi method

An indirect method can be written in matrix form such as

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}, \tag{9.1}$$

where T is known as the *iteration matrix* and $\mathbf{c}$ is some vector. Whilst writing indirect methods in this way is not particularly useful for performing the calculations, it allows us to analyse the stability and convergence of a method.

The Jacobi method can be written in matrix form by splitting the coefficient matrix A into lower triangular, diagonal, and upper triangular matrices, L, D and U respectively, such that $A = D - L - U$, e.g.,

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}, \qquad D = \begin{pmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{pmatrix},$$

$$L = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -a_{21} & 0 & 0 & 0 \\ -a_{31} & -a_{32} & 0 & 0 \\ -a_{41} & -a_{42} & -a_{43} & 0 \end{pmatrix}, \qquad U = \begin{pmatrix} 0 & -a_{12} & -a_{13} & -a_{14} \\ 0 & 0 & -a_{23} & -a_{24} \\ 0 & 0 & 0 & -a_{34} \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Rewriting the linear system $A\mathbf{x} = \mathbf{b}$ using L, D and U and rearranging gives

$$\begin{aligned}(D - L - U)\mathbf{x} &= \mathbf{b}\\ -(L + U)\mathbf{x} + D\mathbf{x} &= \mathbf{b}\\ D\mathbf{x} &= (L + U)\mathbf{x} + \mathbf{b}\\ \mathbf{x} &= D^{-1}(L + U)\mathbf{x} + D^{-1}\mathbf{b}.\end{aligned}$$

Let the $\mathbf{x}$ vector on the left-hand side be the values at iteration k and the $\mathbf{x}$ vector on the right-hand side be the previous values at iteration $k - 1$, then the Jacobi method written in matrix form is

$$\mathbf{x}^{(k)} = D^{-1}(L + U)\mathbf{x}^{(k-1)} + D^{-1}\mathbf{b}. \tag{9.2}$$

Hence the iteration matrix for the Jacobi method is

$$T_J = D^{-1}(L + U). \tag{9.3}$$

9.1.2 The residual

Computational methods that use iteration to refine a solution require some kind of convergence criteria that will halt the iterations once the desired accuracy is obtained. The problem is that in most cases, the exact solution of the system is unknown so we cannot calculate the error between the estimated solution and the exact solution. Fortunately we can calculate a vector that is related to the error thus giving some idea of the accuracy of the current solution. This vector is called the *residual*.

Consider the linear system

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

If $\tilde{\mathbf{x}}$ is an approximation of the exact solution $\mathbf{x}$ then the vector of errors is defined by

$$\mathbf{e} = \mathbf{x} - \tilde{\mathbf{x}},$$

therefore

$$A\mathbf{e} = A(\mathbf{x} - \tilde{\mathbf{x}}) = A\mathbf{x} - A\tilde{\mathbf{x}} = \mathbf{b} - A\tilde{\mathbf{x}},$$

which leads to the definition of the residual.

Definition 9.1.1. The residual is defined by

$$\mathbf{r} = \mathbf{b} - A\tilde{\mathbf{x}}, \tag{9.4}$$

where $\tilde{\mathbf{x}}$ is an approximation of the solution to the linear system $A\mathbf{x} = \mathbf{b}$. Note that $\tilde{\mathbf{x}} = \mathbf{x}$ only if $\mathbf{r} = \mathbf{0}$.

We can use the norm of the residual to give a scalar value indicating the accuracy of our solutions. The convergence criteria used is

$$|\mathbf{r}| < tol$$

where *tol* is some small value. The smaller the value of the tolerance the more accurate the solution but more iterations will be needed. However, setting the tolerance too high will give quicker convergence but with loss of accuracy.

Example 9.1.1. Use the Jacobi method to solve the following linear system with starting values $x_i^{(0)} = 0$ iterating until $|\mathbf{r}| < 10^{-4}$

$$\begin{aligned} 7x_1 &+ 2x_2 - 3x_3 &= -2, \\ -2x_1 &+ 5x_2 + x_3 &= -7.5, \\ 2x_1 &- 2x_2 - 6x_3 &= -7. \end{aligned}$$

The Jacobi iterations are

$$\begin{aligned} x_1^{(k)} &= \frac{1}{7}\left(-2 - 2x_2^{(k-1)} + 3x_3^{(k-1)}\right), \\ x_2^{(k)} &= \frac{1}{5}\left(-7.5 + 2x_1^{(k-1)} - x_3^{(k-1)}\right), \\ x_3^{(k)} &= -\frac{1}{6}\left(-7 - 2x_1^{(k-1)} + 2x_2^{(k-1)}\right). \end{aligned}$$

The first iteration gives

$$\begin{aligned} x_1^{(1)} &= \frac{1}{7}\left(-2 - 2x_2^{(0)} + 3x_3^{(0)}\right) = \frac{1}{7}[-2 - 2(0) + 3(0)] = -0.28571, \\ x_2^{(1)} &= \frac{1}{5}\left(-7.5 + 2x_1^{(0)} - x_3^{(0)}\right) = \frac{1}{5}[-7.5 + 2(0) - 0] = -1.50000, \\ x_3^{(1)} &= -\frac{1}{6}\left(-7 - 2x_1^{(0)} + 2x_2^{(0)}\right) = -\frac{1}{6}[-7 - 2(0) + 2(0)] = 1.16667. \end{aligned}$$

Calculating the residual gives

$$\mathbf{r} = \begin{pmatrix} -2 \\ -7.5 \\ -7 \end{pmatrix} - \begin{pmatrix} 7 & 2 & -3 \\ -2 & 5 & 1 \\ 2 & -2 & -6 \end{pmatrix} \begin{pmatrix} -0.28571 \\ -1.50000 \\ 1.16667 \end{pmatrix} = \begin{pmatrix} -2 \\ -7.5 \\ -7 \end{pmatrix} - \begin{pmatrix} -8.50000 \\ -5.76191 \\ -4.57143 \end{pmatrix} = \begin{pmatrix} 6.50000 \\ -1.73095 \\ -2.42857 \end{pmatrix},$$

therefore $|\mathbf{r}| = \sqrt{6.5^2 + 1.73095^2 + 2.42857^2} = 7.15325$. The second iteration is then

$$\begin{aligned} x_1^{(2)} &= \frac{1}{7}\left(-2 - 2x_2^{(1)} + 3x_3^{(1)}\right) = \frac{1}{7}[-2 - 2(-1.50000) + 3(1.16667)] = 0.64286, \\ x_2^{(2)} &= \frac{1}{5}\left(-7.5 + 2x_1^{(1)} - x_3^{(1)}\right) = \frac{1}{5}[-7.5 + 2(-0.28571) - 1.16667] = -1.84762, \\ x_3^{(2)} &= \frac{1}{6}\left[-7 - 2x_1^{(1)} + 2x_2^{(1)}\right) = \frac{1}{6}[-7 - 2(-0.28571) + 2(-1.50000)] = 1.57143. \end{aligned}$$

Calculating the residual gives

$$\mathbf{r} = \begin{pmatrix} -2 \\ -7.5 \\ -7 \end{pmatrix} - \begin{pmatrix} 7 & 2 & -3 \\ -2 & 5 & 1 \\ 2 & -2 & -6 \end{pmatrix} \begin{pmatrix} 0.64286 \\ -1.84762 \\ 1.57143 \end{pmatrix} = \begin{pmatrix} -2 \\ -7.5 \\ -7 \end{pmatrix} - \begin{pmatrix} -3.90951 \\ -8.95239 \\ -4.44762 \end{pmatrix} = \begin{pmatrix} 1.90951 \\ 1.45239 \\ -2.55238 \end{pmatrix},$$

therefore $|\mathbf{r}| = \sqrt{1.90951^2 + 1.45239^2 + 2.55258^2} = 3.50290$. As you can see, the calculation by hand of the iteration values for an indirect method is time consuming and tedious. A MATLAB program has been written to carry out the calculations and the iteration values for $x_i^{(k)}$ are shown in Table 9.1.

Here it took 14 iterations to converge to a solution where $|\mathbf{r}| < 10^{-4}$. Comparing the final two iterations shows that we can state the solution correct to four decimal places $x_1 = 1.0000$, $x_2 = -1.5000$ and $x_3 = 2.0000$. □

Table 9.1: The Jacobi iterations for Example 9.1.1.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\|\mathbf{r}\|$
0	0.00000	0.00000	0.00000	10.45227
1	-0.28571	-1.50000	1.16667	7.15325
2	0.64286	-1.84762	1.57143	3.50290
3	0.91565	-1.55714	1.99683	0.70644
4	1.01497	-1.53311	1.99093	0.26235
5	1.00557	-1.49220	2.01602	0.10995
6	1.00464	-1.50098	1.99926	0.03925
7	0.99996	-1.49800	2.00187	0.01953
8	1.00023	-1.50039	1.99932	0.00679
9	0.99982	-1.49977	2.00021	0.00303
10	1.00002	-1.50011	1.99986	0.00137
11	0.99997	-1.49996	2.00005	0.00055
12	1.00001	-1.50002	1.99998	0.00024
13	1.00000	-1.49999	2.00001	0.00010
14	1.00000	-1.50000	2.00000	0.00004

9.1.3 MATLAB code

The function below in Listing 9.1 calculates the solution of a linear system $A\mathbf{x} = \mathbf{b}$ using the Jacobi method.

Listing 9.1: MATLAB function for solving the linear system $A\mathbf{x} = \mathbf{b}$ using the Jacobi method.

```
%function x = jacobi(A,b,tol)

A = [4 2 1; -2 5 1; 1 0 3]
b = [3; 4; 5]
tol = 10^-4

% jacobi.m by Jon Shiach
%
% This function uses the Jacobi method to solve the linear system Ax = b.
% Iterations are calculated until the norm of the residual is less than
% tol.

% calculate the size of the system and initialise x, xkm1 and r
n = size(A,1);
x = zeros(n,1);
xkm1 = x;
r = norm(b - A*x);

% output column headings and starting values
hline = repmat('-',1,5+11*(n+1));       % horizontal line
fprintf('\n%s\n  k  ',hline)
fprintf('|    x(%1i)  ',[1:n])
fprintf('|   |r|\n')
fprintf('%s\n',hline)
fprintf(' %3i ',0)
fprintf('| %8.5f ',x)
fprintf('| %8.5f\n',r)

% perform iterations until convergence is reached
k = 0;
while r > tol
```

```
    % use Jacobi method to calculate new values of x
    for i = 1 : n
        x(i) = b(i);
        for j = 1 : n
            if i ~= j
                x(i) = x(i) - A(i,j)*xkm1(j);
            end
        end
        x(i) = x(i)/A(i,i);
    end

    % calculate residual
    r = norm(b - A*x);

    % update iteration counter and xkm1
    k = k + 1;
    xkm1 = x;

    % output current estimate
    fprintf(' %3i ',k)
    fprintf('| %8.5f ',x)
    fprintf('| %8.5f\n',r)
end

fprintf('%s\n',hline)
```

9.2 The Gauss-Seidel method

In the Jacobi method, the values of $x_i^{(k)}$ are calculated from the values obtained at the previous iteration $x_i^{(k-1)}$ for $i = 1, \ldots, n$. The speed of convergence of the method can be improved by using the values of $x_j^{(k)}$ where $j < i$ as soon as they are known to calculate $x_i^{(k)}$. This method is known as the Gauss-Seidel method.

Consider the linear system

$$\begin{array}{ccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1n}x_n & = & b_1, \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + & \cdots & + & a_{2n}x_n & = & b_2, \\ a_{31}x_1 & + & a_{32}x_2 & + & a_{33}x_3 & + & \cdots & + & a_{3n}x_n & = & b_3, \\ \vdots & & \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & a_{m3}x_3 & + & \cdots & + & a_{mn}x_n & = & b_n. \end{array} \tag{9.5}$$

The first equation can be rearranged to give the iterative scheme

$$x_1^{(k)} = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)} - \ldots - a_{1n}x_n^{(k-1)}\right),$$

which is equivalent to the Jacobi solution for $x_1^{(k)}$. The remaining equations in Eq. (9.5) can be

written using the newly calculated values of $x_j^{(k)}$ where $j < i$, i.e.,

$$
\begin{aligned}
x_2^{(k)} &= \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k-1)} - a_{24}x_4^{(k-1)} - \dots a_{2n}x_n^{(k-1)}\right),\\
x_3^{(k)} &= \frac{1}{a_{33}}\left(b_3 - a_{31}x_1^{(k)} - a_{32}x_3^{(k)} - a_{34}x_4^{(k-1)} - a_{35}x_5^{(k-1)} - \dots - a_{3n}x_n^{(k-1)}\right),\\
&\vdots\\
x_i^{(k)} &= \frac{1}{a_{ii}}\left(b_i - a_{i1}x_i^{(k)} - \dots - a_{i,i-1}x_{i-1}^{(k)} - a_{i,i+1}x_{i+1}^{(k-1)} - \dots - a_{i,n}x_n^{(k-1)}\right),\\
&\vdots\\
x_n^{(k)} &= \frac{1}{a_{nn}}\left(b_n - a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} - a_{n3}x_3^{(k)} - \dots - a_{n,n-1}x_{n-1}^{(k)}\right).
\end{aligned}
$$

Therefore, the Gauss-Seidel iterative scheme can be written as

$$x_i^{(k)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right). \tag{9.6}$$

The Gauss-Seidel method for solving a linear system of the form $A\mathbf{x} = \mathbf{b}$ is outlined in Algorithm 5. Note that the Gauss-Seidel method does not require the old values $\mathbf{x}^{(k-1)}$ to be saved since it updates $\mathbf{x}$ using the most recent known values.

Algorithm 5 Gauss-Seidel method

Require: An $n \times n$ coefficient matrix A and right-hand side vector $\mathbf{b}$ for a linear system $A\mathbf{x} = \mathbf{b}$ and a convergence tolerance *tol*.
  $\mathbf{x} = (0, \dots, 0)^T$
  $r \leftarrow |\mathbf{b} - A\mathbf{x}|$
  $k \leftarrow 0$
  while $r > tol$ **do**
    for $i = 1, \dots, n$ **do**
      $x_i \leftarrow \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j\right)$
    end for
    $r \leftarrow |\mathbf{b} - A\mathbf{x}|$
    $k \leftarrow k + 1$
  end while
  return x

9.2.1 Matrix form of the Gauss-Seidel method

The Gauss-Seidel method can be written in matrix form using a similar method seen in Section 9.1.1. Rewriting the linear system $A\mathbf{x} = \mathbf{b}$ in terms of the lower, diagonal and upper matrices L, D and U gives

$$(D - L - U)\mathbf{x} = \mathbf{b}.$$

The Gauss-Seidel method uses the elements from the lower triangular and diagonal matrices for the coefficients of $x_i^{(k)}$ and the upper diagonal elements for the coefficients of $x_i^{(k-1)}$. Therefore we can split the left-hand side to give

$$(D - L)\mathbf{x}^{(k)} - U\mathbf{x}^{(k-1)} = \mathbf{b}.$$

Rearranging results in the matrix form of the Gauss-Seidel method

$$\mathbf{x}^{(k)} = (D-L)^{-1}U\mathbf{x}^{(k-1)} + (D-L)^{-1}\mathbf{b}, \tag{9.7}$$

hence the iteration matrix for the Gauss-Seidel method is

$$T_{GS} = (D-L)^{-1}U. \tag{9.8}$$

Example 9.2.1. Use the Gauss-Seidel method to solve the following linear system with starting values $x_i^{(0)} = 0$ iterating until $\epsilon^{(k)} < 10^{-4}$

$$\begin{aligned} 7x_1 &+ 2x_2 - 3x_3 &= -2,\\ -2x_1 &+ 5x_2 + x_3 &= -7.5,\\ 2x_1 &- 2x_2 - 6x_3 &= -7. \end{aligned}$$

The Gauss-Seidel iterations are

$$\begin{aligned} x_1^{(k)} &= \frac{1}{7}\left(-2 - 2x_2^{(k-1)} + 3x_3^{(k-1)}\right),\\ x_2^{(k)} &= \frac{1}{5}\left(-7.5 + 2x_1^{(k)} - x_3^{(k-1)}\right),\\ x_3^{(k)} &= -\frac{1}{6}\left(-7 - 2x_1^{(k)} + 2x_2^{(k)}\right). \end{aligned}$$

The first iteration gives

$$\begin{aligned} x_1^{(1)} &= \frac{1}{7}\left(-2 - 2x_2^{(0)} + 3x_3^{(0)}\right) = \frac{1}{7}[-2 - 2(0) + 3(0)] = -0.28571,\\ x_2^{(1)} &= \frac{1}{5}\left(-7.5 + 2x_1^{(1)} - x_3^{(0)}\right) = \frac{1}{5}[-7.5 + 2(-0.28571) - 0] = -1.61429,\\ x_3^{(1)} &= -\frac{1}{6}\left(-7 - 2x_1^{(1)} + 2x_2^{(1)}\right) = -\frac{1}{6}[-7 - 2(-0.28571) + 2(-1.61429)] = 1.60952. \end{aligned}$$

Calculating the residual gives

$$\mathbf{r} = \begin{pmatrix} -2\\ -7.5\\ -7 \end{pmatrix} - \begin{pmatrix} 7 & 2 & -3\\ -2 & 5 & 1\\ 2 & -2 & -6 \end{pmatrix}\begin{pmatrix} -0.28571\\ -1.61429\\ 1.60952 \end{pmatrix} = \begin{pmatrix} -2\\ -7.5\\ -7 \end{pmatrix} - \begin{pmatrix} -10.05711\\ -5.89051\\ -6.99996 \end{pmatrix} = \begin{pmatrix} 8.05711\\ -1.60948\\ -0.00004 \end{pmatrix},$$

therefore $|\mathbf{r}| = \sqrt{8.05711^2 + 1.60948^2 + 0.00004^2} = 8.21633$. A MATLAB program was written to calculate the remaining iteration values which are shown in Table 9.2.

Table 9.2: The Gauss-Seidel iterations for Example 9.2.1.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\lvert\mathbf{r}\rvert$
0	0.00000	0.00000	0.00000	10.45227
1	-0.28571	-1.61429	1.60952	8.21633
2	0.86531	-1.47578	1.94703	0.80925
3	0.97038	-1.50125	1.99054	0.18663
4	0.99631	-1.49959	1.99863	0.02243
5	0.99930	-1.50001	1.99977	0.00440
6	0.99990	-1.49999	1.99997	0.00059
7	0.99998	-1.50000	1.99999	0.00011
8	1.00000	-1.50000	2.00000	0.00002

Here it took 8 iterations to converge to a solution where $|\mathbf{r}| < 10^{-4}$. Comparing the final two iterations we can state the solution correct to four decimal places as $x_1 = 1.0000$, $x_2 = -1.5000$ and $x_3 = 2.0000$. □

Example 9.2.2 (Worked example). Use the Gauss-Seidel method to solve the following linear system with starting values $x_i^{(0)} = 0$.

$$\begin{aligned} 10x_1 &- x_2 + x_3 = -16, \\ 3x_1 &- 10x_2 + 2x_3 = -18, \\ x_1 &- x_2 - 10x_3 = 2. \end{aligned}$$

The Gauss-Seidel iterations are:

Working to four decimal places, fill in the table below for the first four iterations of the Gauss-Seidel method.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\lvert\mathbf{r}\rvert$
0				
1				
2				
3				
4				

Therefore the solution correct to decimal places is

$$x_1 =$$

$$x_2 =$$

$$x_3 =$$

□

9.3 Convergence behaviour of iterative methods

It is not possible to calculate the specific computational complexity for an indirect method like we saw with the direct methods since we don't know how many iterations it will take to converge to a solution. However, we can tell if a method will converge for a particular linear system and make comparisons between the rate of which various indirect methods converge to a solution.

Theorem 9.3.1. *Let* $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ *be a sequence of vectors defined using the iterative scheme*

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}, \tag{9.9}$$

where $\mathbf{x}^{(k)}$ *is the value of the vector* $\mathbf{x}$ *at iteration* k*,* T *is an iteration matrix and* $\mathbf{c}$ *is any vector. The iterative scheme will converge to the solution of the linear system* $A\mathbf{x} = \mathbf{b}$ *if the norm of* T *is less than 1, i.e.,* $\|T\| < 1$*.*

Proof. If Eq. (9.9) converges to a solution, $\mathbf{x}^{(k)} \to \mathbf{x}$ as $k \to \infty$ then

$$\mathbf{x} = T\mathbf{x} + \mathbf{c}, \tag{9.10}$$

subtracting Eq. (9.10) from Eq. (9.9) gives

$$\mathbf{x}^{(k)} - \mathbf{x} = T\left(\mathbf{x}^{(k-1)} - \mathbf{x}\right). \tag{9.11}$$

Let $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}$ be the error between the computed solution and the actual solution at the kth iteration then Eq. (9.11) becomes

$$\mathbf{e}^{(k)} = T\mathbf{e}^{(k-1)} = T\left(T\mathbf{e}^{(k-2)}\right) = T^2\mathbf{e}^{(k-2)} = \ldots = T^k\mathbf{e}^{(0)}. \tag{9.12}$$

Recall the following rule for matrix norms given on page 81

$$\|AB\| \leq \|A\|\|B\|,$$

and applying to $\mathbf{e}^{(k)} = T^k\mathbf{e}^{(0)}$ results in

$$|\mathbf{e}^{(k)}| = |T^k\mathbf{e}^{(0)}| \leq \|T\|\|T^{k-1}\||\mathbf{e}^{(0)}| \leq \|T\|^2\|T^{(k-2)}\||\mathbf{e}^{(0)}| \leq \ldots \leq \|T\|^k|\mathbf{e}^{(0)}|. \tag{9.13}$$

Therefore if $\|T\| < 1$ then $\|T\|^k \to 0$ as $k \to \infty$ so $\mathbf{e}^{(k)} \to 0$ which means the $\mathbf{x}^{(k)} = \mathbf{x}$, i.e., if the norm of the iteration matrix is less than one, the iterative scheme in Eq. (9.9) will eventually converge to the solution. □

9.3.1 Rate of convergence

The rate of convergence is of interest because the faster an iterative scheme converges to a solution, the less computation effort is required. Consider the iteration of the error $\mathbf{e}$

$$\mathbf{e}^{(k)} = T\mathbf{e}^{(k-1)}, \tag{9.14}$$

where T has n independent eigenvectors $\mathbf{v}_i$ with corresponding eigenvalues λ_i. For the initial error we can write

$$\mathbf{e}^{(0)} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \ldots + \alpha_n\mathbf{v}_n = \sum_{i=1}^{n} \alpha_i\mathbf{v}_i,$$

where α_i is some scalar (Parnell, 2013). Iterating Eq. (9.14) gives (recall that $T\mathbf{v} = \lambda\mathbf{v}$)

$$\begin{aligned}
\mathbf{e}^{(1)} &= T\left(\sum_{i=1}^{n}\alpha_i\mathbf{v}_i\right) = \sum_{i=1}^{n}\alpha_i T\mathbf{v}_i = \sum_{i=1}^{n}\alpha_i\lambda_i\mathbf{v}_i,\\
\mathbf{e}^{(2)} &= T\left(\sum_{i=1}^{n}\alpha_i\lambda_i\mathbf{v}_i\right) = \sum_{i=1}^{n}\alpha_i\lambda_i T\mathbf{v}_i = \sum_{i=1}^{n}\alpha_i\lambda_i^2\mathbf{v}_i,\\
&\vdots\\
\mathbf{e}^{(k)} &= \sum_{i=1}^{n}\alpha_i\lambda^k\mathbf{v}_i.
\end{aligned}$$

If $|\lambda_1| > |\lambda_i|$ for $i = 2, \ldots, n$ then

$$\mathbf{e}^{(k)} = \alpha_1\lambda_1^k\mathbf{v}_1 + \sum_{i=2}^{n}\alpha_i\lambda_i^k\mathbf{v}_i = \lambda_1^k\left[\alpha_1\mathbf{v}_1 + \sum_{i=2}^{n}\alpha_i\left(\frac{\lambda_i}{\lambda_1}\right)^k\mathbf{v}_i\right],$$

since $\left(\dfrac{\lambda_i}{\lambda_1}\right)^k \to 0$ as $k \to \infty$ therefore

$$\mathbf{e}^{(k)} \approx \alpha_1\lambda_1^k\mathbf{v}_1.$$

This means that the error at the kth iteration varies by a factor of the kth power of the largest eigenvalue of T. Recall that the spectral radius is $\rho(T) = \max_i |\lambda_i|$, so the spectral radius of the iteration matrix is used as a measure of the rate of convergence of an iterative scheme (the lower the value of the spectral radius, the faster the convergence).

Example 9.3.1. Examine the rate of convergence of the Jacobi and Gauss-Seidel methods when used to solve the following linear system

$$\begin{pmatrix} 7 & 2 & -3\\ -2 & 5 & 1\\ 2 & -2 & -6 \end{pmatrix}\begin{pmatrix} x_1\\ x_2\\ x_3 \end{pmatrix} = \begin{pmatrix} -2\\ -7.5\\ -7 \end{pmatrix}.$$

For this system the matrices D, L and U are

$$D = \begin{pmatrix} 7 & 0 & 0\\ 0 & 5 & 0\\ 0 & 0 & -6 \end{pmatrix}, \qquad L = \begin{pmatrix} 0 & 0 & 0\\ 2 & 0 & 0\\ -2 & 2 & 0 \end{pmatrix}, \qquad U = \begin{pmatrix} 0 & -2 & 3\\ 0 & 0 & -1\\ 0 & 0 & 0 \end{pmatrix}.$$

The iteration matrices for the Jacobi and Gauss-Seidel methods are

$$\begin{aligned}
T_J &= D^{-1}(L+U) = \begin{pmatrix} 0 & -2/7 & 3/7\\ 2/5 & 0 & -1/5\\ 1/3 & -1/3 & 0 \end{pmatrix},\\
T_{GS} &= (D-L)^{-1}U = \begin{pmatrix} 0 & -2/7 & 3/7\\ 0 & -4/35 & -1/35\\ 0 & -2/35 & 16/105 \end{pmatrix}.
\end{aligned}$$

The eigenvalues of T_J are

$$\begin{aligned}
\lambda_1 &= -0.429,\\
\lambda_2 &= 0.2145 + 0.2069i,\\
\lambda_3 &= 0.2145 - 0.2069i,
\end{aligned}$$

so $\rho(T_J) = 0.429$. The eigenvalues of T_{GS} are

$$\lambda_1 = 0,$$
$$\lambda_2 = -0.1203,$$
$$\lambda_3 = 0.1584,$$

so $\rho(T_{GS}) = 0.1584$. This indicates that the Gauss-Seidel will converge faster than the Jacobi method for this system which was seen in examples 9.1.1 and 9.2.1.

Calculation of eigenvalues can be a tricky process so we can use computer packages to do the work for us. The MATLAB commands that were used to calculate T_J, T_{GS}, $\rho(T_J)$ and $\rho(T_{GS})$ is given below.

```
>> A = [7 2 -3 ; -2 5 1 ; 2 -2 -6];
>> D = diag(diag(A));
>> L = -tril(A,-1);
>> U = -triu(A,1);
>> TJ = inv(D)*(L + U);
>> rats(TJ)

ans =

       0            -2/7            3/7
     2/5              0            -1/5
     1/3            -1/3              0

>> TGS = inv(D - L)*U;
>> rats(TGS)

ans =

       0            -2/7            3/7
       0            -4/35          -1/35
       0            -2/35          16/105

>> eig(TJ)

ans =

  -0.4290 + 0.0000i
   0.2145 + 0.2069i
   0.2145 - 0.2069i

>> rho_TJ = max(abs(eig(TJ)))

rho_TJ =

    0.4290

>> eig(TGS)

ans =

         0
   -0.1203
    0.1584

>> rho_TGS = max(abs(eig(TGS)))

rho_TGS =
```

```
0.1584
```

□

9.3.2 Gershgorin's theorem

The problem with using the spectral radius to calculate the rate of convergence for an iterative scheme is that the calculation of the eigenvalues of the iteration matrix T is non-trivial for the majority of cases. However, can be make use of Gershgorin's theorem to give a bound on the value of the spectral radius of a matrix when the matrix is large

Theorem 9.3.2 (Gershgorin's theorem). *Let A be an $n \times n$ matrix and $R_i = \sum_{j \neq i} |a_{ij}|$ be the sum of the absolute values of the non-diagonal elements in the ith row. All of the eigenvalues of A are bounded by the circle $D(a_{ij}, R_i)$ with centre a_{ij} and radius R_i.*

Proof. Let λ be an eigenvalue of A with the corresponding eigenvector $\mathbf{x}$. Let x_i be the largest element in $\mathbf{x}$. Since $A\mathbf{x} = \lambda\mathbf{x}$ then

$$\sum_{j=1}^{n} a_{ij}x_j = \lambda x_i$$

$$a_{ii}x_i + \sum_{j=1, j\neq i}^{n} a_{ij}x_j = \lambda x_i$$

$$\sum_{j=1, j\neq i}^{n} a_{ij}x_j = x_i(\lambda - a_{ii}).$$

Taking the absolute values of both sides

$$|x_i||\lambda - a_{ii}| \leq \left| \sum_{j=1, j\neq i}^{n} a_{ij}x_j \right|.$$

Divide both sides by $|x_i|$ (note that $x_i \neq 0$)

$$|\lambda - a_{ii}| \leq \sum_{j=1, j\neq i}^{n} \left| \frac{a_{ij}x_j}{x_i} \right|,$$

since $\dfrac{x_j}{x_i} \leq 1$ then

$$|\lambda - a_{ii}| \leq \sum_{j=1, j\neq i}^{n} |a_{ij}| = R_i.$$

Therefore λ must lie within the circle centred at a_{ii} with radius $R_i = \sum_j |a_{ij}|$. □

Gershgorin's theorem is represented graphically in Fig. 9.1 where the circle $D(a_{ij}, R_i)$ is plotted on the complex plane.

Corollary 9.3.1. *The eigenvalues of an $n \times n$ matrix A are bounded by the circle $D(a_{ij}, R_j)$ with a centred a_{ij} and radius R_j where $R_j = \sum_{i \neq j} |a_{ij}|$ is the sum of the non-diagonal elements in the jth column.*

Proof. Same as the proof above with the summation changed to $\displaystyle\sum_{i=1, i\neq j}^{n} |a_{ij}|$. □

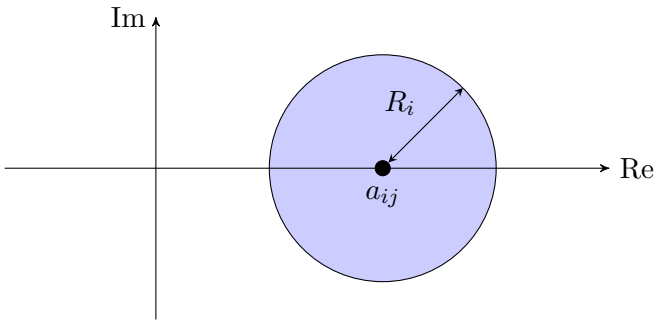


Figure 9.1: Graphical representation of the circle $D(a_{ij}, R_i)$ from Gershgorin's theorem.

We can use corollary 9.3.1 to reduce the bounds of the eigenvalues by choosing the smallest radii calculated using row or column elements.

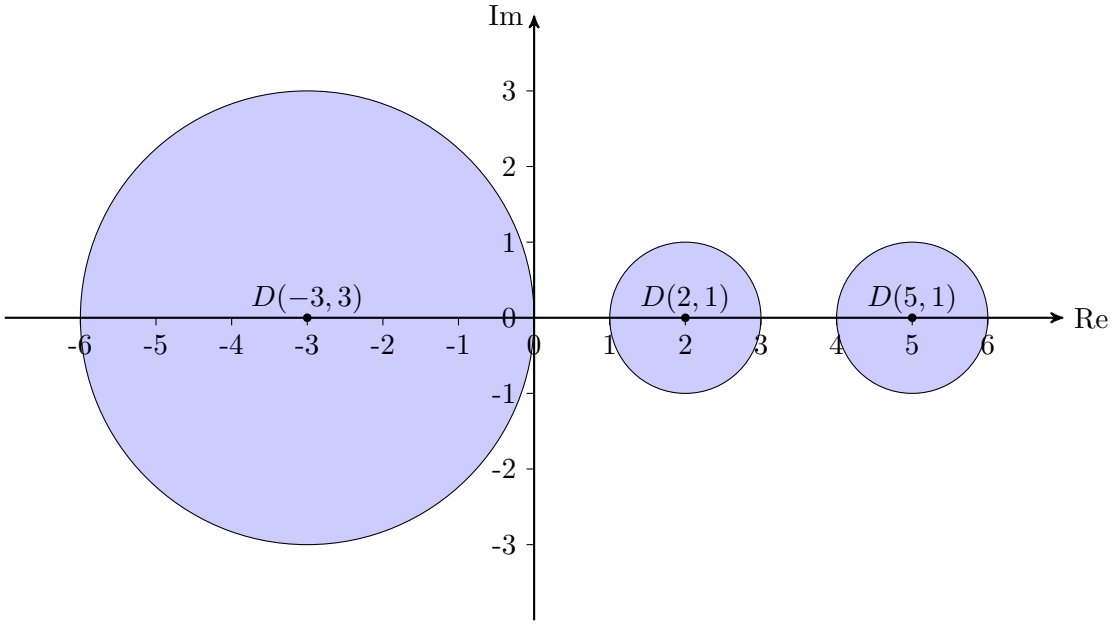
Example 9.3.2. Use Gershgorin's theorem to estimate an upper bound for the eigenvalues of the following matrix

$$A = \begin{pmatrix} 5 & -2 & 0 \\ 1 & -3 & 2 \\ 0 & -1 & 2 \end{pmatrix}.$$

Using Gershgorin's theorem

$$\begin{aligned} a_{11} &= 5, & R_1 &= \min(2+0, 1+0) = 1, \\ a_{22} &= -3, & R_2 &= \min(1+2, 2+1) = 3, \\ a_{33} &= 2, & R_3 &= \min(0+1, 0+2) = 1. \end{aligned}$$

Therefore the eigenvalues lie within the circles $D(5,1)$, $D(-3,3)$ and $D(2,1)$ (shown graphically below). The actual eigenvalues are $\lambda_1 = 4.7644$, $\lambda_2 = -2.25$ and $\lambda_3 = 1.4898$ (calculated using MATLAB) which are all in the circles found using Gershgorin's theorem.



□

Example 9.3.3. Use Gershgorin's theorem to compare the rate of convergence when using the Jacobi and Gauss-Seidel methods to solve the following linear system:

$$\begin{pmatrix} 7 & 2 & -3 \\ -2 & 5 & 1 \\ 2 & -2 & -6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -2 \\ -7.5 \\ -7 \end{pmatrix}.$$

It was seen in example 9.3.1 that

$$T_J = \begin{pmatrix} 0 & -2/7 & 3/7 \\ 2/5 & 0 & -1/5 \\ 1/3 & -1/3 & 0 \end{pmatrix}.$$

Using Gershgorin's theorem

$$\begin{aligned} a_{11} &= 0, & R_1 &= \min\left(\frac{2}{5}+\frac{1}{3}, \frac{2}{7}+\frac{3}{7}\right) = \frac{5}{7} = 0.714, \\ a_{22} &= 0, & R_2 &= \min\left(\frac{2}{5}+\frac{1}{5}, \frac{2}{7}+\frac{1}{3}\right) = \frac{3}{5} = 0.6, \\ a_{33} &= 0, & R_3 &= \min\left(\frac{1}{3}+\frac{1}{3}, \frac{3}{7}+\frac{1}{5}\right) = \frac{3}{5} = 0.6, \end{aligned}$$

therefore $|\lambda_1| \leq 0 + 0.714 = 0.714$, $|\lambda_2| \leq 0 + 0.6 = 0.6$ and $|\lambda_3| \leq 0 + 0.6$ so

$$\rho(T_J) \leq 0.714.$$

The iteration matrix for the Gauss-Seidel method is

$$T_{GS} = \begin{pmatrix} 0 & -2/7 & 3/7 \\ 0 & -4/35 & -1/35 \\ 0 & -2/35 & 16/105 \end{pmatrix}.$$

Using Gershgorin's theorem

$$\begin{aligned} R_1 &= \min\left(\frac{2}{7}+\frac{3}{7}, 0\right) = 0, \\ R_2 &= \min\left(\frac{1}{35}, \frac{2}{7}+\frac{2}{35}\right) = \frac{1}{35} = 0.0286 \\ R_3 &= \min\left(\frac{16}{105}, \frac{3}{7}+\frac{1}{25}\right) = \frac{2}{35} = 0.01571, \end{aligned}$$

therefore $|\lambda_1| \leq 0 + 0 = 0$, $|\lambda_2| \leq 0.1143 + 0.0286 = 0.1429$ and $|\lambda_3| \leq 0.1524 + 0.0571 = 0.2095$ so

$$\rho(T_{GS}) = \max(0.714, 0.6, 0.6) = 0.2095.$$

Comparing the upper bounds of the spectral radii shows that the Gauss-Seidel will converge faster than the Jacobi method. In Examples 9.1.1 and 9.2.1 we saw that the Jacobi and Gauss-Seidel methods took 14 and 8 iterations respectively to converge to a solution, which supports our anaylsis. □

9.3.3 MATLAB Code

The MATLAB function shown in Listing 9.2 calculates the upper bound of the spectral radius of a matrix `T` using Gershgorin's theorem.

Listing 9.2: MATLAB function that uses Gershgorin's theorem to estimate the upper bound of $\rho(T)$.

```
function rho = gershgorins(T)

% gershgorins.m by Jon Shiach
%
% This function approximates the spectal radius of a square matrix T using
% Geshgorin's theorem

n = size(T,1);
rho = 0;
for i = 1 : n
    R = min(sum(abs(T(:,i))),sum(abs(T(i,:)))) - abs(T(i,i));
    rho = max(abs(T(i,i)) + R,rho);
end
```

9.3.4 Gershgorin's theorem vs `eig`

For small matrices ($n < 100$), using MATLAB's `eig` command to calculate the spectral radius of a matrix will return an answer within a second. However, for larger matrices (which are common in practical problems) using this command requires significant computational resources and it is then that Gershgorin's theorem comes in useful. To demonstrate the efficiency of Gershgorin's theorem both the `eig` command and the `gershgorins` function given in Listing 9.2 was used to calculate the spectral radius of a random 1000×1000 matrix and the CPU time taken for each method was recorded and shown in Table 9.3

Table 9.3: Comparison between using the `eig` command and Gershgorin's theorem to calculate the spectral radius of a 1000×1000 matrix.

Method	$\rho(T)$	CPU time (s)	Normalised CPU time
`eig` command	500.37	0.8632	1.0000
Gershgorin's theorem	518.25	0.0162	0.0188

Although Gershgorin's theorem over estimated the value of the spectral radius but it is still a reasonable estimate. Gershgorin's theorem took less than 2% of the time that the `eig` command took to calculate the exact spectral radius. Since we use the spectral radius to compare convergence rates of indirect methods, it does not matter if the estimated value differs slightly from the exact value. This analysis was also conducted for a 10000×10000 random matrix and the `eig` command crashed the computer due to the computational resources required whereas Gershgorin's theorem returned an estimate in 1.7 seconds.

9.4 The Successive Over-Relaxation (SOR) method

The *Successive Over-Relaxation* or *SOR* method improves on the rate of convergence of the Gauss-Seidel method by over-relaxing the solution at every iteration. What this means is that the updated values of x_i are calculated using a weighted sum of the values from the previous iteration and the values calculated from the Gauss-Seidel method at the current iteration. To do this we introduce a *relaxation parameter* ω into the iterative scheme.

Consider a single equation from the generic linear system given in Eq. (9.5)

$$a_{i1}x_1 + \ldots + a_{i,i-1}x_{i-1} + a_{ii}x_i + a_{i,i+1}x_{i+1} + \ldots + a_{i,n}x_n = b_i,$$

using summation notation this can be written as

$$\sum_{j=1}^{i-1} a_{ij}x_j + a_{ii}x_i + \sum_{j=i+1}^{n} a_{ij}x_j = b_i. \tag{9.15}$$

To introduce the relaxation parameter ω, we split the coefficient a_{ii} into two parts such that

$$a_{ii} = \left(1 - \frac{1}{\omega}\right) a_{ii} + \frac{1}{\omega} a_{ii}, \tag{9.16}$$

where $\omega \in \mathbb{R}$, so Eq. (9.15) becomes

$$\sum_{j=1}^{i-1} a_{ij}x_j + \left[\left(1 - \frac{1}{\omega}\right) + \frac{1}{\omega}\right] a_{ii}x_i + \sum_{j=i+1}^{n} a_{ij} = b_i.$$

Rearranging and simplifying gives

$$\begin{aligned}
\left[\left(1 - \frac{1}{\omega}\right) + \frac{1}{\omega}\right] a_{ii}x_i &= b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j \\
(\omega - 1)a_{ii}x_i + a_{ii}x_i &= \omega b_i - \omega \sum_{j=1}^{i-1} a_{ij}x_j - \omega \sum_{j=i+1}^{n} a_{ij}x_j \\
a_{ii}x_i &= (1 - \omega)a_{ii}x_i + \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij} \right) \\
x_i &= (1 - \omega)x_i + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j \right).
\end{aligned}$$

Using the Gauss-Seidel method the first summation in the bracket is calculated using values $x_j^{(k)}$ where $j < i$ and the second summation is calculated using $x_j^{(k-1)}$ where $j > i$. Let the x_i term on the left-hand side be $x_i^{(k)}$ and the x_i term on the right-hand side by $x_i^{(k-1)}$ then

$$x_i^{(k)} = (1 - \omega)x_i^{(k-1)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} \right), \tag{9.17}$$

which is the SOR method. Since the term in the bracket is the Gauss-Seidel method, the SOR method is often written as follows

$$x_i^{(k)} = (1 - \omega)x_i^{(k-1)} + \omega \bar{x}_i^{(k)},$$

where $\bar{x}_i^{(k)}$ is the value of $x_i^{(k)}$ calculated using the Gauss-Seidel method.

Kahan (1958) proved that ω must lie in the interval [0,2] in order for the iterative scheme to converge. When $0 < \omega < 1$ the method is called an *under-relaxation method* and are used to obtained convergence of systems that do not converge using the Gauss-Seidel method. When $1 < \omega < 2$ the method is called an *over-relaxation method* and are used to accelerate the rate of convergence for systems that do converge using the Gauss-Seidel method. These methods are particularly useful for solving linear systems that occur in the numerical solutions of certain partial differential equations (Burden and Faires, 2011). Note that when $\omega = 1$ the SOR method is equivalent to the Gauss-Seidel method.

Algorithm 6 SOR method

Require: An $n \times n$ coefficient matrix A and right-hand side vector $\mathbf{b}$ for a linear system $A\mathbf{x} = \mathbf{b}$
a relaxation parameter $\omega \in [0, 2]$ and a convergence tolerance *tol*.
$\mathbf{x} = (0, \ldots, 0)^T$ and $\mathbf{x}^{(k-1)} = (0, \ldots, 0)^T$
$r \leftarrow |\mathbf{b} - A\mathbf{x}|$
$k \leftarrow 0$
while $r > tol$ **do**
  for $i = 1, \ldots, n$ **do**

$$x_i \leftarrow (1-\omega)x_i^{(k-1)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j\right)$$

  end for
  $r \leftarrow |\mathbf{b} - A\mathbf{x}|$
  $k \leftarrow k + 1$
end while
return x

Example 9.4.1. Use the Gauss-Seidel and SOR methods to solve the following linear system with starting values $x_i^{(0)} = 0$ iterating until $|\mathbf{r}| < 10^{-4}$. Use a value of $\omega = 1.25$ as the relaxation parameter for the SOR method

$$\begin{array}{rcrcrcl} 4x_1 & + & 3x_2 & & & = & 17, \\ 3x_1 & + & 4x_2 & - & x_3 & = & 14, \\ & - & x_2 & + & 4x_3 & = & 13. \end{array}$$

The Gauss-Seidel iterations are:

$$\begin{aligned} x_1^{(k)} &= \frac{1}{4}\left(17 - 3x_2^{(k-1)}\right), \\ x_2^{(k)} &= \frac{1}{4}\left(14 - 3x_1^{(k)} + x_3^{(k-1)}\right), \\ x_3^{(k)} &= \frac{1}{4}\left(13 + x_2^{(k)}\right), \end{aligned}$$

therefore using a relaxation parameter $\omega = 1.25$ the SOR iterations are:

$$\begin{aligned} x_1^{(k)} &= -0.25x_1^{(k-1)} + \frac{1.25}{4}\left(17 - 3x_2^{(k-1)}\right), \\ x_2^{(k)} &= -0.25x_2^{(k-1)} + \frac{1.25}{4}\left(14 - 3x_1^{(k)} + x_3^{(k-1)}\right), \\ x_3^{(k)} &= -0.25x_3^{(k-1)} + \frac{1.25}{4}\left(13 + x_2^{(k)}\right). \end{aligned}$$

The values of $x_i^{(k)}$ for both the Gauss-Seidel and SOR given in Tables 9.4 and 9.5 respectively. The Gauss-Seidel method took 24 iterations to converge to a solution of $x_1 = 2.000$, $x_2 = 3.000$ and $x_3 = 4.000$ correct to three decimal places. The SOR method only took 10 iterations to converge to the same solution. □

Table 9.4: Gauss-Seidel iterations for Example 9.4.1.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\|\mathbf{r}\|$
0	0.00000	0.00000	0.00000	25.57342
1	4.25000	0.31250	3.32812	3.45765
2	4.01562	1.32031	3.58008	3.03392
3	3.25977	1.95020	3.73755	1.89620
4	2.78735	2.34387	3.83597	1.18512
5	2.49210	2.58992	3.89748	0.74070
6	2.30756	2.74370	3.93593	0.46294
7	2.19222	2.83981	3.95995	0.28934
8	2.12014	2.89988	3.97497	0.18084
9	2.07509	2.93743	3.98436	0.11302
10	2.04693	2.96089	3.99022	0.07064
11	2.02933	2.97556	3.99389	0.04415
12	2.01833	2.98472	3.99618	0.02759
13	2.01146	2.99045	3.99761	0.01725
14	2.00716	2.99403	3.99851	0.01078
15	2.00448	2.99627	3.99907	0.00674
16	2.00280	2.99767	3.99942	0.00421
17	2.00175	2.99854	3.99964	0.00263
18	2.00109	2.99909	3.99977	0.00164
19	2.00068	2.99943	3.99986	0.00103
20	2.00043	2.99964	3.99991	0.00064
21	2.00027	2.99978	3.99994	0.00040
22	2.00017	2.99986	3.99997	0.00025
23	2.00010	2.99991	3.99998	0.00016
24	2.00007	2.99995	3.99999	0.00010

Table 9.5: SOR iterations for Example 9.4.1.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\|\mathbf{r}\|$
0	0.00000	0.00000	0.00000	1.00000
1	5.31250	-0.60547	3.87329	5.87479
2	4.55200	1.46927	3.55332	5.96006
3	2.79706	2.49585	3.95412	1.75724
4	2.27337	2.85541	3.96629	0.71501
5	2.06721	2.96260	3.99674	0.16789
6	2.01826	2.99122	3.99807	0.05142
7	2.00367	2.99815	3.99990	0.00997
8	2.00082	2.99967	3.99992	0.00256
9	2.00011	2.99996	4.00001	0.00034
10	2.00001	3.00000	4.00000	0.00007

Example 9.4.2 (Worked example). Use the SOR method to solve the following linear system with starting values $x_i^{(0)} = 0$ and a relaxation parameter of $\omega = 1.2$.

$$\begin{aligned} 3x_1 &- x_2 &&= 1, \\ -x_1 &+ 3x_2 - 2x_3 &&= 5, \\ &- 2x_2 + 3x_3 &&= -4. \end{aligned}$$

The SOR method for this system using $\omega = 1.2$ is

Working to five decimal places, fill in the table below for the first four iterations of the SOR method.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\lvert\mathbf{r}\rvert$
0				
1				
2				
3				
4				

Therefore the solution correct to decimal places is

$$x_1 =$$

$$x_2 =$$

$$x_3 =$$

□

9.4.1 Matrix form of the SOR method

To derive the matrix form of the SOR we write the coefficient matrix A as a sum of the matrices D, L and U that we saw in Sections 9.1.1 and 9.2.1

$$A = D - L - U.$$

Introducing the relaxation parameter ω by splitting the matrix D in the same way we did in Eq. (9.16)

$$A = \left(1 - \frac{1}{\omega}\right) D + \frac{1}{\omega} D - L - U.$$

Substituting into the linear system $A\mathbf{x} = \mathbf{b}$ gives

$$\left[\left(1 - \frac{1}{\omega}\right) D + \frac{1}{\omega} D - L - U\right] \mathbf{x} = \mathbf{b}$$
$$\left(\frac{1}{\omega} D - L\right) \mathbf{x} + \left[\left(1 - \frac{1}{\omega}\right) D - U\right] \mathbf{x} = \mathbf{b}.$$

Multiply both sides by ω

$$\begin{aligned}(D - \omega L)\mathbf{x} + [(\omega - 1)D - \omega U]\mathbf{x} &= \omega \mathbf{b} \\ (D - \omega L)\mathbf{x} &= [(1 - \omega)D + \omega U]\mathbf{x} + \omega \mathbf{b} \\ \mathbf{x} &= (D - \omega L)^{-1}[(1 - \omega)D + \omega U]\mathbf{x} + (D - \omega L)^{-1}\omega \mathbf{b}.\end{aligned}$$

If we let the $\mathbf{x}$ values on the left-hand side be the values at iteration k and the $\mathbf{x}$ values on the right-hand side be the values at the previous iteration then we can write

$$\mathbf{x}^{(k)} = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]\mathbf{x}^{(k-1)} + (D - \omega L)^{-1}\omega \mathbf{b}, \tag{9.18}$$

which is the matrix form of the SOR method hence the iteration matrix for the SOR method is

$$T_{SOR} = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]. \tag{9.19}$$

9.4.2 Optimal value of the relaxation parameter ω

Recall that the spectral radius is defined as the largest eigenvalue of a matrix and that the rate of convergence of an iterative scheme is measured by the spectral radius of the iteration matrix. The iteration matrix given in Eq. (9.19) is a function of ω so the spectral radius of the iteration matrix for the SOR matrix will also depend upon the value of ω. Using MATLAB, we can perform an analysis to find an estimate for the optimal value of ω that minimises $\rho(T_{SOR})$.

Example 9.4.3. Find an estimate for the optimal value of the relaxation parameter ω when the SOR method is used to solve the following linear system

$$\begin{aligned} 4x_1 &+ 3x_2 &&= 17, \\ 3x_1 &+ 4x_2 &- x_3 &= 14, \\ &- x_2 &+ 4x_3 &= 13. \end{aligned}$$

Using MATLAB to calculate $\rho(T_{SOR})$ for this system over the range $\omega = [0, 2]$ produces the plot in Fig. 9.2. By inspection, an estimate for the optimal values of the relaxation parameter is between $\omega = 1.2$ and $\omega = 1.3$.

The MATLAB commands are to produce this plot are given below.

```
% define D, L and U matrices
A = [4 3 0 ; 3 4 -1 ; 0 -1 4];
D = diag(diag(A));
L = -tril(A,-1);
U = -triu(A,1);

% generate vector of omega values
omega = 0 : 0.01 : 2;

% loop through omega and calculate the spectral radius of the iteration matrix
for i = 1 : length(omega)
    T_SOR = inv(D - omega(i)*L)*((1 - omega(i))*D + omega(i)*U);
    rho_SOR(i) = max(abs(eig(T_SOR)));
end
```

```
% plot omega against rho
plot(omega,rho_SOR)
grid on
xlabel('omega','fontsize',16)
ylabel('spectral radius','fontsize',16)
```

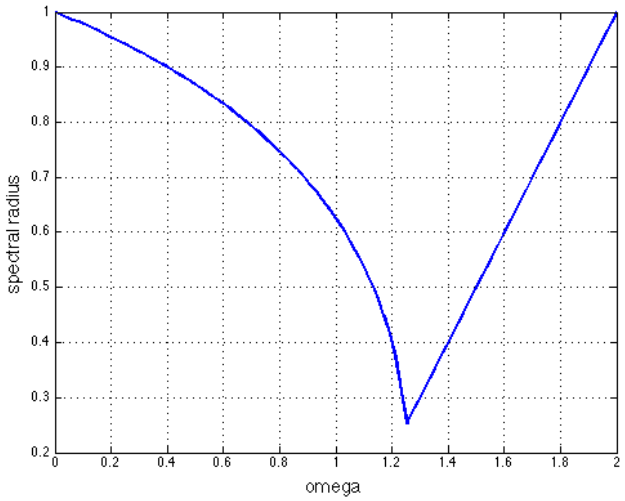


Figure 9.2: Plot of $\rho(T_{SOR})$ for different values of ω for the system in example 9.4.3.

□

Theorem 9.4.1. *Given a linear system $A\mathbf{x} = \mathbf{b}$, if A is positive definite and tridiagonal then the optimal choice of ω for the SOR method is*

$$\omega = \frac{2}{1 + \sqrt{1 - \rho(T_J)^2}},$$

where T_J is the iteration matrix for the Jacobi method.

Example 9.4.4. Find the optimal value of the relaxation parameter ω when the SOR method is used to solve the following linear system

$$\begin{aligned} 4x_1 &+ 3x_2 &&= 17,\\ 3x_1 &+ 4x_2 - x_3 &&= 14,\\ &- x_2 + 4x_3 &&= 13. \end{aligned}$$

Recall that the iteration matrix for the Jacobi method is

$$T_J = D^{-1}(L + U),$$

then

$$T_J = \begin{pmatrix} 1/4 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/4 \end{pmatrix} \begin{pmatrix} 0 & -3 & 0 \\ -3 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -0.75 & 0 \\ -0.75 & 0 & 0.25 \\ 0 & 0.25 & 0 \end{pmatrix}.$$

Calculate the eigenvalues of T_J

$$\begin{aligned} 0 = \det(T_J - \lambda I) &= \begin{vmatrix} -\lambda & -0.75 & 0 \\ -0.75 & -\lambda & 0.25 \\ 0 & 0.25 & -\lambda \end{vmatrix} \\ &= -\lambda(\lambda^2 - 0.0625) + 0.5625\lambda \\ &= -\lambda(\lambda^2 - 0.625), \end{aligned}$$

therefore $\lambda_1 = 0$, $\lambda_2 = -\sqrt{0.625}$ and $\lambda_3 = \sqrt{0.625}$ so $\rho(T_J) = \sqrt{0.625}$. The optimal value of ω is

$$\omega = \frac{2}{1 + \sqrt{1 - 0.625}} \approx 1.24,$$

which confirms our estimation from Example 9.4.3. □

9.5 Tutorial exercises: Indirect methods

1. Perform the first five iterations of the Gauss-Seidel methods for the linear system below calculating the residual for each iteration. Use starting values of $x_i^{(0)} = 0$ and work to four decimal places.

$$\begin{aligned} 4x_1 + x_2 - x_3 + x_4 &= 14, \\ x_1 + 4x_2 - x_3 - x_4 &= 10, \\ -x_1 - x_2 + 5x_3 + x_4 &= -15, \\ x_1 - x_2 + x_3 + 3x_4 &= 3. \end{aligned}$$

2. Copy from the shared area or download from Moodle the MATLAB function `jacobi` given in Section 9.1.3 on page 122. How many iterations are required to solve the linear system below correct to three decimal places? Use starting values of $x_i^{(0)} = 0$ and work to four decimal places.

$$\begin{aligned} 3x_1 + x_2 + x_3 &= 1, \\ x_1 + 6x_2 + 2x_3 &= 0, \\ x_1 - x_2 + 7x_3 &= 4. \end{aligned}$$

   State your solution and the accuracy obtained.

3. Use Gershgorin's theorem to estimate the upper bounds of the spectral radius of the linear system in the question above for the Gauss-Seidel method.

4. Use the SOR method to solve the linear system below using a relaxation parameter of $\omega = 1.2$. Perform five iterations working to four decimal places with starting values of $x_i^{(0)} = 0$.

$$\begin{aligned} 10x_1 - x_2 &= 9, \\ 3x_1 + 10x_2 - 2x_3 &= 7, \\ -2x_2 + 10x_3 &= 4. \end{aligned}$$

5. Determine the optimal value of ω for the following linear system:

$$\begin{aligned} 10x_1 + 5x_2 &= 6, \\ 5x_1 + 10x_2 - 4x_3 &= 25, \\ -4x_2 + 8x_3 - x_4 &= -11, \\ -x_3 + 5x_4 &= -11. \end{aligned}$$

The solutions can be found on page 165.

Chapter 10

QR Factorisation

The methods for solving linear systems in the previous chapters have involved solving $A\mathbf{x} = \mathbf{b}$ where A is a square matrix. Certain situations may arise where a linear system has more equations than unknowns, i.e., the system is overdetermined. In these cases there is generally no solution to $\mathbf{x}$ that satisfies $A\mathbf{x} = \mathbf{b}$ exactly, instead what we can do is find $\mathbf{x}$ that most closely satisfies the linear system using a method called linear least squares.

10.1 Linear least squares

Definition 10.1.1 (Linear least squares problem). Given an $m \times n$ matrix A and an $m \times 1$ vector $\mathbf{b}$ find an $n \times 1$ vector $\mathbf{x}$ that minimises the square of norm of the residual, $|\mathbf{r}|^2$.

The solution to the Linear Least Squares (LLS) problem will involve solving an overdetermined system which cannot be calculated using LU or Cholesky factorisation in its current form. Either we use a method that can solve overdetermined systems or we can use the normal equations to convert the LLS problem into a square system.

Theorem 10.1.1 (Normal equations). *Let $A\mathbf{x} = \mathbf{b}$ be an overdetermined linear system where A is an $m \times n$ matrix, $\mathbf{x}$ is an $n \times 1$ vector and $\mathbf{b}$ is an $m \times 1$ right-hand side vector where $m > n$. The values of $\mathbf{x}$ for which minimises $|\boldsymbol{r}|^2$ satisfies*

$$A^T A\mathbf{x} = A^T\mathbf{b}, \tag{10.1}$$

where $A^T A$ and $A^T\mathbf{b}$ are $n \times n$ matrices. Equation (10.1) *is known as the* normal equations.

Proof. Let $\mathbf{r}$ be the residual of $A\mathbf{x} = \mathbf{b}$ and $|\mathbf{r}|$ be the Euclidean norm. The square of the Euclidean norm can be written using matrix multiplication as

$$|\mathbf{r}|^2 = \mathbf{r} \cdot \mathbf{r} = \mathbf{r}^T\mathbf{r},$$

therefore using Eq. (9.4)

$$|\mathbf{r}|^2 = (A\mathbf{x} - \mathbf{b})^T(A\mathbf{x} - \mathbf{b}) = (\mathbf{x}^T A^T - \mathbf{b}^T)(A\mathbf{x} - \mathbf{b}) = \mathbf{x}^T A^T A\mathbf{x} - 2\mathbf{x}^T A^T\mathbf{b} + \mathbf{b}^T\mathbf{b}.$$

Take the derivative of $|\mathbf{r}|^2$ with respect to $\mathbf{x}$

$$\frac{d}{d\mathbf{x}}|\mathbf{r}|^2 = 2A^T A\mathbf{x} - 2A^T\mathbf{b}.$$

$|\mathbf{r}|^2$ is a minimum when $\dfrac{d}{d\mathbf{x}}|\mathbf{r}|^2 = 0$ so

$$\begin{aligned} 0 &= 2A^T\mathbf{b} - 2A^T A\mathbf{x} \\ A^T A\mathbf{x} &= A^T\mathbf{b}, \end{aligned}$$

which is Eq. (10.1). □

The solution of the normal equations can be calculated using Cholesky factorisation since A^TA is a positive definite matrix. However, the normal equations are prone to inaccuracies and are generally not recommended when solving LLS problems, hence the need for QR factorisation.

10.2 QR factorisation

QR factorisation (also known as *QR decomposition*) is a procedure for factorising the $m \times n$ coefficient matrix A into the product of two matrices, Q and R such that

$$A = QR,$$

where Q is an orthogonal matrix and R is an upper triangular matrix. Orthogonal matrices can be used to transform vectors without changing the Euclidean norm of the vector, hence they can be used to solve the LLS problem.

Definition 10.2.1. A set of vectors $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ is *orthogonal* if $\mathbf{v}_i \cdot \mathbf{v}_j = 0$ for $i \neq j$ and *orthonormal* if

$$\mathbf{v}_i \cdot \mathbf{v}_j = \begin{cases} 1 & i = j, \\ 0 & i \neq j. \end{cases}$$

Example 10.2.1. The vectors $\mathbf{v}_1 = \left(\frac{3}{5}, 0, \frac{4}{5}\right)$ and $\mathbf{v}_2 = \left(\frac{4}{5}, 0, -\frac{3}{5}\right)$ are orthogonal since

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \left(\frac{3}{5}, 0, \frac{4}{5}\right) \cdot \left(\frac{4}{5}, 0, -\frac{3}{5}\right) = \frac{12}{25} - \frac{12}{25} = 0,$$

furthermore $\mathbf{v}_1$ and $\mathbf{v}_2$ are orthonormal since

$$\mathbf{v}_1 \cdot \mathbf{v}_1 = \left(\frac{3}{5}, 0, \frac{4}{5}\right) \cdot \left(\frac{3}{5}, 0, \frac{4}{5}\right) = \frac{9}{25} + \frac{16}{25} = 1,$$
$$\mathbf{v}_2 \cdot \mathbf{v}_2 = \left(\frac{4}{5}, 0, -\frac{3}{5}\right) \cdot \left(\frac{4}{5}, 0, -\frac{3}{5}\right) = \frac{16}{25} + \frac{9}{25} = 1.$$

□

Definition 10.2.2. An *orthogonal matrix* is a matrix where the columns of the matrix are orthonormal vectors. If A is an orthogonal matrix if

$$A^TA = I,$$

or alternatively $A^{-1} = A^T$.

Example 10.2.2. Show that the following matrix is an orthogonal matrix

$$A = \begin{pmatrix} 0.8 & -0.6 \\ 0.6 & 0.8 \end{pmatrix}.$$

Evaluating A^TA

$$A^TA = \begin{pmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{pmatrix} \begin{pmatrix} 0.8 & -0.6 \\ 0.6 & 0.8 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

therefore A is an orthogonal matrix. □

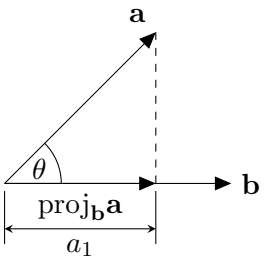


Figure 10.1: The projection of **a** onto **b**.

Theorem 10.2.1 (Vector projection). *The orthogonal projection of a vector* **a** *onto another vector* **b** *is denoted* $\text{proj}_{\mathbf{b}}\mathbf{a}$ *and produces a vector pointing in the same direction as* **b** *defined by*

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{b}|^2}\mathbf{b}. \tag{10.2}$$

Proof. Let **a** and **b** be two vectors and $\text{proj}_{\mathbf{b}}\mathbf{a}$ is the projection of **a** onto **b** (Fig. 10.1). The vector $\text{proj}_{\mathbf{b}}\mathbf{a}$ is a multiple of the unit vector $\hat{\mathbf{b}}$, i.e.,

$$\text{proj}_{\mathbf{b}}\mathbf{a} = a_1\hat{\mathbf{b}}. \tag{10.3}$$

The definition of the dot product is

$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos(\theta),$$

therefore

$$\cos(\theta) = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|}.$$

The vectors **a** and $\text{proj}_{\mathbf{b}}\mathbf{a}$ form two sides of a right-angled triangle, therefore

$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{a_1}{|\mathbf{a}|}.$$

Equating the two expressions for $\cos(\theta)$ and rearranging gives

$$a_1 = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{b}|},$$

therefore

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{b}|}\frac{\mathbf{b}}{|\mathbf{b}|} = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{b}|^2}\mathbf{b}.$$

which is Eq. (10.2). Note that if $\hat{\mathbf{b}}$ is a unit vector then

$$\text{proj}_{\mathbf{b}}\mathbf{a} = (\mathbf{a}\cdot\hat{\mathbf{b}})\hat{\mathbf{b}}.$$

□

Definition 10.2.3. The *span* of a set of vectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$ is the set of all linear combinations of these vectors, i.e., the set of all vectors of the form

$$\alpha_1\mathbf{u}_1 + \alpha_2\mathbf{u}_2 + \ldots + \alpha_n\mathbf{u}_n,$$

where $\alpha_i \in \mathbb{R}$.

10.3 QR factorisation using the Gram-Schmidt process

The calculation of the orthogonal matrix Q can be achieved by using the *Gram-Schmidt process*. Given a matrix consisting of n of linearly independent column vectors $A = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$ we wish to find a matrix that consists of n orthogonal vectors $U = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n)$ where the span of U is that same as the span of A.

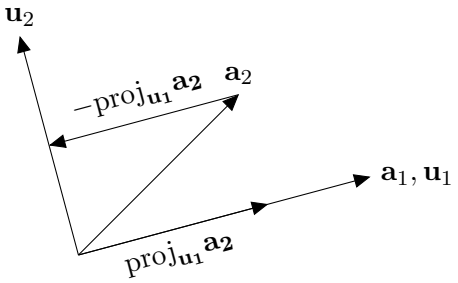


Figure 10.2: The Gram-Schmidt process finds the vector orthogonal to $\mathbf{a}_1$.

Consider Fig. 10.2 where we have two vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ that are non-orthogonal. If we let $\mathbf{u}_1 = \mathbf{a}_1$ then the vector $\mathbf{u}_2$ that is orthogonal to $\mathbf{u}_1$ can be found by subtracting the projection of $\mathbf{a}_2$ onto $\mathbf{u}_1$ from $\mathbf{a}_2$, i.e.,

$$\mathbf{u}_2 = \mathbf{a}_2 - \text{proj}_{\mathbf{u_1}}\mathbf{a_2}.$$

For the next vector $\mathbf{u}_3$ we need to find a vector that is orthogonal to both $\mathbf{u}_1$ and $\mathbf{u}_2$. To do this we simply subtract the part of $\mathbf{a}_3$ that is not orthogonal to $\mathbf{u}_1$ and $\mathbf{u}_2$ from $\mathbf{a}_3$, i.e.,

$$\mathbf{u}_3 = \mathbf{a}_3 - \text{proj}_{\mathbf{u_1}}\mathbf{a_3} - \text{proj}_{\mathbf{u_2}}\mathbf{a_3}.$$

If we carry on in this way for all vectors in A then the matrix U is calculated using

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{a}_1, \\
\mathbf{u}_2 &= \mathbf{a}_2 - \frac{\mathbf{a}_2 \cdot \mathbf{u}_1}{|\mathbf{u}_1|^2}\mathbf{u}_1, \\
\mathbf{u}_3 &= \mathbf{a}_3 - \frac{\mathbf{a}_3 \cdot \mathbf{u}_1}{|\mathbf{u}_1|^2}\mathbf{u}_1 - \frac{\mathbf{a}_3 \cdot \mathbf{u}_2}{|\mathbf{u}_2|^2}\mathbf{u}_2, \\
&\vdots \\
\mathbf{u}_j &= \mathbf{a}_j - \sum_{i=1}^{j-1} \frac{\mathbf{a}_j \cdot \mathbf{u}_i}{|\mathbf{u}_i|^2}\mathbf{u}_i.
\end{aligned}$$

Recall that the matrix Q is an orthogonal matrix which means each column vector is a unit vector, therefore we need to normalise each $\mathbf{u}_i$ vector to give orthonormal vectors that form the columns of Q, i.e., $Q = (\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n)$ where

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{a}_1, & \mathbf{q}_1 &= \frac{\mathbf{u}_1}{|\mathbf{u}_1|}, \\
\mathbf{u}_2 &= \mathbf{a}_2 - \frac{\mathbf{a}_2 \cdot \mathbf{u}_1}{|\mathbf{u}_1|^2}\mathbf{u}_1, & \mathbf{q}_2 &= \frac{\mathbf{u}_2}{|\mathbf{u}_2|}, \\
\mathbf{u}_3 &= \mathbf{a}_3 - \frac{\mathbf{a}_3 \cdot \mathbf{u}_1}{|\mathbf{u}_1|^2}\mathbf{u}_1 - \frac{\mathbf{a}_3 \cdot \mathbf{u}_2}{|\mathbf{u}_2|^2}\mathbf{u}_2, & \mathbf{q}_3 &= \frac{\mathbf{u}_3}{|\mathbf{u}_3|}, \\
&\vdots & &\vdots \\
\mathbf{u}_j &= \mathbf{a}_j - \sum_{i=1}^{j-1} \frac{\mathbf{a}_j \cdot \mathbf{u}_i}{|\mathbf{u}_i|^2}\mathbf{u}_i, & \mathbf{q}_j &= \frac{\mathbf{u}_j}{|\mathbf{u}_j|}.
\end{aligned}$$

Since $\mathbf{q} = \hat{\mathbf{u}}$ then

$$\frac{\mathbf{a}_j \cdot \mathbf{u}_i}{|\mathbf{u}|^2}\mathbf{u}_i = (\mathbf{q}_i \cdot \mathbf{a}_j)\mathbf{q}_i,$$

so

$$\begin{aligned}
\mathbf{a}_1 &= (\mathbf{q}_1 \cdot \mathbf{a}_1)\mathbf{q}_1, \\
\mathbf{a}_2 &= (\mathbf{q}_1 \cdot \mathbf{a}_2)\mathbf{q}_1 + (\mathbf{q}_2 \cdot \mathbf{a}_2)\mathbf{q}_2, \\
\mathbf{a}_3 &= (\mathbf{q}_1 \cdot \mathbf{a}_3)\mathbf{q}_1 + (\mathbf{q}_2 \cdot \mathbf{a}_3)\mathbf{q}_2 + (\mathbf{q}_3 \cdot \mathbf{a}_3)\mathbf{q}_3, \\
&\vdots \\
\mathbf{a}_j &= \sum_{i=1}^{j} (\mathbf{q}_i \cdot \mathbf{a}_j)\mathbf{q}_i
\end{aligned}$$

where $\mathbf{q}_j \cdot \mathbf{a}_j = |\mathbf{u}_j|$. If $A = QR$ then

$$r_{ij} = \mathbf{q}_i \cdot \mathbf{a}_j. \tag{10.4}$$

So the QR factorisation of A is

$$A = \begin{pmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \dots \end{pmatrix} \begin{pmatrix} \mathbf{q}_1 \cdot \mathbf{a}_1 & \mathbf{q}_1 \cdot \mathbf{a}_2 & \mathbf{q}_1 \cdot \mathbf{a}_3 & \dots \\ 0 & \mathbf{q}_2 \cdot \mathbf{a}_2 & \mathbf{q}_2 \cdot \mathbf{a}_3 & \dots \\ 0 & 0 & \mathbf{q}_3 \cdot \mathbf{a}_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

The calculation of the QR factorisation of a matrix A using the Gram-Schmidt process is outlined in Algorithm 7.

Algorithm 7 QR factorisation using the Gram-Schmidt process

Require: An $m \times n$ coefficient matrix A
  $r_{11} \leftarrow |\mathbf{a}_1|$
  $\mathbf{q}_1 \leftarrow \dfrac{\mathbf{a}_1}{r_{11}}$
  for $j = 2, \dots, n$ **do**
    for $i = 1, \dots, j-1$ **do**
      $r_{ij} \leftarrow \mathbf{q}_i \cdot \mathbf{a}_j$
    end for
    $\mathbf{u}_j \leftarrow \mathbf{a}_j - \sum_{i=1}^{j-1} r_{ij}\mathbf{q}_i$
    $r_{jj} \leftarrow |\mathbf{u}_j|$
    $\mathbf{q}_j \leftarrow \dfrac{\mathbf{u}_j}{r_{jj}}$
  end for
  return $Q = (\mathbf{q}_1, \dots, \mathbf{q}_n)$ and R

Example 10.3.1. Determine the QR factorisation of the following matrix using the Gram-Schmidt process:

$$A = \begin{pmatrix} 12 & -51 & 4 \\ 6 & 167 & -68 \\ -4 & 24 & -41 \end{pmatrix}.$$

Let $\mathbf{u}_1 = \mathbf{a}_1$ then for the first column ($j = 1$):

$$r_{11} = |\mathbf{u}_1| = \sqrt{12^2 + 6^2 + (-4)^2} = \sqrt{196} = 14,$$
$$\mathbf{q}_1 = \frac{\mathbf{u}_1}{r_{11}} = \begin{pmatrix} 12/14 \\ 6/14 \\ -4/14 \end{pmatrix} = \begin{pmatrix} 6/7 \\ 3/7 \\ -2/7 \end{pmatrix}.$$

Second column ($j = 2$):

$$r_{12} = (\mathbf{q}_1 \cdot \mathbf{a}_2) = \begin{pmatrix} 6/7 \\ 3/7 \\ -2/7 \end{pmatrix} \cdot \begin{pmatrix} -51 \\ 167 \\ 24 \end{pmatrix} = 21,$$
$$\mathbf{u}_2 = \mathbf{a}_2 - r_{12}\mathbf{q}_1 = \begin{pmatrix} -51 \\ 167 \\ 24 \end{pmatrix} - 3\begin{pmatrix} 6 \\ 3 \\ -2 \end{pmatrix} = \begin{pmatrix} -69 \\ 158 \\ 30 \end{pmatrix},$$
$$r_{22} = |\mathbf{u}_2| = \sqrt{(-69)^2 + 158^2 + 30^2} = \sqrt{30625} = 175,$$
$$\mathbf{q}_2 = \frac{\mathbf{u}_2}{r_{22}} = \begin{pmatrix} -69/175 \\ 158/175 \\ 6/35 \end{pmatrix}.$$

Third column ($j = 3$)

$$r_{13} = (\mathbf{q}_1 \cdot \mathbf{a}_3) = \begin{pmatrix} 6/7 \\ 3/7 \\ -2/7 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ -68 \\ -41 \end{pmatrix} = -14,$$
$$r_{23} = (\mathbf{q}_2 \cdot \mathbf{a}_3) = \begin{pmatrix} -69/175 \\ 158/175 \\ 6/35 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ -68 \\ -41 \end{pmatrix} = -70,$$
$$\mathbf{u}_3 = \mathbf{a}_3 - r_{13}\mathbf{q}_1 - r_{23}\mathbf{q}_2 = \begin{pmatrix} 4 \\ -68 \\ -41 \end{pmatrix} + 14\begin{pmatrix} 6/7 \\ 3/7 \\ -2/7 \end{pmatrix} + 70\begin{pmatrix} -69/175 \\ 158/175 \\ 6/35 \end{pmatrix} = \begin{pmatrix} -58/5 \\ 6/5 \\ -33 \end{pmatrix},$$
$$r_{33} = |\mathbf{u}_3| = \sqrt{\left(-\frac{58}{5}\right)^2 + \left(\frac{6}{5}\right)^2 + (-33)^2} = \sqrt{1225} = 35,$$
$$\mathbf{q}_3 = \frac{\mathbf{u}_3}{r_{33}} = \begin{pmatrix} -58/175 \\ 6/175 \\ -33/35 \end{pmatrix}.$$

Therefore the QR factorisation of A is

$$A = \begin{pmatrix} 6/7 & -69/175 & -58/175 \\ 3/7 & 158/175 & 6/5 \\ -2/7 & 6/35 & -33/35 \end{pmatrix} \begin{pmatrix} 14 & 21 & -14 \\ 0 & 175 & -70 \\ 0 & 0 & 35 \end{pmatrix}.$$

□

Example 10.3.2 (Worked example). Determine the QR factorisation of the following matrix using the Gram-Schmidt process:

$$A = \begin{pmatrix} 9 & 0 & 26 \\ 12 & 0 & -7 \\ 0 & 4 & 4 \\ 0 & -3 & -3 \end{pmatrix}.$$

□

10.4 Solving linear systems using QR factorisation

Consider the solution to the linear system $A\mathbf{x} = \mathbf{b}$. Since $A = QR$ then

$$\begin{aligned} QR\mathbf{x} &= \mathbf{b} \\ R\mathbf{x} &= Q^{-1}\mathbf{b}. \end{aligned}$$

Taking advantage of the orthogonal nature of Q, i.e., $Q^{-1} = Q^T$, we can write this equation as

$$R\mathbf{x} = Q^T\mathbf{b},$$

and since R is upper triangular the solution of $\mathbf{x}$ can be obtained through back substitution.

Example 10.4.1. Solve the following linear system using QR factorisation

$$\begin{aligned} 12x_1 &- 51x_2 + 4x_3 &= -43, \\ 6x_1 &+ 167x_2 - 68x_3 &= 451, \\ -4x_1 &+ 24x_2 - 41x_3 &= 180. \end{aligned}$$

It was shown in Example 10.3.1 that $A = QR$ where

$$Q = \begin{pmatrix} 6/7 & -69/175 & -58/175 \\ 3/7 & 158/175 & 6/5 \\ -2/7 & 6/35 & -33/35 \end{pmatrix}, \qquad R = \begin{pmatrix} 14 & 21 & -14 \\ 0 & 175 & -70 \\ 0 & 0 & 35 \end{pmatrix}.$$

Writing $A\mathbf{x} = \mathbf{b}$ in the form $R\mathbf{x} = Q^T\mathbf{b}$

$$\begin{pmatrix} 14 & 21 & -14 \\ 0 & 175 & -70 \\ 0 & 0 & 35 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6/7 & 3/7 & -2/7 \\ -69/175 & 158/175 & 6/35 \\ -58/175 & 6/175 & -33/35 \end{pmatrix} \begin{pmatrix} -43 \\ 451 \\ 180 \end{pmatrix} = \begin{pmatrix} 105 \\ 455 \\ -140 \end{pmatrix}$$

Solve $\mathbf{x}$ by back substitution

$$\begin{aligned} x_3 &= -\frac{140}{35} = -4, \\ x_2 &= \frac{1}{175}[455 + 70(-4)] = 1, \\ x_1 &= \frac{1}{14}[105 + 14(-4) - 21] = 2. \end{aligned}$$

□

10.5 Tutorial exercises: QR factorisation

1. Which of the following matrices are orthogonal?

   (a) $A = \begin{pmatrix} 0.1507 & 0.9886 \\ 0.9886 & -0.1507 \end{pmatrix}$;

   (b) $B = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$;

   (c) $C = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$.

2. Determine the QR factorisation of the following matrices using the Gram-Schmidt process:

   (a) $A = \begin{pmatrix} 1 & 1 \\ -1 & 0 \end{pmatrix}$;

   (b) $B = \begin{pmatrix} 6 & 6 & 1 \\ 3 & 6 & 1 \\ 2 & 1 & 1 \end{pmatrix}$;

   (c) $C = \begin{pmatrix} 1 & 2 & 1 \\ 1 & 4 & 3 \\ 1 & -4 & 6 \\ 1 & 2 & 1 \end{pmatrix}$.

3. Solve the following linear system using QR factorisation:

$$\begin{aligned} 6x_1 &+ 6x_2 + x_3 = 22, \\ 3x_1 &+ 6x_2 + x_3 = 16, \\ 2x_1 &+ x_2 + x_3 = 9. \end{aligned}$$

The solutions can be found on page 166.

Bibliography

Burden, R.L. and J.D. Faires (2011). *Iterative techniques in matrix algebra: Relaxation techniques for solving linear systems*. URL: {http://www.math.ust.hk/~mamu/courses/231/Slides/CH07_4A.pdf}.

Butcher, J.C. (1987). *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience publication. J. Wiley. ISBN: 9780471910466.

— (2008). *Numerical Methods for Ordinary Differential Equations*. J. Wiley. ISBN: 978-0-470-72335-7.

Heckbert, P (2000). *Boundary Value Problems [online]*. Carnegie Mellon University. Available at: http://www.cs.cmu.edu/~ph/859B/www/notes/ode/bvp.html [Accessed 6 January 2014].

Householder, A. S. (1958). "Unitary triangularization of a nonsymmetric matrix". In: *Journal of the ACM* 5.4, pp. 339 –342.

Johnson, P (2008). *Boundary Value Problems [online]*. Manchester University. Available at: http://www.maths.manchester.ac.uk/~pjohnson/CFD/Lectures/boundary_value_probs.pdf [Accessed 6 January 2014].

Kahan, W. (1958). "Gauss-Seidel methods of solving large systems of linear equations". PhD thesis. Toronto, Canada: University of Toronto.

Parnell, C. (2013). *Numerical Analysis Chapter 2: Iterative Methods*. St Andrews University. URL: {http://www-solar.mcs.st-andrews.ac.uk/~clare/Lectures/num-analysis/Numan_chap2.pdf}.

Press, W.H., S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery (1992). *Numerical Recipes in Fortran 77: The Art of Scientific Computing*. Cambridge University Press. URL: {http://apps.nrbook.com/fortran/index.html}.

Stensby, J. (2013). *Analytical and Computational Methods*. The University of Alabama in Huntsville. URL: {http://www.ece.uah.edu/courses/ee448/}.

Appendix A

Geometric series of matrices

The analysis of the stability of explicit Runge-Kutta methods (Chapter 4) requires the geometric series of matrices.

Definition A.0.1. Let T be a square matrix, the series $\{S_n\}_{n\geq 0}$ is defined by

$$S_n = I + T + T^2 + \ldots + T^{n-1};$$

This series is called the *geometric series of matrices*. The series converges if $\{S_n\}_{n\geq 0}$ converges, then we can write

$$\lim_{n\to\infty} S_n = \sum_{n=0}^{\infty} T^n.$$

Lemma A.0.1. *For any square matrix T*

$$(I - T)\sum_{k=0}^{n} T^k = I - T^{n+1}.$$

Proof. First consider the case when $n = 1$:

$$(I - T)(I + T) = (I + T) - (T + T^2) = I - T^2.$$

When $n = 2$

$$(I - T)(I + T + T^2) = (I + T + T^2) - (T + T^2 + T^3) = I - T^3.$$

Therefore

$$\begin{aligned}(I - T)(I + T + T^2 + \ldots + T^n) = (I + T + T^2 + \ldots + T^{n-1} + T^n) \\ - (T + T^2 + \ldots + T^{n-1} + T^n + T^{n+1})\end{aligned}$$

$$\therefore (I - T)\sum_{k=0}^{n} T^k = I - T^{n+1}.$$

□

Theorem A.0.1. *Let T be a square matrix with $\|T\| < 1$. Then $I - T$ is non-singular and*

$$(I - T)^{-1} = \sum_{k=0}^{n} T^k. \tag{A.1}$$

Proof. First prove that $I - T$ is non-singular using proof by contradiction. Suppose $I - T$ is singular. Then there exists a non-zero vector $\mathbf{x}$ such that $(I - T)\mathbf{x} = 0$. Therefore $|\mathbf{x}| = |T\mathbf{x}|$*. Since $|T\mathbf{x}| \leq \|T\||\mathbf{x}|$ then $|\mathbf{x}| = |T\mathbf{x}| \leq \|T\||\mathbf{x}|$ which means $\|T\| \geq 1$. This is a contradiction so $I - T$ must be singular.

To prove Eq. (A.1) we can use lemma A.0.1

$$(I - T)\sum_{k=0}^{n} T^k = I - T^{n+1}.$$

Using the property of matrix norms that states $\|T^k\| \leq \|T\|^k$ then since $\|T\| < 1$ then $T^k \to 0$ as $k \to \infty$. Therefore

$$\lim_{n\to\infty}(I - T)\sum_{k=0}^{n} T^k = I$$
$$\sum_{k=0}^{n} T^k = (I - T)^{-1}$$

□

*Readers unfamiliar with vector and matrix norms should read Section 6.3 on page 81

Appendix B

Tutorial exercises solutions

B.1 ODEs Preliminaries

Solutions to the tutorials exercises on page 13.

1. (a) Euler method:

t	y_{Euler}
0.00	1.000000
0.10	1.000000
0.20	1.010000
0.30	1.029802
0.40	1.058934
0.50	1.096708
0.60	1.142299
0.70	1.194824
0.80	1.253410
0.90	1.317236
1.00	1.385561

(b) Second-order Runge-Kutta method:

t	y_{RK2}
0.00	1.000000
0.10	1.005000
0.20	1.019828
0.30	1.044064
0.40	1.077074
0.50	1.118080
0.60	1.166240
0.70	1.220707
0.80	1.280676
0.90	1.345413
1.00	1.414263

2. Let $y_1 = y$ and $y_2 = y'$ then

$$\begin{aligned} y_1' &= y_2, & y_1(1) &= 1, \\ y_2' &= y_2 + ty_1, & y_2(1) &= 2. \end{aligned}$$

(a) Euler method:

t	y	y'
1.00	1.000000	2.000000
1.20	1.400000	2.600000
1.40	1.920000	3.456000
1.60	2.611200	4.684800
1.80	3.548160	6.457344
2.00	4.839629	9.026150
2.20	6.644859	12.767232
2.40	9.198305	18.244416
2.60	12.847189	26.308486
2.80	18.108886	38.250721
3.00	25.759030	56.041842

(b) Second-order Runge-Kutta method:

t	y	y'
1.00	1.000000	2.000000
1.20	1.460000	2.728000
1.40	2.095200	3.819184
1.60	2.994086	5.468844
1.80	4.293043	7.982668
2.00	6.203779	11.844067
2.20	9.057625	17.824640
2.40	13.377581	27.166686
2.60	19.996376	41.886939
2.80	30.251314	65.285589
3.00	46.308217	102.805390

B.2 Explicit Runge-Kutta methods

Solutions to the tutorials exercises on page 28.

1.

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{4} & \frac{1}{4} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 1 & -2 & 2 & \\ \hline & \frac{1}{6} & 0 & \frac{2}{3} & \frac{1}{6} \end{array}$$

2.

$$\begin{aligned} k_1 &= hf(t, y), \\ k_2 &= hf\left(t_n + \frac{1}{4}h, y_n + \frac{1}{4}k_1\right), \\ k_3 &= hf\left(t_n + \frac{1}{2}h, y_n - \frac{1}{2}k_1 + k_2\right), \\ k_4 &= hf\left(t_n + h, y_n + \frac{1}{4}k_1 + \frac{3}{4}k_3\right), \\ y_{n+1} &= y_n + \frac{1}{9}(4k_2 + 3k_3 + 2k_4). \end{aligned}$$

3.

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{3} & \frac{1}{3} & & \\ 1 & -1 & 2 & \\ \hline & 0 & \frac{3}{4} & \frac{1}{4} \end{array}$$

4.

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{5} & \frac{1}{5} & & & \\ \frac{3}{4} & -\frac{31}{32} & \frac{55}{32} & & \\ 1 & 9 & -\frac{120}{11} & \frac{32}{11} & \\ \hline & 0 & \frac{125}{264} & \frac{16}{33} & \frac{1}{24} \end{array}$$

5. $h = 0.1$

n	t_n	k_1	k_2	$y_{computed}$	y_{exact}	$\lvert y_{computed} - y_{exact} \rvert$
0	0.0			2.000000	2.000000	0.000000
1	0.1	0.200000	0.210000	2.205000	2.205171	0.000171
2	0.2	0.210500	0.221550	2.421025	2.421403	0.000378
3	0.3	0.222103	0.234313	2.649233	2.649859	0.000626
4	0.4	0.234923	0.248416	2.890902	2.891825	0.000923
5	0.5	0.249090	0.263999	3.147447	3.148721	0.001275

$h = 0.05$

n	t_n	k_1	k_2	$y_{computed}$	y_{exact}	$\lvert y_{computed} - y_{exact} \rvert$
0	0.00			2.000000	2.000000	0.000000
1	0.05	0.100000	0.102500	2.101250	2.101271	0.000021
2	0.10	0.102563	0.105191	2.205127	2.205171	0.000044
3	0.15	0.105256	0.108019	2.311764	2.311834	0.000070
4	0.20	0.108088	0.110993	2.421305	2.421403	0.000098
5	0.25	0.111065	0.114118	2.533897	2.534025	0.000129
6	0.30	0.114195	0.117405	2.649696	2.649859	0.000163
7	0.35	0.117485	0.120859	2.768868	2.769068	0.000199
8	0.40	0.120943	0.124491	2.891585	2.891825	0.000239
9	0.45	0.124579	0.128308	3.018029	3.018312	0.000283
10	0.50	0.128401	0.132322	3.148390	3.148721	0.000331

$h = 0.025$

n	t_n	k_1	k_2	$y_{computed}$	y_{exact}	$\lvert y_{computed} - y_{exact} \rvert$
0	0.000			2.000000	2.000000	0.000000
1	0.025	0.050000	0.050625	2.050313	2.050315	0.000003
2	0.050	0.050633	0.051274	2.101266	2.101271	0.000005
3	0.075	0.051282	0.051939	2.152876	2.152884	0.000008
4	0.100	0.051947	0.052621	2.205160	2.205171	0.000011
5	0.125	0.052629	0.053320	2.258134	2.258148	0.000014
6	0.150	0.053328	0.054037	2.311816	2.311834	0.000018
7	0.175	0.054045	0.054772	2.366225	2.366246	0.000021
8	0.200	0.054781	0.055525	2.421378	2.421403	0.000025
9	0.225	0.055534	0.056298	2.477294	2.477323	0.000029
10	0.250	0.056307	0.057090	2.533993	2.534025	0.000033
11	0.275	0.057100	0.057902	2.591494	2.591531	0.000037
12	0.300	0.057912	0.058735	2.649817	2.649859	0.000041
13	0.325	0.058745	0.059589	2.708985	2.709031	0.000046
14	0.350	0.059600	0.060465	2.769017	2.769068	0.000051
15	0.375	0.060475	0.061362	2.829936	2.829991	0.000056
16	0.400	0.061373	0.062283	2.891764	2.891825	0.000061
17	0.425	0.062294	0.063226	2.954524	2.954590	0.000066
18	0.450	0.063238	0.064194	3.018240	3.018312	0.000072
19	0.475	0.064206	0.065186	3.082936	3.083014	0.000078
20	0.500	0.065198	0.066203	3.148637	3.148721	0.000084

As h is halved the error decreases by a factor of 4 which is expected since this is a second-order method.

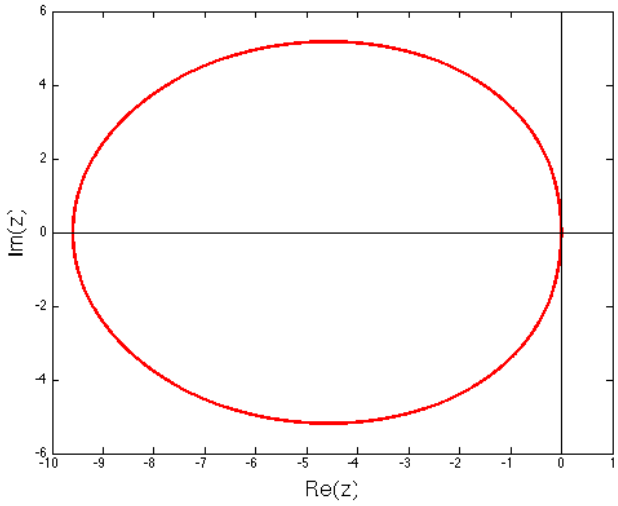
B.3 Impicit Runge-Kutta methods

Solutions to the tutorial exercises on page 39

1. 3
2. $$\begin{array}{c|cc} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$$
3. $$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$
4. $$\begin{array}{c|cc} 1-\frac{1}{2}\sqrt{2} & 1-\frac{1}{2}\sqrt{2} & 0 \\ 3-\frac{3}{2}\sqrt{2} & 2-\sqrt{2} & 1-\frac{1}{2}\sqrt{2} \\ \hline & \frac{1}{2}-\sqrt{2} & -\frac{3}{2}-\sqrt{2} \end{array}$$
5. DIRK methods are lower triangular meaning they can be solved sequentially hence are much more computationally efficient than other implicit methods.

B.4 Stability

Solutions to the tutorial exercises on page 51

1. $R(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4$.
2. $R(z) = \dfrac{1 + \frac{1}{2}z + \frac{1}{12}z^2}{1 - \frac{1}{2}z + \frac{1}{12}z^2}$, $E(y) = 0$.
3. 

4. $R(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4 + \frac{1}{252}z^5$. Third-order method.

B.5 Boundary Value Problems

Solutions to the tutorial exercises on page 75

1. (a), (b) and (c) have unique solutions.
2. (a) $s = 1$

t	y_1	y_2
0.00	1.0000	1.0000
0.50	1.5000	1.0000
1.00	2.0000	1.5000
1.50	2.7500	2.5000
2.00	4.0000	4.0000

(b) $s = -1$

t	y_1	y_2
0.00	1.0000	-1.0000
0.50	0.5000	-1.0000
1.00	0.0000	-0.5000
1.50	-0.2500	0.5000
2.00	0.0000	2.0000

3.

t	y	y_{exact}	$\lvert y_{exact} - y \rvert$
0.00	1.0000	1.0000	0.0000
0.50	1.2500	0.8750	0.3750
1.00	1.5000	1.0000	0.5000
1.50	2.0000	1.6250	0.3750
2.00	3.0000	3.0000	0.0000

4.

t	y
0.00	1.0000
0.50	0.8750
1.00	1.0000
1.50	1.6250
2.00	3.0000

B.6 LU factorisation

Solutions to the tutorials exercises on page 106.

1.
$$U = \begin{pmatrix} 2 & 3 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \qquad L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}.$$

2. $x_1 = -2$, $x_2 = 3$, $x_3 = 1$.

3.
$$U = \begin{pmatrix} 3 & 9 & 5 \\ 0 & -2 & 5/3 \\ 0 & 0 & -1/2 \end{pmatrix}, \qquad P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \qquad L = \begin{pmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & 1/2 & 1 \end{pmatrix}.$$

4. $x_1 = 1$, $x_2 = 3$, $x_3 = -2$.

B.7 Cholesky factorisation

Solutions to the tutorials exercises on page 117.

1. (a) is not positive definite and (b) is positive definite.
2. $-2 \leq \alpha \leq \frac{3}{2}$
3. (a)
$$L = \begin{pmatrix} 2 & 0 \\ -3 & 1 \end{pmatrix}.$$
(b)
$$L = \begin{pmatrix} 4 & 0 & 0 \\ 3 & 2 & 0 \\ 5 & 2 & 2 \end{pmatrix}.$$
(c)
$$L = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -2 & 5 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 4 & 2 & 3 & 1 \end{pmatrix}.$$
4.
$$L = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 2 & 0 \\ -2 & 3 & 2 \end{pmatrix}.$$
$x_1 = -1$, $x_2 = 0$, $x_3 = 2$.

5.
```
>> a = [1 2 4 7 11 ;
2 13 23 38 58 ;
4 23 77 122 182 ;
7 38 122 294 430 ;
11 58 182 430 855];
>> b = [50 265 769 1571 2548]';
>> L = cholesky(A)

L =

     1     0     0     0     0
     2     3     0     0     0
     4     5     6     0     0
     7     8     9    10     0
    11    12    13    14    15

>> y = L\b

y =

    50
    55
    49
    34
    15

>> x = L'\y

x =

     5
     4
     3
     2
     1
```

B.8 Indirect methods

Solutions to the tutorials exercises on page 141.

1. The Gauss-Seidel method for this system is

```
 k |   x(1)   |   x(2)   |   x(3)   |   x(4)   |   |r|
-----------------------------------------------------
 0 |  0.0000 |  0.0000 |  0.0000 |  0.0000 | 23.0217
 1 |  3.5000 |  1.6250 | -1.9750 |  1.0333 |  4.8397
 2 |  2.3417 |  1.6792 | -2.4025 |  1.5800 |  1.1707
 3 |  2.0846 |  1.7732 | -2.5444 |  1.7444 |  0.4334
 4 |  1.9845 |  1.8039 | -2.5912 |  1.8035 |  0.1493
 5 |  1.9504 |  1.8155 | -2.6075 |  1.8242 |  0.0531
-----------------------------------------------------
```

2.

```
  k |   x(1)   |   x(2)   |   x(3)   |   |r|
-------------------------------------------
  0 |  0.0000 |  0.0000 |  0.0000 |  4.1231
  1 |  0.3333 |  0.0000 |  0.5714 |  1.6176
  2 |  0.1429 | -0.2460 |  0.5238 |  0.4135
  3 |  0.2407 | -0.1984 |  0.5159 |  0.1041
  4 |  0.2275 | -0.2121 |  0.5087 |  0.0346
  5 |  0.2345 | -0.2075 |  0.5086 |  0.0085
  6 |  0.2330 | -0.2086 |  0.5083 |  0.0027
  7 |  0.2334 | -0.2083 |  0.5083 |  0.0007
  8 |  0.2333 | -0.2084 |  0.5083 |  0.0002
  9 |  0.2333 | -0.2083 |  0.5083 |  0.0001
-------------------------------------------
```

   The Jacobi method takes 8 iterations to solve the system correct to three decimal places. The solution is $x_1 = 0.233$, $x_2 = -0.208$ and $x_3 = 0.508$.

3.

4. $\rho(T_{GS}) \leq \max(0 + 0, 0.0556 + 0.2778, 0.0079 + 0.0556) = 0.3333$.

5. $\omega = 1.1535$.

B.9 QR factorisation

Solutions to the tutorials exercises on page 151.

1. A and B are orthogonal and C is non-orthogonal.
2. (a) $Q = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ and $R = \frac{1}{\sqrt{2}}\begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$.

   (b) $Q = \frac{1}{7}\begin{pmatrix} 6 & -2 & -3 \\ 3 & 6 & 2 \\ 2 & -3 & 6 \end{pmatrix}$ and $R = \frac{1}{7}\begin{pmatrix} 49 & 56 & 11 \\ 0 & 21 & 1 \\ 0 & 0 & 5 \end{pmatrix}$.

   (c) $Q = \frac{1}{6}\begin{pmatrix} 3 & 1 & -2\sqrt{2} \\ 3 & 3 & 3\sqrt{2} \\ 3 & -5 & \sqrt{2} \\ 3 & 1 & -2\sqrt{2} \end{pmatrix}$ and $R = \frac{1}{18}\begin{pmatrix} 9 & 9 & 99 \\ 0 & 108 & -57 \\ 0 & 0 & 33\sqrt{2} \end{pmatrix}$.
3. $x_1 = 2$, $x_2 = 1$, $x_3 = 4$.