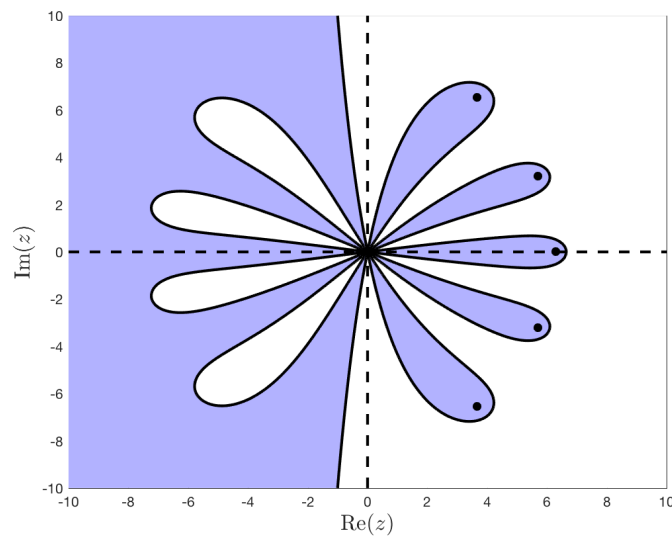


**6G7Z3001 Advanced Ordinary Differential Equations &
Dynamical Systems**

Runge-Kutta Methods

Dr Jon Shiach

2020



Contents

1	Deriving Runge-Kutta Methods	1
1.1	ODEs preliminaries	1
1.2	General form of a Runge-Kutta method	2
1.3	Trees	3
1.4	Deriving Explicit Runge-Kutta methods	7
1.5	Tutorial exercises	11
2	Adaptive Step Size Control	13
2.1	Accuracy of Runge-Kutta methods	14
2.2	Step doubling	16
2.3	Adaptive step size control	18
2.4	Solving systems of equations using explicit Runge-Kutta methods	21
2.5	Tutorial exercises	24
3	Implicit Runge-Kutta Methods	25
3.1	General form of an implicit Runge-Kutta method	25
3.2	Newton's method	27
3.3	Solving systems of equations using IRK methods	29
3.4	Tutorial exercises	37
4	Stability & Order Stars	39
4.1	Absolute stability	39
4.2	Stability of systems of ODEs	40
4.3	Relative stability	43
4.4	Padé approximants	44
4.5	Stiffness	47
4.6	Stiff solvers	49
4.7	Tutorial exercises	50
5	Applications of ODEs: the N-body problem	51
5.1	Gravitation	52
5.2	The two-body problem	53
5.3	The three-body problem	56
5.4	The N-body problem formulation	59
5.5	Modelling the solar system	60
5.6	Tutorial Exercises	62
A	Solutions to tutorial exercises	65
A.1	Deriving Runge-Kutta Methods	65

A.2	Adaptive Step Size Control	67
A.3	Stability & Order Stars	68
A.4	Implicit Runge-Kutta Methods	71
A.5	Applications of ODEs – the N body problem	76

B Ephemeris for solar system bodies 79

Teaching schedule

The teaching schedule for Jon Shiach's part of the unit is given in the table below.

Week	Date (w/c)	Content
2	21/09/2020	Chapter 1 – Deriving Runge-Kutta methods ODEs preliminaries, rooted trees, using rooted trees to derive explicit Runge-Kutta methods. Chapter 2 – Adaptive step size control Solving systems of equations using Runge-Kutta methods, estimating accuracy, step doubling and adaptive step size control Coursework assignment handed out (25/09/2020)
3	28/09/2020	Chapter 2 – Adaptive step size control (cont.) Chapter 3 – Implicit Runge-Kutta methods Stiff solvers, Newton's method, solution procedure for implicit Runge-Kutta methods
4	05/10/2020	Chapter 4 – Stability and order stars Absolute and relative stability, stability of systems of ODEs, Padé approximants and order stars
5	12/10/2020	Chapter 5 – Applications of ODEs: The N-body problem Gravitation, two-body, three-body and N-body problems, modelling the solar system
6	19/10/2020	Consolidation of materials and revision Coursework assignment deadline (23/10/2020)

Chapter 1

Deriving Runge-Kutta Methods

The Runge-Kutta methods are a class of numerical methods used for the calculation of solutions to *Ordinary Differential Equations* (ODEs). Named after German mathematicians Carl Runge (1856 – 1927) and Martin Kutta (1867 – 1944), Runge-Kutta methods are an example of *single step* methods because calculation of the solution at the next step only requires values from the current step unlike the class of *multistep methods* that require values from multiple previous steps. Since the advent of computer technology has enabled computational methods to be applied to solve differential equations, the Runge-Kutta methods have become popular with mathematicians, physicists and engineers due to their convenience and accuracy.

The derivation of explicit Runge-Kutta methods uses the Taylor series to determine the values of the coefficients in the method. This is simple for low order methods but soon becomes unwieldy for higher order methods (order 3 and above). New Zealand mathematician John Butcher (1987; 2009) developed a method that uses rooted trees to generate the order conditions for Runge-Kutta methods that significantly simplifies the derivation of higher order methods.

1.1 ODEs preliminaries

1.1.1 Ordinary Differential Equations (ODEs)

An ODE is an equation that is written in terms of a function of a single dependent variable and derivatives of this function. Let x be the independent variable and y is a function of x then an ODE can be written in the form

$$y^{(N)}(x) = f(x, y, y', y'', \dots, y^{(N-1)}), \quad (1.1)$$

where y' , y'' etc. denote derivatives of y with respect to x^* . The *order* of an ODE is the order of the largest derivative in the ODE. The solution to an ODE is the function y over the interval $x \in [a, b]$ where $x \in \mathbb{R}$.

1.1.2 Initial Value Problems (IVPs)

The solution of an ODE is an infinite number of functions that satisfy the ODE unless the solution $y(x)$ is known at a particular value of x . If it is known that the solution to an ODE when $x = x_0$ is $y(x_0) = \alpha$ where α is some value then we write this as

*In this chapter, x is used to represent the dependent variable instead of the more common t to avoid confusion when using trees that are represented by t .

$$y^{(N)}(x) = f(x, y, y', y'', \dots, y^{(N-1)}), \quad x \in [x_{\min}, x_{\max}], \quad y(x_{\min}) = \alpha. \quad (1.2)$$

For most practical applications, the solution may be known at the start of some process, e.g., the concentration of a substance is known before some chemical reaction. Problems of the form Eq. (1.2) are known as an *initial value problems*.

1.1.3 Higher order ODEs

The numerical methods used to solve ODEs only apply to first-order ODEs. An N^{th} order ODE can be written as a system of $N + 1$ first-order ODEs by defining a number of functions that equal the derivatives in the ODE. For example given the ODE in Eq. (1.1), let $y_1 = y$, $y_2 = y'$, $y_3 = y''$ etc. then

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= y_3, \\ &\vdots \\ y_N' &= f(x, y_1, y_2, \dots, y_{N-1}). \end{aligned}$$

1.2 General form of a Runge-Kutta method

A Runge-Kutta method for solving an ODE of the form shown in Eq. (1.2) is

$$y_{n+1} = y_n + h \sum_{i=1}^s b_i k_i, \quad (1.3)$$

where

$$k_i = f(x_n + c_i h, y_n + h \sum_{j=1}^s a_{ij} k_j), \quad (1.4)$$

and y_n is a known solution for $x = x_n$, y_{n+1} is the unknown solution for $x = x_{n+1}$ that is being computed, h is the *step length* which is defined as $h = x_{n+1} - x_n$, k_i where $i = 1, \dots, s$ are the intermediate *stage values* and a_{ij} , b_i and c_i where $i, j = 1, \dots, s$ are coefficients that define the specific Runge-Kutta method.

It is common to express Runge-Kutta methods in the form of a *Butcher tableau* where the coefficients are arranged such that

$$\begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b} \end{array} \quad (1.5)$$

For example, an s -stage Runge-Kutta method the Butcher tableau would take the form

$$\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\
\hline
& b_1 & b_2 & \cdots & b_s
\end{array} \tag{1.6}$$

1.2.1 Explicit Runge-Kutta methods

Explicit Runge-Kutta (ERK) methods are methods where the k_i stage values are expressed explicitly in terms of known values, i.e., the calculation of each stage depends only on known values of the previous stages. For example,

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

or alternatively as the Butcher tableau

$$\begin{array}{c|cccc}
0 & 0 & & & \\
c_2 & a_{21} & & & \\
c_3 & a_{31} & a_{32} & & \\
\vdots & \vdots & \vdots & \ddots & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \\
\hline
& b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array} \tag{1.7}$$

The advantages of ERK methods is that they are straight forward to compute although a disadvantage are that they can struggle to solve stiff systems (see Chapter 3).

1.3 Trees

A *tree* is a graph where any two vertices are connected by exactly one path. A graph G is a tree if the following conditions are satisfied:

- G is connected (it is possible to define a path to join one vertex to any other vertex in G)
- G has no cycles (a path that starts and ends at the same vertex)
- A cycle is formed if any edge is add to G
- If any edge is removed from G then it is disconnected
- If G has n vertices then it has $n - 1$ edges

- A tree can be known as a *rooted tree* where a special node is singled out. Rooted trees are usually represented with the root vertex drawn at the bottom of the tree

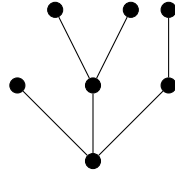
A tree can be known as a *rooted tree* where a special vertex is singled out to be the *root vertex*. Rooted trees are usually represented with the root vertex drawn at the bottom of the tree. From herein in these notes will refer to rooted trees by the term 'trees'.

For convenience we define τ as the tree \bullet (a tree with just one vertex and no edges) and to write $t = [t_1 t_2 \dots t_\nu]$ as the tree formed by adjoining the roots of the trees $t_1 t_2 \dots t_\nu$ to the root of τ . If t_1, t_2, \dots, t_m are distinct and occur k_1, k_2, \dots, k_m times amongst t_1, t_2, \dots, t_ν then we write

$$t = [t_1^{k_1} t_2^{k_2} \dots t_m^{k_m}],$$

and $\sum_{i=1}^m k_i = \nu$ (Butcher, 2009).

For example, consider the tree below



this can be represented using tree notation as

$$t = [t_1 t_2 t_3] = \left[\begin{array}{c} \bullet \\ \bullet \end{array} \begin{array}{c} \bullet \bullet \\ \bullet \end{array} \begin{array}{c} \bullet \\ \bullet \end{array} \right] = [\tau [\tau^2] [\tau]].$$

1.3.1 Properties of rooted trees

Let T denote the set of rooted trees t then the following properties apply (Lambert, 1991):

- The *order*, denoted by $r(t)$, is the number of vertices in the tree t and can be calculated using the following recursive relation

$$r(\tau) = 1, \tag{1.8a}$$

$$r([t_1^{k_1} t_2^{k_2} \dots t_m^{k_m}]) = 1 + \sum_{i=1}^m k_i r(t_i). \tag{1.8b}$$

- The *symmetry*, denoted by $\sigma(t)$, is the order of the automorphism[†] group of t and can be calculated using the following recursive relation

$$\sigma(\tau) = 1, \tag{1.9a}$$

$$\sigma([t_1^{k_1} t_2^{k_2} \dots t_m^{k_m}]) = \prod_{i=1}^m k_i! \sigma(t_i)^{k_i}. \tag{1.9b}$$

[†]“An automorphism of a graph is a graph isomorphism with itself, i.e., a mapping from the vertices of the given graph G back to vertices of G such that the resulting graph is isomorphic with G .” (Weisstein, 2016)

- The *density*, denoted by $\gamma(t)$, is defined by the following recursive relation

$$\gamma(\tau) = 1, \quad (1.10a)$$

$$\gamma([t_1^{k_1} t_2^{k_2} \dots t_m^{k_m}]) = r([t_1^{k_1} t_2^{k_2} \dots t_m^{k_m}]) \prod_{i=1}^m \gamma(t_i)^{k_i}. \quad (1.10b)$$

An alternative, and somewhat easier, way of calculating $\gamma(t)$ is to assign the leaf vertices a value of 1 and all other vertices the value of the sum of the values of their child vertices plus 1. $\gamma(t)$ is then the product of all of the values of the vertices.

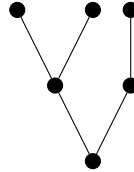
- $\alpha(t)$ is the number of ways of labelling within an ordered set and is calculated using

$$\alpha(t) = \frac{r(t)!}{\sigma(t)\gamma(t)}. \quad (1.11)$$

- $\beta(t)$ is the number of ways of labelling within an unordered set and is calculated using

$$\beta(t) = \frac{r(t)!}{\sigma(t)}. \quad (1.12)$$

Example 1.3.1. Determine the (i) order; (ii) symmetry; (iii) density and (iv) number of labelling combinations for the following tree



Using the notation defined here we have

$$t = [t_1 t_2] = \left[\begin{array}{c} \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \quad \begin{array}{c} \bullet \\ | \\ \bullet \end{array} \right] = [[\tau^2] [\tau]].$$

- (i) The order $r(t)$ is calculated using Eqs. (1.8a) and (1.8b)

$$\begin{aligned} r(t) &= 1 + r(t_1) + r(t_2) \\ &= 1 + r([\tau^2]) + r([\tau]) \\ &= 1 + (1 + 2r(\tau)) + (1 + r(\tau)) \\ &= 1 + (1 + 2(1)) + (1 + 1) \\ &= 6. \end{aligned}$$

Of course this could easily be determined by simply counting the vertices in t .

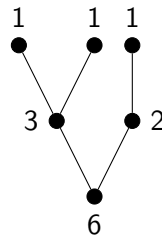
(ii) The symmetry $\sigma(t)$ is calculated using Eqs. (1.9a) and (1.9b)

$$\begin{aligned}
 \sigma(t) &= k_1! \sigma(t_1)^{k_1} k_2! \sigma(t_2)^{k_2} \\
 &= 1! \sigma([\tau^2])^1 1! \sigma([\tau])^1 \\
 &= 2! \sigma(\tau)^2 1! \sigma(\tau)^1 \\
 &= 2(1)^2 \times 1 \\
 &= 2
 \end{aligned}$$

(iii) The density $\gamma(t)$ is calculated using Eqs. (1.10a) and (1.10b)

$$\begin{aligned}
 \gamma(t) &= r(t) \gamma(t_1) \gamma(t_2) \\
 &= 6r([\tau^2]) \gamma(\tau)^2 r([\tau]) \gamma(\tau) \\
 &= 6 \times 3(1)^2 \times 2(1) \\
 &= 36.
 \end{aligned}$$

Using the alternative way of calculating $\gamma(t)$ we can assign the following values to the vertices where the values of a vertex is one more than the sum of the values of its children



and calculating the product of all of the vertex values we have

$$\gamma(t) = 6 \times 3 \times 2 \times 1 \times 1 \times 1 = 36.$$





(iv) The values of $\alpha(t)$ and $\beta(t)$ are calculated using Eqs. (1.11) and (1.12)

$$\begin{aligned}
 \alpha(t) &= \frac{r(t)!}{\sigma(t) \gamma(t)} = \frac{6!}{2(36)} = 10, \\
 \beta(t) &= \frac{r(t)!}{\sigma(t)} = \frac{6!}{2} = 360.
 \end{aligned}$$

□

The values of $r(t)$, $\sigma(t)$, $\gamma(t)$, $\alpha(t)$ and $\beta(t)$ for all trees up to third-order are shown in Table 1.1.

Table 1.1: Rooted trees up to third-order with their values of $r(t)$, $\sigma(t)$, $\gamma(t)$, $\alpha(t)$ and $\beta(t)$.

Tree				
t	τ	$[\tau]$	$[\tau^2]$	$[[\tau]]$
$r(t)$	1	2	3	3
$\sigma(t)$	1	1	2	1
$\gamma(t)$	1	2	3	6
$\alpha(t)$	1	1	1	1
$\beta(t)$	1	2	3	6

1.4 Deriving Explicit Runge-Kutta methods

The derivation of a Runge-Kutta method is achieved by comparing the Taylor series expansion of the general first-order ODE

$$y' = f(x, y),$$

to the equivalent Taylor series expansion of the general Runge-Kutta method from Eqs. (1.3) and (1.4). This produces a series of conditions that the values of a_{ij} , b_i and c_i that need to satisfy in order to produce a Runge-Kutta method.

1.4.1 Taylor expansion of the first-order ODE

Consider the well known Taylor series

$$y(x + h) = y + \sum_{p=1}^{\infty} \frac{h^p}{p!} y^{(p)}. \quad (1.13)$$

To use this to solve the first-order autonomous ODE

$$y' = f(y), \quad (1.14)$$

we need to determine the second-order, third-order etc. derivatives. Using the chain and product rules we have

$$\begin{aligned} y'' &= f'(y)y' \\ &= f'(y)f(y), \\ y''' &= f''(y)f(y)y' + f'(y)f'(y)y' \\ &= f''(y)f(y)f(y) + f'(y)f'(y)f(y), \\ &\vdots \end{aligned}$$

To continue to evaluate higher order derivatives this approach is going to become increasingly complicated, however, Butcher observed that there were similarities between these expressions and rooted trees. Introducing a notation such that $\mathbf{f} = f(y)$, $\mathbf{f}' = f'(y)$, $\mathbf{f}'' = f''(y)$ etc. then



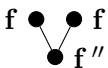
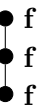
$$y' = \mathbf{f}, \quad (1.15a)$$

$$y'' = \mathbf{f}'\mathbf{f}, \quad (1.15b)$$

$$y''' = \mathbf{f}''(\mathbf{f}, \mathbf{f}) + \mathbf{f}'\mathbf{f}'\mathbf{f}. \quad (1.15c)$$

In the term $\mathbf{f}''(\mathbf{f}, \mathbf{f})$, each of the two \mathbf{f} vectors is one of the operands of the bi-linear operator \mathbf{f}'' (Butcher, 2009). Consider Table 1.2 where the derivatives of the first-order autonomous ODE in Eq. (1.14) have been tabulated next to their corresponding rooted trees.

Table 1.2: Elementary differentials and their equivalent rooted trees.

Elementary differential	Rooted tree
\mathbf{f}	
$\mathbf{f}'\mathbf{f}$	
$\mathbf{f}''(\mathbf{f}, \mathbf{f})$	
$\mathbf{f}'\mathbf{f}'\mathbf{f}$	

Note that a leaf vertex denotes \mathbf{f} and for all other vertices the order of the derivative is the number of child vertices of a vertex. These trees can be linked to *elementary differentials* in Eqs. (1.15a) to (1.15c) by the following

$$F(\tau) = y' = \mathbf{f}, \quad (1.16a)$$

$$F([t_1 t_2 \dots t_\nu]) = y^{(N)} = \mathbf{f}^{(N)}(F(t_1), F(t_2), \dots, F(t_\nu)). \quad (1.16b)$$

The Taylor series expansion, Eq. (1.13), of the ODE in Eq. (1.14) for $x_0 + h$ can be written as

$$y(x_0 + h) = y(x_0) + \sum_{t \in T} \frac{h^{r(t)}}{r(t)!} F(t), \quad (1.17)$$

where T is the set of all rooted trees and $F(t)$ is an elementary differential of the tree t . Using the definition of $\alpha(t)$ in Eq. (1.11), we can simplify Eq. (1.17) to

$$y(x_0 + h) = y(x_0) + \sum_{t \in T} \frac{h^{r(t)}}{\sigma(t)\gamma(t)} F(t). \quad (1.18)$$

Equation (1.18) is the Taylor series expansion of the first-order ODE in Eq. (1.14) written in terms of the trees t .

1.4.2 Taylor series expansion of a numerical approximation

To determine the the values of a_{ij} , b_i and c_i that define a Runge-Kutta method, we need to define expressions that give the order conditions for the trees. For this we use *elementary weights* which is a polynomial function $\Phi(t)$ for a the tree t . The elementary weights are applied to the derivatives in Eq. (1.17) to give the Taylor series expansion of the *computed* solution at $x_0 + h$

$$y(x_0 + h) = y(x_0) + \sum_{t \in T} \frac{\beta(t)h^{r(t)}}{r(t)!} \Phi(t)F(t),$$

which can be simplified using the definition of $\beta(t)$ in Eq. (1.12) to

$$y(x_0 + h) = y(x_0) + \sum_{t \in T} \frac{h^{r(t)}}{\sigma(t)} \Phi(t) F(t). \quad (1.19)$$

So now we have a Taylor series for the numerical approximation in Eq. (1.19) and a Taylor series for the exact solution to the ODE in Eq. (1.18). We need these two to be equivalent up to a given order of accuracy p so equating these results in the following condition on $\Phi(t)$

$$\Phi(t) = \frac{1}{\gamma(t)}, \quad (1.20)$$

for all trees such that $r(t) \leq p$. Equation (1.20) are known as the *order conditions* for the Runge-Kutta method.

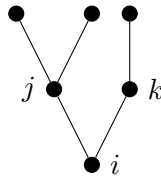
1.4.3 Determining the elementary weights

Given the tree t , the elementary weights, $\Phi(t)$, are determined using the following procedure:

1. for each non-leaf vertex of a tree associate a label i, j, \dots where i is attached to the root vertex;
2. write down a sequence of factors for which the first is b_i ;
3. for each edge of the tree that does not terminate in a leaf vertex write down another factor, a_{ij} say, where an edge joins vertex i to vertex j ;
4. finally for each leaf vertex write down a factor c_j say where j is the label attached to the parent vertex;
5. sum the product of the factors for all possible choices of the labels.

Example 1.4.1. Find the elementary weight of the tree $t = [[\tau^2] [\tau]]$.

Sketching the tree t and labelling the non-leaf vertices gives



therefore

$$\Phi(t) = \sum_{i,j,k} b_i a_{ij} a_{ik} c_j c_k = \sum_{i,j,k} b_i a_{ij} a_{ik} c_j^2 c_k.$$

□

1.4.4 Deriving the order conditions of an ERK method

For a p^{th} order method we need to satisfy Eq. (1.20) for all trees in T with $r(t) \leq p$. This produces a system of nonlinear equations in terms of a_{ij} , b_i and c_i . Since for ERK methods $c_1 = 0$ and the matrix A is lower triangular (Eq. (1.6)) then $a_{ij} = 0$ where $i \leq j$.

Example 1.4.2. Use trees to derive a second-order Runge-Kutta method.

Since we are deriving a second-order ERK, T is the set of all rooted trees of order 2 or less

$$T = \left\{ \bullet, \begin{array}{c} \bullet \\ | \\ \bullet \end{array} \right\} = \{\tau, [\tau]\}.$$

A second-order ERK has $s = 2$ stages so the elementary weights of τ and $[\tau]$ are

$$\begin{aligned} \Phi(\tau) &= \sum_{i=1}^2 b_i = b_1 + b_2, \\ \Phi([\tau]) &= \sum_{i=1}^2 b_i c_i = b_1 c_1 + b_2 c_2, \end{aligned}$$

therefore to satisfy Eq. (1.20) we have

$$\begin{aligned} b_1 + b_2 &= 1, \\ b_1 c_1 + b_2 c_2 &= \frac{1}{2}. \end{aligned}$$

Since $c_1 = 0$ and including the row sum condition the order conditions for a second-order ERK method are

$$\begin{aligned} b_1 + b_2 &= 1, \\ b_2 c_2 &= \frac{1}{2}, \\ a_{21} &= c_2. \end{aligned}$$

Choosing $c_2 = 1$ gives $b_1 = b_2 = 1/2$ and $a_{21} = 1$ so

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

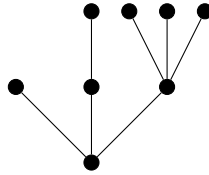
which is the classical second-order explicit Runge-Kutta method

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

□

1.5 Tutorial exercises

- For the tree below, express it using tree notation and calculate $r(t)$, $\sigma(t)$ and $\gamma(t)$.



- Without sketching the tree, calculate $r(t)$, $\sigma(t)$ and $\gamma(t)$ for the tree $t = [\tau[\tau]^2[[\tau^2]]]$.
- Sketch the following trees, determine $r(t)$, $\Phi(t)$, $\gamma(t)$ and write down the expression for the corresponding differentials $F(t)$.
 - $t = [\tau[[\tau^2]\tau]]$
 - $t = [[\tau^2][[\tau^3]][\tau]]$
- Write out the expansion of Eq. (1.17) to third-order and show that it is equivalent to the third-order Taylor series expansion of $y' = f(y)$ given in Eq. (1.13) .
- Use trees to derive the order conditions for a third-order explicit Runge-Kutta method. Hence derive the third-order ERK method when $c_3 = 1$
- Determine $\gamma(t)$ and $\Phi(t)$ for all fifth-order rooted trees.

The solutions to these exercises can be found on page [65](#)

Chapter 2

Adaptive Step Size Control

Runge-Kutta methods are commonly used to solve first-order Initial Value Problems (IVP) of the form

$$y' = f(t, y), \quad t \in [t_{start}, t_{end}], \quad y(t_{start}) = y_0,$$

where y is a function of the dependent variable t^* and y_0 is the known initial value of the solution at $t = t_{start}$.

The approach used by computational methods to solve IVPs begin with the known initial solution y_0 and then integrate the solution over some small step in t such that $t_1 = t_{start} + h$ to obtain an approximation of the solution y_1 at $t = t_1$. This procedure is repeated for y_2, y_3 etc. until we have an approximation of the solution at $t = t_{end}$. Runge-Kutta methods are called *single step* methods since information at a single step, y_n , is required to advance the solution to the next step y_{n+1} whereas *multistep* methods require information from multiple previous steps, y_{n-1}, y_{n-2}, \dots .

The accuracy of the approximation to the solution to the ODE is dependent upon three factors:

- the order of accuracy of the computational method;
- the size of the step length h used to advance the solution;
- the behaviour of the solution.

Improving the order of accuracy of the method is often not straightforward and can place restrictions on the applicability of the method. Reducing the step length used will improve the accuracy of the approximation but since more steps are required to advance through the domain, this will increase the computational cost. The behaviour of the solution will have an affect on the accuracy of the approximation because computational methods are more accurate where the solution is slowly varying and less accurate where there are rapid variations in the solution.

A simple implementation of a Runge-Kutta method may use constant step length for all calculations. The advantage of this is that the step length can be selected to produce an approximation to a desired accuracy and to ensure stability requirements. However, using a fixed step length does not allow the method to take advantage of increasing h where the behaviour of the solution allows. *Adaptive step size control* is a method that attempts to control the value of h based on accuracy requirements.

*From this chapter onward, the independent variable will be represented by t as is commonly used in the literature as opposed to x used in the previous chapter.

2.1 Accuracy of Runge-Kutta methods

Before we discuss step size control an analysis of the accuracy of Runge-Kutta methods is required. Consider the classical fourth-order Runge-Kutta method (RK4) which can be written as

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right), \\ k_4 &= f(t_n + h, y_n + hk_3), \\ y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \end{aligned} \quad (2.1)$$

where y_n is the approximated solution of $y(t_n)$ and h is the step length. We will investigate the derivation of the RK4 method using Simpson's rule.

2.1.1 Derivation of the RK4 method using Simpson's rule

Consider the curve of y over a single step from t_n to t_{n+1} . The value y_{n+1} can be determined by integrating $f(y)$, e.g.,

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} f(y) dt. \quad (2.2)$$

Recall Simpson's rule for estimating a direct integral

$$\int_a^b f(x)dx \approx \frac{h}{6} \left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right), \quad (2.3)$$

then applied to compute an approximation of the integral in Eq. (2.2) we have

$$\int_{t_n}^{t_{n+1}} f(y)dt \approx \frac{h}{6} (f(y_n) + 4f(y_{n+\frac{1}{2}}) + f(y_{n+1})),$$

therefore

$$y_{n+1} = y_n + \frac{h}{6} (f(y_n) + 4f(y_{n+\frac{1}{2}}) + f(y_{n+1})). \quad (2.4)$$

Let $k_1 = f(y_n)$, $k_4 = f(y_{n+1})$ and k_2 and k_3 be defined by

$$f(y_{n+\frac{1}{2}}) = \frac{k_1 + k_2}{2},$$

then Eq. (2.4) can be written as

$$y_{n+1} = y_n + \frac{h}{6} \left(k_1 + 4 \left(\frac{k_2 + k_3}{2} \right) + k_4 \right) = y_n + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4),$$

which is the RK4 method given in Eq. (2.1).

2.1.2 Error estimates

In most practical cases an exact solution to a problem is unknown so it is not possible to calculate the error between the exact solution and a numerical approximation. We can however compare the accuracy of different methods by examining how quickly the error will tend towards zero as the step length is reduced. This uses what is known as 'big-oh' notation.

Definition 2.1.1. Let $f(h)$ be a function of h and define $f(h) = O(h^n)$ such that the following is satisfied

$$\lim_{h \rightarrow 0} \frac{f(h)}{h^n} = C,$$

where $C > 0$.

When h is sufficiently small

$$O(h^n) \approx Ch^n, \quad (2.5)$$

then as $h \rightarrow 0$, $O(h^n) \rightarrow 0$ at a rate faster than $h^n \rightarrow 0$.

To examine the error in the RK4 method, we calculate an estimate of the *local truncation error*, est , using the error for Simpson's rule which is

$$est = \frac{(b-a)^5}{2880} f^{(4)}(x), \quad x \in (a, b). \quad (2.6)$$

Applied to the RK4 method with step length h we have

$$est = \frac{h^5}{2880} y^{(4)}(t).$$

The *global truncation error* $E(y, h)$ after n steps of the RK4 method with step length h is the summation of the local truncation errors for each step, i.e.,

$$E(y, h) = \sum_{n=1}^n \frac{h^5}{2880} y^{(4)}(t) \approx \frac{(b-a)h^4}{2880} y^{(4)}(t) = O(h^4).$$

Using Eq. (2.5) we have

$$E(y, h) = Ch^4,$$

thus halving the step length gives

$$E\left(y, \frac{h}{2}\right) = C\left(\frac{h}{2}\right)^4 = \frac{Ch^4}{16} \approx \frac{1}{16}E(y, h),$$

so reducing the step length by a factor of 2 reduces the global truncation error by a factor of $1/16 = 2^{-4}$.

2.1.3 Improving the accuracy of the numerical solution

We have seen that one way to improve the accuracy of a solution is to simply halve the step length. However this requires many more function evaluations and is computationally inefficient. Another way to improve accuracy is to use a higher order Runge-Kutta method. The problem with this is that the number of stages required does not increase in a linear fashion. For example, for $s \geq 5$ where s is the number of stages, there are no explicit Runge-Kutta methods of order p where $s = p$. To obtain a fifth-order method requires $s = 6$ stages and a $s = 7$ stage method still only results in a fifth-order method.

2.2 Step doubling

The simplest attempt at controlling the step size is by the use of the *step doubling* method. For each step in the method, two solutions are computed using an order p method: $y_{n+1,h}$ is calculated using a single step of length h and $y_{n+1,h/2}$ is calculated using two steps of length $h/2$. These solutions are compared and used to calculate an estimation of the difference between $y_{n+1,h/2}$ and that of an order $p+1$ method. If the difference is less than a desired accuracy, the solution $y_{n+1,h/2}$ is considered accurate enough and is used for y_{n+1} . Furthermore if the error is a lot less than the desired accuracy, the step length h is doubled for the next step thus resulting in fewer steps needed to compute the solution. However, if the error is greater than a desired accuracy, the current step is considered to have failed, the step length is halved and the current step is repeated.

Of course in computing every step at least twice we are increasing the number of calculations required. Consider the RK4 method Eq. (2.1) with step doubling applied. To compute a single step of length h , $y_{n+1,h}$, we require 4 function evaluations for the stage values k_i (Fig. 2.1). For two steps of length $h/2$ we require 8 function evaluations, however, the value of k_1 is the same for both the one step solution and the first step of the two step solution so in total we have 11 function evaluations. Therefore, by checking whether we can double the step length we have increased the computational cost by a factor of just $11/8 = 1.375$ (Press et al., 1993).

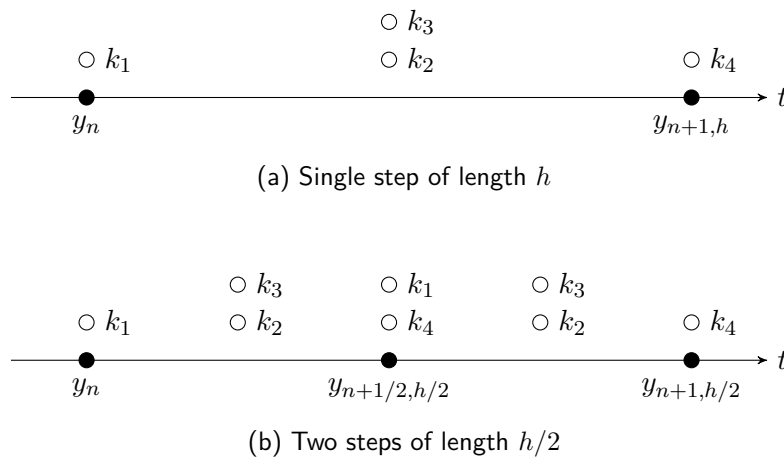


Figure 2.1: Function evaluations for a single step of step doubling when applied to RK4.

2.2.1 Estimating the error

To determine whether we are able to double the step size whilst maintaining a desired accuracy we need to estimate the error between the order p two-step solution and that of an order $p+1$ method. If \tilde{y} denotes the exact solution over a single time step and y_1 and y_2 denote computed solutions using one and two

steps of an order p method, then the exact solutions and computed solutions are related by the following

$$\tilde{y} = y_1 + Ch^{p+1} + O(h^{p+2}), \quad (2.7a)$$

$$\tilde{y} = y_2 + 2C \left(\frac{h}{2}\right)^{p+1} + O(h^{p+2}). \quad (2.7b)$$

Taking the difference between the two computed solutions and solving for C gives

$$\begin{aligned} |y_1 - y_2| &= Ch^{p+1} \left(1 - \frac{1}{2^p}\right) \\ \therefore C &= \frac{|y_1 - y_2|}{(1 - 2^{-p})h^{p+1}}. \end{aligned} \quad (2.8)$$

Substituting C into Eq. (2.7b) results in

$$\tilde{y} = y_2 + \frac{|y_1 - y_2|}{2^p - 1} + O(h^{p+2}), \quad (2.9)$$

So what we have achieved in Eq. (2.9) is increase the order of the accuracy of an order p method to order $p + 1$. For example using RK4, $p = 4$ and

$$\tilde{y} = y_2 + \frac{|y_1 - y_2|}{15} + O(h^6).$$

Let est be the difference between an order p approximation $y_{n+1,h/2}$ and the truncation error for an order $p + 1$ method, then

$$est = \frac{|y_1 - y_2|}{2^p - 1}. \quad (2.10)$$

When applied to solve a system of ODEs, Eq. (2.10) is calculated for each equation in the system and the largest value used to determine whether we need to decrease h .

2.2.2 The step doubling algorithm

The step doubling algorithm is outlined in Algorithm 1.

Algorithm 1 Step doubling method for solving an IVP

Require: A first-order IVP of the form $Y' = f(t, Y)$, $t \in [t_{start}, t_{end}]$, $y(t_{start})$, an initial step length h and desired accuracy tol .

$t_0 \leftarrow t_{start}$

$y_0 \leftarrow y(t_{start})$

$n \leftarrow 0$

Define Runge-Kutta parameters A , b and c

while $t_n < t_{end}$ **do**

$h \leftarrow \min(h, t_{end} - t_n)$

▷ Ensure the h does not overshoot t_{end}

Apply RK method to calculate $y_{n+1, h/2}$

▷ two steps of $h/2$

Apply RK method to calculate $y_{n+1, h}$

▷ one step of h

$est \leftarrow \frac{|y_{n+1, h} - y_{n+1, h/2}|}{2^p - 1}$

if $est > tol$ **then**

▷ Solution does not satisfy accuracy requirements

$h \leftarrow h/2$

else

▷ Solution satisfies accuracy requirements

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

$y_{n+1} \leftarrow y_{n+1, h/2}$

$n \leftarrow n + 1$

if $est \ll tol$ **then**

▷ Solution well within accuracy tolerance

$h \leftarrow 2h$

end if

end if

end while

return (t_0, t_1, \dots, t_n) and (y_0, y_1, \dots, y_n)

2.3 Adaptive step size control

The disadvantages of using the step doubling method is that we have to compute three solutions of the Runge-Kutta method and that we can only double the step length if the solution allows, i.e., it is a double or nothing method. The *adaptive step size control* method attempts to find the largest possible step length that still maintains a desired accuracy.

The solution procedure is as follows: the solution over one step of length h is calculated using a method of order p and $p + 1$. The two solutions are then used to calculate an estimate of the error err ; if $err < tol$ then the step has succeeded, the solution is updated using the $(p + 1)^{\text{th}}$ -order solution, the step length h is increased and the solver advances to the next step; else if $err > tol$ then the step has failed, the step length h is decreased and the current step is repeated.

Using Eq. (2.5) the estimated error using a step length of h is

$$est = Ch^{p+1}.$$

We need to ensure that for a new step length h_{new} we have

$$tol = Ch_{new}^{p+1}.$$

Defining a ratio between the new step length and the current step length $r = h_{new}/h$, then

$$\frac{Ch_{new}^{p+1}}{Ch^{p+1}} = \frac{tol}{est}$$

$$r^{p+1} = \frac{tol}{est}$$

$$r = \left(\frac{tol}{est} \right)^{1/(p+1)}.$$

For robustness, a step-size controller has to respond as smoothly as possible with to abrupt changes in the behaviour of the solution. This means that the step size should not decrease or increase from one step to the next by an excessive ratio. Also, if the user specified tolerance, given as a bound on the norm of the local truncation error estimate, is ever exceeded re-computation and loss of performance will result. Hence, to guard against this as much as possible a 'safety factor' is usually introduced into the computation. If h_{new} is the estimated step size to give a predicted truncation error equal to the tolerance then some smaller value, such as $0.9h_{new}$ is typically used instead. Combining all these ideas we can give a formula for arriving at a factor r , to give a new step size $h_{new} = rh$ following a step for which the error estimate is est . The ratio r is given by

$$r = \max \left(0.5, \min \left(2.0, 0.9 \left(\frac{tol}{est} \right)^{1/(p+1)} \right) \right). \quad (2.11)$$

The three constants, 0.5, 2.0 and 0.9, are all arbitrary and can be regarded as design parameters.

2.3.1 Embedded Runge-Kutta methods

The adaptive step size control method discussed in the previous section requires two solutions or orders p and $p + 1$ to compute the error estimate err . We saw in Section 2.2 that this can be achieved by invoking an order p method using step lengths h and $h/2$, however this means calculating three steps of a method and is computationally inefficient. Fehlberg (1969) derived a Runge-Kutta method where different weightings applied to the stage values can result in a method of order 4 or 5, so a single application of the method can be used to calculate est . Since Fehlberg's original method, several other Runge-Kutta methods that use the same stage values for producing solutions of different orders have been suggested and these methods are known as *embedded Runge-Kutta methods*.

The general form of an fifth-order Runge-Kutta formula is

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + c_2h, y_n + ha_{21}k_1), \\ &\vdots \\ k_6 &= hf(t_n + c_6h, y_n + h(a_{61}k_1 + a_{62}k_2 + a_{63}k_3 + a_{64}k_4 + a_{65}k_5)), \\ y_{n+1} &= y_n + h(b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4 + b_5k_5 + b_6k_6) + O(h^6). \end{aligned}$$

and the embedded fourth-order formula is

$$y_{n+1}^* = y_n + h(b_1^*k_1 + b_2^*k_2 + b_3^*k_3 + b_4^*k_4 + b_5^*k_5 + b_6^*k_6) + O(h^5).$$

To calculate the estimate of the error we simply subtract the order 4 solution from the order 5 solution

$$est = y_{n+1} - y_{n+1}^* = h \sum_{i=1}^6 (b_i - b_i^*)k_i.$$

Note that we do not need to compute y_{n+1}^* to calculate est , we just need the stage values k_i which have already been computed for the fifth-order solution y_{n+1} . The Butcher tableau for an embedded Runge-Kutta method is the same as that for a standard Runge-Kutta method with the addition of the weight for

the fourth-order method, \mathbf{b}^* , listed underneath those for the fifth-order method, \mathbf{b} , i.e.,

\mathbf{c}	A
	\mathbf{b}
	\mathbf{b}^*

The Butcher tableau for three common embedded Runge-Kutta methods are given below

- Runge-Kutta-Fehlberg 4(5) method (RK45) (Fehlberg, 1969)

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	0

(2.12)

- Dormand-Prince method (RKDP) (Dormand and Prince, 1980)

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

(2.13)

- Cash-Karp method (RFCK) (Cash and Karp, 1990)

0						
$\frac{1}{5}$	$\frac{1}{5}$					
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				
$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			
1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		
$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13828}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	
	$\frac{37}{378}$	0	$\frac{250}{621}$	$\frac{125}{594}$	0	$\frac{512}{1771}$
	$\frac{2825}{27648}$	0	$\frac{18575}{48384}$	$\frac{13525}{55296}$	$\frac{277}{14336}$	$\frac{1}{4}$

(2.14)

2.4 Solving systems of equations using explicit Runge-Kutta methods

Most practical problems require the solution of a system of ODEs rather than one single ODE. We can easily adapt the Runge-Kutta methods to cope with systems of multiple ODEs by using matrices.

Consider a system of N first-order ODEs

$$\begin{aligned}
 y_1' &= f_1(t, y_1, y_2, \dots, y_N), \\
 y_2' &= f_2(t, y_1, y_2, \dots, y_N), \\
 &\vdots \\
 y_N' &= f_N(t, y_1, y_2, \dots, y_N),
 \end{aligned}$$

we can write this using vector notation as

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}),$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} f_1(t, \mathbf{y}) \\ f_2(t, \mathbf{y}) \\ \vdots \\ f_N(t, \mathbf{y}) \end{pmatrix}.$$

The general form of a Runge-Kutta method shown in Eqs. (1.3) and (1.4) then becomes

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \sum_{i=1}^s \mathbf{k}_i b_i, \quad (2.15a)$$

$$\mathbf{k}_i = \mathbf{f} \left(t_n + c_i h, \mathbf{y}_n + h \sum_{j=1}^s \mathbf{k}_j a_{ij} \right), \quad (2.15b)$$

where \mathbf{y}_n is a vector containing the current known solutions, \mathbf{y}_{n+1} is a vector of the solutions advanced by one step of length h and \mathbf{k}_i is a vector of the intermediate stage values (all of these vectors are column vectors).

Defining K as an $s \times N$ matrix containing the vectors \mathbf{k}_i , i.e.,

$$K = \begin{pmatrix} \mathbf{k}_1^T \\ \mathbf{k}_2^T \\ \vdots \\ \mathbf{k}_s^T \end{pmatrix}.$$

then Eq. (2.15b) can be calculated using

$$\begin{aligned} \mathbf{k}_i &= \mathbf{f}(t_n + c_i h, \mathbf{y}_n + h(a_{i1}\mathbf{k}_1 + a_{i2}\mathbf{k}_2 + \dots + a_{is}\mathbf{k}_s)) \\ &= \mathbf{f} \left(t_n + c_i h, \mathbf{y}_n + h \begin{pmatrix} a_{i1} & a_{i2} & \dots & a_{is} \end{pmatrix} \begin{pmatrix} \mathbf{k}_1^T \\ \mathbf{k}_2^T \\ \vdots \\ \mathbf{k}_s^T \end{pmatrix} \right) \\ &= \mathbf{f}(t_n + c_i h, \mathbf{y}_n + h[A]_{i,*}K), \end{aligned}$$

where $[A]_{i,*}$ denotes the i th row of A . Once the stage values have been calculated we can also use matrix multiplication to update the solution with Eq. (2.15a)

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h(b_1\mathbf{k}_1 + b_2\mathbf{k}_2 + \dots + b_s\mathbf{k}_s) \\ &= \mathbf{y}_n + h\mathbf{b}^T K. \end{aligned}$$

2.4.1 Adaptive step size control algorithm

The algorithm for solving a system of ODEs using an ERK method with adaptive step size control is outlined in Algorithm 2.

Algorithm 2 Solving a system of ODEs using an ERK with adaptive step size control.

Require: A first-order IVP of the form $\mathbf{y}'(t) = f(t, \mathbf{y})$, $t \in [a, b]$, $\mathbf{y}(a)$, initial step length h and the accuracy tolerance tol

$t_0 \leftarrow a$

$\mathbf{y}_0 \leftarrow \mathbf{y}(a)$

$n \leftarrow 0$

Define embedded ERK parameters A , \mathbf{b} , \mathbf{b}^* and \mathbf{c}

while $t_n < b$ **do**

$K \leftarrow 0$

▷ K is an $N \times s$ matrix

for $i = 1, \dots, s$ **do**

$[K]_{i,*} \leftarrow f(t_n + c_i h, y_n + h[A]_{i,*} K)$

▷ $[K]_{i,*}$ denotes the i th row of K

end for

$est \leftarrow h \max |K^T(\mathbf{b} - \mathbf{b}^*)|$

if $est < tol$ **then**

$\mathbf{y}_{n+1} \leftarrow \mathbf{y}_n + h\mathbf{b}^T K$

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

$n \leftarrow n + 1$

end if

$r \leftarrow \max \left(0.5, \min \left(2.0, 0.9 \left(\frac{tol}{est} \right)^{1/(p+1)} \right) \right)$

$h \leftarrow rh$

end while

return $\mathbf{t} = (t_0, t_1, \dots, t_n)$ and $\mathbf{y} = (y_0, y_1, \dots, y_n)$

2.5 Tutorial exercises

1. An IVP is given by

$$y' = -21y + e^{-t}, \quad t \in [0, 1], \quad y_0 = 0.$$

- (a) Write a MATLAB program to compute one step of Fehlberg's method for this IVP using a step length of $h = 0.05$.
 - (b) Use Eq. (2.11) to calculate the size of the new step length using $tol = 10^{-4}$. What does the value of est tell you about the step you have just calculated?
 - (c) Modify your program so that computes the solution over the whole domain. How many successful steps, failed steps and function evaluations were used in total? Produce a plot of your solution.
2. The Lotka - Volterra equations describe the dynamics of the interaction between two species, one is a predator and the other is the prey

$$\begin{aligned}\dot{x} &= \alpha x - \beta xy, \\ \dot{y} &= \sigma xy - \gamma y,\end{aligned}$$

where x and y are the populations of the prey and predator species respectively and $\alpha, \beta, \sigma, \gamma > 0$ are parameters that describe the interaction between the two species.

- (a) Use the RK4 method to compute the solution to the Lotka-Volterra equations over the domain $t \in [0, 20]$ with initial populations $x = y = 1$ and parameters $\alpha = 2/3$, $\beta = 4/3$, $\sigma = \gamma = 1$ using a step length of $h = 0.1$. Produce a plot of the values of x and y against t on the same set of axes.
- (b) Repeat the calculations from question 1 using the RK4 method with step doubling with an error tolerance of $tol = 10^{-6}$. You may use the MATLAB function `RKSD.m` on the Moodle area for this unit to help you.
- (c) Repeat the calculations from question 2 using MATLAB's `ode45` solver.
- (d) Produce a table that displays the CPU time, the number of function evaluations, number of failed steps and number of successful steps for each method used to solve this problem.
- (e) Access the code for the MATLAB function `ode45` using the command 'edit `ode45`' in the MATLAB command window and identify the embedded Runge-Kutta method used.

The solutions to these exercises are given on page 67.

Chapter 3

Implicit Runge-Kutta Methods

We have already seen that Explicit Runge-Kutta (ERK) methods are very useful when solving ODEs. However, for certain types of problems ERK methods will struggle due to stability constraints which require the step lengths to be very small. Implicit Runge-Kutta (IRK) methods have much less restrictive stability constraints allowing us to solve these problems.

3.1 General form of an implicit Runge-Kutta method

The general form of an Implicit Runge-Kutta (IRK) method can be expressed as the following Butcher tableau.

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

which leads to the following expressions for the stage values k_i

$$\begin{aligned} k_1 &= f(t_n + c_1 h, y_n + h \sum_{j=1}^s a_{1j} k_j), \\ k_2 &= f(t_n + c_2 h, y_n + h \sum_{j=1}^s a_{2j} k_j), \\ &\vdots \\ k_s &= f(t_n + c_s h, y_n + h \sum_{j=1}^s a_{sj} k_j). \end{aligned}$$

The solution is updated using a weighted sum of the stage values

$$y_{n+1} = y_n + h \sum_{i=1}^s b_i k_i. \tag{3.1}$$

For an explicit method the calculation of the stage values is straight forward, we calculate k_1 which is then used in the calculation of k_2 and so on until k_s . With an implicit method things are more complicated since the expressions for the stage values are implicit relationships. For example, consider the expression for k_1 with the summation term expanded out

$$k_1 = f(t_n + c_1 h, y_n + h(a_{11}k_1 + a_{12}k_2 + \dots a_{1s}k_s)).$$

Here k_1 also appears on the right-hand side of the expression so how do we calculate it? One way could be to write the expressions for the stage values as a system of equations, let $Y_i = y_n + h \sum_{j=1}^s a_{ij}k_j$ then

$$k_i = f(t_n + c_i h, Y_i),$$

so

$$Y_i = y_n + h \sum_{j=1}^s a_{ij}f(t_n + c_j h, Y_j), \quad (3.2)$$

and Eq. (3.1) becomes

$$y_{n+1} = y_n + h \sum_{i=1}^s b_i f(t_n + c_i h, Y_i). \quad (3.3)$$

To compute a step of an IRK method we solve Eq. (3.2) for Y_i which are then used to update the solution using Eq. (3.3).

Example 3.1.1. Compute one step of the third-order RadauIA method using $h = 0.1$ for the following IVP

$$y' = t - y, \quad t_0 = 0, \quad y(0) = 1.$$

The Butcher tableau for the RadauIA method is

0	1/4	-1/4
2/3	1/4	5/12
	1/4	3/4

Using Eq. (3.2) we have

$$\begin{aligned} Y_1 &= y_n + h(a_{11}f(t_n + c_1 h, Y_1) + a_{12}f(t_n + c_2 h, Y_2)), \\ Y_2 &= y_n + h(a_{21}f(t_n + c_1 h, Y_1) + a_{22}f(t_n + c_2 h, Y_2)). \end{aligned}$$

In this case $f(t, y) = t - y$ and substituting the RadauIA method parameters

$$Y_1 = y_n + h \left(\frac{1}{4}(t_n + 0h - Y_1) - \frac{1}{4} \left(t_n + \frac{2}{3}h - Y_2 \right) \right),$$

$$Y_2 = y_n + h \left(\frac{1}{4}(t_n + 0h - Y_1) + \frac{5}{12} \left(t_n + \frac{2}{3}h - Y_2 \right) \right).$$

Rearranging so that Y_1 and Y_2 are on the left-hand side

$$\begin{aligned} \left(1 + \frac{h}{4}\right) Y_1 - \frac{h}{4} Y_2 &= y_n - \frac{1}{6} h^2, \\ \frac{h}{4} Y_1 + \left(1 + \frac{5}{12} h\right) Y_2 &= y_n + \frac{2}{3} t_n + \frac{5}{18} h^2 \end{aligned}$$

which can be written as the matrix equation

$$\begin{pmatrix} 1 + h/4 & -h/4 \\ h/4 & 1 + 5h/12 \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} y_n - \frac{1}{6} h^2 \\ y_n + \frac{2}{3} t_n + \frac{5}{18} h^2 \end{pmatrix}.$$

For the first step $t_0 = 0$, $y_0 = 1$, $h = 0.1$ which leads to the linear system

$$\begin{pmatrix} 1.0250 & -0.0250 \\ 0.0250 & 1.0417 \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} 0.9833 \\ 1.0694 \end{pmatrix}.$$

which is solved to give $Y_1 = 0.9984$, $Y_2 = 1.0027$. We now use Eq. (3.3) to update the solution

$$\begin{aligned} y_1 &= y_0 + h(b_1 f(t_0 + c_1 h, Y_1) + b_2 f(t_0 + c_2 h, Y_2)) \\ &= 1 + 0.1 \left(\frac{1}{4}(0 - 0.9984) + \frac{3}{4} \left(0 + \frac{2}{3}(0.1) - 1.0027 \right) \right) \\ &= 0.9048. \end{aligned}$$

□

Example 3.1.1 was fairly trivial, the IVP being solved involved a single linear ODE. This meant that it was a simple to solve for Y_i since we had a linear system and can use one of many methods (e.g., LU factorisation). Also, this IVP was described by a single equation where in practice we will have systems of equations to solve.

3.2 Newton's method

Newton's method (often called *the Newton-Raphson method*) is a numerical method for finding the roots of a differentiable function. Consider Fig. 3.1 that shows the function $g(x)$. Given an estimate of the root $x^{(k)}$, the tangent line at the point $g(x^{(k)})$ is determined and where this tangent line intersects the x -axis becomes the new estimate $x^{(k+1)}$. The next estimate of the root is determined where the tangent line drawn at $g(x^{(k+1)})$ crosses the x -axis and so on. The process continues until two successive guess values converge to a given tolerance (some small value that determines the level of accuracy required).

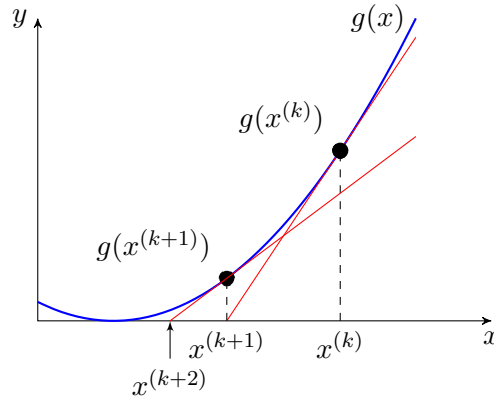


Figure 3.1: Newton's method for calculating the root of the function $g(x)$.

The tangent of the line to $g(x)$ at $x = x_k$ is

$$y = g(x^{(k)}) + g'(x^{(k)})(x - x^{(k)}).$$

This tangent line crosses the horizontal axis at $(x^{(k+1)}, 0)$ so

$$0 = g(x^{(k)}) + g'(x^{(k)})(x^{(k+1)} - x^{(k)}), \quad (3.4)$$

which can be rearranged to make $x^{(k+1)}$ the subject

$$x^{(k+1)} = x^{(k)} - \frac{g(x^{(k)})}{g'(x^{(k)})}. \quad (3.5)$$

Equation (3.5) is Newton's method for finding the root of a single variable function.

3.2.1 Newton's method for a system of equations

We can extend Newton's method for the one variable case as seen above to solve a system of N homogeneous equations in N unknowns, i.e.,

$$\begin{aligned} g_1(x_1, x_2, \dots, x_N) &= 0, \\ g_2(x_1, x_2, \dots, x_N) &= 0, \\ &\vdots \\ g_N(x_1, x_2, \dots, x_N) &= 0. \end{aligned}$$

Introducing vector notation such that $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$ and $\mathbf{g}(\mathbf{x}) = (g_1, g_2, \dots, g_N)^T$ then we can write the above system as the vector equation

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

For the one variable case the linear approximation of the function $g(x)$ using the known value $x^{(k)}$ was shown in Eq. (3.4). The vector form of this equation is

$$\mathbf{0} = \mathbf{g}(\mathbf{x}^{(k)}) + J(\mathbf{x}^{(k)})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) \quad (3.6)$$

where $J(\mathbf{x}^{(k)})$ is the $n \times n$ Jacobian matrix

$$J(\mathbf{x}^{(k)}) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_N}{\partial x_1} & \cdots & \frac{\partial g_N}{\partial x_N} \end{pmatrix}. \quad (3.7)$$

Rearranging Eq. (3.6) gives

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J^{-1}(\mathbf{x}^{(k)})\mathbf{g}(\mathbf{x}^{(k)}). \quad (3.8)$$

This is the vector equivalent of the single variable equation Eq. (3.5). Unfortunately the calculation of the inverse matrix $J^{-1}(\mathbf{x}^{(k)})$ is too computationally expensive to be of practical use. Instead we use a two-stage process, let $\Delta\mathbf{x} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ then Eq. (3.8) can be written as

$$J(\mathbf{x}^{(k)})\Delta\mathbf{x} = -\mathbf{g}(\mathbf{x}^{(k)}). \quad (3.9)$$

We know the values of $J(\mathbf{x}^{(k)})$ and $-\mathbf{g}(\mathbf{x}^{(k)})$ so this is a linear system that can be solved to give $\Delta\mathbf{x}$. Once we have a solution for $\Delta\mathbf{x}$ we can calculate the improved estimate $\mathbf{x}^{(k+1)}$ using

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}. \quad (3.10)$$

3.3 Solving systems of equations using IRK methods

Solving systems of ODEs using an IRK method presents the added complication that the calculation of the stage values has to be done for each equation in the system. This means that for a system of N ODEs, each stage value Y_i will be an N element vector. To simplify the notation we can make use of tensor products.

Definition 3.3.1. Tensor product

The *tensor product* (also known as the *Kronecker tensor product*) of two matrices A and B is denoted by $A \otimes B$ and is defined by the following

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \otimes \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} & a_{12} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \\ a_{21} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} & a_{22} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix}.$$

Consider the solution of a system of N first-order ODEs

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}),$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} f_1(t, \mathbf{y}) \\ f_2(t, \mathbf{y}) \\ \vdots \\ f_N(t, \mathbf{y}) \end{pmatrix}.$$

Applying an IRK method to solve this system gives

$$\mathbf{k}_i = \mathbf{f}(t_n + c_i h, \mathbf{y}_n + h \sum_{j=1}^s a_{ij} \mathbf{k}_j),$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \sum_{i=1}^s b_i \mathbf{k}_i.$$

Let $Y_i = \mathbf{y}_n + h \sum_{j=1}^s a_{ij} \mathbf{k}_j$ then

$$Y_i = \mathbf{y}_n + h \sum_{j=1}^s a_{ij} \mathbf{f}(t_n + c_j h, Y_j) \quad (3.11)$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \sum_{i=1}^s b_i \mathbf{f}(t_n + c_i h, Y_i) \quad (3.12)$$

Writing Eq. (3.11) for $i = 1 \dots s$

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_s \end{pmatrix} = \begin{pmatrix} \mathbf{y}_n \\ \mathbf{y}_n \\ \vdots \\ \mathbf{y}_n \end{pmatrix} + h \begin{pmatrix} \sum_{j=1}^s a_{1j} \mathbf{f}(t_n + c_j h, Y_j) \\ \sum_{j=1}^s a_{2j} \mathbf{f}(t_n + c_j h, Y_j) \\ \vdots \\ \sum_{j=1}^s a_{sj} \mathbf{f}(t_n + c_j h, Y_j) \end{pmatrix}$$

Each summation in the vector on the right-hand side is applied to all equations in the system, for example the first summation results in the vector

$$\sum_{j=1}^s a_{1j} \mathbf{f}(t_n + c_j h, Y_j) = \begin{pmatrix} \sum_{j=1}^s a_{1j} f_1(t_n + c_j h, Y_j) \\ \sum_{j=1}^s a_{1j} f_2(t_n + c_j h, Y_j) \\ \vdots \\ \sum_{j=1}^s a_{1j} f_N(t_n + c_j h, Y_j) \end{pmatrix},$$

and similar for all other elements. The terms within the summations on the right-hand side can be written using the $N \times N$ identity matrix I_N

$$\begin{pmatrix} a_{1j}f_1(t_n + c_jh, Y_j) \\ a_{1j}f_2(t_n + c_jh, Y_j) \\ \vdots \\ a_{1j}f_N(t_n + c_jh, Y_j) \end{pmatrix} = a_{1j} \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} f_1(t_n + c_jh, Y_j) \\ f_2(t_n + c_jh, Y_j) \\ \vdots \\ f_N(t_n + c_jh, Y_j) \end{pmatrix} = a_{1j}I_N \mathbf{f}(t_n + c_jh, Y_j).$$

So for all s stages of the IRK method we can write

$$\begin{aligned} \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_s \end{pmatrix} &= \begin{pmatrix} \mathbf{y}_n \\ \mathbf{y}_n \\ \vdots \\ \mathbf{y}_n \end{pmatrix} + h \begin{pmatrix} a_{11}I_N & a_{12}I_N & \cdots & a_{1s}I_N \\ a_{21}I_N & a_{22}I_N & \cdots & a_{2s}I_N \\ \vdots & \vdots & \ddots & \vdots \\ a_{s1}I_N & a_{s2}I_N & \cdots & a_{ss}I_N \end{pmatrix} \begin{pmatrix} \mathbf{f}(t_n + c_1h, Y_1) \\ \mathbf{f}(t_n + c_2h, Y_2) \\ \vdots \\ \mathbf{f}(t_n + c_sh, Y_s) \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{y}_n \\ \mathbf{y}_n \\ \vdots \\ \mathbf{y}_n \end{pmatrix} + h(A \otimes I_N) \begin{pmatrix} \mathbf{f}(t_n + c_1h, Y_1) \\ \mathbf{f}(t_n + c_2h, Y_2) \\ \vdots \\ \mathbf{f}(t_n + c_sh, Y_s) \end{pmatrix} \end{aligned}$$

We can now write the original IRK method for a system of ODEs by defining the following vectors

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_s \end{pmatrix}, \quad F(t_n \mathbf{e} + h\mathbf{c}, Y) = \begin{pmatrix} \mathbf{f}(t_n + c_1h, Y_1) \\ \mathbf{f}(t_n + c_2h, Y_2) \\ \vdots \\ \mathbf{f}(t_n + c_sh, Y_s) \end{pmatrix}, \quad \mathbf{e} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix},$$

where Y and $F(t_n \mathbf{e} + h\mathbf{c}, Y)$ are $Ns \times 1$ element vectors then we can write Eq. (3.2) as

$$Y = \mathbf{e} \otimes \mathbf{y}_n + h(A \otimes I_N)F(t_n \mathbf{e} + h\mathbf{c}, Y), \quad (3.13)$$

and Eq. (3.1) becomes

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h(\mathbf{b}^T \otimes I_N)F(t_n \mathbf{e} + h\mathbf{c}, Y). \quad (3.14)$$

We solve Eq. (3.13) using Newton's method and then update the solution using Eq. (3.14).

3.3.1 Calculating the stage values

To solve the matrix equation Eq. (3.13) for Y we need to apply Newton's method. Writing it in the form $G(Y) = 0$ we have

$$G(Y) = Y - \mathbf{e} \otimes \mathbf{y}_n - h(A \otimes I_N)F(t_n \mathbf{e} + h\mathbf{c}, Y), \quad (3.15)$$

and the Jacobian matrix is

$$J(Y) = I_{Ns} - h(A \otimes I_N)F_Y(t_n \mathbf{e} + h\mathbf{c}, Y), \quad (3.16)$$

where $F_Y \in \mathbb{R}^{Ns \times Ns}$. This is equivalent to

$$J(Y) = \begin{pmatrix} I_N - ha_{11}\mathbf{f}_Y(t_n + c_1h, Y_1) & -ha_{12}\mathbf{f}_Y(t_n + c_2h, Y_2) & \cdots & -ha_{2s}\mathbf{f}_Y(t_n + c_sh, Y_s) \\ -ha_{21}\mathbf{f}_Y(t_n + c_1h, Y_1) & I_N - ha_{22}\mathbf{f}_Y(t_n + c_2h, Y_2) & \cdots & -ha_{2s}\mathbf{f}_Y(t_n + c_sh, Y_s) \\ \vdots & \vdots & \ddots & \vdots \\ -ha_{s1}\mathbf{f}_Y(t_n + c_1h, Y_1) & -ha_{s2}\mathbf{f}_Y(t_n + c_2h, Y_2) & \cdots & I_N - ha_{ss}\mathbf{f}_Y(t_n + c_sh, Y_s) \end{pmatrix}.$$

The terms $\mathbf{f}_Y(t_n + c_ih, Y_i) \in \mathbb{R}^{N \times N}$ are Jacobian matrices for the system of ODEs being solved. These will change as we iterate Newton's method and the values of $F(t_n, Y_i)$ change. Calculating the Jacobian matrix at every iteration can be very expensive to compute for large systems so instead use an approximation of F_Y

$$F_Y(t, Y) \approx I_s \otimes \mathbf{f}_Y(t_n, \mathbf{y}_n),$$

so Eq. (3.16) is simplified to

$$J(Y) \approx I_{Ns} - h(A \otimes I_N)(I_s \otimes \mathbf{f}_Y(t_n, \mathbf{y}_n)). \quad (3.17)$$

Note that $\mathbf{f}_Y(t_n, \mathbf{y}_n)$ is constant for the current step and only changes when t_n and \mathbf{y}_n are updated so it can be pre-calculated prior to performing the Newton iterations. In cases where it is not possible to evaluate the derivatives so we can use finite-difference approximations of the individual derivatives, i.e.,

$$\mathbf{f}_Y(t_n, \mathbf{y}_n) \approx \frac{\mathbf{f}(t_n, \mathbf{y}_n + \delta y) - \mathbf{f}(t_n, \mathbf{y}_n)}{\delta y},$$

for each element in $\mathbf{f}_Y(t_n, \mathbf{y}_n)$ where δy is some small value. Modern sophisticated solvers offer a choice of whether to calculate the Jacobian matrix at each step or even monitor the convergence rate of Newton's method to determine whether the Jacobian matrix requires calculating to reduce the computation time required.

Once the Jacobian matrix is calculated the Newton iterations proceed by solving the linear system

$$J(Y)\Delta Y = -G(Y^{(k)}), \quad (3.18)$$

for ΔY and then updating $Y^{(k)}$ using

$$Y^{(k+1)} = Y^{(k)} + \Delta Y, \quad (3.19)$$

where $Y^{(0)} = \mathbf{e} \otimes \mathbf{y}_n$. Iterations continue until $\|\Delta Y\| < \text{tol}$. The steps used to solve an IVP using an IRK method are given in Algorithm 3.

Algorithm 3 Solving an IVP using an IRK method.

Require: An IVP defined using a system of N first-order ODEs of the form $\mathbf{y}'(t) = f(t, \mathbf{y})$, $t \in [t_{\min}, t_{\max}]$, $\mathbf{y}(a)$, initial step length h and an accuracy tolerance tol

$\mathbf{y}_0 \leftarrow \mathbf{y}(a)$
 $t_0 \leftarrow t_{\min}$
 $n \leftarrow 0$
 Define IRK parameters A , \mathbf{b} , \mathbf{c} and s

while $t_n < t_{\max}$ **do**
 $Y \leftarrow \mathbf{e} \otimes \mathbf{y}_n$
 $J(Y) \leftarrow I_{Ns} - h(A \otimes I_N)(I_s \otimes \mathbf{f}_{\mathbf{y}}(t_n, \mathbf{y}_n))$
 $\text{err} \leftarrow 1$
 while $\text{err} > \text{tol}$ **do**
 $G(Y) \leftarrow Y - \mathbf{e} \otimes \mathbf{y}_n - h(A \otimes I_N)F(t_n \mathbf{e} + h\mathbf{c}, Y)$
 Solve $J(Y)\Delta Y = -G(Y)$ for ΔY
 $Y \leftarrow Y + \Delta Y$
 $\text{err} \leftarrow \|\Delta Y\|$
 end while
 $\mathbf{y}_{n+1} \leftarrow \mathbf{y} + h(\mathbf{b}^T \otimes I_N)F(t_n \mathbf{e} + h\mathbf{c}, Y)$
 $t_{n+1} \leftarrow t_n + h$
 $n \leftarrow n + 1$
end while
return $\mathbf{t} = (t_0, t_1, \dots, t_n)$ and $\mathbf{y} = (\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_n)$

Example 3.3.1. Calculate a single step of the third-order Radau IA method using $h = 0.1$ for the following IVP

$$\begin{aligned} y_1' &= 2y_1 + y_2, & y_1(0) &= 1, \\ y_2' &= y_1 y_2, & y_2(0) &= 0. \end{aligned}$$

Recall that the Radau IA method parameters are

$$A = \begin{pmatrix} 1/4 & -1/4 \\ 1/4 & 5/12 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 2/3 \end{pmatrix}.$$

In this case it is possible to differentiate $\mathbf{f}(t_n, \mathbf{y}_n)$ so

$$\mathbf{f}_y(t, \mathbf{y}) = \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ y_2 & y_1 \end{pmatrix},$$

therefore we can calculate the Jacobian matrix using Eq. (3.17)

$$\begin{aligned} J(Y) &= I_4 - h(A \otimes I_2)(I_2 \otimes \mathbf{f}_y(t, \mathbf{y})) \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - h \begin{pmatrix} 1/4 & 0 & -1/4 & 0 \\ 0 & 1/4 & 0 & -1/4 \\ 1/4 & 0 & 5/12 & 0 \\ 0 & 1/4 & 0 & 5/12 \end{pmatrix} \begin{pmatrix} 2 & 1 & 0 & 0 \\ y_2 & y_1 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & y_2 & y_1 \end{pmatrix} \\ &= \begin{pmatrix} 1 - h/2 & -h/4 & h/4 & h/4 \\ -hy_2/4 & 1 - hy_1/4 & hy_2/4 & hy_1/4 \\ -h/4 & -h/4 & 1 - 5h/12 & -5h/12 \\ -hy_2/4 & -hy_1/4 & -5hy_2/12 & 1 - 5hy_1/12 \end{pmatrix}, \end{aligned}$$

so for the first step of the IRK method, $y_1 = 1$, $y_2 = 0$ and $h = 0.1$, and the Jacobian matrix is

$$J(Y) = \begin{pmatrix} 0.9500 & -0.0250 & 0.0500 & 0.0250 \\ 0 & 0.9750 & 0 & 0.0250 \\ -0.0500 & -0.0250 & 0.9167 & -0.0417 \\ 0 & -0.0250 & 0 & 0.9583 \end{pmatrix}$$

Now we need to perform the Newton iterations. Initialising $Y^{(0)}$ and $F(t_0\mathbf{e} + h\mathbf{c}, Y^{(0)})$

$$Y^{(0)} = \mathbf{e} \otimes \mathbf{y}_0 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \Rightarrow \quad F(t_0\mathbf{e} + h\mathbf{c}, Y^{(0)}) = \begin{pmatrix} 2 \\ 0 \\ 2 \\ 0 \end{pmatrix},$$

so

$$\begin{aligned} G(Y^{(0)}) &= Y^{(0)} - \mathbf{e} \otimes \mathbf{y}_0 - h(A \otimes I_2)F(t_0\mathbf{e} + h\mathbf{c}, Y^{(0)}) \\ &= \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} - 0.1 \begin{pmatrix} 1/4 & 0 & -1/4 & 0 \\ 0 & 1/4 & 0 & -1/4 \\ 1/4 & 0 & 5/12 & 0 \\ 0 & 1/4 & 0 & 5/12 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1.3333 \\ 0 \end{pmatrix}. \end{aligned}$$

Solving $J(Y)\Delta Y = -G(Y^{(0)})$ gives $\Delta Y = (-0.0076, 0, 0.1450, 0)^T$ therefore

$$Y^1 = Y^{(0)} + \Delta Y = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -0.0076 \\ 0 \\ 0.1450 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.9924 \\ 0 \\ 1.1450 \\ 0 \end{pmatrix},$$

and $\|\Delta Y\| = 0.1452$. Doing the same calculations using $Y^{(1)}$ results in very small values for ΔY (less than 10^{-13}) and Newton's method is considered to have converged after just 2 iterations to the solution $Y = Y^{(1)}$.

Finally we just need to update the solution using Eq. (3.14)

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{y}_0 + h(\mathbf{b}^T \otimes I_2)F(t_0\mathbf{e} + h\mathbf{c}, Y) \\ &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 0.1 \begin{pmatrix} 1/4 & 0 & 3/4 & 0 \\ 0 & 1/4 & 0 & 3/4 \end{pmatrix} \begin{pmatrix} 1.9847 \\ 0 \\ 2.2901 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 1.2214 \\ 0 \end{pmatrix}. \end{aligned}$$

□

The MATLAB code used to compute the solution to Example 3.3.1 is shown in listing 3.1. Note that the `kron` command computes the tensor product and inline functions are used to define the system of ODEs, the vector $F(te + hc, Y)$ and the Jacobian matrix.

```
% Clear workspaces
clear
clc

% Define ODE functions and Jacobian
f = @(t, y) [ 2 * y(1) + y(2) ; y(1) * y(2) ];
F = @(t, Y) [ f(t(1), Y(1:2)) ; f(t(1), Y(3:4)) ];
Jac = @(t, y) [ 2, 1 ; y(2), y(1) ];

% Define IVP parameters
y0 = [1 ; 0];           % initial values of y
t0 = 0;                 % initial value of t
N = 2;                  % no. equations in the system
h = 0.1;                % step length
tol = 1d-6;             % accuracy tolerance

% Define Radau method parameters
A = [ 1/4, -1/4 ; 1/4, 5/12 ];
b = [ 1/4 ; 3/4 ];
c = [ 0 ; 2/3 ];
s = 2;

% Initialise solution arrays
y = [y0'];
t = [t0];
n = 1;

% Use Newton's method to solve for the stage values Y
e = ones(s, 1);
```

```

Y = kron(e, y(n, :)'');
J = eye(N * s) - h * kron(A, eye(N)) * kron(eye(s), Jac(t(n), y(n, :)));
err = 1;

while err > tol
    G = Y - kron(e, y(n, :)'') - h * kron(A, eye(N)) * F(t(n) * e + h * c, Y);
    DY = J \ -G;
    Y = Y + DY;
    err = norm(DY);
end

% Update solution
y(n+1, :) = ( y(n, :)' + h * kron(b', eye(N)) * F(t(n) * e + h * c, Y) )';
t(n+1) = t(n) + h;

```

Listing 3.1: MATLAB code used to compute a single step of the Radau IA method in Example 3.3.1

3.4 Tutorial exercises

1. An IVP is described by the following

$$y' = 4ty, \quad t \in [0, 1], \quad y(0) = 1.$$

- (a) Compute a single step of the third-order Radau IA method for this problem using a step length $h = 0.1$.

$$\begin{array}{c|cc} 0 & 1/4 & -1/4 \\ 2/3 & 1/4 & 5/12 \\ \hline & 1/4 & 3/4 \end{array}$$

- (b) Write a MATLAB program to compute the solutions over the domain $t \in [0, 1]$.

2. The motion of a simple pendulum is described by the following ODE

$$\ddot{\theta} = -\frac{g}{\ell} \sin(\theta),$$

where θ is the angle between the chord and the vertical, $g = 9.81\text{ms}^{-2}$ is the acceleration due to gravity and ℓ is the length of the chord.

A pendulum of length $\ell = 1\text{m}$ is pulled so that $\theta = 1$ and then released.

- (a) Write this ODE as a system of 2 first-order ODEs
- (b) Compute a single step of the fourth-order Gauss-Legendre method given by the Butcher tableau below using a step length $h = 0.1$ and a convergence tolerance of $\text{tol} = 10^{-4}$.

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

- (c) Calculate the solution over $t \in [0, 10]$.

3. Solve the problem from question 2 using the `ode23s` solver and compare the solution with the one you obtained in question 2. How many function evaluations and solutions of linear systems were required for the `ode23s` solver?

The solutions to these exercises are given in Appendix [A.4](#) on page [71](#).

Chapter 4

Stability & Order Stars

4.1 Absolute stability

The stability of ODE solver methods can be examined using the linear test equation

$$y' = \lambda y \quad (4.1)$$

where $\lambda \in \mathbb{C}$. A single step of an ODE solver advances the solution by adding the step length h to the current value dependent variable t_n and approximates y_{n+1} using the current value of the solution y_n . We can write an ODE solver using

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

where $z = h\lambda$ and $R(z)$ is known as the *stability function* for the method under consideration. For single step methods, the stability function can be represented by

$$R(z) = \frac{P(z)}{Q(z)}, \quad (4.2)$$

where P and Q are polynomial functions. For example, consider the Euler method applied to solve Eq. (4.1)

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

therefore the stability function for the Euler method is

$$R(z) = 1 + z, \quad (4.3)$$

so $P(z) = 1 + z$ and $Q(z) = 1$. Doing similar for the implicit Euler method results in

$$R(z) = \frac{1}{1 - z}, \quad (4.4)$$

so $P(z) = 1$ and $Q(z) = 1 - z$.

4.1.1 The region of absolute stability

Since we do not want the error to accumulate as we step through the solution we require $|R(z)| \leq 1$ for our methods to remain stable. The values of $z = h\lambda$ for which this condition is met is known as the *region of absolute stability* and is formally defined by

$$S = \{z \in \mathbb{C} : |R(z)| \leq 1\}. \quad (4.5)$$

Of course $h \in \mathbb{R}$ but $\lambda \in \mathbb{C}$ so the values of h where a method will remain stable can be achieved by plotting $R(z)$ for $z \in \mathbb{C}$. The region where $R(z) \leq 1$ gives the values of h for which the method is stable can be determined by computing $|R(z)|$ and plotting the contour where $|R(z)| = 1$ which represents the boundary of S . For example, the stability regions for the Euler and implicit Euler methods defined using Eqs. (4.3) and (4.4) are plotted in Fig. 4.1. The shaded regions denote $z \in S$ so we can see that the Euler method is stable for $h \in [-2, 0]$ and for the implicit Euler method is stable for $h \in \mathbb{R} \setminus [0, 2]$.

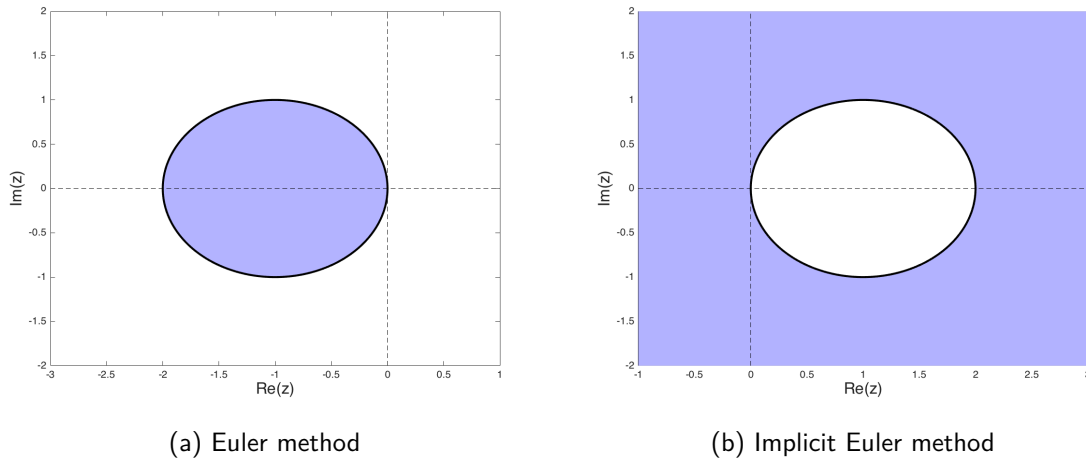


Figure 4.1: Plots of the region of absolute stability $S = \{z \in \mathbb{C} : |R(z)| \leq 1\}$ for the Euler and implicit Euler methods (shaded region denotes $z \in S$).

4.2 Stability of systems of ODEs

4.2.1 Linear systems of ODEs

Consider a system of N linear ODEs

$$\mathbf{y}' = A\mathbf{y}, \quad (4.6)$$

where A is an diagonalisable constant $N \times N$ matrix with eigenvectors \mathbf{r}_i such that $A\mathbf{r}_i = \lambda_i\mathbf{r}_i$ and λ_i is the eigenvalue of \mathbf{r}_i . If $R = (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_m)$ and D is a diagonal matrix of eigenvalues (Leveque, 2007), i.e.,

$$D = \begin{pmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \\ & & & \lambda_m \end{pmatrix} \quad (4.7)$$

then

$$A = RDR^{-1} \implies D = R^{-1}AR.$$

Introducing the transformation $\mathbf{u} = R^{-1}\mathbf{y}$

$$\begin{aligned}\mathbf{y}' &= A\mathbf{y} \\ R^{-1}\mathbf{y}' &= R^{-1}A\mathbf{y} \\ R^{-1}\mathbf{y}' &= R^{-1}ARR^{-1}\mathbf{y} \\ \mathbf{u}' &= D\mathbf{u}.\end{aligned}$$

So we now have a diagonal system of equations where the i^{th} equation is

$$u_i' = \lambda_i u_i.$$

Applying the Euler method to solve Eq. (4.6) results in

$$\mathbf{y}_{n+1} = \mathbf{y}_n + hA\mathbf{y}_n,$$

which can be transformed using $\mathbf{u}_n = R^{-1}\mathbf{y}_n$ into

$$\mathbf{u}_{n+1} = \mathbf{u}_n + hD\mathbf{u}_n,$$

so we now have N independent applications of the Euler method to solve N linear ODEs, e.g.,

$$u_{i,n+1} = u_{i,n} + h\lambda_i u_{i,n} = (1 + h\lambda_i)u_{i,n}.$$

The solution for $y_{i,n}$ can be obtained by reversing the transformation, i.e., $y_{i,n} = Ru_{i,n}$. The stability of a method used to solve the system of ODEs can be examined by considering the stability of each of the individual equation in the transformed system. The equation with the most stringent stability criteria is used to determine the step length for the solution method.

Example 4.2.1. Consider the equation for the motion of a pendulum

$$\ddot{\theta} + a\theta + b\dot{\theta} = 0$$

where a is related to the length of the pendulum and b is a dampening factor. Let $\theta_1 = \theta$ and $\theta_2 = \dot{\theta}$ then

$$\begin{aligned}\dot{\theta}_1 &= \theta_2, \\ \dot{\theta}_2 &= -a\theta_1 - b\theta_2\end{aligned}$$

which can be written as $\dot{\boldsymbol{\theta}} = A\boldsymbol{\theta}$ where $\boldsymbol{\theta} = (\theta_1, \theta_2)^T$ and

$$A = \begin{pmatrix} 0 & 1 \\ -a & -b \end{pmatrix}$$

The eigenvalues of A are $\lambda = \frac{1}{2}(-b \pm \sqrt{b^2 - 4a})$. The choice of the ODE solver and step length will depend on values of a (which relate to the length of the pendulum) and b (the damping factor). For example, if $b > 0$ then $\text{Re}(\lambda) < 0$ and the Euler method may be unstable of certain values of h whereas the implicit Euler method would be fine. \square

4.2.2 Non-linear systems of ODEs

For non-linear systems the linear stability analysis shown in Section 4.2.1 cannot be applied since the eigenvalues of A are not constant. Instead we seek to approximate the non-linear system using a linear system by assuming a slowly varying solution near the equilibrium points and a small step in the dependent variable.

Consider the following non-linear system of ODEs with two equations

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}),$$

where $\mathbf{y} = (y_1, y_2)^T$, $\mathbf{f}(\mathbf{y}) = (f_1(\mathbf{y}), f_2(\mathbf{y}))$ and an equilibrium point $\bar{\mathbf{y}} = (\bar{y}_1, \bar{y}_2)^T$. Using the first-order multivariate Taylor series expansion we have

$$\begin{aligned} f_1(\mathbf{y}) &= f_1(\bar{\mathbf{y}}) + \frac{\partial f_1(\bar{\mathbf{y}})}{\partial y_1}(y_1 - \bar{y}_1) + \frac{\partial f_1(\bar{\mathbf{y}})}{\partial y_2}(y_2 - \bar{y}_2), \\ f_2(\mathbf{y}) &= f_2(\bar{\mathbf{y}}) + \frac{\partial f_2(\bar{\mathbf{y}})}{\partial y_1}(y_1 - \bar{y}_1) + \frac{\partial f_2(\bar{\mathbf{y}})}{\partial y_2}(y_2 - \bar{y}_2). \end{aligned}$$

Since $\bar{\mathbf{y}}$ is an equilibrium point $f_1(\bar{\mathbf{y}}) = f_2(\bar{\mathbf{y}}) = 0$ and define $\mathbf{u} = (u_1, u_2)^T$ where $u_1 = y_1 - \bar{y}_1$ and $u_2 = y_2 - \bar{y}_2$ then

$$\begin{aligned} u_1' &= \frac{\partial f_1(\bar{\mathbf{y}})}{\partial y_1}u_1 + \frac{\partial f_1(\bar{\mathbf{y}})}{\partial y_2}u_2, \\ u_2' &= \frac{\partial f_2(\bar{\mathbf{y}})}{\partial y_1}u_1 + \frac{\partial f_2(\bar{\mathbf{y}})}{\partial y_2}u_2. \end{aligned}$$

This is a linear system of the form $\mathbf{u}' = J\mathbf{u}$ where the coefficient matrix is

$$J = \begin{pmatrix} \frac{\partial f_1(\bar{\mathbf{y}})}{\partial y_1} & \frac{\partial f_1(\bar{\mathbf{y}})}{\partial y_2} \\ \frac{\partial f_2(\bar{\mathbf{y}})}{\partial y_1} & \frac{\partial f_2(\bar{\mathbf{y}})}{\partial y_2} \end{pmatrix},$$

which is known as the *Jacobian matrix*. The stability analysis for a non-linear system is based upon the linearised system $\mathbf{u}' = J\mathbf{u}$ which consists of individual ODEs of the form $u_i' = \lambda_i u_i$ assuming that the numerical method behaves in a similar way for the non-linear case.

Definition 4.2.1 (Equilibrium points). The equilibrium points for the non-linear system of differential equations $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$ are the values of \mathbf{y} where $\mathbf{y}' = 0$ for all t .

Example 4.2.2. Determine the equilibrium points for the system

$$\begin{aligned} y_1' &= -4y_2 + 2y_1y_2 - 8, \\ y_2' &= 4y_2^2 - y_1^2. \end{aligned}$$

The equilibrium points are found by finding the values of y_1 and y_2 that satisfy

$$\begin{aligned} 0 &= -4y_2 + 2y_1y_2 - 8, \\ 0 &= 4y_2^2 - y_1^2. \end{aligned}$$

The first equation can be rearranged to $y_1 = 2y_2$ and substituting into the first equation gives

$$\begin{aligned} 0 &= -4y_2 + 4y_2^2 - 8 \\ \therefore 0 &= (y_2 - 2)(y_2 + 1), \end{aligned}$$

So the solution is $y_2 = 2, -1$ and $y_1 = 4, -2$ so this system has two equilibrium points at $(-2, -1)$ and $(4, 2)$. \square

4.3 Relative stability

We have seen that we use the test equation $y' = \lambda y$ to determine the region of absolute stability where the errors of a numerical method will not increase over the solution. However, if we consider that the solution of the test equation is $y(t) = e^{\lambda t}$ then if the solution decreases exponentially then we should also expect the errors to decrease exponentially. On the other hand if the solution increases exponentially then the errors can also increase exponentially just so long as they do not exceed the growth of the solution.

This concept is the rationale behind the *relative stability* where the stability function is compared to e^z such that $|R(z)| \leq |e^z|$ and the *region of relative stability* is defined as the set of all values of $z = h\lambda$ that satisfy this condition, e.g.,

$$A = \{z \in \mathbb{C} : |R(z)| \leq |e^z|\} = \left\{z \in \mathbb{C} : \left| \frac{R(z)}{e^z} \right| \leq 1\right\}. \quad (4.8)$$

4.3.1 Order stars

Consider the stability functions of the second and fourth-order explicit Runge-Kutta methods (second and fourth-order series expansions of e^z)

$$R(z) = 1 + z + \frac{z^2}{2} \quad (RK2), \quad (4.9)$$

$$R(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24} \quad (RK4), \quad (4.10)$$

The relative stability regions defined by Eq. (4.8) have been plotted in Fig. 4.2. This is done by defining a surface in the complex plane $R(z)$ using the real and imaginary parts of z and plotting the contour line where $|R(z)/e^z| = 1$. These plots were given the name *order stars* by Wanner, Hairer, and Nørsett (1978) in their seminal paper in which they showed that these regions are useful in determining the stability properties and order of methods.

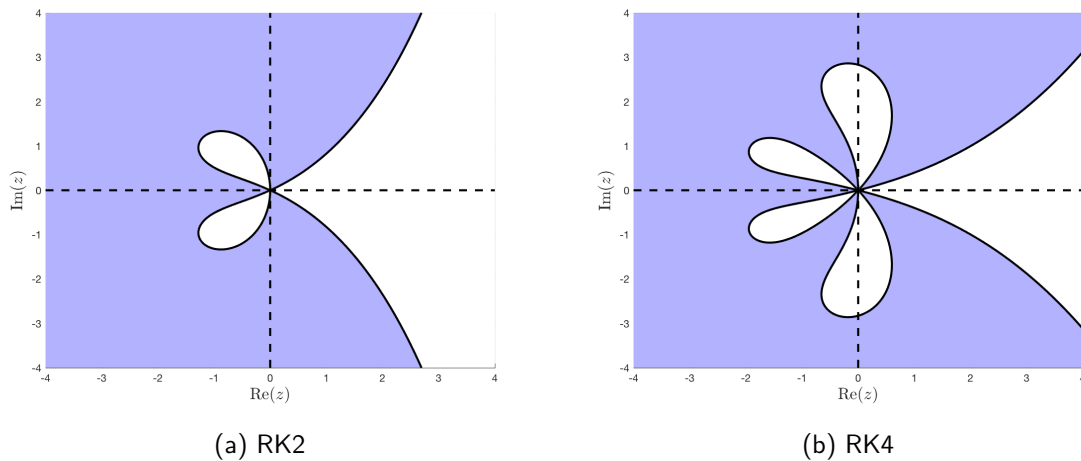


Figure 4.2: Relative stability plots or 'order stars' for the RK2 and RK4 methods.

The main noticeable feature of order stars is they consist of the following three regions

$$A_- = \{z \in \mathbb{C} : |R(z)| < |e^z|\},$$

$$A_0 = \{z \in \mathbb{C} : |R(z)| = |e^z|\},$$

$$A_+ = \{z \in \mathbb{C} : |R(z)| > |e^z|\}.$$

The shaded regions in Fig. 4.2 represent A_+ whereas the white regions represent A_- and the boundary between the two represented by the black line is A_0 .

The shaded and white regions converge at the origin and the behaviour near the origin is linked to the order of the method. If $R(z) = e^z + Cz^{p+1} + O(h^{p+2})$ then since $e^z \approx 1$ near to the origin

$$e^{-z}R(z) \approx 1 + Cz^{p+1}.$$

Tracing out a circle around the origin with a very small radius δ so that $z = \delta e^{2\pi i \theta}$ then $z^{p+1} = \delta^{p+1} e^{2(p+1)\pi i \theta}$ goes around a smaller circle centred at the origin $p+1$ times (Leveque, 2007). It follows that the order star of a p^{th} order method has $p+1$ shaded regions and $p+1$ white regions converging at the origin and hence the order star can be used to determine the order of the method (this is seen in Fig. 4.2).

4.3.2 A stability

A desirable property of numerical ODE solvers and one that is particularly useful for the solution of stiff systems is for a method to be *A-stable*. The region of absolute stability for A-stable methods covers the whole of the left-hand side of the complex plane \mathbb{C}^- . Due to this property, there is no limit to the size of h for which the method remains stable. However, this does not mean that any value of h should be used since too large a value may impact on accuracy requirements.

Recall that the stability function can be expressed as the rational polynomial

$$R(z) = \frac{P(z)}{Q(z)}.$$

A method is considered to be A-stable if the following criteria are met:

Criterion A. All poles are in \mathbb{C}^+ (the right-hand of the complex plane).

Criterion B. $E(y) = Q(iy)Q(-iy) - P(iy)P(-iy) \geq 0$ for all $y \in \mathbb{R}$.

It is quite easy to determine whether a method satisfies criterion A since we just need to determine the value of z for which $Q(z) = 0$. This also shows that no explicit method can be A-stable since for an explicit method $Q(z) = 1$. Using order stars means that the poles for the shaded region A_+ should be in the right-hand side of the complex plane \mathbb{C}^+ .

Criterion B on the other hand can result in some complicated algebra where the imaginary terms do not cancel out. (Wanner, Hairer, and Nørsett, 1978) showed that criterion B can be expressed as follows:

Criterion B'. $A_+ \cap i\mathbb{R} = \emptyset$

i.e., the shaded regions of the order star of an A-stable method do not cross the imaginary axis.

4.4 Padé approximants

We have seen that criterion A means that explicit Runge-Kutta methods cannot be A-stable so we can only consider implicit Runge-Kutta methods. The stability functions of implicit Runge-Kutta methods belong to the family of Padé approximants of the exponential function (Hairer and Wanner, 1999).

A Padé approximant of order L/A of the function $f(x)$ is denoted by

$$R_{L/M}(x) = \frac{P_L(x)}{Q_M(x)} = \frac{\sum_{i=0}^L a_i x^i}{\sum_{i=0}^M b_i x^i}, \quad (4.11)$$

where $P_L(x)$ and $Q_M(x)$ are polynomial functions of order L and M respectively and $Q_M(0) = 1$. We require the Padé coefficients a_i and b_i to satisfy

$$f(x) = \frac{P_L(x)}{Q_M(x)}. \quad (4.12)$$

up to order $L + M$, therefore

$$\begin{aligned} f(0) &= R(0), \\ f'(0) &= R'(0), \\ f''(0) &= R''(0), \\ &\vdots \\ f^{(L+M)}(0) &= R^{(L+M)}(0), \end{aligned}$$

so we can write

$$f(x) - R_{L/M}(x) = O(x^{L+M+1}). \quad (4.13)$$

In other words the Padé approximant of order L/M is equivalent to $f(x)$ up to order $L + M$. The Padé coefficients a_i and b_i are found by equating the series expansion of $f(x) = \sum_{i=0}^{L+M} c_i x^i$ with Eq. (4.11), i.e.,

$$c_0 + c_1 x + c_2 x^2 + \cdots + c_{L+M} x^{L+M} = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_L x^L}{1 + b_1 x + b_2 x^2 + \cdots + b_M x^M} + O(x^{L+M+1}).$$

Multiplying by the denominator and equating the coefficients for orders 0, 1, 2 etc. results in the following set of equations

$$\begin{aligned} a_0 &= c_0, \\ a_1 &= c_1 + c_0 b_1, \\ a_2 &= c_2 + c_1 b_1 + c_0 b_2, \\ a_3 &= c_3 + c_2 b_1 + c_1 b_2 + c_0 b_3, \\ &\vdots \end{aligned}$$

Since the values c_i are known we have a system of $L + M + 1$ equations and the same number of unknowns, a_i and b_i . The solution of this system gives the coefficients for the L/M Padé approximant of $f(x)$.

Example 4.4.1. Determine the Padé approximant $\exp_{1/2}(x)$.

Since $L = 1$ and $M = 2$ we compare the series expansion of $\exp(x)$ up to order $L + M = 3$.

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} = \frac{a_0 + a_1 x}{1 + b_1 x + b_2 x^2},$$

therefore

$$\begin{aligned} a_0 &= 1, \\ a_1 &= 1 + b_1, \\ 0 &= \frac{1}{2} + b_1 + b_2, \\ 0 &= \frac{1}{6} + \frac{1}{2}b_1 + b_2. \end{aligned}$$

The solution to this system is

$$a_0 = 1, \quad a_1 = \frac{1}{3}, \quad b_1 = -\frac{2}{3}, \quad b_2 = \frac{1}{6}.$$

and the Padé approximant is

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

□

4.4.1 Padé table for $\exp(z)$

The Padé approximants up to order 2/2 are given in Table 4.1. These Padé approximants are the stability functions of single step (i.e., Runge-Kutta) methods. For example, the approximants in the first row are the stability functions for the explicit Runge-Kutta methods as shown in Eqs. (4.9) and (4.10). Birkhoff and Varga (1965) showed that the entries on the main diagonal ($L = M$) are stability functions of A-stable methods. Ehle (1969) extended this work and showed that the two sub-diagonals also represent A-stable methods which was verified later by (Wanner, Hairer, and Nørsett, 1978).

Table 4.1: Padé approximants of $\exp(z)$ up to order 2/2.

$M \backslash L$	0	1	2
0	1	$1 + z$	$1 + z + \frac{1}{2}z^2$
1	$\frac{1}{1 - z}$	$\frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}$	$\frac{1 + \frac{2}{3}z + \frac{1}{6}z^2}{1 - \frac{1}{3}z}$
2	$\frac{1}{1 - z + \frac{1}{2}z^2}$	$\frac{1 + \frac{1}{3}z}{1 - \frac{2}{3}z + \frac{1}{6}z^2}$	$\frac{1 + \frac{1}{2}z + \frac{1}{12}z^2}{1 - \frac{1}{2}z + \frac{1}{12}z^2}$

To summarise, a L/M Padé approximant of $\exp(z)$ represents the stability function of an A-stable method if

$$0 \leq M - L \leq 2. \quad (4.14)$$

The order stars for $R(z)$ from the $L = 2$ column of the Padé table have been plotted in Fig. 4.3. Here it can be clearly seen that $\exp_{2/2}(z)$ and $\exp_{2/3}(z)$ represent A-stable methods whereas $\exp_{2/1}(z)$ and $\exp_{2/5}(z)$ do not thus verifying Eq. (4.14).

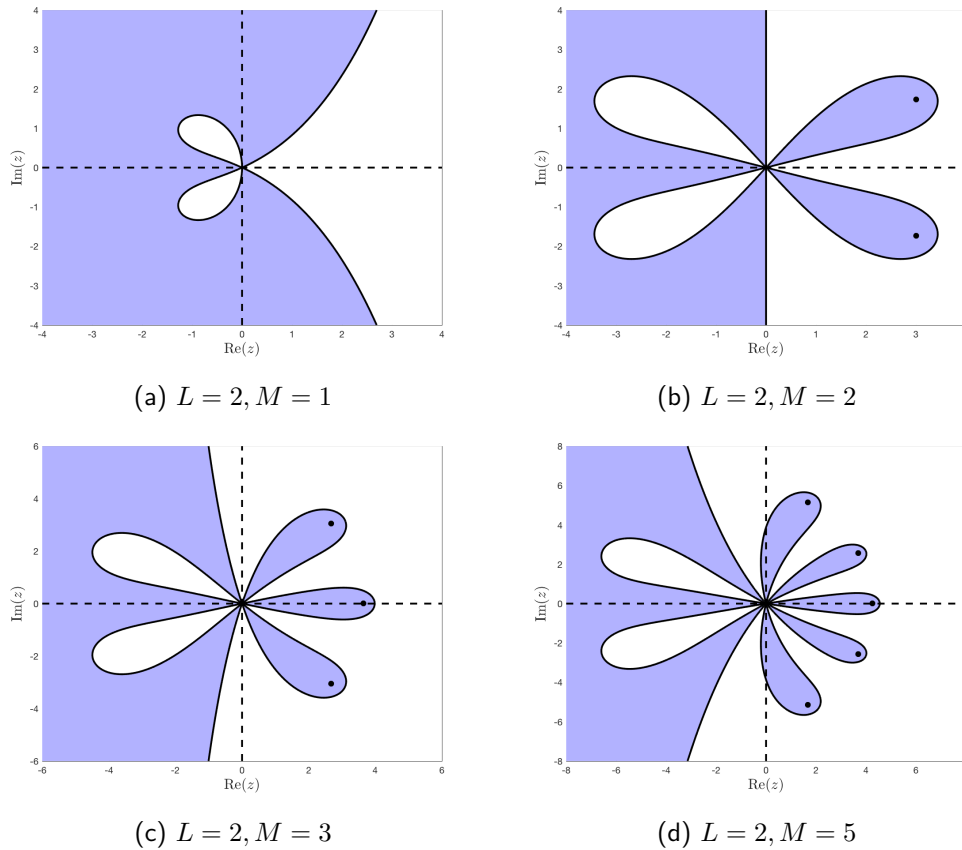


Figure 4.3: The order stars for the Padé approximants show that Runge-Kutta methods are only A-stable for $M - L \in [0, 2]$.

4.5 Stiffness

An important consideration in choosing an ODE solver is the issue of *stiffness*. The precise definition is not fully defined but (Lambert, 1991) provides a good verbal definition

If a numerical method is forced to use, in a certain interval of integration, a step length which is excessively small in relation to the smoothness of the exact solution in that interval, then the problem is said to be stiff in that interval.

In other words a stiff system is where the choice of step length is made to satisfy the stability constraints as opposed to accuracy constraints.

For example consider the following linear equation

$$y'' + 1001y' + 1000y = 0$$

Rewriting this second-order ODE as a system of two first-order ODEs gives

$$y_1' = y_2, \tag{4.15a}$$

$$y_2' = -1001y_2 - 1000y_1, \tag{4.15b}$$

which is in the form $\mathbf{y}' = A\mathbf{y}$ where

$$A = \begin{pmatrix} 0 & 1 \\ -1000 & -1001 \end{pmatrix}.$$

The eigenvalues of A can be used to diagonalise the system Eqs. (4.15a) and (4.15b) so we have equations of the form $u'_i = \lambda u_i$ which are used to determine the range of values for which the step length will remain stable for a particular method. Consider the region of absolute stability for the Euler method

$$|1 + h\lambda| \leq 1,$$

taking the real part of λ gives

$$\begin{aligned} -1 &\leq 1 + h\lambda \leq 1 \\ -2 &\leq h\lambda \leq 0 \\ \therefore h &\leq -\frac{2}{\lambda} \end{aligned}$$

The eigenvalues of A in this case are $\lambda_1 = -1$ and $\lambda_2 = -1000$ so we have $h \leq 2$ for λ_1 and $h \leq 0.002$ for λ_2 .

This variability in the values of the eigenvalues (and therefore the step lengths) gives rise to the idea of *stiffness ratio* which is defined as

$$R = \frac{\max_i |\operatorname{Re}(\lambda_i)|}{\min_i |\operatorname{Re}(\lambda_i)|}, \quad (4.16)$$

such that when R is large the system is considered stiff.

Example 4.5.1. The well known van der Pol oscillator is described by the following second-order ODE

$$y'' - \mu(1 - y^2)y' + y = 0.$$

Writing this as a system of two first-order ODEs gives

$$y'_1 = y_2, \quad (4.17a)$$

$$y'_2 = \mu(1 - y_1^2)y_2 - y_1. \quad (4.17b)$$

The stability of a non-linear system can be determined by approximating the system using a linear system close to an equilibrium point. Equations (4.17a) and (4.17b) have an equilibrium point at $(0,0)$ and the Jacobian matrix is

$$J = \begin{pmatrix} 0 & 1 \\ -2\mu y_1 y_2 - 1 & \mu(1 - y_1^2) \end{pmatrix},$$

so

$$J_{(0,0)} = \begin{pmatrix} 0 & 1 \\ -1 & \mu \end{pmatrix},$$

which has eigenvalues $\lambda = (\mu \pm \sqrt{\mu^2 - 4})/2$. The value of μ will determine the values of the eigenvalues and therefore the stiffness of the system. For example, when $\mu = 2$ the stiffness ratio is

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

whereas when $\mu = 1000$

$$R = \frac{\frac{1}{2}(1000 + \sqrt{999996})}{\frac{1}{2}(1000 - \sqrt{999996})} = \frac{999.999}{0.001} = 999999.$$

□

4.6 Stiff solvers

Explicit methods are unsuitable for solving stiff systems due to the restriction placed on the value of the step length due to stability. Implicit methods however have a much less restrictive stability region so can be used to solve stiff systems.

For example, consider the solution to the van der Pol equations from Example 4.5.1 for $\mu = 100$ (giving $R = 9999$) over the domain $t \in [0, 500]$ with initial values $y_0 = (2, 0)$. The MATLAB solvers `ode45` (non-stiff) and `ode15s` (stiff) solvers have been used to compute the solution to this problem which is plotted in Fig. 4.4. The behaviour of the solution shows extremely rapid change in the value of y which is characteristic of a stiff system.

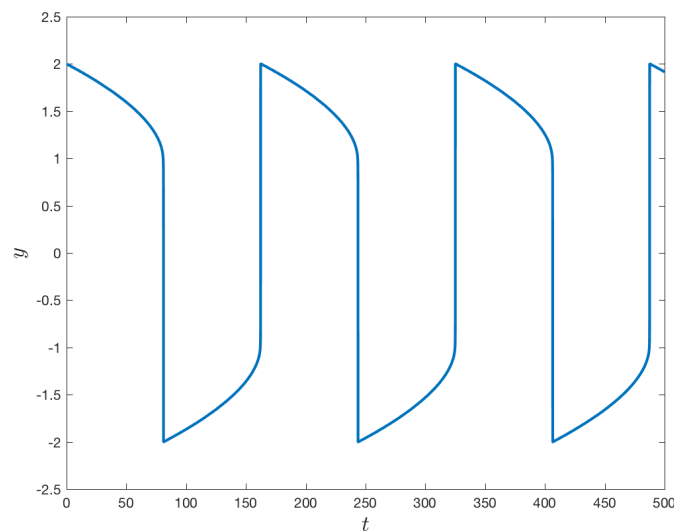


Figure 4.4: The solution to the van der Pol equations over the domain $t \in [0, 500]$ and initial conditions $y_0 = (2, 0)$.

The statistics for the two solvers were outputted to the command window and given below. The non-stiff `ode45` solver took 2.88 seconds to compute the solution whereas the stiff `ode15s` solver took just 0.22 seconds. The statistics from the `ode45` solver shows that there were a large number of failed steps that implies the step length was often too large to maintain the required accuracy.

```
ode45
-----
27375 successful steps
1776 failed attempts
174907 function evaluations
Elapsed time is 2.880050 seconds.

ode15s
-----
885 successful steps
306 failed attempts
2716 function evaluations
54 partial derivatives
394 LU decompositions
2553 solutions of linear systems
Elapsed time is 0.222540 seconds.
```

4.7 Tutorial exercises

1. A pendulum can be described by the ODE

$$\ddot{\theta} + \frac{g}{\ell} \sin(\theta) = 0,$$

where θ is the angular displacement from the vertical, g is the acceleration due to gravity and ℓ is the length of the chord.

Given a pendulum of length $\ell = 1$ and assuming $g = 9.81\text{ms}^{-2}$

- (a) Determine the equilibrium points for the system.
 - (b) For each equilibrium point, calculate the Jacobian matrix.
 - (c) Calculate the eigenvalues for this system.
 - (d) Determine the range of values of h for which the Euler method is stable for this problem.
2. Add another row to Table 4.1 by finding the Padé approximants $\exp_{L/3}(z)$ for $L = 0, 1, 2$.
 3. Plot the order stars for your Padé approximants found in question 3 and state whether they represent an A-stable method or not.
 4. The Padé approximants in the lower diagonal of the Padé table (Table 4.1), i.e., $L = M - 1$, represent the stability functions of the Radau IIA methods (Hairer and Wanner, 1999). Plot the order star for a 9th order Radau IIA method and thus determine whether it is A-stable or not.

The solutions to these exercises can be found on page 68.

Chapter 5

Applications of ODEs: the N-body problem

The motion of the planets and other celestial objects has fascinated astronomers and mathematicians for millennia. In ancient times primitive societies believed that the presence of the stars in the night sky were representations of the Gods or supernatural beings. The ancient astronomers observed that some objects in the night sky travelled faster than others and were given the name *astē planētēs* in ancient Greek meaning 'wandering star'. This is from where the modern English word 'planets' derives.

It was Nicolaus Copernicus who first proposed the *heliocentric* model in his book *On the Revolutions of the Heavenly Bodies* that placed the Sun, and not the Earth, at the centre of the solar system. This was met with much resistance by the Holy Roman Church since it directly contradicts statements in the Bible and was treated as heresy.

The motion of two heavenly bodies that became known as the two-body problem was analysed by Johannes Kepler in 1609 and was later solved by Isaac Newton in 1687 (Wolfram, 2002). The three-body problem became a central topic in mathematical physics up until the early 1900s. It was Henri Poincaré who showed that the three-body problem could not be solved using algebra and integrals and in the process discovered the chaotic nature of the solutions that was an early foray into what would later become known as chaos theory.

With the advent of the computer, approximations of the three-body, and therefore the N-body problem, have been made possible using computational Ordinary Differential Equations (ODEs) solvers. This work has been fundamental in the study of the universe.



(a) Nicolaus Copernicus (1473 – 1543) (b) Johannes Kepler (1571 – 1630) (c) Isaac Newton (1643 – 1727) (d) Henri Poincaré (1854 – 1912)

Figure 5.1: The main protagonists of the study of the motions of celestial bodies and the N-body problem

5.1 Gravitation

The gravitational force F acting on an object i caused by another object j is related to the gravitational constant G , the mass of the two objects m_i and m_j and the distance between the two bodies r_{ij} by the *inverse square law*

$$F = G \frac{m_i m_j}{r_{ij}^2}. \quad (5.1)$$

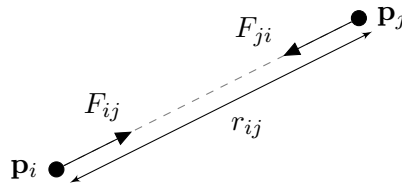


Figure 5.2: Two bodies are attracted towards each other by the force of gravitation.

Consider Fig. 5.2, the gravitational force acting on body i will be acting in the direction of the vector pointing towards body j , F_{ij} . Let $\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i$ be the vector pointing from i to j and $r_{ij} = \|\mathbf{p}_{ij}\|$ then Eq. (5.1) can be written in vector form as

$$F_{ij} = G \frac{m_i m_j}{r_{ij}^3} \mathbf{p}_{ij}. \quad (5.2)$$

If there are more than two bodies in the system, the gravitational force acting on body i will be the sum of Eq. (5.2) for all other bodies in the system, i.e.,

$$\mathbf{F}_i = \sum_{j=1, j \neq i}^N G \frac{m_i m_j}{r_{ij}^3} \mathbf{p}_{ij}. \quad (5.3)$$

Newton's second law of motion is

$$\mathbf{F} = m\mathbf{a}$$

so we can rewrite Eq. (5.3) as

$$m_i \mathbf{a}_i = \sum_{j=1, j \neq i}^N G \frac{m_i m_j}{r_{ij}^3} \mathbf{p}_{ij}$$

$$\mathbf{a}_i = \sum_{j=1, j \neq i}^N G \frac{m_j}{r_{ij}^3} \mathbf{p}_{ij}$$

Since acceleration is the second derivative of space with respect to time, the motion of a body is governed by the second-order ODE

$$\ddot{\mathbf{p}}_i = \sum_{j=1, j \neq i}^N G \frac{m_j}{r_{ij}^3} \mathbf{p}_{ij}. \quad (5.4)$$

5.2 The two-body problem

The simplest N-body problem is two-body problem. Consider Fig. 5.3 where two bodies at positions \mathbf{p}_1 and \mathbf{p}_2 with mass m_1 and m_2 are travelling with velocities \mathbf{v}_1 and \mathbf{v}_2 .

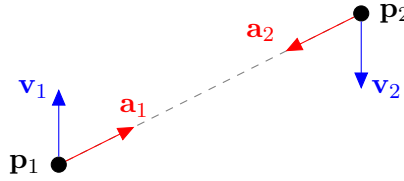


Figure 5.3: The two-body problem

Since there are only two bodies in this system the gravitational force acting on each body is only influenced by the position and mass of the other body. Using Eq. (5.4) the motion of the two bodies is defined by the system

$$\ddot{\mathbf{p}}_1 = G \frac{m_2}{r_{12}^3} \mathbf{p}_{12},$$

$$\ddot{\mathbf{p}}_2 = -G \frac{m_1}{r_{12}^3} \mathbf{p}_{12}$$

Note that $\mathbf{p}_{21} = -\mathbf{p}_{12}$ is used in the second equation to avoid repeating a similar calculation. Given that velocity is the derivative of space with respect to time then $\mathbf{v} = \dot{\mathbf{p}}$ and we can write this as a system of four first-order ODEs

$$\dot{\mathbf{p}}_1 = \mathbf{v}_1,$$

$$\dot{\mathbf{p}}_2 = \mathbf{v}_2,$$

$$\dot{\mathbf{v}}_1 = G \frac{m_2}{r_{12}^3} \mathbf{p}_{12},$$

$$\dot{\mathbf{v}}_2 = -G \frac{m_1}{r_{12}^3} \mathbf{p}_{12}.$$

If $\mathbf{p} \in \mathbb{R}^2$ then $\mathbf{p} = (x, y)$ and if $\mathbf{v} = (u, v)$ then the full system of equations are

$$\dot{x}_1 = u_1,$$

$$\begin{aligned}
\dot{y}_1 &= v_1, \\
\dot{x}_2 &= u_2, \\
\dot{y}_2 &= v_2, \\
\dot{u}_1 &= G \frac{m_2}{r_{12}^3} (x_2 - x_1), \\
\dot{v}_1 &= G \frac{m_2}{r_{12}^3} (y_2 - y_1), \\
\dot{u}_2 &= -G \frac{m_1}{r_{12}^3} (x_2 - x_1), \\
\dot{v}_2 &= -G \frac{m_1}{r_{12}^3} (y_2 - y_1).
\end{aligned}$$

So for two bodies in \mathbb{R}^2 we have a system of eight first-order ODEs.

5.2.1 Two-body problem solution

To demonstrate the solution to the two-body problem consider the example of two bodies of mass $m_1 = m_2 = 1$ are located at positions $\mathbf{p}_1 = (-1, 0)$ and $\mathbf{p}_2 = (1, 0)$ with velocities $\mathbf{v}_1 = (0, 1)$ and $\mathbf{v}_2 = (0, -1)$. The gravitational constant for this problem is $G = 10$. The two-body system can be written as an IVP in the form $\dot{\mathbf{y}} = f(t, \mathbf{y})$ where

$$\mathbf{y} = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ u_1 \\ v_1 \\ u_2 \\ v_2 \end{pmatrix}, \quad f(t, \mathbf{y}) = \begin{pmatrix} y_5 \\ y_6 \\ y_7 \\ y_8 \\ Gm_2(y_3 - y_1)/r_{12}^3 \\ Gm_2(y_4 - y_2)/r_{12}^3 \\ -Gm_1(y_3 - y_1)/r_{12}^3 \\ -Gm_1(y_4 - y_2)/r_{12}^3 \end{pmatrix}, \quad \mathbf{y}(0) = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ -1 \end{pmatrix}.$$

The MATLAB program shown in listing 5.1 was written to solve the two-body problem defined here. The program uses the `ode113` solver to solve the system and no stability problems were encountered. A plot of the solution at $t = 5$ is shown in Fig. 5.4.

```

% Clear workspaces
clear
clc

% Define parameters
G = 10; % Gravitational constant
fps = 30; % Frames per second in animation

% Define body positions, velocities and mass
p1 = [ -1 , 0 ];
p2 = [ 1 , 0 ];
v1 = [ 0 , 1 ];
v2 = [ 0 , -1 ];
mass = [ 1, 1 ];

% Define IVP parameters
tspan = linspace(0, 2, 2*30);
y0 = [ p1 , p2 , v1 , v2 ];

% Solve ODE

```

```

[t, y] = ode113(@t, y)two_body_ode(t, y, G, mass), tspan, y0);

% Plot solution
for i = 1 : length(t)

    clf
    hold on
    plot(y(i, 1), y(i, 2), "bo", "markerfacecolor", "b")      % bodies
    plot(y(i, 3), y(i, 4), "ro", "markerfacecolor", "r")
    plot(y(1:i, 1), y(1:i, 2), "b-", "linewidth", 2)         % trajectories
    plot(y(1:i, 3), y(1:i, 4), "r-", "linewidth", 2)
    hold off

    axis([-1.5, 1.5, -1, 1])
    box on
    title(sprintf("$t$ = %1.2f", t(i)), "fontsize", 16, "interpreter", "latex")
    xlabel("$x$", "fontsize", 16, "interpreter", "latex")
    ylabel("$y$", "fontsize", 16, "interpreter", "latex")
    shg
    pause(1 / fps)
end

% -----
function dy = two_body_ode(~, y, G, m)

% This function defines the ODE function for the two body problem in 2D.

% Calculate differences in co-ordinates and distance between bodies
x12 = y(3) - y(1);
y12 = y(4) - y(2);
r12 = 1 / sqrt(x12^2 + y12^2)^3;

% Calculate dy vector
dy = zeros(size(y));
dy(1:4) = y(5:end);
dy(5) = G * m(2) * x12 * r12;
dy(6) = G * m(2) * y12 * r12;
dy(7) = - G * m(1) * x12 * r12;
dy(8) = - G * m(1) * y12 * r12;

end

```

Listing 5.1: MATLAB program to define and solve the two-body problem.

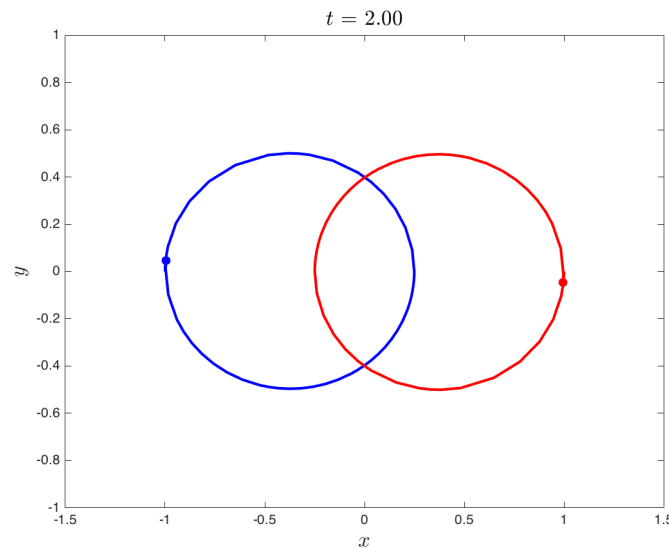


Figure 5.4: Plot of the solution to the two-body problem.

The evolution of the solution shows that initially the two bodies are travelling in opposite vertical directions. The gravitational pull of the other body affects the trajectory and the two bodies begin to travel towards each other.

5.3 The three-body problem

The introduction of a third body into the system requires that the gravitational forces acting on a single body from the other two bodies are combined to calculate the acceleration. Consider the diagram in Fig. 5.5 where three bodies are at positions \mathbf{p}_1 , \mathbf{p}_2 and \mathbf{p}_3 are travelling with velocities \mathbf{v}_1 , \mathbf{v}_2 and \mathbf{v}_3 . The forces acting on body 1, F_{12} and F_{13} , are calculated using the respective masses and distances of bodies 2 and 3 and summed to give \mathbf{a}_1 .

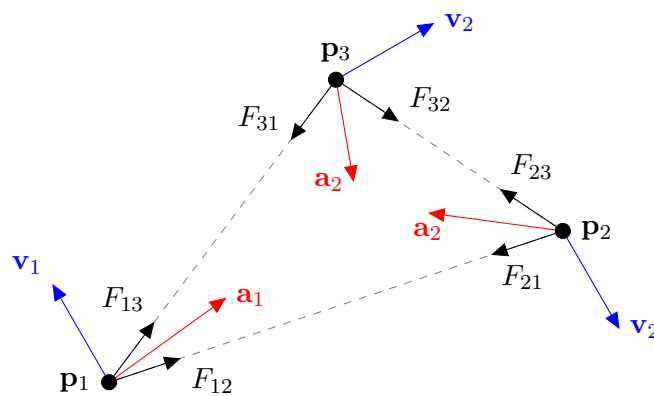


Figure 5.5: The three-body problem

Using Eq. (5.4) for the three body system we have

$$\begin{aligned}\dot{\mathbf{p}}_1 &= \mathbf{v}_1, \\ \dot{\mathbf{p}}_2 &= \mathbf{v}_2,\end{aligned}$$

$$\begin{aligned}
\dot{\mathbf{p}}_3 &= \mathbf{v}_3, \\
\dot{\mathbf{v}}_1 &= G \frac{m_2}{r_{12}^3} \mathbf{p}_{12} + G \frac{m_3}{r_{13}^3} \mathbf{p}_{13}, \\
\dot{\mathbf{v}}_2 &= -G \frac{m_1}{r_{12}^3} \mathbf{p}_{12} + G \frac{m_3}{r_{23}^3} \mathbf{p}_{23}, \\
\dot{\mathbf{v}}_3 &= -G \frac{m_1}{r_{13}^3} \mathbf{p}_{13} - G \frac{m_2}{r_{23}^3} \mathbf{p}_{23},
\end{aligned}$$

where $r_{ij} = \|\mathbf{p}_{ij}\|$. If $\mathbf{p} \in \mathbb{R}^2$ we now have the system of 12 ODEs

$$\begin{aligned}
\dot{x}_1 &= u_1, \\
\dot{y}_1 &= v_1, \\
\dot{x}_2 &= u_2, \\
\dot{y}_2 &= v_2, \\
\dot{x}_3 &= u_3, \\
\dot{y}_3 &= v_3, \\
\dot{u}_1 &= G \frac{m_2}{r_{12}^3} (x_2 - x_1) + G \frac{m_3}{r_{13}^3} (x_3 - x_1), \\
\dot{v}_1 &= G \frac{m_2}{r_{12}^3} (y_2 - y_1) + G \frac{m_3}{r_{13}^3} (y_3 - y_1), \\
\dot{u}_2 &= -G \frac{m_1}{r_{12}^3} (x_2 - x_1) + G \frac{m_3}{r_{23}^3} (x_3 - x_2), \\
\dot{v}_2 &= -G \frac{m_1}{r_{12}^3} (y_2 - y_1) + G \frac{m_3}{r_{23}^3} (y_3 - y_2), \\
\dot{u}_3 &= -G \frac{m_1}{r_{13}^3} (x_3 - x_1) - G \frac{m_2}{r_{23}^3} (x_3 - x_2), \\
\dot{v}_3 &= -G \frac{m_1}{r_{13}^3} (y_3 - y_1) - G \frac{m_2}{r_{23}^3} (y_3 - y_2),
\end{aligned}$$

5.3.1 The three-body solution

To demonstrate the solution of the three body problem consider a system with three points of mass $m_1 = m_2 = m_3 = 1$ positioned at the three vertices of an equilateral triangle centred at the origin with side length 1 such that $\mathbf{p}_1(0) = (0, \sqrt{3}/3)$, $\mathbf{p}_2(0) = (-1/2, -\sqrt{3}/6)$ and $\mathbf{p}_3(0) = (1/2, -\sqrt{3}/6)$ (Fig. 5.6). The velocities of the three bodies are zero with the exception on $\mathbf{v}_1(0)$ which is a given a small positive velocity in the x direction and the gravitational constant was set as $G = 1$. Plots of the solution trajectories for slightly varying values of \mathbf{v}_1 can be seen in Fig. 5.7 which shows that small changes in the initial conditions results in vastly different solutions, an observation first made by Poincaré.

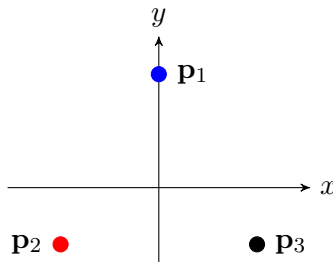
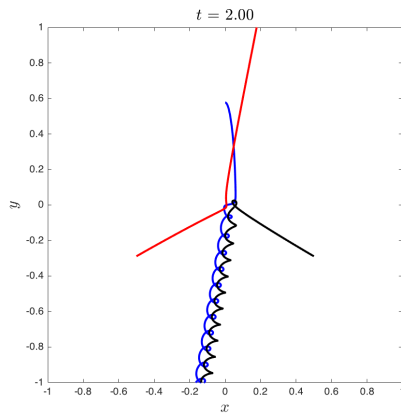
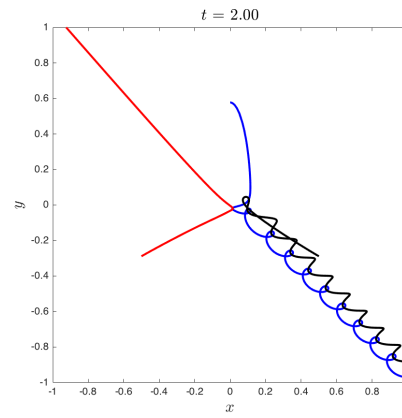
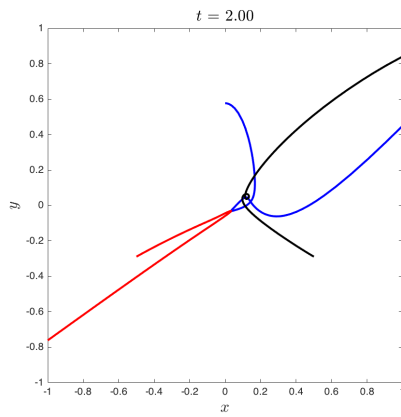
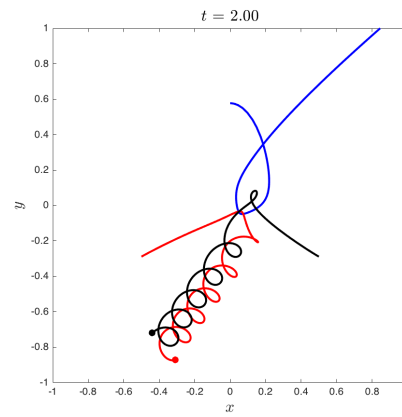
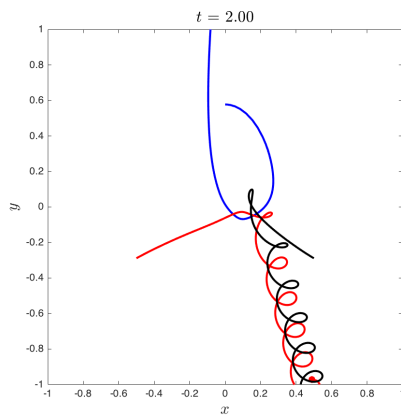
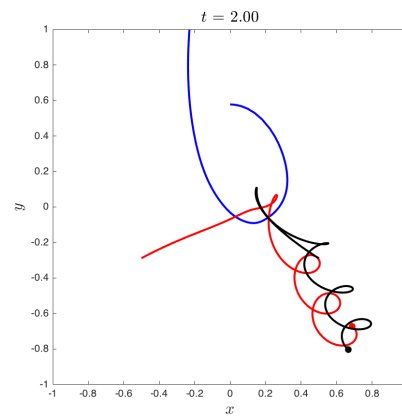


Figure 5.6: Initial positions for the three-body problem.

(a) $\mathbf{v}_3 = (0.1, 0, 0)$ (b) $\mathbf{v}_3 = (0.2, 0, 0)$ (c) $\mathbf{v}_3 = (0.3, 0, 0)$ (d) $\mathbf{v}_3 = (0.4, 0, 0)$ (e) $\mathbf{v}_3 = (0.5, 0, 0)$ (f) $\mathbf{v}_3 = (0.6, 0, 0)$ Figure 5.7: Solution trajectories for the three body problem with slightly different velocities for \mathbf{v}_3 .

```

function dy = N_body_ode(~, y, G, m)

% Defines the ODE function for the N body problem in 3 dimensions

% Calculate number of bodies in the system
N = length(y) / 6;

% Define first half of dy vector
dy = zeros(size(y));
dy(1 : 3 * N) = y(3 * N + 1 : end);

% Loop through the bodies
for i = 1 : N - 1
    for j = i + 1 : N

        % Calculate distance between body i and body j
        xij = y(3 * j - 2) - y(3 * i - 2);
        yij = y(3 * j - 1) - y(3 * i - 1);
        zij = y(3 * j) - y(3 * i);
        r = 1 / sqrt(xij^2 + yij^2 + zij^2) ^ 3;

        % Calculate sum of forces for body i and j
        dy(3 * N + 3 * i - 2) = dy(3 * N + 3 * i - 2) + G * m(j) * xij * r;
        dy(3 * N + 3 * i - 1) = dy(3 * N + 3 * i - 1) + G * m(j) * yij * r;
        dy(3 * N + 3 * i) = dy(3 * N + 3 * i) + G * m(j) * zij * r;
        dy(3 * N + 3 * j - 2) = dy(3 * N + 3 * j - 2) - G * m(i) * xij * r;
        dy(3 * N + 3 * j - 1) = dy(3 * N + 3 * j - 1) - G * m(i) * yij * r;
        dy(3 * N + 3 * j) = dy(3 * N + 3 * j) - G * m(i) * zij * r;

    end
end
end

```

Listing 5.2: MATLAB function that defines the ODE function for the N-body problem in \mathbb{R}^3 .

5.4 The N-body problem formulation

The formulation for N bodies follows naturally from the three-body problem. Using Eq. (5.4) for N bodies and $\mathbf{p} \in \mathbb{R}^3$ where $\mathbf{p} = (x, y, z)$ and $\mathbf{v} = (u, v, w)$ we will have a system of $6N$ equations, i.e.,

$$\begin{aligned}
 \dot{x}_i &= u_i, \\
 \dot{y}_i &= v_i, \\
 \dot{z}_i &= w_i, \\
 \dot{u}_i &= \sum_{j=1, j \neq i} G \frac{m_j}{r_{ij}^3} (x_j - x_i), \\
 \dot{v}_i &= \sum_{j=1, j \neq i} G \frac{m_j}{r_{ij}^3} (y_j - y_i), \\
 \dot{w}_i &= \sum_{j=1, j \neq i} G \frac{m_j}{r_{ij}^3} (z_j - z_i).
 \end{aligned}$$

where $i = 1, \dots, N$. The MATLAB function given in listing 5.2 defines the ODE function for N bodies in \mathbb{R}^3 . Note that this function uses the fact that $\mathbf{p}_{ij} = -\mathbf{p}_{ji}$ to reduce the number of calculations.

5.5 Modelling the solar system

The most common application of the N-body problem is the modelling of the motions of bodies in a solar system. This allows us to calculate the trajectories of the planets and was necessary for NASA's Apollo missions to the moon and exploration of Mars and the ESA's Rosetta mission to Comet 67P/Churyumov-Gerasimenko.

Ephemeris giving best estimates of the positions, velocities and physical parameters for solar system bodies is available from NASA's JPL Horizons system (Giorgini, 2016) and have been included in Appendix B (correct for 00:00 on the 1st January 2017). The value of the gravitational constant is approximately $G = 6.67404 \times 10^{-11} \text{m}^3 \text{kg}^{-1} \text{s}^{-2}$.

5.5.1 Sun, Earth and Moon

The data for the Sun, Earth and Moon was used with the code for the three-body problem to model the motion of these bodies. A plot of the solution after one orbit of the Earth (i.e., 1 year) is shown in Fig. 5.8. The position of the Moon has been artificially moved further away from the Earth so that it can be distinguished from the Earth in the solution plot. It can be clearly seen that the very large mass of the Sun relative to the masses of the Earth and Moon means that Sun's position remains virtually stationary. The Moon orbits the Earth since the force exerted by the gravitational pull of the Earth is much larger than that of the Sun due to the inverse square law.

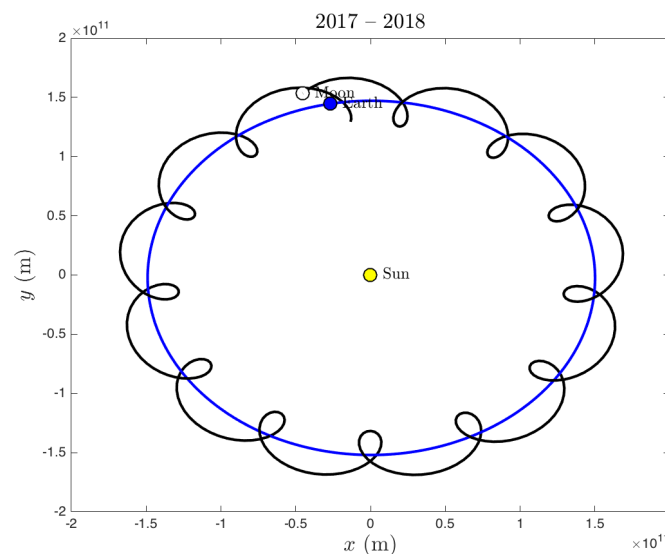


Figure 5.8: The solution of the three-body problem for modelling the orbit of the Earth and Moon around the sun.

5.5.2 Sun, the eight planets and Pluto

The motion of the Sun, the eight planets of the solar system and Pluto were modelled using an 10-body formulation and can be seen in Fig. 5.9. Due to the distances involved it is difficult to see the trajectories of the closest four planets of the solar system so they have been omitted from this plot. The plot shows the solution after 250 years (at the start of the year 2267). The plot in Fig. 5.10 is the same plot as Fig. 5.9 but aligned to show that the plane of Pluto's orbit is different from that of the planets of the solar system.

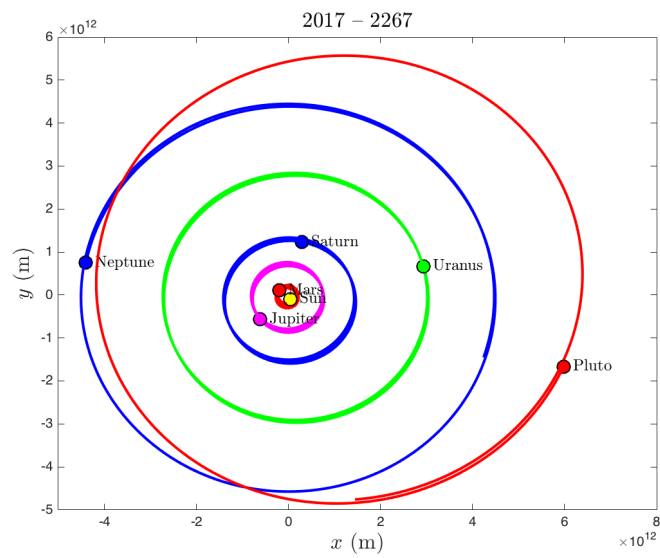


Figure 5.9: The solution of the 10-body problem for modelling the orbits of the planets of the solar system (and Pluto).

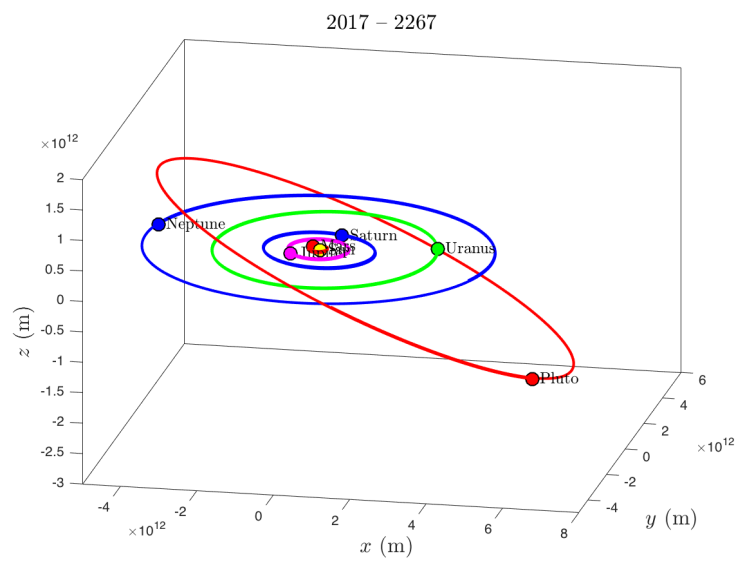


Figure 5.10: A plot of the the orbits of the planets and Pluto aligned to show the plane of Pluto's orbit.

5.6 Tutorial Exercises

1. Solve the two-body problem using the [ode45](#) solver for initial positions $\mathbf{p}_1 = (0, 0)$, $\mathbf{p}_2 = (1, 0)$, the velocity of body 1 $\mathbf{v}_1(0) = (0, 0)$, the mass of body 2 $m_2 = 1$, $G = 1$ and the following values for velocity of body 2 and mass of body 1.

(a) $\mathbf{v}_2(0) = (-5, -5)$, $m_1 = 100$;

(b) $\mathbf{v}_2(0) = (0, -1)$, $m_1 = 10$.

Produce a plot showing the trajectories of the two bodies after $t = 5\text{s}$ has elapsed.

2. Use the MATLAB function `N_body_ode.m` found on page [59](#) (also available on the Moodle area for this unit) to define the three-body problems shown in Section [5.3.1](#) and reproduce the plots shown in Fig. [5.7](#). Use the [ode23s](#) solver to solve the ODE.
3. Solve the three body problem with $G = 1$, $m_1 = m_2 = m_3 = 1$ and the following initial positions and velocities: $\mathbf{p}_1 = (-0.3668, 0.0074)$, $\mathbf{p}_2 = (0.4888, 0.0064)$, $\mathbf{p}_3 = (-0.1193, -0.0135)$, $\mathbf{v}_1 = (0.1229, 0.7474)$, $\mathbf{v}_2 = (-0.0193, 1.3692)$, $\mathbf{v}_3 = (-0.1036, -2.1167)$.
4. Download the file `solar.mat` from the Moodle area for this unit that contains the ephemeris given in Appendix [B](#). Load the data using the command '`load solar`' and use the [ode113](#) solver to model the orbits of the first five planets in the solar system (Mercury to Mars) around the Sun for the five year period following 00:00 01/01/2017. Note that the ephemeris gives the positions of the bodies in Astronomical Units ($1\text{AU} = 149597870700\text{m}$), velocities in AU/day, mass in kg and the value of the gravitational constant is approximately $G = 6.67404 \times 10^{-11} \text{m}^3 \text{kg}^{-1} \text{s}^{-2}$.
5. Extend your model from question 4 to model the orbits of all of the planets in the solar system (plus Pluto) for the period 00:00 01/01/2017 to 00:00 01/01/2117. Produce a plot of the motion of the Sun over this period. How can this information be used to detect other solar systems?

The solutions to these exercises can be found on page [76](#).

References

- Birkhoff, G. and R.S. Varga (1965). "Discretization errors for well set Cauchy problems". In: *International Journal of Mathematical Physics* 44, pp. 1–23.
- Butcher, J.C. (1987). *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience publication. J. Wiley. ISBN: 9780471910466.
- Butcher, J.C. (2009). "Trees and numerical methods for ordinary differential equations". In: *Numer. Algor.* 53 (2), pp. 153–170.
- Cash, J.R. and A.H. Karp (1990). "A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides". In: *ACM Transactions on Mathematical Software* 16, pp. 201–222.
- Dormand, J.R. and P.J. Prince (1980). "A family of embedded Runge-Kutta formulae". In: *Journal of Computational and Applied Mathematics* 6.1, pp. 19–26.
- Ehle, B.L. (1969). *On Padé approximations to the exponential function and A stable methods for the numerical solution of initial value problems*, Research Report CSRR 2010. Tech. rep. Ontario, Canada: Department of AACS, University of Waterloo.
- Fehlberg, E. (July 1969). *Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems*. Tech. rep. NASA.
- Giorgini, J.D. (2016). *HORIZONS Web-Interface*. URL: <http://ssd.jpl.nasa.gov/horizons.cgi> (visited on 11/17/2016).
- Hairer, E. and G. Wanner (1999). "Stiff differential equations solved by Radau methods". In: *Journal of Computational and Applied Mathematics* 111, pp. 93–111.
- Lambert, J. D. (1991). *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*. New York, NY, USA: John Wiley & Sons, Inc. ISBN: 0-471-92990-5.
- Leveque, R.J. (2007). *Finite Difference Methods for Ordinary Differential Equations*. Society for Industrial and Applied Mathematics (SIAM).
- Press, William H. et al. (1993). *Numerical Recipes in FORTRAN; The Art of Scientific Computing*. 2nd. New York, NY, USA: Cambridge University Press. ISBN: 0521437164.

Wanner, G., E. Hairer, and S.P. Nørsett (1978). "Order stars and stability theorems". In: *BIT* 18, pp. 475–489.

Weisstein, Eric W. (2016). *Graph Automorphism [online]*. URL: <http://mathworld.wolfram.com/GraphAutomorphism.html> (visited on 09/21/2016).

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media. Chap. 7, p. 972.

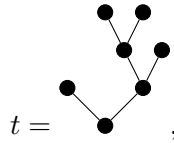
Appendix A

Solutions to tutorial exercises

A.1 Deriving Runge-Kutta Methods

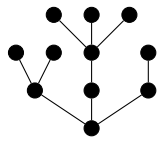
The solutions to the tutorial exercises on page 11.

1. $t = [\tau[\tau][\tau^3]], r(t) = 8, \sigma(t) = 6, \gamma(t) = 64$
2. $r(t) = 10, \sigma(t) = 4, \gamma(t) = 480$
3. (a)



$$\begin{aligned}
 t &= , \\
 r(t) &= 7, \\
 \Phi(t) &= \sum b_i a_{ij} a_{jk} c_j c_k^2, \\
 \gamma(t) &= 7, \\
 F(t) &= \mathbf{f}''(\mathbf{f}, \mathbf{f}''(\mathbf{f}''(\mathbf{f}, \mathbf{f})), \mathbf{f}).
 \end{aligned}$$

(b)



$$\begin{aligned}
 t &= , \\
 r(t) &= 11, \\
 \Phi(t) &= \sum b_i a_{ij} a_{ik} a_{il} a_{km} c_j^2 c_\ell c_m^3, \\
 \gamma(t) &= 1320, \\
 F(t) &= \mathbf{f}'''(\mathbf{f}''(\mathbf{f}, \mathbf{f}), \mathbf{f}'\mathbf{f}'''(\mathbf{f}, \mathbf{f}, \mathbf{f}), \mathbf{f}'\mathbf{f}).
 \end{aligned}$$




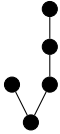

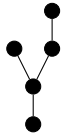
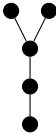

5. The order conditions for a third-order Runge-Kutta method are

$$\begin{aligned}
 b_1 + b_2 + b_3 &= 1, \\
 b_2 c_2 + b_3 c_3 &= \frac{1}{2},
 \end{aligned}$$

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

$$b_3 a_{32} c_2 = \frac{1}{6}.$$

6.

Tree				
t	$[\tau^4]$	$[\tau^2 [\tau]]$	$[\tau [\tau^2]]$	$[\tau [[\tau]]]$
$\gamma(t)$	5	10	15	30
$\Phi(t)$	$\sum b_i c_i^4$	$\sum b_i a_{ij} c_i^2 c_j$	$\sum b_i a_{ij} c_i c_j^2$	$\sum b_i a_{ij} a_{jk} c_i c_k$
Tree				
t	$[[\tau^3]]$	$[[\tau [\tau]]]$	$[[[\tau^2]]]$	$[[[[\tau]]]]$
$\gamma(t)$	20	40	60	120
$\Phi(t)$	$\sum b_i a_{ij} c_j^3$	$\sum b_i a_{ij} a_{jk} c_j c_k$	$\sum b_i a_{ij} a_{jk} c_k^2$	$\sum b_i a_{ij} a_{jk} a_{kl} c_l$

A.2 Adaptive Step Size Control

Solutions to the tutorial exercises on page 24.

1. (a) $y(0.05) = 0.030113$
- (b) $h_{new} = 0.044043$. Since $est = 1.137712 \times 10^{-4}$ which is larger than $tol = 10^{-4}$ the step is repeated using h_{new} .
- (c) There were 11 successful steps, 1 failed steps and 72 function evaluation used in total.

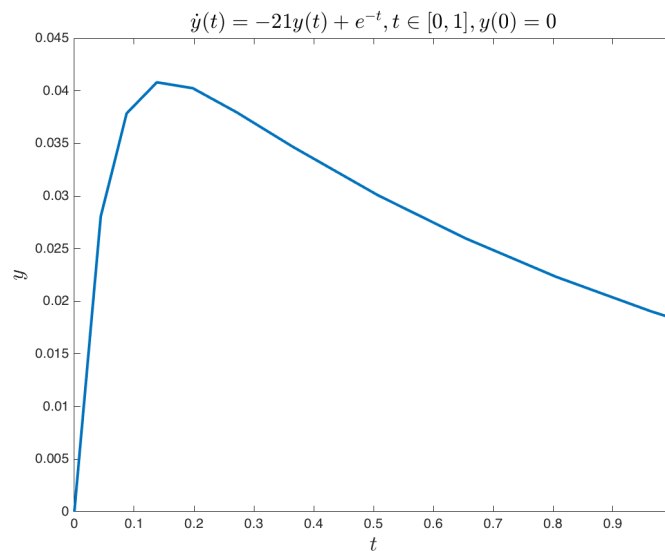


Figure A.1: Solution to the IVP using Fehlberg's Runge-Kutta method with adaptive step size control.

2. (a)

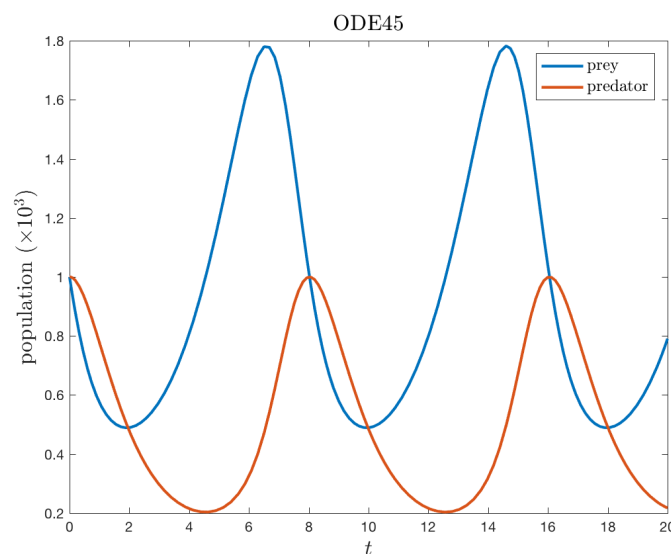


Figure A.2: Solution to the Lotka-Volterra equations using the RK4 method.

- (b) The details of each of the methods used to solve the Lotka-Volterra equations are shown in Table A.1. Note that your CPU times will be different from the ones shown below.

Table A.1: Solver details for the methods used to solve the Lotka-Volterra equations.

Method	CPU time (s)	successful steps	failed steps	function evaluations
RK4	0.0498	-	-	800
RK4 with step doubling	0.0083	65	9	814
ODE45	0.0066	62	2	385

- (c) The `ode45` solver uses the Dormand-Prince embedded Runge-Kutta method.

A.3 Stability & Order Stars

Solutions to the tutorial exercises on page 50.

1. (a) Equilibrium points at $(n\pi, 0)$ where $n \in \mathbb{Z}$.

(b) $J = \begin{pmatrix} 0 & 1 \\ -g \cos(\theta_1) & 0 \end{pmatrix}$ therefore

$$J_{(0,0)} = \begin{pmatrix} 0 & 1 \\ -g & 0 \end{pmatrix},$$

$$J_{(n\pi,0)} = \begin{pmatrix} 0 & 1 \\ g & 0 \end{pmatrix}, \quad n = 1, 3, \dots$$

$$J_{(n\pi,0)} = J_{(0,0)}, \quad n = 2, 4, \dots$$

- (c) For the equilibrium point at $(n\pi, 0)$ where n is even, $\lambda_1 = i\sqrt{g}$ and $\lambda_2 = -i\sqrt{g}$.

For the equilibrium point at $(n\pi, 0)$ where n is odd, $\lambda_1 = \sqrt{g}$ and $\lambda_2 = -\sqrt{g}$.

- (d) For this problem the Euler method is stable for step lengths in the interval $h \in [0, 1/\sqrt{g}] \approx [0, 0.319]$

2. The equilibrium points are at $(0, 0)$ and $(\gamma/\delta, \alpha/\beta)$. The Jacobian is

$$J = \begin{pmatrix} \alpha - \beta y & -\beta x \\ \delta y & \delta x - \gamma \end{pmatrix}$$

For the equilibrium point at $(0, 0)$, $\lambda_1 = \alpha$ and $\lambda_2 = -\gamma$.

For the equilibrium point at $(\frac{\gamma}{\delta}, \frac{\alpha}{\beta})$, $\lambda_1 = i\sqrt{\alpha\gamma}$ and $\lambda_2 = -i\sqrt{\alpha\gamma}$.

- 3.

$$\exp_{0/3}(z) = \frac{1}{1 - z + \frac{1}{2}z^2 - \frac{1}{6}z^3},$$

$$\exp_{1/3}(z) = \frac{1 + \frac{1}{4}z}{1 - \frac{3}{4}z + \frac{1}{4}z^2 - \frac{1}{24}z^3},$$

$$\exp_{2/3}(z) = \frac{1 + \frac{2}{5}z + \frac{1}{20}z^2}{1 - \frac{3}{5}z + \frac{3}{20}z^2 - \frac{1}{60}z^3}.$$

4. The order stars for $\exp_{0/3}(x)$, $\exp_{1/3}(x)$ and $\exp_{2/3}(x)$ are plotted in Fig. A.3. The Padé approximant $\exp_{0/3}(x)$ is not an A function but $\exp_{1/3}(x)$ and $\exp_{2/3}(x)$ are A functions.

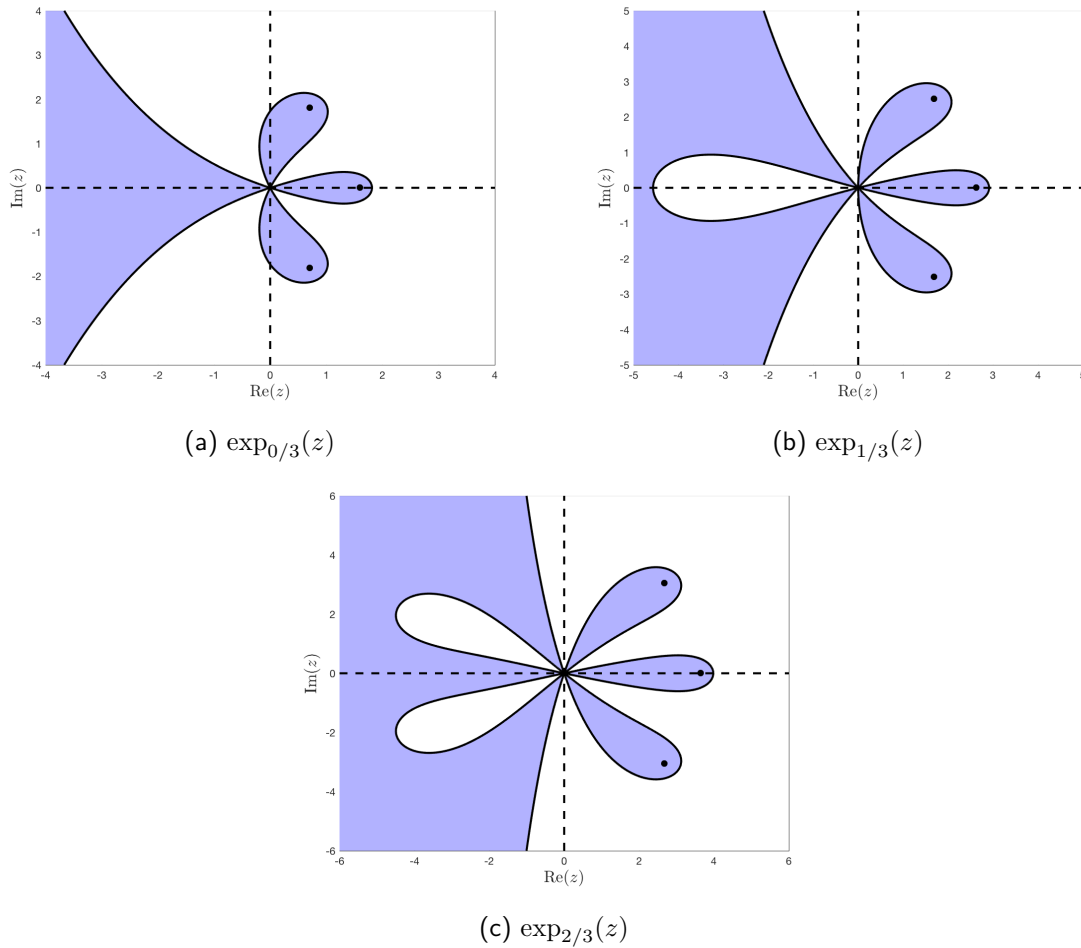


Figure A.3: Order stars for $\exp_{L/3}(x)$ where $L = 0, 1, 2$.

5. For a 9th order Radua IIA method the stability function is

$$\exp_{4/5}(z) = \frac{1 + \frac{4}{9}z + \frac{1}{12}z^2 + \frac{1}{126}z^3 + \frac{1}{3024}z^4}{1 - \frac{5}{9}z + \frac{5}{36}z^2 - \frac{5}{252}z^3 + \frac{5}{3024}z^4 - \frac{1}{15120}z^5}$$

The order star for $\exp_{4/5}(z)$ is plotted in Fig. A.4. This shows that the 9th order Radua IIA method is an A stable method.

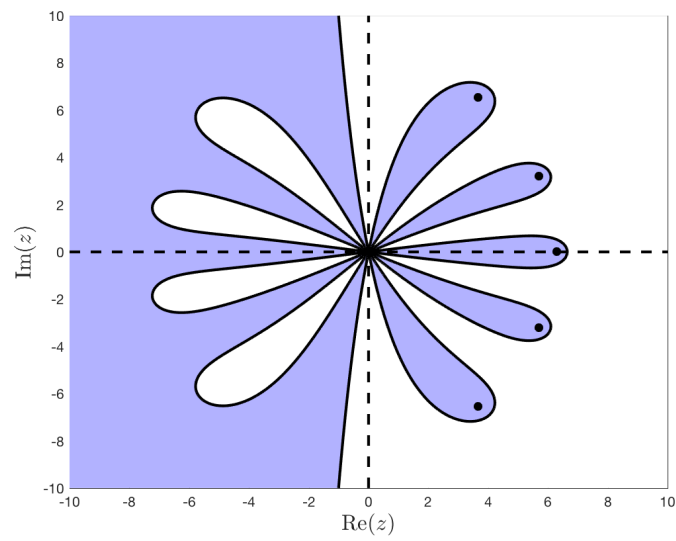


Figure A.4: Order star for $\exp_{4/5}(z)$ that represents the stability function for the 9th-order Radau IIA method.

A.4 Implicit Runge-Kutta Methods

Solutions to the tutorial exercises on page 37.

1. (a) The stage values for the RadauIA method are calculated by solving

$$\begin{pmatrix} 1 - ht_n & t_n + 2h/3 \\ -t_n & 1 - 5(t_n + 2h/3)/3 \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} y_n \\ y_n \end{pmatrix},$$

For the first step, $t_0 = 0$, $y_0 = 1$ and $h = 0.1$ so $Y_1 = 0.9933$ and $Y_2 = 1.0112$. The solution for the first step is

$$y_1 = y_0 + h \left(\frac{1}{4}Y_1 + \frac{3}{4}Y_2 \right) = 1 + 0.1 \left(\frac{1}{4}(0.9933) + \frac{3}{4}(1.0112) \right) = 1.1007.$$

- (b) The MATLAB code used to compute the solution is given below

```
% irk_q1.m by Jon Shiach
%
% This program calculates the solution to the IVP
%
%           y' = 4ty,    0 <= t <= 1,    y(0) = 1
%
% using the third-order RadauIA method with step length h = 0.1.

% Clear workspaces
clear
clc

% Define solution parameters
tspan = [ 0, 1 ];
y0 = 1;
h = 0.1;
nsteps = (tspan(2) - tspan(1)) / h;

% Setup solutions arrays
y = y0;
t = tspan(1);
yout = zeros(nsteps+1, 1);
tout = zeros(nsteps+1, 1);
yout(1) = y;
tout(1) = t;

% Step through domain
for n = 1 : nsteps

    % Solve for the stage values
    A = [ 1 - h*t, h*(t + 2*h/3) ; -h*t, 1 - 5* h*(t + 2*h/3)/3 ];
    rhs = [ y ; y ];
    Y = A \ rhs;

    % Update solution
    y = y + h*(1/4*Y(1) + 3/4*Y(2));
    t = t + h;
    yout(n+1) = y;
    tout(n+1) = t;

end

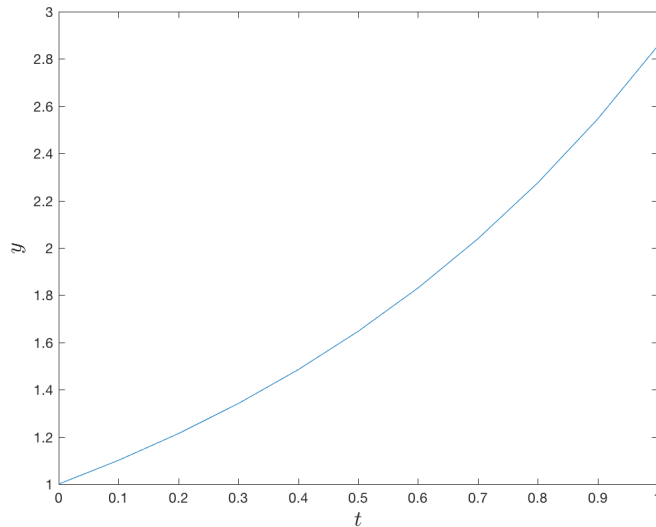
% Plot solution
```

```

plot(tout, yout)
xlabel('$t$', 'fontsize', 16, 'interpreter', 'latex')
ylabel('$y$', 'fontsize', 16, 'interpreter', 'latex')

```

The solution to this IVP is plotted below.



2. (a) Let $y_1 = \theta$ and $y_2 = y_1'$ then

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= -\frac{g}{\ell} \sin(y_1). \end{aligned}$$

- (b) We can differentiate this system

$$\mathbf{f}_y(t, y) = \begin{pmatrix} 0 & 1 \\ -g \cos(y_1)/\ell & 0 \end{pmatrix}$$

and since $h = 0.1$, $\ell = 1$, $g = 9.81$ and $y_1 = 1$ then the Jacobian for the fourth-order Gauss-Legendre method is (using Eq. (3.16))

$$J(Y) = I_4 - h(A \otimes I_2)(I_2 \otimes \mathbf{f}_y) = \begin{pmatrix} 1.0000 & -0.0250 & 0 & 0.0039 \\ 0.1325 & 1.0000 & -0.0205 & 0 \\ 0 & -0.0539 & 1.0000 & -0.0250 \\ 0.2855 & 0 & 0.1325 & 1.0000 \end{pmatrix}.$$

Performing Newton's iterations

$$\begin{aligned} G(Y^{(0)}) &= \begin{pmatrix} 0 \\ 0.1744 \\ 0 \\ 0.6510 \end{pmatrix}, \quad \Delta Y = \begin{pmatrix} -0.0019 \\ -0.1747 \\ -0.0256 \\ -0.6471 \end{pmatrix}, \quad Y^1 = \begin{pmatrix} 0.9981 \\ -0.1747 \\ 0.9744 \\ -0.6471 \end{pmatrix}, \quad \|\Delta Y\| = 0.6708, \\ G(Y^1) &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0.0001 \end{pmatrix}, \quad \Delta Y = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0.0001 \end{pmatrix}, \quad Y^2 = \begin{pmatrix} 0.9981 \\ -0.1747 \\ 0.9744 \\ -0.6470 \end{pmatrix}, \quad \|\Delta Y\| = 0.0001. \end{aligned}$$

The solution for the first step is

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{y}_0 + h(\mathbf{b}^T \otimes I_2)F(t_0, Y) \\ &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 0.1 \begin{pmatrix} 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \end{pmatrix} \begin{pmatrix} 0.9981 \\ -0.1747 \\ 0.9744 \\ -0.6470 \end{pmatrix} = \begin{pmatrix} 0.9589 \\ -0.8181 \end{pmatrix}. \end{aligned}$$

(c) The MATLAB code used to compute the solution is given below

```
% irk_q2.m by Jon Shiach 2018
%
% This program solves the van der Pol equation using an IRK method

% Clear workspaces
clear
clc

% Define solution parameters
global N s g l
tspan = [ 0, 20 ];           % domain
y0 = [1 ; 0];               % initial values of y
N = 2;                      % no. equations in the system
h = 0.1;                    % step length
nsteps = (tspan(2) - tspan(1)) / h;
tol = 1d-4;
g = 9.81;                   % acceleration due to gravity
l = 1;                      % length of the chord

% Define fourth-order Gauss-Legendre method
A = [1/4, 1/4-sqrt(3)/6 ;
     1/4+sqrt(3)/6, 1/4 ];
b = [ 1/2 ; 1/2 ];
c = [1/2 - sqrt(3)/6 ; 1/2 + sqrt(3)/6];
s = 2;

% Initialise solution arrays
y = y0;
t = tspan(1);
yout = zeros(nsteps+1, N);
tout = zeros(nsteps+1, 1);
yout(1, :) = y';
tout(1) = t;
e = ones(s, 1);

% Step through the domain
for n = 1 : nsteps + 1

% Initialise Y and calculate Jacobian matrix
Y = kron(e, y);
J = eye(N*s) - h*kron(A, eye(N))*kron(eye(s), Jac(t, y));

% Perform Newton iterations to solve for Y
err = 1;
while err > tol
    G = Y - kron(e, y) - h*kron(A, eye(N))*F(t, Y);
    Delta_Y = J \ -G;
    Y = Y + Delta_Y;
    err = norm(Delta_Y);
end
end
```

```

end

% Update solution
y = y + h*kron(b', eye(N))*F(t, Y);
t = t + h;
yout(n+1, :) = y';
tout(n+1) = t;
end

% plot solution
plot(tout, yout(:, 1))
axis([tspan, -1.5, 1.5])

% =====
function dy = f(~, y)

% This function defines the ODE system
global g l
dy = [ y(2) ;
      -g/l*sin(y(1)) ];
end

% =====
function J = Jac(~, y)

% This function calculates the Jacobian matrix for the ODE system
global g l
J = [ 0 1 ; -g/l*cos(y(1)), 0 ];

end

% =====
function Fout = F(t, Y)

% This function defines the vector F
global N s
Fout = zeros(N*s, 1);
j = 1;
for i = 1 : s
    Fout(j : j + N - 1) = f(t, Y(j : j + N - 1));
    j = j + N;
end

end

```

3. The MATLAB commands used to compute the solution using `ode23s` are given below.

```

options = odeset('Stats', 'on');
[t, y] = ode23s(@f, tspan, y0, options);

```

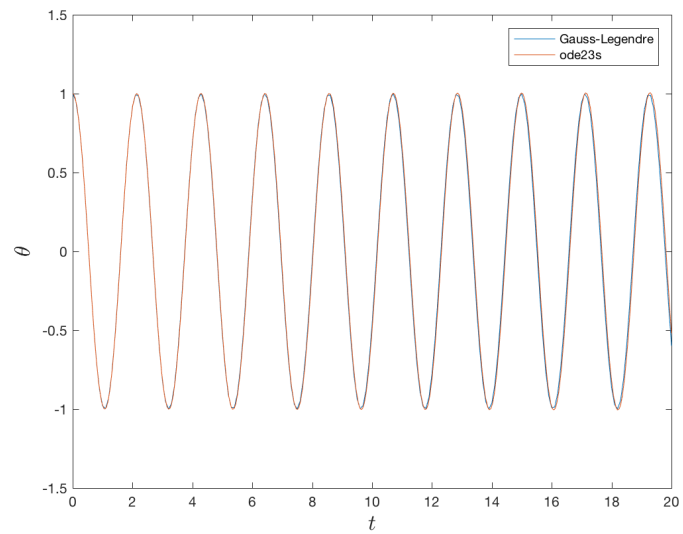
The solver statistics are

```

326 successful steps
50 failed attempts
1732 function evaluations
326 partial derivatives
376 LU decompositions
1128 solutions of linear systems

```

The solutions computed using the Gauss-Legendre IRK method and `ode23s` are plotted below.



A.5 Applications of ODEs – the N body problem

Solutions to the tutorial exercises on page 62.

1.

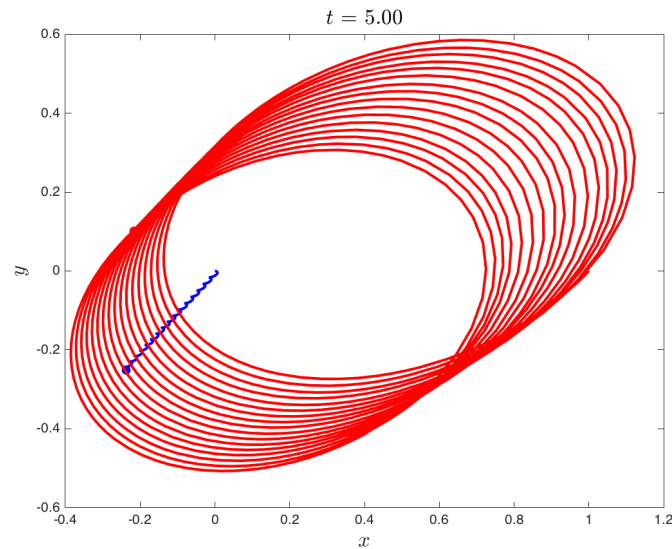


Figure A.5: Solution to a 2-body problem with initial conditions $\mathbf{p}_1 = (0, 0)$, $\mathbf{p}_2(0) = (1, 0)$, $\mathbf{v}_1 = (0, 0)$, $\mathbf{v}_2 = (-5, -5)$, $m_1 = 1$, $m_2 = 100$ and $G = 1$.

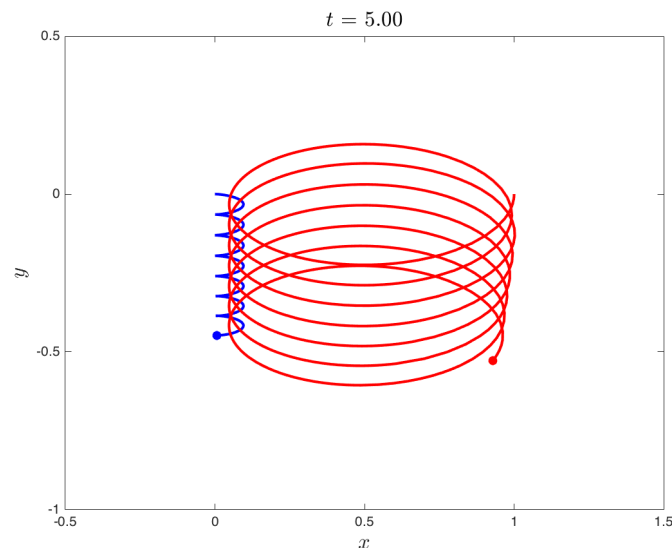


Figure A.6: Solution to a 2-body problem with initial conditions $\mathbf{p}_1 = (0, 0)$, $\mathbf{p}_2(0) = (1, 0)$, $\mathbf{v}_1 = (0, 0)$, $\mathbf{v}_2 = (0, -1)$, $m_1 = 1$, $m_2 = 10$ and $G = 1$.

3. The three bodies move in a periodic figure 8 orbit.

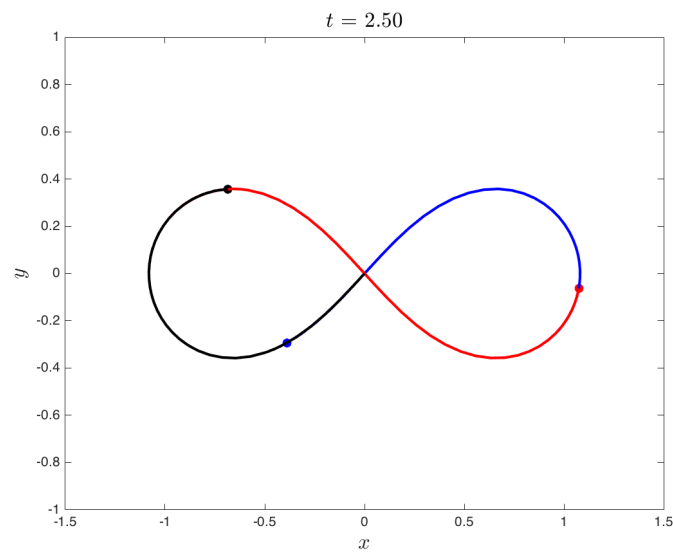


Figure A.7: Solution to a 3-body problem that exhibits periodic figure 8 orbit.

4.

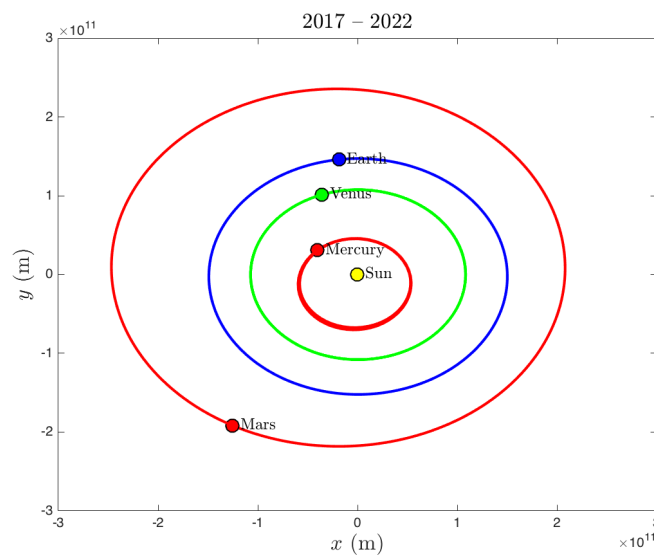


Figure A.8: The orbits of the four closest planets to the Sun.

5. Despite its mass relative to the other bodies in the solar system, the Sun's position does change due to the gravitational force exerted by the planets. If another star moves in a similar way then it suggests that it is at the centre of a solar system.

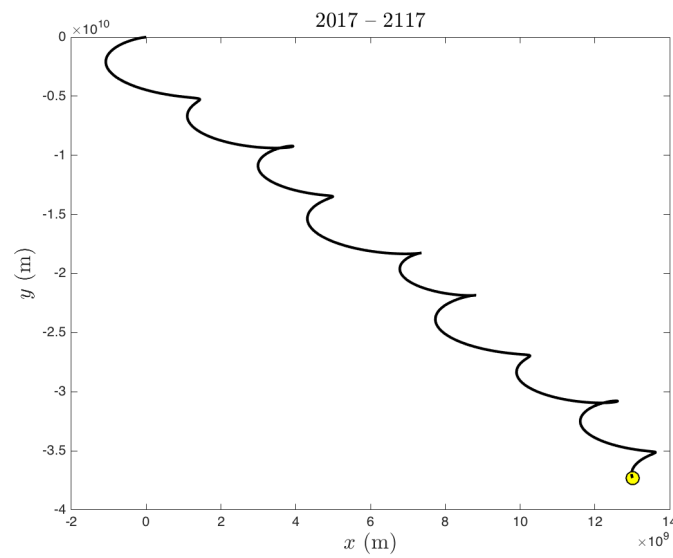


Figure A.9: The motion of the Sun's position in space due to the gravitational force exerted by the planets of the solar system.

Appendix B

Ephemeris for solar system bodies

The data in these tables was obtained from the NASA's Jet Propulsion Laboratory (JPL) Horizons system (Giorgini, 2016) <http://ssd.jpl.nasa.gov/horizons.cgi>.

Table B.1: Co-ordinates for the bodies in the solar system at 00:00 1st January 2017 (units are AU).

Body	x	y	z
Sun	$0.0000000000E + 00$	$0.0000000000E + 00$	$0.0000000000E + 00$
Mercury	$-1.4337194579E - 01$	$2.8370937153E - 01$	$3.6335706120E - 02$
Venus	$4.6732439979E - 01$	$5.5082312871E - 01$	$-1.9414673058E - 02$
Earth	$-1.7961365192E - 01$	$9.6679492050E - 01$	$-3.6687303846E - 05$
Mars	$1.3547028701E + 00$	$3.8687490303E - 01$	$-2.5140097262E - 02$
Jupiter	$-5.3597336247E + 00$	$-1.0126710502E + 00$	$1.2413595220E - 01$
Saturn	$-1.8684958796E + 00$	$-9.8697045476E + 00$	$2.4592349924E - 01$
Uranus	$1.8339238172E + 01$	$7.8262157953E + 00$	$-2.0839483086E - 01$
Neptune	$2.8336082160E + 01$	$-9.6934185530E + 00$	$-4.5341880267E - 01$
Pluto	$9.6600843904E + 00$	$-3.1802586396E + 01$	$6.0755835269E - 01$
Moon	$-1.7787813595E - 01$	$9.6484001964E - 01$	$5.3542927250E - 05$

Table B.2: Velocities for the bodies in the solar system at 00:00 1st January 2017 (units are AU/day).

Body	u	v	w
Sun	$0.0000000000E + 00$	$0.0000000000E + 00$	$0.0000000000E + 00$
Mercury	$-3.0768488424E - 02$	$-1.1623533642E - 02$	$1.8729708227E - 03$
Venus	$-1.5485267593E - 02$	$1.2998526018E - 02$	$1.0718324616E - 03$
Earth	$-1.7200383605E - 02$	$-3.2111862156E - 03$	$7.9277707382E - 07$
Mars	$-3.3070296717E - 03$	$1.4653201202E - 02$	$3.8822858604E - 04$
Jupiter	$1.3117028869E - 03$	$-7.0652781505E - 03$	$-1.6241637397E - 09$
Saturn	$5.1777132886E - 03$	$-1.0608588095E - 03$	$-1.8754455925E - 04$
Uranus	$-1.5704503553E - 03$	$3.4272524407E - 03$	$3.2938367343E - 05$
Neptune	$9.9776070300E - 04$	$2.9824553077E - 03$	$-8.4702821626E - 05$
Pluto	$3.0712757363E - 03$	$2.4670547535E - 04$	$-9.2747754344E - 04$
Moon	$-1.6780783692E - 02$	$-2.8109343090E - 03$	$-4.7697885503E - 05$

Table B.3: Mass of the major bodies in the solar system (units are kg).

Body	Mass	Body	Mass	Body	Mass
Sun	$1.988544E + 30$	Mercury	$3.302000E + 23$	Venus	$4.868500E + 24$
Earth	$5.972190E + 24$	Mars	$6.418500E + 23$	Jupiter	$1.898130E + 27$
Saturn	$5.683190E + 26$	Uranus	$8.681030E + 25$	Neptune	$1.024100E + 26$
Pluto	$1.307000E + 22$	Moon	$7.349000E + 22$		