Geometric Numerical Integration

Numerical Methods for Qualitative Preservation of Ordinary Differential Equations

Will Woolfenden

10628968

Supervisor: Dr. Marcus Webb

Final submission for MATH40000 Double Project for the 2023/24 academic year

Department of Mathematics
School of Natural Sciences
The University of Manchester
United Kingdom
May 14, 2024

Abstract

Geometric methods for integrating dynamical systems aim to preserve behaviour from the system in the numerical solution. General-purpose integration schemes are introduced in order to understand the behaviour of numerical solutions to ODEs and build the foundation for geometric integration. Symplectic integrators develop from the structure of Hamiltonian systems. These methods are generally implicit but can be explicit if the Hamiltonian is separable. Examples show the utility of symplectic methods in physical scientific applications. Backward error analysis shows that the symplectic method exactly solves a problem whose modified Hamiltonian has closeness to the original depending on the order of the method used. Positivity preserving methods in our analysis are formulated for problems involving a graph-Laplacian matrix. Methods which are of current interest are second order, which is shown, and employ the matrix exponential. Two adjustments to a positivity preserving method are proposed, where if an approximation of the matrix exponential will unconditionally preserve positivity then it can be employed in order to reduce cost.

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 An Introduction to Numerical Methods . . . 3
- 1.3 Structure Preservation . . . 6
- 1.4 Implicit Methods, Stability . . . 6
- 1.5 Utility and Cost of Qualitative Preservation . . . 8
- 1.6 Literature Review . . . 9
- 1.7 Structure . . . 9

2 Symplectic Integration **11**
- 2.1 Hamiltonian Systems and Numerical Methods . . . 11
- 2.2 Further Hamiltonian Dynamics . . . 14
- 2.3 Generalised Symplectic Methods . . . 21
- 2.4 Analysis of Symplectic Methods . . . 28
- 2.5 Applications . . . 33

3 Positivity Preservation **35**
- 3.1 Positive Solutions to ODEs . . . 35
- 3.2 Positivity Preserving Methods . . . 37
- 3.3 Approximation of the Matrix Exponential . . . 48
- 3.4 Wider Positivity Preservation . . . 64

4 Review and Discussion **66**
- 4.1 Improvements . . . 66
- 4.2 Review . . . 67
- 4.3 Conclusion . . . 69

A Supplementary Content **72**
- A.1 Preliminary . . . 72
- A.2 Positivity Preservation . . . 73

B MATLAB Implementations **76**
- B.1 Numerical Integration . . . 76
- B.2 Symplectic Integration . . . 79
- B.3 Positivity Preservation . . . 84

Chapter 1

Introduction

1.1 Motivation

The aim of this project is to discuss and analyse methods for the preservation of qualitative behaviour when solving ordinary differential equations (ODEs) numerically. We wish to provide the reader with an understanding of how numerical methods can be formulated with the goal of qualitative preservation, which involves exploring the formulation and modification of problems themselves in order to be solved in these ways. As we will find, methods relevant to our interests are explored and derived using fundamentally different approaches to the design of conventional methods. The term "geometric" refers to an inherent quality of some system. For example, we might have a mathematical model describing the motion of some object over the surface of a sphere. The "geometric" property of this model is that its solutions must describe a point on this surface. The term "numerical integration" refers to numerical methods which are used to approximate solutions of, for our purposes, ordinary differential equations.

We will discuss properties of numerical methods and how the motivation for geometric numerical integration arises. The popular numerical integration schemes that we will explore provide a numerical approximation to the solution of an ODE, by knowing the definition of the problem in its most general form. The classical methods are general, and will provide a numerical solution to any defined problem, however these methods are not designed for preservation of anything beyond the definition of the derivative itself. This is not to say these methods need improving in any way, as their formulation is arguably the correct approach for developing a general purpose numerical integration scheme for solving ODEs. If we have some general problem and no notion of its invariant properties, we can use a classical scheme to solve it numerically. If we have a different problem, and some definite qualities that we wish to preserve, we can use these same classical methods to obtain a numerical solution. However this is an approximation and has no guarantee of actually preserving our properties of interest. The alternative would be to propose an integrator specifically with qualitative preservation in mind. If we then wanted to solve another problem *without* any notion of geometric preservation, then we need a general "classical" method anyway. We will develop an understanding of these methods themselves and their properties before our discussion on geometric numerical integration. For reference, when speaking of "classical" integration schemes, we mean methods such as Euler or the popular multi-stage Runge-Kutta methods we see in MATLAB.

One interesting facet of geometric numerical integration is that in general, we cannot simply modify a popular method in order to preserve geometric qualities. Rather, we must define a method which requires information on that quality itself. For example, the positivity preserving methods we

will study later require the formulation of problem as an equation which we know preserves positivity itself. The key here is that not only must the method be specially designed for geometric preservation, but the problem itself must be expressed in this less general form, which itself analytically admits preservation of this same quality. This is what allows these geometric integration schemes to preserve these quantities, which is our goal, as well as actually integrating the system in the same fashion as classical methods.

This is not to say that popular classical integrators do not preserve qualitative behaviour, but the distinction is that they do not do so unconditionally. This problem mostly arises in long timespan integrations. When solving a problem using a popular integrator, it is possible to accrue enough error such that the behaviour of the system changes. Our goal is to investigate specialised numerical integration schemes which preserve quality, regardless of all the other parameters given to the method.

Geometric numerical integration methods can be summarised as more specialised integrators for more specialised problems. This report explores two facets of geometric numerical integration. The first is symplectic integration, where the flow map defined by a method must preserve area, or the n-dimensional equivalent of "area", when acting on any region in the phase plane. The second is positivity, where all the variables of interest are inherently positive quantities.

1.2 An Introduction to Numerical Methods

To start, we will give a brief discussion of explicit numerical methods. A system of ordinary differential equations often arises when constructing a mathematical model to describe some sort of system: a set of variables and a description of how they change in time. In absolute most general form, a system of ODEs has the appearance

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(t, x) \tag{1.1}$$

where the left-hand side expresses a change of x in time, while the right-hand side is some arbitrary function on t and x. This is a first-order ODE, but in generality we can use this form to express ODEs of any order by considering the solution to be a vector of variables. Consider the example second order problem

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = \mathrm{e}^{tx}.$$

In this case, let us write this equation in terms of a vector $\mathbf{x}$ of derivatives:

$$\mathbf{x} := \begin{pmatrix} x \\ \dot{x} \end{pmatrix}.$$

For compactness, we use the dot notation as a shorthand for a time derivative. We can write our problem as

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ \dot{x} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \mathrm{e}^{tx} \end{pmatrix}$$

which can be expressed in general as

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = F(t, \mathbf{x})$$

where F is an arbitrary function again, by no means linear in $\mathbf{x}$. This framework is fundamental to the methods we will look into for solving ODEs, hence we introduce it now. The problems we will

look at will involve vector solutions in general, so our notation will usually instead denote the whole vector quantity of interest as x, and make sure to individually distinguish its elements if needed.

Since there is a mixed tx term, this is a nonlinear equation. This means that we cannot write a solution $x(t)$ in closed form using analytical methods. Instead, we need to use a numerical method. A numerical method provides us with a sequence of values $(x_i)_{i=1}^n$ for given points in time $(t_i)_{i=1}^n$, where each x_i is an approximation to the true solution $x_i \approx x(t_i)$. This is the "numerical integration" of this project. Numerical methods, especially the ones we will look at, are aimed at solving Initial Value Problems (IVPs), where we want to solve the ODE given an initial condition $x(t=0) = x_0$. Alternatively, a boundary value problem specifies conditions on the boundary of a domain. For example, consider a second order ODE[1] with constraints at $x(t=0)$ and $x(t=t_n)$. Numerical methods for boundary value problems are fundamentally different to the methods for initial value problems we will study, hence we will not consider them in this report.

For a first example, we will consider Euler's method. This method starts by assuming a value $x(t_n) = x_n$ and considering the next value. We denote the *step size* by h, being the difference $t_{n+1} - t_n$. Then we consider the next value by evaluating the Taylor series

$$x(t_n + h) = x(t_n) + h\dot{x}(t_n) + \frac{h^2}{2}\ddot{x}(t_n) + \frac{h^3}{6}\dddot{x}(t_n) + \ldots = \sum_{k=0}^{\infty} \frac{h^k x^{(k)}(t_n)}{k!}.$$

On the left-hand side, the next point in time t_{n+1} is the same as $t_n + h$. On the right-hand side, if we assume h is small then we can simplify this problem by assuming that every term of h of order 2 or greater is negligibly small. We write this as $\mathcal{O}(h^2)$. The big-O notation means that as $h \to 0$, the expression is bounded above by some constant times h^2. The whole equation reduces to

$$x(t_{i+1}) = x_i + hf(t_i, x_i) + \mathcal{O}(h^2).$$

We know what f is, since it defines the ODE. We also assumed a value x_i for a value (or approximation) of $x(t_i)$. If we ignore the $\mathcal{O}(h^2)$ term, we are left with what is generally referred to as the forward Euler method

$$x_{i+1} = x_i + hf(t_i, x_i) \tag{1.2}$$

for solving an ODE. This is an example of an explicit method, coming from how the next term is obtained using only information which is immediately available.

One element that we will focus on is the concept of error and accuracy of a numerical method. Euler's method is first-order accurate, meaning that the expressions for the true solution (by Taylor series) and the approximation (by the method) are the same up to and including the term in h of order 1. Here we define the concept of truncation error as $\tau(h) = x(t_i) - x_i$, being the error between the true solution and the approximation for a single step of size h. As we have seen, the local truncation error of Euler's method is $\mathcal{O}(h^2)$. We also want to consider the concept of global truncation error, which is the error accumulated over an entire numerical solution. This is important because each step comes from a previous point, and only the initial value is given as exact. The local error of Euler's method is $\mathcal{O}(h^2)$, and assume it requires N timesteps in order to compute a numerical solution. Then the global error is $\mathcal{O}(Nh^2)$. But N is proportional to $1/h$ since $h = (t_n - t_0)/N$, and so the global error is $\mathcal{O}(h)$. This argument can be generalised to any method with uniform timestep, but also applies to the error of a general, variable time-step method [14]

The problem of global error analysis arises when considering how we may need a computation to match a certain tolerance. Assume we need the global error of a numerical solution to be below ϵ. This means we need the sum of the truncation errors across the whole computation to be below

[1] A problem of order n requires n conditions in order to have a unique solution.

this threshold. Ignore how we compute or estimate the truncation error, which we will explore later. Assume we perform a computation and our global error is less than or equal to 8ϵ. We need to divide the global error by 8. For a method with global truncation error $\mathcal{O}(h^3)$, we can half the step size and run the computation again to match this tolerance, since if $Ch^3 \approx 8\epsilon$ then $C(h/2)^3 \approx \epsilon$. Whereas for a lower order method, this would be much more expensive, perhaps prohibitively more so.

Despite Euler's method being only first order, it is still a convergent method.

Definition 1.1. A numerical method is convergent if, as $h \to 0$, the numerical solution $(x_i)_{i=1}^n$ converges to the true solution $x(t)$.

This means that we can always get a good enough approximation from any convergent method as long as we can decrease the step size indefinitely. That being said, it would be computationally expensive to keep reducing the step size of Euler's method.

If we want to improve the order at which error decreases, we may start by considering a Runge-Kutta (RK) method. These are methods given by

$$x_{n+1} = x_n + h\left(\sum_{i=1}^{s} b_i k_i\right)$$

where the b_i are coefficients and the k_i are "guesses" of the form

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

By using linear combinations of the k_i, we want to reduce the truncation error to a certain order. For compactness, a Runge-Kutta method is often expressed as its Butcher tableau

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \dots & a_{ss} \\ \hline & b_1 & \dots & b_s \end{array}$$

which is just an expression of matrices and vectors, analogous to

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

Runge-Kutta methods are extremely popular choices for higher order integrators. Consider a two stage RK method

$$x_{n+1} = x_n + h(b_1 k_1 + b_2 k_2)$$

with the k_i defined

$$\begin{aligned} k_1 &= f\left(t_n, x_n\right) \\ k_2 &= f\left(t_n + c_2 h, x_n + h a_{21} k_1\right) \end{aligned}$$

from their defintions. Note that the expressions are simplified, coming from how k_i does not use any information from k_j if $j \geq i$. This is required in order for the method to be explicit. Equivalently, the A matrix is strictly lower triangular. The necessary conditions [21] are $b_1 + b_2 = 1$, $b_2 c_2 = 1/2$ and $a_{21} = c_2$ for the method to be second order accurate. These conditions are underdetermined,

so there are several configurations to choose from. A popular choice for a second order RK method is also referred to as the *explicit trapezium method*

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

where $c_1 = 0, c_2 = 1$ corresponds to evaluating k_1 at the initial point and using k_1 to make a forward Euler estimate of x_{n+1}, to use in computing k_2. We then take an average of the two, hence the "trapezium" name. Written out in full, the method is

$$x_{n+1} = x_n + \frac{h}{2}\left(f(t_n, x_n) + f(t_n + h, x_n + hf(t_n, x_n))\right)$$

1.3 Structure Preservation

The methods discussed, particularly the Runge-Kutta methods, are excellent choices when we may be crudely looking for an approximate solution without any regard to the concept of an "invariant". Many systems admit invariant quantities. Dynamical systems describing the motion of rigid bodies admits conservation of energy. Chemical reaction systems admit conservation of mass. However, in no part of our discussion on numerical methods did we acknowledge this requirement. Methods such as Euler and the explicit Runge-Kutta schemes we have established are designed in order to provide an approximate solution in the sense of having a particular truncation error. There is no guarantee that these methods will preserve any invariant quantity of a system. This is not to say they are guaranteed not to do so, and we will see methods only slightly generalised from those already discussed, which do manage to behave well in terms of qualitative preservation.

However, before we begin introducing structure-preserving qualities, we will first establish more results and provide discussions on the behaviour of numerical methods. We have only discussed explicit methods so far, whereas a lot can be gained from the introduction of implicit methods. We introduce these properties now since they are important later on when developing more powerful methods in terms of preservation of structure.

1.4 Implicit Methods, Stability

Now that we have an understanding of some numerical methods, let us consider their use. The *linear test problem* is the ODE and initial condition given by

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= \lambda x \\ x(0) &= x_0. \end{aligned}$$

If λ has negative real part, then x goes to zero as $t \to \infty$. Consider the forward Euler method applied to this problem. The iteration is

$$\begin{aligned} x_{n+1} &= x_n + hf(t_n, x_n) \\ &= x_n + h\lambda x_n \\ &= (1 + h\lambda)x_n. \end{aligned}$$

Admittedly, we wouldn't be using this example if it was going to work perfectly. We can see that if we repeatedly apply the method, we get the expression

$$x_n = (1 + h\lambda)^n x_0.$$

The numerical solution goes to zero if $|1 + h\lambda| < 1$. Therefore, suppose the true solution goes to zero, meaning λ must be negative. The numerical solution goes to zero if $-2 < h\lambda < 0$. For values of the timestep h where $h > 2/|\lambda|$, this behaviour is not respected. This concept is called A-stability. The definition itself extends to complex values of $h\lambda$, which must be considered when considering A-stability of a given numerical method.

Definition 1.2. A numerical method is A-stable if the numerical solution of $\dot{x} = \lambda x$ goes to zero for all values of $h\lambda$ in the left half of the complex plane.

Euler's method is not A-stable because even if $h\lambda$ is negative, there are values of h for which the numerical solution does not decay. If we considered the explicit trapezium method from earlier, we would find that this method is also not A-stable. In fact, there are no explicit Runge-Kutta methods which are A-stable[2]. For our purposes, we need to introduce the concept of an *implicit* method. The simplest of these is the implicit (backward) Euler method

$$x_{n+1} = x_n + f(x_{n+1}). \tag{1.3}$$

This method requires knowing the value of x_{n+1} in order to compute x_{n+1}. A step starts at x_n and integrates using the tangent of the curve f at x_{n+1} instead of x_n. In actual implementation, the backward Euler method involves solving the equation $y = x + f(y)$ for $x_{n+1} = y$, which is potentially non-linear. In general, implementing an implicit method can be expected to involve solving a system of nonlinear equations at each step. Iterative methods for solving nonlinear equations often use descent methods to converge to a solution, and stop when the change in the iteration is within a given tolerance. These iterative methods are not expected to solve a nonlinear equation in a given finite number of steps. As such, implementing implicit methods can be expensive.

Notions of computational cost aside, we are able to achieve better results in terms of stability when considering implicit methods. Consider the backward Euler method applied to the linear test problem:

$$\begin{aligned} x_{n+1} &= x_n + f(x_{n+1}) \\ &= x_n + h\lambda x_{n+1}. \end{aligned}$$

Rearranging, the equation becomes

$$(1 - h\lambda)x_{n+1} = x_n$$

or equivalently

$$x_{n+1} = \frac{1}{1 - h\lambda} x_n.$$

In order for the numerical solution to go to zero, we need this expression on the right-hand side to be less than 1 in modulus, which is the same as requiring $|1 - h\lambda| > 1$. Considering that h must be real and positive, this inequality is not satisfied only for $0 < h\lambda < 2$. If we consider problems where λ is negative, then this is impossible, and so the backward Euler method is clearly A-stable. This is a huge improvement on the forward Euler method, which very easily failed to respect decay of the solution.

[2]We will not prove this now, but it will be justified later.

We can also notice, however, that it is possible for backward Euler to decay when the actual solution does not. This is because there are positive values of $h\lambda$ which satisfy the inequality required for the backward Euler solution to decay to zero. As a final example on A-stability, we introduce the *implicit midpoint* method, given by

$$x_{n+1} = x_n + hf\left(\frac{x_n + x_{n+1}}{2}\right). \tag{1.4}$$

applying to the linear test problem again we obtain

$$x_{n+1} = x_n + h\lambda\frac{x_{n+1} + x_n}{2}$$

which rearranges to

$$\left(1 - \frac{h\lambda}{2}\right)x_{n+1} = \left(1 + \frac{h\lambda}{2}\right)x_n$$

and so the numerical solution goes to zero if

$$\left|\frac{2 + h\lambda}{2 - h\lambda}\right| < 1.$$

This inequality is satisfied if $h\lambda$ is closer to -2 than to 2, hence the regions of growth and decay are exactly the right and left halves of the complex plane. The implicit midpoint method attains better qualities for respect of growth or decay of the true solution in comparison to the Euler methods.

The reader may notice that the implicit midpoint method we have examined is similar in appearance to the explicit trapezium method from earlier, in that the evaluation of the right hand side involves a combination of the point x_n and a second stage. The implicit method uses x_{n+1} directly, while the explicit method uses a guess in the form of a forward Euler estimate. The trapezium scheme uses an average of exaluations of f, while the midpoint evaluates f at the midpoint (average) of the two points. For linear ODEs such as the linear test problem, trapezium and midpoint schemes are identical.

We have explored how modifications of general methods are able to respect A-stability. The next step is to begin introducing more general qualities of ODE systems which we preserve, and the methods that are capable of preserving them. The reason we have given such a rigorous introduction is that A-stability is strongly related to symplecticity, which we will soon begin discussing. We also want to make sure we have a strong understanding of some basic numerical methods, not just in terms of error, but also in terms of their cost to implement. For example, we have already discussed how explicit methods are much more appealing in terms of their cost. We may want to consider how much a method improves by introducing implicit components, and how much it may justify the increased cost of the method. This problem can be further generalised to the differences inherent in geometric methods.

1.5 Utility and Cost of Qualitative Preservation

For many problems we will explore in this report, we want to evaluate the accuracy of a method. In order to do this, we might wish to evaluate an error metric by comparing an approximation to an exact solution. For problems where we cannot write a solution analytically, we don't have an exact solution and will need to use a really good approximation instead. The easiest way to do this is to use a standard integrator with a really low error tolerance. For our purposes, computations

are implemented in MATLAB using the `ode45()` function. This scheme uses a pair[3] of order 4 and 5 Runge-Kutta methods, which are not geometric methods. Therefore it seems like a poor decision to put all this work into investigating geometric methods, and then when we need to analyse them we compare their solutions to methods that completely ignore the desire for these properties. The question to consider is that in practice, it may or may not be worth developing structure preserving methods when the solution from a standard integrator will suffice. This is a problem we will occasionally come back to.

1.6 Literature Review

The first large chapter of this project focuses on symplectic integration. An introduction to symplecticity and Hamiltonian dynamics is given in the book "Numerical Hamiltonian Problems" by Sanz-Serna and Calvo [29], originally published in 1994. The authors give an overview of Hamiltonian mechanics, aiming to give readers an accessible introduction to the topic, and is one of the earlier texts in the field. Plenty of theory is consistent between the work we cover and the theory explored in this text.

The book "Geometric Numerical Integration" by Hairer, Lubich and Wanner [13] was originally published in 2002, and gives an extremely robust review of the theory of geometric integration. The authors explore symplectic integration of Hamiltonian systems, as well as integrators that preserve time symmetry, first integrals and Lie group structure. A rigorous understanding of the theory behind these methods is given, as well as plenty of visual demonstrations on the behaviour of these geometric integrators as compared to schemes such as Euler and explicit Runge-Kutta. The book serves as an excellent reference text for the subject, and provides several results in symplectic integration which we will look into.

Our second subject of interest is positivity preservation. The research paper on "Positivity-Preserving Methods for Ordinary Differential Equations" [4] is the basis for the methods we explore in the later part of this project. The discussion was published in 2022, and is the first to explore problems considering the graph-Laplacian matrix. The authors give some examples of positivity-preserving integrators for these problems, as well as discussing potential approximations for expensive computations. Positivity preservation, while part of the subject of geometric numerical integration, is a much more recent field of study. The integrators proposed by the paper are novel, and provide interest in the development of the subject.

1.7 Structure

We have already introduced the theory of numerical methods and their properties. The rest of this project is presented in two chapters where we explore the theory of geometric numerical integration and its applications. The first chapter focuses on the symplectic integration of systems that can be described in Hamiltonian dynamics. We explore some of the theory for symplectic integration schemes and analyse their effectiveness when applied to particular problems.

The second of these chapters introduces positivity preservation. We investigate integration methods in this area, and consider how new schemes could be developed. However, we also delve further into the properties of the integration schemes proposed in [4], and how they could be improved. A large section of analysis is dedicated to the methods of approximation available, such that the

[3] The difference between two approximations can be used to estimate the true error

order of these methods can be preserved. Aside from discussing methods of approximation, we also propose our own adjustments to current numerical integration methods for positivity preservation.

Chapter 2

Symplectic Integration

2.1 Hamiltonian Systems and Numerical Methods

2.1.1 Hamiltonian Dynamics

A Hamiltonian system on (q, p) [13] is given by

$$\dot{q} = \frac{\partial H}{\partial p} \qquad\qquad \dot{p} = \frac{-\partial H}{\partial q}.$$

We use q to denote position, and p to denote momentum. This is a special form of a general system of ODEs $\dot{x} = f(t, x)$. For this part of our discussion, we will primarily be concerned with *autonomous* systems of the form $\dot{x} = f(x)$.

We describe a system with its Hamiltonian $H(q, p)$, used to describe the total energy present. We can write the system as

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} q \\ p \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \dfrac{\partial H}{\partial q} \\ \dfrac{\partial H}{\partial p} \end{bmatrix}. \tag{2.1}$$

If we write $x = (q, p)^\top$ and denote the matrix as J, then the statement of a Hamiltonian system is of the form

$$\dot{x} = J\nabla H(x). \tag{2.2}$$

We call J the *symplectic matrix*. The Hamiltonian is a first integral of the system

For a d-dimensional problem, we instead denote the variables $q_1, \ldots, q_n, p_1, \ldots, p_n$ as the position and momentum in the directions of each basis vector respectively. Write $x = (q_1, \ldots, q_n, p_1, \ldots, p_n)$. Define J to be the matrix defined in blocks

$$J = \begin{bmatrix} 0_n & I_n \\ -I_n & 0_n \end{bmatrix} \tag{2.3}$$

and the problem can be written as $\dot{x} = J\nabla H(x)$. This is the general form for a Hamiltonian system. Note that for a problem in d dimensions, x belongs to $\mathbb{R}^{2d}$. The ∇ operator is specifically defined

in the (q, p) order for our purposes

$$\nabla = \begin{pmatrix} \frac{\partial}{\partial q_1} \\ \vdots \\ \frac{\partial}{\partial q_d} \\ \frac{\partial}{\partial p_1} \\ \vdots \\ \frac{\partial}{\partial q_d} \end{pmatrix}$$

2.1.2 The Simple Harmonic Oscillator

First, we look at the simple harmonic oscillator $m\ddot{x} = -kx$. In Hamiltonian variables this is

$$\dot{q} = \frac{p}{m}, \quad \dot{p} = m\ddot{q} = -kq.$$

A suitable Hamiltonian [9] is

$$H = \frac{1}{2m}p^2 + \frac{1}{2}kq^2.$$

Applying the forward Euler method to the problem, we get

$$\begin{aligned} q_{n+1} &= q_n + h\frac{\partial H}{\partial p}(q_n, p_n) &&= q_n + \frac{p_n}{m} \\ p_{n+1} &= p_n - h\frac{\partial H}{\partial q}(q_n, p_n) &&= p_n - kq_n \end{aligned}$$

which can be written as a matrix equation

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{bmatrix} 1 & 1/m \\ -k & 1 \end{bmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}.$$

The matrix defines a map from one value of the numerical solution to the next. We can consider a similar concept for the true solution of the problem, called the *flow*.

2.1.3 The Flow Map

Definition 2.1. The flow φ_t is a map of the system from an initial time to the time t. Specifically, we write $\varphi_t(\alpha) = x(t)$ given the initial condition $x(t = 0) = \alpha$. Hence the flow $\varphi_t : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is a map from the initial point of the system to the time t.

The Jacobian matrix $\varphi_t'(\alpha)$ is often called the sensitivity of the flow. It describes the relative change in the problem per change in initial conditions. We show the element-wise expression:

$$\begin{aligned} [\varphi_t'(\alpha)]_{ij} &= \left[\frac{\partial}{\partial \alpha}\varphi_t(\alpha)\right]_{ij} \\ &= \left[\frac{\partial}{\partial \alpha}x(t)\right]_{ij} \\ &= \frac{\partial x_i(t)}{\partial \alpha_j} \end{aligned}$$

for $i, j = 1, \ldots, 2d$, which is the Jacobian matrix.

Definition 2.2. A flow is symplectic if it satisfies $\varphi_t'(\alpha)^\top J \varphi_t'(\alpha) = J$, where J is the symplectic matrix.

Along with the flow of the system, we must consider the *numerical flow*, which is the map from x_n to x_{n+1} defined by a numerical method.

Definition 2.3. A numerical method defines a flow map Ψ_h by

$$x_{n+1} = \Psi_h(x_n).$$

The sensitivity of the numerical flow is affected similarly by a change in initial conditions. However, due to the discrete form of a numerical solution we will not give a general form. The sensitivity of the numerical flow map can however be written explicitly when applying a given method to a given problem.

2.1.4 Symplectic Flow

The flow of a Hamiltonian system is, by definition, symplectic. Recall the example of the simple harmonic oscillator. We can demonstrate symplecticity with the actual flow map. Note the general closed form solution:

$$\begin{aligned} q(t) &= A\cos\left(\sqrt{\frac{k}{m}}t\right) + B\sin\left(\sqrt{\frac{k}{m}}t\right) \\ p(t) &= B\sqrt{km}\cos\left(\sqrt{\frac{k}{m}}t\right) - A\sqrt{km}\sin\left(\sqrt{\frac{k}{m}}t\right) \end{aligned}$$

for arbitrary constants A and B. Impose initial conditions: suppose $q(t=0) = q_0,\ p(t=0) = p_0$. Define $\omega = \sqrt{k/m}$ for shorthand, and apply the initial conditions to find that the coefficients are $A = q_0,\ B = p_0/m\omega$ Hence the particular solution is

$$\begin{aligned} q(t) &= q_0\cos(\omega t) + \frac{p_0}{m\omega}\sin(\omega t) \\ p(t) &= p_0\cos(\omega t) - q_0 m\omega\sin(\omega t). \end{aligned}$$

We are in the position to evaluate the Jacobian entry-wise.

$$\varphi_t'\begin{pmatrix} q_0 \\ p_0 \end{pmatrix} = \begin{bmatrix} \frac{\partial q}{\partial q_0} & \frac{\partial q}{\partial p_0} \\ \frac{\partial p}{\partial q_0} & \frac{\partial p}{\partial p_0} \end{bmatrix} = \begin{bmatrix} \cos(\omega t) & \frac{1}{m\omega}\sin(\omega t) \\ -m\omega\sin(\omega t) & \cos(\omega t) \end{bmatrix},$$

and if we now plug this into the symplectic identity we get

$$\begin{aligned} \varphi_t'(x_0)^\top J \varphi_t'(x_0) &= \begin{bmatrix} \cos(\omega t) & -m\omega\sin(\omega t) \\ \frac{1}{m\omega}\sin(\omega t) & \cos(\omega t) \end{bmatrix}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} \cos(\omega t) & \frac{1}{m\omega}\sin(\omega t) \\ -m\omega\sin(\omega t) & \cos(\omega t) \end{bmatrix} \\ &= \begin{bmatrix} m\omega\sin(\omega t) & \cos(\omega t) \\ -\cos(\omega t) & \frac{1}{m\omega}\sin(\omega t) \end{bmatrix}\begin{bmatrix} \cos(\omega t) & \frac{1}{m\omega}\sin(\omega t) \\ -m\omega\sin(\omega t) & \cos(\omega t) \end{bmatrix} \\ &= \begin{bmatrix} m\omega\sin(\omega t)\cos(\omega t) - m\omega\sin(\omega t)\cos(\omega t) & \sin^2(\omega t) + \cos^2(\omega t) \\ -\cos^2(\omega t) - \sin^2(\omega t) & -\frac{1}{m\omega}\cos(\omega t)\sin(\omega t) + \frac{1}{m\omega}\cos(\omega t)\sin(\omega t) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = J. \end{aligned}$$

Hence the flow is symplectic.

2.1.5 A symplectic integrator

With symplectic integration, we are interested in numerical methods for which the numerical flow also attains symplecticity. One such example is the symplectic Euler method [13] given by

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} q_n \\ p_n \end{pmatrix} + hJ\nabla H(q_n, p_{n+1}).$$

For an arbitrary problem this method is implicit. However, many problems have a separate Hamiltonian that allows the iteration to be performed in two explicit steps. The simple harmonic oscillator has a Hamiltonian $H(q,p) = \frac{1}{2m}p^2 + \frac{1}{2}kq^2$ which can be written as $H(q,p) = V(q) + T(p)$, where $V(q)$ represents kinetic energy and $T(p)$ represents potential energy of the system [13, 9]. We expand out the method as

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} q_n \\ p_n \end{pmatrix} + hJ \begin{pmatrix} V'(q_n) \\ T'(p_{n+1}) \end{pmatrix} = \begin{pmatrix} q_n + \frac{h}{m}p_{n+1} \\ p_n - hkq_n \end{pmatrix}.$$

On inspection, we can perform this iteration separably: compute p_{n+1} using p_n and q_n, then use q_n and p_{n+1} to compute q_{n+1}. Important to note is that this is specifically the symplectic Euler-VT method, since we evaluate V' and then T', and the symplectic Euler-TV method is an alternative which computes in the opposite order analogously. This is of course assuming that the Hamiltonian is separable, which it may not be. For simplicity we will stick with the VT method for examples. We find an expression for the numerical flow:

$$\begin{aligned} \begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} &= \begin{pmatrix} q_n + \frac{h}{m}(p_n - h\omega q_n) \\ p_n - h\omega q_n \end{pmatrix} \\ &= \begin{bmatrix} 1 - \frac{h^2k}{m} & \frac{h}{m} \\ -hk & 1 \end{bmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix} \equiv \Phi_h \begin{pmatrix} q_n \\ p_n \end{pmatrix} \end{aligned}$$

and just like before, test the symplectic identity

$$\Phi^\top J \Phi = \begin{bmatrix} \left(1 - \frac{h^2k}{m}\right)(-hk) + \left(\frac{h^2k}{m} - 1\right)(-hk) & 1 - \frac{h^2k}{m} + \frac{h^2k}{m} \\ -\frac{h^2k}{m} + \frac{h^2k}{m} - 1 & \frac{h}{m} - \frac{h}{m} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = J.$$

Hence this method is symplectic for this example.

2.2 Further Hamiltonian Dynamics

2.2.1 Derivatives of Flow Maps

The flow map $\varphi_t : x_0 \to x(t)$ gives us powerful insights into the behaviour of Hamiltonian integration. Before giving a result on the symplecticity of Hamiltonian systems, we need some results on flow maps. We will show expressions for the time derivatives of $\varphi_t(\alpha)$ and $\varphi_t'(\alpha)$. Starting with the flow, its time derivative is

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\left(\varphi_t(x_0)\right) &= \frac{\mathrm{d}}{\mathrm{d}t}x(t) \\ &= \dot{x}(t) \\ &= J\nabla H(x(t)) \\ &= J\nabla H\left(\varphi_t(x_0)\right). \end{aligned}$$

We have a similar form for the time derivative of the sensitivity:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left(\varphi_t'(x_0)\right) &= \frac{\partial}{\partial x_0}\frac{\mathrm{d}}{\mathrm{d}t}\varphi_t(x_0)\\ &= \frac{\partial}{\partial x_0}\frac{\mathrm{d}}{\mathrm{d}t}x(t)\\ &= \frac{\partial}{\partial x_0}\dot{x}(t)\\ &= \frac{\partial}{\partial x_0}J\nabla H(x(t))\\ &= J\nabla^2 H(x(t))\left(\frac{\partial}{\partial x_0}x(t)\right)\\ &= J\nabla^2 H(\varphi_t(x_0))\varphi_t'(x_0).\end{aligned}$$

2.2.2 Symplectic Flow of a Hamiltonian System

Earlier, we looked at the symplecticity of the flow of a particular Hamiltonian system. We will now generalise this result to any Hamiltonian system. Symplecticity of the flow is equivalent to the system itself being Hamiltonian.

Theorem 2.4. *A dynamical system is Hamiltonian if and only if its flow is symplectic.*

Proof. ($\Rightarrow$) Consider the flow of a Hamiltonian system at time $t = 0$. By definition, $\varphi_t(x_0) = x(t)$ given $x(0) = x_0$, therefore $\varphi_0(x_0) = x_0$. Hence the sensitivity at $t = 0$ is $\varphi_0'(x_0) = I$ the identity matrix. The symplectic identity is satisfied trivially: $(\varphi_0'(x_0))^\top J\varphi_0'(x_0) = IJI = J$.

Now, instead of finding an expression of the symplectic identity at time t, we show that this quantity is unchanging in time. By differentiating, the expression distributes by the product rule:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\varphi_t'(x_0)^\top J\varphi_t'(x_0)\right) = \left(\frac{\mathrm{d}}{\mathrm{d}t}\varphi_t'(x_0)^\top\right)J\varphi_t'(x_0) + \varphi_t'(x_0)^\top J\left(\frac{\mathrm{d}}{\mathrm{d}t}\varphi_t'(x_0)\right).$$

We can find an expression for the derivative term:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\varphi_t'(x_0) &= \frac{\mathrm{d}}{\mathrm{d}t}J\nabla H(\varphi_t(x_0))\\ &= J\nabla^2 H(\varphi_t(x_0))\varphi_t'(x_0).\end{aligned}$$

Now plug this back in, becoming

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left(\varphi_t'(x_0)^\top J\varphi_t'(x_0)\right) &= \left(J\nabla^2 H(\varphi_t)\varphi_t'\right)^\top J\varphi_t' + \varphi_t' J\left(J\nabla^2 H(\varphi_t)\varphi_t'\right)\\ &= (\varphi_t')^\top\nabla^2 H(\varphi_t)^\top J^\top J\varphi_t' + (\varphi_t')^\top J^2\nabla^2 H(\varphi_t)\varphi_t'(x_0)\\ &= (\varphi_t')^\top\nabla^2 H(\varphi_t)^\top\varphi_t' - (\varphi_t')^\top\nabla^2 H(\varphi_t)\varphi_t'\end{aligned}$$

since $J^\top J = I$ and $J^2 = -I$. Under the assumption that the Hessian matrix $\nabla^2 H(\varphi_t)$ is symmetric, this expression evaluates to zero and hence the symplectic identity is satisfied for all t and we are done.

($\Leftarrow$) Assuming that the sensitivity satisfies the symplectic identity, we want to show that the system is Hamiltonian. For a general system, we have $\dot{x} = f(x)$. We want to show that we can

write $f(x) = J\nabla H(x)$ for some function x, in order for the system to be Hamiltonian. The flow is symplectic, so it satisfies

$$\varphi_t'(x_0)^\top J\varphi_t'(x_0) = J.$$

Differentiating this expression gives, similar to earlier,

$$\varphi_t'(x_0)^\top \frac{\partial f}{\partial x_0}(\varphi_t(x_0))^\top J\varphi_t'(x_0) + \varphi_t'(x_0)^\top J\frac{\partial f}{\partial x_0}(\varphi_t(x_0))\varphi_t'(x_0) = 0.$$

By taking factors on the left and right, this is

$$\varphi_t'(x_0)^\top \left[\frac{\partial f}{\partial x_0}(\varphi_t(x_0))^\top J + J\frac{\partial f}{\partial x_0}(\varphi_t(x_0))\right]\varphi_t'(x_0) = 0.$$

This must be true for all t, so if we set $t = 0$ the sensitivity matrix appearing on both the right and left is the identity. Because $J = -J^\top$, we have that

$$\left[J\frac{\partial f}{\partial x_0}\right] = \left[J\frac{\partial f}{\partial x_0}\right]^\top$$

i.e. the matrix is symmetric. The required result is given in [13], namely that $f(x)$ can be written in the form $J\nabla H(x)$. □

We now properly understand the link between Hamiltonian mechanics and symplectic integration. The formal definition for a symplectic integrator is that the method maintains the form $\mathrm{d}q_i \wedge \mathrm{d}p_i$ with $i = 1, \ldots, d$ for an d-dimensional problem [13]. The form is an infinitesimal area generated by the infinitesimals in q, p. For a one-dimensional problem, we can produce the phase portrait by plotting p against q. A single point in the phase portrait represents the state of the dynamical system at a fixed point in time. The flow and the numerical flow are maps between points in the phase portrait. If we think of the phase portrait as a solution space for a one-dimensional dynamical system, we can say that symplectic methods preserve the area of the solution space. This is the paradigm of symplectic integration: by maintaining area of the phase space under mapping of the numerical flow, we maintain qualitative behaviour of the ODE over long timespans. Furthermore, the conservation of area is equivalent to a symplectic method maintaining a first integral of the system, which is the Hamiltonian H. We will however make it clear that a symplectic integrator does not exactly preserve the Hamiltonian of the problem, this being something we will explore in more detail.

In order to apply a symplectic integration method to a problem, we need an expression for the Hamiltonian of that problem, and we need a numerical method for which the symplecticity of the flow is preserved.

2.2.3 The Simple Pendulum

The simple pendulum is the one-dimensional system defined by

$$ml\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -mg\sin(\theta)$$

For this problem, we define $q = \theta$, $p = \dot{\theta}$ and the Hamiltonian [9] as

$$H(q, p) = \frac{1}{2}p^2 + k^2(1 - \cos(q))$$

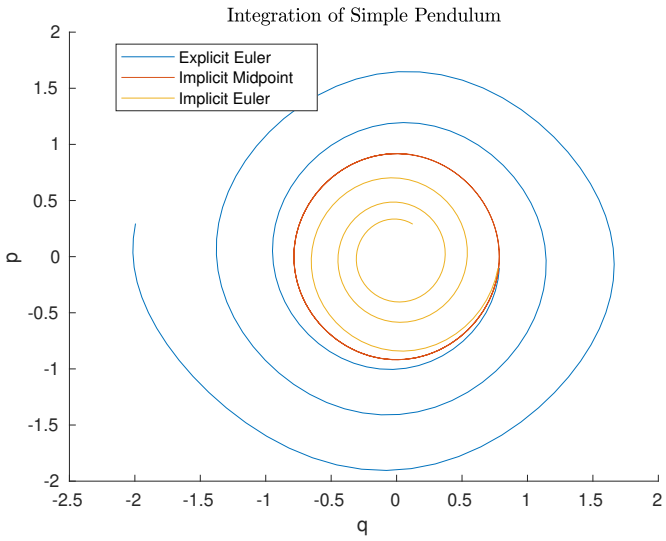

Figure 2.1: Integration of the simple pendulum as a Hamiltonian mechanics problem, plotted as a phase portrait. The explicit Euler method gradually diverges, a consequence of it lacking A-stability. The implicit Euler method converges to zero. Using implicit midpoint, we stay on the path followed by the pendulum, since this method is symplectic.

where $k^2 = g/l$. This Hamiltonian can be obtained by integrating the system, and is therefore a conserved quantity. We will look at a selection of numerical methods applied to this problem. Recall Euler's method, which takes the form $x_{n+1} = x_n + hJ\nabla H(x_n)$ for a Hamiltonian system. This expands to:

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix}^E = \begin{pmatrix} q_n \\ p_n \end{pmatrix} + h \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} k^2 \sin(q_n) \\ p_n \end{pmatrix} = \begin{pmatrix} q_n + hp_n \\ p_n - hk^2 \sin(q_n) \end{pmatrix}.$$

The Implicit Midpoint method gives us

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix}^M = \begin{pmatrix} q_n \\ p_n \end{pmatrix} + h \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} k^2 \sin\left(\dfrac{q_n + q_{n+1}}{2}\right) \\ \dfrac{p_n + p_{n+1}}{2} \end{pmatrix} = \begin{pmatrix} q_n + h\left(\dfrac{p_n + p_{n+1}}{2}\right) \\ p_n - hk^2 \sin\left(\frac{q_n + q_{n+1}}{2}\right) \end{pmatrix}.$$

Finally, applying the Symplectic Euler-VT method yields

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix}^S = \begin{pmatrix} q_n \\ p_n \end{pmatrix} + h \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} k^2 \sin(q_n) \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} q_n + hp_{n+1} \\ p_n - hk^2 \sin(q_n) \end{pmatrix} \tag{2.4}$$

See Figure 2.1, where we have applied three methods to the simple pendulum problem. Only when applying the implicit midpoint method do the oscillations remain on a loop in the phase portrait. The Symplectic Euler method is not shown, since it is identical to the result obtained from the implicit midpoint method. The implicit midpoint method is a symplectic method, and probably the simplest to demonstrate.

Symplectic methods preserve the oscillation of the pendulum. This means the pendulum oscillates consistently, without any drift. In the explicit and implicit Euler methods, we observe a divergence or convergence of the bounds of oscillation respectively.

It is important to note at the moment that the only examples we have looked at have closed form solutions. Our results so far only serve for understanding the definition of symplecticity. We now look at some stronger results about numerical methods.

2.2.4 The Adjoint Flow Map

We may now want to look at forming higher order symplectic methods. In order to do so, we need to consider generalised properties of flow maps which define these methods. One such property is the *adjoint* map.

Definition 2.5. Given a method defined by a numerical flow Ψ_h, the adjoint Ψ_h^* is the method that satisfies

$$\Psi_{-h}^* = \Psi_h^{-1}.$$

In words, stepping backward with the adjoint method is equivalent to stepping forward with the inverse. Adjoint methods are very useful. Given an arbitrary method Ψ_h, the method $\Psi_{h/2} \circ \Psi_{h/2}^*$ is symmetric. A symmetric method is a method that satisfies $\Psi_h^{-1} = \Psi_{-h}$. This means if we integrate forwards in time by a particular step, and then integrate back, we return to the original value since we are equivalently applying the inverse. Time symmetric methods are geometric integrators themselves, preserving time symmetry, however this is not something we will explore.

We can show that the implicit Euler method is the adjoint of explicit Euler [29]. First, let Ψ_h^I denote the implicit method and let Ψ_h^E denote the explicit method. We want to show that $\Psi_h^E = (\Psi_{-h}^I)^{-1}$. This is equivalent to $\Psi_{-h}^I \circ \Psi_h^E(x) = x$. If we expand this composition, we obtain

$$\begin{aligned}
\Phi_{-h}^I\left(\Phi_j^E(x)\right) &= \Phi_{-h}^I\left(x + hf(x)\right) \\
&= x + hf(x) - hf\left(\Phi_{-h}^I\left(x + hf(x)\right)\right) \\
&= x + hf(x) - hf\left(\Phi_{-h}^I\left(\Phi_h^E(x)\right)\right),
\end{aligned}$$

and $\Phi_{-h}^I(\Phi_h^E(x)) = x$ solves this equation. Hence $(\Phi_{-h}^I)^{-1} = \Phi_h^E$. Note that the explicit and implicit Euler methods cannot be symmetric because they are adjoint.

There are several properties of symplectic methods which may be of interest now. We have not shown that (1) the adjoint of a symplectic method is symplectic, and (2) the composition of symplectic methods is symplectic. These are relatively simple properties which can be proven from the defintions, but it will help our understanding to cover these.

Proposition 2.6. The adjoint of a symplectic method is symplectic.

Proof. First of all, the definition of the adjoint method states $\Phi_h^* = \Phi_{-h}^{-1}$. By algebraic manipulation,

$$\begin{aligned}
&\Phi_h^\top J \Phi_h = J \\
\Rightarrow\ &\Phi_{-h}^\top J \Phi_{-h} = J \\
\Rightarrow\ &J = (\Phi_{-h}^{-1})^\top J (\Phi_{-h}^{-1}) = (\Phi_h^*)^\top J (\Phi_h^*)
\end{aligned}$$

and so clearly the adjoint is symplectic. □

This result will be used in tandem with the composition of maps.

Proposition 2.7. If Φ_h and Ψ_h are two symplectic maps, then their composition $\Phi_h \circ \Psi_h$ is symplectic.

Proof. This is done similarly. We have

$$\begin{aligned}(\Phi_h\Psi_h)^\top J(\Phi_h\Psi_h) &= \Psi_h^\top \Phi_h^\top J \Phi_h \Psi_h \\ &= \Psi_h^\top J \Psi_h \\ &= J.\end{aligned}$$

Therefore the composition of symplectic maps is symplectic. □

We can now introduce a new method using these results.

2.2.5 The Störmer-Verlet Method

Earlier, we looked at the Symplectic Euler method, which is a modification of the Euler methods for Hamiltonian integration. If the Hamiltonian is separable such that $H(q,p) = V(q) + T(p)$, then the Symplectic Euler method is explicit. This is because in the Symplectic Euler-VT step, we use p_n and q_n to compute p_{n+1}, then use p_{n+1} and q_n to compute q_{n+1}. This is exactly the method we applied in our earlier example introducing the Symplectic Euler method. However, the Symplectic Euler method is generally implicit because a given Hamiltonian for a problem may not be separable. Denote the Symplectic Euler step by Ψ_h. The *Störmer-Verlet* method [13, 9] is defined by the step $\Psi^*_{h/2} \circ \Psi_{h/2}$. Formally, this is

$$\begin{aligned}p_{n+\frac{1}{2}} &= p_n - \frac{h}{2}\nabla_q H\left(q_n, p_{n+\frac{1}{2}}\right) \\ q_{n+1} &= q_n + \frac{h}{2}\left(\nabla_p H\left(q_n, p_{n+\frac{1}{2}}\right) + \nabla_p H\left(q_{n+1}, p_{n+\frac{1}{2}}\right)\right) \\ p_{n+1} &= p_{n+\frac{1}{2}} - \frac{h}{2}\nabla_q H\left(q_{n+1}, p_{n+\frac{1}{2}}\right).\end{aligned}$$

The Störmer-Verlet method is time-symmetric and second order accurate [9]. We start with a step of the symplectic Euler method of length $h/2$, followed by its adjoint in which we step in (q,p) in the opposite order. This explains the structure where we have a step in q of length h in the middle of two steps in p of length $h/2$.

We have defined the Störmer-Verlet method by the map $\Psi^*_{h/2} \circ \Psi_{h/2}$, but we may also define a method as $\Psi_{h/2} \circ \Psi^*_{h/2}$, which is an alternative Störmer-Verlet scheme which is also time-symmetric and second order.

The composition of a method with its adjoint leads to a method which is necessarily of even order [29, 14, 9].

2.2.6 Mechanics

Before the next example, it helps to define context around the mechanics of Hamiltonian system, In many cases, a Hamiltonian system has a Hamiltonian H that we can express as $H(q,p) = T(p)+V(q)$. The Hamiltonian is a conserved quantity of the system, such as the total energy being exchanged. A *Lagrangian* for a system can be defined as $L(q,p) = T(p) - V(q)$, a difference in the energies. The natural conjugate momentum is defined by

$$p_k := \frac{\partial L}{\partial \dot{q}_k} \tag{2.5}$$

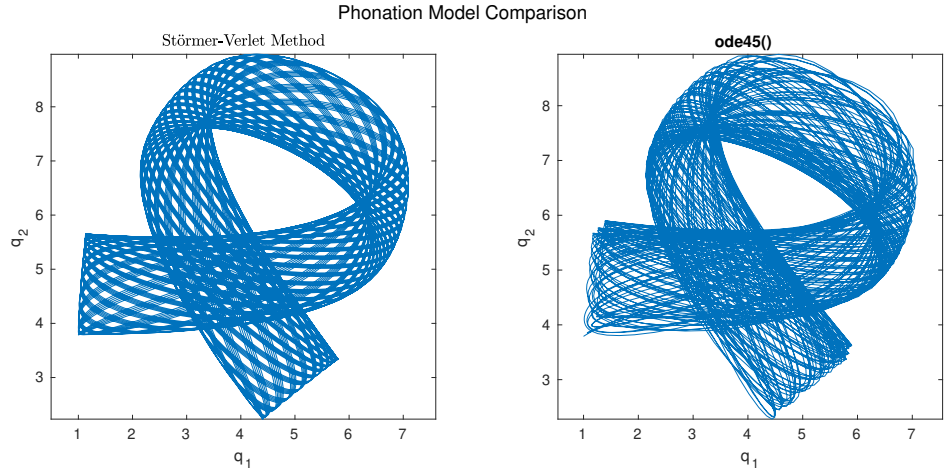

Figure 2.2: Integration of the two mass model for phonation for $t \in [0, 1000]$. Figures show the oscillation bounds, plotting q_2 against q_1. In the left figure, we observe that a symplectic method retains the bounds of oscillation, whereas using `ode45()` the explicit Runge-Kutta method is not as well behaved and the bounds are not as clear. Parameters used are $\alpha = 1, \lambda = 0.5, \beta = 3, \omega = 0.3$. Initial positions $q_1 = 1$ and $q_2 = 3.79$ with momentum zero.

for $k = 1, \ldots, n$. If we have a problem formulated generally, position is usually already defined. Natural conjugate momentum means we can obtain an expression for momentum which aligns with the properties of Hamiltonian dynamics. It often helps when defining and solving our own problems for the variables to be defined in this way.

2.2.7 A Model for Phonation

This example is based on the two mass model from "Models for Phonation"[1]. A vocal cord is modelled by two stiffness-coupled masses, and their one-dimensional displacements are given by u and v. The governing equation for motion, from Newton's law, is

$$\frac{\mathrm{d}^2 u}{\mathrm{d}t^2} = 1 - u + \beta\left(1 - \frac{1}{u^2}\right) + \omega(v - u)$$
$$\alpha\frac{\mathrm{d}^2 v}{\mathrm{d}t^2} = \lambda(1 - v) + \beta\left(1 - \frac{1}{v^2}\right) + \omega(u - v)$$

We first express this as a Hamiltonian problem. Denote $q_1 = u$, $q_2 = v$. We define momentum $p_1 = \dot{q}_1$, $p_2 = \alpha\dot{q}_2$. A Hamiltonian, obtained by integrating the system, is

$$H = \frac{1}{2}\left(p_1^2 + \frac{1}{\alpha}p_2^2\right) + \frac{\omega}{2}(q_1 - q_2)^2 - F(q_1) - G(q_2)$$

[1] `https://willwoolf.github.io/phon.pdf`

where

$$F(q_1) = q_1 - \frac{1}{2}q_1^2 + \beta\left(q_1 + \frac{1}{q_1}\right)$$
$$G(q_2) = \lambda\left(q_2 - \frac{1}{2}q_2^2\right) + \beta\left(q_2 + \frac{1}{q_2}\right).$$

Therefore, in Hamiltonian variables the coupled ODEs can be expressed as

$$\dot{q}_1 = p_1 \qquad \dot{q}_2 = \frac{1}{\alpha}p_2$$
$$\dot{p}_1 = F'(q_1) - \omega(q_1 - q_2) \qquad \dot{p}_2 = G'(q_2) - \omega(q_2 - q_1)$$

stating the problem in Hamiltonian form. Since the Hamiltonian is a conserved quantity, we can use it to evaluate the behaviour of the integration method. We can apply the Hamiltonian function to our numerical solutions, to verify that the quantity is in fact conserved by the numerical method. A symplectic method does not preserve the Hamiltonian, but it preserves a particular Hamiltonian. The closeness of this Hamiltonian can be shown via backward error analysis. A regular integration method, such as an explicit Runge-Kutta scheme, will fail to preserve the Hamiltonian and the constant will diverge. If the value changes by a significant amount, we can observe loss of qualitative behaviour. Observe how, in Figure 2.2, we have shown an immediate comparison of the clarity of results from using a symplectic integrator, versus using the efficient but not as stable explicit Runge-Kutta methods applied by MATLAB's `ode45()` integrator.

2.3 Generalised Symplectic Methods

2.3.1 Conjugacy

Before looking at more symplectic methods, we will cover a brief result on conjugacy of methods. Consider the implicit midpoint and trapezium methods. Implicit midpoint is the method

$$\Phi_h^M(x_n) = x_{n+1} = x_n + hf\left(\frac{x_n + x_{n+1}}{2}\right).$$

The trapezium method is similar:

$$\Phi_j^T(x_n) = x_{n+1} = x_n + h\left(\frac{f(x_n) + f(x_{n+1})}{2}\right)$$

The implicit midpoint method is symplectic, but the trapezium method is not. Interestingly, the trapezium method has excellent behaviour in preserving phase-space volume, despite not being a symplectic method. It can be shown that these are conjugate methods [29], meaning that they exhibit similar long-term behaviour. Two methods Ψ_h, Φ_h are conjugate if there exists a map χ such that $\Phi_h = \chi^{-1}\Psi_h\chi$. Consider applying a method Φ_h N times. Conjugacy shows that

$$\begin{aligned}(\Phi_h)^N &= \left(\chi^{-1}\Psi_h\chi\right)^N \\ &= \underbrace{\left(\chi^{-1}\Psi_h\chi\right)\left(\chi^{-1}\Psi_h\chi\right)\ldots\left(\chi^{-1}\Psi_h\chi\right)}_{N} \\ &= \chi^{-1}(\Psi_h)^N\chi.\end{aligned}$$

Therefore, conjugate methods remain separated by the conjugacy map χ for an arbitrary number of iterations.

2.3.2 Runge-Kutta Methods

These are methods of the form

$$x_{n+1} = x_n + h\left(\sum_{i=1}^{s} b_i k_i\right)$$

as defined in the introduction, equivalently defined by the Butcher tableau

$$\begin{array}{c|c} c & A \\ \hline & b^\top. \end{array}$$

We will first consider A-stability.

Consider the linear test problem $\dot{x} = \lambda x$. We have already explored stability for the basic Euler methods. We will look at a general explicit 3-stage method.

Example 2.8. First, find expressions for the k_i:

$$\begin{aligned} k_1 &= \lambda x_n, \\ k_2 &= \lambda\left(x_n + h a_{21} k_1\right) \\ &= \left(\lambda + h a_{21}\lambda^2\right) x_n, \\ k_3 &= \lambda\left(x_n + h a_{31} k_1 + h a_{32} k_2\right) \\ &= \lambda\left(x_n + h a_{31}\lambda x_n + h a_{32}\left(\lambda + h a_{21}\lambda^2\right) x_n\right) \\ &= \left(\lambda + h a_{31}\lambda^2 + h a_{32}\lambda^2 + h^2 a_{32} a_{21}\lambda^3\right) x_n. \end{aligned}$$

therefore

$$\begin{aligned} x_{n+1} &= x_n + h\left(b_1 k_1 + b_2 k_2 + b_3 k_3\right) \\ &= x_n + h b_1\lambda x_n + h b_2\left(\lambda + h a_{21}\lambda^2\right) x_n + h b_3\left(\lambda + h a_{31}\lambda^2 + h a_{32}\lambda^2 + h^2 a_{32} a_{21}\lambda^3\right) x_n \\ &= \left(1 + \left(b_1 + b_2 + b_3\right) h\lambda + \left(b_2 a_{21} + b_3(a_{31} + a_{32})\right) h^2\lambda^2 + \left(b_3 a_{32} a_{21}\right) h^3\lambda^3\right) x_n. \end{aligned}$$

This coefficient term is a polynomial on $h\lambda$, which we denote by $r(h\lambda)$. To ensure A-stability, we require that $|r(h\lambda)| < 1$ for $h\lambda$ in the left half of the complex plane. For this example of an explicit method, it cannot be A-stable since the stability function is a polynomial. Any non-constant polynomial $p(y)$ will diverge to $\pm\infty$ as $|y| \to \infty$, hence the bound will not be attained.

We will now consider the A-stability of Runge-Kutta methods in general.

2.3.3 A-Stable Runge-Kutta Methods

We start by considering the A-stability of Runge-Kutta methods. We know that if these exist, they cannot be explicit. This analysis borrows from [21]. We will consider the linear test problem $\dot{x} = \lambda x$ in the scalar case for simplicity. Recall that the formulation of the k_i is given by

$$k_i = x_n + h\lambda\sum_{j=1}^{s} a_{ij} k_j$$

for $i = 1, \ldots, s$. Let k be the vector of k_i terms, in which case we have

$$k = e x_n + h\lambda A k$$

where e is the vector of ones. This expression rearranges to

$$k = [I - h\lambda A]^{-1} e x_n$$

assuming the inverse exists. Therefore the method can be expressed as

$$\begin{aligned} x_{n+1} &= x_n + h\lambda \sum_{i=1}^{s} b_i k_i \\ &= x_n + h\lambda b^\top k \\ &= x_n + h\lambda b^\top [I - h\lambda A]^{-1} e x_n \\ &= \left[1 + h\lambda b^\top [I - h\lambda A]^{-1} e\right] x_n. \end{aligned}$$

We have a direct operation which describes the RK method, which we define for simplicity as

$$r(z) = 1 + zb^\top [I - zA]^{-1} e \tag{2.6}$$

such that the method takes the form $x_n = [r(h\lambda)]^n x_0$. We can assume $x_0 = 1$ without loss of generality so the method is

$$x_n = [r(h\lambda)]^n$$

If we can classify that $r(z) < 1$ for any z with negative real part, then the method is A-stable.

In order to characterise this expression of r, involving the inverse of a matrix, we need to define the *adjugate* matrix in order to give a particular expression for a matrix inverse.

Definition 2.9 (The Adjugate Matrix [18])**.** For an invertible matrix A we can write its inverse as

$$A^{-1} = \frac{\text{adj}(A)}{\det(A)}$$

where $\det(A)$ is the determinant of A and $\text{adj}(A)$ is the transpose of the cofactor matrix C, so

$$\text{adj}(A) = C^\top$$

where

$$C_{ij} = (-1)^{i+j} \det \bar{A}_{ij}$$

where $\bar{A}_{ij}$ is the submatrix obtained by removing the i-th row and j-th column from A.

We can show that this expression for the inverse holds for a 2×2 matrix.

Example 2.10. Consider the matrix given by

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

The determinant is given by $\det(A) = ad - bc$. For each entry in the cofactor matrix, we remove row i and column j to compute a determinant of a 1×1 submatrix. Applying the formula from the definition, we end up with

$$C = \begin{bmatrix} d & -c \\ -b & a \end{bmatrix}$$

Therefore,

$$A^{-1} = \frac{\text{adj}(A)}{\det(A)} = \frac{1}{\det(A)} C^\top = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

which should be a familiar expression for a 2×2 matrix inverse.

We use this relation to characterise the matrix inverse term which appears in $r(z)$, see Equation 2.6, which allows us to write the stability function clearly as a rational function.

Lemma 2.11 (Iserles (2009) [21])**.** The stability function of an s-stage Runge-Kutta method can be written as a rational function where the numerator and denominator are degree s polynomials.

Proof. The term $[I - hA]^{-1}$ appearing in Equation 2.6 can be written as

$$[I - zA]^{-1} = \frac{\text{adj}(I - zA)}{\det(I - zA)}.$$

For an s-stage Runge-Kutta method, A has size s. The determinant of $I - zA$ is a polynomial on z of degree at most s. Similarly, every entry in the adjugate is the determinant of an $(s-1) \times (s-1)$ submatrix, hence is a polynomial of degree at most $s - 1$. We can re-write Equation 2.6 as

$$r(z) = 1 + \frac{1}{\det(I - zA)} \left(zb^\top \text{adj}(I - zA)e \right)$$

The 1 doesn't change the degree of the function, and both the $\det(I - zA)$ and $zb^\top \left[\text{adj}(I - zA)\right] e$ terms are degree s polynomials. Therefore the stability function for a Runge-Kutta method can be written as a rational function on $h\lambda$, being a quotient of two polynomials of degree s.

The inclusion of the determinant term simplifies this expression when we consider an explicit method. In this case, A is strictly lower triangular, and therefore $I - zA$ is a lower triangular matrix with ones on the diagonal. Therefore $\det(I - zA) = 1$ and the stability function is a degree s polynomial. □

The stability function can also be written in the simpler form

$$r(z) = \frac{\det(I - zA + zeb^\top)}{\det(I - zA)} \tag{2.7}$$

given in [21]. It remains to show that there are Runge-Kutta methods for which the expression $r(z) < 1$ is satisfied for z with negative real part. Iserles shows that if a stability function is a particular kind of approximation to the exponential e^z, then the corresponding method will be A-stable. The function e^z satisfies the bound we are looking for in a stability function.

Lemma 2.12 (Iserles (2009) [21])**.** Suppose $r(z)$ is the stability function for a Runge-Kutta method of order p. Then

$$r(z) = \mathrm{e}^z + \mathcal{O}(z^{p+1})$$

Proof. The stability function came from the fact that the method can be expressed as $x_{n+1} = r(h\lambda)x_n$. Similarly, $x_{n+1} = \mathrm{e}^{h\lambda} x_n$ is the analytical solution to the linear test problem at $x(t_{n+1})$ given that $x(t_n) = x_n$. Since the method is accurate to order p, we must have that $\mathrm{e}^{h\lambda} - r(h\lambda) = \mathcal{O}((h\lambda)^{p+1})$ by definition. □

In order to preserve the condition of A-stability, we use the Padé approximation.

Definition 2.13 (The Padé Approximant [19])**.** The Padé $[n, m]$ approximant to a function $f(z)$ is the unique rational function $p_{n,m}(z)/q_{n,m}(z)$ that satisfies

$$f(z) - \frac{p_{n,m}(z)}{q_{n,m}(z)} = \mathcal{O}(z^{p+q+1}).$$

where $p_{n,m}$ is a polynomial of degree n and $q_{n,m}$ of degree m.

Padé approximants are able to remain bounded as $|z| \to \infty$ [30]. A polynomial approximation will always diverge for large z and cannot have any discontinuity. This motivates the use of Padé approximations in this context as stability functions for Runge-Kutta methods. We now want to find Runge-Kutta methods for which the stability functions are Padé approximations to the exponential.

Remark 2.14. The $[1, 1]$ Padé approximant to the exponential is given by

$$r_{1,1}(z) = \frac{p_{1,1}(z)}{q_{1,1}(z)} = \frac{1 + \frac{z}{2}}{1 - \frac{z}{2}}$$

and the $[2, 2]$ approximant is

$$r_{2,2}(z) = \frac{p_{2,2}(z)}{q_{2,2}(z)} = \frac{1 + \frac{z}{2} + \frac{z^2}{12}}{1 - \frac{z}{2} + \frac{z^2}{12}}$$

If p has degree higher than q, then the approximation will diverge for $|z| \to \infty$ so these approximations are unusable. We will consider primarily the diagonal ($n = m$) Padé approximants. We call an approximation A-acceptable if it obeys the bound required for the method to be A-stable [21]. Diagonal Padé approximants to the exponential are A-acceptable [30].

We will now introduce the Gauss-Legendre Runge-Kutta (GLRK) methods in general. Recall the implicit midpoint rule, which has Butcher tableau

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

This is a second order method but has $s = 1$, which is an optimal order for the number of steps. This is the general aim for the Gauss-Legendre Runge-Kutta methods, achieving order $2s$ for s steps. The fourth order GLRK method on two stages is given by the tableau

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

These methods are implicit and can be complicated and expensive to implement, but they have excellent behaviour. In this context, we want to know about stability. The following is a simplification of a theorem from [21] using knowledge of A-acceptability [30].

Theorem 2.15. *The Gauss-Legendre Runge-Kutta methods are A-stable.*

Proof. Assume we apply a GLRK method on s stages to the linear test problem $\dot{x} = \lambda x$. By Lemma 2.11, the stability function of this method can be written as a rational function where the numerator and denominator are degree s polynomials. Applying Lemma 2.12, this rational function is accurate to the exponential to degree $2s$ since this is the order of the method. Since the Padé approximation to the exponential is unique, this rational function is the $[s, s]$ Padé approximant and hence the method is A-stable. □

2.3.4 Symplectic Runge-Kutta Methods

We have shown that all A-stable Runge-Kutta methods are necessarily implicit, with examples of particular A-stable methods. We will now examine the theory behind symplectic Runge-Kutta methods. Again, consider a Runge-Kutta method defined by the stability function $r(z)$. We first cover a result on the stability functions of symplectic Runge-Kutta methods.

Proposition 2.16 (Hairer, Leone [12])**.** If a stability function $r(z)$ method satisfies

$$r(z)r(-z) = 1$$

for all $z \in \mathbb{C}$ then there exists a symplectic Runge-Kutta method which has stability function $r(z)$

Proof. We will not cover the proof, which is given in the paper [12] and uses results outside of our analysis. □

Hairer and Leone give an example of how a method of this kind relates to the requirement of symplecticity.

Example 2.17 (The Linear Oscillator [12])**.** Consider the system on complex u given by

$$\dot{u} = i\omega u.$$

with $\omega > 0$ and the initial condition $u(t = 0) = 1$. This is the linear oscillator, as the trajectory is the circle of radius 1. If we apply a Runge-Kutta method to this problem, we obtain the numerical solution $u_n = [r(i\omega h)]^n$. Therefore, in order for this RK method to be symplectic we require $|r(i\omega h)| = 1$ given $h > 0$, since the absolute value of u has to remain constant. We can let $z = \omega h$ and require $|r(iz)| = 1$. However, this is equivalent to requiring $r(z)r(-z) = 1$ which is our condition.

This is a very powerful result on symplectic RK methods. We can guarantee the existence of a symplectic RK method if we have any stability function that satisfies our proposition. Note that this provides a distinction between symplectic and A-stable RK methods, since a stability function may induce a symplectic RK method which is not A-stable.

In order to explore symplectic methods we will again consider diagonal Padé approximants to the exponential. It turns out that all diagonal Padé approximants to the exponential are of the form

$$r_{n,n}(z) = \frac{p(z)}{p(-z)}$$

for a polynomial $p(z)$ of degree n, consequence of [19]. Explicitly, the form is

$$p_{n,m}(z) = \sum_{j=0}^{n} \frac{(n+m-j)!n!}{(n+m)!j!(n-j)!} z^j$$
$$q_{n,m}(z) = \sum_{j=0}^{m} \frac{(n+m-j)!m!}{(n+m)!j!(m-j)!} (-z)^j.$$

If we take $n = m$ for a diagonal approximation then the requirement is clear. Now, if we consider $r(z)$ to be a diagonal Padé approximation to the exponential then

$$r(z)r(-z) = \frac{p(z)}{p(-z)} \frac{p(-z)}{p(z)} = 1$$

for a polynomial $p(z)$, and hence the Runge-Kutta method with stability function $r(z)$ is symplectic. However, we already know from the previous discussion on A-stable methods that these are the GLRK methods. Hence the GLRK methods are symplectic.

There is another result on the properties of symplectic RK methods that we can consider.

Theorem 2.18 (Hairer, Lubich, Wanner (2006) [13])**.** *If a Runge-Kutta method satisfies*

$$b_i a_{ij} + b_j a_{ji} - b_i b_j = 0 \tag{2.8}$$

for all $i, j = 1, \ldots, s$ then it is symplectic.

Proof. This proof is in two parts. First, we want to show that any Runge-Kutta method must satisfy this rule in order to preserve quadratic invariants of the system. Then, we show that any Runge-Kutta method which preserves these invariants is in fact symplectic.

Part I: Start by considering the recurrence $x_1 = x_0 + h\sum_{j=1}^{s} b_i k_i$ and let C be an arbitrary matrix, in which case we can express a quadratic as

$$x_1^\top C x_1 = x_0^\top C x_0 + h\sum_{i=1}^{s} b_i k_i^\top C x_0 + h\sum_{j=1}^{s} b_j x_0^\top C k_j + h^2 \sum_{i=1}^{s}\sum_{j=1}^{s} b_i b_j k_i^\top C k_j \tag{2.9}$$

by expanding. We can make a simplification by writing $k_i = f(X_i) = f\left(x_0 + h\sum_{j=1}^{s} a_{ij} k_j\right)$, which is the definition of the k_i evaluations. The f is the function which defines the dynamical system. We work out the expansion:

$$\begin{aligned}
x_1^\top C x_1 &= x_0^\top C x_0 + h\sum_{i=1}^{s} b_i f(X_i)^\top C X_i - h\sum_{i=1}^{s} b_i f(X_i)^\top C \left(h\sum_{j=1}^{s} a_{ij} k_j\right) \\
&\quad + h\sum_{j=1}^{s} b_j X_j^\top C f(X_j) - h\sum_{j=1}^{s} b_j \left(h\sum_{i=1}^{s} a_{ji} k_i\right) C f(X_j) \\
&\quad + h^2 \sum_{i=1}^{s}\sum_{j-1}^{s} b_i b_j k_i^\top C k_j \\
&= x_0^\top C x_0 + h\sum_{i=1}^{s} b_i \left(\frac{\mathrm{d}}{\mathrm{d}t} X_i^\top C X_i\right) + h^2 \sum_{i=1}^{s}\sum_{j=1}^{s} (b_i b_j - b_i a_{ij} - b_j a_{ji}) k_i^\top C k_j
\end{aligned}$$

Note the application of product rule differentiation, namely that

$$X^\top C f(X) + f(X)^\top C X = \frac{\mathrm{d}}{\mathrm{d}t}(X^\top C X)$$

and this quadratic first integral. Therefore we require $b_i b_j - b_i a_{ij} - b_j a_{ji} = 0$ in order for the method to preserve quadratic invariants.

Part II: Our dynamical system is defined as $\dot{x} = f(x)$ with the initial condition $x(0) = x_0$. If we differentiate this problem with respect to x_0, we obtain

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\varphi_t'(x_0) &= f'(\varphi_t(x_0))\varphi_t'(x_0), \\
\varphi_0'(x_0) &= I
\end{aligned} \tag{2.10}$$

This is a differential equation involving the analytical flow map $\varphi_t(x_0)$. We will use the result that differentiating with respect to the initial condition and applying a symplectic RK method are commutative: either way we apply these operations, we end with the same result [13]. The analytic symplecticity criterion is $\varphi_t^\top J\varphi_t = J$, which is a quadratic invariant of the system. Assuming that Ψ denotes a symplectic RK method, we have $\Psi_n^\top J\Psi_n$ as the numerical approximation of the symplecticity constant. If we apply a symplectic RK method to the original problem, we obtain the numerical flow Ψ_n. Equivalently, if we apply a symplectic RK method to the variational equation 2.10, we have a numerical sensitivity Ψ_n'. Because the operations are commutative, applying a symplectic RK method to the variational equation respects the symplecticity condition. This is because this operation has the same result as applying the symplectic RK method first, and then differentiating with respect to x_0. Therefore, Runge-Kutta methods which preserve quadratic first integrals are symplectic. □

The Gauss-Legendre Runge-Kutta methods, which we know are symplectic, also satisfy the identity given in Theorem 2.18 [13]. However, the goal of this section has been to introduce properties that provide insight on symplectic RK methods. We have only given the example of these methods, and the reader should understand that Theorem 2.18 is not a requirement for a symplectic RK scheme.

2.3.5 Gauss-Legendre Runge-Kutta Methods

The Gauss methods are powerful choices for Runge-Kutta integration schemes. We have shown that explicit RK methods are not A-stable, since their stability functions are polynomials. Similarly, none of our analysis has concerned explicit symplectic RK methods. We cannot satisfy Proposition 2.16 with any explicit method since a stability function is not bounded as $|z| \to \infty$. Theorem 2.18 will only be valid for an implicit method.

In comparison, the Gauss-Legendre Runge-Kutta methods satisfy both A-stability and symplecticity. They are implicit methods and therefore more challenging to implement, however they compensate by their quality-preserving properties which are far better than general-purpose explicit methods. Furthermore, they are capable of order $\mathcal{O}(h^{2s})$ accuracy on s stages [21].

2.4 Analysis of Symplectic Methods

For all the following analysis, we consider the behaviour of a symplectic integrator and how it depends on the step size used. When applying a symplectic integrator, the step size must be constant [9] so we can regard it as an independed parameter.

2.4.1 Backward Error Analysis

When we perform numerical integration on a dynamical system given by $\dot{x} = f(x)$, we obtain a numerical solution in the form of the iteration $x_{n+1} = \Phi_h(x_n)$, where Φ is a method of our choice. This is a numerical solution, which may converge to the exact solution as $h \to 0$, but it is not exact. In backward error analysis, we look at the problem from another perspective. Instead of considering the closeness of our numerical solution to the system, we think of the numerical solution as an exact solution to a perturbed problem, and analyse the perturbation of this new problem to the original.

These methods and theorems follow the methods in Chapter IX of [13]. We want to find a modified equation $\dot{\tilde{x}} = f_h(\tilde{x})$ which is similar to $\dot{x} = f(x)$, and which is exactly solved by the

obtained numerical solution, i.e, $x_n = \tilde{x}(nh)$. We expect the perturbed problem to be of the form

$$\dot{\tilde{x}} = f_h(\tilde{x}) = f(\tilde{x}) + hf_2(\tilde{x}) + h^2 f_3(\tilde{x}) + \dots,$$

namely as a polynomial expansion about the original problem. Important to note is that this series is not guaranteed to converge as an infinite sum. Instead, we require a truncation of the series, which we perform by identifying bounds on the functions f_i, and truncate such that an infinimum of upper bounds is attained [9]. We want to match this expression to the numerical method such that $\tilde{x}(t+h) \equiv \Phi_h(\tilde{x}(t))$. Now consider the expansion of the perturbed problem as a Taylor series about a fixed time t. Write

$$\tilde{x}(t+h) = \tilde{x}(t) + h\dot{\tilde{x}}(t) + \frac{h^2}{2}\ddot{\tilde{x}}(t) + \dots$$

and recall we have assumed that $\tilde{x}(t) = f_h(\tilde{x})$. We can use this to expand the first few terms for clarity:

$$\begin{aligned}\tilde{x}(t+h) = \tilde{x}(t) &+ (f_h(\tilde{x}))\, h \\ &+ (f_h'(\tilde{x}) f_h(\tilde{x}))\, h^2 \\ &+ ((f_h''(\tilde{x}) + f_h'(\tilde{x})) f_h'(\tilde{x}) f_h(\tilde{x}))\, h^3 \\ &+ \dots\end{aligned}$$

However, recall from the defintion of the perturbed problem that $f_h(x)$ is a polynomial on h. Therefore, the terms expand as

$$\begin{aligned}\tilde{x}(t+h) = \tilde{x}(t) &+ \big(f(\tilde{x}) + hf_2(\tilde{x}) + h^2 f_3(\tilde{x}) + \dots\big)\, h \\ &+ \frac{1}{2!}\left((f'(\tilde{x}) + hf_2'(\tilde{x}) + \dots)(f(\tilde{x}) + hf_2(\tilde{x}) + \dots)\right) h^2 \\ &+ \frac{1}{3!}\left(((f''(\tilde{x}) + f'(\tilde{x})) + \dots)(f'(\tilde{x}) + \dots)(f(\tilde{x}) + \dots)\right) h^3 \\ &+ \dots\end{aligned}.$$

Now consider the numerical method. Assume that it takes the form

$$\Phi_h(x) = x + hf(x) + h^2 d_2(x) + h^3 d_3(x) + \dots$$

where we can find the d_i functions from the defined method. In order to satisfy the eqivalence we want, we match the coefficients of h in $\tilde{x}(t+h)$ and $\Phi_h(\tilde{x}(t))$:

$$\begin{aligned}&h^0 : x = x \\ &h^1 : f(x) = f(x) \\ &h^2 : d_2(x) = f_2(x) + \frac{1}{2!} f'(x) f(x) \\ &h^3 : d_3(x) = f_3(x) + \frac{1}{2!}(f'(x) f_2(x) + f_2'(x) f(x)) + \frac{1}{3!}(f''(x) f'(x) f(x) + f'(x) f'(x) f(x)) \\ &\quad\vdots\end{aligned}$$

Since we can find the d_i from the definition of the numerical method, and we know $f(x)$ from the definition of the system, we can rearrange these expressions to find the functions f_i that define the system. Therefore it is clear that the numerical solution provides a perturbed problem for which

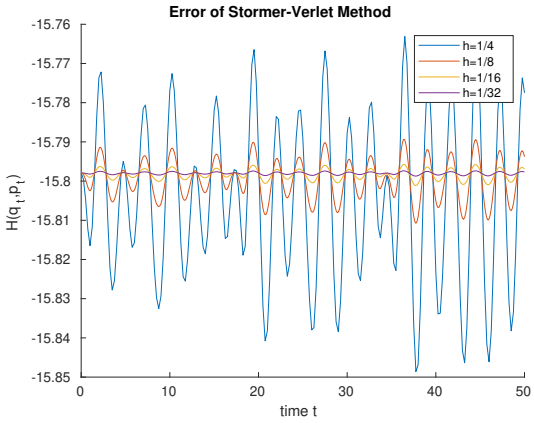

Figure 2.3: Evaluating the Hamiltonian for the model for phonation, integrated using the Störmer-Verlet method. Initial conditions are the same as that of Figure 2.2. The error of the method is shown to be $\mathcal{O}(h^2)$ in evaluating the Hamiltonian.

it is an exact solution at discrete time points. Furthermore, if we assume that the given method is $\mathcal{O}(h^p)$, then we have that $f_i = 0$ for $i \leq p$. This is because the expansions of $\tilde{x}(t+h)$ and $x(t+h)$ will be the same up to $\mathcal{O}(h^p)$.

We next want to consider a Hamiltonian system, and the nature of a modified Hamiltonian obtained from a numerical method.

2.4.2 The Perturbed Hamiltonian

We move on to discuss results on the closeness of the Hamiltonian that corresponds to the problem solved by the numerical method. This is an interesting result from backward error analysis. If we can deduce that the modified Hamiltonian is sufficiently close to the original, this merits the numerical solution in terms of preservation of quality.

Before considering the modified Hamiltonian, we need to introduce the concept of a generating function. This methodology is covered in [9] in a wider analysis. These are important when considering a change of coordinates. First, recall Hamilton's equations

$$\dot{q}_i = \frac{\partial H}{\partial p_i}, \qquad \dot{p}_i = \frac{-\partial H}{\partial q_i}.$$

Define a new coordinate pair $Q_i = Q_i(q,p,t)$ and $P_i = P_i(q,p,t)$. There must be another Hamiltonian $\tilde{H}$ which describes these transformed coordinates

$$\dot{Q}_i = \frac{\partial \tilde{H}}{\partial P_i}, \qquad \dot{P}_i = \frac{-\partial \tilde{H}}{\partial Q_i}.$$

This modified Hamiltonian tells us about the behaviour of the solution in this new coordinate system. The motivation for this transformation is the aim of describing the behaviour of the system in a potentially simpler form using a different coordinate system. There is a relation between these coordinates which is given by

$$\sum_{i=1}^{d} p_i \mathrm{d}q_i - H\mathrm{d}t = \sum_{i=1}^{d} P_i \mathrm{d}Q_i - \tilde{H}\mathrm{d}t + \mathrm{d}F(q,p). \tag{2.11}$$

This is an equation involving differential forms $\mathrm{d}q_i, \mathrm{d}t$, etc. but the key takeaway is the relationship involving the expression $\mathrm{d}F$ between coordinate systems. The relationship between the Hamiltonians themselves is

$$\tilde{H}(Q,P,t) = H(q,p,t) + \frac{\partial F}{\partial t}.$$

We call F a generating function. This is because the definition of F can be used to reconstruct the coordinate transformation. Note also that we wrote $\mathrm{d}F(q,p)$ on the original coordinates above, but if we know the transformation then we can write $F(q,p)$ and $F(Q,P)$ analogously by inverting the transformation. This is because $F(Q,P) = F(Q(q,p),P(q,p))$. We now introduce a result on the modified Hamiltonian.

Theorem 2.19 (Hairer, Lubich, Wanner 2006)**.** *Consider a generating function for a numerical method $\Phi_h(q,p)$ given by*

$$F(q,P,h) = hF_1(q,P) + h^2F_2(q,P) + \dots \tag{2.12}$$

where the functions F_i are defined on some domain D which is an open set. The modified Hamilton's equations are

$$\dot{q}_i = \frac{\partial \tilde{H}}{\partial p_i}, \qquad\qquad \dot{p}_i = \frac{-\partial \tilde{H}}{\partial q_i}.$$

where the modified Hamiltonian is

$$\tilde{H}(q,p) = H(q,p) + hH_2(q,p) + h^2H_3(q,p) + \dots \tag{2.13}$$

where if the method is order p, then $H_i = 0$ for $i \leq p$. The H_i are defined and smooth on D. Therefore, the closeness of the modified Hamiltonian to the original is $\mathcal{O}(h^p)$.

If this domain D is the entire space of points q, P then this result is globally defined, but over some restricted domain the result still holds but only locally. A proof is given in [13], but the result features in both and [9] gives examples. The proof makes use of mixed-variable generating functions. If we have a generating function F on (q,P), then

$$Q = \frac{\partial F}{\partial P}(q,P), \qquad\qquad p = \frac{\partial F}{\partial q}(q,P).$$

Furthermore, the proof requires a particular kind of generating function, being the function $\tilde{F}$ obtained from the solution of the Hamilton-Jacobi partial differential equation

$$\frac{\partial \tilde{F}}{\partial t}(q,P,t) = \tilde{H}\left(P, q + \frac{\partial \tilde{F}}{\partial P}(q,P,t)\right)$$

with initial condition $\tilde{S}(q,P,0) = 0$. This detail is necessary, and more detail is given in [13]. We now consider the proof for this result.

Proof. Let P, Q be the coordinates for the exact solution of the modified equation defined by the perturbed Hamiltonian $\tilde{H}$. We first want a generating function $\tilde{F}(q,P,t)$ defining the coordinate transformation. It is given (in the text) that if $\tilde{F}$ is a solution to the Hamilton-Jacobi PDE, then it is a unique solution which defines the map

$$Q = q + \frac{\partial \tilde{F}}{\partial P}(q,P,t), \qquad\qquad p = P + \frac{\partial \tilde{F}}{\partial q}(q,P,t).$$

This is an expression involving t, and our numerical method is developed using the parameter h. We want $\tilde{F}$ here to match the expression $F(q, P, h)$ given in the statement of the theorem at $t = h$. We can start by considering $\tilde{F}$ as a series expansion around $t = h$, which will take the form

$$\tilde{F}(q, P, t) = t\tilde{F}_1(q, P, h) + t^2\tilde{F}_2(q, P, h) + \dots$$

If we plug this into the Hamilton-Jacobi PDE, we can compare powers of t to obtain expressions for the terms in the series. The results come out by Taylor expansion in one dimension since $\tilde{H}$ evaluates at P. The first few terms are

$$\begin{aligned} \tilde{F}_1(q, P, h) &= \tilde{H}(q, P) \\ 2\tilde{F}_2(q, P, h) &= \frac{\partial \tilde{H}}{\partial q}(q, P, h) \cdot \frac{\partial \tilde{F}_1}{\partial P}(q, P, h) \\ &\dots \end{aligned} \tag{2.14}$$

The notation on the arguments is not important since these are just functions, but we stick with (q, P) for consistency. We have expressions for $\tilde{F}_j$ in terms of derivatives of $\tilde{H}$, and we also have an expression for $\tilde{H}$ about H in powers of h. If we let $\tilde{F}_j$ be *another* series of the form

$$\tilde{F}_j(q, P, h) = \tilde{F}_{j1}(q, P) + h\tilde{F}_{j2}(q, P) + h^2\tilde{F}_{j3}(q, P) + \dots$$

then we can use this expansion, and the expansion of $\tilde{H}$ in Equation 2.13, both in powers of h, and match coefficients by plugging the terms into the entries in Equation 2.14. Trivially, $\tilde{F}_{1k}(q, P) = H_k(q, P)$ from the first entry, so we have the original Hamiltonian plus a series in powers of h. For $j = 2$, we get terms such as

$$\begin{aligned} 2\tilde{F}_{21}(q, P) &= \frac{\partial H}{\partial q}(q, P) \cdot \frac{\partial H}{\partial P}(q, P) \\ 2\tilde{F}_{22}(q, P) &= \frac{\partial H_2}{\partial q}(q, P) \cdot \frac{\partial H}{\partial P}(q, P) + \frac{\partial H}{\partial q}(q, P) \cdot \frac{\partial H_2}{\partial q}(q, P) \\ &\dots \end{aligned}$$

In general for $j > 1$ we have that $\tilde{F}_{jk}(q, P)$ is a function depending on derivatives of H_l for $l \leq k$[2]. Finally, recall the purpose of the generating function. The function $F(q, P)$ defines the numerical method. Therefore $\tilde{F}(q, P, h)$ needs to match $F(q, P, h)$ as defined in Equation 2.12. We get the requirements

$$\begin{aligned} F_1(q, P) &= \tilde{F}_{11}(q, P) \\ F_2(q, P) &= \tilde{F}_{12}(q, P) + \tilde{F}_{21}(q, P) \\ &\dots \end{aligned}$$

from comparing coefficients of h. Recall that we have $\tilde{F}_{1k}(q, P) = H_k(q, P)$ from the first row of Equation 2.14. Therefore, the generating expression is a perturbation from the original Hamiltonian and we have

$$F_j(q, P) = H_j(q, P) + \mathcal{D}_j(H_k(q, P))$$

where $\mathcal{D}_j$ is some function of derivatives of H_k for $k \leq j$. This expression allows us to determine the H_j given a generating function.

[2]The book states $l < k$, however equality is attained, for example $\tilde{F}_{22}$ involves derivatives of H_2.

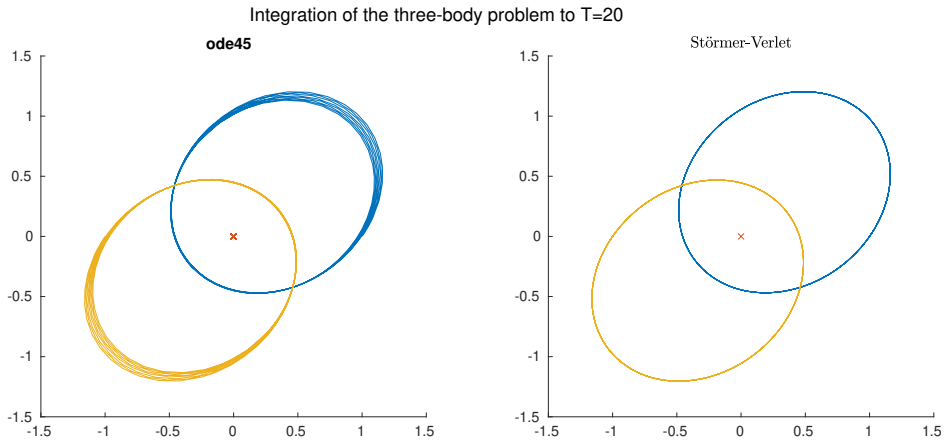

Figure 2.4: Integration of the three-body problem for a timespan $t \in [0, 20]$. Initial conditions are configured to construct an orbit. Two masses orbit a stationary third mass at the origin. Initial conditions on the $y = 0$ line at $(1, 0)$, $(0, 0)$ and $(-1, 0)$. Initial momentum is rotationally symmetric, approximately $(0.35, 0.53)$ at $(1, 0)$.

Recall the definition of the generating function from Equation 2.11. Assume that the numerical method defined by the generating function F is $\mathcal{O}(h^r)$. This generating function F itself is the same order. Since we match terms in F and $\tilde{F}$, both are $\mathcal{O}(h^r)$ i.e. the highest order term is h^{r+1}. Therefore, we must have that $\tilde{F}_{jk} = H_k = 0$ for $k \leq r$.

Furthermore, since the F_j are defined in terms of H_j, they must be defined on the same domain D. □

The direct implication is that a higher order method will converge faster to the true *qualitative behaviour*. Figure 2.3 shows the phonation problem from earlier. We evaluate the Hamiltonian using its closed form expression using the numerical solution generated by the Störmer-Verlet scheme. As we can see, the perturbed Hamiltonian approaches the unmodified Hamiltonian by $\mathcal{O}(h^2)$, the order of the method.

2.5 Applications

2.5.1 Example - The Three-Body Problem

The three-body problem [27] is a huge point of interest in celestial mechanics. It describes the motion of three point-masses in a closed system which move under gravitational acceleration to each other. An important subclass is the *restricted* three-body problem, in which one mass is relatively large in magnitude, and can be regarded as fixed compared to the other two bodies. The restricted three-body problem can be used to model the motion of the earth and moon relative to the sun.

The classical form of the dynamics for the three-body problem is

$$\ddot{x}_1 = -Gm_2\frac{x_1 - x_2}{|x_1 - x_2|^3} - -Gm_3\frac{x_1 - x_3}{|x_1 - x_3|^3}$$
$$\ddot{x}_2 = -Gm_3\frac{x_2 - x_3}{|x_2 - x_3|^3} - -Gm_1\frac{x_2 - x_1}{|x_2 - x_1|^3}$$
$$\ddot{x}_3 = -Gm_1\frac{x_3 - x_1}{|x_3 - x_1|^3} - -Gm_2\frac{x_3 - x_2}{|x_3 - x_2|^3}$$

where x_i is the vector position of the point-mass particle with mass m_i. In Hamiltonian form [27], this is the problem

$$\dot{q}_i = \frac{\partial H}{\partial p_i}, \qquad \dot{p}_i = \frac{-\partial H}{\partial q_i}.$$

where

$$H(q,p) = -\frac{Gm_1m_2}{|q_1 - q_2|} - \frac{Gm_2m_3}{|q_2 - q_3|} - \frac{Gm_3m_1}{|q_3 - q_1|} + \frac{p_1^2}{2m_1} + \frac{p_2^2}{2m_2} + \frac{p_3^2}{2m_3}.$$

for particles with position q_i and momentum p_i.

See Figure 2.4 for a visualisation of the trajectories of the three body problem under a particular set of initial conditions. The integration from `ode45()` shows clear drift from the original path of the orbit. A very high relative tolerance of 10^{-12} was used for this method, so this is arguably the best approximation we can hope to compute with an explicit method. In comparison, the Störmer-Verlet scheme is only second order accurate, but we are able to maintain the path of orbit over the same timespan because it is a symplectic method. Any deviation from the orbit path is not visible.

See Appendix B.2.1 for implementations and tests of symplectic integration schemes.

2.5.2 Review

We have given an overview of the structure of problems in Hamiltonian mechanics and how the property of symplecticity is directly related. We have introduced several symplectic integration methods and the theory behind them. We have demonstrated these with examples to show the utility of these methods. It is clear that when the nature of the Hamiltonian is important to the behaviour of the problem, it would be beneficial to use a symplectic method.

Symplectic integration is extremely prevalent in celestial mechanics, where we may need to integrate a Hamiltonian system over a long timespan in order to estimate the motion of celestial bodies far into the future. The three-body problem is a simple but insightful demonstration of problems such as these. Other applications in physics are molecular dynamics and particle physics.

Having covered symplectic integration, the theory for which is extremely prevalent and researched, we now move on to the field of positivity preservation, which is a much more recent and unclear area of geometric numerical integration.

Chapter 3

Positivity Preservation

3.1 Positive Solutions to ODEs

3.1.1 Motivation

There are many problems which motivate the need for numerical methods which preserve positivity of the solution. For example, consider the problem of simulating a chemical reaction. We start with a finite amout of positive-valued species at the initial stage, and apply some numerical method to produce a result. Despite being able to apply methods which have high orders of accuracy, traditional methods will not preserve positivity unconditionally. If our numerical solution indicates that the concentrations of any number of species become negative, then our solution is not qualitatively accurate to a "true" solution.

Consider a problem given by $\dot{x} = f(x)$, with the initial condition $x(0) = x_0$. For a one-dimensional problem, the true solution is positive if, given $x_0 \geq 0$, we have that $x(t) \geq 0$ for $t > 0$. Positivity of the numerical solution can be expressed as the condition that if $x_i \geq 0$ then $x_{i+1} \geq 0$ for all $i = 1, \ldots, N$ timesteps in the computation. Like always, x may be vector-valued. If $x \in \mathbb{R}^d$, then the condition for positivity of the numerical solution applies element-wise to each component $x_i^{(k)}$ for $k = 1, \ldots, d$. Importantly, entries *can* be zero.

Methods which preserve positivity are a current area of research. It is difficult to produce numerical methods which both preserve positivity and maintain high order accuracy. Moreover, it is very difficult to formulate positivity preserving methods for general problems. Our approach will be to consider particular cases, where we can reduce the problem $\dot{x} = f(x)$ to a particular form, and formulate positivity-preserving methods which are effective for these problems.

3.1.2 The Graph-Laplacian Matrix

Before looking at positivity-preserving methods, we will consider problems which themselves must admit positivity-preserving solutions. The notion of the graph-Laplacian matrix and the methods centered around it are taken from [4].

Our main focus will be on problems that we can write in the form

$$\dot{x} = A(x)x$$

where A is a graph-Laplacian matrix [4].

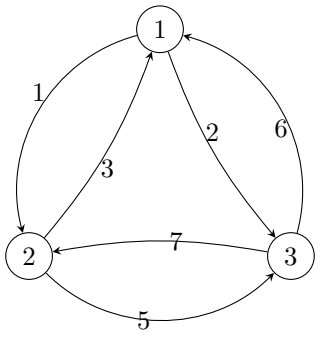

Figure 3.1: A directed graph, given for the demonstration of a graph-Laplacian matrix, taken from Wikipedia [1]. Nodes are indexed from 1 to 3 and each directed edge has a given weight. Each node is connected to the other.

Definition 3.1. A graph-Laplacian matrix $A \in \mathbb{R}^{d\times d}$ is a square matrix that satisfies

- $\mathbf{1}^\intercal A = \mathbf{0}^\intercal$ (its column sums are zero).
- For all $i, j = 1, \ldots, d$ where $i \neq j$ we have $A_{ij} \geq 0$.
- For all $i = 1, \ldots, d$ we have $A_{ii} \leq 0$

Note that here, A can depend on x. The entries in $A(x)$ can contain entries of x in any fashion, as long as the definition is satisfied.

The name "graph-Laplacian" comes from graph theory, and the definition is not universally agreed upon. For our needs, we consider a graph-Laplacian matrix for a directional graph to be the in-degree matrix (empty diagonal) minus the in-degree adjacency matrix (diagonal) of the graph. Consider the graph given by Figure 3.1. The in-degree matrix D for this graph is

$$D = \begin{bmatrix} 9 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

where d_{ii} is the in-degree of vertex i and $d_{ij} = 0$ for $i \neq j$. The adjacency matrix C is

$$C = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 0 & 5 \\ 6 & 7 & 0 \end{bmatrix}$$

where c_{ij} is the weight of the edge from vertex i to vertex j. We define the Laplacian matrix A for a graph by

$$A = C - D. \tag{3.1}$$

We have covered this case for integers, however in general we will consider graph-Laplacian matrices over real numbers. If a problem admits the reduction to graph-Laplacian form like the above, we can show that the solution retains positivity.

Theorem 3.2 (Preservation of Positivity [4])**.** *Solutions of $\dot{x} = A(x)x$ retain positivity if A is graph-Laplacian.*

Proof. Consider the solution $x(t^*)$ at time t^*. Assume that there are indices $k \in \{1, \dots, d\}$ where all entries $x_k(t^*)$ are zero and the rest are strictly positive. Then the k-th equation in the matrix system is

$$\dot{x}_k(t^*) = \sum_{l=1}^{d} A_{kl}(x(t^*))x_l(t^*).$$

Entry A_{kk} is negative, but we assumed $x_k(t^*) = 0$ so the only contributions to the sum are the A_{kl} and x_l for $l \neq k$. Some of the x_l may be zero but all the rest are positive, so we have $\dot{x}_k \geq 0$. Note that equality is attained if all the x_k are zero which is the trivial solution. Assume instead that every entry $x_k(t^*)$ is non-zero and hence positive. In this case there are no restrictions on the derivative of the k-th entry - the A_{kk} term means this derivative can be negative. The x_k term can go to zero, but since the derivative is strictly non-negative at zero, we will retain positivity. □

We have shown that the graph-Laplacian form is a suitable structure for problems where we are concerned about positivity preservation. From now on, for a vector solution $x \in \mathbb{R}^d$ we will write $x \geq 0$ to mean that each element $x_k \geq 0$ for $k = 1, \dots, d$. Before moving on to numerical methods that preserve positivity of these problems, we will consider a few points of note.

3.1.3 Non-autonomous Systems

Our analysis so far has focused on problems $\dot{x} = A(x)x$, which are autonomous. Examples in positivity preservation can often be non-autonomous, which we would write as $\dot{x} = A(x, t)x$. The particular details of problems involving the graph-Laplacian matrix can be easily analogised to the non-autonomous case. As such, we will usually only consider problems in the autonomous form, unless it is necessary to the analysis.

Some problems can involve several different timescales, in which case time-dependence is very important to account for.

3.1.4 Conservation of Mass

When problems describe an exchange of matter, it is useful for this matter to be conserved in our solution. First, let e be the vector of ones. Conservation of mass is the condition that x satisfies $e^\top x = C$ for some constant C. Mass is conserved for graph-Laplacian problems because

$$\frac{\mathrm{d}}{\mathrm{d}t} e^\top x = e^\top \dot{x} = e^\top A(x)x = \mathbf{0}^\top x = 0$$

hence clearly $e^\top x$ is constant. In the literature [4] we often have $C = 1$ for simplicity, there is no loss of generality because this can be applied to some scaling of the initial condition such that $e^\top x_0 = 1$.

Mass preservation can be generalised to higher order problems involving inner products with the variable of interest. These problems concern conservation of other properties, which we will not explore in depth.

3.2 Positivity Preserving Methods

3.2.1 Brute force approach

Before considering the problem of positivity preserving methods in depth, we will first consider the method which we will refer to as "fixing" or "clipping". We use the forward Euler method

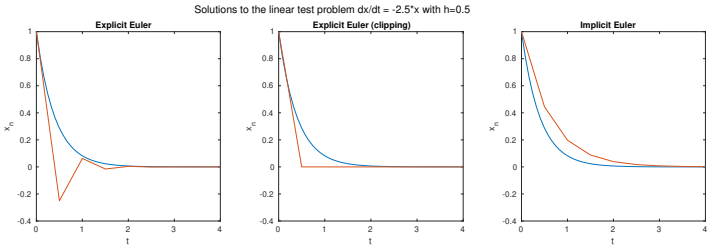

Figure 3.2: Comparison of methods applied to the linear test problem. True solution is marked in blue on each plot.

$x_{i+1} = x_i + hf(x_i)$, except at every step we perform a fix by considering every entry in x_i and if it is negative, we just set it to zero. We can write this formally as the method

$$x_{i+1} = x_i + hf(x_i)$$
$$x_{i+1} = \mathcal{H}_+(x_{i+1})$$

where $\mathcal{H}_+$ is the thresholder that sets all negative entries to zero. This is equivalent to a projection of x to a vector subspace of lower dimension, determined by in which dimensions x is positive.

This method is very cheap, and it preserves positivity, so it seems like an easy choice. The fixing step to eliminate negative entries can be applied in general to any numerical method. See Figure 3.2, where we have compared regular explicit Euler, explicit Euler with clipping, and implicit Euler methods applied to the linear test problem. Regular Euler's method fails to preserve positivity unconditionally, while the clipping method adjusts negative solutions to zero. Backward Euler appears to preserve positivity.

Euler's method serves as a useful baseline to demonstrate experimental ideas. Being a first-order accurate method, it does not serve as an effective choice for general applications. We will instead look at different approaches to preserving positivity of the numerical solution, using the graph-Laplacian structure explored earlier.

3.2.2 General Solutions

Consider the problem $\dot{x} = Ax$, with initial condition $x(t = 0) = x_0$, with a constant square matrix A. The general solution to this problem is

$$x(t) = \mathrm{e}^{tA} x_0$$

where the matrix exponential is analogised from the scalar case

$$\mathrm{e}^A = I + A + \frac{1}{2}A^2 + \frac{1}{6}A^3 + \ldots = \sum_{j=0}^{\infty} \frac{A^j}{j!}.$$

We have shown that in the case where A is graph-Laplacian, solutions preserve positivity.

We can't immediately apply this exponential solution to the problem $\dot{x} = A(x)x$ because of the non-linear right hand side, which will lead to a different expansion via. product rule. To see why, we will demonstrate the expansion. Assume that the solution for the graph-Laplacian problem is $x(t) = \exp(tA(x))x_0$. Then

$$\dot{x} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\mathrm{e}^{A(x)t} x_0\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left[A(x)t\right] \mathrm{e}^{A(x)t} x_0 = \left[A(x) + tA'(x)A(x)x\right] x$$

which does not satisfy the problem. Note that this expension involves a derivative of $A(x)$. More on this later.

3.2.3 Exponential Euler's Method and Matrix-Vector Taylor Series

We said that we can't apply the exponential solution $x(t) = \exp(tA)x_0$ to the problem $\dot{x} = A(x)x$. We will show this by demonstrating what happens if we do. We propose a method of the form

$$x_{n+1} = \mathrm{e}^{hA(x_n)}x_n. \tag{3.2}$$

This method fixes A for each step. Therefore, one step is equivalently providing the analytical exponential solution to the problem $\dot{x} = A(y)x$, where $y = x(t = 0) = x_n$. In our scope of analysis we will call this the "exponential Euler method", due to its similarity to the classic method. Considering the true solution after a timestep, we want to find an expression for the *truncation error*. We write $x(t_n) = x_n$ at the current step and look at the difference between $x(t_n + h)$ (the true value) and x_{n+1} (the method). However, first we need to consider the Taylor expansion on $x(t + h)$

$$x(t + h) = x(t) + h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t) + \dots$$

which involves derivatives of the nonlinear vector function $A(x)x$

$$x(t + h) = x + hA(x)x + \frac{h^2}{2}\left[A'(x)A(x)xx + A(x)^2x\right] + \mathcal{O}(h^3)$$

writing $x = x(t)$. As we can see, this involves derivatives of $A(x)$, and associativity of matrix multiplication breaks. To clarify, multiplication following from left to right should always perform defined operations. Derivatives of $A(x)$ are rank-incrementing tensors: since $A \in \mathbb{R}^{d\times d}$, the first derivative is $A'(x) \in \mathbb{R}^{d\times d\times d}$. The k-th order derivative satisfies $A^{(k)}(x) \in \mathbb{R}^{d^{k+2}}$. This analysis is essential for deriving the order of methods for problems of the form we are working with.

Evaluating the truncation error of the exponential Euler method gives us the expression

$$\begin{aligned}
\tau(h) = x(t_n + h) - x_{n+1} &= x(t_n + h) - \mathrm{e}^{hA(x_n)}x_n \\
&= x(t_n) + h\dot{x}(t_n) + \frac{h^2}{2}\ddot{x}(t_n) + \mathcal{O}(\langle^{\ni}) - \mathrm{e}^{hA(x_n)}x_n \\
&= x_n + hA(x_n)x_n + \frac{h^2}{2}\left(A'(x_n)A(x_n)x_nx_n + A(x_n)^2x_n\right) + \mathcal{O}(h^3) \\
&- \left(I + hA(x_n) + \frac{h^2}{2}A(x_n)^2 + \mathcal{O}(h^3)\right)x_n \\
&= \frac{h^2}{2}\left(A'(x_n)A(x_n)x_nx_n\right) + \mathcal{O}(h^3)
\end{aligned}$$

hence the exponential Euler method is first order accurate.

3.2.4 The Second-Order Strang Splitting Method

The exponential Euler method involved considering the problem with one fixed component in order to apply the exponential solution. We will consider the separation of components in more depth as in [4] so that we can justify the construction of higher order methods.

We have the problem $\dot{x} = A(x)x$. Introduce the variable z where the initial conditions for x and z satisfy $x(t=0) = x_0 = z_0 = z(t=0)$. We then write the problem in two parts as

$$\begin{aligned} \dot{x} &= A(z)x \\ \dot{z} &= A(x)z. \end{aligned}$$

By separating the problem into two, we are essentially solving the problem twice. However, the way we have distributed x and z means we can decompose this into two separate problems where we solve one for x and one for z. The first problem is

$$\begin{aligned} \dot{x} &= A(z)x \\ \dot{z} &= 0 \end{aligned} \tag{3.3}$$

which has solution

$$\begin{aligned} x(t) &= \mathrm{e}^{tA(z_0)}x_0 \\ z(t) &= z_0 \end{aligned}$$

while the second problem is of alternate form for x and z

$$\begin{aligned} \dot{x} &= 0 \\ \dot{z} &= A(x)z \end{aligned} \tag{3.4}$$

which has solution

$$\begin{aligned} x(t) &= x_0 \\ z(t) &= \mathrm{e}^{tA(x_0)}z_0. \end{aligned}$$

It may appear confusing as to why we split the problem into two parts. We know that if a problem is given by $\dot{x} = A(x)x$ then its solutions preserve positivity. We also know that if a problem is given instead by $\dot{x} = Ax$ for any constant matrix A, then it has general solution given by $x(t) = \exp(tA)x_0$. Therefore, by splitting the problem into two problems on x and z, the solutions are given by the exponential form since the matrices are fixed in each problem, and the solutions preserve positivity because the matrices are of graph-Laplacian form. Therefore these solutions preserve positivity [4], which we can use to begin constructing methods which maintain this.

The primary method given in [4], which is a solution to Equations 3.3 and 3.4 is given by the equations

$$\begin{aligned} x_{n+\frac{1}{2}} &= \exp\left(\frac{h}{2}A(z_n)\right)x_n \\ z_{n+1} &= \exp\left(hA(x_{n+\frac{1}{2}})\right)z_n \\ x_{n+1} &= \exp\left(\frac{h}{2}A(z_{n+1})\right)x_{n+\frac{1}{2}}. \end{aligned} \tag{3.5}$$

The solution x_n, z_n or $(x_n + z_n)/2$ is second order accurate. We perform a half step in x, followed by a full step in z and then a second half-step in x, similar to Verlet integration. In [4], it is stated that $|x_n - z_n|$ can be used as an estimate of the truncation error, for the method to be used adaptively with a variable time step.

We will provide our own analysis on this method, by producing our own results on the truncation error of these methods. First, we evaluate the truncation error of z. The whole numerical method can

be written compactly by writing the explicit expression for $x_{n+\frac{1}{2}}$ inside the term. Denote $x := x(t_n)$ and $A := A(x)$ for compactness, also $A' := A'(x)$ and the same generalised for any derivatives. For an attempt of ease of notation, square brackets contain objects which are matrices or higher rank tensors, while parentheses contain objects which are vectors. We can begin by writing the stage directly as

$$z_{n+1} = \exp\left[hA\left(\exp\left(\frac{h}{2}A(z_n)\right)x_n\right)\right]z_n$$

and we note that $x_n = z_n = x(t_n)$, so we can swap every $z + n$ for x. In full,

$$z_{n+1} = \exp\left[hA\left(\exp\left(\frac{h}{2}A(x)\right)x\right)\right]x. \tag{3.6}$$

Now start evaluating the truncation error.

$$\begin{aligned}
x(t_n + h) - z_{n+1} &= x(t_n + h) - \exp\left[hA\left(\exp\left[\frac{h}{2}A\right]x\right)\right]x \\
&= x(t_n + h) - \left[I + h\left[A\left(\exp\left[\frac{h}{2}A\right]x\right)\right]\right. \\
&\quad + \frac{h^2}{2}\left[A\left(\exp\left[\frac{h}{2}A\right]x\right)\right]^2 \\
&\quad \left. + \frac{h^3}{6}\left[A\left(\exp\left[\frac{h}{2}A\right]x\right)\right]^3\right]x + \mathcal{O}(h^4).
\end{aligned}$$

The inner term needs to be expanded out

$$\begin{aligned}
A\left(\exp\left[\frac{h}{2}A\right]x\right) &= A\left(\left[I + \frac{h}{2}A + \frac{h^2}{8}A^2\right]x\right) + \mathcal{O}(h^3) \\
&= A\left(x + \frac{h}{2}Ax + \frac{h^2}{8}A^2x\right) + \mathcal{O}(h^3) \\
&= A + A' \cdot \left(\frac{h}{2}Ax + \frac{h^2}{8}A^2x\right) + A''(x)\left(\frac{h^2}{8}AxAx\right) + \mathcal{O}(h^3) \\
&= A + \frac{h}{2}A'Ax + \frac{h^2}{8}\left(A'A^2x + A''AxAx\right) + \mathcal{O}(h^3).
\end{aligned}$$

Note that an internal expansion leads to a big-O term inside the brackets, which we can take outside the brackets by Taylor expansion centred at the collection of terms before the big-O expression. If

we plug this back in, the terms expand as

$$\begin{aligned}
&x(t_n+h) - \left[I + h\left[A + \frac{h}{2}A'Ax + \frac{h^2}{8}\left[A'A^2x + A''AxAx\right]\right]\right.\\
&\quad + \frac{h^2}{2}\left[A + \frac{h}{2}A'Ax\right]\left[A + \frac{h}{2}A'Ax\right] + \frac{h^3}{6}A^3\Bigg]x + \mathcal{O}(h^4)\\
&= x(t_n+h) - \left[I + hA + \frac{h^2}{2}\left[A'Ax + A^2\right]\right.\\
&\quad + \frac{h^3}{8}\left[A'A^2x + A''AxAx + 2A'AxA + 2AA'Ax + \frac{4}{3}A^3\right]\Bigg]x + \mathcal{O}(h^4)\\
&= x(t_n+h) - \Bigg(x + hAx + \frac{h^2}{2}\left(A'Axx + A^2x\right)\\
&\quad + \frac{h^3}{8}\left(A'A^2xx + A''AxAxx + 2A'AxAx + 2AA'Axx + \frac{4}{3}A^3x\right)\Bigg) + \mathcal{O}(h^4).
\end{aligned}$$

We can evaluate the Taylor expansion for x itself

$$\begin{aligned}
x(t_n+h) &= x + h\dot{x} + \frac{h^2}{2}\ddot{x} + \frac{h^3}{6}\dddot{x} + \mathcal{O}(h^4)\\
&= x + hAx + \frac{h^2}{2}\left(A'Axx + A^2x\right)\\
&\quad + \frac{h^3}{6}\left(A''AxAxx + A'A'Axxx + 2A'A^2xx + AA'Axx + A^3x\right) + \mathcal{O}(h^4).
\end{aligned} \tag{3.7}$$

Written on its own, the expansion of z_{n+1} is

$$\begin{aligned}
z_{n+1} = x + hAx &+ \frac{h^2}{2}\left(A'Axx + A^2x\right)\\
&+ \frac{h^3}{8}\left(A'A^2xx + A''AxAxx + 2A'AxAx + 2AA'Axx + \frac{4}{3}A^3x\right)\\
&+ \mathcal{O}(h^4)
\end{aligned} \tag{3.8}$$

By comparing terms, the truncation error of the z component is

$$\tau_z(h) = h^3\left(\frac{1}{24}A''AxAxx + \frac{1}{6}A'A'Axxx + \frac{5}{24}A'A^2xx - \frac{1}{12}AA'Axx - \frac{1}{4}A'AxAx\right) + \mathcal{O}(h^4). \tag{3.9}$$

Now look at the truncation error provided by the x part of the method. We already have, from above, an expression for z_{n+1} in terms of powers of h, which will make the evaluation easier. Write the expression for x_{n+1}

$$x_{n+1} = \exp\left[\frac{h}{2}A(z_{n+1})\right]x_{n+\frac{1}{2}}$$

which is equivalently

$$x_{n+1} = \exp\left[\frac{h}{2}A(z_{n+1})\right]\exp\left[\frac{h}{2}A\right]x. \tag{3.10}$$

Expand the expression

$$\begin{aligned} x_{n+1} &= \exp\left[\frac{h}{2}A\left(x + hAx + \frac{h^2}{2}\left(A'Axx + A^2x\right) + \mathcal{O}(h^3)\right)\right]\exp\left[\frac{h}{2}A\right]x \\ &= \exp\left[\frac{h}{2}A + \frac{h}{2}A'\cdot\left(hAx + \frac{h^2}{2}\left(A'Axx + A^2x\right)\right)\right]\exp\left[\frac{h}{2}A\right]x + \mathcal{O}(h^4) \\ &= \exp\left[\frac{h}{2}A + \frac{h^2}{2}A'Ax + \frac{h^3}{4}\left[A'A'Axx + A'A^2x\right]\right]\exp\left[\frac{h}{2}A\right]x + \mathcal{O}(h^4). \end{aligned}$$

Note that because of the $(h/2)$ scale, taking out the $\mathcal{O}(h^3)$ term by expansion leads to an $\mathcal{O}(h^4)$ term outside the expression. This saves writing z_{n+1} up to the order h^3 term, since it would not be included in the final expression up to order $\mathcal{O}(h^3)$. Using the definition of the exponential for both parts, we get

$$\begin{aligned} x_{n+1} = &\left[I + \left[\frac{h}{2}A + \frac{h^2}{2}A'Ax + \frac{h^3}{4}\left[A'A'Axx + A'A^2x\right]\right]\right. \\ &+ \frac{1}{2}\left[\frac{h^2}{4}A^2 + \frac{h^3}{4}\left[AA'Ax + A'AxA\right]\right] \\ &\left.+ \frac{1}{6}\left[\frac{h^3}{8}A^3\right]\right] \times \left[I + \frac{h}{2}A + \frac{h^2}{8}A^2 + \frac{h^3}{48}A^3\right]x + \mathcal{O}(h^4). \end{aligned}$$

We have only included every term up to third order, and then taken all higher order terms outside of the expansions. We can rearrange this into powers of h to get

$$\begin{aligned} x_{n+1} = &\left[I + \frac{h}{2}A + \frac{h^2}{8}\left[4A'Ax + A^2\right]\right. \\ &\left.+ \frac{h^3}{48}\left[12A'A'Axx + 12A'A^2x + 6AA'Ax + 6A'AxA + A^3\right]\right] \\ &\times \left[I + \frac{h}{2}A + \frac{h^2}{8}A^2 + \frac{h^3}{48}A^3\right]x + \mathcal{O}(h^4). \end{aligned}$$

Multiplying this expression, considering only terms up to third order, we get

$$\begin{aligned} x_{n+1} = &\left[I + hA + \frac{h^2}{8}\left[4A'Ax + 4A^2\right]\right. \\ &\left.+ \frac{h^3}{48}\left[12A'A'Axx + 12A'A^2x + 6AA'Ax + 6A'AxA + 2A^3\right] + \frac{h^3}{16}\left[4A'AxA + 2A^3\right]\right]x \\ &+ \mathcal{O}(h^4). \end{aligned}$$

By collecting terms, simplifying, and multiplying through by x, we get the expression

$$\begin{aligned} x_{n+1} = x &+ hAx + \frac{h^2}{2}\left(A'Axx + A^2x\right) \\ &+ \frac{h^3}{8}\left(2A'A'Axxx + 2A'A^2xx + AA'Axx + 3A'AxAx + \frac{4}{3}A^3x\right) \\ &+ \mathcal{O}(h^4) \end{aligned} \tag{3.11}$$

Comparing this to the true expansion from Equation 3.7, we can evaluate the truncation error $x(t_n + h) - x_{n+1}$ as

$$\tau_x(h) = h^3 \left(\frac{1}{6} A'' A x A x x - \frac{1}{12} A' A' A x x x + \frac{1}{12} A' A^2 x x + \frac{1}{24} A A' A x x - \frac{3}{8} A A' A x x \right) + \mathcal{O}(h^4). \quad (3.12)$$

We have obtained direct expressions for the truncation errors for both components of the splitting method. Both expressions for truncation error, from Equations 3.9 and 3.12, indicate that the method is second-order accurate.

The multiplication of the tensors as we have written them is not defined to satisfy associativity. With the aim of relating the notation of Butcher [8], we evaluate terms in the derivatives to provide some clarity. We start with the definition of the system

$$\dot{x} = f(x) = A(x)x$$

We consider derivatives in the general case:

$$\begin{aligned}
\dot{x} &= f(x) \\
\ddot{x} &= f'(x)\dot{x} = f'(x)f(x) = f'f \\
\dddot{x} &= f''(x)f(x)f(x) + f'(x)f'(x)f(x) = f''(f,f) + f'f'f
\end{aligned}$$

We can express terms for the derivatives just in f, which only go down to x derivatives

$$\begin{aligned}
f(x) &= A(x)x = Ax \\
f'(x) &= \frac{\partial f}{\partial x} = \frac{\partial}{\partial x}\left(A(x)x\right) = A'(x)x + A(x) = A'x + A \\
f''(x) &= \frac{\partial}{\partial x}\left(A'(x)x + A(x)\right) = A''(x)x + A'(x) + A'(x) = A''x + 2A'
\end{aligned}$$

and we can write the derivatives of x in a way that resembles the convention

$$\begin{aligned}
\dot{x} &= f = Ax \\
\ddot{x} &= f'f = \left[A'x + A\right](Ax) \\
\dddot{x} &= f''(f,f) + f'f'f = \left[A''x + 2A'\right](Ax, Ax) + \left[A'x + A\right]\left[A'x + A\right](Ax)
\end{aligned}$$

where $A''x + 2A'$ is a third rank tensor, $A'x + A$ is a matrix and Ax is a vector.

We will now briefly consider statements from [4] regarding the properties of the splitting method. We have mentioned the authors' claim that the difference $|x_{n+1} - z_{n+1}|$ can be taken as an estimate of the error of the method. Using the expansions in Equations 3.8 and 3.8, we can write their difference as

$$z_{n+1} - x_{n+1} = \frac{h^3}{8}\left(A''AxAxx + AA'Axx - A'A^2xx - A'AxAx - 2A'A'Axx\right) + \mathcal{O}(h^4). \quad (3.13)$$

This expression on the difference between methods contains every expression in the truncation error for z (Equation 3.9), of which there are five. This may motivate its use as an estimate of the error. However on comparing the two, there is a difference of sign for three expressions, and no consistency of scaling. The difference expression does not contain the term $AA'Axx$, which appears in the truncation error for x.

One potentially interesting component of this analysis comes from how the different stages eliminate different elements in the truncation errors. We may be motivated to take a linear combination

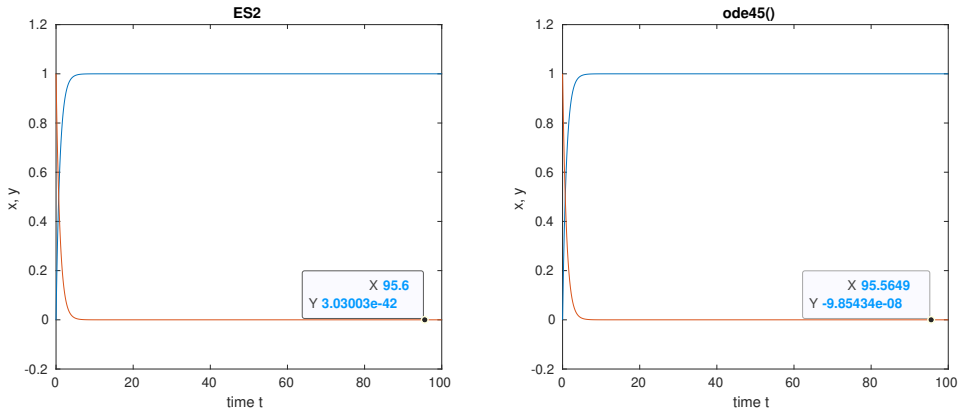

Figure 3.3: Test problem on the system given by Equation 3.14. The left figure uses the "ES2" method given by Equation 3.5 taking its x values as opposed to the z values. ES2 preserves positivity of the solution throughout. The right figure uses MATLAB's `ode45()` integrator, with a reduced tolerance in order to break positivity. The values used were $a = 0$ with initial condition $x_0 = 0$, $y_0 = 1$ running for a timespan up to $T = 100$. Confusingly the MATLAB graph indexes the horizontal axis X and the vertical Y, whereas we plot x and y on the vertical axis against t. The point labels are necessary to indicate that the MATLAB integrator does not preserve positivity.

of x_{n+1} and z_{n+1} such that we can obtain a new truncation error which is, in some sense, minimised. Arguments are given in the Appendix, since the method borrows from convex optimisation [6] and is not entirely relevant to the discussion. Particularly, the approach is similar to Ralston [28]. We find that the appropriate linear combination, of the form $y_{n+1} = \mu z_{n+1} + \lambda x_{n+1}$, would be theoretically best for $\mu = 17/24, \lambda = 7/24$.

The author acknowledges that the majority of this analysis on the truncation error of the splitting method is limited, and there is a lack of clarity regarding the evaluation of the terms which appear in the Taylor expansions. Improvements to the results could be made, however we have decided to leave this analysis as it stands in the case of diminishing returns. The analysis is the best we could achieve in this regard.

3.2.5 Example - A Linear Problem on Two Variables

Consider the system given by

$$\frac{\mathrm{d}x}{\mathrm{d}t} = y - ax, \quad \frac{\mathrm{d}y}{\mathrm{d}t} = ax - y \tag{3.14}$$

where a is a given constant. This problem is explored briefly for demonstrative purposes in [7]. The problem admits formulation into the graph-Laplacian form

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} -a & 1 \\ a & -1 \end{bmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

where the matrix is constant.

A numerical solution is given in Figure 3.3. We show that a conventional explicit method, such as the Runge-Kutta method(s) applied in MATLAB's `ode45()` integrator does not unconditionally preserve positivity. The formulation of the problem into graph-Laplacian form and the application of

a nonnegative initial condition indicates that our solution should preserve positivity, but MATLAB's integrator does not maintain this. Therefore we have shown that positivity may not be maintained by conventional integrators, but it is succesful for the ES2 integrator which we know to unconditionally preserve positivity.

Implementation of ES2 with further testing is provided in Appendix B.3.1.

3.2.6 The Magnus Integrator

An alternative method suggested in [4] is the second order Magnus integrator. We will first discuss the Magnus expansion [23, 3], which allows us to construct this method. The Magnus expansion considers the linear ODE given by

$$\dot{x} = A(t)x$$

with initial condition $x(t=0) = x_0$. Since A is not time-independent, the exponential solution we have discussed earlier does not hold. We state that the problem is linear, but the time dependence analogises to the cases we have looked at for a matrix $A(x)$ or $A(t,x)$. The Magnus expansion considers the solution of this problem to be of the form

$$x(t) = \exp(\Omega(t))x_0$$

for some matrix function $\Omega(t)$ which can be expressed in terms of the governing matrix A. We write $\Omega(t)$ as an infinite sum over $\Omega_j(t)$ functions. From the literature [3], we have formulae for these entries. Specifically, we write

$$\begin{aligned}
\Omega(t) &= \Omega_1(t) + \Omega_2(t) + \ldots = \sum_{j=0}^{\infty} \Omega_j(t) \\
\Omega_1(t) &= \int_0^t A(\tau)\mathrm{d}\tau \\
\Omega_n(t) &= \sum_{k=1}^{n-1} \frac{B_k}{k!} \int_0^t S_n^{(k)}(\tau)\mathrm{d}\tau
\end{aligned}$$

where B_k are the Bernoulli numbers [2] and the $S_n^{(j)}$ matrices are defined

$$\begin{aligned}
S_n^{(1)} &= [\Omega_{n-1}, A] \\
S_n^{(n-1)} &= \mathrm{ad}_{\Omega_1}^{n-1}(A) \\
S_n^{(j)} &= \sum_{m=1}^{n-j} \left[\Omega_m, S_{n-m}^{(j-1)}\right] \quad \text{for } 2 \le j \le n-1.
\end{aligned}$$

where $[\cdot,\cdot]$ is the matrix commutator $[A,B] = AB - BA$ and $\mathrm{ad}_A^j B = [A, \mathrm{ad}_A^{(j-1)} B]$ where $\mathrm{ad}_A^0 B = B$. For clarity, the matrix integral is defined element-wise since t is a scalar. We truncate the Magnus expansion to attain an approximation of the solution to a particular order. Taking just the first term Ω_1, we obtain

$$x(t) = \exp\left(\int_0^t A(\tau)\mathrm{d}\tau\right) x_0$$

which we can approximate with a first order method to get

$$x_{n+1} = \exp\left(hA(t_n)\right) x_n.$$

Note that we are given $A = A(t)$ in the definition of the Magnus expansion, while problems we are interested in have $A = A(x)$ or $A = A(x,t)$. The use of the Magnus expansion is in being able to express the solution to the time-dependent matrix ODE as something computable. By taking the first order term in the expansion and forming a first order approximation, we construct a first-order method which is the same as the "exponential Euler method" we saw earlier. This outlines an idea for constructing better approximations, by taking higher order Magnus expansions and higher order approximations.

In [4], the second order Magnus solution is given by

$$x(t) = \exp\left[\int_0^t A\left(e^{\tau A(x_0)}x_0\right)\mathrm{d}\tau\right]x_0$$

for a problem with initial condition $x(t=0) = x_0$. We can swap x_0 for x_n and this provides the Magnus solution to second order at $t_n + h$, which we then approximate to produce a numerical method. Using the trapezium method by evaluating the integrand at both ends and taking the midpoint, we obtain the method

$$x_{n+1} = \exp\left[\frac{h}{2}\left[A(x_n) + A\left(\exp\left[hA(x_n)\right]x_n\right)\right]\right]x_n.$$

Alternatively, using the midpoint rule the method is given by

$$x_{n+1} = \exp\left[hA\left(\exp\left[\frac{1}{2}hA(x_n)\right]x_n\right)\right]x_n. \tag{3.15}$$

We will consider the second order Magnus integrator as the approximation using the midpoint rule, which is the same options taken by the authors of [4]. The authors also provide more detail on the formulation for higher order methods. Both the trapezium and midpoint rule approximations are second-order accurate. More properties of the Magnus expansion are explored in [3].

The formulation of the second-order Magnus integrator (midpoint) is identical to the z_{n+1} stage of the second order Strang splitting method for one step [4]. See Equation 3.6 for the definition with Taylor-expanded form in Equation 3.8. See Equation 3.9 for the truncation error.

3.2.7 Example - The MAPK Cascade

The Mitogen-activated protein kinase (MAPK) cascade model is an autonomous system on six variables, given in the form of $\dot{x} = A(x)x$, where A is given by

$$A(x) = \begin{bmatrix} -k_7 - k_1x_2 & 0 & 0 & k_2 & 0 & k_6 \\ 0 & -k_1x_1 & k_5 & 0 & 0 & 0 \\ 0 & 0 & -k_3x_1 - k_5 & k_2 & k_4 & 0 \\ (1-\alpha)k_1x_2 & \alpha k_1x_1 & 0 & -k_2 & 0 & 0 \\ 0 & 0 & k_3x_1 & 0 & -k_4 & 0 \\ k_7 & 0 & 0 & 0 & 0 & -k_6 \end{bmatrix}.$$

The problem describes the behaviour of different chemical species involved in a biochemical reaction involving enzymes at a microcellular level [11]. The form of this problem is taken from [4], which itself is modified from the properties discussed in [11]. The coefficients k_j are given as $k_1 = 100/3, k_2 = 1/3, k_3 = 50, k_4 = 1/2, k_5 = 10/3, k_6 = 1/10, k_7 = 7/10$. Note that the values involving the term α show that this graph does not admit graph-Laplacian structure as we have defined. For $\alpha = 1$,

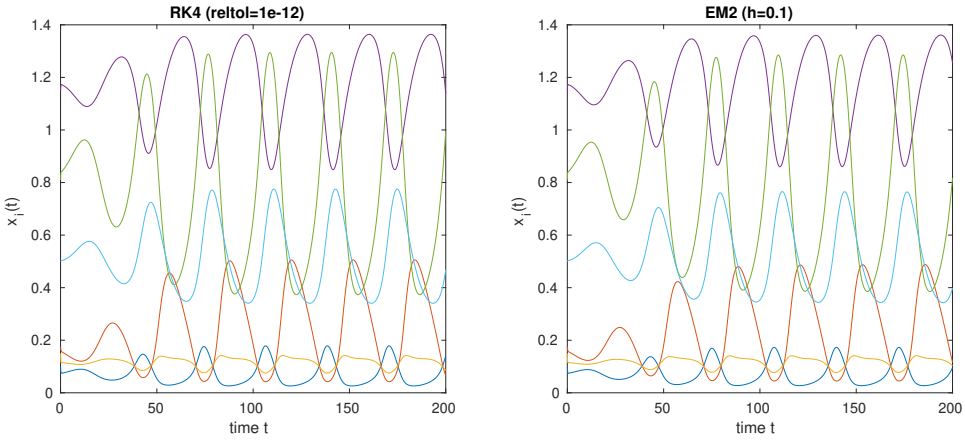

Figure 3.4: Graphic of the trajectories given by the MAPK cascade problem. The Runge-Kutta method applied by `ode45()` has a relative tolerance set to 10^{-12} which is extremely accurate. This gives us our best feasible approximation, to compare other methods to. This is shown in the left figure. The figure on the right shows the solution computed using the EM2 Magnus integrator, with a uniform step size of $h = 0.1$. The variables x_i are the concentrations of different chemical species involved in the reaction.

the first column sum is zero but the second is not, and for $\alpha = 0$ the opposite is true. Furthermore the fourth column sum is never zero since we have defined k_2 to be non-zero. This is addressed by [4], and we state that this problem adheres to the requirements of positivity preservation as long as $\alpha \in [0, 1]$. On inspection, this is the condition to require positivity of the entries involving α, which themselves are not on the diagonal. Hence this criteria is sufficient to guarantee positivity by the Theorem we stated earlier. We take $\alpha = 0.1$ for our computations, acknowledging that any value in $[0, 1]$ is suitable.

The MAPK cascade is an autonomous system which developes a sustained oscillation cycle. See Figure 3.4 for a visualisation of the behaviour of the system. We consider the EM2 integration method for solving this problem, with a time step of $h = 0.1$. We compare this with the output provided by MATLAB's `ode45()`, using a very low relative tolerance, which we consider the "exact" solution in a sense that we cannot obtain a better approximation. The behaviours of both are extremely similar, however the implementation of EM2 has a recogniseable error, which we can most easily notice when the paths of the values intersect. We can also make the assertion that EM2 preserves positivity for this problem, by inspection of the graph. This result is consistent if we extend the timespan of the solution.

Implementation of positivity preserving methods for this problem are provided in Appendix B.3.3

3.3 Approximation of the Matrix Exponential

This section involves some linear algebra material which we have not covered. A summary of the required information is given in the Appendix if needed.

3.3.1 Challenges, series computation

The positivity preservation methods discussed so far all have one problem in common, being the use of the matrix exponential. Computing the matrix exponential is significantly expensive, so we hope to introduce some form of approximation by which we can retain a given order of a method. We might first think to compute the exponential directly from the definition [16]:

$$\mathrm{e}^A = I + A + \frac{1}{2}A^2 + \frac{1}{6}A^3 + \ldots = \sum_{j=0}^{\infty} \frac{A^j}{j!}.$$

If we are working in finite precision floating point arithmetic, we can assume that at some point the series will converge such that the sums to n and $n+1$ are represented exactly the same [26]. In this case, we take the sum to n

$$\mathrm{e}^A \approx \sum_{j=0}^{n} \frac{A^j}{j!}.$$

and using a modified Horner's form [22] we can express this as

$$\begin{aligned}\sum_{j=0}^{n} \frac{A^j}{j!} &= I + A + \frac{1}{2!}A^2 + \ldots + \frac{1}{n!}A^n \\ &= I + A\left(I + \frac{1}{2}A\left(I + \ldots\left(\ldots\left(I + \frac{1}{n}A\right)\right)\right)\right).\end{aligned}$$

There are n matrix additions, $n-1$ divisions of a matrix by a scalar and $n-1$ matrix multiplications required to compute this expression. A multiplication of two $d \times d$ matrices requires $\mathcal{O}(d^3)$[1] floating point operations. Even with this improvement in efficiency, using this approach to compute a matrix exponential appears expensive. We will now show that it is also unstable.

The following example is an adjustment of the first demonstration from [26]. Consider the matrix given by

$$A = \begin{bmatrix} -121 & 60 \\ -160 & 79 \end{bmatrix}.$$

Computing the series form of the exponential directly in double-precision arithmetic sums to $N = 139$ terms, and gives us the solution

$$\mathrm{e}_s^A = \begin{bmatrix} -2.6531 & -97.1835 \\ -61.9581 & -184.0359 \end{bmatrix}.$$

The true form of A can be written as a conjugate transformation

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -41 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}^{-1}.$$

and so the exponential can be written as

$$\mathrm{e}^A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} \mathrm{e}^{-1} & 0 \\ 0 & \mathrm{e}^{-41} \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}^{-1} \approx \begin{bmatrix} -0.7358 & 0.5518 \\ -1.4715 & 1.1036 \end{bmatrix}.$$

Clearly the exponential computed via series approximation is not an adequate estimate.

Since this implementation is clearly inadequate, we look for alternative methods for computing the matrix exponential.

[1] For computer algebra operations, a lower order is better because we often deal with large problems. In comparison, we want numerical methods to have higher order because h is often very small.

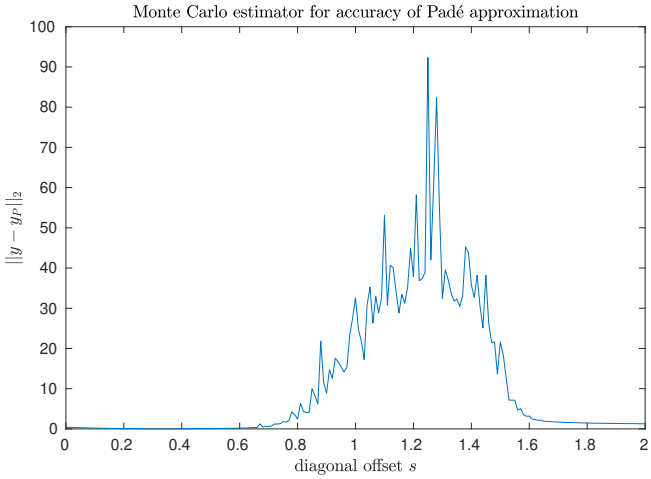

Figure 3.5: A Monte-Carlo estimate to evaluate the error of the Padé approximation of the matrix exponential. For each value of s in the interval $[0, 2]$, we generate a sample of random 10×10 graph-Laplacian matrix and positive vector pairs with entries uniformly distributed on $[0, 1]$, then evaluate the 2-norm error between $\exp(A)x$ using the MATLAB `expm()` function, and the Padé approximation $Q^{-1}Px$. However, we use the decomposition $A = \bar{A} + \hat{a}I$ and define $\hat{a} = -s|\max(a_{ii})|$. We use the Padé method to estimate $\exp(\bar{A})$ where its diagonal is modified from A depending on s. If $s \geq 1$ then $\bar{A}$ is entirely nonnegative, whereas if $s < 1$ then there is at least one negative entry on the diagonal of $\bar{A}$. This figure shows that there is an interval of values for s such that it is unsuitable to take the Padé approximation of $\bar{A}$. The rough shape of the curve could come from the randomness of the sample matrices and vectors themselves.

3.3.2 The Padé Approximation, Scaling and Squaring

Recall the Padé approximation to the matrix exponential, which we explored when looking at symplectic and A-stable Runge-Kutta methods in Chapter 2. We generalise this concept to functions on matrices. Given a function $f(A)$, there is a unique $[n, m]$ Padé approximant [16] $R_{mn}(A)$ given by the pair $P_{nm}(A), Q_{nm}(A)$ and formed by $R_{nm}(A) = [Q_{nm}(A)]^{-1} P_{nm}(A)$. We will refer to P as the numerator matrix and Q^{-1} as the denominator matrix. Formulae for Padé approximants for certain matrix functions are known, such as the exponential. When computing the product $Q^{-1}P$ we always use a method to solve the system $QX = P$ rather than directly computing the inverse and product.

We said that formulae for the Padé approximation for the exponential are known. From [26, 16] again, we have

$$\begin{aligned} P_{nm}(A) &= \sum_{j=0}^{n} \frac{(n+m-j)!n!}{(n+m)!j!(n-j)!} A^j \\ Q_{nm}(A) &= \sum_{j=0}^{m} \frac{(n+m-j)!m!}{(n+m)!j!(m=j)!} (-A)^j. \end{aligned} \tag{3.16}$$

This is the approximant at zero. A general Padé approximant is taken about a point, such that it is equal to the approximated function at that point, similar to a Taylor expansion. In some cases we

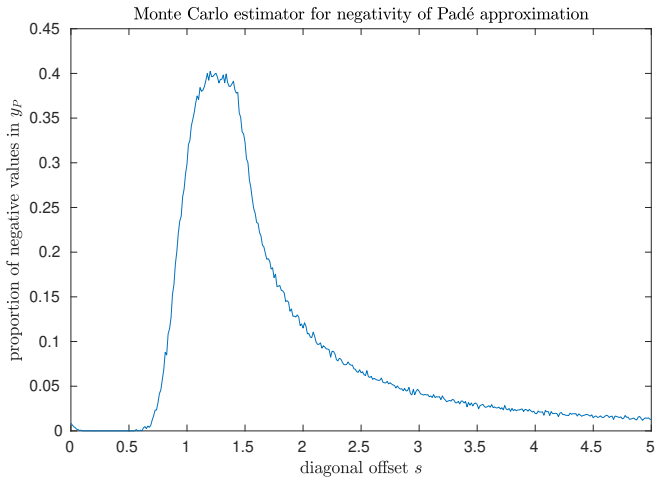

Figure 3.6: Monte-Carlo estimate for the negativity of the Padé approximation of $\exp(A)x$. Uses the same configuration as Figure 3.5, the only difference being the evaluation of the dependent variable. We can identify a region of small s for which the approximation appears suitable for preserving positivity. Not indentifiable from the previous figure is the slow descent as s increases, implying that these values are not suitable despite showing low error before. The improved smoothness of this curve could imply that the behaviour of the negativity is inherent to the approximation.

might take the approximant about a different point, but here zero is no less suitable than anywhere else.

In [4] it is stated that the second order diagonal Padé approximation to the exponential is positivity preserving. This is not true and we will provide a counterexample, alongside an investigation regarding how the approximation can be adjusted to preserve positivity. The $[1,1]$ Padé approximation to the exponential of A is

$$D_{11}(A) = \left[I - \frac{1}{2}A\right]^{-1}\left[I + \frac{1}{2}A\right].$$

Consider the matrix given by

$$A = \begin{bmatrix} -4 & 1 & 0 \\ 2 & -1 & 2 \\ 2 & 0 & -2 \end{bmatrix}$$

and vector

$$x = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}.$$

Clearly A is graph-Laplacian and x is positive. To four significant figures the product of the exponential with the vector is

$$\mathrm{e}^A x = \begin{pmatrix} 0.9422 \\ 3.8506 \\ 1.2071 \end{pmatrix} \tag{3.17}$$

so positivity is preserved. However if we compute the $[1,1]$ Padé approximant and compute the product we obtain

$$[D_{11}(A)]\,x = \left[I - \frac{1}{2}A\right]^{-1}\left[I + \frac{1}{2}A\right]x = \begin{pmatrix} -0.0167 \\ 1.1500 \\ 0.3667 \end{pmatrix}$$

where we fail to preserve positivity. A method is proposed in which we exploit the properties of the exponential by writing $A = \bar{A} + \hat{a}I$, such that $\bar{A}$ is entirely nonnegative. Then

$$\mathrm{e}^{A} = \mathrm{e}^{\bar{A}+\hat{a}I} = \mathrm{e}^{\hat{a}}\mathrm{e}^{\bar{A}}$$

and we compute the $[1,1]$ Padé approximant for $\bar{A}$. In our example $\hat{a} = -4$ and

$$\mathrm{e}^{\hat{a}}[D_{11}(\bar{A})]x = \begin{bmatrix} 0.5833 \\ 1.7500 \\ -0.8333 \end{bmatrix}.$$

So clearly the second order Padé method does not preserve positivity.

There is one result which we can utilise in order to ensure positivity.

Lemma 3.3 (Series Inverse [18])**.** If A is a matrix which satisfies $||A||_2 < 1$, assuming $I - A$ is invertible we can write the inverse of $[I - A]$ as

$$[I - A]^{-1} = \sum_{k=0}^{\infty} A^k.$$

Proof. The condition on the norm $||A||_2 < 1$ is required in order for the series to converge. Define the series up to N by U_N :

$$U_N := \sum_{k=0}^{N} A^k.$$

Then multiply $[I - A]$ by this series

$$\begin{aligned} U_N\,[I - A] &= \left[I + A + A^2 + \ldots + A^{N-1} + A^N\right][I - A] \\ &= [I - A] + \left[A - A^2\right] + \ldots + \left[A^{N-1} - A^N\right] + \left[A^N - A^{N+1}\right] \\ &= I + [A - A] + \left[A^2 - A^2\right] + \ldots + \left[A^N - A^N\right] - A^{N+1} \\ &= I - A^{N+1}. \end{aligned}$$

As $N \to \infty$, A^N tends to the zero matrix because $||A||_2 < 1$ as assumed. Hence this product converges to the identity and we have an expression for the inverse. Thus,

$$[I - A]^{-1} = \lim_{N\to\infty} U_N = \sum_{k=0}^{\infty} A^k.$$

□

This result is especially useful when considering the reduction $A = \bar{A} + \hat{a}I$. We have $\bar{A}$ is entirely positive by definition. If we also have this bound on the spectral norm of $\bar{A}$, then we know that the inverse of $[I - \bar{A}]$ is the series on powers of A, all of which must be positive. Therefore, the inverse of $[I - \bar{A}]$ is positive. In this context, we can guarantee that this approximation is positivity

preserving. This also relates to the Padé approximation - if this holds then the $[1,1]$ approximation has the numerator and denominator matrices both positive. We will look at this later.

A possible explanation for the mistake in [4] could be the alternative approximations the authors use. Considering the step h, the diagonal Padé approximation is the product of two matrices

$$\mathrm{e}^{hA} \approx \left[I - \frac{h}{2}A\right]^{-1} \left[I + \frac{h}{2}A\right].$$

The following will appear confusing due to the scaling of $h/2$ in both matrices, which is necessary to the Padé approximation of $\exp(hA)$. However all the comments are identical up to scaling.

We can guarantee positivity of the Padé approximation if both the numerator and denominator matrices are entirely positive. The denominator matrix is itself a first order approximation to the matrix exponential $\exp((h/2)A)$ and is in fact guaranteed to be positive (more on this later). However the numerator matrix has no guarantee of positivity: if $\hat{a}$ is the entry in A of maximum absolute value, then this matrix has at least one negative entry for $h > 2/|\hat{a}|$. For the example we have given, $\hat{a} = -4$ by inspection. Alternatively, recall $A = \bar{A} + \hat{a}I$ and consider the second order approximation involving only $\bar{A}$. This is the same as $\exp(A)$ up to scaling, as shown earlier. Since $\bar{A}$ is entirely nonnegative the numerator matrix must be nonnegative. However this time there is no condition on the positivity of the denominator matrix. We can give an informal proof: assume $\bar{A}$ is entirely positive, with at least one zero on the diagonal. Then consider $I - h\bar{A}$. This has at least one positive entry on the diagonal, but only has entirely positive diagonal for h less than $(\max(\bar{a}_{ii}))^{-1}$. Entries in $I - h\bar{A}$ not on the diagonal will be negative. Assuming the bound is *not* satisfied, there will be one row in $I - h\bar{A}$ which is entirely negative. For example,

$$I - h\bar{A} = \begin{bmatrix} 1 & - & - \\ - & - & - \\ - & - & ? \end{bmatrix}.$$

The '?' entry cannot be greater than 1, it is only included for generality. Now assume its inverse is entirely positive. This is the denominator matrix, so we are hoping this is true. Consider their product:

$$\left[I - h\bar{A}\right]\left[I - h\bar{A}\right]^{-1} = \begin{bmatrix} 1 & - & - \\ - & - & - \\ - & - & ? \end{bmatrix} \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix}$$

Consider their product. Because this is the product of a matrix and its inverse, it should form the identity matrix. However we assumed that the i-th row in $[I - h\bar{A}]$ is negative, and the i-th column in its inverse is positive (because we assumed the whole matrix is positive). Therefore the entry ii of their product, which is the identity matrix, is either zero or negative. This is a contradiction since entry ii of the identity matrix is 1. Therefore it is not guaranteed that $[I - h\bar{A}]$ to be positive.

The explanation for the error could lie in both approximations. If we are approximating $\exp hA$, then the denominator matrix is entirely positive, but we cannot say the same for the numerator. If we instead use the decomposition $A = \bar{A} + \hat{a}I$ and take the exponenial of $\bar{A}$, then the numerator matrix is guaranteed to be positive but the denominator may be negative.

See Figure 3.5 for a visualisation of the stability of computing the Padé approximation. By using different values of $\hat{a}$ in the decomposition $A = \bar{A} + \hat{a}I$ for the reduction $\exp(A) = \exp(\hat{a})\exp(\bar{A})$, the diagonal of $\bar{A}$ may or may not have negative values. Particularly, we define $\hat{a} = -s|\max(a_{ii})|$ using a positive scaling parameter s. If $s \geq 1$ we are using the method proposed by the authors in [4] to let $\bar{A}$ be strictly positive. We can clearly identify from the given figures that there are values of the scaling parameter for which the approximations are unsuitable, despite appearing as suitable from

its introduction in the paper. Figure 3.6 evaluates negativity, clearly isolating a region of scaling of $\hat{a}$ such that the approximation is accurate and positivity preserving.

In MATLAB, the matrix exponential can be computed using the `expm()` function. This implementation is very stable, due to the implementation of the scaling and squaring method [17, 26]. The implementation of scaling and squaring is motivated by the identity that

$$\mathrm{e}^{A} = \left[\mathrm{e}^{\frac{A}{r}}\right]^{r}.$$

The method takes r to be a given power of 2, say such that an approximation of $\exp(A/r)$ is stable by A/r having a norm bounded below a threshold [16], and then repeatedly squares the result to get an approximation for $\exp(A)$. This process provides much more stability in using the Padé approximation, or even the series definition, of the matrix exponential.

3.3.3 A first-order approximation

In our discussion of the Padé approximation we looked at guaranteed positivity of particular approximations. We will now discuss some stronger results, centred around the approximation

$$\mathrm{e}^{hA} = [I - hA]^{-1} + \mathcal{O}(h^2)$$

which is first-order accurate and guaranteed to be positivity preserving. First, note that when looking at the Padé approximation we considered the positivity of the approximation

$$\mathrm{e}^{hA} \approx \left[I - \frac{h}{2}A\right]^{-1}\left[I + \frac{h}{2}A\right].$$

What is interesting is that both Q^{-1} and P in this Padé approximation are themselves first order approximations of $\exp((h/2)A)$.

Understanding the positivity of this approximation involves Gershgorin's theorem for the eigenvalues of a matrix.

Theorem 3.4 (Gershgorin [10, 18])**.** *The eigenvalues of a matrix A lie in the union of the n discs $\mathcal{D}_i$ where*

$$\mathcal{D}_i = \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j=1, j\neq i}^{n} |a_{ij}| \right\}.$$

Proof. Proof omitted, see [18], pages $389 - 390$. □

Gershgorin's theorem is a very useful bound on the eigenvalues - the disc D_i is centered at a_{ii} and its radius is the sum of the absolute values of the non-diagonal entries in the rest of the column. This means that for a graph-Laplacian matrix, the discs containing the eigenvalues are centered at negative values on the real axis, and because a graph-Laplacian matrix has a zero column sum, these discs are restrained to the left half of the complex plane. Assume A has eigenvalues λ_i. Assuming $h > 0$, $I - hA$ has eigenvalues $1 - h\lambda_i$. Since the eigenvalues of the inverse matrix are the inverses of the eigenvalues,[2] the eigenvalues of $[I - hA]^{-1}$ are $1/(1 - h\lambda)$. From Gershgorin's theorem, the eigenvalues of A have negative real part, so these eigenvalues of the first order exponential approximation belong to the strictly positive half of the complex plane. We know that $[I - hA]$ has

[2]For matrix M and eigenpair μ, y, $y = M^{-1}My = M^{-1}\mu y$ and divide by μ.

positive diagonal entries and negative off-diagonal entries, and we can use Gershgorin's theorem to show that the diagonal of $[I-hA]^{-1}$ must be positive. The proof that this inverse is entirely positive is given in [4].

The positivity preserving methods from Equations 3.5, 3.15 retain positivity and second order when some of the matrix exponentials are replaced with this first order substitution. We evaluate the truncation error of the second order Magnus integrator when substituting the inside exponential with an approximation.

$$\begin{aligned}\tau_{\mathrm{m}}(h) &= x(t_n+h) - \exp\left[hA\left(\left[I-\frac{h}{2}A(x_n)\right]^{-1}x_n\right)\right]x_n \\ &= x(t_n+h) - \exp\left[hA\left(\left[\exp\left(\left[\frac{h}{2}A(x_n)+\mathcal{O}(h^2)\right]x_n\right)\right]x_n\right)\right]x_n\end{aligned}$$

We consider the exponential term and how the internal Taylor expansion brings the $\mathcal{O}(h^2)$ term out of A

$$\begin{aligned}&\exp\left[hA\left(\left[\exp\left(\left[\frac{h}{2}A(x_n)\right]x_n\right)+\mathcal{O}(h^2)\right]x_n\right)\right]x_n \\ &= \exp\left[hA\left(\left[\exp\left(\left[\frac{h}{2}A(x_n)\right]x_n\right)\right]x_n\right)+\mathcal{O}(h^3)\right]x_n\end{aligned}$$

We can take the $\mathcal{O}(h^3)$ out to show that this is clearly the regular second order Magnus method plus a remainder in h^3 and hence is second order. We have already looked at the order of the second order Magnus integrator in its original form.

3.3.4 A Proposed Method using Series Expansion

In this subsection and the next, we develop our own ideas from those provided by the authors in [4], aiming to use approximations to aid the computational cost of these numerical methods. We propose an approximation which can be used in the context of positivity preservation. Theoretically, the approximations should be flexible and positivity preserving. We will investigate the properties of using the series definition of the exponential, while also considering the priority of positivity preservation. First, recall for a graph-Laplacian matrix A the reduction $A = \bar{A} + \hat{a}I$ such that $\bar{A}$ is nonnegative. Therefore $\exp(A) = \exp(\bar{A}+\hat{a}I) = \exp(\hat{a})\exp(\bar{A})$. In the general case, distribution of the matrix exponential involves the commutator [16], however the identity matrix commutes with any other square matrix of the same size.

Then, we approximate this matrix exponential using the series definition

$$\exp(\bar{A}) \approx \sum_{k=0}^{N}\frac{1}{k!}A^k =: T_N(\bar{A}).$$

Clearly T_N is an N-th order approximation to the matrix exponential. Therefore

$$\exp(A) = \exp(\hat{a})T_N(\bar{A}) + \mathcal{O}(h^{N+1}).$$

However, we apply the approximation to both the matrix and the scalar exponentials, so our approximation takes the form

$$\exp(A) = T_N(\hat{a})T_N(\bar{A}) + \mathcal{O}(h^{N+1}). \tag{3.18}$$

Furthermore, since $\bar{A}$ is entirely positive, we would expect the approximation to be positivity preserving. We acknowledge that we have given notable attention earlier to the impracticality of using

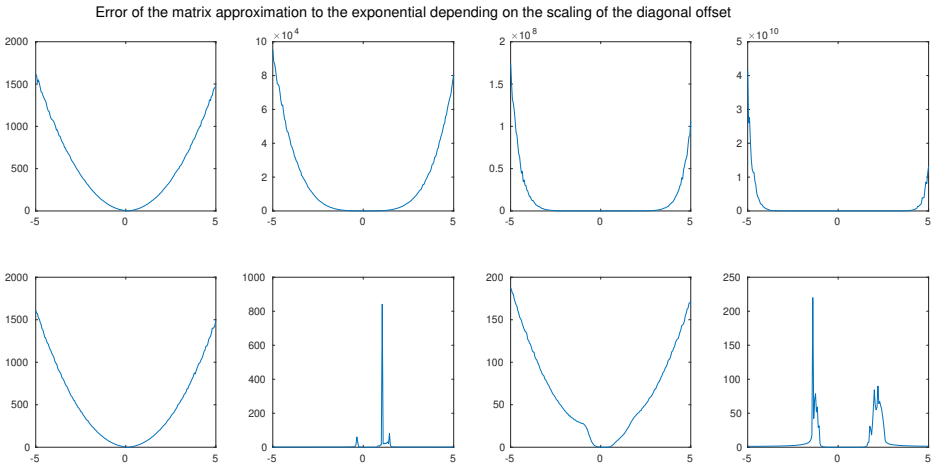

Figure 3.7: Comparison of how the scaling parameter for the diagonal offset, as seen in Figure 3.5 affects the error of the approximation to the matrix exponential. The vertical axis is the evaluation of $||y - \bar{y}||_2$ where $\bar{y}$ is a Monte-Carlo estimate of $y = \exp(A)x$ using random 10×10 linear systems. The true value y is evaluated using `expm()`. Top row: approximation using the series definition of the matrix exponential. Bottom row: approximation using the Padé method. Column 1: first order (Padé $[1, 0]$) approximations. Column 2: second order (Padé $[1, 1]$) approximations. Column 3: fifth order (Padé $[3, 2]$) approximations. Column 4: tenth order (Padé $[5, 5]$) approximations. The diagonal Padé methods maintain stability for large offset.

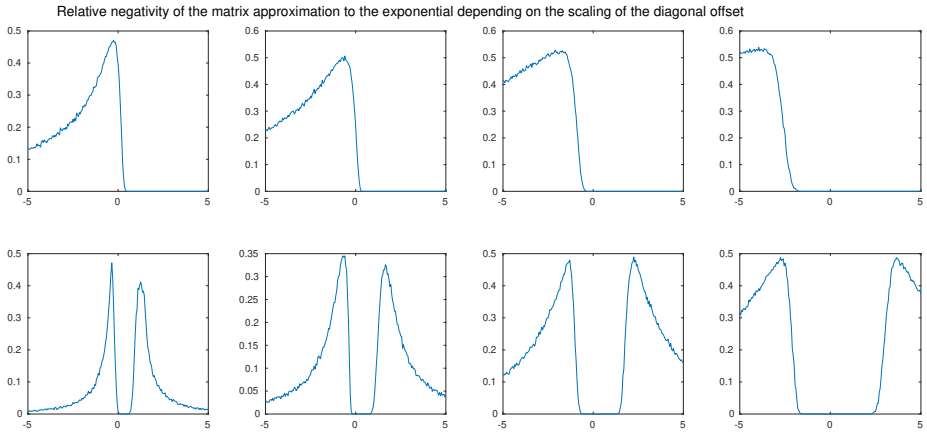

Figure 3.8: Results exploring relative negativity of the approximation of the matrix exponential, as in Figure 3.7. Top row: approximation using the series definition of the matrix exponential. Bottom row: approximation using the Padé method. The orders of approximations used are not the same as the previous figure, since we stick to diagonal Padé approximations. Column 1: second order (Padé $[1, 1]$) approximations. Column 2: fourth order (Padé $[2, 2]$) approximations. Column 3: tenth order (Padé $[5, 5]$) approximations. Column 4: twentieth order (Padé $[10, 10]$) approximations. The diagonal Padé methods maintain stability for large offset.

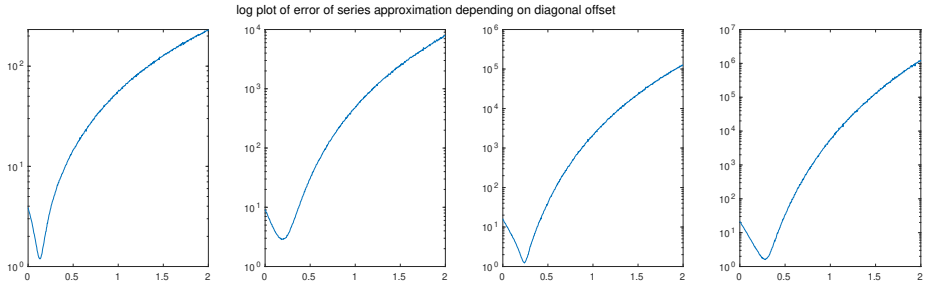

Figure 3.9: Further visualisation of the error of the series approximation to the product of matrix exponential and vector. Logarithmic vertical axis is used for clarity. From left to right: error of the approximation of orders 1 to 4. Using the decomposition $A = \bar{A} + \hat{a}I$, where $\hat{a} = -s|\max(a_{ii})|$, we approximate the matrix exponential using a separation. There is clearly always a minimiser over s depending on the order of the method used which gives the best convergence. Again using a Monte Carlo estimator for 10×10 random systems

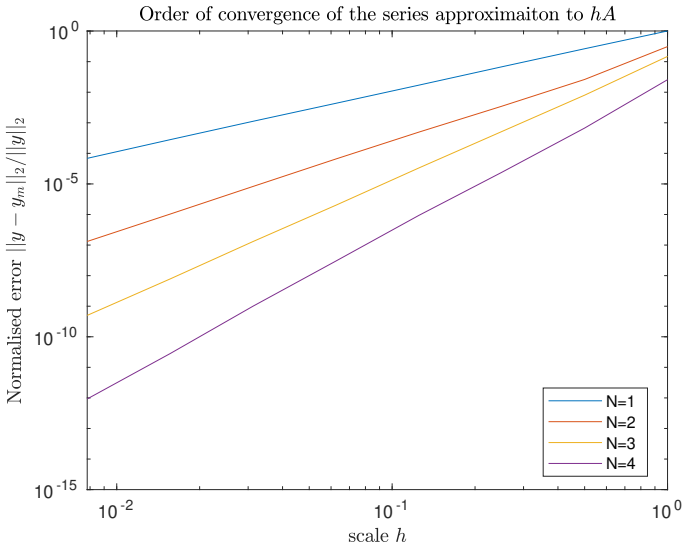

Figure 3.10: Order of convergence of the series approximations of $\exp(hA)x$ depending on h, tested on samples of random 4×4 systems. The diagonal offset used was $s = 0.5$, which as shown in Figure 3.9 appears to be close to a minimiser for low order approximations.

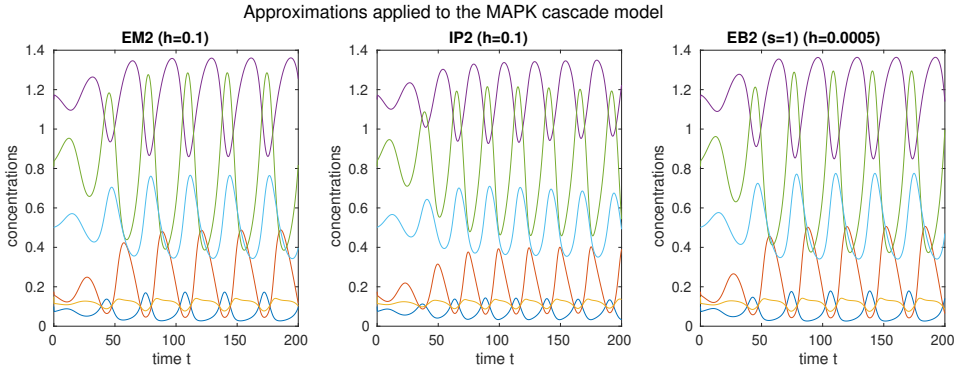

Figure 3.11: Visualisation of the second order Magnus integration when different approximations to the matrix exponential are used. Left: EM2, using two matrix exponentials with `expm()`. Middle: IP2, using a first order positivity preserving approximation and a second order Padé. Right: EB2, using first and second order approximations to the exponential from the series definition.

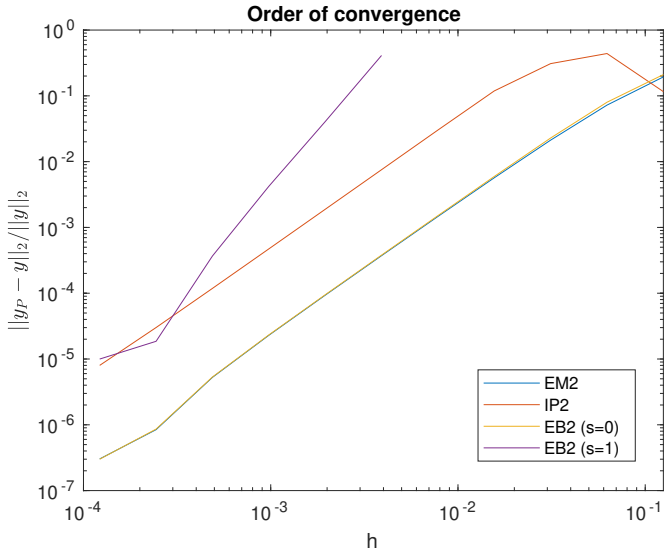

Figure 3.12: Order of convergence for EM2, IP2 and EB2 integration methods. We plot the relative error of the solution at end time T against the step size h used in the integration scheme. The EM2 method is second order accurate [4], and IP2 is also second order since their gradients match. EB2 is tested for both $s = 0$ and $s = 1$, but is only theoretically positivity preserving for $s = 1$.

the series expansion on its own, and now we are using the series expansion in our approximation. We will return to this point later. See Figures 3.7 and 3.8, evaluating the behaviour of approximations to the matrix exponential and vector product. As we mentioned earlier, the computation of the matrix exponential by a series expansion can be very unstable, which we observe here. This brings attention to the first problem with this approach: the series approximation by itself is fairly poor for low order approximations, compared to Padé.

Consider the example given at the beginning of the discussion of the Padé approximation. We have the matrix

$$A = \begin{bmatrix} -4 & 1 & 0 \\ 2 & -1 & 2 \\ 2 & 0 & -2 \end{bmatrix}$$

and vector

$$x = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}.$$

We evaluate $y = \exp(A)x$ in MATLAB using `expm()` and provide an approximation $y_m = T_N(\hat{a})T_N(\bar{A})x$ given the required order N. We have visualised this approximation in Figure 3.9, where we can identify positions of optimality for the approximations depending on s. The minimum region of values for s required for optimal error is always strictly less than 1. Note that this leads to another problem: in order for the method to preserve positivity in theory, we require $s \geq 1$, however the optimal values of s for our lower order approximations do not satisfy this constraint. Despite this, we continue. Instead of comparing the convergence of different order approximations to the matrix exponential, it may be insightful to apply these methods to an integrator and compare with the methods and

approximations have already been established.

Note the results given in Figure 3.10. The convergence of these methods follow steeper gradients as N increases as the remainder in $\mathcal{O}(h^N)$ decreases. This figure generates a random graph-Laplacian matrix A and random vector x, and then evaluates a relative error $||y - y_N||_2/||y||_2$, where $y = \exp(A)x$ and $y_N = T_N(A)x$. We can deduce that the error should be $\mathcal{O}(||A||^{N+1})$ for some subordinate matrix norm on A. Figure 3.10 also uses a value of the scaling parameter s which we recognise does not guarantee positivity preservation, instead it has been chosen to improve the approximation in a sense of relative error. This demonstrates a third problem with using the series approximation: the approximation is unlikely to converge for low order N, and the norm of the matrix plays a large role in the error of the approximation. Increasing the scaling parameter s means we are taking the exponential of a matrix with a larger diagonal, and hence with a larger spectral norm. Therefore if we increase s to ensure positivity, our approximation will be, in a sense of relative error, worse.

We will now evaluate an experiment, considering how we can apply approximations to the matrix exponential to numerical methods. We will consider three methods, all of which are variations of the EM2 Magnus integrator, using

- EM2: Two matrix exponentials, computed using `expm()`.
- IP2: One first order approximation using $[I - hA]^{-1}$ and one second order diagonal Padé approximation.
- EB2: One first order and one second order approximation using the series method proposed, with scaling parameter chosen to consider positivity preservation.

We will examine each of these methods applied to the MAPK cascade.

For IP2, we used the reduction $A = \bar{A} + \hat{a}I$ with $a = -s|\max(a_{ii})|$ and chose $s = 1$ to adhere to the method outlined by [4], namely that this should be positivity preserving. In our discussion on the Padé approximation we showed why positivity is not necessarily preserved. For the EB2 method, we used first and second order series approximations with both $s = 0$ and $s = 1$. From our testing, $s = 0$ meant the approximation was more usable but not unconditionally positivity preserving. We found that increasing s makes the computation of EB2 far more expensive, and unusable for moderate h. It would not be useful to consider negative values of s since we have discussed that this is not appropriate for positivity preservation. First, see Figure 3.11. We show the behaviour of the integration from EM2, IP2 and EB2 on separate figures. The EM2 and IP2 integrations appear similar, which follows from the results concluded by [4], being that the EM2 method is second order accurate. Furthermore, the IP2 method retains the second order accuracy of EM2, since it uses a positivity-preserving first-order approximation, followed by a second order Padé approximation. It is important to recall that we have shown that this Padé approximation is not positivity preserving. The EB2 method, using series exponentials, is surprisingly accurate to EM2.

Finally, observe Figure 3.12, where we show the convergences of the methods. To compute the order of convergence of these methods, we obtain a solution to the problem using `ode45()` with an extremely low error tolerance, which we refer to as the master solution. We integrate the system up to time $T = 200$ and store the final value. Then, given a value of h, we integrate the system again using a positivity preserving method. We repeat this for a range of values of h decreasing in magnitude in order to plot the rate of convergence. We use the vector 2-norm to obtain a relative error given by $||y_P - y||_2/||y||_2$, where y is our master solution and y_P is our approximation. This provides sufficient data to visualise the order of convergence of a numerical method of our choice. We repeat this for all the methods we wish to analyse.

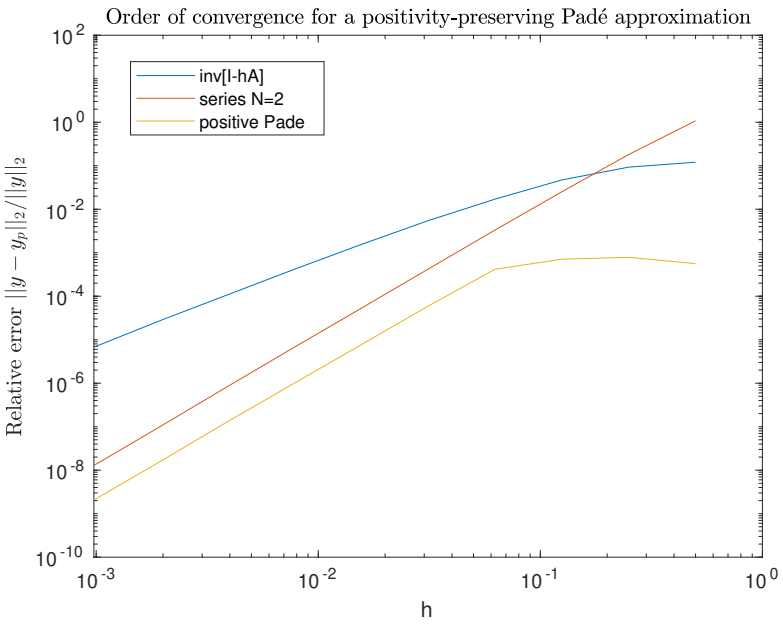

Figure 3.13: Order of convergence for the positivity preserving modification to the $[1,1]$ Padé approximation to the matrix exponential. The order matches a given second order approximation. The construction of this approximation guarantees positivity.

Since EM2 and IP2 are both second order, they converge to the "exact" solution at the same rates. We have graphed EB2 choosing both $s = 0$ and $s = 1$. For $s = 0$, the approximation is not guaranteed to preserve positivity, but the approximation follows the order of EM2 closer than the IP2 approximation. Choosing $s = 1$ means we are approximating the exponential in a way that guarantees positivity. However, the approximation becomes prohibitively expensive, and IP2 is a better method for timesteps $h > 10^{-3}$. In fact, for timesteps $h > 10^{-2}$ the error of the approximation is so large that the method is unusable - the error is outside the visible region of the graph. For extremely small timesteps $h \approx 10^{-4}$ EB2 is only ever as good as IP2. Essentially, the error between EM2 and EB2 is that we can save on computing matrix exponentials with EB2, but we may need to perform a thousand timesteps in the place of one.

The main problem with EB2 is that when we apply the reduction $A = \bar{A} + \hat{a}I$ and start working with the diagonal offset parameter s, we want $s \geq 1$ in order to guarantee positivity, but optimal error is attained out of this bound (recall Figure 3.9). This problem gets worse when we look at how the diagonal offset affects $\bar{A}$, because if s is sufficiently large then it scales with the spectral radius of $\bar{A}$, and taking a low order sereis approximation of the exponential of this matrix is going to be less accurate as this increases. In simpler terms, by using the series approximation and being given control of the diagonal offset, it is challenging to have both positivity and convergence.

3.3.5 A Proposed Improvement to the Padé Approximation

Developing from before, we will consider using more robust techniques to again improve the cost of a numerical integration scheme from [4]. The following method applies many results that we

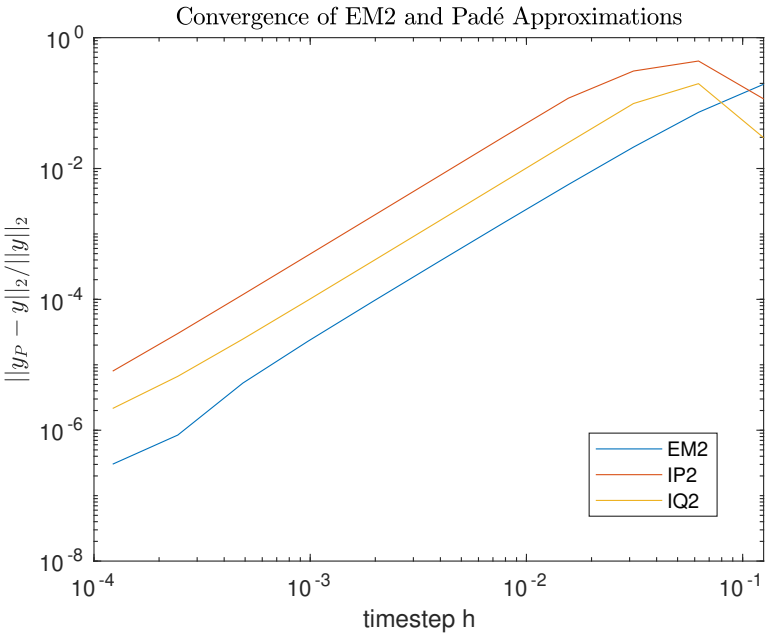

Figure 3.14: Methods EM2, IP2 and IQ2 applied to the MAPK problem, evaluating the global error as a function of the timestep. All methods are clearly second order, however EM2 uses directly computed matrix exponentials from `expm()`, while IQ2 uses positivity-preserving approximations. IP2 is as we have defined earlier. EM2 solution computed in 196.7 seconds. IP2 solution computed in 324.5 seconds. IQ2 solution computed in 346.0 seconds.

have explored in this section. First, the construction of the Padé approximation. Second, the implementation of scaling and squaring. Third, the decomposition $A = \bar{A} + aI$. Finally, Lemma 3.3 on the positivity of the inverse $[I - A]^{-1}$.

We propose a numerical integration scheme which is another modification of EM2, except using two matrix exponential approximations which are positivity preserving. This method is identical to the IP2 method we introduced in the previous section, except for the modification of the second order Padé approximation. Hence this is the change that we will discuss here.

Recall Lemma 3.3 - clearly scaling is useful. Then using the Padé approximation,

$$\begin{aligned} \mathrm{e}^{hA} &= \left[\mathrm{e}^{\frac{hA}{2^m}}\right]^{2^m} \\ &= \left[\left[I - \frac{hA}{2^{m+1}}\right]^{-1}\left[I + \frac{hA}{2^{m+1}}\right] + \mathcal{O}(h^2)\right]^{2^m} \\ &= \left[\left[I - \frac{hA}{2^{m+1}}\right]^{-1}\left[I + \frac{hA}{2^{m+1}}\right]\right]^{2^m} + \mathcal{O}(h^2) \end{aligned}$$

so clearly this is a valid approximation method.

Our approximation method is as follows. First, we deconstruct A as per $A = \bar{A} + \hat{a}I$, as we have looked into earlier. We ignore any scaling parameter in this choice, implicitly taking $s = 1$ by having $\hat{a}$ being the most negative entry on the diagonal of A. We can then approximate the exponential by taking $\exp(A) = \exp(a)\exp(\bar{A})$ and approximating the exponential of $\bar{A}$. We can choose to approximate the scalar or compute it directly, as computing the scalar exponential is not considered computationally expensive - we have found there is not a significant error in either case. For the Padé $[1, 1]$ approximation of $\bar{A}$, we can guarantee the numerator matrix will be positive regardless of its scaling. In order to ensure positivity of the denominator matrix, we find a scaling 2^m such that $\bar{A}/2^m$ satisfies the bound required in Lemma 3.3. There is a trick here. In order to satisfy the bound, we need to know the 2-norm of $\bar{A}$. However, we know that $||A||_2 < \sqrt{d}||A||_1$ in general [18], where d is the dimension of the matrix. The 1 norm is the maximum column sum of the matrix. However, since A is a graph-Laplacian matrix, the maximum column sum of $\bar{A}$ is $\hat{a}$ by construction. Hence we don't need to make any computations on the properties on the matrix. We compute the Padé approximation for the scaled-down matrix, then repeatedly square to return to the original scale. This gives us a $[1, 1]$ Padé approximation which guarantees positivity.

This is a simplified version of the actual method implemented by MATLAB to compute the matrix exponential. Cost is reduced by computing the $[1, 1]$ approximation, and by not needing to compute any matrix norms.

We have shown the results of this implementation in Figures 3.13 and 3.14. The former indicates that the matrix approximation is itself second order accurate, while the latter indicates that EM2 maintains second order accuracy when this approximation is implemented. Rate of convergence is computed again by comparing the final values of the solution to that of a master solution, obtained using `ode45()` method with extremely low tolerance, and computing a relative error in the 2-norm. We can see that IQ2 retains second order accuracy. The IQ2 method is unconditionally positivity preserving, unlike the IP2 method proposed by the authors in [4], as we have explored earlier. Furthermore, we can see from Figure 3.14 that the IQ2 method provides a closer approximation than IP2.

Note that in Figure 3.14 we have stated the computation times of both methods for solving the IVP. Despite our implementation taking longer than the internal function, this is expected since the

core included functions for MATLAB run using compiled FORTRAN code which is much faster in general than the interpreter for MATLAB itself [25].

Implementation of our positivity preserving Padé approximation in integration schemes is given in Appendix B.3.2.

3.4 Wider Positivity Preservation

3.4.1 The Implicit Euler Method

Consider a system defined in its most general form by $\dot{x} = f(t, x)$. Let y be the solution to the backward Euler method, so given $x = x_n$ we require $y = x_{n+1}$ to satisfy.

$$y = x + hf(t, y)$$

The following result shows that this method is positivity preserving.

Theorem 3.5 (Hundsdorfer, Verwer (2003) [20])**.** *The implicit Euler method is unconditionally positivity preserving for any positive step size $h > 0$.*

Proof. The expression for the method is $y = x + hf(t, y)$, which we assume is continuous depending on h, where t and x are fixed. This does require the continuity of f. We write this method as a function $y(h)$, where the result depends on a parameter h for fixed t and x.. Our aim is to show that if x is nonnegative then so is $y(h)$ for all h, however it is sufficient to prove this for a given h. This is because y is continuous on h, and h is chosen arbitrarily, so y will stay within the positive region. Assume that, given some positive h_0, we have that $y(h) > 0$ for $h < h_0$, except for the i-th entry where we assume $y_i(h_0) = 0$. Then we have

$$0 = y_i(h_0) = x_i + h_0 f_i(t, y(h_0)).$$

The coefficient h_0 is positive, and we assumed v_i is nonnegative. In order for the problem to preserve positivity we require that $f_i(t, y(h_0))$ be nonnegative. In the case that $x_i = 0$ and the same for f_i, then $y_i = 0$ so the i-th entry in the solution remains at zero. Otherwise, we have a contradiction, meaning that if x is positive then the implicit Euler method preserves positivity. □

The proof, as given in the textbook [20], causes some confusion with the implementation of the implicit Euler method at a fixed t. If it was instead given for some evaluation of $f(t, y(h))$ where $t = t_0 + h$, the process would be more clear since we are clearly computing f at the time t_{n+1} and value $y = x_{n+1}$ of the solution.

The result lends some similarity to when we discussed positivity preservation of a problem governed by a graph-Laplacian matrix in Theorem 3.2. The assumption of the behaviour of the derivative at zero is key to the result, since this is the behaviour that preserves positivity. When discussing earlier in this chapter, we used the result in a linear case, but have now been able to show it generally. Interestingly, the connection between these two results is even more evident when we consider how this does in fact apply to problems of the form $\dot{x} = A(x)x$, where A is a graph-Laplacian matrix. The implicit Euler method applied to this problem is

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

which rearranges to

$$x_{n+1} = \left[I - hA(x_{n+1})\right]^{-1} x_n$$

which we know is positivity preserving. The positivity preservation of this matrix can be seen as another case of this general result on the behaviour of the backward Euler method.

It is widely recognised that there are no methods which, when applied to generally formulated problems, preserve positivity unconditionally and also have truncation error of order greater than 1. This result is stated in [5], and referenced in [4, 20] and others.

3.4.2 Review

We have given an overview on current methods in positivity preservation. The focus of our analysis has been the formulation and improvement of second-order methods. This contrasts to symplectic and more general geometric integrators, where we have fourth and higher order schemes available. In a sense, this is one limitation of the analysis we have provided, namely that we have studied improvements over current methods, but have not provided any framework for how we would formulate higher-than-second-order methods. This itself is still an active area of research.

Having finished our analysis of positivity preservation, we provide a discussion to review all the topics from this report.

Chapter 4

Review and Discussion

4.1 Improvements

4.1.1 Wider Geometric Integration

There are many facets of geometric numerical integration which we have not covered in this report. Many of our results on symplectic integration borrow from the book by Hairer, Lubich and Wanner [13], which covers the topic in far more depth. We have covered the results that we believe to be key for understanding symplectic integrators. However, there are properties of symplectic methods, and approaches for constructing methods we have explored, which lend themselves to different disciplines within the broader subject.

One of these topics is time-symmetry. First, recall the definition of the adjoint flow map. From Chapter 2, when forming methods, if we have a numerical flow denoted Φ_h, the adjoint is the map defined $\Phi_h^* = \Phi_{-h}^{-1}$. A method is time-symmetric if it is self-adjoint, that is $\Phi_h^* = \Phi_h$. The definition of the adjoint means this identity can be written equivalently as $\Phi_h^{-1} = \Phi_{-h}$. If we have a general method Φ_h, then the composition $\Phi_{h/2} \circ \Phi_{h/2}^*$ is time-symmetric. We can prove this by composing a step of the method in h with a step in $-h$ as follows

$$\begin{aligned}
\left(\Phi_{h/2} \circ \Phi_{h/2}^*\right) \circ \left(\Phi_{-h/2} \circ \Phi_{-h/2}^*\right) &= \Phi_{h/2} \circ \left(\Phi_{h/2}^* \circ \Phi_{-h/2}\right) \circ \Phi_{-h/2}^* \\
&= \Phi_{h/2} \circ \left(\Phi_{-h/2}^{-1} \circ \Phi_{-h/2}\right) \circ \Phi_{-h/2}^* \\
&= \Phi_{h/2} \circ \mathrm{I} \circ \Phi_{-h/2}^* \\
&= \Phi_{h/2} \circ \Phi_{-h/2}^* \\
&= \Phi_{h/2} \circ \Phi_{h/2}^{-1} \\
&= \mathrm{I}.
\end{aligned}$$

where I is the identity map. Each step only follows from associativity or the definition of the adjoint. Therefore,

$$\left(\Phi_{h/2} \circ \Phi_{h/2}^*\right)^{-1} = \left(\Phi_{-h/2} \circ \Phi_{-h/2}^*\right)$$

hence the method is self-adjoint. Recall that the Störmer-Verlet method, which we constructed in this way, is time-symmetric.

Methods for preserving time-symmetry have their use in the same fields as symplectic integrators [15]. The property means that if we integrate forward in time to a certain point, we can integrate backwards in time using our last result as an initial condition, and we will return to the original starting value. Time-symmetric methods[1] are able to accomodate variable time-steps, unlike symplectic methods [13], hence they are popular choices in particle physics and related fields, where problems involve precise integration of oscillatory systems.

4.1.2 Properties of Symplectic Methods

Our focus on symplectic methods was developed from the assumption that its qualities come from the Hamiltonian structure. Since the symplectic identity is inherent to the model problem being Hamiltonian, a lot of analysis can be performed on the relationship between the symplectic integrator and the Hamiltonian system. However, this motivates us to analyse a symplectic integrator in the context of a Hamiltonian problem, and not as a general method. For our positivity preserving methods, we analysed and justified the truncation errors of methods, whereas these properties were not explored in as much depth for symplectic integration. This could be justified by the reason that this field is much more complete, and the properties of these methods are much more understood in the field of numerical analysis, hence we do not feel the need to explain these results. However, there are many general properties of symplectic methods which we have not explored, for example the merits to slower growth in error [13].

4.1.3 Computations in Positivity Preservation

The methods we have explored for positivity preservation are second order at best. This motivates the development of higher-order methods in this field, which is not something we have explored. We have, however, given an introduction to the Magnus expansion, the form for which can be used for formulating methods. This is explored in more detail by the authors in [4], but we have not included it. They introduce a third-order method, however it does not guarantee positivity unconditionally and requires seven matrix exponentials.

To design a third-order method, for example, we would first consider the expression of the solution $x(t) = \exp(\Omega(t))x_0$, and write a third-order (in A) Magnus expansion. We would then design our method as a third-order (in h) positivity preserving approximation to this expansion.

For the methods we have explored, the problem remains of computing the matrix exponential. We were able to formulate a positivity preserving method which only approximated these exponentials, however the depth of this subject goes much deeper than we have explored. One such approach would be Krylov methods for approximating the matrix exponential [24]. Were this a project on numerical linear algebra, this would be an interesting avenue to explore. However, the methodology itself is very different to the approximation methods we have focused on. We have decided instead to focus on the approximation methods seen, and develop these effectively for use in our problems.

4.2 Review

4.2.1 Summary of Symplectic Integration

In Chapter 2 of this project, we explored symplectic integration in the context of Hamiltonian dynamics. Our goal was to explore symplectic integration methods and understand how they are

[1] Time-symmetric methods which are not also symplectic

constructed. We showed symplecticity of a small selection of methods. The implicit midpoint and Störmer-Verlet schemes are both symplectic second-order methods, however their structures are inherently different, with the Verlet scheme constructed using the symplectic Euler method, a particular modification for Hamiltonian systems, while the implicit midpoint scheme is formulated as a second order implicit method, which is symplectic when applied to a Hamiltonian problem.

We looked at results for symplectic Runge-Kutta methods, in order to expand the theory to methods capable of arbitrary order. With the statement that symplectic Runge-Kutta methods need satisfy the statement of Theorem 2.18, we are able to justify the symplectic nature of arbitrary methods. These RK methods are necessarily implicit, building further on our understanding of the applications for implicit methods. The Störmer-Verlet scheme is enticing because it is explicit for a separable Hamiltonian, but we can understand that this may not be the case in general. Furthermore, these RK methods preserve quadratic invariant quanitites, the necessary condition for symplecticity. Outside of symplectic integration, preservation of invariant quanitites is its own area of study.

Finally, we studied a fundamental result on how the numerical solution can be thought of as an exact solution to a modified problem. We considered the relationship between the Hamiltonian of the modified problem, and that of the original. We were able to state and prove that the order of the numerical method is the order of closeness between these two quantities. This gives us two ways of thinking about the numerical solution. First is that if the method is convergent, then the numerical solution will become more accurate as the step size decreases. Therefore, a higher order method will converge to this solution faster. On the other hand, if we decrease the step size then we also know that the modified Hamiltonian will converge closer to the original. A higher order symplectic method means that this modified Hamiltonian will converge faster. Either way, we converge to the same result. We get the true qualitative behaviour.

The example of the gravitational three-body problem is an excellent demonstration of the motivation for symplectic integration. The symplectic method preserves the trajectories of the orbit, while the general purpose explicit RK method undergoes drift, inherent change of a quality of the system. This is the qualitative preservation of symplectic integration. When a symplectic integrator is used in its many applications, the quality is an inherent requirement of the solution.

4.2.2 Summary of Positivity Preservation

For our exploration of positivity preservation, we aimed to introduce the key methods introduced by the authors of [4], explore their results on inherent positivity, and suggest modifications to the methods.

To start, we introduced the concept of the graph-Laplacian matrix and showed its role in the preservation of positivity for the solution to the initial value problem. This was necessary in order to introduce numerical methods for solving the problem while preserving positivity. A very important result was how the matrix exponential of a graph-Laplacian matrix applies to positivity preservation. We know that if the problem is governed by a graph-Laplacian matrix, then the solutions preserve positivity. We also know that if the problem is governed by a constant matrix, we can write the solution in terms of the matrix exponential. Therefore, if we can break the problem into separable components involving constant graph-Laplacian matrices, then we know we can use the matrix exponential to produce a solution that preserves positivity. This is the approach we justified when exploring the second order Strang splitting (ES2) method. We also covered the second order Magnus (EM2) integrator, covering the theory of the Magnus expansion as the solution to a time-dependent system of ODEs. Despite not exploring the theory in detail, we included the information on the construction of the Magnus expansion for completeness.

Unlike our study on symplectic integration, a large part of our motivation for studying positivity

preservation is the potential to develop new contributions to the field. This could be explored in two ways: first, by constructing higher order, cheaper-than-expected, numerical methods for positivity preservation, or by optimising current methods using approximation theory, while maintaining the positivity preserving nature. We chose the second option, exploring the theory of different methods for approximating the matrix exponential, while restricting our options to those which still unconditionally preserve positivity. We looked at both the series and Padé approximations to the matrix exponential, exploring the differences between the two, and ended up with two proposed modifications to the EM2 method. The first option replaced a matrix exponential with a second order series approximation, chosen up to scaling in order to guarantee positivity. However, error bounds on the series approximation meant that the method, while being second order, required an extremely small time step in order to work at all. The second option was to use a second order Padé approximation, with the implementation of scaling and squaring in order to maintain stability. We found that this method worked, as a second order positivity preserving integrator with only first and second order approximations to the matrix exponential. The implementation of this method in its current state is slower than the regular implementation, however the process is, in theory, much more optimised.

4.3 Conclusion

If we let h be the difference from one real number to the next, then any convergent method with step size h is symplectic, positivity preserving, and arguably the perfect geometric integrator. Unfortunately, this is a rather difficult parameter to achieve. Instead, we have to settle for a geometric integration scheme, based on the quality we want to preserve. For the problems we have explored, this requires rewriting the problem in a form that inherently admits this quality on its own - we cannot have a symplectic integrator without a Hamiltonian. Once we have a problem as we want it, we can pick an integration method. We have looked at this two ways. First, for symplectic integration we have explored some of the available options and demonstrated their qualities. A lot of our examples lend themselves to the Störmer-Verlet method, which we hope to show is an effective choice as a general symplectic integrator. However, we took a different approach to positivity preservation by analysing and adjusting the methods available. We spent a large portion of our writing exploring the numerical analysis of approximations to the matrix exponential, in order to reduce the cost of the integration methods while maintaining accuracy. We hope that these proposed adjusted methods are valuable in the preservation of positivity.

The reader should have developed an understanding of the motivation and the execution of geometric numerical integration. We have shown examples of how qualities, inherent to a system, are not respected by general-purpose methods, particularly in the case of symplectic integration. In turn, we have explored methods which are centred around the aim of quality preservation. Both facets, both qualities we have explored have clear applications in the natural sciences, such as Hamiltonian systems for rigid body dynamics, and positive systems for chemical kinetics. However, at any point we may question the necessity of a geometric integrator. Despite being able to achieve qualitative preservation, we lack the ease of adjustment found in general purpose integrators. Willing to save effort, we may think that a general purpose method will be fine if we just set the error tolerance low enough. We hope to have shown that this is not the argument that geometric numerical integration aims to win, rather we aim to construct a different kind of integrator whose primary purpose is qualitative preservation. Fortunately, they also provide a good approximation to the solution.

Bibliography

[1] Mar. 2024. URL: https://en.wikipedia.org/wiki/Laplacian_matrix.

[2] *Bernoulli numbers*. @url = https://encyclopediaofmath.org/index.php?title=Bernoulli_numbers, Accessed: 03-04-2024.

[3] S. Blanes et al. "The Magnus expansion and some of its applications". In: *Physics Reports* 470.5-6 (Jan. 2009), pp. 151–238. DOI: 10.1016/j.physrep.2008.11.001.

[4] Sergio Blanes, Arieh Iserles, and Shev Macnamara. "Positivity-Preserving Methods for Ordinary Differential Equations". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 56 (2022).

[5] Catherine Bolley and Michel Crouzeix. "Conservation de la positivité lors de la discrétisation des problèmes d'évolution paraboliques". In: *RAIRO. Analyse numérique* 12.3 (1978), pp. 237–245.

[6] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[7] N. Broekhuizen et al. "An improved and generalized second order, unconditionally positive, mass conserving integration scheme for Biochemical Systems". In: *Applied Numerical Mathematics* 58.3 (Mar. 2008), pp. 319–340. DOI: 10.1016/j.apnum.2006.12.002.

[8] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016, pp. 165, 176.

[9] Fernando Casas and Sergio Blanes. *A concise introduction to geometric numerical integration*. CRC Press, 2016, pp. 24–25, 105–106, 146, 136–138.

[10] Semyon Aranovich Gershgorin. "Uber die abgrenzung der eigenwerte einer matrix". In: . 6 (1931), pp. 749–754.

[11] Otto Hadač et al. "Minimal oscillating subnetwork in the Huang-Ferrell model of the MAPK cascade". In: *Plos one* 12.6 (2017), e0178457.

[12] Ernst Hairer, Pierre Leone, et al. "Some properties of symplectic Runge-Kutta methods". In: *New Zealand J. of Math* 29.2 (2000), pp. 133–149.

[13] Ernst Hairer, Gerhard Wanner, and Christian Lubich. *Geometric Numerical Integration*. Springer, 2006, pp. 28–30, 34, 101–102, 159, 186, 191–193.

[14] Ernst Hairer, Gerhard Wanner, and Sylvert Paul Nørsett. "Solving Differential Equations I: Nonstiff Problems". In: (1993), pp. 220, 468–469.

[15] David M Hernandez and Edmund Bertschinger. "Time-symmetric integration in astrophysics". In: *Monthly Notices of the Royal Astronomical Society* 475.4 (Jan. 2018), pp. 5570–5584. ISSN: 0035-8711. DOI: 10.1093/mnras/sty184. eprint: https://academic.oup.com/mnras/article-pdf/475/4/5570/24166802/sty184.pdf. URL: https://doi.org/10.1093/mnras/sty184.

[16] Nicholas J Higham. "10. Matrix Exponential". In: *Functions of Matrices*, pp. 233–267. DOI: 10.1137/1.9780898717778.ch10. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898717778.ch10. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898717778.ch10.

[17] Nicholas J Higham. "The scaling and squaring method for the matrix exponential revisited". In: *SIAM Journal on Matrix Analysis and Applications* 26.4 (2005), pp. 1179–1193.

[18] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012, pp. 22, 351, 364–365, 389–390.

[19] PM Hummel and CL Seebeck Jr. "A generalization of Taylor's expansion". In: *The American Mathematical Monthly* 56.4 (1949), pp. 243–247.

[20] Willem H Hundsdorfer and Jan G Verwer. *Numerical solution of time-dependent advection-diffusion-reaction equations*. Vol. 33. Springer, 2003, p. 123.

[21] Arieh Iserles. "Runge-Kutta Methods". In: *A first course in the numerical analysis of differential equations*. 44. Cambridge university press, 2009.

[22] Donald E Knuth. *The Art of Computer Programming: Seminumerical Algorithms, Volume 2*. Addison-Wesley Professional, 2014, pp. 486–488.

[23] Wilhelm Magnus. "On the exponential solution of differential equations for a linear operator". In: *Communications on Pure and Applied Mathematics* 7.4 (Nov. 1954), pp. 649–673. DOI: 10.1002/cpa.3160070404.

[24] Karl Meerbergen and Miloud Sadkane. "Using Krylov approximations to the matrix exponential operator in Davidson's method". In: *Applied numerical mathematics* 31.3 (1999), pp. 331–351.

[25] Cleve Moler and Jack Little. "A history of MATLAB". In: *Proceedings of the ACM on Programming Languages* 4.HOPL (2020), pp. 28–29.

[26] Cleve Moler and Charles Van Loan. "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later". In: *SIAM review* 45.1 (2003), pp. 3–49.

[27] Zdzislaw E Musielak and Billy Quarles. "The three-body problem". In: *Reports on Progress in Physics* 77.6 (2014), p. 065901.

[28] Anthony Ralston. "Runge-Kutta methods with minimum error bounds". In: *Mathematics of computation* 16.80 (1962), pp. 431–437.

[29] Jesus-Maria Sanz-Serna and Mari-Paz Calvo. *Numerical hamiltonian problems*. Vol. 7. Courier Dover Publications, 2018, pp. 37, 181–182.

[30] Gerhard Wanner, Ernst Hairer, and Syvert Paul Nørsett. "Order stars and stability theorems". In: *BIT Numerical Mathematics* 18 (1978), pp. 475–489.

Appendix A

Supplementary Content

A.1 Preliminary

A.1.1 Linear Algebra

Ascertaining the convergence of computing the matrix exponential requires bounds on matrix norms, particularly in the implementation of scaling and squaring. There are three vector p-norms of interest, where for a vector x we have

$$||x||_p = \left(\sum_{i=1}^{d} x_i^p \right)^{\frac{1}{p}}.$$

We are only ever concerned with $p = 1$ (sum of moduli), $p = 2$ (the "Euclidean norm") and $p = \infty$ (modulus of maximal element). Matrix norms are induced by vector norms:

$$||A||_p = \max_x \frac{||Ax||_p}{||x||_p}.$$

The matrix 2-norm is the most theoretically interesting since it is the spectral radius of the matrix - the modulus of the maximal eigenvalue. The 1-norm is the maximum column sum of the moduli of the entries, and the ∞ norm is the same for the row sums. However the norms are equivalent, being that for p and q norms on a matrix A there exist constants α, β such that

$$\alpha||A||_q \leq ||A||_p \leq \beta||A||_q.$$

Therefore bounds on one norm can be expressed as bounds on any other defined norm. We use this in the scaling and squaring implementation to avoid computing a matrix 2-norm [17].

A.1.2 The MATLAB ODE Solvers

The recommended functions for solving ordinary differential equations in MATLAB are presented as `odexy()`, where `x` and `y` are whole numbers. These indicate the order of convergence of the methods that they apply in solving the given ODE. For our purposes, we use `ode45()`, which uses the Dormand-Prince pair of order 4 and 5 explicit Runge-Kutta methods. The function can be

represented in a Butcher tableau

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

The first row representing $b^\top$ is the linear combination of the k_i which is fifth-order accurate, while the second row corresponds to the fourth-order method. The implementation then uses the difference between these methods as an estimate for the error.

A.1.3 The MATLAB Matrix Exponential

The MATLAB function for computing the matrix exponential employs Padé approximation alongside scaling and squaring. Given a matrix A, a scaling parameter s is chosen such that

$$||A/2^s||_\infty < 1/2$$

is satisfied. The $(6,6)$ Padé approximation of $A/2^s$ is computed and then squared s times. This approximation is accurate within $\epsilon \approx 10^{-15}$, which is approximately the machine precision for standard double precision 64-bit arithmetic [26, 17].

A.2 Positivity Preservation

A.2.1 Convex Optimisation for ES2

Denote a vector g of the elements which appear in the expansions of $x(t_n+h)$ and its approximations.

$$g := \begin{pmatrix} A''AxAxx \\ A'A'Axxx \\ A'A^2xx \\ A'AxAx \\ AA'Axx \\ A^3x \end{pmatrix}.$$

We ignore the fact that this is a vector of vectors which is technically not defined. We denote it as a vector in order to write linear combinations of elements as vector inner products. Define the following vectors

$$\begin{aligned} v &:= \begin{pmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & 0 & \frac{1}{6} & \frac{1}{6} \end{pmatrix}^\top \\ u_z &:= \begin{pmatrix} \frac{1}{8} & 0 & \frac{1}{8} & 0 & \frac{1}{4} & \frac{1}{6} \end{pmatrix}^\top \\ u_x &:= \begin{pmatrix} 0 & \frac{1}{4} & \frac{1}{4} & \frac{3}{8} & \frac{1}{8} & \frac{1}{6} \end{pmatrix}^\top. \end{aligned}$$

Evaluating $v^\top g$ gives us the expansion of the actual value of $x(t_n + h)$ at order h^3. We also have $u_z^\top g$ and $u_x^\top g$, which are the expansions of the z and x components of the method at the same order. Let μ, λ be scalars for us to take a linear combination of the x and z methods. The truncation error, denoted here by τ, is

$$\begin{aligned}\tau &= v^\top g - \mu(u_z^\top g) - \lambda(u_x^\top g) \\ &= (v - \mu u_z - \lambda u_x)^\top g.\end{aligned}$$

We aim to optimise this method by minimising the 2-norm of the vector on the left, since we have control over the parameters μ, λ in our optimisation. It also serves to note that g is not a vector in the traditional sense and we have no knowledge on the scaling of its entries, therefore it is ignored. Therefore our problem is of the form

$$\text{minimise } ||v - \mu u_z - \lambda u_x||_2.$$

We have the additional contraint that $\mu + \lambda = 1$ in mind, because the method must be second order accurate. We can block the vectors and scalars to rewrite the problem in the form

$$\text{minimise } f(\mu, \lambda) = ||v - \begin{bmatrix} u_z & u_x \end{bmatrix} \begin{pmatrix} \mu \\ \lambda \end{pmatrix} ||_2^2.$$

The squared norm makes the computations easier. If we let this be our objective function, then the constraint function is $g(\mu, \lambda) = (\mu + \lambda - 1)$. The Lagrangian is

$$L(\mu, \lambda, k) = \sum_{i=1}^{6} (v_i - \mu u_{z(i)} - \lambda u_{x(i)})^2 + k(\mu + \lambda - 1)$$

we take partial derivatives of the Lagrangian in order to find the minimiser

$$\begin{aligned}\frac{\partial L}{\partial \mu} &= \sum_{i=1}^{6} 2(v_i - \mu u_{z(i)} - \lambda u_{x(i)})(-u_{z(i)}) + k = 2(\mu u_z^\top u_z + \lambda u_x^\top u_z - v^\top u_z) + k \\ \frac{\partial L}{\partial \lambda} &= \sum_{i=1}^{6} 2(v_i - \mu u_{z(i)} - \lambda u_{x(i)})(-u_{x(i)}) + k = 2(\mu u_z^\top u_x + \lambda u_x^\top u_x - v^\top u_x) + k \\ \frac{\partial L}{\partial k} &= \lambda + \mu - 1.\end{aligned}$$

When all the partial derivatives are zero, this can be written as the linear system

$$\begin{bmatrix} 2u_z^\top u_z & 2u_x^\top u_z & 1 \\ 2u_z^\top u_x & 2u_x^\top u_x & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{pmatrix} \mu \\ \lambda \\ k \end{pmatrix} = \begin{pmatrix} 2v^\top u_z \\ 2v^\top u_x \\ 1 \end{pmatrix}.$$

We can write this numerically because we know all the values in the vectors, so the system is equivalently

$$\begin{bmatrix} 70 & 52 & 288 \\ 52 & 178 & 288 \\ 288 & 288 & 0 \end{bmatrix} \begin{pmatrix} \mu \\ \lambda \\ k \end{pmatrix} = \begin{pmatrix} 76 \\ 100 \\ 288 \end{pmatrix}$$

having scaled the system to representation in integer values. The solution is

$$\mu = \frac{17}{24}$$
$$\lambda = \frac{7}{24}$$
$$k = \frac{5}{128}.$$

Appendix B

MATLAB Implementations

Some code is repeated in separate files.

B.1 Numerical Integration

B.1.1 Classical Numerical Methods

```
%% ODE solvers: forward and backward Euler methods

TSPAN = [0 5];
Y0 = [1];

% [T1,X1] = forwardEulerMethod(@(x)lineartest(x),TSPAN,Y0)
% [T2,X2] = clippingMethod(@(x)lineartest(x),TSPAN,Y0)
% [T2,X2] = implicitMidpointMethod(@(x)pendulum(x),TSPAN,Y0)
% [T3,X3] = backwardEulerMethod(@(x)pendulum(x),TSPAN,Y0)



hold on
% subplot(121)
% plot(X1(:,1),X1(:,2),X2(:,1),X2(:,2));
% subplot(122)
% plot(T1,X1(:,1),T2,X2(:,1));

for N = 2:2:20
    h = 5/N;
    [T,X] = exponentialEulerMethod(@(x) lineartest(x), TSPAN, Y0, h)
    plot(T,X)
end

%plot(T1, X1);
% plot(T2, X2);
%
% X = linspace(0,5,50);
% Y = exp(-2.5*X);
%
% plot(X, Y);

% plot(X1(:,1), X1(:,2))
% plot(X2(:,1), X2(:,2))
```

```
% plot(X3(:,1), X3(:,2))

function dxdt = oscillator(x)
dxdt = [x(2); -x(1)];
end

function dxdt = lineartest(x)
lambda = -2.5;
dxdt = lambda;
end

%linear test problem is unstable if |xi| < |xi - h l xi|
% |1/(1-hl)| < 1

function dxdt = pendulum(x)
k = 2;
dxdt = [(k^2)*sin(x(1)); x(2)];
end

function [TOUT, YOUT] = forwardEulerMethod(ODEFUNC, TSPAN, Y0)
h = 0.5;
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;

for t = TOUT
    YOUT = [YOUT y];
    y = y + h*ODEFUNC(y);
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end

function [TOUT, YOUT] = clippingMethod(ODEFUNC, TSPAN, Y0)
h = 0.5;
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;

for t = TOUT
    YOUT = [YOUT y];
    y = y + h*ODEFUNC(y);
    for i = 1:size(y)
        if y(i) < 0
            y(i) = 0;
        end
    end
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end

function [TOUT, YOUT] = exponentialEulerMethod(ODEFUNC, TSPAN, Y0, h)
YOUT = [Y0];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;

for t = TOUT(2:end)
```

```
    y = expm(h*ODEFUNC(y))*y;
    YOUT = [YOUT y];
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end


function [TOUT, YOUT] = backwardEulerMethod(ODEFUNC, TSPAN, Y0)
h = 0.1;
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;

for t = TOUT
    y_init = y + h*ODEFUNC(y);
    targetfunc = @(x) x - y - h*ODEFUNC(x);
    y_iter = fsolve(targetfunc,y_init);
    y = y + h*ODEFUNC(y_iter);
    YOUT = [YOUT y];
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end


function [TOUT, YOUT] = implicitMidpointMethod(ODEFUNC, TSPAN, Y0)
h = 0.1;
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;

for t = TOUT
    y_init = y + h*ODEFUNC(y);
    targetfunc = @(x) x - y - h*ODEFUNC((x+y)./2);
    y_iter = fsolve(targetfunc,y_init);
    y = y + h*ODEFUNC((y_iter + y)./2);
    YOUT = [YOUT y];
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end


function [TOUT, YOUT] = trapeziumMethod(ODEFUNC, TSPAN, Y0)
h = 0.01;
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;

for t = TOUT
    y_init = y + h*ODEFUNC(y);
    targetfunc = @(x) x - y - h*(ODEFUNC(x) + ODEFUNC(y))./2;
    y_iter = fsolve(targetfunc,y_init);
    y = y + h*(ODEFUNC(y_iter) + ODEFUNC(y))/2;
    YOUT = [YOUT y];
end
```

```

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end
```

B.2 Symplectic Integration

B.2.1 Hamiltonian Methods

Three-Body Problem and methods for Hamiltonian framework. Störmer-Verlet integration scheme.

```
%% Symplectic Euler-VT Solver for arbitrary dimensional problem

% xinit = [0;
%     0;
%     10*cos(2*pi/3 + pi/4);
%     10*sin(2*pi/3 + pi/4);
%     10*cos(4*pi/3 + pi/4);
%     10*sin(4*pi/3 + pi/4);
%     0;
%     0;
%     0;
%     -100;
%     100;
%     0
% ];


% R1 [ .746156, 0] ; R2 [-0.373078, .238313]; R3 [-0.373078, - .238313] (8)
% V1[ 0, .324677] ; V2 [.764226 , -.162339]; V3 [-[.764226 , -.162339

% xinit = [0.746156;
%     0;
%     -0.373078;
%     0.238313;
%     -0.373078;
%     -0.238313;
%     0;
%     0.324677;
%     0.764226;
%     -0.162339;
%     -0.764226;
%     -0.162339;
% ];

xinit = [1;
    0;
    0;
    0;
    -1;
    0;
    0.3471128135672417;
    0.532726851767674;
    0;
    0;
    -0.3471128135672417;
    -0.532726851767674
];


```

```
d = 6;
J = [
    zeros(6), eye(6);
    -eye(6), zeros(6);
]

% xinit = [3.8, 1.8, 0, 0]';
% X0 = [pi/4, 0]';
TSPAN = [0 20];
% opts = odeset('RelTol',1e-12)
% [T,X] = ode45(@(t,x) J*threeBody(t,x), TSPAN, xinit);
[T,X] = Verlet(@(t,x) threeBody(t,x), TSPAN, xinit);

% [T1,X1] = forwardEulerMethod(@(x) pendulum(x), TSPAN, X0);
% [T2,X2] = implicitMidpointMethod(@(x) pendulum(x), TSPAN, X0);
% [T3,X3] = backwardEulerMethod(@(x) pendulum(x), TSPAN, X0);

hold on

% plot(X1(:,1), X1(:,2))
% plot(X2(:,1), X2(:,2))
% plot(X3(:,1), X3(:,2))

plot(X(:,1),X(:,2))
plot(X(:,3),X(:,4),'x')
plot(X(:,5),X(:,6))

hold off


%% Functions

function dH = oscillator(x)
dH = [x(1); x(2)];
end

function dH = pendulum(x)
k=-1.2;
dH = [
    (k^2)*sin(x(1));
    x(2)
];
end

function dH = phonator(~, x)
lamda = 0.8;
beta = 3;
omega = 0.3;

q1 = x(1);
q2 = x(2);
p1 = x(3);
p2 = x(4);

f = @(x) 1-x + beta*(1 - 1./x.^2);
g = @(x) lamda*(1-x) + beta*(1 - 1./x.^2);

dH = [
    omega*(q1 - q2) - f(q1);
    omega*(q2 - q1) - g(q2);
```

```
    p1;
    p2;
];
end

function Y = phonatorHamilEval(X)
q1 = X(:,1);
q2 = X(:,2);
p1 = X(:,3);
p2 = X(:,4);

lamda = 0.8;
beta = 3;
omega = 0.3;

F = @(x) x - 0.5*x.^2 + beta*(x + 1./x);
G = @(y) lamda*(y - 0.5*y.^2) + beta*(y + 1./y);

Y = 0.5*(p1.^2 + p2.^2) + 0.5*omega*(q1-q2).^2 - F(q1) - G(q2);
end

function dH = threeBody(~,x)
%% Hamiltonian for the three body problem
m1 = 1/3; m2 = m1; m3 = m1;
G = 9.8;

q1 = [x(1); x(2)];
q2 = [x(3); x(4)];
q3 = [x(5); x(6)];
p1 = [x(7); x(8)];
p2 = [x(9); x(10)];
p3 = [x(11); x(12)];

r12 = q1-q2;
d12 = norm(r12);
r23 = q2-q3;
d23 = norm(r23);
r31 = q3-q1;
d31 = norm(r31);

dHdq1 = (G*m2*m1)*(-r12)./(d12^3) + (G*m3*m1)*(r31)./(d31^3);
dHdq2 = (G*m1*m2)*(r12)./(d12^3) + (G*m3*m2)*(-r23)./(d23^3);
dHdq3 = (G*m1*m3)*(-r31)./(d31^3) + (G*m2*m3)*(r23)/(d23^3);
dHdp1 = p1./m1;
dHdp2 = p2./m2;
dHdp3 = p3./m3;

dH = [
    -dHdq1;
    -dHdq2;
    -dHdq3;
    dHdp1;
    dHdp2;
    dHdp3
];

end


%3 body problem
```

```
%y3 project

%asymptotic convergence (poincare)

%conjugacy (trapezium and impl midpoint)

% make explicit, then also apply different methods (implicit)
function [TOUT, XOUT] = SymplecticEulerMethod(ODEHAMIL, TSPAN, X0)
%% Symplectic Euler Method (First Order)
%{
SymplecticEulerMethod(ODEHAMIL, TSPAN, X0) integrates the ODE defined by

    X' = J.ODEHAMIL(X)

ODEHAMIL is the Hamiltonian for the ODE. J is the block matrix [0 I; -I 0].
Applies the Symplectic Euler method.
X0 must be a column vector corresponding to initial conditions for the ODE.
TSPAN is the period of integration.
%}
h = 2^(-6);
[m,n] = size(X0);
dim = max(m,n)/2
TOUT = TSPAN(1):h:TSPAN(end);
J = [zeros(dim), eye(dim); -eye(dim), zeros(dim)]
X_init = X0;
XOUT = zeros(size(TOUT, 2), size(X0, 1));
XOUT(1,:) = X0;

for k = 2:size(TOUT, 2)

%perform an estimate using regular Euler.
X_guess = X_init + h*J*ODEHAMIL(TOUT(k), X_init);

%use fsolve to complete the method, applying the regular euler guess as a
%starting point
X = fsolve(@(x) x - X_init - h*J*ODEHAMIL(TOUT(k), [X_init(1:dim,:) ; x(dim+1:2*dim
    ,:)]),X_guess,optimoptions("fsolve","Display","none"));
XOUT(k,:) = X;
X_init = X;
end

TOUT = TOUT';

end

function p_out = Kinetic(ODEHAMIL,t,p)
    H = ODEHAMIL(t, [zeros(6,1); p]);
    p_out = H(7:12);
end

function q_out = Potential(ODEHAMIL,t,q)
    H = ODEHAMIL(t, [q, zeros(6,1)]);
    q_out = H(1:6);
end
function [TOUT, XOUT] = Verlet(ODEHAMIL, TSPAN, X0)
%% Stormer-Verlet Integrator
dim = size(X0)

h = 2^(-10);
[m,n] = size(X0);
```

```
dim = max(m,n)/2
TOUT = TSPAN(1):h:TSPAN(end);
J = [zeros(dim), eye(dim); -eye(dim), zeros(dim)]
X_init = X0;
XOUT = zeros(size(TOUT, 2), size(X0, 1));
XOUT(1,:) = X0;

q = X0(1:dim)
p = X0(dim+1:2*dim)

%this method is explicit via. computing the separated hamiltonian
% very poorly written
for k = 2:size(TOUT, 2)
    t = TOUT(k);

    p_half = p - (h/2)*Potential(ODEHAMIL, t, q);
    q_next = q + h*Kinetic(ODEHAMIL, t, p_half);
    p_next = p_half - (h/2)*Potential(ODEHAMIL, t, q_next);

    XOUT(k,:) = [q_next' p_next'];

    p = p_next;
    q = q_next;
end

TOUT = TOUT'
end

function [TOUT, YOUT] = forwardEulerMethod(ODEHAMIL, TSPAN, Y0)
%% The classic
h = 0.001;
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;
dim = size(Y0,1)/2;
J = [zeros(dim), eye(dim); -eye(dim), zeros(dim)]

for t = TOUT
    y = y + h*J*ODEHAMIL(y);
    YOUT = [YOUT y];
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end

function [TOUT, YOUT] = implicitMidpointMethod(ODEHAMIL, TSPAN, Y0)
%% Symplectic method!
h = 0.1;
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;
dim = size(Y0,1)/2;
J = [zeros(dim), eye(dim); -eye(dim), zeros(dim)]

for t = TOUT
    y_init = y + h*J*ODEHAMIL(y);
    targetfunc = @(x) x - y - h*J*ODEHAMIL((x+y)./2);
    y_iter = fsolve(targetfunc,y_init);
    y = y + h*J*ODEHAMIL((y_iter + y)./2);
```

```
    YOUT = [YOUT y];
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end


function [TOUT, YOUT] = backwardEulerMethod(ODEHAMIL, TSPAN, Y0)
%% Feels like we only go backwards
h = 0.1;
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;
dim = size(Y0,1)/2;
J = [zeros(dim), eye(dim); -eye(dim), zeros(dim)]

for t = TOUT
    y_init = y + h*J*ODEHAMIL(y);
    targetfunc = @(x) x - y - h*J*ODEHAMIL(x);
    y_iter = fsolve(targetfunc,y_init);
    y = y + h*J*ODEHAMIL(y_iter);
    YOUT = [YOUT y];
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end
```

B.3 Positivity Preservation

B.3.1 Second Order Positivity Preserving Methods

ES2 and EM2 applied to the stratospheric reaction model from [4]. Approximations implemented in modifications of ES2.

```
%% exponential splitting method on chemical kinetics problem

% [T,Z,X] = threeExponentialMethod(@A, [0 0.3], [1 0 0]');
% [T2, Z2, X2] = twoSubsOneExp(@A, [0 0.3], [1 0 0]');

%gblerr(0.3);
tspan = [12*3600, 72*3600];
init = [9.906e1, 6.624e8, 5.326e11, 1.697e16, 8.725e8, 2.240e8]';
% [TM,XM] = MagnusEM2(@(t,y) B(t,y), tspan, init, 30);
[TS,XS] = threeExponentialMethod(@(t,y) B(t,y), tspan, init, 30);



% [TE, XE] = forwardEulerMethod(@(t,y) B(t,y)*y, tspan, init, 0.01);
% [TC, XC] = clippingMethod(@(t,y) B(t,y)*y, tspan, init, 0.01);
%
% [TRK, XRK] = ode45(@(t,y) B(t,y)*y, tspan, init);

semilogy(TS,XS)
xlim(tspan)
ylim([10, 1e16])

```

```
%% generate the invariants

MC = [1,1,3,2,1,2]*(XC');
ME = [1,1,3,2,1,2]*(XE');
MRK = [1,1,3,2,1,2]*(XRK');

M2C = [0 0 0 0 1 1]*(XC');
M2E = [0 0 0 0 1 1]*(XE');
M2RK = [0 0 0 0 1 1]*(XRK');

%% Linear Test Problem

tspan = [0 4];
xinit = [1];
lambda = -2.5

ltp = @(t,x) lambda*x;

[TE, XE] = forwardEulerMethod(ltp, tspan, xinit, 0.5)
[TI, XI] = backwardEulerMethod(ltp, tspan, xinit, 0.5)
[TC, XC] = clippingMethod(ltp, tspan, xinit, 0.5)

ts = linspace(0,4,50)
xs = exp(lambda.*ts)


%% plotting

semilogy(TE,XE)
title("Euler simulation of a stratospheric reaction model")
xlabel("time t")
ylabel("concentration")
legend("O^{1D}","O","O_3","O_2","NO","NO_2")
ylim([1e-4,1e16])
xlim(tspan)


%% invariants

semilogy(TE,M2E)
title("Euler")
xlabel("time t")
ylabel("invariant mass")
xlim(tspan)



%% ode functions


function [TOUT, YOUT] = forwardEulerMethod(ODEFUNC, TSPAN, Y0, h)
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2);
y = Y0;

for t = TOUT
    YOUT = [YOUT y];
    y = y + h*ODEFUNC(t,y);
end

```

```
TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end


function [TOUT, YOUT] = clippingMethod(ODEFUNC, TSPAN, Y0,h)
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2);
y = Y0;

for t = TOUT
    YOUT = [YOUT y];
    y = y + h*ODEFUNC(t,y);
    y(y<0) = 0;
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end

function [TOUT, YOUT] = backwardEulerMethod(ODEFUNC, TSPAN, Y0, h)
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2)
y = Y0;

for t = TOUT
    YOUT = [YOUT y];
    y_init = y + h*ODEFUNC(t,y);
    targetfunc = @(x) x - y - h*ODEFUNC(t+h,x);
    y_iter = fsolve(targetfunc,y_init);
    y = y_iter;
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end

function [TOUT, YOUT] = exponentialEulerMethod(ODEFUNC, TSPAN, Y0, h)
YOUT = [];
TOUT = TSPAN(1):h:TSPAN(2);
y = Y0;

for t = TOUT
    YOUT = [YOUT y];
    y = expm(h*A(t,y))*y
end

TOUT = transpose(TOUT);
YOUT = transpose(YOUT);
end

function [TOUT, ZOUT, XOUT] = threeExponentialMethod(ODEMATR, TSPAN, X0, h)
TOUT = TSPAN(1):h:TSPAN(2);
x = X0;
z = X0;
dim = size(X0,1);
XOUT = zeros(size(TOUT, 2),dim);
ZOUT = XOUT;
XOUT(1,:) = x';
ZOUT(1,:) = z';
```

```
143
144 for k = 2:size(TOUT,2)
145     t = TOUT(k);
146
147     x_hf = expm((h/2)*ODEMATR(t,z))*x;
148
149     z_n = expm(h*ODEMATR(t,x_hf))*z;
150     ZOUT(k,:) = z_n';
151
152     x_n = expm((h/2)*ODEMATR(t,z_n))*x_hf;
153     XOUT(k,:) = x_n';
154
155     x = x_n;
156     z = z_n;
157 end
158
159 TOUT = transpose(TOUT);
160 end
161
162 function [TOUT, XOUT] = MagnusEM2(ODEMATR, TSPAN, X0, h)
163 TOUT = TSPAN(1):h:TSPAN(2);
164 x = X0;
165 dim = size(X0,1);
166 XOUT = zeros(size(TOUT, 2),dim);
167 XOUT(1,:) = x';
168
169 for k = 2:size(TOUT,2)
170     t = TOUT(k);
171     % two matrix exponentials
172     U = (h/2)*ODEMATR(t,x);
173     V = expm(U)*x;
174     W = h*ODEMATR(t,V);
175     x_n = expm(W)*x;
176
177     XOUT(k,:) = x_n';
178
179     x = x_n;
180 end
181
182 TOUT = transpose(TOUT);
183 end
184
185
186 function [TOUT, ZOUT, XOUT] = approximatedExponentials(ODEMATR, TSPAN, X0, h)
187 TOUT = TSPAN(1):h:TSPAN(2)
188 x = X0;
189 z = X0;
190 dim = size(X0,1);
191 XOUT = zeros(size(TOUT, 2),dim);
192 ZOUT = XOUT;
193 XOUT(1,:) = x';
194 ZOUT(1,:) = z';
195
196 for k = 2:size(TOUT,2)
197     x_hf = (eye(3) - (h/2)*ODEMATR(z))\x;
198
199     z_n = (eye(3) - h*ODEMATR(x_hf))\z;
200     ZOUT(k,:) = z_n';
201
202     x_n = (eye(3) - (h/2)*ODEMATR(z_n))\x_hf;
```

```
    XOUT(k,:) = x_n';

    x = x_n;
    z = z_n;
end

TOUT = transpose(TOUT);
end

function [TOUT, ZOUT, XOUT] = twoSubsOneExp(ODEMATR, TSPAN, X0, h)
TOUT = TSPAN(1):h:TSPAN(2)
x = X0;
z = X0;
dim = size(X0,1);
XOUT = zeros(size(TOUT, 2),dim);
ZOUT = XOUT;
XOUT(1,:) = x';
ZOUT(1,:) = z';

for k = 2:size(TOUT,2)
    % approximation
    x_hf = (eye(3) - (h/2)*ODEMATR(z))\x;

    % approximation
    z_n = (eye(3) - h*ODEMATR(x_hf))\z;
    ZOUT(k,:) = z_n';

    % matrix exponential
    x_n = expm((h/2)*ODEMATR(z_n))*x_hf;
    XOUT(k,:) = x_n';

    x = x_n;
    z = z_n;
end

TOUT = transpose(TOUT);
end

function [TOUT, ZOUT, XOUT] = oneSub(ODEMATR, TSPAN, X0, h)
TOUT = TSPAN(1):h:TSPAN(2)
x = X0;
z = X0;
dim = size(X0,1);
XOUT = zeros(size(TOUT, 2),dim);
ZOUT = XOUT;
XOUT(1,:) = x';
ZOUT(1,:) = z';

for k = 2:size(TOUT,2)
    % approximation
    x_hf = (eye(3) - (h/2)*ODEMATR(z))\x;

    % matrix exponential
    z_n = expm(h*ODEMATR(x_hf))*z;
    ZOUT(k,:) = z_n';

    % matrix exponential
    x_n = expm((h/2)*ODEMATR(z_n))*x_hf;
    XOUT(k,:) = x_n';

```

```
    x = x_n;
    z = z_n;
end

TOUT = transpose(TOUT);
end

function GL = A(y)
GL = [
    -0.04, y(3)*1e4, 0;
    0.04, y(2)*(-3e7), 0;
    0, y(2)*(3e7), 0
];
end

function v = sig(t, TR, TS)
TL = mod((t/3600),24);
if (TR <= TL) & (TL <= TS)
    k = (2*TL - TR - TS)/(TS - TR);
    v = 0.5*(1 + cos(pi*abs(k)*k));
else
    v = 0;
end

end

function STR = B(t,y)
TR = 4.5;
TS = 19.5;
k1 = 2.643*(1e-10)*sig(t,TR,TS)^3;
k2 = 8.018*1e-17;
k3 = 6.12*(1e-4)*sig(t,TR,TS);
k4 = 1.576*1e-15;
k5 = 1.07*(1e-3)*sig(t,TR,TS)^2;
k6 = 7.11*1e-11;
k7 = 1.2*10^-10;
k8 = 6.062*1e-15;
k9 = 1.069*1e-11;
k10 = 1.289*(1e-2)*sig(t,TR,TS);

gamma = k3 + k5 + k4*y(2) + k7*y(1) + k8*y(5);
STR = [
    -(k6 + k7*y(3)), 0,                                 k5,                   0,
          0,         0;
    k6,              -(k2*y(4) + k4*y(3) + k9*y(6)), k3,                   2*k1,
          0,         k10;
    0,               k2*y(4)/3,                         -gamma,               2*k2*y(2)
    /3,     0,         0;
    k7*y(3)/2,       k4*y(3)+k9*y(6)/2,                 (gamma + k7*y(1)/2), -(k1 + k2*y
    (2)), 0,         k9*y(2)/2;
    0,               0,                                 0,                    0,
          -k8*y(3),  k10+k9*y(2);
    0,               0,                                 0,                    0,
          k8*y(3),   -(k10+k9*y(2))
];
end

function f = Drv(~,y)
g = A(y);
f = g*y;
```

```
end

function gblerr(T)
tspan = [0, T];
yinit = [1 0 0]';
DIVS = 18;
D = 1:DIVS;
opts = odeset('RelTol',1e-10);
[TM, XM] = ode45(@(t,y) Drv(t,y), tspan, yinit, opts);
exact = XM(end,:)';

H = 0.3./(2.^(D-1));

ERR = zeros(DIVS,3)

for d = D
    h = 0.3/(2^(d-1));
    [T1, ~, X1] = threeExponentialMethod(@A, tspan, yinit, h);
    [T2, ~, X2] = oneSub(@A, tspan, yinit, h);
    [T3, ~, X3] = twoSubsOneExp(@A, tspan, yinit, h);
    eee = X1(end,:)';
    aee = X2(end,:)';
    aae = X3(end,:)';
    ERR(d, 1) = norm(exact - eee, 2);
    ERR(d, 2) = norm(exact - eae, 2);
    ERR(d, 3) = norm(exact - aae, 2);
end

loglog(H,ERR)
end
```

B.3.2 Testing of Matrix Exponential Approximations

```
%% pade counterexample

A = [
    -4 1 0;
    2 -1 2;
    2 0 -2
]

firstorderapproximation = inv(eye(3) - A)

eig(firstorderapproximation)

x = [0.75; 0.25; 0.5];

pade_denominator = inv(denominator(1,1,A))
pade_numerator = numerator(1,1,A)
trueexpA = expm(A)

a = -max(max(abs(diag(diag(A)))))
ThrA = A - a*eye(size(A))

pade_threshden = inv(denominator(1,1,ThrA));
pade_threshnum = numerator(1,1,ThrA);

secondpade = denominator(1,1,A)\numerator(1,1,A)

```

```
y_true = trueexpA*x
y_brute = secondpade*x
y_optim = padeproduct(1,1,A,x,0)

%% sanity test


N = 7
n = 1:N


samples = 500

Y = zeros(N,1)
for k = 1:N
    E = [];
    for iter = 1:samples
        A = GL(10);
        x = rand(10,1);

        y = expm(A)*x;
        ym = positiveSeriesExp(n(k),A,0.4)*x;
        E = [E norm(y-ym,2)/norm(y,2)];
    end
    Y(k) = mean(E);
end

plot(n,Y)


%% order of convergence test

D = 8;
d = zeros(1,D);
d(1) = 1;
for k = 2:D
    d(k) = 2*d(k-1);
end
H = 1./d;

samples = 500;

Y = zeros(1,D)
for r = 1:D
    h = H(r)
    E = []
    for iter = 1:samples
        A = GL(4);
        x = rand(4,1);

%         m_approx = positiveSeriesExp(1,h*A,0.5);
        y = expm(h*A)*x;
%         ym = m_approx*x;
        ym = superPade(h*A,x);

        E = [E norm(y-ym,2)/norm(y,2)];
    end
    Y(r) = mean(E);
end

```

```
loglog(H,Y)


%% further pade counterexample

A = [
    -4 1 0;
    2 -1 2;
    2 0 -2
]
x = [3; 1; 2];

S = linspace(0,1,21)
Y = []
for s = linspace(0,1,21)
    y = expm(A)*x;
    ym = positiveSeriesExp(2,A,0.35)*x;
    Y = [Y norm(y-ym,2)];
end
semilogy(S,Y)

% ym = positiveSeriesExp(A,4,s)*x;
% y = expm(A)*x;

%% compare the true exponential to the Pade estimator
N = 10;
A = rand(N);
A = A - diag(diag(A));
for k = 1:N
    A(k,k) = -sum(A(:,k));
end

x = rand(N,1)

h = rand()
eig(A)
eig(inv(eye(N)-h*A))

trueexpA = expm(A);
y = trueexpA*x; % should preserve positivity
y_approx = padeproduct(1,1,A,x,-0.1) % hmm
error = norm(y - y_approx,2)

%% compare superpade

% A = GL(10)
% a = -max(max(abs(diag(diag(A)))))
% thrA = A - a*eye(size(A))
% [d,~] = size(A)
%
% scale = norm(thrA,1)/((2^1)*(1-a))
%
% p2 = ceil(log2(sqrt(d)*abs(a)/(1-a)))-1
% inv(denominator(1,1,thrA./p));

A = GL(5);
x = rand(5,1);
y = expm(A)*x
yp = superPade(A,x)

```

```
147
148 %% Another Monte Carlo
149
150 err = []
151 N = 10
152 E = zeros(1,N)
153 for order = 1:N
154     M = 10;
155     err = zeros(M,1);
156     for samples = 1:100
157         A = h*rand(M);
158         A = A - diag(diag(A));
159         for k = 1:M
160             A(k,k) = -sum(A(:,k));
161         end
162         x = rand(M,1);
163
164         y = expm(A)*x;
165
166 %         [n,m] = nsplit(order);
167
168 %         ym = padeproduct(n,m,A,x,0);
169         ym = positiveSeriesExp(order,A,0.5)*x;
170         if any(ym(ym<0))
171             sprintf("NOOOOOOOO")
172         end
173         err(samples) = norm(y-ym,2);
174     end
175     E(order) = mean(err);
176 end
177 X = 1:N
178 plot(X, E)
179
180
181 %% Monte Carlo Estimator for behaviour of approximations
182
183 divs = 501;
184 startval = 0;
185 endval = 2;
186 SCL = linspace(startval,endval,divs);
187 E = zeros(divs,1);
188 samples = 500;
189 for S = 1:divs
190     s = SCL(S);
191     err = zeros(1,samples);
192     for M = 1:samples
193         % generate a 10x10 linear system
194         N = 10;
195         A = GL(N);
196         x = rand(N,1);
197
198         y = expm(A)*x;
199         % s is used here
200 %         y_approx = padeproduct(10,10,A,x,s);
201 %         y_approx = positiveSeriesExp(3,A,s)*x;
202         y_approx = superPade(A,x);
203
204         % error
205         err(M) = norm(y-y_approx, 2);
206
```

```
        % negativity
%         err(M) = size(y_approx(y_approx<0),1)/size(y,1);
    end
    E(S) = mean(err);
end
semilogy(SCL,E)

%% Monte Carlo Estimator for convergence of approximations

divs = 10;
E = zeros(divs,1);
H = E;
samples = 500;
for S = 1:divs
    lambda = 2^(-S);
    err = zeros(1,samples);
    for M = 1:samples
        % generate a 10x10 linear system
        N = 10;
        A = GL(N);
        x = rand(N,1);

        y = expm(lambda*A)*x;
%         y_approx = padeproduct(10,10,A,x,s);
%         y_approx = positiveSeriesExp(2,lambda*A,0)*x;
        y_approx = superPade(lambda*A,x);
%         y_approx = (eye(N) - lambda*A)\x;

        % error
        err(M) = norm(y-y_approx, 2)/norm(y, 2);

        % negativity
%         err(M) = size(y_approx(y_approx<0),1)/size(y,1);
    end
    E(S) = mean(err);
    H(S) = lambda;
end
loglog(H,E)

%% Monte Carlo estimator AGAIN: f(order, offset) = error

N = 10;
endval = 5;
divs = 101;
F = zeros(N,divs);
samples = 500;

for n = 1:N
    for S = 1:divs
        s = (S-1)*endval/(divs-1);
        ERR = zeros(samples,1);
        for k = 1:samples
            % make a graph-laplacian random system
            A = GL(10);
            x = rand(10,1);

            % evaluate y and the series approximation
            y = expm(A)*x;
            ym = positiveSeriesExp(n,A,s)*x;

```

```
            % evaluate error
            ERR(k) = norm(y-ym,2);
        end
        F(n,S) = mean(ERR);
    end
end
S = linspace(0,endval,divs)
n = 1:10
mesh(S,n,F)




%% functions

function Y = padeproduct(n,m,A,x,scl)
%% Computes the Pade n,m approximation of [e^A]x
%scl is a scale for the thresholder
a = -scl*max(max(abs(diag(diag(A)))));
ThrA = A - a*eye(size(A));

P = numerator(n,m,ThrA);
Q = denominator(n,m,ThrA);
p = numerator(n,m,a);
q = denominator(n,m,a);
% (q\p)*(Q\P)
Y = (p/q)*(Q\(P*x));
end

function [a,b] = nsplit(k)
%% a+b = k
k_iseven = mod(k+1,2);
a = floor(k/2) + not(k_iseven);
b = floor(k/2);
end

function MAT = numerator(n,m,A)
%% Pade Numerator matrix from formula
MAT = zeros(size(A));
for j = 0:n
    num = factorial(n + m - j)*factorial(n);
    den = factorial(n+m)*factorial(j)*factorial(n-j);
    MAT = MAT + (num/den)*mpower(A,j);
end
end

function MAT = denominator(n,m,A)
%% Pade denominator matrix
MAT = zeros(size(A));
for j = 0:m
    num = factorial(n + m - j)*factorial(m);
    den = factorial(n+m)*factorial(j)*factorial(m-j);
    MAT = MAT + (num/den)*mpower((-A),j);
end
end

function y = superPade(A,x)
%% 1,1 Positivity preserving Pade approximation using magic
a = -max(max(abs(diag(diag(A)))));
[d,~] = size(A);
```

```
ThrA = A - a*eye(size(A));

% abs(a) is the 2-norm of thresholded A, which is useful for getting
% stability conditions
% get the power for "squaring"
m = 0;
r = 10;
while r >= 1
    m = m + 1;
    r = sqrt(d)*abs(a)/(2^m);
end

P = numerator(1,1,ThrA./(2^m));
Q = denominator(1,1,ThrA./(2^m));

% also approximate the scalar
p = numerator(1,1,a/(2^m));
q = denominator(1,1,a/(2^m));


z = p/q;
Z = Q\P;
for k = 1:m
    Z = Z*Z;
    z = z*z;
end

y = z*Z*x;
end


function M = T(n,A)
%% truncated exponential
M = eye(size(A));
for k = n:-1:1
    M = M * A./k;
    M = M + eye(size(A));
end
end

function MAT = positiveSeriesExp(n,A,s)
%% Evaluates an approximation of the matrix exponential
% get positive A and diagonal offset
% series approximation
a = -s*max(max(abs(diag(diag(A)))));
ThrA = A - a*eye(size(A));

MAT = T(n,a)*T(n,ThrA);

end

function A = GL(N)
%% generates a graph-laplacian matrix of dimension N
A = rand(N);
A = A - diag(diag(A));
for k = 1:N
    A(k,k) = -sum(A(:,k));
end
end
```

B.3.3 Second Order Methods IP2, EB2, IQ2

Numerical methods with exponential approximations applied to the MAPK cascade.

```
%% magnus integrator


init = [0.1, 0.175, 0.15, 1.15, 0.81, 0.5]'
tspan = [0 200];
% opts = odeset(RelTol=1e-12); %12 is as good as it gets
% [T_master,X_master] = ode45(@(t,y) MAPK(t,y)*y, tspan, init, opts);

[T_em2, X_em2] = MagnusEM2(@MAPK, tspan, init, 0.1);
[T_ip2, X_ip2] = MagnusIP2(@MAPK, tspan, init, 0.1);
[T_eb2, X_eb2] = MagnusEB2(@MAPK, tspan, init, 0.0005);

subplot(131)
% plot(T_master, X_master)
plot(T_em2, X_em2)

subplot(132)
plot(T_ip2, X_ip2)

subplot(133)
plot(T_eb2, X_eb2)



%% interpolate data

X_EM2 = interp1(T_em2, X_em2, T_master);
X_IP2 = interp1(T_ip2, X_ip2, T_master);
X_EB2 = interp1(T_eb2, X_eb2, T_master);

% subplot(141)
% plot(T_master, X_master)
%
% subplot(142)
% plot(T_master, X_EM2)
%
% subplot(143)
% plot(T_master, X_IP2)
%
% subplot(144)
% plot(T_master, X_EB2)

%interp1(old time points, data points, new time points) returns resampled
%data

%% error

subplot(131)
plot(T_master, vecnorm((X_EM2-X_master),2,2))

subplot(132)
plot(T_master, vecnorm((X_IP2-X_master),2,2))

subplot(133)
plot(T_master, vecnorm((X_EB2-X_master),2,2))

%% error data
```

```
58
59 % for each value of the timestep
60 % % compute the solution of the chosen method at the end time
61 % % take the norm of the error at the end
62 % plot global error against timestep
63 PN = 11;
64 H = zeros(PN,1);
65 for i = 1:PN
66     H(i) = 1/(2^(i+2));
67 end
68
69 H_special = H./16;
70
71 X_true = X_master(end,:)
72
73 %begin
74 %% EM2
75
76 E = zeros(size(H)) % vector of global errors
77 for k = 1:PN
78     h=H(k);
79     [~, X_emsol] = MagnusEM2(@MAPK, tspan, init, h);
80     x_emfin = X_emsol(end,:);
81     E(k) = norm(x_emfin' - X_true',2)/norm(X_true',2);
82 end
83 loglog(H,E)
84
85 %% IP2
86 % same for IP2
87 E2 = zeros(size(H))
88 for k = 1:PN
89     h=H(k);
90     [~, X_ipsol] = MagnusIP2(@MAPK, tspan, init, h);
91     x_ipfin = X_ipsol(end,:);
92     E2(k) = norm(x_ipfin' - X_true',2)/norm(X_true',2);
93 end
94 loglog(H,E2)
95
96 %% EB2
97 % same for EB2
98 E3 = zeros(size(H)) % vector of global errors
99 for k = 1:PN
100     h=H(k);
101     [~, X_ebsol] = MagnusEB2(@MAPK, tspan, init, h);
102     x_ebfin = X_ebsol(end,:);
103     E3(k) = norm(x_ebfin' - X_true',2)/norm(X_true',2);
104 end
105 loglog(H,E3)
106
107 %% IQ2
108 % same for IQ2
109 E3 = zeros(size(H)) % vector of global errors
110 for k = 1:PN
111     h=H(k);
112     [~, X_iqsol] = MagnusIQ2(@MAPK, tspan, init, h);
113     x_iqfin = X_iqsol(end,:);
114     E3(k) = norm(x_iqfin' - X_true',2)/norm(X_true',2);
115 end
116 loglog(H,E3)
117
```

```
%% functions

function [TOUT, XOUT] = MagnusEM2(ODEMATR, TSPAN, X0, h)
%% Second order Magnus integrator
TOUT = TSPAN(1):h:TSPAN(2);
x = X0;
dim = size(X0,1);
XOUT = zeros(size(TOUT, 2),dim);
XOUT(1,:) = x';

for k = 2:size(TOUT,2)
    t = TOUT(k);
    % two matrix exponentials
    U = (h/2)*ODEMATR(t,x);
    V = expm(U)*x;
    W = h*ODEMATR(t,V);
    x_n = expm(W)*x;

    XOUT(k,:) = x_n';

    x = x_n;
end

TOUT = transpose(TOUT);
end

function [TOUT, XOUT] = MagnusIP2(ODEMATR, TSPAN, X0, h)
%% Magnus EM2 using inverse and pade approximations
TOUT = TSPAN(1):h:TSPAN(2);
x = X0;
dim = size(X0,1);
XOUT = zeros(size(TOUT, 2),dim);
XOUT(1,:) = x';

for k = 2:size(TOUT,2)
    t = TOUT(k);
    % inverse (solve) and Pade
    U = (h/2)*ODEMATR(t,x);
    V = (eye(size(U)) - U)\x;
    W = h*ODEMATR(t,V);
    x_n = padeproduct(1,1,W,x,1);

    XOUT(k,:) = x_n';

    x = x_n;
end

TOUT = transpose(TOUT);
end


function [TOUT, XOUT] = MagnusEB2(ODEMATR, TSPAN, X0, h)
%% Magnus EM2 using series approximations
TOUT = TSPAN(1):h:TSPAN(2);
x = X0;
dim = size(X0,1);
XOUT = zeros(size(TOUT, 2),dim);
XOUT(1,:) = x';

for k = 2:size(TOUT,2)
```

```
    t = TOUT(k);
    % series approximations of exponential, first and second order, with
    % diagonal offset 0
    s = 1;
    U = (h/2)*ODEMATR(t,x);
    V = positiveSeriesExp(1,U,s)*x;
    W = h*ODEMATR(t,V);
    x_n = positiveSeriesExp(2,W,s)*x;

    XOUT(k,:) = x_n';

    x = x_n;
end

TOUT = transpose(TOUT);
end

function [TOUT, XOUT] = MagnusIQ2(ODEMATR, TSPAN, X0, h)
%% Magnus EM2 using inverse and positive pade approximation
TOUT = TSPAN(1):h:TSPAN(2);
x = X0;
dim = size(X0,1);
XOUT = zeros(size(TOUT, 2),dim);
XOUT(1,:) = x';

for k = 2:size(TOUT,2)
    t = TOUT(k);
    % inverse (solve) and Pade
    U = (h/2)*ODEMATR(t,x);
    V = (eye(size(U)) - U)\x;
    W = h*ODEMATR(t,V);
    x_n = superPade(W,x);

    XOUT(k,:) = x_n';

    x = x_n;
end

TOUT = transpose(TOUT);
end


function Y = padeproduct(n,m,A,x,scl)
%% Computes the Pade n,m approximation of [e^A]x
%scl is a scale for the thresholder
a = -scl*max(max(abs(diag(diag(A)))));
ThrA = A - a*eye(size(A));

P = numerator(n,m,ThrA);
Q = denominator(n,m,ThrA);
p = numerator(n,m,a);
q = denominator(n,m,a);
% (q\p)*(Q\P)
Y = (p/q)*(Q\(P*x));
end

function MAT = numerator(n,m,A)
%% Pade Numerator matrix from formula
MAT = zeros(size(A));
for j = 0:n
```

```
    num = factorial(n + m - j)*factorial(n);
    den = factorial(n+m)*factorial(j)*factorial(n-j);
    MAT = MAT + (num/den)*mpower(A,j);
end
end

function MAT = denominator(n,m,A)
%% Pade denominator matrix
MAT = zeros(size(A));
for j = 0:m
    num = factorial(n + m - j)*factorial(m);
    den = factorial(n+m)*factorial(j)*factorial(m-j);
    MAT = MAT + (num/den)*mpower((-A),j);
end
end


function MAT = positiveSeriesExp(n,A,s)
%% Evaluates an approximation of the matrix exponential
% get positive A and diagonal offset
% series approximation
a = -s*max(max(abs(diag(diag(A)))));
ThrA = A - a*eye(size(A));

MAT = T(n,a)*T(n,ThrA);
end

function y = superPade(A,x)
%% 1,1 Positivity preserving Pade approximation using magic
a = -max(max(abs(diag(diag(A)))));
[d,~] = size(A);
ThrA = A - a*eye(size(A));

% abs(a) is the 2-norm of thresholded A, which is useful for getting
% stability conditions
% get the power for "squaring"
m = 0;
r = 10;
while r >= 1
    m = m + 1;
    r = sqrt(d)*abs(a)/(2^m);
end

P = numerator(1,1,ThrA./(2^m));
Q = denominator(1,1,ThrA./(2^m));

% also approximate the scalar
p = numerator(1,1,a/(2^m));
q = denominator(1,1,a/(2^m));


z = p/q;
Z = Q\P;
for k = 1:m
    Z = Z*Z;
    z = z*z;
end

y = z*Z*x;
end
```

```
298
299 function M = T(n,A)
300 %% truncated exponential
301 M = eye(size(A));
302 for k = n:-1:1
303     M = M * (A./k);
304     M = M + eye(size(A));
305 end
306 end
307
308 function GL = A(y)
309 GL = [
310     -0.04, y(3)*1e4, 0;
311     0.04, y(2)*(-3e7), 0;
312     0, y(2)*(3e7), 0
313 ];
314 end
315
316 function GL = MAPK(~,y)
317 a = 0.1;
318 k1 = 100/3;
319 k2 = 1/3;
320 k3 = 50;
321 k4 = 1/2;
322 k5 = 10/3;
323 k6 = 1/10;
324 k7 = 7/10;
325 GL = [
326     -k7-k1*y(2) 0 0 k2 0 k6;
327     0 -k1*y(1) k5 0 0 0;
328     0 0 -k3*y(1)-k5 k2 k4 0;
329     (1 - a)*k1*y(2) a*k1*y(1) 0 -k2 0 0;
330     0 0 k3*y(1) 0 -k4 0;
331     k7 0 0 0 0 -k6
332 ];
333 end
334
335 function f = Drv(~,y)
336 g = A(y);
337 f = g*y;
338 end
```