Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

Jenny Margareta Mørk

Handling Discontinuities in Numerical ODE Methods

Master's thesis in Applied Physics and Mathematics
Supervisor: Tor Nordam
June 2021

NTNU
Norwegian University of
Science and Technology

Jenny Margareta Mørk

Handling Discontinuities in Numerical ODE Methods

Master's thesis in Applied Physics and Mathematics
Supervisor: Tor Nordam
June 2021

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

Preface and Acknowledgements

This report presents the work I did for my Master's Thesis in Applied Physics at the Norwegian University of Science and Technology (NTNU). The choice of topic reflects the subjects I have found the most interesting during my studies, so this project has really felt like the perfect conclusion to my five years at NTNU.

The project would not have become what it is without my supervisor, Tor Nordam, who has helped me shape it into something that I am proud to have my name on. I have learned so much and I am very grateful for his assistance. I truly feel like I could not have asked for a better supervisor.

I would also like to thank my friends and fellow students at the Applied Physics and Mathematics programme, and the student organisation Nabla, for making these five years the best years of my life. It has been a great journey and although I am sad to see it end I am also excited to see where we will all go in this next chapter of our lives. Thanks are also due to all the friends I made while I was on exchange at KTH in Stockholm.

Finally, I would like to thank my family, for always believing in me, and for being so interested in everything I do even when they don't understand it. Throughout everything I have done they have been my biggest support and my biggest fans, and for that I am so grateful.

Trondheim, June 2021
Jenny Margareta Mørk

Abstract

It is a common goal when doing any task to want to find the approach that gives the best results without requiring too much effort. For example, modelling the spread of pollutants in the ocean requires computations of a large number of trajectories, so one would be interested in finding the most efficient way to compute these trajectories that still gives an accurate enough result. Given a velocity field one can compute trajectories by setting up an ordinary differential equation and solving it using numerical integration. The velocity field in the ocean is made up by currents which can be measured or modelled, but only at discrete positions and times. To be able to use this velocity field in trajectory computations one must therefore interpolate the data from these discrete points, and the choice of interpolation scheme affects the accuracy one can obtain from a given solver.

This study investigates two methods that are commonly used in numerical integration. One is the well-known fixed-step Runge-Kutta method of order 4, and the other is the variable-step embedded Runge-Kutta method of order 5(4), known as Dormand-Prince 5(4). These are tested in combination with linear, quadratic, cubic, and quintic spline interpolation. Since functions interpolated using spline interpolation have discontinuous derivatives of some order, special-purpose variants of the two integration methods are created, which are designed to handle the discontinuities in the spatial dimensions of the velocity field better. The study considers a time-independent velocity field in two spatial dimensions.

Results show that which solver is the best choice depends on the interpolation scheme. For linear and quadratic spline interpolation, both special-purpose methods performed better than their regular counterparts. The special-purpose 4^{th}-order Runge-Kutta method was also found to improve the order of convergence compared to the regular variant when combined with both linear and quadratic spline interpolation. For cubic and quintic spline interpolation the regular integration methods obtained the same accuracy as their corresponding special-purpose variants at a slightly lower computational cost.

The results found here can be useful for ocean transport problems and other applications where trajectories are found from discrete velocity fields. However, they would be even more valuable in combination with a procedure that handles discontinuities in the temporal dimension of a discrete time-dependent velocity field. Such a study would be a natural follow-up to this work.

Sammendrag

Et vanlig ønske i mange typer arbeid er å finne den fremgangsmåten som gir best resultat uten å kreve for mye innsats. For eksempel så krever modellering av hvordan forurensende stoffer sprer seg i havet at man beregner et stort antall baner, og det er derfor av interesse å finne den mest effektive måten å beregne disse banene på som fremdeles gir nøyaktige nok resultater. Dersom man er gitt et hastighetsfelt så kan man beregne disse banene ved å sette opp en ordinær differensialligning og løse den ved hjelp av numerisk integrasjon. Havstrømmene utgjør hastighetsfeltet i havet, og disse strømmene kan måles, men kun ved diskrete posisjoner og tider. For å kunne bruke dette feltet i beregninger av baner er man derfor nødt til å interpolere dataene fra de diskrete punktene, og hvilken type interpolasjon man velger har konsekvenser for hvilken nøyaktighet man kan oppnå med en gitt integrator.

Denne studien ser på to metoder som er mye brukt i numerisk integrasjon. Den ene er den velkjente Runge-Kutta-metoden av orden 4, implementert med fast tidssteg, og den andre er et innebygget Runge-Kutta-par av orden 5(4), kjent som Dormand-Prince 5(4), implementert med varierende tidssteg. Disse metodene er testet i kombinasjon med lineær, kvadratisk, kubisk, og kvintisk spline-interpolasjon. Siden funksjoner som er interpolert med spline-interpolasjon har diskontinuerlige deriverte av en viss orden blir det også konstruert en spesial-variant av hver av de to metodene, som er spesialdesignet til å håndtere diskontinuiteter i den romlige dimensjonen av hastighetsfeltet. Studien anvender seg av et hastighetsfelt som er uavhengig av tid og definert i to romlige dimensjoner.

Resultatene viser at hvilken integrator som er optimal avhenger av valg av interpolasjonsmetode. For lineær og kvadratisk spline-interpolasjon gir spesial-variantene bedre resultater enn de vanlige metodene. Det ble også funnet at spesial-varianten av den fjerdeordens Runge-Kutta-metoden ga én orden bedre konvergens enn den vanlige fjerdeordens Runge-Kutta metoden i kombinasjon med lineær og kvadratisk spline-interpolasjon. For kubisk og kvintisk spline-interpolasjon ble det funnet at de opprinnelige variantene av integrasjonsmetodene oppnådde samme nøyaktighet som spesial-variantene på en noe mindre kostbar måte.

Resultatene av denne studien kan være nyttige for transportproblemer i havet, og i andre anvendelser der man beregner baner fra diskrete hastighetsfelt. De ville imidlertid vært enda mer verdifulle i kombinasjon med en prosedyre som kan håndtere diskontinuiteter i den temporale dimensjonen av diskrete tidsavhengige hastighetsfelt. En slik studie ville være en naturlig fortsettelse av dette arbeidet.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In April 2010 the oil drilling rig Deepwater Horizon exploded, leading to what is considered the biggest accidental marine oil spill in history. It has been estimated that somewhere between 3 and 4.5 million barrels of oil leaked out into the Gulf of Mexico [1, 2]. Currents, wind and waves made the oil spread over a large region of the ocean, causing a lot of damage to a wide range of organisms, including fish, sea mammals and birds. In total, more than 112 000 km^2 of the ocean surface and more than 2100 km of shoreline across five states in the USA was to some extent covered with oil as a result of the explosion [1].

In addition to oil spill, another thing that greatly affects ecosystems in the ocean and on the shoreline is the presence of plastic debris, and in particular microplastics [3]. It has been estimated that between 1.15 and 12.7 million metric tons of plastic enters the ocean in a year [4, 5], and some research has found that this plastic debris pollution can have dramatic consequences and be very dangerous for a number of different species [6]. Since this plastic waste comes from humans it is our responsibility to address this problem and find a way to solve it.

Hardesty et al. [7] state that the first necessary step in the process of solving the plastic pollution problem is to obtain more knowledge about the plastic debris in the ocean, including where it was disposed, where it accumulates, and the pathways it takes to get from where it originated to where it ends up. For several reasons this knowledge can be difficult to obtain from empirical observations alone [7], and it is therefore of great benefit for practical purposes to find a way to compute these pathways numerically based on empirical data.

In the event of an oil spill disaster like the Deepwater Horizon explosion, being able to predict how the oil spreads can be very helpful when developing cleanup strategies [8]. An effective method for computing particle pathways in the ocean is thus of great value in connection with oil spill accidents as well as in studying the microplastic problem. The currents in the ocean make up a so-called velocity field, which can be thought of as a map describing in what direction a particle at a certain position will move. Such maps can be used to predict the movements and spreading of particles in the ocean, including oil droplets and microplastic.

Computing the paths of oil spills and plastic waste numerically from a given velocity field can be done in several ways, for example by what is called Lagrangian particle tracking. In the Lagrangian framework one determines the trajectories of individual particles by following them one by one as they travel through the velocity field [9, 10]. This type of framework has been used in many applications, including attempts to predict the movement of the Deepwater Horizon oil spill [11], the pathways of microplastic [12, 13, 14] as well as spreading of other pollutants [15, 16], icebergs [17], and jellyfish [18] in the ocean.

For all applications it is of interest to make a prediction that satisfies certain requirements in terms of accuracy and computational cost. In computations of particle trajectories, like in many other areas of life, one would like to get the best possible result with as little effort as possible. Some ways of doing things give excellent results but may require a lot of work to get there, while other ways are easier but produce results that are not as satisfactory. The ideal way is one that gives a result we can be satisfied with without having to work too hard for it. This is true in life in general, as well as in particle trajectory computations.

Finding a particle trajectory from a velocity field essentially means solving an ordinary differential equation (ODE), which is a common problem that can be solved in many different ways. Ordinary differential equations are equations on the form

$$\boldsymbol{x}' = \frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}(t), t), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \tag{1.1}$$

where $\boldsymbol{x}(t)$ describes some property at a time t, and $\boldsymbol{f}(\boldsymbol{x}(t), t)$ describes the rate at which $\boldsymbol{x}$ changes when t changes. If $\boldsymbol{x}$ is an n-dimensional vector, that is, $\boldsymbol{x} = (x_1, x_2, .., x_n) \in \mathbb{R}^n$, then $\boldsymbol{f}(\boldsymbol{x}, t) = (f_1(\boldsymbol{x}, t), f_2(\boldsymbol{x}, t), ..., f_n(\boldsymbol{x}, t))$. These types of equations are very common in natural sciences. Examples include Newton's well-known second law of motion, $\boldsymbol{F} = m\boldsymbol{a}$ (see e.g. [19] p. 115) which describes how the forces acting on an object affect its movements. Another well-known example is the logistic equation, $y' = ry(1-y)$ where r is some parameter (see e.g. [20] p. 22), which models the growth rate of a population. Equation (1.1) can also describe how a drop of oil or some other pollutant moves in the ocean. If $\boldsymbol{x}(t)$ is the position of an oil droplet at time t and $\boldsymbol{f}(\boldsymbol{x}(t), t)$ is the velocity field describing the ocean currents, then solving Eq. (1.1) will give the trajectory describing the motion of the droplet. Figure 1.1 shows an example of what a velocity field might look like if $\boldsymbol{x} = (x, y)$. The longer the arrow at a position is, the faster a particle at that position will move along the direction in which the arrow points.

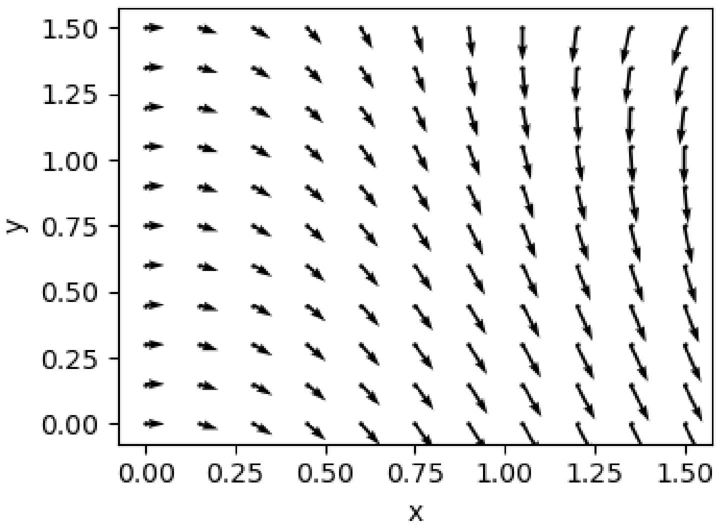


Figure 1.1: Example of a velocity field $f(x, y)$.

Solving equations on the form of Eq. (1.1) can in some cases be done analytically by integration, which then yields an exact expression for $\boldsymbol{x}(t)$. However, sometimes analytical solutions are too difficult, or even impossible, to compute, and in those cases one has to use numerical integration instead. The concept of numerical integration will be introduced more thoroughly in section 2.1, but in short it is about using the knowledge of the rate-of-change $\boldsymbol{f}$ of the property $\boldsymbol{x}$ to compute an approximation of what the value of $\boldsymbol{x}$ will be after some time t. One does this by increasing the time t by small increments h and evaluating $\boldsymbol{f}$ to see how $\boldsymbol{x}$ changes as a result of the increment. Some methods use the same increment h throughout the computations, while others adjust the size of h automatically based on an error estimate that will be explained in detail in section 2.1.3. The numerical solution is not exact, but by choosing the right method one can in most cases find an approximation that is accurate enough.

Accuracy of the result of numerical integration depends on what method one uses and how big the increments h are. The idea is that as $h \to 0$ the approximation will approach the exact solution, and the rate of convergence depends on the method. The order of a numerical integration method refers to how fast the approximation converges toward the exact solution when the time increment

h is reduced. Methods of higher order can achieve a better accuracy with a bigger time step h, while lower-order methods must use a smaller increment h to obtain the same accuracy.

Since higher-order methods can achieve higher accuracy with a bigger time step h one might think that the higher order the better. However, this might not always be the case. Many higher order methods tend to be more costly per step than the lower-order ones. That is, higher-order methods usually need to evaluate $\boldsymbol{f}$ more times than the lower-order methods do in order to compute the next step, and this evaluation can require a lot of work. The higher-order method can obtain a higher accuracy for a given number of steps than a lower order-method, but if the accuracy of the lower-order method is sufficient then the additional work associated with the higher-order method is unnecessary. Hence, to find the best method to solve a given problem, one should think about how accurate one needs the solution to be, and choose the method that can provide that accuracy with the least amount of computations.

In most cases when $\boldsymbol{f}$ is given by an analytical expression, like $f(y,t) = ry(1-y)$ for the logistic equation, the choice of method is the only thing one has to think about when attempting to solve Eq. (1.1). However, if we want to predict the path of an oil droplet in the ocean things become more complicated. The reason for that is simply that the function $\boldsymbol{f}$ is then unknown. However, thanks to organisations like the Norwegian Meteorological Institute [21] and others, some data describing ocean currents is freely available online.

The data provided by the Norwegian Meteorological Institute describes the ocean currents, but only at discrete positions and times, not as the analytical expression we need to solve Eq. (1.1). Luckily, there is a trick one can use in situations like this, and that trick is called interpolation. The concept of interpolation will be expanded on in section 2.2, but for now it suffices to know that interpolating a set of given data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, where $\boldsymbol{f}(\boldsymbol{x}_i) = \boldsymbol{y}_i$ means to create a function $\boldsymbol{g}(\boldsymbol{x})$ such that $\boldsymbol{g}(\boldsymbol{x}_i) = \boldsymbol{y}_i$ for all the given pairs $(\boldsymbol{x}_i,\, \boldsymbol{y}_i)$. Going back to the oil droplet again one can say that we use interpolation to find a function $\boldsymbol{g}(\boldsymbol{x}, t)$ that we can evaluate for any $\boldsymbol{x}$ and any t, while knowing that if we evaluate $\boldsymbol{g}$ at a point that is given in the set of data for the ocean currents, then the value $\boldsymbol{g}$ gives us will be the same as the exact value from the data.

The problem with interpolation is that, depending on the order or the type of interpolation scheme, one might come across discontinuities in the interpolated velocity field. If the velocity field is discontinuous at a position $\boldsymbol{x} = (x, y)$ it could for example mean that if the particle approaches the position $\boldsymbol{x}$ from the left it is told to go up when it reaches that point, while if the particle approaches the same position from the right it is told to go down. Encountering such a discontinuity when attempting to predict the path of some particle can be problematic, and it tends to affect the accuracy of the final result. It is therefore important to be aware of the discontinuities and to deal with them in an appropriate way.

Hairer et al. [22] suggest three ways of dealing with discontinuities:

(i) ignoring them, and letting a variable-step solver adjust the step size to one that gives a small enough error,

(ii) using a numerical integration routine that detects and handles discontinuities (see e.g. [23]), or

(iii) stopping and restarting integration at the discontinuities.

The second strategy is useful when the exact locations of the discontinuities are unknown. However, as will become clear in section 2.2, the locations of the discontinuities are known when we are dealing with interpolated velocity fields. Nordam and Duran [24] have applied the third strategy to discontinuities in the temporal dimension, but not in space. They did however suggest an approach to dealing with the discontinuities in spatial dimensions, and it is this suggested approach that will be applied here. In short the approach makes use of the fact that when the velocity field is defined on the corners of the cells in a 2D grid, discontinuities occur on the boundaries of the cells. By combining a procedure that detects boundary crossings with an iterative root-finding method and a so-called dense output solution, one can determine the exact time at which the discontinuity

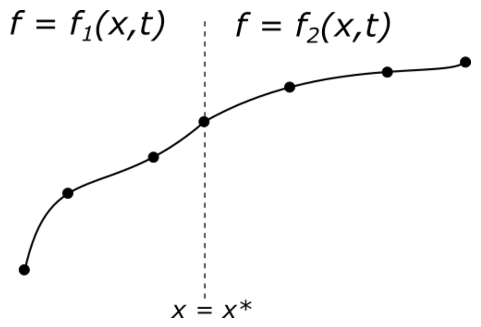


Figure 1.2: Illustration of an integrated trajectory where integration is stopped and restarted at the discontinuity at x^* in the velocity field $f(x,t)$. The dots represent steps taken by the solver, and the dashed line indicates the position x^*.

is crossed and adjust the time step h in order to stop and restart integration at that exact point. The approach will be explained in detail in section 3.1.

The same procedure is also mentioned by Hénon [25] when discussing how to create exact Poincaré maps, which are mappings of the intersections between the trajectory of an N-dimensional dynamical system and an $(N-1)$-dimensional surface-of-section S known as a Poincaré section. In nonlinear dynamics Poincaré maps are a useful tool for studying for example periodicity of orbits, stability of fixed points, or the flow of chaotic systems (see e.g., [20]), and studying the Poincaré map can often be both simpler and more enlightening than studying the actual trajectory.

In the same note Hénon also mentions the case of discontinuities in $\boldsymbol{f}$ or its low-order derivatives. He suggests an alternative method for stopping and restarting integration at certain positions that is applicable both for Poincaré section intersections and discontinuities, and while his method seems very promising for such applications, it may not always be applicable. Therefore, this study will apply the dense output procedure mentioned above.

1.1 Outline of This Study

The aim of this study is to create special-purpose numerical integration methods that can solve ODEs like Eq. (1.1) using strategy (iii) above. That is, the size of the increments h is controlled to make sure that integration is stopped and restarted at the discontinuities in the spatial dimension of the interpolated velocity field. Figure 1.2 illustrates the situation when $\boldsymbol{x} = x$, that is, when $\boldsymbol{x}$ is a scalar. The solid line represents the trajectory, the dots on the line represent steps taken by the integrator, and the dashed line marks a position $x = x^*$ where the velocity field f has a discontinuity. For $x \leq x^*$ the field is given by $f = f_1(x,t)$ while for $x \geq x^*$ the field is given by $f = f_2(x,t)$, where $f_1 \neq f_2$. Taking a step that lands exactly at x^*, as in figure 1.2, ensures that f is continuous within each step.

As will become clear in section 2.3 of chapter 2, the stopping and restarting at the discontinuities is expected to affect the accuracy compared to when the discontinuities are simply stepped over. This work will therefore assess the performance of the special-purpose methods in comparison with two common numerical integration methods. In particular, their performance in terms of accuracy and computational effort will be studied. The methods will be combined with different orders of spline interpolation with the goal of finding the optimal combination of integration method and interpolation scheme for applications where interpolation of the velocity field is necessary. To test whether the procedure that is used to detect and locate the boundary crossings works, an attempt is made to recreate some results from a study on the Lorenz attractor.

The numerical methods applied in this study are a 4$^{\text{th}}$-order Runge-Kutta method and an embedded Dormand-Prince method of order 5(4), both of which are popular choices for solving ODEs numerically in general [22, 26, 27], and in Lagrangian ocean modelling in particular [12, 28, 29, 30]. A description of both methods in some detail will be provided in section 2.1. The construction of the special-purpose variants of both methods will be described in detail in section 3.2. All methods will be combined with spline interpolation of order 2 (linear), 3 (quadratic), 4 (cubic), and 6 (quintic), which will all be introduced in section 2.2, after a general introduction to interpolation. Section 2.3 will discuss how the order of spline interpolation method affects the order of integration. In addition to the special-purpose integrators, chapter 3 will also describe the event location procedure in detail, and explain how errors are estimated from the numerical results. In chapter 4 numerical results will be presented and discussed, and chapter 5 will present the conclusion and suggest some ideas for future work on this topic.

Chapter 2

Theory

This chapter will first give a brief introduction to numerical integration in general, as well as provide details of the Runge-Kutta methods used in this study. Next, a general description of the concept of interpolation will be given, before the spline interpolation schemes used here will be explained in some detail. The final section will then explain how the order of the interpolation scheme affects the effective order of the numerical integration method.

2.1 Numerical Integration

To explain the concept of numerical integration, it is convenient to begin with a general example. Assume that we have an ordinary differential equation

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \tag{2.1}$$

giving the time evolution of some physical system described by $\boldsymbol{x}$. There are many ways to solve such an equation numerically, and a simple but instructive example is what is best known as Euler's method.

To understand Euler's method it is instructive to think of the derivative $\boldsymbol{f}(\boldsymbol{x}, t)$ as a velocity field. That is, we imagine that a particle at the position $\boldsymbol{x}$ at time t is moving with a velocity given by $\boldsymbol{f}(\boldsymbol{x}, t)$. Recall for example figure 1.1 from the previous chapter, which shows an example of what such a vector field might look like in the case that $\boldsymbol{x} = (x, y)$. The arrows represent the velocity vectors at the given positions.

A particle at a point x in the field will move along the velocity vector at that point. At constant velocity v, distance = velocity × time, or $\Delta x = v\Delta t$, when assuming that x denotes position and t denotes time. This means that after a period of time Δt, a particle with initial position x_0 will have moved to the position

$$x_1 = x_0 + \Delta x = x_0 + v\Delta t = x_0 + f(x_0, t_0)\Delta t.$$

This is of course not true if the velocity v is not constant during the period Δt, but for small enough Δt it is an acceptable approximation. Euler's method follows this reasoning; Given an initial position x_0 and time t_0 we evaluate the velocity field $f(x_0, t_0)$, take one small step of duration Δt along the direction of $f(x_0, t_0)$, and end up at the position $x_1 = x_0 + \Delta t f(x_0, t_0)$. At x_1 we take a step in the direction given by $f(x_1, t_0 + \Delta t)$ and end up at $x_2 = x_1 + \Delta t f(x_1, t_0 + \Delta t)$, and so on. The general rule for step $n + 1$ can be written as

$$x_{n+1} = x_n + \Delta t f(x_n, t_0 + n\Delta t), \tag{2.2}$$

which is the formula for Euler's method for the differential equation in Eq. (2.1) with initial condition $x(t_0) = x_0$. The generalisation to problems where $\boldsymbol{x}$ is multidimensional is straightforward.

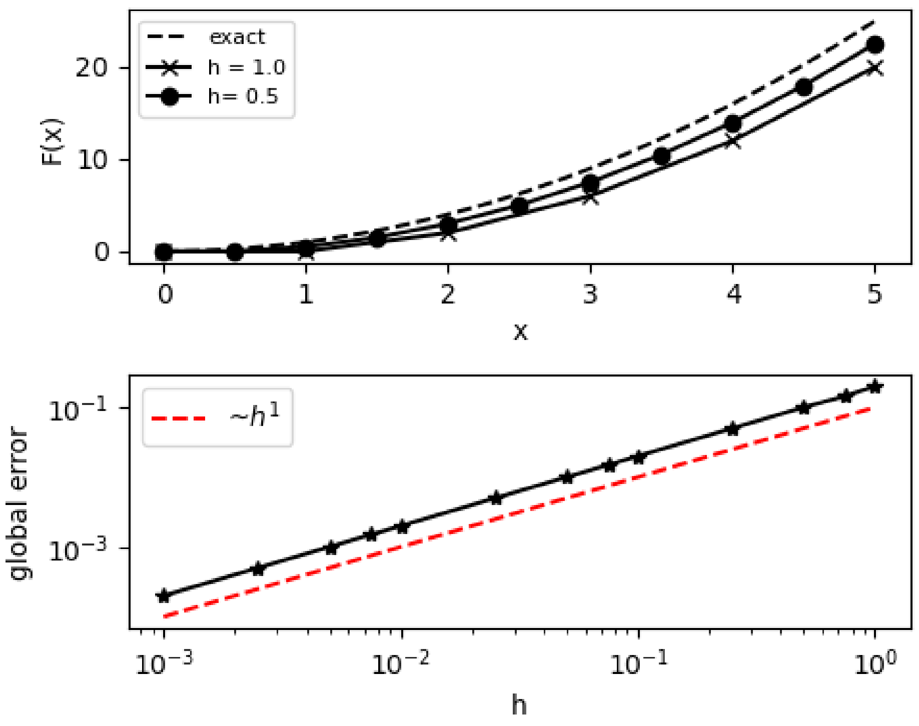


Figure 2.1: Top: Example of ODE solved with Euler's method with different step sizes h. Bottom: Global error versus step size h for Euler's method.

A common convention is to denote the step size in numerical integration methods by h instead of Δt, and from now on this report will follow that convention.

Euler's method has the benefit of being simple and intuitive, and since it only requires one evaluation of the function f per step, it is also computationally efficient compared with higher order methods with the same fixed step size, as we will see later. However, the weakness of this method will soon become clear.

Figure 2.1 shows an example of how well Euler's method approximates the solution

$$F(x) \;=\; y(x) \;=\; x^2$$

to the ODE

$$\frac{dy}{dx} = f(x) = 2x,$$

from $y(x_0 = 0) = 0$ to $x = 5$. The top panel shows the exact solution as well as two numerical approximations with different step sizes h, and the bottom plot shows how the global error scales with h. The global error is the difference between the numerical value y_N at simulation time $t_N = 5$ and the exact solution $y(5) = 25$. Section 2.1.2 will say more about error estimates in numerical integration. The dashed line in the bottom plot in figure 2.1 indicates a slope proportional to the step size h, and it is clear that the global error scales with h^1. This means that Euler's method is a method of order 1. In order to obtain high accuracy in the numerically computed result with Euler's method one would therefore need a quite small step size and hence a large number of steps. As a result, if high accuracy is required, the benefit of the few function evaluations per step is lost because the total number of needed steps becomes large.

Luckily, more sophisticated methods have been developed, that require fewer steps to obtain the same, or even better, accuracy as Euler's method. One example is the Runge-Kutta family of methods, and in particular the well-known and popular 4th-order Runge-Kutta method, often referred to as "the" Runge-Kutta method, or just RK4. When Runge-Kutta methods are introduced and studied in more detail in section 2.1.1 it will become clear that Euler's method is in fact included in the Runge-Kutta family, although when we refer to Runge-Kutta methods we typically only think about the higher order methods.

Another thing that can improve the ratio of accuracy to computational cost is the use of a variable-step method. In Euler's method and the other standard Runge-Kutta methods, the time step h is usually kept fixed. However, if the solution to the ODE has regions with slow change as well as regions with fast change, it could be a good idea to implement a step size control in the ODE solver, that adjusts the step size based on what the solution looks like. This allows the solver to take longer steps if the solution for example remains constant for some time, making it more efficient than a fixed-step method in this region, while also allowing it to take shorter steps in regions where the solution increases or decreases abruptly, thus potentially increasing both accuracy and efficiency compared to a fixed-step method. Such a step size selection algorithm will be explained in section 2.1.3.

The most common [27] variable-step approach is to use so-called embedded formulae, which are methods of order $p(\hat{p})$ that compute a second approximation $\hat{x}_1$ of order $\hat{p}$ in addition to the main approximation x_1 of order p. The difference between these approximations is used in the step size selection algorithm as an error estimate. A popular choice among embedded Runge-Kutta formulae is the Dormand-Prince method of order 5(4), which will also be explained in further detail in section 2.1.1.

2.1.1 Runge-Kutta Methods

Since the Euler method given by Eq. (2.2) was first described, many other, more accurate methods for numerical integration have been introduced. What many of these methods have in common is that the Euler step is part of their basis. By adding extra Euler steps with different step sizes Runge [31] and Heun [32] were able to construct new, more accurate methods. Later Kutta [33] then constructed the general formulation of what is now known as Runge-Kutta methods, which is a family of methods of different orders, which are all based on Euler steps. The simplest method in the Runge-Kutta family is in fact the original Euler method, which can now be described as a 1-stage 1st-order Runge-Kutta method.

Generally, an s-stage explicit Runge-Kutta method, where s is an integer, can be defined as

$$\begin{aligned}
k_1 &= f(x_0, t_0 + c_1 h) \\
k_2 &= f(x_0 + h a_{21} k_1, t_0 + c_2 h) \\
k_3 &= f(x_0 + h(a_{31} k_1 + a_{32} k_2), t_0 + c_3 h) \\
&\;\;\cdots \\
k_s &= f(x_0 + h(a_{s1} k_1 + ... + a_{s,s-1} k_{s-1}), t_0 + c_s h) \\
x_1 &= x_0 + h(b_1 k_1 + ... + b_s k_s),
\end{aligned} \tag{2.3}$$

where $a_{21}, a_{31}, a_{32}, ..., a_{s1}, a_{s2}, ..., a_{s,s-1}, b_1, ..., b_s, c_2, ..., c_s$ are method-specific real coefficients. Usually [22], the coefficients c_i, a_{ij} satisfy $c_1 = 0$ and the relation

$$c_i = \sum_{j=1}^{i-1} a_{ij},$$

for $i = 2, ..., s$, which expresses that all the points where the function f is evaluated are first-order approximations to the exact solution.

It has become customary to write Runge-Kutta methods in a so-called Butcher tableau. Table 2.1 shows the Butcher tableau representation of the general s-stage method in Eq. (2.3).

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

Table 2.1: Butcher tableau representation of the general formulation of explicit Runge-Kutta methods (Eq. (2.3)).

This study will consider two Runge-Kutta methods, one with a fixed time step and one with varying time step. The fixed step method is the method that is commonly referred to as "the" Runge-Kutta method, and best known as RK4. RK4 is a 4th-order method introduced by Kutta [33], with coefficients given in table 2.2. This is not the most precise 4th-order Runge-Kutta method, but it is very popular [22].

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

Table 2.2: Coefficients for "the" Runge-Kutta method, RK4.

The variable-step method considered in this study is the 7-stage embedded Runge-Kutta formula of order 5(4), introduced by Dormand and Prince [34]. The idea behind embedded Runge-Kutta formulae is to find two Runge-Kutta methods that both use the same function values, i.e. the same coefficients c_i and a_i in table 2.1, but different coefficients $b_i \neq \hat{b}_i$. The coefficients b_i, $\hat{b}_i$ must be chosen such that

$$x_1 = x_0 + h \sum b_i k_i$$

is of order p and

$$\hat{x}_1 = x_0 + h \sum \hat{b}_i k_i$$

is of order $\hat{p}$, where usually $\hat{p} = p - 1$ or $\hat{p} = p + 1$ [22].

The method considered here is known as Dormand-Prince 5(4) or DOPRI5, and has coefficients as given by table 2.3. The formula gives a 5th-order approximation x_1 as well as a 4th-order approximation $\hat{x}_1$, and integration is continued with the 5th-order approximation. This is called local extrapolation [22].

One of the benefits of the DOPRI5 method is that it can be implemented with something called the First Same As Last (FSAL) principle. Notice in table 2.3 that the coefficients a_i of the 7th stage, k_7, are equal to the coefficients b_i in the 5th-order approximation x_1. This means that k_7 of one step is equal to k_1 of the next step. If k_7 is passed on from one step to the next one does therefore not have to compute k_1 in the next step. This is the FSAL principle. As a result, every step except the first one will only need to evaluate f 6 times, instead of 7.

The DOPRI5 method is commonly used and is for example implemented in the `ode45` solver in MATLAB [26] and in the `scipy.integrate.RK45` solver for Python [35]. It is implemented here with a variable step size h, meaning that h will be adjusted automatically based on an error estimate. The step size selection algorithm will be described in section 2.1.3, but first some concepts used in error estimates will be introduced.

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$\frac{-56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{4}$	
x_1	$\frac{35}{384}$	0	$\frac{500}{113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{4}$	0
$\hat{x}_1$	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

Table 2.3: Coefficients for the Dormand & Prince method of order 5(4), DOPRI5.

2.1.2 Error Estimates

There are two important measures for the error in numerical integration. One is called the *local* error and the other is called the *global* error. The local error, also called the one-step error, is the error in just one step using a numerical integration method. Assume that the exact solution at t_0, denoted $x(t_0)$, equals the starting point x_0 of the numerical solution. That is, $x_0 = x(t_0)$. After one step of size h, the numerical solution x_1 is, in Henrici's [36] notation,

$$x_1 = x_0 + h\Phi(t_0, x_0, h),$$

where Φ is called the increment function. For Runge-Kutta methods,

$$\Phi(t_0, x_0, h) = \sum_{i=1}^{s} b_i k_i(t_0, x_0, h).$$

The local error e after this one step is given by

$$e = |x(t_0 + h) - x_1|, \tag{2.4}$$

i.e. the difference between the numerical approximation x_1 and the exact solution evaluated at $t_0 + h$. Note the requirement $x_0 = x(t_0)$ for this to be true.

The global error E of a numerical method is defined as the error of the computed solution after several steps. Mathematically it can be expressed as

$$E = |x(t_N) - x_N|, \tag{2.5}$$

where $x(t_N)$ denotes the exact solution $x(t)$ evaluated at $t = t_N$ and x_N denotes the numerically computed solution after N steps. It is still assumed that $x(t_0) = x_0$. Note that the global error is not simply a sum of local errors, because of the assumption in the local error that the starting point of the step is exact. If the local error of any step is nonzero, then this error will affect every subsequent step.

2.1.3 Step Size Selection

The DOPRI5 method is implemented here with a variable time step, meaning that

$$t_n = t_{n-1} + h_n,$$

where in general $h_i \neq h_j$ for $i \neq j$. The step size h_n is updated at each step, based on an error estimation procedure. The procedure used here is described by Hairer et al. in [22] (pp. 167–168).

Given some starting step size h one may compute the two approximations x_1 and $\hat{x}_1$. If the order of $\hat{x}_1$ is higher than the order of x_1 then the difference $|x_1 - \hat{x}_1|$ is an estimate of the local error in x_1. In the case that x is a vector we want this error to satisfy

$$|x_{1,i} - \hat{x}_{1,i}| \leq sc_i,$$

with

$$sc_i = Atol_i + \max(|x_{0,i}|, |x_{1,i}|) \cdot Rtol_i,$$

componentwise, where $Atol_i$ and $Rtol_i$ are user-specified tolerances. If $Atol_i = 0$ relative errors are considered, and if $Rtol_i = 0$ absolute errors are considered, but usually both are nonzero [22]. This algorithm ensures that the local error estimate is lower than the given tolerance.

Note that in DOPRI5 integration is continued with the higher-order approximation. That is, the order of x_1 is higher than the order of $\hat{x}_1$. This is called local extrapolation. In that case the difference $|x_1 - \hat{x}_1|$ is not an estimate of the local error [22], but it can still be used for the purpose of step size control in the same way.

Given the values sc_i one may now compute a normalised measure of the error

$$e = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_{1,i} - \hat{x}_{1,i}}{sc_i}\right)^2}.$$

In order to find an optimal step size, e is compared to 1. That is,

$$h_{\text{opt}} = h \cdot \left(\frac{1}{e}\right)^{\frac{1}{q+1}},$$

where $q = \min(p, \hat{p})$, and we recall that h is the starting step size. For DOPRI5, $q = \min(5, 4) = 4$.

Before proceeding, we should introduce a safety factor, in order to make sure that the probability is high that the "error" e will be acceptable in the next step, and to make sure that h is not increased or decreased too fast. Introducing the safety factors fac and $facmax$ we may now compute the next step

$$h_{\text{new}} = \min(facmax \cdot h, fac \cdot h_{\text{opt}}).$$

For suggestions on choices for the values of fac and $facmax$, see e.g., [22] p. 168. In this work, the values $fac = 0.8$ and $facmax = 2.5$ are chosen, and they are kept constant for all computations.

If $e \leq 1$ the step is accepted, and the solution is advanced with x_1. A new step is then tried with $h = h_{\text{new}}$. If, however, $e > 1$, the step is rejected and the same computations as described above are repeated with h_{new} as h.

2.2 Interpolation

As mentioned in chapter 1, velocity fields in the ocean are not given by analytic expressions. Rather, they are given by discrete sets of measured or modelled data representing the momentary field $f(x_n, t_n)$ at discrete positions x_n and times t_n. To be able to evaluate the field at an arbitrary position x and time t one must therefore turn to interpolation.

Interpolating a set of data points means to find a function that passes through those points. It may even be described as the opposite of function evaluation, in which a function is given and one wishes to compute points on the curve given by that function. Any function that passes through the set of data points is said to interpolate the points, regardless of how it looks in between the points. Interpolation is thus not the same as function approximation, although the concepts are related.

There are many ways to perform interpolation, for example nearest neighbour interpolation, polynomial interpolation, and spline interpolation, and they all have both benefits and drawbacks.

Nearest neighbour interpolation is an interpolation method that only uses the value at one single point to approximate the value at points nearby. If the point x is closer to the point x_i than any other point in the data set, it is simply assigned the value $f(x_i)$. The benefit of nearest neighbour interpolation is that it requires very little work, but the potential disadvantage is that the interpolated curve is discontinuous.

A more popular choice is polynomial interpolation, a method that constructs a polynomial that passes through all the data points. According to the Main Theorem of Polynomial Interpolation (see e.g., [27] p. 141) there exists a unique polynomial $P(x)$ of degree $n-1$ or less that interpolates a set of n data points (x_i, y_i), where the x_i are distinct. Two popular methods for finding this unique polynomial are Lagrange interpolation and Newton's divided differences. The clear advantage of polynomial interpolation is that there always exists a solution that interpolates all the data points, and that this solution - being a polynomial - is continuous and smooth. A disadvantage of polynomial interpolation is what is known as Runge's phenomenon, which may occur when the data points are equally spaced and the polynomial degree is high.

Runge's phenomenon is nicely demonstrated by the Runge example, where the function

$$f(x) = \frac{1}{1 + 12x^2}$$

is interpolated at evenly spaced points in the interval $[-1, 1]$. Figure 2.2 shows the function $f(x)$ as well as the so-called Lagrange polynomial interpolated from 20 evenly spaced points. We see that the interpolated polynomial oscillates a lot near the ends of the interval, and it is this oscillation that is known as Runge's phenomenon.

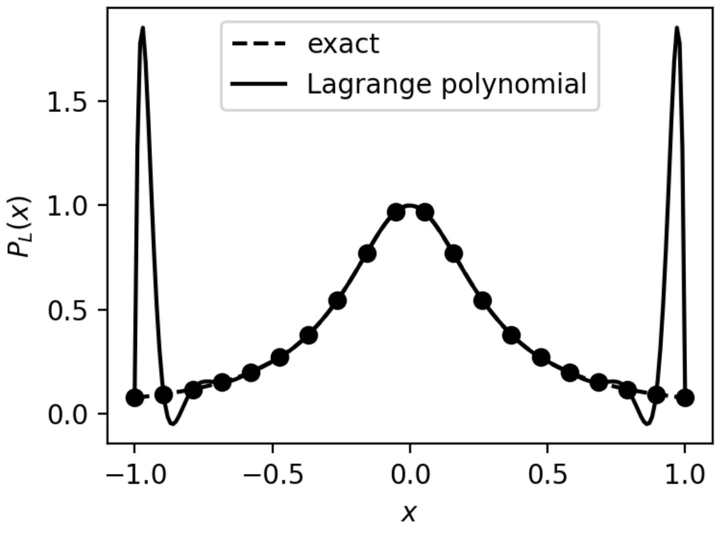


Figure 2.2: Illustration of Runge's phenomenon in the interpolated function $P_L(x)$.

Luckily, there are ways to avoid Runge's phenomenon. If one is free to choose the spacing of the data points, one can avoid Runge's phenomenon by choosing the optimal spacing; This is known as Chebyshev interpolation. However, as mentioned above, in many real-life applications one only has a given set of data points, and no known analytical expression and hence no option to choose other data points. If this is the case, a good alternative to polynomial interpolation is spline interpolation.

Spline interpolation is a lot like polynomial interpolation, but instead of using just one (usually high-degree) polynomial to interpolate all data points one uses several low-degree polynomials. Keeping the polynomial-degree low reduces the possibility of the wild oscillations observed in Runge's phenomenon. Splines produce smooth solutions with continuous derivatives up to some order, and are thus a good alternative in situations where this is a concern.

How many derivatives are continuous depends on the order of the spline. When the method is of order 2 the first derivative is discontinuous, when the method is of order 3 the second derivative is discontinuous, and so on. In other words, if one requires n continuous derivatives, one should choose a spline interpolation method of order $n+2$. The order of an interpolation scheme is equal to the polynomial degree plus one [37][1]. This study will consider spline interpolation of order 2 (linear splines), 3 (quadratic splines, sometimes also called parabolic splines), 4 (cubic splines) and 6 (quintic splines), all of which will now be presented.

2.2.1 Linear Splines

Linear spline interpolation can be thought of as simply "connecting the dots". Imagine that we are given a set of data points $(x_1, y_1), ..., (x_n, y_n)$, where the x_i are distinct and in increasing order. If we draw a straight line from point (x_1, y_1) to point (x_2, y_2), another line from (x_2, y_2) to (x_3, y_3), and so on, we end up with $n-1$ separate line segments connecting the n points. Straight lines are the same as polynomials of degree 1, so linear spline interpolation is an interpolation scheme of order 2.

With n data points, a linear spline $S(x)$ consists of $n-1$ line segments $S_i(x)$ on the general form

$$S_i(x) = a_i x + b_i, \quad \text{on } [k_i, k_{i+1}], \tag{2.6}$$

for $i = 1, ..., n-1$. The points k_j are called the knots of $S(x)$. The knots of a spline are the points at which the different segments are connected to each other, and in the case of linear spline interpolation, the knots are at the same positions as the data points x_j. That is, $k_i = x_i$.

The coefficients a_i, b_i in Eq. (2.6) can be determined by imposing the continuity conditions

$$S_i(x_i) = y_i,$$

and

$$S_i(x_{i+1}) = y_{i+1}$$

for $i = 1, ..., n-1$. Inserting these conditions in Eq. (2.6) we get the equations

$$\begin{aligned} S_i(x_i) &= a_i x_i + b_i = y_i \\ S_i(x_{i+1}) &= a_i x_{i+1} + b_i = y_{i+1}. \end{aligned} \tag{2.7}$$

Solving Eqs. (2.7) for b_i gives the equations

$$b_i = y_i - a_i x_i \tag{2.8}$$

$$b_i = y_{i+1} - a_i x_{i+1} \tag{2.9}$$

for the coefficients b_i. Equating these two expressions and solving for a_i gives

$$\begin{aligned} y_i - a_i x_i &= y_{i+1} - a_i x_{i+1} \\ a_i(x_{i+1} - x_i) &= y_{i+1} - y_i \\ a_i &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i}. \end{aligned} \tag{2.10}$$

Inserting Eq. (2.8) and Eq. (2.10) into Eq. (2.6) gives the general expression

$$S_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i), \quad x_i \le x \le x_{i+1}$$

for the line segments that make up the linear spline. Note that if one uses Eq. (2.9) instead of Eq. (2.8) the general expression will look slightly different, but the resulting line will be identical since the two expressions for b_i must be equivalent.

[1]There are several different definitions of the order of an interpolation scheme. For example, Press et al. [38] define it as the number of points used minus one. In this work, the definition by C. de Boor [37] given in the text will be used.

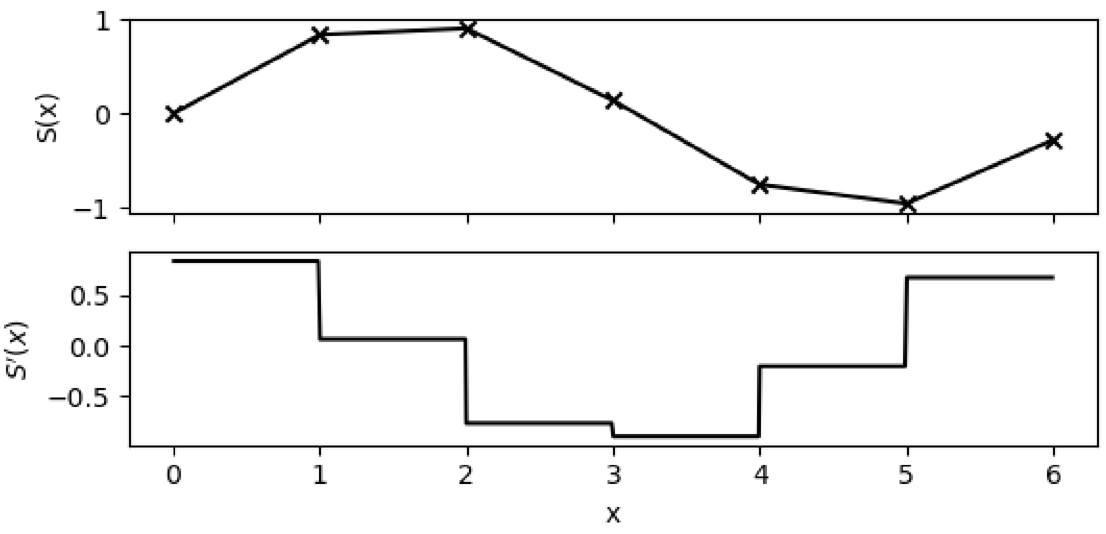


Figure 2.3: A linear spline $S(x)$ and its first derivative $S'(x)$.

As an example of linear interpolation, consider figure 2.3. The top panel shows seven points that have been interpolated using linear spline interpolation. One can easily see that each pair of points is connected by a straight line.

Linear spline interpolation is simple and can successfully interpolate an arbitrary set of n data points. However, as is clear from figure 2.3, linear splines lack smoothness, so the derivative is not continuous. The bottom panel in figure 2.3 shows the derivative of the interpolated solution in the top panel. It is immediately clear that the derivative is not continuous, and that the discontinuities are at the same values of x as the data points. The reason why this happens is because when x crosses a knot k_i, the spline $S(x)$ equals a different polynomial $S_i(x)$. That is, if for example $k_1 \leq x \leq k_2$ then $S(x) = S_1(x)$, but when $k_2 \leq x \leq k_3$ then $S(x) = S_2(x)$, which is generally a different polynomial. As a result, the first derivative $S'(x)$ of the spline will be discontinuous at the knots, and for linear splines $k_i = x_i$, so the derivatives are discontinuous at the data points.

2.2.2 Quadratic Splines

Increasing the order of interpolation by one, to quadratic spline interpolation, one would expect that the first derivative is continuous but the second is not. Figure 2.4 shows the same data points as in figure 2.3 interpolated using a quadratic spline. From the top the panels in the figure show the spline $S(x)$ itself, its first derivative $S'(x)$, and its second derivative $S''(x)$. It is clear that the first derivative is continuous but the second is not, just as expected.

The thing to note about the second derivative $S''(x)$ in figure 2.4 is that while it is discontinuous, the discontinuities are not at the same places as they were for the first derivative of the linear spline in figure 2.3. Instead, the second derivative is discontinuous at the midpoint between two data points. The explanation behind this lies in the construction of quadratic splines. Quadratic splines $S(x)$ are piecewise quadratic functions. That is, they consist of several quadratic polynomials on the general form

$$S_i(x) = a_i x^2 + b_i x + c_i, \quad \text{on } [k_i, k_{i+1}], \tag{2.11}$$

where again k_j are the knots of $S(x)$. Recall that for linear splines the knots are at the same positions as the data points, but the same is not true for quadratic splines. Rather, it has been found that when creating quadratic splines it can be better to choose the midpoints between the data points as knots. That is, $k_i = (x_i + x_{i+1})/2$. For this study it suffices to just know that the knots are at the midpoints for quadratic interpolation, but the interested reader can refer to chapter VI in [37] for more information on this choice.

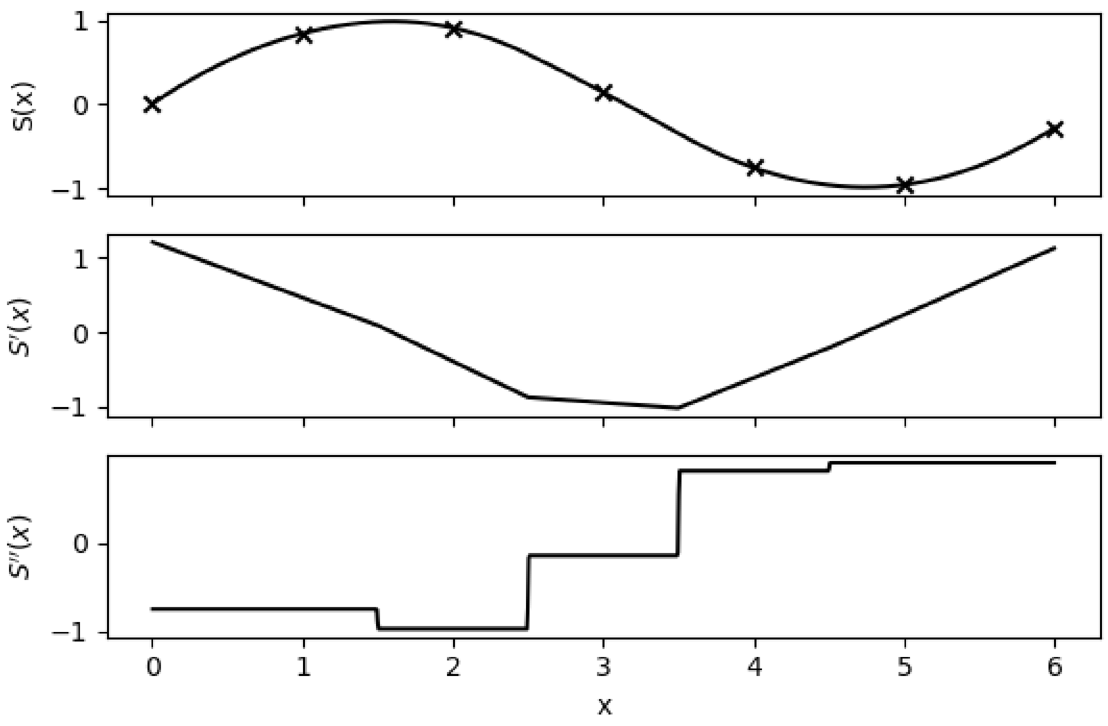


Figure 2.4: A quadratic spline $S(x)$ and its first two derivatives.

With these values for the knots k_i, $S(x)$ consists of n quadratic polynomials $S_i(x)$ on the form of Eq. (2.11). These polynomials must satisfy

$$S_i(x_i) = y_i, \quad i = 1, ..., n$$

$$S_i(k_i) = v_i, \quad i = 1, ..., n$$

and

$$S_i(k_{i+1}) = v_{i+1}, \quad i = 1, ..., n$$

where v_1 and v_{n+1} are free and v_j, $j = 2, ..., n$, must be chosen so that

$$S_i(k_{i+1}) = S_{i+1}(k_{i+1}), \quad i = 1, ..., n-1$$

and

$$S_i'(k_{i+1}) = S_{i+1}'(k_{i+1}). \quad i = 1, ..., n-1$$

These conditions ensure that the resulting spline $S(x)$ is continuous and has a continuous first order derivative. The second derivative $S(x)$ will be discontinuous at the knots k_i, which we recall are chosen as the midpoints between the data points for quadratic splines.

2.2.3 Cubic Splines

Since linear splines connect the data points using straight lines (or degree 1 polynomials), and quadratic splines connect them using quadratic (degree 2) polynomials, it is easy to guess that cubic splines use cubic polynomials, or polynomials of degree 3, to connect the data points. Generally, considering a set of n data points (x_1, y_1), ..., (x_n, y_n), a cubic spline interpolating these points is a set of $n-1$ cubic polynomials on the form

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \quad \text{on } [k_i, k_{i+1}] \tag{2.12}$$

Just like for linear splines, the k_i are here the same as the data points x_i. The $4n-4$ coefficients are determined from the conditions that the spline itself must be continuous, and so must the first and second order derivatives.

The requirement that the spline itself must be continuous once again leads to the $2n-2$ equations

$$S_i(x_i) = y_i, \quad i = 1, ..., n-1$$

and

$$S_i(x_{i+1}) = y_{i+1}, \quad i = 1, ..., n-1.$$

Furthermore, the requirement that the slope of two adjacent splines must be the same gives the $n-2$ equations

$$S'_i(x_{i+1}) = S'_{i+1}(x_{i+1}), \quad i = 1, ..., n-2.$$

These equations ensure that the first derivative of the spline is continuous. Finally, cubic splines must have continuous second derivatives, a condition that yields the $n-2$ equations

$$S''_i(x_{i+1}) = S_{i+1}(x_{i+1}), \quad i = 1, ..., n-2.$$

These $4n-6$ equations ensure that cubic splines are continuous and smooth, have continuous and smooth first order derivatives, and continuous second order derivatives. However, since there are a total of $4n-4$ unknowns in the polynomials given by Eq. (2.12), the system is still underdetermined and has infinitely many solutions. The remaining equations are introduced as what is known as endpoint conditions, and there are several options. The most common choice of endpoint conditions for cubic splines is to require that the spline $S(x)$ has an inflection point, i.e. a point at which the curvature changes sign, at each end of the interval $[x_1, x_n]$, giving what is called a natural spline. To read more about the other options for endpoint conditions for cubic splines, see e.g. [27] (pp. 173–176).

Mathematically, the natural spline condition can be expressed by the two equations

$$S''_1(x_1) = 0$$

and

$$S''_{n-1}(x_n) = 0.$$

With these additional two equations we now have $4n-4$ equations in $4n-4$ unknowns and can thus find the cubic spline that interpolates the n given data points. It is worth noting that cubic splines are particularly practical, since the resulting set of equations is tridiagonal [38] and can be solved efficiently with for example LU decomposition (see [38] pp. 42–43).

Figure 2.5 shows the same data points as in figure 2.3 interpolated using cubic spline interpolation, as well as the first three derivatives. It is clear that the spline $S(x)$ is both continuous and smooth. From figure 2.5 it is also clear that the first and second order derivatives are continuous, as expected. The third derivative, however, is not continuous. Just like for linear splines, the discontinuities are at the same x-positions as the given data points, and that is because the knots k_i once again coincide with the data points x_i.

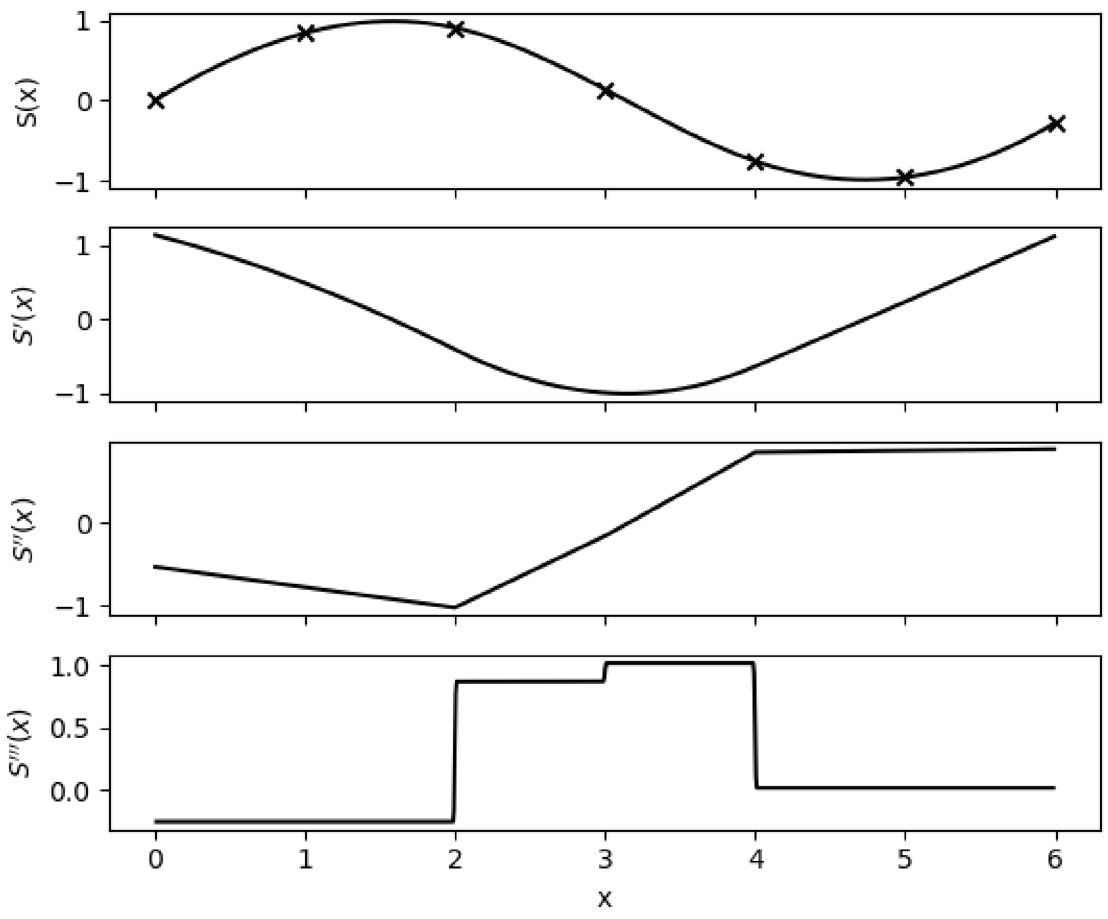


Figure 2.5: A cubic spline $S(x)$ and its first three derivatives.

2.2.4 Quintic Splines

Order 6 spline interpolation is called quintic spline interpolation. The description of quintic splines follows the same pattern as for the lower-order splines, so to avoid too much repetition only a brief description will be provided here.

Quintic splines use 5th-degree polynomials to interpolate the data points. That is, the quintic spline $S(x)$ that interpolates n data points (x_i, y_i) consists of $n-1$ polynomials on the form

$$S_i(x) = a_i x^5 + b_i x^4 + c_i x^3 + d_i x^2 + e_i x + f_i, \quad \text{on } [k_i, k_{i+1}] \tag{2.13}$$

where $k_i = x_i$ here as well. The coefficients are determined by setting up a system of equations in a similar manner as above. The requirements for a quintic spline is that the derivatives up to and including order 4 are continuous, which leads to the $6n - 10$ equations

$$\begin{aligned}
S_i(x_i) &= y_i, & i &= 1, ..., n-1 \\
S_i(x_{i+1}) &= y_{i+1}, & i &= 1, ..., n-1 \\
S_i'(x_{i+1}) &= S_{i+1}'(x_{i+1}), & i &= 1, ..., n-2 \\
S_i''(x_{i+1}) &= S_{i+1}''(x_{i+1}), & i &= 1, ..., n-2 \\
S_i'''(x_{i+1}) &= S_{i+1}'''(x_{i+1}), & i &= 1, ..., n-2 \\
S_i^{(4)}(x_{i+1}), &= S_{i+1}^{(4)}(x_{i+1}). & i &= 1, ..., n-2
\end{aligned}$$

Together with four endpoint conditions one can use the continuity conditions to determine the coefficients in Eq. (2.13) that make $S(x)$ interpolate the given data points.

2.3 The Relationship Between Integration and Interpolation Order

Hairer et al. [22] present the following theorem (Theorem 3.1 p. 157 in [22]):

Theorem 1. *If the Runge-Kutta method in Eq.* (2.3) *is of order p and if all partial derivatives of $f(x,t)$ up to order p exist (and are continuous), then the local error of Eq.* (2.3) *admits the rigorous bound*

$$||x(t_0+h)-x_1|| \leq h^{p+1}\left(\frac{1}{(p+1)!}\max_{\tau\in[0,1]}||x^{(p+1)}(t_0+\tau h)|| + \frac{1}{p!}\sum_{i=1}^{s}|b_i|\max_{\tau\in[0,1]}||k_i^{(p)}(\tau h)||\right)$$

and hence also

$$||x(t_0+h)-x_1|| \leq Ch^{p+1}. \tag{2.14}$$

Theorem 1 states that if all partial derivatives of the right-hand side $\boldsymbol{f}$ in Eq. (2.1) exist up to and including order p, then the local error given by Eq. (2.4) of a Runge-Kutta method of order p is bounded by Ch^{p+1} for some constant C. Consider as an example RK4 combined with quintic spline interpolation. RK4 is a 4th-order Runge-Kutta method, and quintic splines are order 6 and have four continuous derivatives. This means that the requirements in Theorem 1 are fulfilled, and it holds that when the velocity field is interpolated using quintic spline interpolation, the local error in a single step of size h taken with RK4 is bounded by Ch^5, for some constant C. Using cubic spline interpolation instead, $f(x,t)$ only has two continuous derivatives, and the same upper bound for the local error cannot be guaranteed. In fact, this upper bound for the local error only holds for spline interpolation of order 6 (like quintic) or higher.

Furthermore, if the local error scales as $\mathcal{O}(h^{p+1})$ then the global error will scale as $\mathcal{O}(h^p)$ [22, 39]. Considering the combination of quintic splines and RK4 again this means that since the local error is bounded by Ch^5, the global error scales with h^4. This means that RK4 can be expected to be 4th-order accurate when used with a velocity field that is interpolated using quintic spline interpolation. With lower order spline interpolation the same bound for the local error does not hold, meaning that one cannot trust that RK4 will actually be 4th-order accurate, in spite of its name.

The consequence of Theorem 1 in practical applications with velocity fields given as discrete data points is that a pth-order integration method might not actually be pth-order accurate if one does not choose an appropriate interpolation scheme. It is therefore of great importance to be aware of this connection between the number of continuous derivatives and the local error bound in Eq. (2.14), and hence the effect the choice of interpolation scheme can have on both accuracy and computational effort.

As stated in chapter 1, Hairer et al. [22] discuss discontinuities in the right hand side $\boldsymbol{f}$ of an ODE, and suggest three strategies for dealing with them. The third strategy is to stop and restart integration at the discontinuities, as was illustrated in figure 1.2 in chapter 1. For convenience, the same illustration is provided in figure 2.6 in this section. The result of this forced stopping is that the velocity field $\boldsymbol{f}$ becomes continuous within each step, even though it is not continuous over the whole trajectory.

Consider figure 2.6. As is indicated in the figure, $f = f_1(x,t)$ for $x \leq x^*$, and $f = f_2(x,t)$ for $x \geq x^*$. If one assumes that f is interpolated using cubic spline interpolation, then f_1 and f_2 are both cubic polynomials, and their derivatives coincide at $x = x^*$ up until order 2. However, $f_1'''(x^*,t) \neq f_2'''(x^*,t)$, and as a result, f has a discontinuity at $x = x^*$ in its 3rd-order derivative. For a 4th-order Runge-Kutta method like RK4 the requirements of Theorem 1 are thus not fulfilled, and the bound in Eq. (2.14) cannot in general be expected to hold. However, since f_1 and f_2 are polynomials they are both infinitely differentiable and steps that are computed using only f_1 or f_2 satisfy the requirements of Theorem 1 and should have a local error bounded by Eq. (2.14). That is, for each step from x_n to x_{n+1} where both x_n,$x_{n+1} \leq x^*$, all the coefficients k_i (see Eq. (2.3)) used to take the step are computed using $f = f_1$, meaning that f is effectively continuous throughout

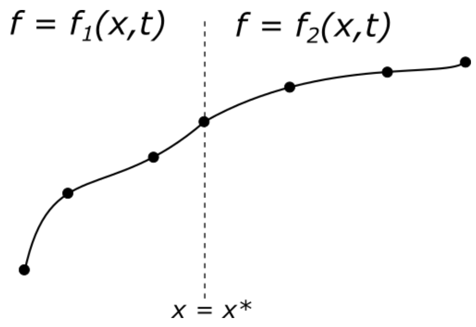


Figure 2.6: Illustration of an integrated trajectory where integration is stopped and restarted at the discontinuity at x^* in the velocity field $f(x, t)$. The dots represent steps taken by the solver, and the dashed line indicates the position x^*. This is the same illustration as in figure 1.2, included again for the convenience of the reader.

the step. The same is also true if both x_n, $x_{n+1} \geq x^*$ because then all the k_i are computed using $f = f_2$. Hence, steps taken entirely on one side of x^* have errors bounded by Eq. (2.14).

The problem with the discontinuous derivatives occurs only when $x_n < x^* < x_{n+1}$ or $x_n > x^* > x_{n+1}$, because then some k_i might be computed using f_1 and others using f_2. As a result, f does not fulfil the requirements for continuous derivatives, and the local error of the step is not bounded by Eq. (2.14). The effect of stopping and restarting integration at the discontinuities should now be more clear. If the n^{th} step of the integration is from x_n to x_{n+1}, where $x_n < x_{n+1} = x^*$ then $f = f_1$ throughout the step, and hence the local error will be bounded by Eq. (2.14). For the next step, $x^* = x_{n+1} < x_{n+1}$ and $f = f_2$ throughout the step. In other words, when stopping and restarting integration at the discontinuity in the field, both the step before the discontinuity and the step after satisfy the requirements for the error bound in Eq. (2.14).

The effect on the local error of the step when it stops at the discontinuity instead of stepping over it is what motivates the construction of the special-purpose integration methods in this study. The idea is that by forcing the integration to stop and restart at the discontinuities in the interpolated velocity field, one can obtain p^{th}-order accuracy with a spline interpolation scheme of lower order than the required $p + 2$. Specifically, if the special-purpose methods work as intended, one should for example be able to obtain 4^{th}-order accuracy for RK4 even with spline interpolation schemes of lower order than quintic.

Chapter 3

Methodology

As mentioned in chapter 1, the main goal of this study is to create special-purpose methods for numerical integration that stop and restart integration at the discontinuities in an interpolated velocity field. However, the event location procedure used in the special-purpose solvers can also be used for other applications, like for example creating Poincaré maps, so the process will first be described in general terms in section 3.1. Then section 3.2 will explain how the special-purpose integrators use the event location procedure to stop and restart integration at discontinuities in the interpolated velocity field. Two special-purpose methods are constructed, one based on the 4th-order Runge-Kutta method (RK4) and one based on the order 5(4) Dormand-Prince pair (DOPRI5), both introduced in section 2.1 in the previous chapter.

The accuracy and performance of the methods will be compared by using them on an interpolated version of the double gyre vector field, which will be introduced in section 4.2.1. The double gyre field has no analytical solution, so the errors of the methods will have to be estimated by purely numerical means. Section 3.3 will explain how this is done.

3.1 Dense Output and Event Location

This section will present the event location procedure for a general event. In the case of an interpolated velocity field, the event will be a discontinuity in the field, while for Poincaré maps the event will be intersection with the defined Poincaré section.

To explain the general methodology, consider as an example the function $\boldsymbol{x}(t) = (x(t), y(t), z(t))$, which is a solution to the ODE

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t). \tag{3.1}$$

When solving Eq. (3.1) numerically we get an approximation to $\boldsymbol{x}(t)$ given by a set of discrete points $\{\boldsymbol{x}_i\}_{i=0}^{k} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{n-1}, \boldsymbol{x}_n, \boldsymbol{x}_{n+1}, ..., \boldsymbol{x}_k\}$ after k steps with k corresponding finite step sizes h_i, which may or may not be identical, depending on the method. Assume that we know that the exact solution $\boldsymbol{x}(t)$ at some point crosses the plane (or point etc., depending on the dimensions of the problem) at $z = z^*$, and the goal is to find out exactly when this happens. Since it is not known in advance when this so-called event occurs, chances are that the exact solution $\boldsymbol{x}^* = (x^*, y^*, z^*)$ will not be in the set $\{\boldsymbol{x}_i\}$. Rather, it is likely that a situation like the one illustrated in figure 3.1 will occur, where $z_n < z^* < z_{n+1}$, for some subsequent numerically computed points $\boldsymbol{x}_n = (x_n, y_n, z_n)$ and $\boldsymbol{x}_{n+1} = (x_{n+1}, y_{n+1}, z_{n+1})$.

Defining t^* as the t such that $z(t^*) = z^*$ it can now be inferred that $t^* \in (t_n, t_{n+1})$. However, in many cases, including the ones considered in this study, a more precise solution is required. To obtain a more precise result one would need to also compute a solution between the points $\boldsymbol{x}_n$ and $\boldsymbol{x}_{n+1}$. One way of doing that is to decrease the step size h and solve Eq. (3.1) in the interval $[\boldsymbol{x}_n, \boldsymbol{x}_{n+1}]$, using the same numerical solver. However, when the number of output points becomes

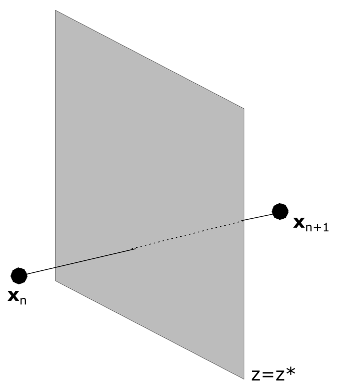


Figure 3.1: Illustration of an event occurring at the plane at $z = z^*$ between two points $\boldsymbol{x}_n = (x_n, y_n, z_n)$ and $\boldsymbol{x}_{n+1} = (x_{n+1}, y_{n+1}, z_{n+1})$ in the numerical approximation.

large, the classical Runge-Kutta methods introduced in section 2.1.1 become inefficient [22]. This motivated the construction of an alternative approach [40] that is now often referred to as dense output formulae or continuous extensions.

3.1.1 Dense Output

Dense output formulae are Runge-Kutta methods which provide continuous approximations to the solution between two numerically computed results x_0 and x_1, allowing interpolation to an arbitrary time $t^* \in [t_0, t_1]$. The idea is to create a polynomial that interpolates the points x_0 and x_1 and approximates the solution throughout the step [41]. Creating the approximating polynomial requires at most only a few additional function evaluations [42], and after creation it can be evaluated as many times as necessary without being computationally expensive [41]. This makes dense output formulas a much more efficient choice than integrating with a smaller step size.

Introducing a variable $\theta = (t - t_0)/(t_1 - t_0)$ so that $\theta \in [0, 1]$, the polynomial approximating the solution between x_0 and x_1 is given by a formula on the form

$$u(\theta) = x_0 + h \sum_{i=1}^{s^*} b_i(\theta) k_i, \tag{3.2}$$

where k_i are the k_i from Eq. (2.3) and $b_i(\theta)$ are polynomials to be determined, such that

$$u(\theta) - x(t_0 + \theta h) = \mathcal{O}(h^{p^*+1}),$$

where $x(t_0 + h)$ is the exact solution $x(t)$ evaluated at time $t = t_0 + h$, s^* is the number of stages of the dense output approximation, and p^* is the order of the dense output method.

The dense output solution for the 4th-order Runge-Kutta method in table 2.2 has order $p^* = 3$, and $s^* = 4$ stages, and its polynomials $b_i(\theta)$ in Eq. (3.2) are [22]

$$\begin{aligned} b_1(\theta) &= \theta - \frac{3\theta^2}{2} + \frac{2\theta^3}{3}, \\ b_2(\theta) &= b_3(\theta) = \theta^2 - \frac{2\theta^3}{3}, \\ b_4(\theta) &= -\frac{\theta^2}{2} + \frac{2\theta^3}{3}. \end{aligned} \tag{3.3}$$

For the order 5(4) Dormand-Prince pair (DOPRI5), the polynomials $b_i(\theta)$ in Eq. (3.2) are [22]

$$\begin{aligned}
b_1(\theta) =&\theta^2(3-2\theta)\cdot\tilde{b}_1\\
&\quad+\theta(\theta-1)^2-\theta^2(\theta-1)^2\cdot 5\cdot(2558722523-314030016\theta)/11282082432,\\
b_2(\theta) =&0,\\
b_3(\theta) =&\theta^2(3-2\theta)\cdot\tilde{b}_3+\theta^2(\theta-1)^2\cdot 100\cdot(882725551-15701508\theta)/32700410799,\\
b_4(\theta) =&\theta^2(3-2\theta)\cdot\tilde{b}_4-\theta^2(\theta-1)^2\cdot 25\cdot(443332067-314030016\theta)/1880347072,\\
b_5(\theta) =&\theta^2(3-2\theta)+\theta^2(\theta-1)^2\cdot 32805\cdot(23143187-3489224\theta)/199316789632,\\
b_6(\theta) =&\theta^2(3-2\theta)\cdot\tilde{b}_6-\theta^2(\theta-1)^2\cdot 55\cdot(29972135-7076736\theta)/822651844,\\
b_7(\theta) =&\theta^2(\theta-1)+\theta^2(\theta-1)^2\cdot 10\cdot(7414447-829305\theta)/29380423.
\end{aligned} \tag{3.4}$$

The coefficients $\tilde{b}_i$ on the right hand side of Eq. (3.4) are the 5th-order coeffiecients b_i in table 2.3.

As a check of the dense output solutions we verify that the resulting polynomial $u(\theta)$ for DOPRI5 satisfies

$$u(0) = x_0, \tag{3.5a}$$
$$u(1) = x_1, \tag{3.5b}$$
$$u'(0) = hf(x_0, t_0), \tag{3.5c}$$
$$u'(1) = hf(x_1, t_0+h). \tag{3.5d}$$

By evaluating Eq. (3.4) at $\theta = 0$ we can easily see that all coefficients $b_i(0) = 0$. Inserting this in Eq. (3.2) gives

$$u(0) = x_0 + h\sum_{i=1}^{7} b_i(0)k_i = x_0 + h\sum_{i=1}^{7} 0\cdot k_i = x_0,$$

i.e. Eq. (3.5a), as expected. Evaluating Eq. (3.4) at $\theta = 1$ we find $b_2(\theta=1) = b_7(\theta=1) = 0$ and for the remaining $b_i(\theta)$ (i.e. for $i = 1, 3, 4, 5, 6$) we find $b_i(\theta = 1) = \tilde{b}_i$. Comparing this result with table 2.3 we see that $b_i(\theta = 1) = \tilde{b}_i$ for all $i = 1, ..., 7$, which means that

$$u(1) = y_0 + h\sum_{i=1}^{7} b_i(1)k_i = y_0 + \sum_{i=1}^{7}\tilde{b}_i k_i = y_1,$$

where we recall that $\tilde{b}_i$ are the 5th-order coefficients for DOPRI5. The last equality comes from the general definition in Eq. (2.3) of Runge-Kutta methods. We see that Eq. (3.5b) is also satisfied.

To verify Eq. (3.5c) and Eq. (3.5d) we must first find the derivative $u'(\theta)$. Differentiating Eq. (3.2) with respect to θ we find that

$$u'(\theta) = h\sum_{i=1}^{7} b_i'(\theta)k_i \tag{3.6}$$

for the DOPRI5 method. By applying the chain rule we find the derivative of $b_1(\theta)$,

$$\begin{aligned}
b_1'(\theta) = {}&2\theta(3-2\theta) - 2\theta^2 + (\theta-1)^2 + 2\theta(\theta-1)\\
&+\big(2\theta(\theta-1)^2 + 2\theta^2(\theta-1)\big)\cdot 5\cdot\frac{2558722523-314030016\theta}{11282082432}\\
&+\theta^2(\theta-1)^2\cdot 5\cdot\frac{31403016}{11282082432}.
\end{aligned}$$

Evaluating $b_1'(\theta)$ at $\theta = 0$ and 1 we find

$$b_1'(0) = 1 \quad\text{and}\quad b_1'(1) = 0. \tag{3.7}$$

Similarly, we find

$$\begin{aligned} b_2'(0) &= 0 \quad \text{and} \quad b_2'(1) = 0 \\ b_3'(0) &= 0 \quad \text{and} \quad b_3'(1) = 0 \\ b_4'(0) &= 0 \quad \text{and} \quad b_4'(1) = 0 \\ b_5'(0) &= 0 \quad \text{and} \quad b_5'(1) = 0 \\ b_6'(0) &= 0 \quad \text{and} \quad b_6'(1) = 0 \\ b_7'(0) &= 0 \quad \text{and} \quad b_7'(1) = 1 \end{aligned} \tag{3.8}$$

for the remaining coefficients $b_2(\theta), ..., b_7(\theta)$. Evaluating the derivative in Eq. (3.6) at $\theta = 0$ and $\theta = 1$ by inserting the coefficients in Eq. (3.7) and Eq. (3.8) we find

$$u'(0) = h \sum_{i=1}^{7} b_i(0) k_i = h k_1 = h f(x_0, t_0),$$

and

$$u'(1) = h \sum_{i=1}^{7} b_i(1) k_i = h k_7 = h f(x_1, t_0 + h),$$

where the final equality in both these equations comes from the definition in Eq. (2.3) and the coefficients in table 2.3. Hence Eqs. (3.5) are all satisfied and we have verified that the coefficients in Eq. (3.4) constitute a dense output solution for the DOPRI5 method. The coefficients in Eq. (3.3) for RK4 can be verified in a similar manner.

3.1.2 Event Location

If we know that the trajectory crosses the event $z = z^*$ somewhere between z_n and z_{n+1}, and we have the dense output solution $u(\theta)$ between z_n and z_{n+1}, we can use an iterative root-finding method to find (an approximation to) the time t^* that solves the equation $z(t^*) = z^*$. The procedure for finding t^* is described by Hairer et al. [22] (pp. 195–196) and it starts by defining a function

$$g(t) = z(t) - z^*, \tag{3.9}$$

such that $g(t^*) = 0$. In figure 3.1 an interval $[z_n, z_{n+1}]$ that contains z^* has already been located, but if this interval is unknown one can simply compute the numerical approximation $g_k = z_k - z^*$ until the z_n and z_{n+1} that result in a sign change ($\text{sgn}(g_n) \neq \text{sgn}(g_{n+1})$) are found. Once the interval is known, $z(t)$ in Eq. (3.9) can be replaced by the dense output approximation $u(\theta)$ between z_n and z_{n+1} (Eq. (3.2)) and an iterative root-finding method can be used to find the root $\theta^* = (t^* - t_n)/(t_{n+1} - t_n)$ of the resulting equation. How θ^* is used will be described in the next section.

Common choices for root-finding methods are the bisection method and Newton's method, both of which are assumed known to the reader, but they are also described in Appendix A. Since the bisection method converges regardless of the choice of initial interval[1] that is the method of choice in this study.

3.2 Integration and Special-Purpose Methods

When computing trajectories from interpolated velocity fields, the events in question are the discontinuities of the field. With a 2D field that is given by exact values on grid points and an interpolated solution elsewhere, the discontinuities will be at the knots of the interpolated field. In the case that the knots coincide with the given data points, the discontinuities will be at the boundaries of the grid cells. This is the case for linear, cubic, and quintic spline interpolation. That

[1]As long as there is only one root in the interval, but in the cases considered here we can safely assume that that condition holds.

is, the events in this case are cell boundary crossings. We recall from section 2.2.2 that the knots are at the midpoints between data points for quadratic spline interpolation, and that as a result the discontinuities in the second order derivative are also at these midpoints. If the interpolation scheme is quadratic spline interpolation the lines the methods interpret as cell boundaries are then shifted accordingly.

Special-purpose variants of both RK4 and DOPRI5 are constructed that use the procedure described in the previous section to detect boundary crossings and find the exact time t^* at which the boundary was crossed. Once t^* is determined the step size can be adjusted to make the step stop at the boundary instead of stepping over it. This section will explain the selection of step size in general, and in the case that a special-purpose method detects that a cell border has been crossed.

3.2.1 RK4: Fixed Time Step

The regular RK4 method operates with a fixed step size h. In other words,

$$t_n = t_{n-1} + h_n \equiv t_0 + nh, \quad n = 1, 2, ..., N,$$

where t_N is the value of t for which the integration is terminated. Note that for the final step, $h_N = \min(h, t_N - t_{N-1})$ in order to ensure that the integration is stopped exactly at the desired end time t_N, even if h does not divide t_N evenly.

The special-purpose RK4 method that is constructed in this work also keeps a fixed step size h for most of the computations. However, if the integrator detects that a border is crossed between time t_{n-1} and time t_n, the θ^* giving the exact time of crossing t^*, where $t_{n-1} < t^* < t_n$, is found using the event location procedure described in the previous section. The time step h_n is then adjusted to a new step h_{adj} given by

$$\begin{aligned} h_{\text{adj}} = \theta^* h_n &= \left(\frac{t^* - t_{n-1}}{t_n - t_{n-1}} \right) h_n \\ &= t^* - t_{n-1}, \end{aligned}$$

where the last equality comes from the relation $h_n = t_n - t_{n-1}$. With this adjusted step size the new time t_n will be

$$t_n = t_{n-1} + h_{\text{adj}} = t_{n-1} + (t^* - t_{n-1}) = t^*.$$

In other words, the time step is adjusted so that instead of stepping across the cell border, integration will stop and restart exactly at the boundary. This stopping and restarting ensures that there are no discontinuities within any step of the integration, and the interpolated field will appear continuous to the solver. Based on the theory in section 2.3 this should in turn affect the accuracy of the result.

It should be noted that the event location procedure requires some extra computational work. Every call to the bisection method involves the same number of function evaluations as one step with the solver (4 for RK4), and after the time of intersection t^* has been found, a new step is taken with the adjusted step size. If a situation like the one illustrated in figure 3.2 occurs, where the trajectory has crossed two cell boundaries, one at $x = x^*$ and one at $y = y^*$, in one step, the time of intersection is computed in both directions. That is, both θ^*_x and θ^*_y is found, and then $\theta^* = \min(\theta^*_x, \theta^*_y)$. Such a situation thus involves not one but two calls to the bisection method. For RK4 this means that computational cost of the stopping and restarting at the cell boundary thus equals two or three additional steps for each boundary crossing.

Note also that this procedure does not guarantee that the adjusted step lands exactly at the cell boundary. The dense output formula given by Eq. (3.2) is an interpolated solution, so there is likely to be some small but finite error in the computation of t^*. It might therefore happen that the adjusted step is a little too short, so that the boundary is crossed again in the following step, thus leading to a new call to the bisection method and hence more computational work. It could also be too long, so that it steps over the boundary and does thus not achieve the expected reduction in local error (see section 2.3).

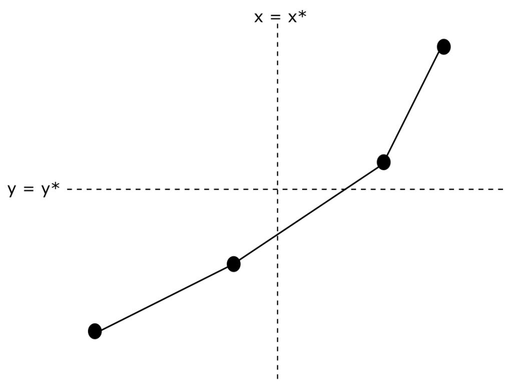


Figure 3.2: Illustration of an integrated trajectory crossing one discontinuity at $y = y^*$ and one discontinuity at $x = x^*$ in one single step.

3.2.2 DOPRI5: Variable Time Step

Both the regular and the special-purpose DOPRI5 method use the procedure for step size selection that was described in section 2.1.3. However, after a step of size h has been taken, the special-purpose method will first detect whether or not a cell border has been crossed. If no boundary has been crossed it will continue with the step size control procedure in section 2.1.3 to determine if the step is accepted or rejected, just like the regular DOPRI5 method. If, however, it finds that a border has been crossed between time t_n and time t_{n+1}, the event location procedure described in section 3.1 will be applied. The adjusted time step is found in the same manner as described in section 3.2.1 and will once again be

$$h_{\text{adj}} = t^* - t_n,$$

where t^* is still the time at which the cell border is crossed. The adjusted time step h_{adj} will then be fed to the step size control procedure. If the step is accepted the position is updated, and h_{new} is set to the original step size h for the next step. This will prevent the solver from taking unnecessarily many small steps right after a border has been crossed.

For both variants the final step h_N will also be adjusted to make the integration terminate exactly at a predefined t_N, as described above. Note also that it was decided to not implement FSAL. The reason for this choice was no other than that it simplifies implementation and makes it possible to use the same function signature for both RK4 and DOPRI5.

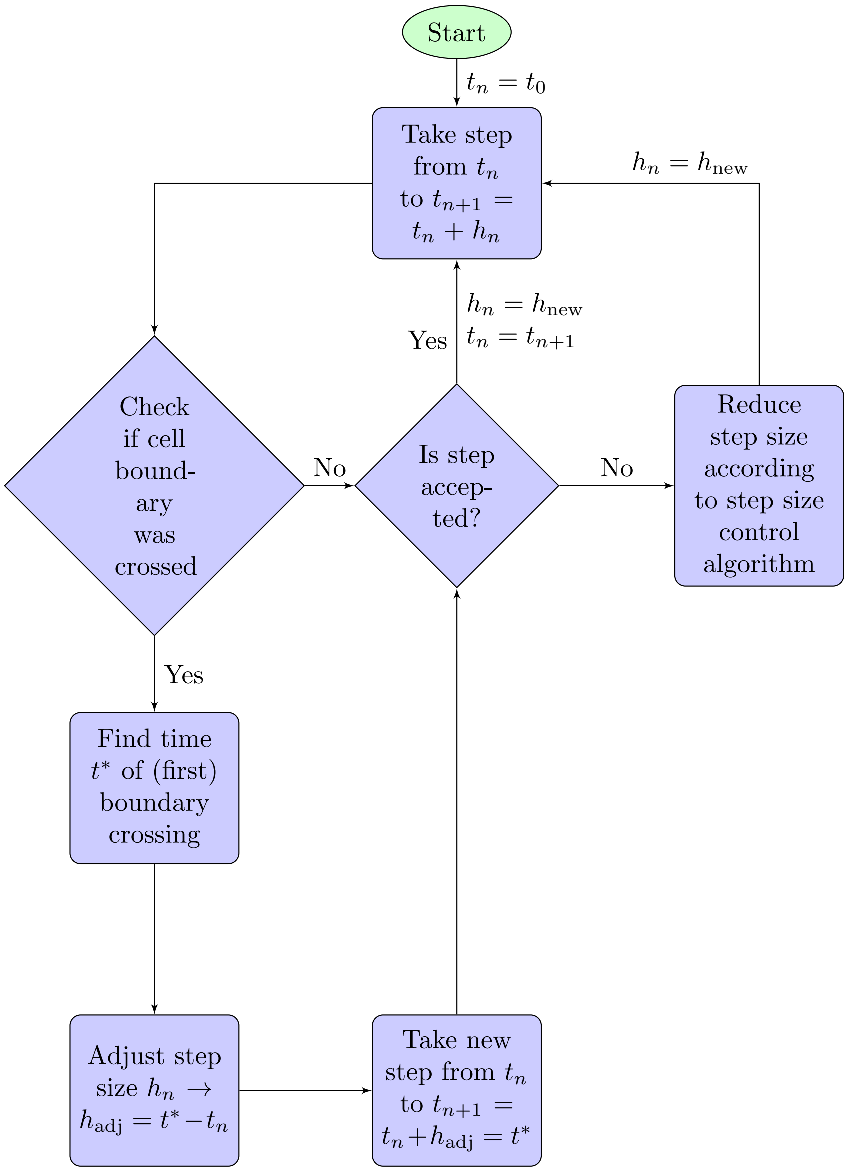
Just as for RK4, theory claims that stopping and restarting exactly at the discontinuities will reduce the global error of the method if the integrator assumes higher order continuous derivatives than is actually the case with the given interpolation scheme. This was explained in section 2.3.

Furthermore, since the step of size h_{adj} should not step over a discontinuity it is likely to have a smaller local error than the original step which stepped over the discontinuity. The adjusted step should therefore stand a greater chance of being accepted by the step size control algorithm, which was described in section 2.1.3. Less rejected steps means less wasted computational work, and the hope for the special-purpose method is that the amount of computational work saved on fewer rejected steps will outweigh the added computational work associated with detecting the borders.

The same notes about additional computational work that were presented for RK4 above are also valid for the special-purpose DOPRI5 method. That is, each call to the bisection method is equivalent in terms of the number of function evaluations to one additional step, and if a border is crossed in both directions during one step, like in figure 3.2, both intersection times are computed. In addition, there is always a chance that the adjusted step might be rejected, particularly if the error in t^* makes the adjusted step stop a little bit after the boundary. If that is the case, the whole procedure will start over. The consequence of an adjusted time step being rejected is thus greater in terms of computational work than if a non-adjusted step is rejected.

3.2.3 Algorithm

In short, the algorithm of the special-purpose methods can be described by the following flowchart:



Start
$t_n = t_0$
Take step from t_n to $t_{n+1} = t_n + h_n$
$h_n = h_{\text{new}}$
$h_n = h_{\text{new}}$ $t_n = t_{n+1}$
Yes
Check if cell boundary was crossed
No
Is step accepted?
No
Reduce step size according to step size control algorithm
Yes
Find time t^* of (first) boundary crossing
Adjust step size $h_n \to h_{\text{adj}} = t^* - t_n$
Take new step from t_n to $t_{n+1} = t_n + h_{\text{adj}} = t^*$

For the special-purpose RK4 method h_{new} is the same as the selected fixed step size h, regardless of whether the length of the previous step was adjusted or not. Note also that for a fixed-step solver every step is accepted. For the special-purpose DOPRI5 method on the other hand, every step must be controlled. If a boundary was crossed in the previous step then h_{new} for the special-purpose DOPRI5 method is set to the length h_n of the original step, otherwise h_{new} keeps the value assigned to it by the step size control algorithm.

3.3 Error Estimates

To investigate whether stopping and restarting integration at cell boundaries affects the accuracy of the numerically computed result, one can compare the global error. We wish to compute the global error for the different solutions, and compare to see which combination of numerical integration method and interpolation scheme gives the most accurate result. However, we recall that in order to compute the global error we need to know the exact solution, and in most situations where numerical integration is used, the exact solution is unknown.

The error must therefore be estimated purely by numerical means. This is done by computing a reference solution $\boldsymbol{x}_{\text{ref}}$, where here $\boldsymbol{x} = (x, y) \in \mathbb{R}^2$, with a small step size h_0, and comparing the other numerical approximations to this reference solution. The error in the reference solution is given by

$$E_{\text{ref}} = \boldsymbol{x}_N(h_0) - \boldsymbol{x}(t_N), \tag{3.10}$$

where $\boldsymbol{x}(t_N)$ is the true but unknown solution and $\boldsymbol{x}_N(h_0)$ is the numerically computed result at t_N with $h = h_0$. Using the reference solution instead of the true solution, the estimate for the global error of calculations with a longer time step h is given by

$$\begin{aligned} E_{\text{est}}(h) &= \boldsymbol{x}_N(h) - \boldsymbol{x}_N(h_0) \\ &= \boldsymbol{x}_N(h) - (E_{\text{ref}} + \boldsymbol{x}(t_N)) \\ &= E_{\text{true}} - E_{\text{ref}}, \end{aligned} \tag{3.11}$$

where $E_{\text{true}} = \boldsymbol{x}_N(h) - \boldsymbol{x}(t_N)$ is the error relative to the true solution and E_{ref} is the error in the reference solution, given by Eq. (3.10). Thus, the approximation $E_{\text{est}}(h)$ is a good estimate of the true global error E_{true} if $E_{\text{ref}} \ll E_{\text{true}}$. The idea is that the smaller the step size h of a numerical integration method is, the more accurate the approximated solution is, and as $h \to 0$ the numerical result converges to the exact result. That is,

$$\lim_{h_0 \to 0} E_{\text{ref}} = 0. \tag{3.12}$$

However, computers have limited precision when storing numbers, and this also affects the precision of arithmetic with floating-point numbers. A double precision floating-point number can be stored with approximately 16 significant digits, and every arithmetic operation can be thought of as introducing an error in the least significant digit. This error is called the roundoff error [38] and accumulates with each operation. Reducing the step size h of an integrator will increase the number of steps which in turn increases the total number of arithmetic operations. As a result, a decrease in h leads to an increase in the net contribution of this roundoff error, and eventually when h becomes small enough this contribution will dominate the global error. Reducing the time step further beyond this point will make the error increase rather than decrease, since the roundoff error will just keep accumulating. Equation (3.12) is thus not entirely true, and the time step h_0 of the reference solution must therefore be chosen with caution.

The best way to select an appropriate value for h_0 is to analyse the convergence of a selected numerical integrator for small h. Plotting the global error as a function of h for small h one should be able to identify the critical value h_c, after which roundoff errors start to dominate. The value h_c would be a good choice for h_0. The results of such an analysis are presented in section 4.2.2 in chapter 4.

Chapter 4

Numerical Results and Discussion

In this chapter, the results of numerical computations using the event location procedure explained in section 3.1 will be presented.

To test whether the event location procedure works, it was used in an attempt to show the fractal properties of the Lorenz attractor in a Poincaré map. Results from this test are presented in section 4.1. The next section, 4.2, investigates some of the properties of a vector field called the double gyre field, which will be used as the velocity field when the performance of the different integration methods are compared in the final section of this chapter. The double gyre vector field has no analytical solution, so section 4.2.2 will explain how the reference solutions needed for the error estimates introduced in section 3.3 are computed. Finally, section 4.3 will present the assessment of the performance of the special-purpose integrators described in section 3.2 compared to their regular counterparts, when combined with different orders of spline interpolation.

4.1 Poincaré Map of the Lorenz Attractor

As a test of the event location procedure a Poincaré map of the Lorenz attractor is computed. The Lorenz attractor was discovered by E. Lorenz [43] when he studied a simplified model of convection rolls in the atmosphere, given by the three equations

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= rx - y - xz \\ \dot{z} &= xy - bz, \end{aligned} \tag{4.1}$$

now known as the Lorenz equations. For certain values of the parameters σ, r, and b, the Lorenz system settles into a structure now known as a strange attractor. The Lorenz attractor appears when the parameters of the Lorenz equations in Eq. (4.1) are set to $\sigma = 10$, $b = \frac{8}{3}$, and $r = 28$, and a trajectory is created starting from the initial position $(x_0, y_0, z_0) = (0, 1, 0)$. If the trajectory is visualized in the (x, z) phase space the phase portrait takes a butterfly-like shape. This phase portrait is perhaps the most well-known phase portrait in the field of chaos theory, and it is shown in figure 4.1.

In figure 4.1 it appears as if the trajectory crosses itself repeatedly, but this is impossible (see e.g. [20] p. 150), and when studying the trajectory in all three dimensions one can see that no such self-intersections occur. The 3D structure looks like a pair of surfaces that appear to merge into one. Again, this is impossible, and upon closer inspection Lorenz [43] found that the surfaces do remain distinct surfaces. He also discovered that instead of two surfaces, the structure consisted of an infinite complex of surfaces, where each surface is extremely close to one of the two surfaces that appear to merge. In other words, the Lorenz attractor is a so-called fractal.

A fractal is a geometric structure that has certain properties, including self-similarity, non-integer dimension, and structure even at arbitrarily small scales. For more details on fractals and these

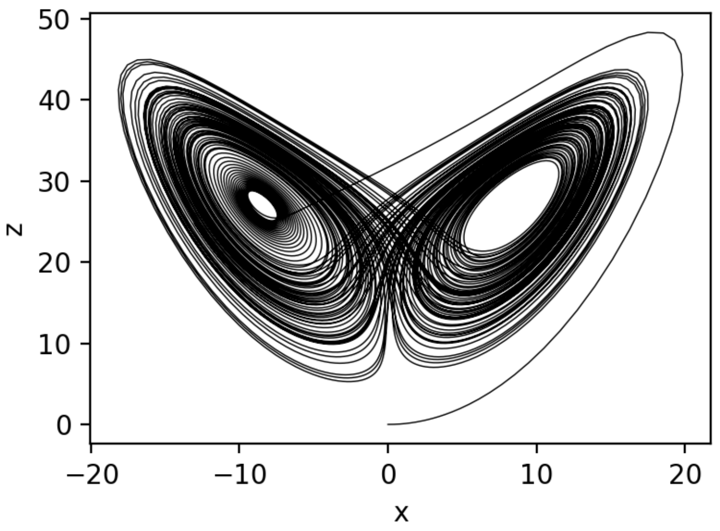


Figure 4.1: Phase potrtrait in the (x, z) phase space of the Lorenz attractor.

properties, see e.g. [20] pp. 405–423. The non-integer fractal dimension of the Lorenz attractor has been numerically computed to be about 2.05 [44]. Not many plots have been made that actually show these last two fractal properties of the Lorenz system, but Viswanath [45, 46] and Viswanath and Şahutoğlu [47] are among those who have done it.

As a test of the event location procedure an attempt has been made to reproduce the plots in figures 4 and 5 in [45], which show the fractal properties of the Lorenz attractor in a Poincaré map. Recall that a Poincaré map is a mapping of the intersections between the trajectory $\boldsymbol{x} = (x, y, z)$ and a plane S called a Poincaré section. Figure 4.2 shows the Poincaré map of the Lorenz system at the Poincaré section given by the plane located at $z = 27$, computed using RK4 with timestep $h = 0.0005$ and the event location procedure described in section 3.1.

At first glance the structure in figure 4.2 does not look like a fractal. However, inspired by Viswanath, only the points in a small interval centered around the position $(-13.764318918851 49, -19.57849004063768)$, denoted (x_c, y_c), are plotted, and the result is shown in figure 4.3a. The dashed lines in figure 4.2 and figure 4.3 cross at the point (x_c, y_c). It turns out that what initially looks like just one line is actually two lines. In [45] Viswanath zooms in again and finds that each of these two lines is also actually two lines. This self-similarity should in theory go on forever, and is one of the fractal properties of the Lorenz attractor. Zooming in on one of the lines in figure 4.3a gives the result in figure 4.3b. When zooming in this far, the results are not as clear as they are in [45], since there are relatively few points in the frame. However, the points that are there do show a clear tendency to fall along two separate lines.

Although there are not enough points in the Poincaré map in figure 4.2 to completely recreate both plots in figure 5 in [45], one can see that the lines are in the same place and with the same distance between them as they are in figure 5 in [45]. The fact that the results produced here correspond so well with the results from [45] at such small margins is a good indication that the event location procedure works.

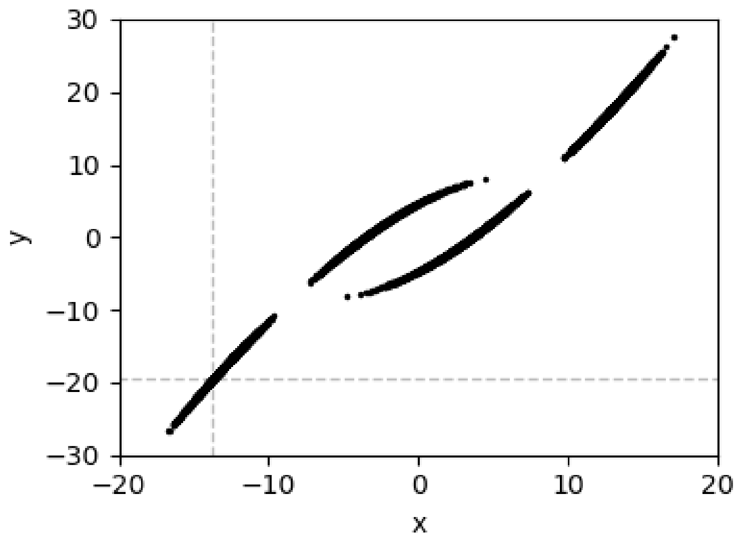
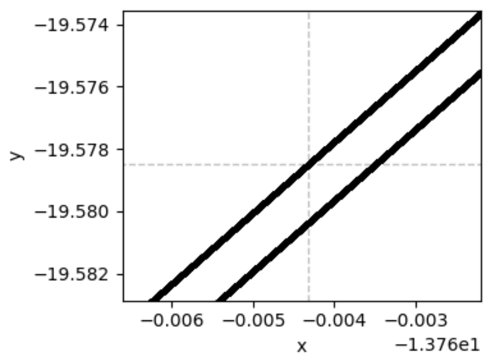
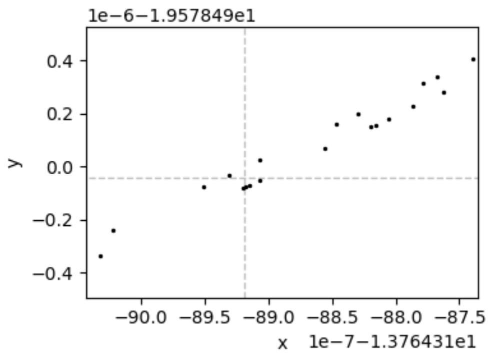


Figure 4.2: Poincaré section at $z = 27$ of the Lorenz attractor.



(a) A section of 4.2 enlarged.



(b) A section of 4.3a enlarged.

Figure 4.3: A section of figure 4.2 enlarged in an attempt to show the fractal properties of the Lorenz attractor.

4.2 The Interpolated Velocity Field

4.2.1 The Double Gyre Field

The vector field that is used in the remaining part of this study is a 2D field called the double gyre field, which is given by

$$\begin{aligned} v_x &= -\pi A \sin\left(\pi g(x,t)\right) \cos\left(\pi y\right) \\ v_y &= \pi A \cos\left(\pi g(x,t)\right) \sin\left(\pi y\right) \frac{\partial g(x,t)}{\partial x}, \end{aligned} \tag{4.2}$$

where

$$\begin{aligned} g(x,t) &= a(t)x^2 + b(t)x, \\ a(t) &= \epsilon \sin\left(\omega t\right), \\ b(t) &= 1 - 2\epsilon \sin\left(\omega t\right). \end{aligned}$$

The velocity field $\boldsymbol{f}(x,y,t) = [v_x(x,y,t), v_y(x,y,t)]$ is defined in the region $0 \leq x \leq 2$, $0 \leq y \leq 1$ in the xy-plane. The equations for the field are taken from Nordam et al. [48]. The double gyre field consists of two vortices rotating in opposite directions. For nonzero values of the parameters ϵ and ω, the line separating the vortices will oscillate left and right along the x-axis as time passes.

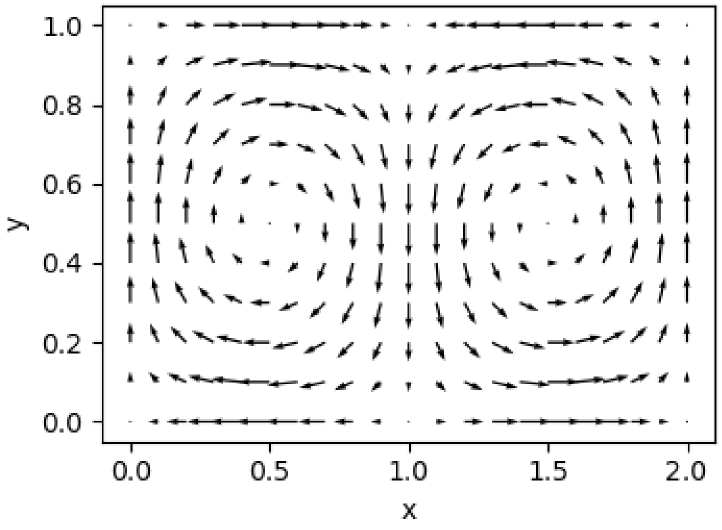


Figure 4.4: The double gyre vector field.

For the computations in this study the parameters are chosen to be $A = 0.1$ and $\omega = 0$ (ϵ arbitrary since $\omega = 0$). Setting $\omega = 0$ makes the double gyre field time independent, i.e. $\boldsymbol{f}(\boldsymbol{x}, t) = \boldsymbol{f}(\boldsymbol{x})$. Figure 4.4 shows what the field looks like with these parameters, and figure 4.5 shows what some trajectories $\boldsymbol{x}$ look like in the (x, y) phase space for several randomly generated initial positions $\boldsymbol{x}_0 = (x_0, y_0)$.

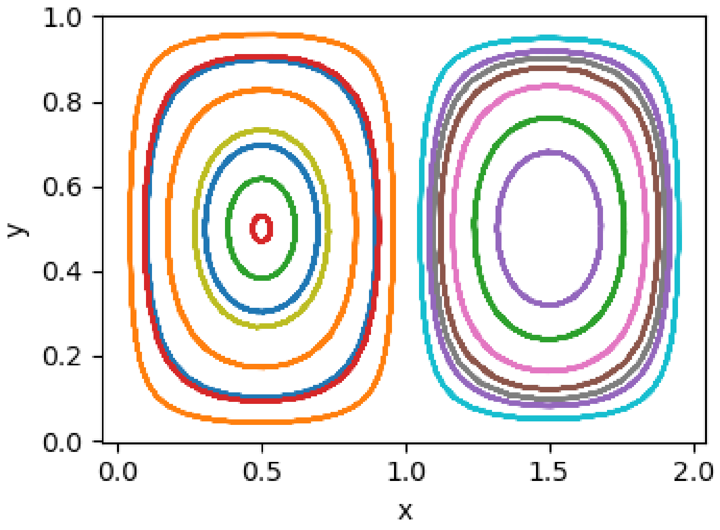


Figure 4.5: Double gyre trajectories under the constraint $\omega = 0$.

To study the effect of the special-purpose numerical integration methods versus the regular numerical integration methods, an interpolated version of the double gyre field is defined. The domain $[0, 2] \times [0, 1]$ is divided into a 0.1×0.1 grid, where the field value $\boldsymbol{f}(x_i, y_i)$ at the grid cell corners (x_i, y_i) are given by the double gyre field defined in Eq. (4.2). For arbitrary points (x, y) within the grid cells the field is interpolated using `RectBivariateSpline` from the `scipy.interpolate` library for Python, which is a method for spline interpolation in two dimensions that allows the user to specify the degree of the spline. For further information on `RectBivariateSpline`, including documentation and source code, see [49].

Partial Derivatives

To support the discussion in section 4.3 later in this chapter, it is helpful to first have a look at the derivatives of the double gyre field, which is given by Eq. (4.2) and visualised in figure 4.4, when it is interpolated with splines of different order n. In order to do that, a short trajectory starting from $\boldsymbol{x}_0 = (x_0, y_0) = (0.1, 0.2)$ was simulated from time $t_0 = 0$ to time $t = 5$, using RK4 with a fixed time step $h = 0.005$. Then the velocity field was evaluated and differentiated at every position (x, y) in the trajectory. For each interpolation scheme the partial derivatives $\partial^m v_x/\partial x^m$ and $\partial^m v_y/\partial y^m$ of the interpolated field were computed, where $m \leq n - 2$.

`RectBivariateSpline`, which was used to interpolate Eq. (4.2), has an associated method `__call__`. The `__call__` method has the option to evaluate the interpolated field itself as well as its derivatives up to and including order $n-2$, where n is the order of the field. Recall that for a spline interpolation scheme of order n, derivatives up to and including order $n-2$ are continuous. The `__call__` method was therefore used to evaluate the field and compute its continuous derivatives. The derivative of order $n - 1$ was then computed from the derivative of order $n - 2$ (or the field itself for linear splines) using forward finite differences.

Figure 4.6 shows the components v_x and v_y of the double gyre field, as well as the first order partial derivatives $\partial v_x/\partial x$ and $\partial v_y/\partial y$, when the field is interpolated with linear spline interpolation. The dashed lines indicate the positions of cell borders. We recall that linear interpolation has a discontinuous first order derivative, and that since the knots are at the data points the discontinuities can be expected to be at the grid cell boundaries. The bottom panels of figure 4.6 confirms this.

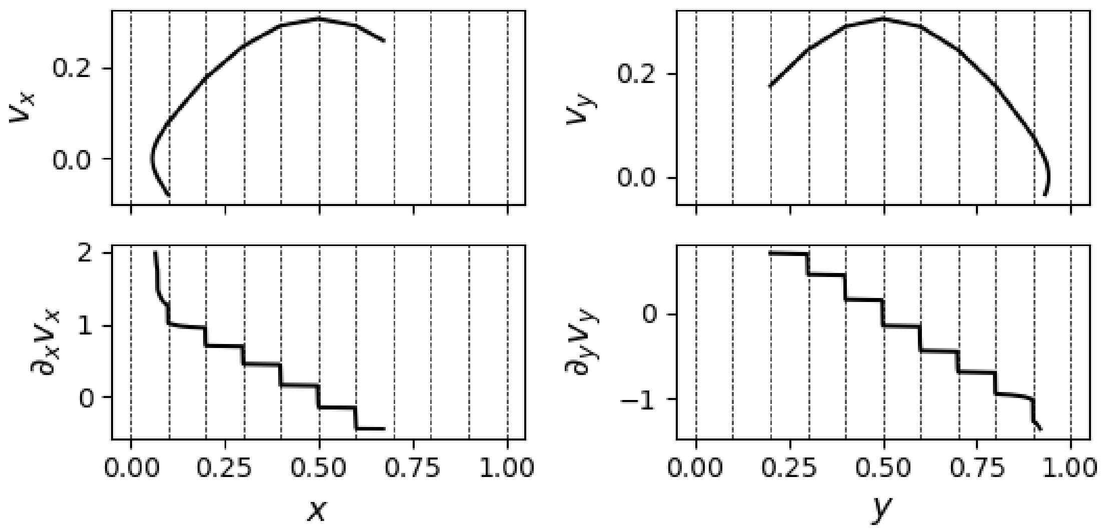


Figure 4.6: The components v_x and v_y of the double gyre field given by Eq. (4.2) and their partial derivatives, when interpolated using linear spline interpolation.

Using quadratic spline interpolation, we expect from the theory in section 2.2.2 that the first order derivative is continuous, but the second order derivative is not. Figure 4.7 shows the components v_x and v_y of the velocity field (top row) as well as the partial derivatives $\partial v_x/\partial x$, $\partial v_y/\partial y$ (middle row), and $\partial^2 v_x/\partial x^2$, $\partial^2 v_y/\partial y^2$ (bottom row). As expected, the first order partial derivatives are continuous and the second order partial derivatives are discontinuous. Notice that the discontinuities are at the midpoint between the cell boundaries, just like they are at the midpoint between data points in the bottom panel of figure 2.4 in section 2.2.2.

Using cubic spline interpolation we expect from the general theory in section 2.2.3 that the first order derivative is smooth and continuous, and that the second order derivative is continuous, but not smooth. Furthermore, we expect that the third order derivative is discontinuous at the cell borders. Figure 4.8 shows the components v_x and v_y of the field (top panel), as well as the partial derivatives up to order 3. As expected, the third order partial derivatives $\partial^3 v_x/\partial x^3$ and $\partial^3 v_y/\partial y^3$

are discontinuous at the cell boundaries.

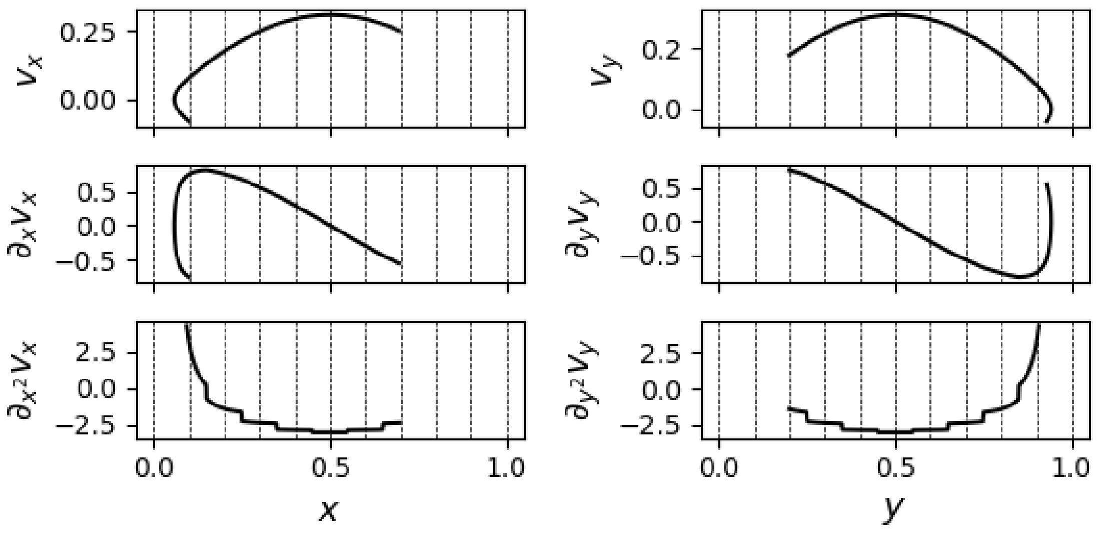


Figure 4.7: The components v_x and v_y of the double gyre field given by Eq. (4.2) and their partial derivatives, when interpolated using quadratic spline interpolation.

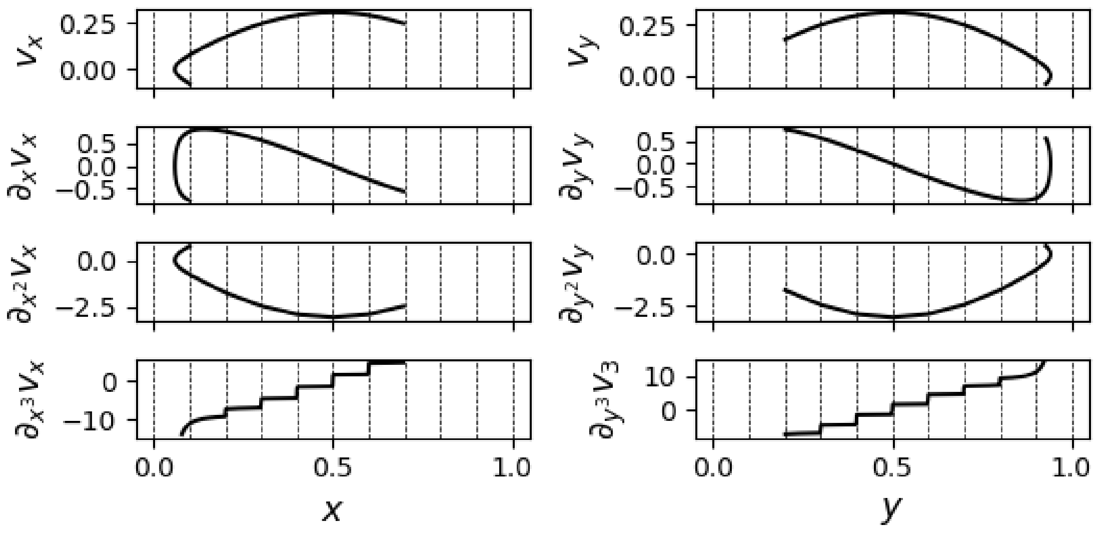


Figure 4.8: The components v_x and v_y of the double gyre field given by Eq. (4.2) and their partial derivatives, when interpolated using cubic spline interpolation.

Quintic spline interpolation is of order 6, and should therefore have 4 continuous derivatives. Like for the other interpolation schemes of even order the discontinuities in the first discontinuous derivative should be at the cell boundaries. Figure 4.9 shows the components of the velocity field and it's derivatives. Again, results correspond well with theory.

One thing to note about the discontinuous derivatives in figures 4.6, 4.7, 4.8, and 4.9 is that they are not piecewise constant, as one might expect from section 2.2 and the plots of the derivatives therein. The reason for this is that when moving along the trajectory the position changes in both the x- and the y-direction at the same time. If instead one had kept e.g. y constant and moved only in the x-direction, then the $(n-1)^{\text{th}}$ derivative of v_x would be piecewise constant.

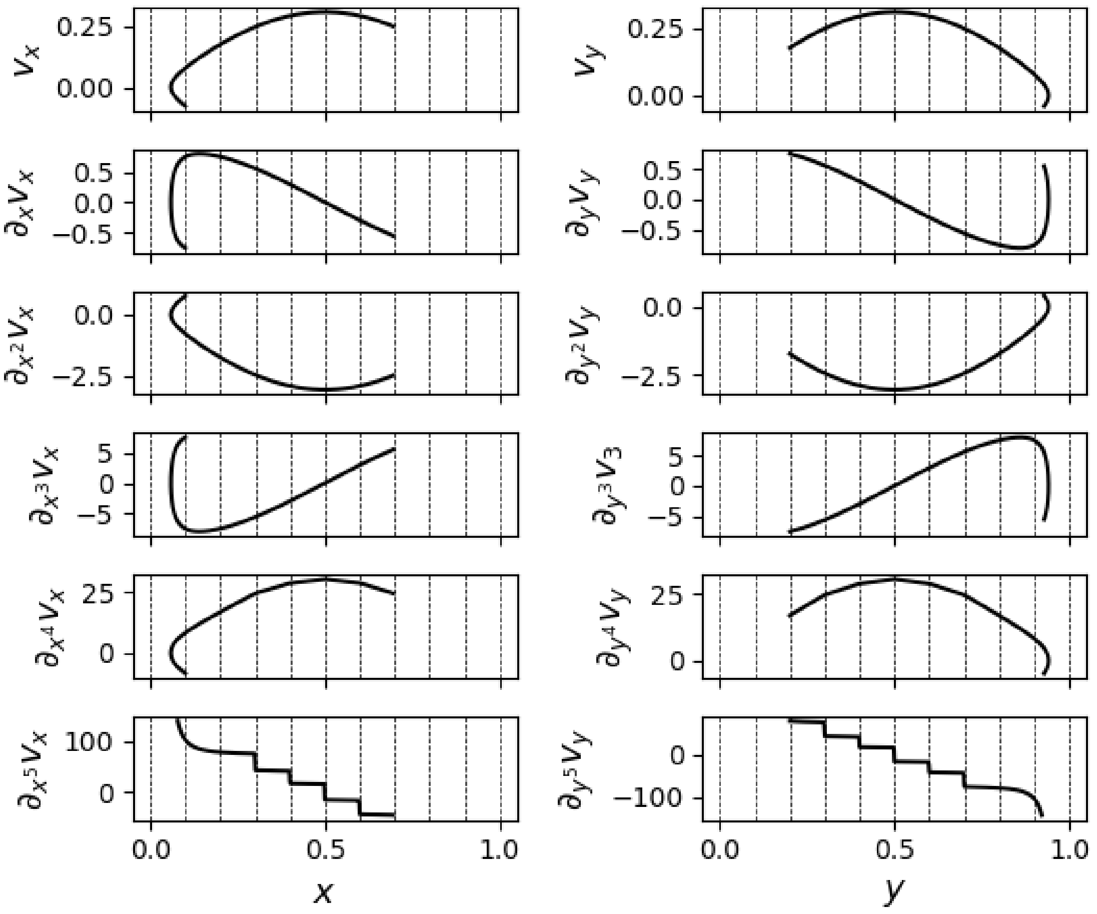


Figure 4.9: The components v_x and v_y of the double gyre field given by Eq. (4.2) and their partial derivatives, when interpolated using quintic spline interpolation.

4.2.2 Reference Solutions

The ODE

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t),$$

where $\boldsymbol{f}$ is the double gyre field given by Eq. (4.2), has no known analytical solution. In order to estimate the errors of the numerical solutions in the next section one must therefore approximate the exact solution, $\boldsymbol{x}(t)$, by a numerically computed reference solution, $\boldsymbol{x}_{\text{ref}}$, as was explained in section 3.3. The error estimate E_{est} given by Eq. (3.11) is only a good approximation to the true error $E_{\text{true}} = \boldsymbol{x}_N - \boldsymbol{x}(t_N)$, where $\boldsymbol{x}_N$ is the numerically computed result after N steps, and $\boldsymbol{x}(t_N)$ is the exact solution at $t = t_N$, if the error of the reference solution $\boldsymbol{x}_{\text{ref}}$, given by Eq. (3.10), is very small. That is, in order to obtain an acceptable estimate E_{est} of the global error of a numerically computed solution $\boldsymbol{x}_N$ one must compute a reference solution $\boldsymbol{x}_{\text{ref}}$ with a global error E_{ref} that is as small as possible.

The reference solutions in this section are computed using RK4 with a small time step h_0. Recall from section 3.3 that when the step size h is small enough the global error is dominated by the roundoff error, so one must be cautious when selecting h_0 for the reference solution. In order to find an appropriate choice for h_0 the global error of RK4 is plotted as a function of step size h in figure 4.10, for a function with a known analytical solution.

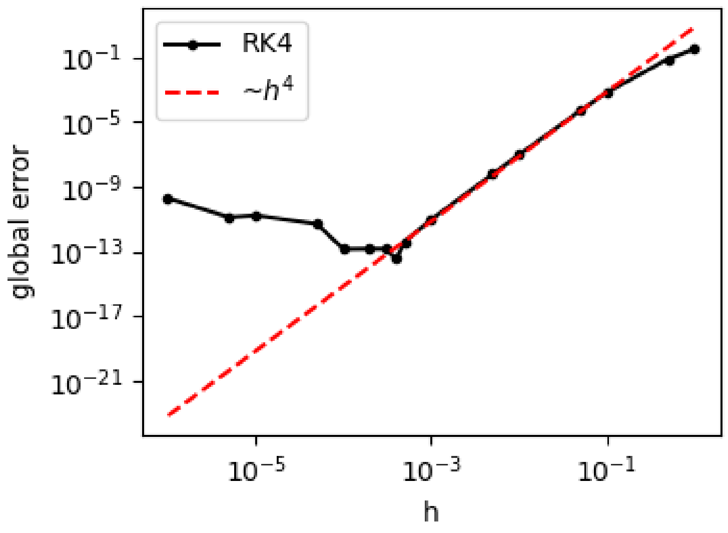


Figure 4.10: Global error as a function of time step h for RK4.

The dashed line in figure 4.10 indicates a slope proportional to h^4, and it is clear that the error scales with h^4 as expected until $h \approx 0.0004$. Beyond this point the roundoff error starts to dominate and reducing the step size further makes the error increase rather than decrease. Based on this result, the step size h_0 for the reference solution $\boldsymbol{x}_{\text{ref}}$ was chosen to be $h_0 = 0.0004$ in order to get a reference solution that is as accurate as possible. With this h_0 the global error of the reference solution, given by Eq. (3.10), should be small enough to make the global error estimate given by Eq. (3.11) a good approximation to the true global error of the numerically computed solution. Note that since different orders of interpolation result in different solutions between the given data points, the same reference solution cannot be used in all cases. Therefore, a new reference solution was computed for each order of interpolation and for every new initial position, in each case using RK4 with a fixed step size $h_0 = 0.0004$.

4.3 Accuracy and Performance

In this section the results comparing the accuracy and performance of the special-purpose integrators to their regular counterparts are presented. Section 4.3.1 will present and discuss the results for the two RK4 methods, section 4.3.2 will present and discuss the results for the two DOPRI5 methods, and section 4.3.3 will discuss how all four methods compare to each other.

4.3.1 RK4

Figure 4.11 illustrates how the regular RK4 method (labelled RK4 (r) in the figures in this section) crosses cell boundaries compared with how the special-purpose RK4 method (labelled RK4 (s-p) in the figures) crosses cell boundaries. The horizontal and vertical lines in the plots indicate where the borders are, and the crosses (RK4 (r)) and dots (RK4 (s-p)) mark the position of each step. As expected, the special-purpose RK4 method ensures that integration is stopped and restarted exactly at cell boundaries, while the regular RK4 method simply steps across them.

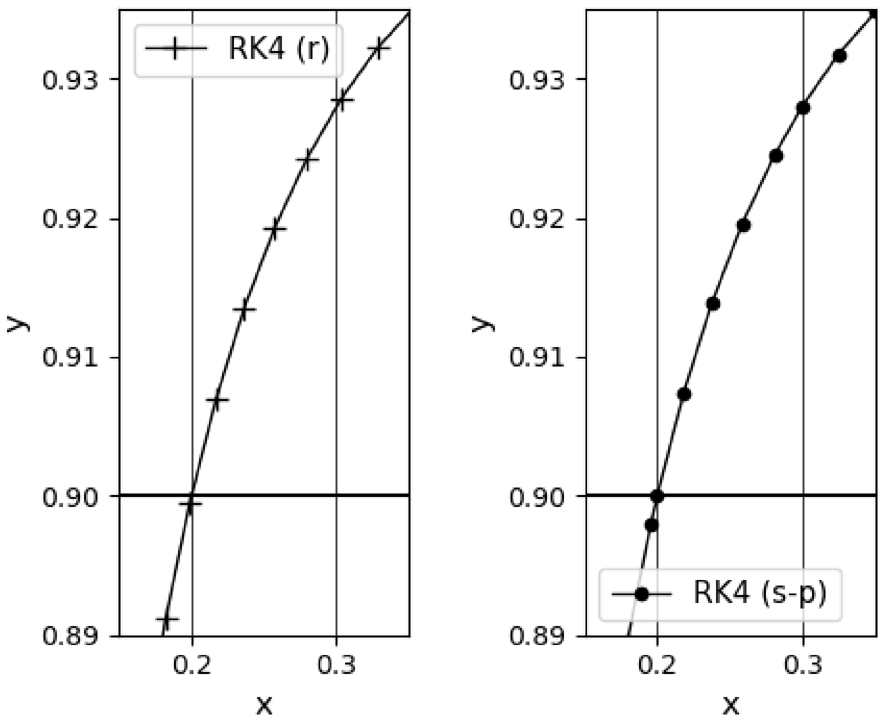
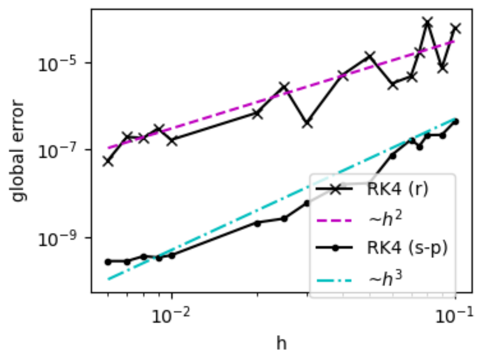


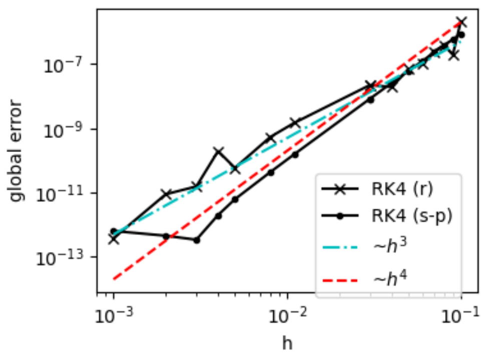
Figure 4.11: Example of cell border crossing in a trajectory computed using the regular RK4 method (left) and the same cell borders crossed in a trajectory using the special-purpose RK4 method (right). The horizontal and vertical lines indicate cell borders. In both cases the field was interpolated using linear spline interpolation.

To compare the accuracy of the two methods, the global error is plotted as a function of time step h for each method and each interpolation scheme. As described in section 3.3, the global error is approximated by Eq. (3.11), i.e. by comparing the numerical result to a reference solution obtained by solving Eq. (4.2) using RK4 with the small time step $h_0 = 0.0004$ found in section 4.2.2. Figure 4.12 shows the result when the field is interpolated using linear (fig. 4.12a), quadratic (fig. 4.12b), cubic (fig. 4.12c), and quintic (fig. 4.12d) spline interpolation. The dashed lines are included to indicate slope.

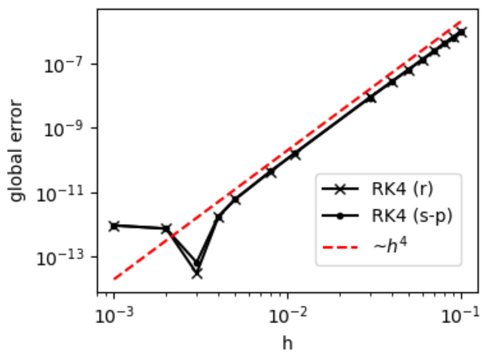
In the case of linear spline interpolation it is clear that the regular RK4 method performs worse than the special-purpose RK4 method in terms of global error at the end of computations. The global error of the regular RK4 method scales with h^2, meaning that combined with linear spline



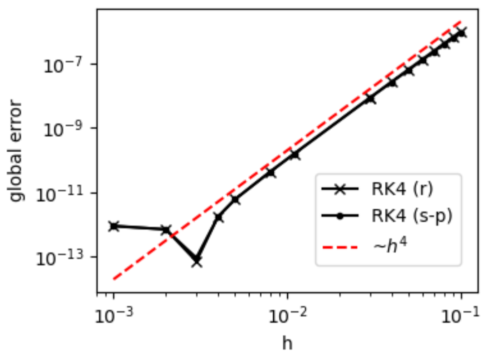
(a) Linear



(b) Quadratic



(c) Cubic



(d) Quintic

Figure 4.12: Global error versus time step h for regular and special-purpose RK4 combined with different orders of spline interpolation.

interpolation, this method is effectively 2nd-order accurate. The special-purpose method, on the other hand, scales with h^3, meaning that it is effectively 3rd-order accurate when combined with linear spline interpolation. This makes sense, because by stopping and restarting at the cell boundaries each step is computed without stepping over any discontinuities, as was explained in section 2.3. Recall from figure 4.6 that the first order derivatives are discontinuous at the cell boundaries. The derivative then becomes effectively continuous within each step, and as a result we can expect accuracy of higher order compared to the regular method that is stepping over the discontinuities.

Increasing the degree of interpolation by one, to quadratic interpolation, it is reasonable to expect that the order of accuracy of the regular RK4 method also increases. In this case the interpolated field has first-order derivatives that are continuous at cell borders. Second-order derivatives exist, but they are not continuous. Figure 4.12b shows the plot of global error versus time step h when the field is interpolated using quadratic spline interpolation. The blue dashed line is proportional to h^3. One can see that the regular RK4 method now scales with h^3.

Recall from figure 4.7 that the discontinuities in the derivatives of the velocity field are shifted to the midpoints between cell boundaries when the field is interpolated using quadratic spline interpolation. Therefore, the lines that the special-purpose method interpret as cell boundaries are shifted by 1/2 grid spacing when quadratic spline interpolation is used, so that integration will still be stopped and restarted at the discontinuity. Figure 4.13 shows the same part of the trajectory as figure 4.11, with the only difference being that in figure 4.13 the velocity field is interpolated using quadratic spline interpolation and in figure 4.11 it is interpolated using linear spline interpolation. The vertical dashed line in figure 4.13 indicates the shifted position of the cell boundary.

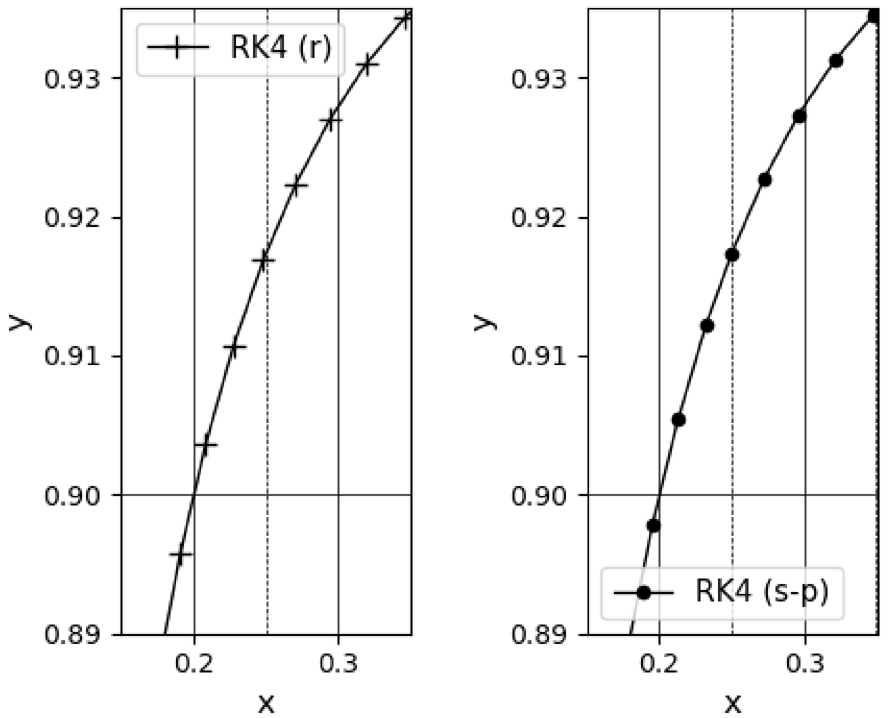


Figure 4.13: Example of cell border crossing in a trajectory computed using the regular RK4 method (left), and the same cell borders crossed in a trajectory computed using the special-purpose RK4 method (right). In both cases the field was interpolated using quadratic spline interpolation. The horizontal and vertical solid lines indicate the actual cell boundaries, and the dashed line indicates the position that the solver interprets as a cell boundary when quadratic spline interpolation is used.

One can see in figure 4.13 that the special-purpose method now stops at the shifted boundary instead of the actual boundary, while the regular method steps over both. As a result one would expect the special-purpose method to be more accurate, and looking back at figure 4.12b again, one can see that this is true. With quadratic spline interpolation, the global error of the special-purpose RK4 method scales with h^4, making it effectively 4th-order accurate, which is the highest order one can expect for RK4.

With cubic spline interpolation figure 4.8 showed that partial derivatives of the interpolated double gyre field are continuous up to, but not including, order three. The third-order partial derivatives are discontinuous at the cell borders, just as predicted from the theory in section 2.2.3, and the special-purpose solver will again stop and restart integration at the cell boundaries. As figure 4.12c shows, the increased order of interpolation once again leads to an increase in order of accuracy for the regular RK4 method. For the special-purpose method the order remains the same, meaning that the global error of both methods now scales as h^4. Comparing with the results for linear and quadratic spline interpolation these results are not surprising, but recalling the theory in section 2.3 one might find them to be a little unexpected.

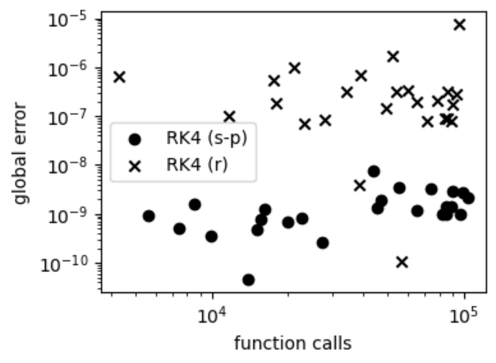
The substance of section 2.3 was that one cannot expect order p accuracy of a numerical integration method unless the derivatives of the right hand side f of the ODE exist and are continuous up to and including order p. Looking back at the results so far in this section it is clear that none of them actually comply with this. Specifically, theory would suggest that one should not expect higher order of accuracy than 2 with cubic splines, since f then only has continuous derivatives up to and including order 2. Still, the results presented here show that 4th-order accuracy is achieved with cubic interpolation. This indicates that order p accuracy is only guaranteed with p continuous derivatives, but it can be achieved even with discontinuities. Results from similar computations in [24] and [48] have followed the same pattern as here, although they provide no theoretical explanation for why. For future research on the topic it would be very interesting to try to find the reason.

The aforementioned theory predicts that quintic spline interpolation is the first order of spline interpolation that has enough continuous derivatives to guarantee 4th-order accuracy. Since 4th-order accuracy is achieved already at two orders lower, it could be interesting to see if the two interpolation schemes give any difference at all in accuracy with RK4. Figure 4.12d shows how the global error scales with time step h when using quintic spline interpolation. Comparing figures 4.12c and 4.12d one can see that there is hardly any difference between the two. The order is the same, and the numerical value of the accuracy is about the same too.

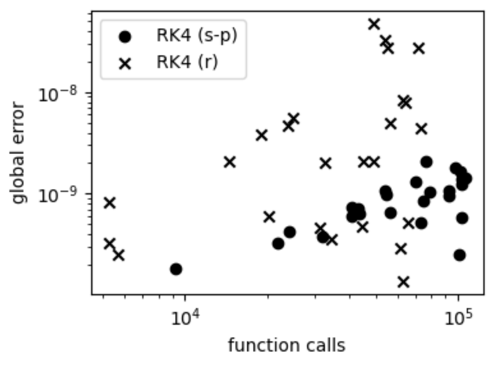
Finally, it is interesting to have a look at the computational costs of the two methods. In this study the number of calls to the function f on the right hand side of the ODE is chosen as a measure of work, rather than the run time of the simulation. The reason for this choice is that the number of function evaluations is a more objective measure. The run time would depend on the machine used to run the simulations, and is more susceptible to variations. Furthermore, run time is in practice proportional to the number of function calls (see [24]).

To investigate the computational costs of the methods, a general analysis of numerical error versus the number of function calls is performed using a Monte Carlo approach. For each combination of integration method and interpolation scheme an ensemble of 25 simulations is run, where each simulation has a random initial position $\boldsymbol{x}_0$ and a random simulation time from $t_0 = 0$ to t_N, where $t_N \in [1, 250]$. Figure 4.14 shows the global error plotted versus the number of function calls for the different interpolation schemes. Every dot/cross in the figure marks the global error and the number of function evaluations for one ensemble member. The time step is kept fixed at $h = 0.01$ for all simulations.

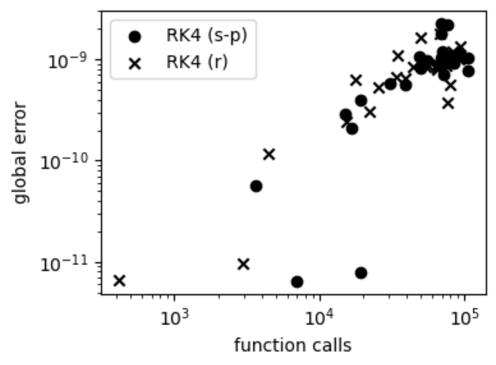
With a regular fixed-step Runge-Kutta integrator the number of steps is given by $N = (t_N - t_0)/h$, and the number of function calls is given by sN, where s is the number of stages. This means that the number of function calls only depends on the duration of the simulation, and the interpolation scheme should not matter. With the special-purpose fixed-step solver constructed here, the number of function calls also depends on how many cell boundaries are crossed, which in turn depends on the initial position in addition to the duration of the simulation. Again, interpolation scheme should not matter. The result in figure 4.14 confirm this - the interpolation scheme does not seem affect the average number of function evaluations for either of the two methods.



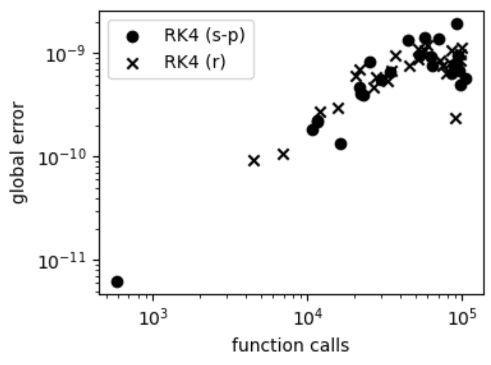
(a) Linear



(b) Quadratic



(c) Cubic



(d) Quintic

Figure 4.14: Global error versus number of function evaluations for regular and special-purpose RK4 combined with different orders of spline interpolation.

Furthermore, one can see that the results in figure 4.14 seem to agree well with the results in figure 4.12. Figure 4.14a shows that with linear spline interpolation the average global error is lower for the special-purpose RK4 method than for the regular RK4 method, which is as expected from figure 4.12a. The same goes for quadratic spline interpolation (fig. 4.14b), for which we also expect from figure 4.12b that the special-purpose method has a higher accuracy. The results for cubic and quintic splines in figures 4.14c and 4.14d show that the average global error is about the same for the two methods, which is as expected from the plots in figure 4.12c and figure 4.12d.

To summarise this section one can say that for the lower-order interpolation schemes the special-purpose RK4 methods do increase the order of accuracy. This improvement does come at the cost of a few more function evaluations, but the difference is so small it is virtually negligible. When the velocity field is interpolated using linear or quadratic spline interpolation one can thus conclude from these results that the special-purpose RK4 method is a better choice than its regular counterpart. For cubic and quintic spline interpolation, on the other hand, results show no difference in the accuracy of the two methods. In fact, it was found that 4th-order accuracy of the regular RK4 method was obtained for order 4 (cubic) spline interpolation, which is contrary to what theory predicts. 4th-order accuracy was also obtained for the special-purpose method with order 3 (quadratic) spline interpolation. Since the special-purpose method adjusts its steps so that the velocity field is effectively continuous within each step it makes sense that 4th-order accuracy is obtained at a lower order interpolation scheme than for the regular method.

4.3.2 DOPRI5

The variable step Dormand-Prince 5(4) (DOPRI5) method (and other similar methods) has an error-estimation routine implemented in the step size selection algorithm (see section 2.1.3). When a variable step method crosses a cell border it will find that the error estimate is much larger due to the discontinuity. As a result, it will reduce the step size and keep reducing it until the step is small enough to yield an acceptable error estimate.

Many rejected steps means many extra function calls and hence much wasted computational work. As mentioned in section 3.2.2, the hope for the special purpose variant is that the computational work saved by increasing the probability that steps close to cell borders will be accepted, will be enough to make up for the extra computational work associated with the event location procedure.

To begin this section it is once again natural to investigate whether the special-purpose method actually does what it is supposed to do. Figure 4.15 shows an example of how the two methods handle the same border crossings as in figure 4.11, when the velocity field is interpolated using linear spline interpolation. The left plot shows the trajectory computed with the regular DOPRI5 method (labelled DOPRI5 (r) in the figures in this section) and the right plot shows the trajectory computed with the special-purpose method (labelled DOPRI5 (s-p) in the figures). Once again the lines in each plot indicate the cell borders, and the crosses (DOPRI5 (r)) and dots (DOPRI5 (s-p)) mark the position of each (accepted) step. All the results for the DOPRI5 methods in this chapter are created with tolerances $Atol = Rtol = 1 \cdot 10^{-10}$ in the step size control procedure.

Looking at figure 4.15 it is immediately clear that at least in this exact region of the trajectory, the regular DOPRI5 method takes more steps than the special purpose DOPRI5 method before the cell boundaries. A possible (and probable) explanation why the regular method takes more steps before the boundary is that it requires more tries to find an accepted step that crosses the border. If the first attempted step is rejected, the next attempt will be a shorter step. If this is rejected too, the next attempt will be an even shorter step. It might eventually happen that an attempted step is short enough to not cross the border, in which case it is likely to be accepted since it is presumably quite short. Once that happens, the next step will be a little longer, and if it is long enough to cross the border, the whole process will start over again. This pattern will continue until one step is eventually short enough to be accepted even though it steps over the discontinuity at the cell border. In contrast, for the special-purpose method, if a step is found to cross a cell boundary it will be adjusted so that it stops at the boundary, making it more probable that it will be accepted.

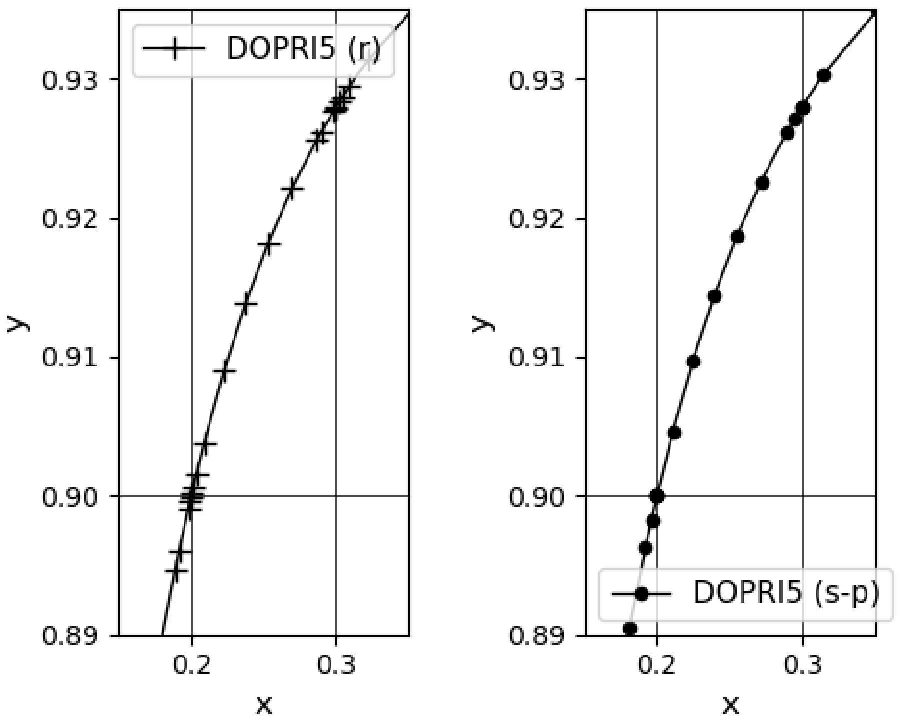


Figure 4.15: Comparison of trajectory computed with regular DOPRI5 method (left) and the special-purpose DOPRI5 method (right), when the velocity field has been interpolated using linear spline interpolation. The horizontal and vertical line represent grid boundaries.

The bottom plot of figure 4.16 supports the suggested explanation above. The plot compares the number of rejected steps as a function of simulation time t for the regular and the special-purpose DOPRI5 method, when using linear spline interpolation. It is clear that the regular DOPRI5 method rejects far more steps than the special-purpose method. The difference is of about one order of magnitude. These two results (the plot of accepted steps in figure 4.15 and the number of rejected steps in figure 4.16) indicate that the special-purpose method does what it is supposed to do.

Figures 4.17, 4.18, and 4.19 show the same as figure 4.16, only for quadratic, cubic, and quintic spline interpolation, respectively. One can see that the total number of rejected steps generally decreases as the interpolation order increases, and that for both cubic and quintic spline interpolation there is hardly any difference between the two methods. In light of the results and discussion in section 4.3.1 this makes sense. If the order of accuracy increases with increasing order of interpolation, one would expect the number of rejected steps to go down, which is exactly what one finds in these results.

Furthermore, by comparing figures 4.16, 4.17, 4.18, and 4.19 one also notices that the gap between the number of rejected steps between the two methods becomes smaller as the interpolation order increases, and that for cubic and quintic spline interpolation there is essentially no difference between the two methods. This indicates that there is no difference in the order of accuracy of DOPRI5 between cubic and quintic spline interpolation. Based on the results in the previous section one might have expected there to be a difference, but according to theory 5^{th}-order accuracy can only be guaranteed with a 7^{th}-order spline interpolation scheme, and we recall that quintic spline interpolation has order 6. Hence, there is no guarantee that there will be a difference in the results obtained with cubic and quintic spline interpolation.

From the discussion so far in this section one may thus conclude that steps taken with the special-purpose DOPRI5 method do have a higher probability of being accepted around grid cell boundaries, as predicted in section 3.2.2, but only for the two lower-order interpolation schemes. For cubic and quintic spline interpolation, there seems to be no difference between the two methods. What remains at this point is to find out if the reduction in the number of rejected steps is enough to reduce the overall computational cost.

We recall from section 3.2.2 that the event location procedure requires a little more computational work in terms of function evaluations. With the special-purpose DOPRI5 method the field f is evaluated 7 times more each time the bisection method is called, followed by an additional 7 times to compute a new step with the adjusted time step h_{adj}. Recall also that there still is a small chance that even the adjusted step can be rejected in the step size selection routine. In comparison, each rejected step only means 7 additional function evaluations. To investigate if the reduced number of rejected steps is enough to make the computations more computationally efficient, consider the top panels in figures 4.16, 4.17,4.18, and 4.19.

The top panels in figures 4.16, 4.17, 4.18, and 4.19 show the number of function evaluations as a function of the simulation time $t = t_N$. The results show that the special-purpose DOPRI5 method generally evaluates the function f more often than the regular DOPRI5 method does, except for when linear spline interpolation is used. However, in each case the difference is of less than one order of magnitude. Since the difference is so small one could thus argue that the extra work is in most cases worth it if it involves an increase in accuracy.

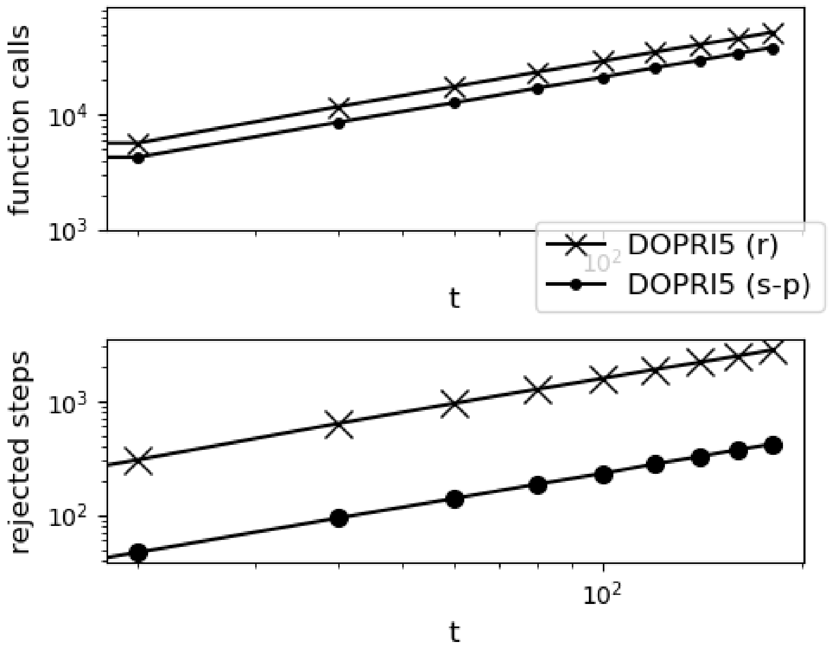


Figure 4.16: Number of function calls (top) and number of rejected steps (bottom) as a function of simulation time t for the regular and the special purpose Dormand-Prince 5(4) method combined with linear spline interpolation.

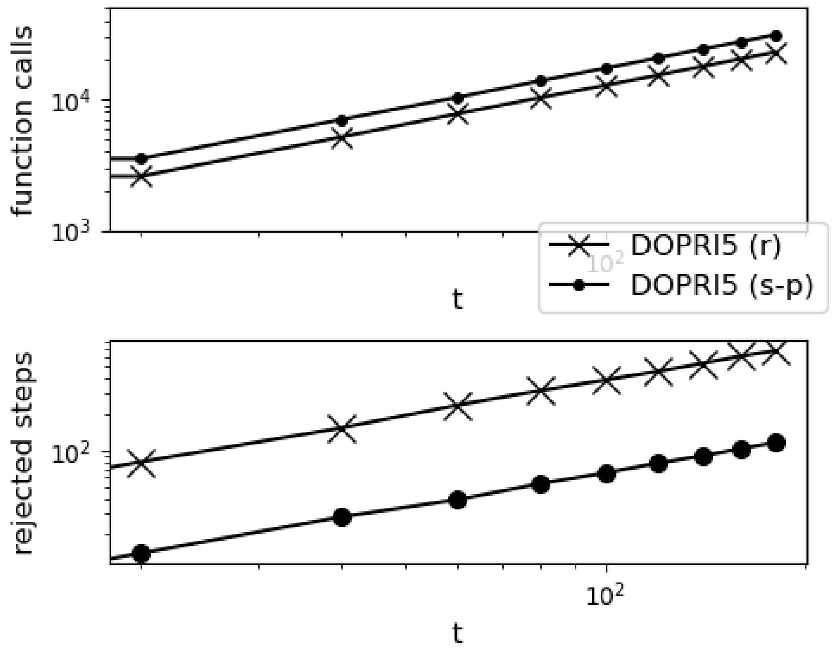


Figure 4.17: Number of function calls (top) and number of rejected steps (bottom) as a function of simulation time t for the regular and the special-purpose DOPRI5 method combined with quadratic spline interpolation.

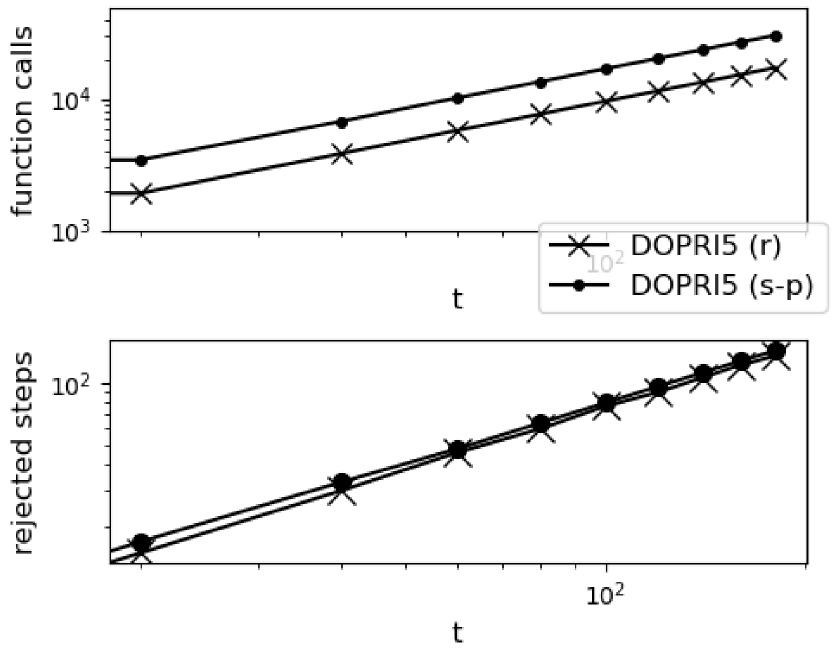


Figure 4.18: Number of function calls (top) and number of rejected steps (bottom) as a function of simulation time t for the regular and the special-purpose DOPRI5 method combined with cubic spline interpolation.

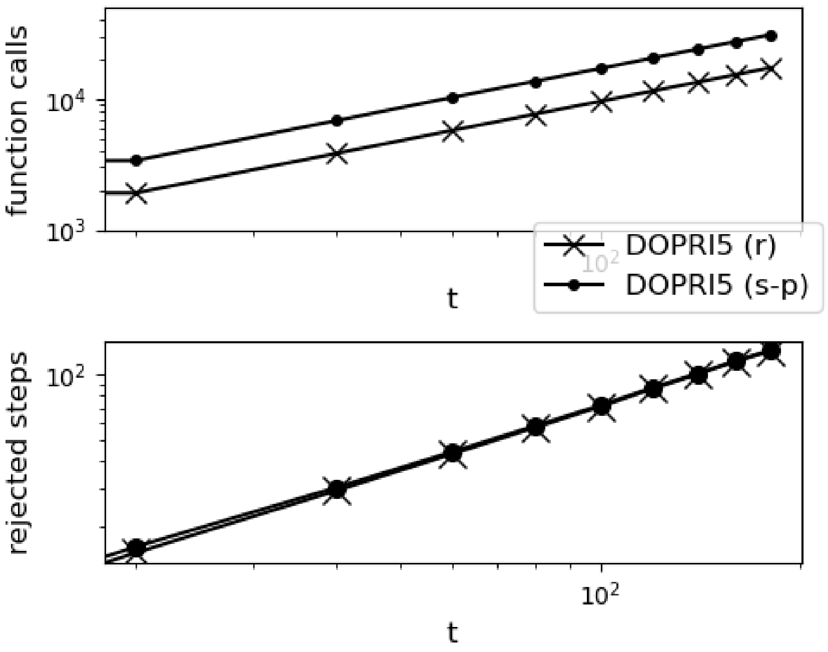
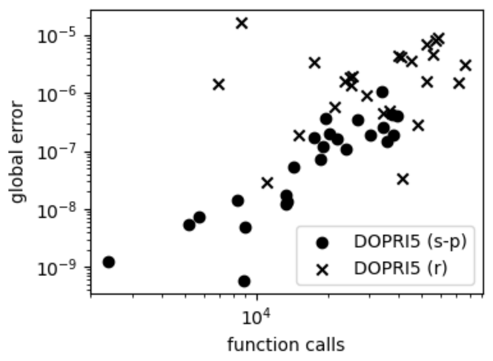
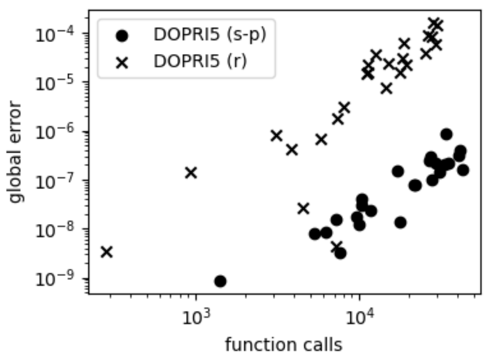


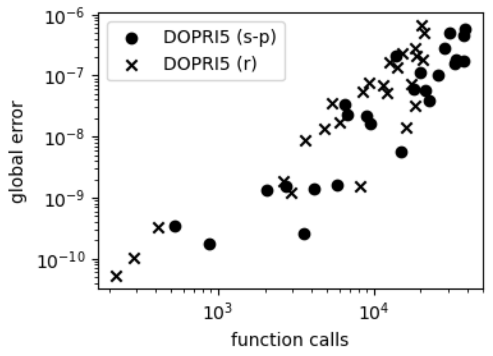
Figure 4.19: Number of function calls (top) and number of rejected steps (bottom) as a function of simulation time t for the regular and the special-purpose DOPRI5 method combined with quintic spline interpolation.



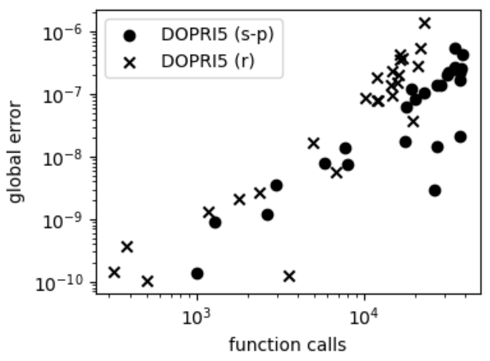
(a) Linear



(b) Quadratic



(c) Cubic



(d) Quintic

Figure 4.20: Global error versus number of function evaluations for regular and special-purpose DOPRI5 combined with different orders of spline interpolation.

Figure 4.20 shows plots of the global error versus the number of function evaluations for all the different interpolation schemes. These plots are created with the same Monte Carlo approach as in figure 4.14. That is, for each combination of integration method and interpolation scheme an ensemble of 25 simulations is run, where each simulation has a random initial position $\boldsymbol{x}_0$ and a random simulation time from $t_0 = 0$ to t_N, where $t_N \in [1, 250]$. Every dot/cross marks the global error and the number of function evaluations for one ensemble member.

With linear spline interpolation (fig. 4.20a) it is clear that the special-purpose DOPRI5 method generally produces more accurate results, and even does so at a lower computational cost. With quadratic spline interpolation it is clear that the computational cost of the special-purpose method is a little higher than that of the regular method, but again the difference in the number of function evaluations is of less than one order of magnitude. In comparison, the difference in global error is of about two orders of magnitude, and in this area it is the special-purpose method that emerges victoriously. Moving on to cubic and quintic spline interpolation (fig. 4.20c and fig. 4.20d, respectively) one can see that the average error of the two methods is approximately the same in both cases. For both interpolation schemes it is also clear that the special-purpose method is a little more computationally costly. Given the previous results in this section and in section 4.3.1 this is not surprising. In in total we find that in light of the results in figure 4.12 and section 4.3.1 in general, all the results in figure 4.20 make sense.

To summarise this section one one could say that once again the special-purpose method performs slightly worse in terms of computational effort than its regular counterpart, with linear spline interpolation being the only exception. In terms of accuracy, on the other hand, the special-purpose method performs better than its regular counterpart when combined with both linear and quadratic spline interpolation. With cubic and quintic spline interpolation, both variants provide the same accuracy, and one would be better off choosing the regular DOPRI5 method as this is the least computationally expensive even if it is by just a small amount.

As a final comment on this section, recall that DOPRI5 was not implemented with FSAL in this work, so the total number of function evaluations could have been somewhat lower than it is in the results presented here. However, this choice does not affect the accuracy of either variant, and neither does it affect the relative difference in computational work between the two variants.

4.3.3 RK4 vs. DOPRI5

This section will discuss which method is the best for each interpolation scheme, and also which combination of method and interpolation scheme seems to be the best overall. For discussions on why the individual methods perform as they do in each case please refer to section 4.3.1 and section 4.3.2 as this will not be discussed here. Note also that the comparisons in this section will be based only on RK4 with step size $h = 0.01$, since that was the value of h that was used to create the plots in figure 4.14 in section 4.3.1, and tolerances $Atol = Rtol = 1 \cdot 10^{-10}$ for DOPRI5. Other values of h will yield different numerical values for the global errors, as figure 4.12 shows. A smaller h will also mean more steps and hence more function evaluations. Other tolerances for DOPRI5 will also affect the results.

Consider linear spline interpolation first. Figure 4.21 shows plots of global error versus number of function evaluations for all four methods combined with linear spline interpolation. The data in these plots are the same as in 4.14a and figure 4.20a in the two preceding sections, but here they are plotted with the same axes for easier comparison. Furthermore, the numerical values for the average global error and number of function calls in each case is provided in table 4.1.

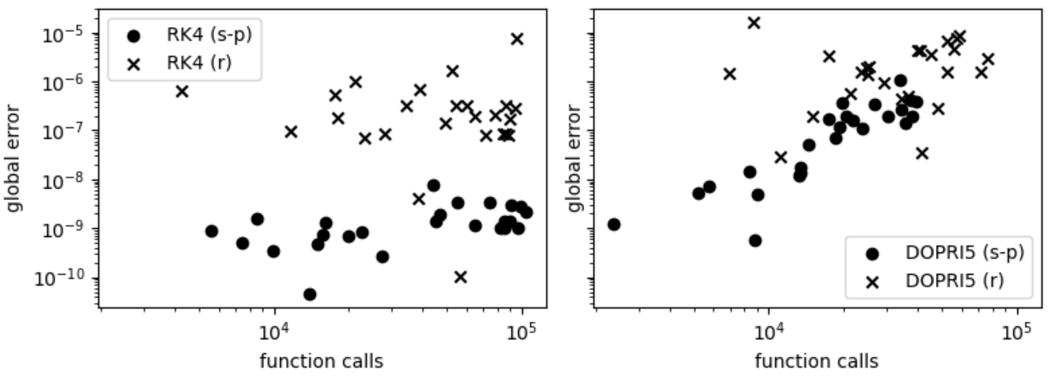


Figure 4.21: Global error versus number of function evaluations for regular and special-purpose methods combined with linear spline interpolation. Left: RK4. Right: DOPRI5.

Table 4.1: Linear spline interpolation: Average values for the global errors and number of function evaluations in figure 4.21.

	RK4 (r)	RK4 (s-p)	DOPRI5 (r)	DOPRI5 (s-p)
Function evaluations	53 926.72	49 011.68	36 810.24	20 434.16
Global error	$6.0205 \cdot 10^{-7}$	$1.6024 \cdot 10^{-9}$	$3.0814 \cdot 10^{-6}$	$1.7281 \cdot 10^{-7}$

It is clear from figure 4.21 and table 4.1 that the method that on average gives the best accuracy is the special-purpose RK4 method, with an average accuracy approximately equal to 10^{-9}. With these parameters the RK4 methods are generally more computationally costly than the DOPRI5 methods, but on the other hand they are also more accurate. Looking at the numerical values in table 4.1 one can see that the special-purpose RK4 method requires approximately 2.5 times more function evaluations than the special-purpose DOPRI5 method, which is the most computationally efficient method under the conditions used here. However, the error for the special-purpose DOPRI5 method is approximately 100 times bigger than for the special-purpose RK4 method, and one can conclude that the increase in computational cost is worth it when the error should be as small as possible. The regular methods are both less accurate than their special-purpose counterparts, and overall, with these parameters the regular DOPRI5 method is the one that performs the worst when combined with linear spline interpolation.

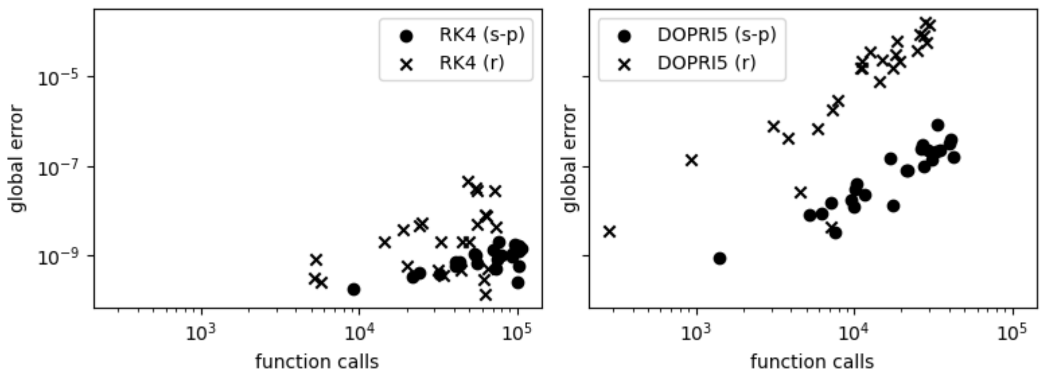


Figure 4.22: Global error versus number of function evaluations for regular and special-purpose methods combined with quadratic spline interpolation. Left: RK4. Right: DOPRI5.

Table 4.2: Quadratic spline interpolation: Average values for the global errors and number of function evaluations in figure 4.22.

	RK4 (r)	RK4 (s-p)	DOPRI5 (r)	DOPRI5 (s-p)
Function evaluations	41 260.80	67 696.32	14 268.00	20 973.16
Global error	$7.5164 \cdot 10^{-9}$	$9.1737 \cdot 10^{-10}$	$3.2543 \cdot 10^{-5}$	$1.4520 \cdot 10^{-7}$

For quadratic spline interpolation, results are presented in figure 4.22 and table 4.2. From figure 4.22 it is immediately clear that the RK4 methods are more accurate than their DOPRI5 counterparts. They are however also the most computationally expensive. With these parameters the special-purpose DOPRI5 obtains approximately the same accuracy as the regular RK4 method, but it requires less than half the number of function evaluations to get there. The regular DOPRI5 performs the worst overall in this simulation ensemble, with an average global error that is 5 orders of magnitude larger than the average global error of the overall best method, which is the special-purpose RK4 method.

Results for cubic spline interpolation are presented in figure 4.23 and table 4.3. Once again, it seems that with these parameters both RK4 methods are superior to the DOPRI5 methods in terms of accuracy, with an average global error that is smaller by several orders of magnitude. From figure 4.23 it is not as easy to say which RK4 method is the better choice, but looking at the numerical values in table 4.3 one can see that the regular method requires slightly fewer function evaluations to obtain approximately the same accuracy. To get the most accurate result, the regular RK4 method would thus be the best choice. However, even from figure 4.23 one can tell that under the conditions applied here the regular DOPRI5 method is noticeably better than all the other methods in terms of the number of function evaluations. If the accuracy provided by this method is sufficient it is then easy to argue that it is the optimal choice.

Table 4.3: Cubic spline interpolation: Average values for the global errors and number of function evaluations in figure 4.23.

	RK4 (r)	RK4 (s-p)	DOPRI5 (r)	DOPRI5 (s-p)
Function evaluations	50 090.40	56 146.24	10 780.32	17 169.36
Global error	$7.6412 \cdot 10^{-10}$	$8.7483 \cdot 10^{-10}$	$1.1350 \cdot 10^{-7}$	$1.2164 \cdot 10^{-7}$

Figure 4.24 and table 4.4 present the results for quintic spline interpolation. Just like in section 4.3.1 and section 4.3.2 the results are very similar to those for cubic spline interpolation. The conclusion for quintic spline interpolation will thus be the same as for cubic spline interpolation, namely that the regular DOPRI5 method is a good choice if one wishes to minimise the computational cost, but that with these parameters both RK4 methods provide a more accurate result. Out of the

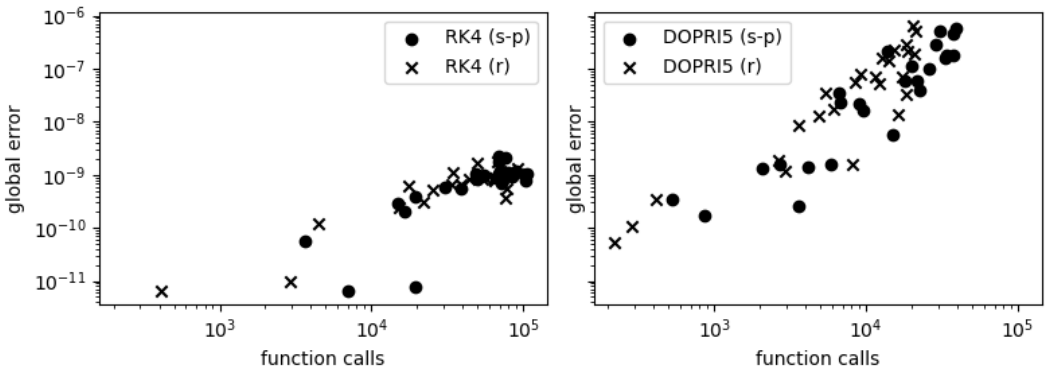


Figure 4.23: Global error versus number of function evaluations for regular and special-purpose methods combined with cubic spline interpolation. Left: RK4. Right: DOPRI5.

two RK4 methods one could argue that the regular method is the better choice, since it requires slightly fewer function evaluations, even though the difference is so small it is practically negligible.

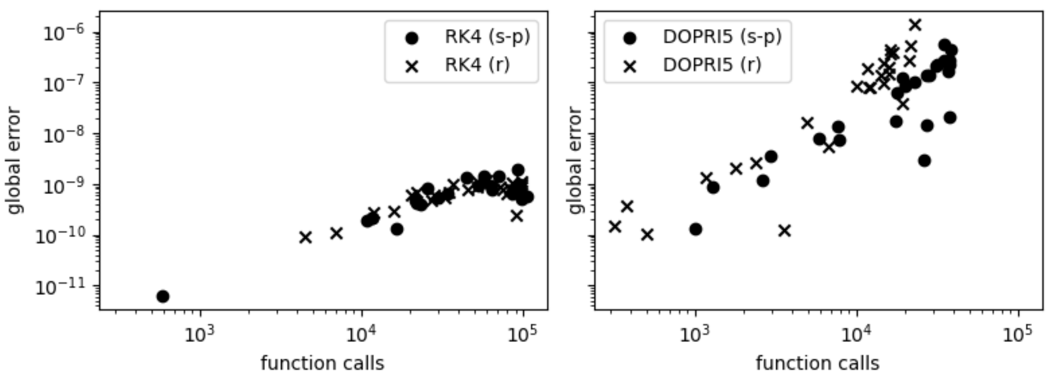


Figure 4.24: Global error versus number of function evaluations for regular and special-purpose methods combined with quintic spline interpolation. Left: RK4. Right: DOPRI5.

Table 4.4: Quintic spline intepolation: Average values for the global errors and number of function evaluations in figure 4.24.

	RK4 (r)	RK4 (s-p)	DOPRI5 (r)	DOPRI5 (s-p)
Function evaluations	51 548.64	52 535.04	11 091.68	22 278.52
Global error	$7.0798 \cdot 10^{-10}$	$7.0950 \cdot 10^{-10}$	$1.8831 \cdot 10^{-7}$	$1.2341 \cdot 10^{-7}$

In total, the best accuracy with the parameters used here is obtained with the two RK4 methods in combination with either cubic or quintic spline interpolation. With these interpolation schemes the error is of the same order of magnitude for both of these methods, so one may argue that the regular RK4 method is the better choice since it requires fewer function evaluations than the special-purpose RK4 method. In terms of computational cost the results presented here indicate that DOPRI5 is usually the best choice. The regular DOPRI5 method is the least computationally costly option when combined with quadratic, cubic, and quintic spline interpolation, and the special-purpose DOPRI5 method is the least costly when combined with linear spline interpolation. Hence, if a somewhat lower accuracy can be accepted then the regular DOPRI5 method could be a good option. Note however that the relative decrease in computational effort is much smaller than the relative decrease in accuracy, meaning that the overall best choice would still be RK4 combined with cubic or quintic spline interpolation.

Chapter 5

Conclusion

The aim of this study has been to construct special-purpose methods for numerical integration to solve of ODEs on the form

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t), \quad \boldsymbol{x}(t_0) = \boldsymbol{x_0}.$$

The solution $\boldsymbol{x}(t)$ of the ODE is a trajectory, and the special-purpose methods make use of an event location procedure to find the exact time at which a specified event occurs on this trajectory. The main idea was to apply the methods to ODEs in which the rate of change $\boldsymbol{f}$ is given by an interpolated velocity field. When the field is interpolated from a set of given data points using n^{th}-order spline interpolation, its $(n-1)^{\text{th}}$ derivative will be discontinuous. Theory states that these discontinuities could affect the order of accuracy of a numerical integration method if its order, p, is higher than $n-2$, where n is still the order of the spline interpolation scheme. With spline interpolation the positions of the discontinuities are known, and the special-purpose methods are constructed to stop and restart integration at these positions. This stopping and restarting makes the interpolated field continuous within each step, and as a result the order of accuracy can be expected to be higher than if the discontinuities are stepped over.

This study was based on the work of Nordam and Duran presented in [24]. Their study was about discontinuities in interpolated velocity fields, but they only considered discontinuities in the temporal dimension, since they are easier to deal with. This work, on the other hand, handles discontinuities only in the spatial dimensions, and can thus be regarded as a supplement to their article.

The special-purpose methods were constructed with two Runge-Kutta methods as a starting point. One is the most common 4^{th}-order method, referred to here as RK4, and the other is the embedded Dormand-Prince pair of order 5(4), referred to here as DOPRI5. RK4 and its special-purpose variant were implemented with a fixed time step h, while DOPRI5 and its special-purpose variant were implemented with a variable time step. The performance of the four methods was compared by applying them to an interpolated form of the two-dimensional double gyre vector field. The field was interpolated using linear spline interpolation, quadratic spline interpolation, cubic spline interpolation, and quintic spline interpolation, which have orders 2, 3, 4, and 6, respectively. Results of the comparison of the performance of the methods were presented in chapter 4. Performance was assessed by examining the accuracy of the final result, measured by global error, and the computational cost, measured by the number of calls to the function $\boldsymbol{f}$ describing the right-hand side of the ODE.

For linear spline interpolation it was found that both special-purpose methods improved accuracy by several orders of magnitude compared to each of their regular counterparts. Figure 4.12a also shows that the special-purpose RK4 method obtained 3^{rd}-order accuracy when combined with linear spline interpolation, while the regular RK4 method only obtained 2^{nd}-order accuracy. The increase in accuracy did come with a small increase in the number of function evaluations for the special-purpose RK4 method, however, the gain in accuracy was big enough in comparison with the increase in computational effort that one can argue that it is worth it. For DOPRI5 with tolerance $Atol = Rtol = 10^{-10}$, the special-purpose method was found to be less costly than its regular counterpart, and in fact it was also the least costly overall when compared to the RK4 methods with fixed $h = 0.01$. Still, the gain in computational efficiency was not enough to make up for the loss in accuracy, so it was concluded that the special-purpose RK4 method would be the best option, at least with the parameters used to create those results.

For quadratic spline interpolation, results found that the special-purpose RK4 method increased the order of accuracy as compared to its regular counterpart. Specifically, the special-purpose RK4 solver was found to be 4$^{\text{th}}$-order accurate when combined with quadratic spline interpolation, while the regular RK4 method was only 3$^{\text{rd}}$-order accurate. For DOPRI5 the special-purpose variant also improved accuracy by two orders of magnitude, and would be the best option of the two variants in spite of a small increase in the computational cost. When comparing DOPRI5 with tolerance $Atol = Rtol = 10^{-10}$ to RK4 with fixed $h = 0.01$ it was concluded that the best option was the special-purpose RK4 method, as it was the most accurate by several orders of magnitude. Again, the special-purpose RK4 method was somewhat more expensive in terms of function evaluations, but not enough to outweigh the gain in accuracy.

Results were found to be almost indistinguishable for cubic and quintic spline interpolation. For RK4 both variants turned out to be 4$^{\text{th}}$-order accurate when combined with both cubic and quintic spline interpolation, but the special-purpose method was again found to be a little more expensive in terms of function evaluations. The difference was however so small that it is practically negligible. For DOPRI5 too the results from both methods were of approximately the same accuracy, but the special-purpose method required a noticeably larger number of function evaluations in both cases. With a fixed step size $h = 0.01$ the RK4 solvers, and in particular the regular variant, were overall found to be the better choice when compared to the variable-step DOPRI5 methods with tolerance $Atol = Rtol = 10^{-10}$, since they were more accurate by several orders of magnitude, and only slightly more computationally costly.

From the results in chapter 4 one may thus conclude that compared to their regular counterparts the special-purpose methods improve performance as defined here when combined with linear or quadratic spline interpolation. With both these interpolation schemes the solver that gave the best performance overall was the special-purpose RK4 method, at least for $h = 0.01$. Quadratic spline interpolation is not commonly used in practice, but the combination of RK4 and linear interpolation is quite common [50, 51, 52]. Hence, these results are potentially quite valuable in transport problems in the ocean or the atmosphere, and in other applications where trajectories are computed from a discrete velocity field.

For cubic and quintic spline interpolation the unaltered methods provided the same accuracy as their corresponding special-purpose variants at a lower computational cost. Again, the RK4 methods with $h = 0.01$ were found to be superior in terms of accuracy, and it was concluded that the best option was the regular RK4 method for both cubic and quintic spline interpolation with the parameters used in these computations. RK4 with cubic/quintic spline interpolation also gave the overall best result, with the lowest global error, although the same global error was also obtained using the special-purpose RK4 method and quadratic spline interpolation.

5.1 Future Work

This study found that using fixed-step RK4 instead of variable-step DOPRI5 is usually a good choice, provided that one chooses the right step size h for RK4. However, since the study only compared the two methods for one step size h and one value for the tolerances in DOPRI5, there is not enough data to draw a general conclusion. For a future study on this topic it would therefore be interesting to investigate this further, and perhaps try to find the ranges in h and $Atol$ and $Rtol$ for which the results will be the same as here, and the ranges for which perhaps DOPRI5 might be a better choice. Such a study should also implement FSAL in DOPRI5. It could also be interesting to include other solvers of different orders in a future study.

Furthermore, it would be interesting to see if it is possible to achieve 4$^{\text{th}}$-order accuracy of RK4 using linear spline interpolation and a special-purpose variant like the one constructed here. In theory, this should be possible if one can manage perfectly to avoid stepping over discontinuities.

Finally, it would also be very interesting to try to implement the event location procedure used here in the Fortran code that was created and used by Nordam and Duran in [24]. This code can be found at [53]. With the event location procedure successfully implemented one could then use the new special-purpose solvers to conduct a follow-up study similar to the work presented here and in [24]. The results of such a study could lead to increased accuracy and/or reduced computational costs, and would therefore be of some interest to practitioners in applied oceanography.

Appendix

A Root-Finding Methods

An iterative root-finding method in computational mathematics is used to find the value $x^* = r$ that makes $f(r) = 0$ for some function $f(x)$. This appendix will present two such methods, where the first one is the one used in this work.

The Bisection Method

The idea of the bisection method is to bracket the root r of the function $f(x)$ by enclosing it in an interval $[a, b]$ where the length $|b - a|$ of the interval goes to zero.

With `a` and `b` being the endpoints of an interval containing the root, the algorithm for the bisection method for some function `f(x)` can be written as

```
while (b-a)/2 > tol:
    c = (a+b)/2
    if f(c) == 0:
        r = c
    if f(a)*f(c) < 0:
        b = c
    else:
        a = c
r = c
```

where then `r` is the approximation to the root r to within a given tolerance `tol`.

The challenge with the bisection method is that if there is more than one root in the initial interval, it might fail. If there is only one root it will always converge, as opposed to Newton's method. In the cases considered here there is only one root x^* between the points x_n and x_{n+1} and the bisection method works just fine.

Newton's method

The principle of Newton's method is illustrated in figure A.1. Starting with a first guess x_0 for the root r, we find the tangent line at the function evaluated at x_0, follow it until it intersects with the x-axis, and use the value of x at the intersection as the next approximation x_1. The equation for the tangent line at a point x_0 is given by

$$y(x) = f(x_0) + f'(x_0)(x - x_0).$$

The intersection with the x-axis happens at $y = 0$, so letting $y(x_1) = 0$ and solving for x_1 we find the expression

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

for the next approximation x_1. The procedure is then repeated until convergence is reached. For $n+1$ steps the procedure can be written as follows.

$$\begin{aligned} x_0 &= \text{initial guess} \\ x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} \\ &\vdots \\ x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \end{aligned}$$

Convergence can be defined as when the difference between to subsequent approximations y_i,y_{i+1} is smaller than some set tolerance *tol*, or, mathematically, $|y_{i+1} - y_i| \leq tol$.

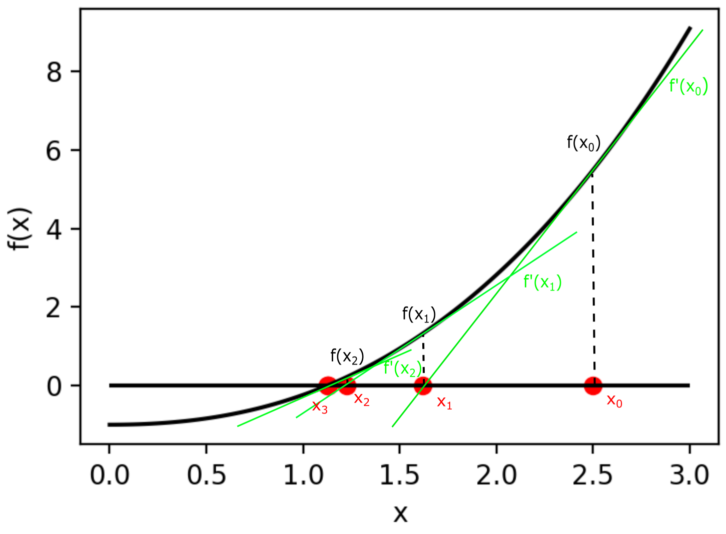


Figure A.1: Graphical illustration of Newton's method

The challenge with Newton's method is that it does not converge for all initial guesses x_0, but as long as x_0 is close enough to the root it will usually converge.

Bibliography

[1] J. Beyer et al. 'Environmental effects of the Deepwater Horizon oil spill: A review'. In: *Marine Pollution Bulletin* 110 (2016), pp. 28–51. DOI: https://doi.org/10.1016/j.marpolbul.2016.06.027.

[2] T. J. Crone and M. Tolstoy. 'Magnitude of the 2010 Gulf of Mexico Oil Leak'. In: *Science* 330 (2010), pp. 634–634. DOI: 10.1126/science.1195840.

[3] R. Li et al. 'The distribution, characteristics and ecological risks of microplastics in the mangroves of Southern China'. In: *Science of The Total Environment* 708 (2020), p. 135025. DOI: https://doi.org/10.1016/j.scitotenv.2019.135025.

[4] J. R. Jambeck et al. 'Plastic waste inputs from land into the ocean'. In: *Science* 347.6223 (2015), pp. 768–771. DOI: 10.1126/science.1260352.

[5] L. C. M. Lebreton et al. 'River plastic emissions to the world's oceans'. In: *Nature Communications* 8 (2017). DOI: https://doi.org/10.1038/ncomms15611.

[6] J. G. B. Derraik. 'The pollution of the marine environment by plastic debris: a review'. In: *Marine Pollution Bulletin* 44 (2002), pp. 842–852. DOI: https://doi.org/10.1016/S0025-326X(02)00220-5.

[7] B. D. Hardesty et al. 'Using Numerical Model Simulations to Improve the Understanding of Micro-plastic Distribution and Pathways in the Marine Environment'. In: *Frontiers in Marine Science* 4 (2017), p. 30. DOI: 10.3389/fmars.2017.00030.

[8] M. Wilson et al. 'Predicting the Movement of Oil'. In: *Oil Spill Science: Sea Grant Programs of the Gulf of Mexico* (2017). (GOMSG-G-17-004).

[9] E. van Sebille et al. 'Lagrangian ocean analysis: Fundamentals and practices'. In: *Ocean Modelling* 121 (2018), pp. 49–75. DOI: https://doi.org/10.1016/j.ocemod.2017.11.008.

[10] A. Bennett. *Lagrangian Fluid Dynamics*. Cambridge University Press, 2006. DOI: https://doi.org/10.1017/CBO9780511734939.

[11] E. North et al. 'Simulating Oil Droplet Dispersal From the Deepwater Horizon Spill With a Lagrangian Approach'. In: *Washington DC American Geophysical Union Geophysical Monograph Series* 195 (2011), pp. 217–226. DOI: 10.1029/2011GM001102.

[12] V. Onink et al. 'Global simulations of marine plastic transport show plastic trapping in coastal zones'. In: *Environmental Research Letters* 16 (2021). DOI: https://doi.org/10.1088/1748-9326/abecbd.

[13] V. Onink et al. 'The Role of Ekman Currents, Geostrophy, and Stokes Drift in the Accumulation of Floating Microplastic'. In: *Journal of Geophysical Research: Oceans* 127 (2019), pp. 1474–1490. DOI: https://doi.org/10.1029/2018JC014547.

[14] M. L. A. Kaandorp, H. A. Dijkstra and E. van Sebille. 'Closing the Mediterranean Marine Floating Plastic Mass Budget: Inverse Modeling of Sources and Sinks'. In: *Environmental Science and Technology* 54 (2020), pp. 11980–11989. DOI: https://doi.org/10.1021/acs.est.0c01984.

[15] H. Rye, M. Reed and N. Ekrol. 'The PARTRACK model for calculation of the spreading and deposition of drilling mud, chemicals and drill cuttings'. In: *Environmental Modelling & Software* 13 (1998), pp. 431–441. DOI: https://doi.org/10.1016/S1364-8152(98)00048-6.

[16] P. P. Povinec et al. 'Dispersion of Fukushima radionuclides in the global atmosphere and the ocean'. In: *Applied Radiation and Isotopes* 81 (2013), pp. 383–392. DOI: https://doi.org/10.1016/j.apradiso.2013.03.058.

[17] R. Marsh et al. 'NEMO–ICB (v1.0): interactive icebergs in the NEMO ocean model globally configured at eddy-permitting resolution'. In: *Geoscientific Model Development* 8 (2015), pp. 1547–1562. DOI: 10.5194/gmd-8-1547-2015.

[18] M. N. Dawson, A. S. Gupta and M. H. England. 'Coupled biophysical global ocean model and molecular genetic analyses identify multiple introductions of cryptogenic species'. In: *Proceedings of the National Academy of Sciences* 102 (2005), pp. 11968–11973. DOI: 10.1073/pnas.0503811102.

[19] H. D. Young and R. A. Freedman. *University Physics with modern physics (volume one).* 13th Edition. Pearson Education Limited, 2011.

[20] S. H. Strogatz. *Nonlinear Dynamics and Chaos.* Second edition. CRC Press, 2018.

[21] The Norwegian Meteorological Institute. *Download services.* URL: https://www.met.no/en/free-meteorological-data/Download-services (visited on 14/06/2021).

[22] E. Hairer, S. P. Nørsett and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems.* Second revised edition. Springer Series in Computational Mathematics. Springer-Verlag Berlin Heidelberg, 2008.

[23] W. H. Enright et al. 'Effective solution of discontinuous IVPs using a Runge-Kutta formula pair with interpolants'. In: *Applied Mathematics and Computation* 27 (1988), pp. 313–335. DOI: https://doi.org/10.1016/0096-3003(88)90030-6.

[24] T. Nordam and R. Duran. 'Numerical Integrators for Lagrangian oceanography'. In: *Geoscientific Model Development* 13 (2020), pp. 5935–5957.

[25] M. Henon. 'On the numerical computation of Poincaré maps'. In: *Physica D: Nonlinear Phenomena* 5 (1982), pp. 412–414. DOI: https://doi.org/10.1016/0167-2789(82)90034-3.

[26] L. F. Shampine, I. Gladwell and S. Thompson. *Solving ODEs with MATLAB.* Cambridge University Press, 2003.

[27] T. Sauer. *Numerical Analysis.* Second Edition. Pearson Education Limited, 2014.

[28] Y. Liu and R. H. Weisberg. 'Evaluation of trajectory modeling in different dynamic regions using normalized cumulative Lagrangian separation'. In: *Journal of Geophysical Research: Oceans* 116 (2011). DOI: https://doi.org/10.1029/2010JC006837.

[29] K. P. Edwards, F. E. Werner and B. O. Blanton. 'Comparison of Observed and Modeled Drifter Trajectories in Coastal Regions: An Improvement through Adjustments for Observed Drifter Slip and Errors in Wind Fields'. In: *Journal of Atmospheric and Oceanic Technology* 23 (2006), pp. 1614–1620. DOI: 10.1175/JTECH1933.1.

[30] T. M. Özgökmen et al. 'On multi-scale dispersion under the influence of surface mixed layer instabilities and deep flows'. In: *Ocean Modelling* 56 (2012), pp. 16–30. DOI: https://doi.org/10.1016/j.ocemod.2012.07.004.

[31] C. Runge. 'Ueber die numerische Auflösung von Differentialgleichungen'. In: *Math. Ann.* 46 (1895), pp. 167–178.

[32] K. Heun. 'Neue Methode zur approximativen Integration der Differentialgleichungen einer unabhängigen Veränderlichen'. In: *Zeitschr. für Math. u. Phys.* 45 (1900), pp. 23–38.

[33] W. Kutta. 'Beitrag zur näherungsweisen Integration totaler Differentialgleichungen'. In: *Zeitschr. für Math. u. Phys.* 46 (1901), pp. 286–288.

[34] J. R. Dormand and P. J. Prince. 'A family of embedded Runge-Kutta formulae'. In: *Journal of Computational and Applied Mathematics* 6 (1980), pp. 19–26.

[35] *scipy.integrate.RK45.* URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.RK45.html (visited on 23/06/2021).

[36] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations.* Wiley, 1962.

[37] C. de Boor. *A Practical Guide to Splines.* Springer-Verlag, 1978.

[38] W. H. Press et al. *Numerical Recipes.* Second Edition. Cambridge University Press, 1992.

[39] R. Kress. *Numerical Analysis.* Graduate Texts in Mathematics. Springer-Verlag, 1998.

[40] M. K. Horn. 'Fourth and fifth-order scaled Runge-Kutta algorithms for treating dense output'. In: *SIAM J.Numer. Anal.* 20 (1983), pp. 558–568.

[41] L. F. Shampine and L. O. Jay. 'Dense Output'. In: *Encyclopedia of Applied and Computational Mathematics.* Ed. by Björn Engquist. Springer Berlin Heidelberg, 2015, pp. 339–345. DOI: 10.1007/978-3-540-70529-1_107.

[42] L: F. Shampine. 'Some Practical Runge-Kutta Formulas'. In: *Mathematics of Computation* 46 (1986), pp. 135–150.

[43] E. N. Lorenz. 'Deterministic Nonperiodic Flow'. In: *Journal of the Atmospheric Sciences* 20 (1963), pp. 387–411.

[44] P. Grassberger and I. Procaccia. 'Measuring the strangeness of strange attractors'. In: *Physica D: Nonlinear Phenomena* 9 (1983), pp. 189–208. DOI: https://doi.org/10.1016/0167-2789(83)90298-1.

[45] D. Viswanath. 'Symbolic dynamics and periodic orbits of the Lorenz attractor'. In: *Nonlinearity* 16 (2003). DOI: https://doi.org/10.1088/0951-7715/16/3/314.

[46] D. Viswanath. 'The fractal property of the Lorenz attractor'. In: *Physica D: Nonlinear Phenomena* 190 (2004), pp. 115–128. DOI: https://doi.org/10.1016/j.physd.2003.10.006.

[47] D. Viswanath and S. Şahutoğlu. 'Complex Singularities and the Lorenz Attractor'. In: *SIAM Review* 52 (2010).

[48] T. Nordam et al. 'Numerical Integration and Interpolation in Marine Pollutant Transport Modelling'. In: *Proceedings of the Forty-first AMOP Technical Seminar, Environment and Climate Change Canada Ottawa, ON, Canada* (2017). https://hdl.handle.net/11250/2653393, pp. 586–609.

[49] *scipy.interpolate.RectBivariateSpline.* URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RectBivariateSpline.html (visited on 09/06/2021).

[50] F. J. Beron-Vera, M. J. Olascoaga and G. J. Goni. 'Oceanic mesoscale eddies as revealed by Lagrangian coherent structures'. In: *Geophysical Research Letters* 35 (2008). DOI: https://doi.org/10.1029/2008GL033957.

[51] A. Berry, T. Dabrowski and K. Lyons. 'The oil spill model OILTRANS and its application to the Celtic Sea'. In: *Marine Pollution Bulletin* 64 (2012), pp. 2489–2501. DOI: https://doi.org/10.1016/j.marpolbul.2012.07.036.

[52] N.A. Patankar and D.D. Joseph. 'Lagrangian numerical simulation of particulate flows'. In: *International Journal of Multiphase Flow* 27 (2001), pp. 1685–1706. DOI: https://doi.org/10.1016/S0301-9322(01)00025-8.

[53] T. Nordam. *nordam/ODE-integrators-for-Lagrangian-particles 0.9.* Version 0.9. Sept. 2020. DOI: 10.5281/zenodo.4041979.