Abstract

A generic flow system can be described as a system whose state depends on flowing streams of energy, material, or information. Traditional examples of such systems are the transport of properties such as pressure, temperature, and matter in fluids, and the transport of charge in electrical currents. However, valuable insight into various phenomena, including — but not limited to — algal blooms in the ocean, and crowd patterns formed by humans, can be obtained by regarding them as flow systems.

Lagrangian coherent structures (henceforth abbreviated to LCSs) can be described as "landscapes" within a multidimensional phase space, which exert a major influence upon the flow patterns in dynamical systems. In terms of predicting future states of complex systems accurately, LCSs provide a simpler means of analysis than the conventional approach of progressively increasing the resoluton of the model(s) involved in numerical simulations. In this context, a complex system is a system which exhibits sensitive dependence on initial conditions. From their variational theory, hyperbolic LCSs are identified as locally most repelling or attracting material surfaces. Somewhat simplified, such surfaces can be considered as generalized trajectories, created by the underlying transport phenomenon. When investigating transport systems, hyperbolic LCSs are of particular interest, as they form the skeleton of observable flow patterns.

Several important, naturally occuring transport phenomena — such as the spread of contaminants by oceanic surface currents — can reasonably be approximated as planar. This is one possible reason why previous LCS research has mostly been conducted for two-dimensional systems; another may be that the inclusion of a third dimension severely increases the underlying mathematical complexity. To the extent that LCS analysis for three-dimensional flows has hitherto been conducted, a common approach has been to merge a set of two-dimensional projections, resulting in quasi-three-dimensional surfaces. This project is centered around combining the underlying variational principles of hyperbolic LCSs with an acknowledged technique for computing invariant manifolds of vector fields, with a view to developing a robust framework for computing hyperbolic LCSs in three-dimensional flow systems.

Our approach is described in detail, from its theoretical fundament, through how we incorporated the underlying LCS theory into the computation of manifolds and subsequently LCS surfaces, to some clever optimizations regarding computational resource management. Using a few reference cases, we demonstrate the efficacy of our method in reproducing simple three-dimensional manifolds and LCSs. Moreover, the LCSs obtained for a simple, yet chaotic flow system are shown to correspond well with those obtained for a time perturbed version of the same system. This suggests that the computed LCS surfaces are quite robust, and that the unavoidable inaccuracies of model data for simulating transport phenomena need not be a hindrance for applying LCS analysis to real-world systems.

It remains to be seen whether computing "truly" three-dimensional LCS surfaces yields a sufficient increase in descriptive power, compared to the aforementioned quasi-three-dimensional approach, to warrant the increase in conceptual complexity and computational resource consumption. This balancing act might be context sensitive.

Sammendrag

Et generisk strømningssystem kan beskrives som et system hvis tilstand avhenger av flyt av energi-, material-, eller informasjonsstrømmer. Tradisjonelle eksempler på slike systemer er transport av trykk, temperatur, og materie i fluidstrømninger, og ladningstransport forårsaket av elektriske strømmer. Interessante sider ved andre fenomen, som algeoppblomstring, og mønsterdannelser i folkemengder, kan avsløres ved å betrakte dem som strømningssystemer.

Lagrange-koherente strukturer (heretter forkortet til LKSer) kan beskrives som «landskap» i et flerdimensjonalt rom, som utøver stor innflytelse på strømningsmønstre i dynamiske systemer. Hva gjelder prediksjoner for fremtidige tilstander i komplekse systemer, kan bruk av LKSer utgjøre et enklere analyseverktøy enn den mer tradisjonelle tilnærmingen basert på å iterativt øke oppløsningen i modellene som ligger til grunn for numeriske simuleringer. I denne sammenheng betegner et komplekst system et system som er svært sensitivt til dets initialbetingelser. Fra deres underliggende variasjonsteori kjennetegnes hyperbolske LKSer som lokalt sterkest frastøtende eller tiltrekkende materialoverflater. Noe forenklet kan denne typen overflater betraktes som en generalisering av banene det underliggende transportfenomenet forårsaker. Ved analyse av transportsystemer er hyperbolske LKSer spesielt interessante, ettersom disse utgjør skjelettet av observable strømningsmønstre.

Flere viktige, naturlige transportfenomen – deriblant spredning av forurensning ved havoverflatestrømninger – kan betraktes som tilnærmet plane. Dette er en mulig årsak til at tidligere utført forskning på LKSer i stor grad har omhandlet todimensjonale systemer; en annen kan være at å introdusere en tredje dimensjon øker den underliggende matematiske kompleksiteten betraktelig. I den grad LKSer i tredimensjonale strømninger har blitt undersøkt så langt, har en vanlig tilnærming vært å sette sammen et sett av planprojeksjoner, hvilket resulterer i kvasi-tredimensjonale overflater. Dette prosjektet går ut på å kombinere de underliggende variasjonsprinsippene for hyperbolske LKSer med en anerkjent teknikk for å beregne invariante mangfoldigheter for vektorfelt, med den hensikt å utvikle et robust rammeverk for å beregne hyperbolske LKSer i tredimensjonale strømningssystemer.

Fremgangsmåten vår beskrives i detalj, fra dens teoretiske fundament, via hvordan vi inkorporerte den underliggende LKS-teorien i beregninger av mangfoldigheter og videre LKS-overflater, til noen fikse løsninger for å begrense ressursforbruket. Ved analyse av noen referansesystemer demonstrerer vi metodens velegnethet for beregning av enkle tredimensjonale mangfoldigheter og LKSer. Videre stemmer LKS-overflatene vi finner i et enkelt, men kaotisk strømningssystem godt overens med overflatene som avdekkes i en tidsperturbert versjon av det samme systemet. Dette tyder på at de beregnede LKS-overflatene er robuste, og at den uunngåelige unøyaktigheten i modelldata for å simulere reelle transportfenomen ikke trenger å stå til hinder for å anvende LKS-analyse på virkelige strømninger.

Det gjenstår å se hvorvidt beregning av «ekte» tredimensjonale LKS-overflater resulterer i tilstrekkelig økt innsikt, sammenlignet med den ovennevnte kvasi-tredimensjonale tilnærmingen, til å forsvare den økte kompleksiteten og det økte ressursforbruket. Denne avveiningen vil muligens avhenge av kontekst.

Preface

The submission of this thesis signifies my completion of all formal requirements for acquiring the degree of MSc within the field of "Physics and Mathematics", with a specialization in Applied Physics, at the Norwegian University of Science and Technology. This thesis accounts for a total of 30 ECTS. All of the underlying work was performed in Trondheim under the supervision of Assoc. Prof. Tor Nordam, during the spring semester of 2018 — that is, my 10^{th} and (for now) final semester as a university student. As a consequence of our close collaboration for the work on our master's projects, fellow student Simon Nordgreen and I present the same results in our respective master's theses.

This thesis builds upon the work I did for my specialization project, which was carried out during the fall semester of 2017 (Løken 2017). The target audience possesses an understanding of physics and mathematics corresponding to what is expected prior to the enrollment in a master's level physics program. Intermediate proficiency with regards to numerical programming pertaining to simulations would be advantageous. The program code which was developed for this project — a combination of modern Fortran, C++ and Python — is hosted at `github`[1]; guidance can be provided upon request. Familiarity with variational principles — in particular regarding stretch and strain in material flows — is not a necessary prerequisite, as all advanced concepts are explained in detail prior to application.

Trondheim, June 2018
Arne Magnus Tveita Løken

[1] https://github.com/arnemagnus/3d_lcs

Acknowledgements

I would like to extend my gratitude to my supervisor, Assoc. Prof. Tor Nordam. Our many discussions over the past academic year has provided me with great insight, while his great knowledge and humour proved excellent tools in terms of relaxing my early-semester nerves. Coupled with his expertise concerning the use of supercomputers for scientific purposes — without the use of which, none of the computations constituting the results presented in this thesis would have been practically feasible — I could not be more satisfied.

My good friend and fellow student Simon Nordgreen, who, much like me, thrived under the guidance of Assoc. Prof. Nordam, definitely deserves a mention. Like for our specialization projects which took place last semester, we have collaborated closely for our respective master's theses work. Our many thorough discussions, and sessions of intensive work, have been invaluable to further enhance my understanding of the underlying variational principles which form the basis of our investigations. Looking back at what we have achieved over the past year, I think it is safe to say that two heads are better than one.

Last, but most certainly not least, I gratefully thank my family for their unconditional support and encouragement throughout.

A. M. T. L.

Table of Contents

List of Figures

List of Tables

Notation

Newton's notation is used for differentiation with respect to time, i.e.:

$$\dot{f}(t) \equiv \frac{\mathrm{d}f(t)}{\mathrm{d}t}.$$

Vectors are denoted by lowercase, upright, bold letters, like this:

$$\boldsymbol{\xi} = \left(\xi_1, \xi_2, \ldots, \xi_n\right).$$

The Euclidean inner product for two vectors $(\boldsymbol{\xi}, \boldsymbol{\psi}) \in \mathbb{R}^n$ is denoted by:

$$\langle \boldsymbol{\xi}, \boldsymbol{\psi} \rangle = \sum_{i=1}^{n} \xi_i \psi_i.$$

The Euclidean norm of a vector $\boldsymbol{\xi} \in \mathbb{R}^n$ is designated as:

$$\|\boldsymbol{\xi}\| = \sqrt{\langle \boldsymbol{\xi}, \boldsymbol{\xi} \rangle}.$$

Matrices and matrix representations of rank-2 tensors are denoted by uppercase, upright, bold letters, as follows:

$$\mathbf{A} = (a_{i,j}) = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}.$$

1 Introduction

When analyzing complex dynamical systems, such as the nonlinear many-body problems arising in transport phenomena by virtue of oceanic currents or atmospheric winds, the conventional approach to predicting future states by means of simulating the trajectories of phase points is frequently insufficient. This is due to the resulting predictions being very sensitive to small changes in time and initial positions. One way of addressing the delicate dependence on initial conditions is to run ensembles of different models for the same underlying physical systems, with increasing spatial and temporal resolution. Another is to run ensembles of simulations for the same physical model, using perturbed initial conditions. These sorts of approaches are made possible because the fundamental dynamics are known — yet, for complex transport systems, the computational cost quickly grows beyond the available resources, in terms of computation time or memory. For many practical purposes, however, microscopic details matter little in comparison to the overarching trends in the system, which means that a less ambitious approach, merely aiming to understand the macroscopics of the transport phenomenon, is often justifiable.

At the turn of the millennium, the concept of Lagrangian coherent structures saw the light of day, emerging from the intersection between nonlinear dynamics, that is, the underlying mathematical principles of chaos theory, and fluid dynamics (Haller and Yuan 2000). These provide a new framework for understanding transport phenomena in conceptual fluid flow systems. Lagrangian coherent structures can be described as time-evolving "landscapes" in a multidimensional space, which dictate macroscopic flow patterns in dynamical systems. In particular, such structures define the interfaces of dynamically distinct, invariant regions. An invariant region in fluid dynamics is characterized as a domain where all particle trajectories that originate within the region, remain in it, although the region itself can move and deform with time. So, simply put; Lagrangian coherent structures enable us to make predictions regarding the future states of flow systems.

There are two possible perspectives regarding the description of fluid flow. The Eulerian approach is to consider the properties of a flow field at a set of fixed points in time and space. An example is the concept of velocity fields, which describe the local and instantaneous velocities at all points within their domains. The Lagrangian point of view, on the other hand, concerns the developing velocity of each fluid element along their paths, as they are transported by the flow. Unlike the Eulerian perspective, the Lagrangian mindset is objective, as in frame-invariant. That is, properties of Lagrangian fields are unchanged by time-dependent translations and rotations of the reference frame. For unsteady flow systems, which are more common than steady flow systems in nature, there exists no self-evident preferred frame of reference. Thus, any transport-dictating dynamical structures should hold for *any* choice of reference frame. This is the main rationale for which *Lagrangian*, rather than *Eulerian*, coherent structures have been pursued.

A generic flow system can be described as a structure whose state depends on flowing streams of energy, material, or information. Conventional examples of flow systems include

the transport of physical properties such as pressure, temperature, or matter in fluids, and the transport of charge in electrical currents. A large variety of phenomena can reasonably be modelled as flow systems, such as the classical harmonic oscillator, or the interaction between predator and prey in closed systems (Strogatz 2014, parts I–II). In doing so, valuable pieces of insight can be obtained from well-understood properties of generic flows. In recent years, analyses based upon Lagrangian coherent structures have been conducted for a variety of naturally occuring phenomena which are not commonly considered as flow systems. Two prominent examples are how Olascoaga et al. (2008) used Lagrangian coherent structures in order to forecast the development of toxic algae in the ocean, and Ali and Shah (2007) used Lagrangian coherent structures to predict the formation and stability of human crowd patterns. As these examples emphasize, the theory of Lagrangian coherent structures is applicable to a wide range of systems.

The focus of this project has been the computation of Lagrangian coherent structures in three-dimensional flow systems. Although the framework for detecting Lagrangian coherent structures is mathematically valid for any number of dimensions, little work has previously been dedicated to three-dimensional systems. The two-dimensional case has been treated extensively (consider, for instance, the works of Haller and Yuan (2000), Farazmand and Haller (2012a) and Onu, Huhn, and Haller (2015)), as many real-world transport systems — such as the transport of garbage patches, or remnants of oil spills, by oceanic surface currents — can reasonably be treated as two-dimensional. Two-dimensional analysis is, however, not always suffucient; the scattering of e.g. volcanic ashes by the Earth's wind systems is an example of an irrefutably three-dimensional transport phenomenon. As concrete areas of application, identifying the Lagrangian coherent structures present in the underlying circulation allows for predicting the spread of oil spills on the ocean surface in order to isolate them and accelerate the cleanup process before the oil is able to reach the coastline, or providing airline companies an opportunity to divert flights which would otherwise be exposed to volcanic ash clouds. Put simply, Lagrangian flow analysis could potentially mitigate human-instigated and natural calamities alike.

The limited extent to which analysis of three-dimensional flow systems with the intention of identifying Lagrangian coherent structures has previously been conducted, is emphasized by the common practice being to compute a series of two-dimensional projections followed by joining them together using some kind of interpolation method (see e.g. the work of Blazevski and Haller (2014)). Although yielding three-dimensional surfaces, this approach potentially neglects intricacies pertaining to fully three-dimensional transport phenomena. A plausible justification for using this approach is the increase in complexity when including the depth dimension, further increasing the amount of computational resources required for full-scale analysis. Moreover — as previously mentioned — many significant transport systems can reasonably be treated as two-dimensional. Hence, the extent to which it is appropriate to substitute analysis of two-dimensional Lagrangian coherent structures with that of their three-dimensional counterparts remains an open question.

For this project, we aimed to adapt the method of geodesic level sets for computing manifolds

of three-dimensional vector fields, as outlined by Krauskopf, Osinga, et al. (2005) (first introduced by Krauskopf and Osinga (2003)), to identify Lagrangian coherent structures as barriers to transport in three-dimensional flow systems. This involved extensive conceptual derivations, in addition to quite a bit of programming ingenuity. Accordingly, the project is best classified as one of *computational physics*, which might be viewed as complementary to the more traditional branches of *theoretical* and *experimental* physics.

This thesis is structured based on the idea that readers possessing at least an undergraduate level of knowledge of physics and mathematics, in addition to a rudimentary understanding of programming, should be able to understand and repeat the numerical investigations which have been conducted. To this end, the immediately forthcoming chapter contains a description of the numerical integration and interpolation schemes which we used to simulate general transport phenomena, in addition to a generic yet brief mathematical description of the kind of flow systems considered — and the Lagrangian coherent structures situated therein, based on their variational theory. In the ensuing chapter, we present the different transport models we considered, followed by a rigorous description of how we implemented the variational principles of Lagrangian coherent structures in a variation of the method of geodesic level sets, in addition to how we made sure to utilize the available computational resources in an efficient manner throughout. Then, we present a few tests we used in order to verify that the computed three-dimensional structures behave as expected for Lagrangian coherent structures, closely followed by a presentation of the Lagrangian coherent structures we identified in the various transport models. Lastly, we discuss the strengths and weaknesses of using our variation of the method of geodesic level sets to compute three-dimensional Lagrangian coherent structures, before drawing conclusions on the project as a whole.

2 Theory

This chapter introduces the mathematical principles which form the foundation of our approach to computing Lagrangian coherent structures in three-dimensional flow systems. Section 2.1 provides a concise description of how numerical integrators (that is, Runge-Kutta solvers) in addition to interpolation methods allow for numerically solving systems of ordinary differential equations. Then, section 2.2 introduces what we define as *flow systems*, in addition to providing a concise description of the underlying mathematics upon which Lagrangian flow analysis is based. Lastly, section 2.3 contains a succinct introduction to Lagrangian coherent structures; perhaps most notably, necessary and sufficent existence criteria for *hyperbolic* three-dimensional Lagrangian coherent structures — which arise from their underlying variational theory — are presented.

2.1 Solving systems of ordinary differential equations

In physics, like other sciences, modeling a system often equates to solving an initial value problem. An initial value problem can be described in terms of an ordinary differential equation (hereafter abbreviated to ODE) of the form

$$\dot{x}(t) = f\big(t, x(t)\big), \quad x(t_0) = x_0, \tag{2.1}$$

where x is an unknown function (scalar or vector) of time t. The function f is defined on an open subset Ω of $\mathbb{R} \times \mathbb{R}^n$, where n is the number of spatial dimensions; that is, the number of components of x. The initial condition (t_0, x_0) is a point in the domain of f, i.e., $(t_0, x_0) \in \Omega$. In higher dimensions (namely, $n > 1$), the differential equation (2.1) generally extends to a family of coupled ODEs

$$\dot{x}_i(t) = f_i\big(t, x_1(t), x_2(t), \ldots, x_n(t)\big), \quad x_i(t_0) = x_{i,0}, \quad i = 1, \ldots, n. \tag{2.2}$$

The system is nonlinear if the function f in equation (2.1), or, if at least one of the functions $\{f_i\}$ in equation (2.2), is nonlinear in one or more of its arguments. For the sake of notational simplicity, the discussion to follow in the rest of this section is based on the one-dimensional case, that is, system (2.1), for $n = 1$. However, all of the considerations also hold for $n > 1$.

Say that the solution of system (2.1) is sought at some time t_f. In order to approximate said solution numerically, the time variable must first be discretized. This is frequently done by defining

$$t_j = t_0 + j \cdot h, \tag{2.3}$$

where t_j is the time level j for integer j, and h is some increment which is smaller than $t_f - t_0$. Typically, the time increment is chosen such that an integer number of step lengths h equals the difference $t_f - t_0$. With the discretized time, the numerical solution of system (2.1) is found by successive applications of some numerical integration method. The Runge-Kutta family of numerical methods for ODE systems is a common choice, and will be elaborated upon in greater detail in section 2.1.1.

All numerical integration schemes fall into one of two categories; explicit and implicit methods. Explicit methods are characterized by computing the state of the system at a later time, based on the state of the system at the current time (in some cases, the state at earlier times are also considered). Implicit methods, however, involve the solution of an equation in which both the current and the later state of the system are involved. Thus, a generic, explicit method for computing an approximation of the system's state at time $t + h$, given its state at t, can be expressed as

$$x(t + h) = F\big(x(t)\big), \tag{2.4a}$$

while, for implicit methods, an equation of the sort

$$G\big(x(t), x(t + h)\big) = 0, \tag{2.4b}$$

is solved to find an approximation of $x(t + h)$.

In solving linear ODEs, implicit methods require the solution of a linear system at every time step. Typically, implicit methods are more computationally demanding than explicit methods. The main selling point of implicit methods is that they are more numerically stable than explicit methods. This property means that implicit methods are particularly well-suited for *stiff* systems, i.e., physical systems with highly disparate time scales (Hairer and Wanner 1996, p.2). For such systems, most explicit methods are unstable, unless the time step h is made exceptionally small, rendering these methods practically useless. For *nonstiff* systems, however, implicit methods behave similarly to their explicit analogues in terms of numerical accuracy and convergence properties.

Irrespective of which kind of numerical integration method is employed (see equation (2.4)), one obtains an approximation of the true solution of system (2.1) *at* the discrete time levels, that is,

$$x_j \approx x(t_j), \tag{2.5}$$

where $x(t)$ is the exact solution at time t. The accuracy of the approximation, however, depends on both the numerical integration method and the time step length h used for the temporal discretization. Given the discrete nature of numerical methods, approximate solutions are not known at arbitrary times t. One way of obtaining approximations of the true solution *inbetween* the discrete time levels is by means of interpolation — a numerical technique which will be elaborated upon in section 2.1.2.

2.1.1 *The Runge-Kutta family of numerical ODE solvers*

In numerical analysis, the Runge-Kutta family of methods is a popular collection of implicit and explicit iterative methods, used in temporal discretization in order to obtain numerical approximations of the *true* solutions of systems like (2.1). The German mathematicians C. Runge and M.W. Kutta developed the first of the family's methods at the turn of the twentieth century (Hairer, Nørsett, and Wanner 1993, p.134). The general outline of what is now known as a Runge-Kutta method is as follows:

Definition 1 *(Runge-Kutta methods).*
Let s be an integer and $\{a_{i,j}\}_{i,j=1}^{s}$, $\{b_i\}_{i=1}^{s}$ and $\{c_i\}_{i=1}^{s}$ be real coefficients.
Let h be the numerical step length used in the temporal discretization.
Then, the method

$$\begin{aligned} k_i &= f\left(t_n + c_i h, x_n + h \sum_{j=1}^{s} a_{i,j} k_j\right), \quad i = 1, \ldots, s, \\ x_{n+1} &= x_n + h \sum_{i=1}^{s} b_i k_i, \end{aligned} \tag{2.6}$$

is called an *s-stage Runge-Kutta method* for the system (2.1).

The main reason to include multiple stages in a Runge-Kutta method is to improve the numerical accuracy of the computed solutions. The *order* of a Runge-Kutta method can be defined as follows:

Definition 2 *(Order of Runge-Kutta methods).*
A Runge-Kutta method, given by equation (2.6), is of *order p* if, for sufficiently smooth systems (2.1), the local error e_n scales as h^{p+1}. That is:

$$e_n = \|x_n - u_{n-1}(t_n)\| \leq K h^{p+1} \tag{2.7}$$

where $u_{n-1}(t)$ is the exact solution of the ODE in system (2.1) at time t, subject to the initial condition $u_{n-1}(t_{n-1}) = x_{n-1}$, and K is a numerical constant. This is true if the Taylor series for the exact solution $u_{n-1}(t_n)$ and the numerical solution x_n coincide up to (and including) the term h^p.

The *global* error

$$E_n = x_n - x(t_n), \tag{2.8}$$

where $x(t)$ is the exact solution of system (2.1) at time t, accumulated by n repeated applications of the numerical method, can be estimated by

$$|E_n| \leq C \sum_{l=1}^{n} |e_l|, \tag{2.9}$$

where C is a numerical constant, depending on both the right hand side of the ODE in system (2.1) and the difference $t_n - t_0$. Making use of definition 2, the global error is limited from above by

$$\begin{aligned} |E_n| &\leq C \sum_{l=1}^{n} |e_l| \leq C \sum_{l=1}^{n} |K_l| \, h^{p+1} \leq C \max_l \left\{|K_l|\right\} n \, h^{p+1} \\ &\leq C \max_l \left\{|K_l|\right\} \frac{t_n - t_0}{h} \, h^{p+1} \leq \widetilde{K} h^p, \end{aligned} \tag{2.10}$$

where $\widetilde{K}$ is a numerical constant. Equation (2.10) demonstrates that, for a p-th order Runge-Kutta method, the global error can be expected to scale as h^p.

In definition 1, the matrix $(a_{i,j})$ is commonly called the *Runge-Kutta matrix*, while the coefficients $\{b_i\}$ and $\{c_i\}$ are known as the *weights* and *nodes*, respectively. Since the 1960s, it has been customary to represent Runge-Kutta methods, given by equation (2.6), symbolically, by means of mnemonic devices known as Butcher tableaus (Hairer, Nørsett, and Wanner 1993, p.134). The Butcher tableau for a general s-stage Runge-Kutta method, as introduced in definition 1, is presented in table 2.1. For explicit Runge-Kutta methods, the Runge-Kutta matrix $(a_{i,j})$ is lower triangular. Similarly, for fully implicit Runge-Kutta methods, the Runge-Kutta matrix is upper triangular. The difference between explicit and implicit methods is outlined in equation (2.4).

Table 2.1: Butcher tableau representation of generic s-stage Runge-Kutta methods.

c_1	$a_{1,1}$	$a_{1,2}$	$\cdots$	$a_{1,s}$
c_2	$a_{2,1}$	$a_{2,2}$	$\cdots$	$a_{2,s}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	$a_{s,1}$	$a_{s,2}$	$\cdots$	$a_{s,s}$
	b_1	b_2	$\cdots$	b_s

During the first half of the twentieth century, a substantial amount of research was conducted in order to develop numerically robust, high-order, explicit Runge-Kutta methods. The idea was that using such methods would mean one could resort to larger time increments h without sacrificing precision in the computed solution. However, the required number of stages s grows quicker than linearly as a function of the required order p. It has been proven that, for $p \geq 5$, no explicit Runge-Kutta method of order p with $s = p$ stages exists (Hairer, Nørsett, and Wanner 1993, p.173). This is one of the reasons for the attention shift from the latter half of the 1950s and onwards, towards so-called *embedded* Runge-Kutta methods.

The basic idea of embedded Runge-Kutta methods is that they, aside from the numerical approximation x_{n+1}, yield a second approximation $\widehat{x}_{n+1}$. The difference between the two approximations then provides an estimate of the local error of the less precise result, which can be used for automatic step size control (Hairer, Nørsett, and Wanner 1993, pp.167–168). The trick is to construct two independent, explicit Runge-Kutta methods which both use the *same* function evaluations. This results in practically obtaining the two solutions for the price of one, in terms of computational complexity. The Butcher tableau of a generic, embedded, explicit Runge-Kutta method is illustrated in table 2.2.

For embedded methods, the coefficients are tuned such that

$$x_{n+1} = x_n + h \sum_{i=1}^{s} b_i k_i \tag{2.11a}$$

is of order p, and

$$\widehat{x}_{n+1} = x_n + h \sum_{i=1}^{s} \widehat{b}_i k_i \tag{2.11b}$$

is of order $\widehat{p}$, typically with $\widehat{p} = p+1$. Which of the solutions is used to continue the numerical integration, depends on the integration method in question. In the following, the solution which is *not* used to continue the integration, will be referred to as the *interpolant* solution.

Table 2.2: Butcher tableau representation of generic, embedded, explicit Runge-Kutta methods.

c_1					
c_2	$a_{2,1}$				
c_3	$a_{3,1}$	$a_{3,2}$			
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
c_s	$a_{s,1}$	$a_{s,2}$	$\cdots$	$a_{s,s-1}$	
	b_1	b_2	$\cdots$	b_{s-1}	b_s
	$\widehat{b}_1$	$\widehat{b}_2$	$\cdots$	$\widehat{b}_{s-1}$	$\widehat{b}_s$

There exists an abundance of Runge-Kutta methods; many of which are fine-tuned for specific constraints, such as problems of varying degrees of stiffness. Based on prior investigations — such as the work done by Løken (2017) — using explicit, high order, embedded Runge-Kutta methods to compute Lagrangian coherent structures (which will be elaborated upon in section 2.3) consistently yields accurate solutions at lower computational cost than the most common fixed step size methods. Accordingly, the Dormand-Prince 8(7) method — consisting of an eighth order solution with a seventh order interpolant — was chosen as the single, multipurpose, numerical ODE solver for this project.

Note that the concept of *order* is less well-defined for embedded methods than for fixed step size methods, as a direct consequence of the adaptive time step. Although the *local* errors of each integration step scale as per equation (2.7), the bound on the *global* (i.e., observable) error suggested in equation (2.10) is invalid, as the time step is, in general, different for each integration step.

Butcher tableau representations of the classical 4^{th}-order Runge-Kutta method and the embedded Dormand-Prince 8(7) method are available in tables 2.3 and 2.4; where the latter has been typeset in landscape orientation for the reader's convenience. Details on how the dynamic time step of the Dormand-Prince 8(7) method was implemented will be presented in section 3.3.2.

Table 2.3: Butcher tableau representing the classical 4^{th}-order Runge-Kutta method.

0				
$1/2$	$1/2$			
$1/2$	0	$1/2$		
1	0	0	1	
	$1/6$	$1/3$	$1/3$	$1/6$

Table 2.4: Butcher tableau for the Dormand-Prince 8(7) embedded Runge-Kutta method. The $\widehat{b}$ coefficients give an 8^{th}-order accurate solution used to continue the integration. The b coefficients yield a 7^{th}-order interpolant, which can be used to estimate the error of the numerical approximation, and to dynamically adjust the time step. The provided coefficients are rational approximations, accurate to about 24 significant decimal digits. For reference, see Prince and Dormand (1981).

0													
$\frac{1}{18}$	$\frac{1}{18}$												
$\frac{1}{12}$	$\frac{1}{48}$	$\frac{1}{16}$											
$\frac{1}{8}$	$\frac{1}{32}$	0	$\frac{3}{32}$										
$\frac{5}{16}$	$\frac{5}{16}$	0	$\frac{-75}{64}$	$\frac{75}{64}$									
$\frac{3}{8}$	$\frac{3}{80}$	0	0	$\frac{3}{16}$	$\frac{3}{20}$								
$\frac{59}{400}$	$\frac{29443841}{614563906}$	0	0	$\frac{77736358}{692538347}$	$\frac{-28693883}{1125000000}$	$\frac{23124283}{1800000000}$							
$\frac{93}{200}$	$\frac{16016141}{946692911}$	0	0	$\frac{61564180}{158732637}$	$\frac{22789713}{633445777}$	$\frac{545815736}{2771057229}$	$\frac{-180193667}{1043307555}$						
$\frac{5490023248}{9719169821}$	$\frac{39632708}{573591083}$	0	0	$\frac{-433636366}{683701615}$	$\frac{-421739975}{2616292301}$	$\frac{100302831}{723423059}$	$\frac{790204164}{839813087}$	$\frac{800635310}{3783071287}$					
$\frac{13}{20}$	$\frac{246121993}{1340847787}$	0	0	$\frac{-37695042795}{15268766246}$	$\frac{-309121744}{1061227803}$	$\frac{-12992083}{490766935}$	$\frac{6005943493}{2108947689}$	$\frac{393006217}{1396673457}$	$\frac{123872331}{1001029789}$				
$\frac{1201146811}{1299019798}$	$\frac{-1028468189}{846180014}$	0	0	$\frac{8478235783}{508512851}$	$\frac{1311729495}{1432422823}$	$\frac{-10304129995}{1701304382}$	$\frac{-48777925059}{3047939560}$	$\frac{15336726248}{1032824649}$	$\frac{-45442868181}{339846796}$	$\frac{3065993473}{5917172653}$			
1	$\frac{185892177}{718116043}$	0	0	$\frac{-3185094517}{667107341}$	$\frac{-477755414}{1098053517}$	$\frac{-703635378}{230739211}$	$\frac{5731566787}{1027545527}$	$\frac{5232866602}{850066563}$	$\frac{-4093664535}{808688257}$	$\frac{3962137247}{1805957418}$	$\frac{65686358}{487910083}$		
1	$\frac{403863854}{491063109}$	0	0	$\frac{-5068492393}{434740067}$	$\frac{-411421997}{543043805}$	$\frac{652783627}{914296604}$	$\frac{11173962825}{925320556}$	$\frac{-13158990841}{6184727034}$	$\frac{3936647629}{1978049680}$	$\frac{-160528059}{685178525}$	$\frac{248638103}{1413531060}$	0	
	$\frac{13451932}{455176623}$	0	0	0	0	$\frac{-808719486}{976000145}$	$\frac{1757004468}{5645159321}$	$\frac{656045339}{265891186}$	$\frac{-3867574721}{1518517206}$	$\frac{465885868}{322736535}$	$\frac{53011238}{667516719}$	$\frac{2}{45}$	0
	$\frac{1400451}{335480064}$	0	0	0	0	$\frac{-59238493}{1068277825}$	$\frac{181606767}{758867731}$	$\frac{56129285}{797845732}$	$\frac{-1041891430}{1371343529}$	$\frac{760417239}{1151165299}$	$\frac{118820643}{751138087}$	$\frac{-528747749}{2220607170}$	$\frac{1}{4}$

2.1.2 *Spline interpolation of discrete data*

All naturally ocurring physical systems can only be known partially, either by means of limited measurements or grid based model output. Thus, interpolating (i.e., estimating) the measurement data becomes a requirement when describing the dynamics of systems which depend on measurement data from inbetween the sampling or grid points. Spline interpolation involves approximating a discretely sampled function by a series of piecewise defined polynomials. According to Stoer and Bulirsch (2002, p.93), spline interpolation is a popular tool within the field of numerical analysis, due to yielding smooth interpolation curves with limited interpolation error when using low degree polynomial pieces. Furthermore, the local nature of spline interpolated functions means that such functions are less prone to oscillatory behaviour when using high order polynomials. This is in stark constrast to regular, global polynomial interpolation, which exhibits strong global dependence on local properties. In particular, if the function to be approximated is badly behaved anywhere within the interval of approximation, then the approximation by global polynomial interpolation is poor everywhere (de Boor 1978, p.17).

A generic interpolation problem can be described in terms of a family of functions

$$\Theta(\mathbf{x}; \beta_0, \ldots, \beta_n), \tag{2.12}$$

each of which characterized by the $n+1$ parameters $\{\beta_i\}$, with $\mathbf{x}$ containing the independent variables of the problem. Given a set of $n+1$ discrete measurements — each defined by a set of coordinates and function values $(\mathbf{x}_i, f_i)$ where $\mathbf{x}_i \neq \mathbf{x}_j$ for $i \neq j$ and $f_i = f(\mathbf{x}_i)$ — the interpolation problem reduces to finding parameters $\{\beta_i\}$ such that

$$\Theta(\mathbf{x}_i; \beta_0, \ldots, \beta_n) = f_i, \quad i = 0, \ldots, n \tag{2.13}$$

is satisfied. According to Stoer and Bulirsch (2002, pp.38–39), spline interpolation problems (amongst others) can be classified as linear interpolation problems, meaning that the family of interpolation functions (cf. equation (2.12)) can be expressed as

$$\Theta(\mathbf{x}; \beta_0, \ldots, \beta_n) = \sum_{i=0}^{n} \beta_i \Theta_i(\mathbf{x}). \tag{2.14}$$

In the following, let the coordinates $\{\mathbf{x}_i\}$, function values $\{f_i\}$, and sampling points $\{(\mathbf{x}_i, f_i)\}$ be denoted by *support abscissas*, *support ordinates*, and *support points*, respectively.

Solving an interpolation problem by means of spline interpolation is done by determining the set of parameters $\{\beta_i\}$ of equations (2.13) and (2.14), with the family of functions $\{\Theta_i\}$ limited to spline functions. These functions, often denoted as *splines*, are connected through the use of a partition. Consider the one-dimensional case for the sake of notational simplicity — the considerations to follow also hold for higher dimensions, but invariably introduces notational clutter. The partition

$$\Delta: \quad \{a = x_0 < x_1 < \cdots < x_n = b\} \tag{2.15}$$

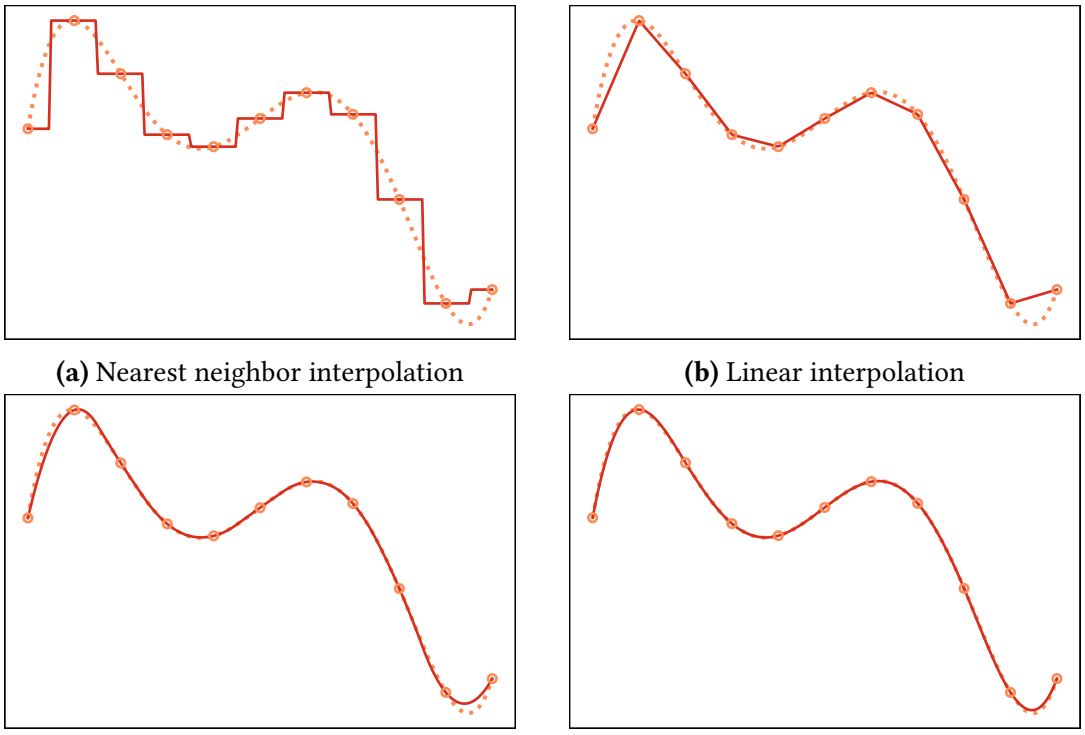
of the closed interval $[a, b]$ determines the domains of the piecewise polynomial spline functions $\mathcal{S}$ in the set $\mathcal{S}_\Delta$. These spline functions are joined at support abscissas, which, in the context of splines, are called *knots*.

Stoer and Bulirsch (2002, p.107) define a *piecewise polynomial function* as follows:

Definition 3 *(Piecewise polynomial functions).*
A real-valued function f is called a *piecewise polynomial function* of *order* k, or *degree* $k-1$, if it, for each $i = 0, \ldots, n-1$, when restricted to the subinterval (x_i, x_{i+1}) of the partition given in equation (2.15), corresponds to a polynomial $p_i(x)$ of degree less than or equal to $k-1$.

In order to obtain a one-to-one correspondence between the function f and the polynomial sequence $\big(p_0(x), p_1(x), \ldots, p_{n-1}(x)\big)$, *define* f at the knots $\{x_i\}_{i=0}^{n-1}$, so that the function becomes continuous from the right.

Accordingly, spline functions $\mathcal{S}_\Delta$ of order k are piecewise polynomial functions which are $k-1$ times continuously differentiable at the interior knots — that is, $\{x_i\}_{i=1}^{n-1}$, of the partition Δ. These k^{th}-order piecewise polynomials are uniquely determined by $k+1$ coefficients, of which k are given by the $k-1$ derivatives and the function value at their left bordering knot, and the last coefficient is given by the function value at the right bordering knot, for each interval in the partition Δ (Stoer and Bulirsch 2002, pp.107–108).



(a) Nearest neighbor interpolation

(b) Linear interpolation

(c) Quadratic B-spline interpolation

(d) Cubic B-spline interpolation

Figure 2.1: A selection of commonly used interpolation methods applied to a discretely sampled, high degree polynomial (dashed). The sampling points (knots) are shown as hollow circles. The splines in (c) and (d) are of second and third order, respectively (see definition 3). Observe how higher order splines yield increasingly accurate and smooth interpolations.

B-splines are a family of nonnegative spline functions which have minimal support for any given degree, smoothness, and domain partition. Furthermore, any spline function of a given degree can be expressed as a linear combination of B-splines of the same degree (Stoer and Bulirsch 2002, pp.107–110). Therefore, B-splines provide the foundation of efficient and

numerically stable computations of splines. A selection of commonly used interpolation methods applied to a discretely sampled, high degree polynomial is shown in figure 2.1. From the figure, the increased precision of higher order splines when applied to continuous functions is readily apparent. Note, however, that higher order interpolation methods require more sampling points than lower order methods. In particular, at least $k + 1$ samples are required in order to construct a k^{th}-order spline. In higher dimensions, that is, with data which depends on several other (independent) variables, the required amount of samples increases rapidly with the interpolation order. Consequently, the use of cubic B-splines constitutes a popular method for general-purpose interpolation, providing a good balance between numerical accuracy and computational complexity.

2.2 The type of flow systems considered

We consider flow in three-dimensional dynamical systems of the form

$$\dot{\mathbf{x}} = \mathbf{v}(t, \mathbf{x}), \quad \mathbf{x} \in \mathcal{U}, \quad t \in \mathcal{I}, \tag{2.16}$$

i.e., systems defined for the finite time interval $\mathcal{I}$ on an open, bounded subset $\mathcal{U}$ of $\mathbb{R}^3$. In addition, the velocity field $\mathbf{v}$ is assumed to be smooth in its arguments. Depending on the exact nature of the velocity field $\mathbf{v}$, analytical particle trajectories, that is, analytical solutions of system (2.16), may or may not exist. The flow particles are assumed to be infinitesimal and massless, i.e., non-interacting *tracers* of the overall circulation.

Consider a finite time interval $[t_0, t_1] \subset \mathcal{I}$ such that all tracer trajectories $\mathbf{x}(t; t_0, \mathbf{x}_0)$ — denoting the time-t position of a tracer which was located at $\mathbf{x}_0$ at time t_0 — in the system given by equation (2.16) are defined for all times $t \in [t_0, t_1]$. Then, the flow map is defined as

$$\boldsymbol{\phi}_{t_0}^{t}(\mathbf{x}_0) = \mathbf{x}(t; t_0, \mathbf{x}_0), \tag{2.17}$$

that is, the flow map mathematically describes the movement of tracers from one point in time to another. In general, the flow map is as smooth as the underlying velocity field (cf. system (2.16)) (Farazmand and Haller 2012a). In Lagrangian flow analysis, the *Jacobian matrix* of the flow map $\boldsymbol{\phi}_{t_0}^{t}$ plays a significant role. Component-wise, the Jacobian matrix of a general vector-valued function $\mathbf{f}$ is defined as

$$(\boldsymbol{\nabla}\mathbf{f})_{i,j} = \frac{\partial f_i}{\partial x_j}, \quad \mathbf{f} = \mathbf{f}(\mathbf{x}) = \big(f_1(\mathbf{x}), f_2(\mathbf{x}), \dots\big), \tag{2.18}$$

which, for our three-dimensional flow, reduces to

$$\boldsymbol{\nabla}\mathbf{f} = \begin{pmatrix} \dfrac{\partial f_1}{\partial x} & \dfrac{\partial f_1}{\partial y} & \dfrac{\partial f_1}{\partial z} \\ \dfrac{\partial f_2}{\partial x} & \dfrac{\partial f_2}{\partial y} & \dfrac{\partial f_2}{\partial z} \\ \dfrac{\partial f_3}{\partial x} & \dfrac{\partial f_3}{\partial y} & \dfrac{\partial f_3}{\partial z}. \end{pmatrix}. \tag{2.19}$$

Making use of the definition of the flow map (cf. equation (2.17)) in conjunction with equation (2.16), one finds the following ordinary differential equation which describes the time evolution of the flow map:

$$\dot{\boldsymbol{\phi}} = \mathbf{v}(t, \boldsymbol{\phi}), \tag{2.20}$$

where t_0, t and $\mathbf{x}_0$ have been omitted in order to avoid notational clutter. These are, however, implicit by context. As the nabla operator is time-independent, equation (2.20) immediately yields an ODE for the time development of the directional derivatives of the flow map, namely

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{u} \cdot \boldsymbol{\nabla})\boldsymbol{\phi} = (\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{v}(t, \boldsymbol{\phi}), \tag{2.21}$$

which holds along any constant (unit) vector $\mathbf{u}$. On a regular Cartesian grid, equation (2.21) provides a coupled set of ODEs describing the time evolution of each component of the Jacobian of the flow map:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \phi_i}{\partial x_j}\right) &= \sum_k \left.\frac{\partial v_i}{\partial x_k}\right|_{(t,\boldsymbol{\phi})} \left.\frac{\partial \phi_k}{\partial x_j}\right|_t, \\ \left.\frac{\partial \phi_i}{\partial x_j}\right|_{t_0} &= \delta_{ij}, \quad \mathbf{x}_0 \in \mathcal{U}, \quad t \in [t_0, t_1], \end{aligned} \tag{2.22}$$

where the Kronecker delta is defined as

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{2.23}$$

The initial conditions for the Jacobi components reflect the fact that, for a regular Cartesian grid, the directional derivative of e.g. the x coordinate in the x direction is 1, but zero in the y and z directions, and so on.

For sufficiently smooth velocity fields, the flow map Jacobian $\boldsymbol{\nabla}\boldsymbol{\phi}_{t_0}^t$ can be computed, which allows the right Cauchy-Green strain tensor field to be defined as

$$\mathbf{C}_{t_0}^t(\mathbf{x}_0) = \left(\boldsymbol{\nabla}\boldsymbol{\phi}_{t_0}^t(\mathbf{x}_0)\right)^* \left(\boldsymbol{\nabla}\boldsymbol{\phi}_{t_0}^t(\mathbf{x}_0)\right), \tag{2.24}$$

where the asterisk refers to the adjoint operation, which, because the Jacobian $\boldsymbol{\nabla}\boldsymbol{\phi}_{t_0}^t$ is real-valued, equates to matrix transposition. Moreover, as the Jacobian of the flow map is invertible, the Cauchy-Green strain tensor $\mathbf{C}_{t_0}^t(\mathbf{x}_0)$ is symmetric and positive definite (Farazmand and Haller 2012a). Thus, it has three real, positive eigenvalues and orthogonal, real eigenvectors. Its eigenvalues λ_i and corresponding unit eigenvectors $\boldsymbol{\xi}_i$ are defined by

$$\begin{aligned} \mathbf{C}_{t_0}^t(\mathbf{x}_0)\boldsymbol{\xi}_i(\mathbf{x}_0) &= \lambda_i \boldsymbol{\xi}_i(\mathbf{x}_0), \quad i = 1, 2, 3, \\ \left\langle \boldsymbol{\xi}_i(\mathbf{x}_0), \boldsymbol{\xi}_j(\mathbf{x}_0) \right\rangle &= \delta_{ij}, \quad 0 < \lambda_1(\mathbf{x}_0) \leq \lambda_2(\mathbf{x}_0) \leq \lambda_3(\mathbf{x}_0), \end{aligned} \tag{2.25}$$

where the Kronecker delta is defined in equation (2.23), and the dependence of λ_i and $\boldsymbol{\xi}_i$ on t_0 and t has been suppressed, for the sake of notational brevity. The geometric interpretation of equation (2.25) is that a fluid element transported by the underlying flow undergoes the most stretching along the $\boldsymbol{\xi}_3$ axis, less stretching along the $\boldsymbol{\xi}_2$ axis, and the least stretching along the $\boldsymbol{\xi}_1$ axis. This concept is shown in figure 2.2.

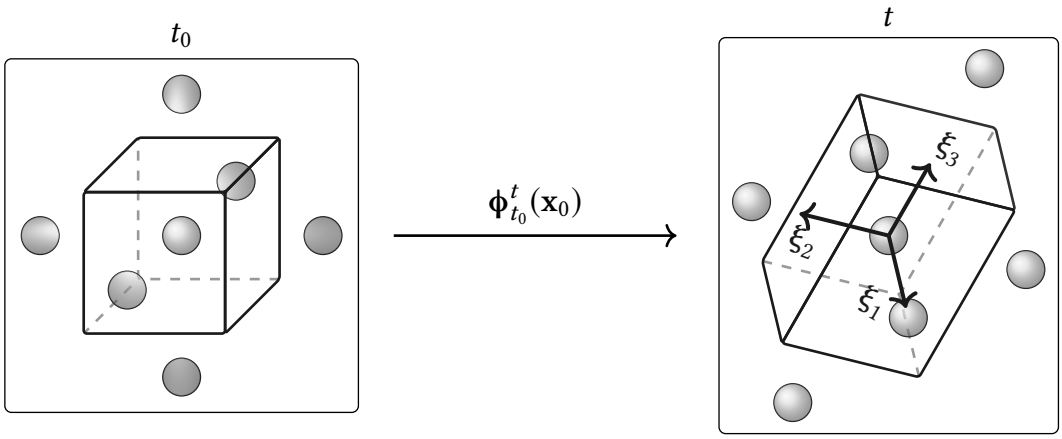


Figure 2.2: Geometric interpretation of the eigenvectors of the Cauchy-Green strain tensor. The central unit cell is stretched and deformed under the flow map $\phi_{t_0}^{t}(\mathbf{x}_0)$. The local stretching is largest in the direction of ξ_3, the eigenvector which corresponds to the largest eigenvalue, λ_3, of the Cauchy-Green strain tensor, defined in equation (2.25). Along the ξ_i axes, the stretch factors are given by $\sqrt{\lambda_i}$, respectively.

As the stretch factors along the ξ_i axes are given by the square roots of the corresponding eigenvalues, for incompressible flow, the eigenvalues satisfy

$$\lambda_1(\mathbf{x}_0)\lambda_2(\mathbf{x}_0)\lambda_3(\mathbf{x}_0) = 1 \quad \forall \ \mathbf{x}_0 \in \mathcal{U}, \tag{2.26}$$

where, in the context of tracer advection, incompressibility is equivalent to the velocity field $\mathbf{v}$ being divergence-free (i.e., $\nabla \cdot \mathbf{v} \equiv 0$ in system (2.16)).

2.3 Definition of Lagrangian coherent structures for three-dimensional flows

A necessary prerequisite for three-dimensional Lagrangian flow analysis is the concept of *material surfaces*, which Oettinger and Haller (2016) define as

Definition 4 *(Material surfaces).*
Consider a set of initial positions forming a two-dimensional surface $\mathcal{M}(t_0)$ at time t_0 in $\mathcal{U}$. Its time-t image, $\mathcal{M}(t)$, is obtained under the flow map as

$$\mathcal{M}(t) = \phi_{t_0}^{t}\big(\mathcal{M}(t_0)\big). \tag{2.27}$$

The union of *all* time-t images, $\cup_{t\in[t_0,t_1]}\mathcal{M}(t)$, is a hypersurface in the extended phase space $\mathcal{U}\times\mathcal{I}$, called a *material surface*.

From here on, the entire material surface will be referred to by the notation $\mathcal{M}(t)$. In the extended phase space $\mathcal{U}\times\mathcal{I}$, the dynamical system governed by equation (2.27) is autonomous. In autonomous systems, different trajectories never intersect; thus, no material surfaces can be crossed by tracers (Strogatz 2014, p.150). However, only special material surfaces create coherence in the phase space $\mathcal{U}$ and thus act as observable transport barriers. Such material surfaces are generally referred to as *Lagrangian coherent structures* (henceforth abbreviated to LCSs).

Subsequently, LCSs can be described as time-evolving surfaces which shape coherent trajectory patterns in dynamical systems, defined over a finite time interval (Haller 2011). There are three main types of LCSs, namely *elliptic*, *hyperbolic* and *parabolic*. Rougly speaking, parabolic LCSs outline cores of jet-like trajectories, elliptic LCSs describe vortex boundaries, whereas hyperbolic LCSs are comprised of overall attractive or repelling manifolds. As such, hyperbolic LCSs practically act as organizing centers of observable tracer patterns (Onu, Huhn, and Haller 2015). Because hyperbolic LCSs provide the most readily applicable insight in terms of forecasting flow in e.g. oceanic currents, such structures have been the focus of this project.

2.3.1 *Hyperbolic LCSs in three dimensions*

The identification of LCSs for reliable forecasting requires necessity and sufficiency conditions, supported by mathematical theorems. Haller (2011) derived a variational LCS theory based on the Cauchy-Green strain tensor, defined by equation (2.24), from which the aforementioned conditions follow. The immediately relevant parts of Haller's theory are given in definitions 5–8 (Haller 2011).

Definition 5 *(Normally repellent material surfaces).*
A *normally repellent material surface* over the time interval $[t_0, t_1]$ is a compact material surface segment $\mathcal{M}(t)$ which is overall repelling, and on which the normal repulsion rate is greater than the tangential repulsion rate.

The required *compactness* of the material surface segment signifies that, in some sense, it must be topologically well-behaved. That the material surface is *overall repelling* means that nearby trajectories are repelled from, rather than attracted towards, the material surface. Lastly, requiring that the *normal* repulsion rate is greater than the *tangential* repulsion rate means that nearby trajectories are in fact driven away from the material surface, rather than being stretched along with it due to shear stress.

Definition 6 *(Repelling LCS).*
A *repelling LCS* over the time interval $[t_0, t_1]$ is a normally repelling material surface $\mathcal{M}(t_0)$ whose normal repulsion admits a pointwise non-degenerate maximum relative to any nearby material surface $\widehat{\mathcal{M}}(t_0)$.

Definition 7 *(Attracting LCS).*
An *attracting LCS* over the time interval $[t_0, t_1]$ is defined as a repelling LCS over the *backward* time interval $[t_1, t_0]$.

Definition 8 *(Hyperbolic LCS).*
A *hyperbolic LCS* over the time interval $[t_0, t_1]$ is a *repelling* or *attracting* LCS over the same time interval.

Note that the above definitions associate LCSs with the time interval $\mathcal{I}$ over which the dynamical system under consideration is known, or, at the very least, where information regarding the behaviour of tracers is sought. Generally, LCSs obtained over a time interval $\mathcal{I}$ do not necessarily exist over different time intervals (Farazmand and Haller 2012a).

For sufficiently smooth three-dimensional flow, the above definitions can be summarized as a set of mathematical existence criteria, based on the Cauchy-Green strain tensor (cf. equation (2.24)) (Haller 2011; Farazmand and Haller 2012a; Karrasch 2012; Farazmand and Haller 2012b). The existence criteria for repelling LCSs are given in theorem 1.

Theorem 1 *(Necessary and sufficient conditions for repelling LCSs in three dimensions).* Consider a compact material surface $\mathcal{M}(t) \subset \mathcal{U}$ evolving over the time interval $[t_0, t_1]$. Then $\mathcal{M}(t)$ is a repelling LCS over $[t_0, t_1]$ if and only if all of the following holds for all initial conditions $\mathbf{x}_0 \in \mathcal{M}(t_0)$:

$$\lambda_2(\mathbf{x}_0) \neq \lambda_3(\mathbf{x}_0) > 1, \tag{2.28a}$$

$$\left\langle \boldsymbol{\xi}_3(\mathbf{x}_0), \mathbf{H}_{\lambda_3}(\mathbf{x}_0)\boldsymbol{\xi}_3(\mathbf{x}_0) \right\rangle < 0 \tag{2.28b}$$

$$\boldsymbol{\xi}_3(\mathbf{x}_0) \perp \mathcal{M}(t_0), \tag{2.28c}$$

$$\left\langle \nabla\lambda_3(\mathbf{x}_0), \boldsymbol{\xi}_3(\mathbf{x}_0) \right\rangle = 0. \tag{2.28d}$$

In theorem 1, $\langle \cdot, \cdot \rangle$ signifies the Euclidean inner product, and $\mathbf{H}_{\lambda_3}$ denotes the Hessian matrix of the largest eigenvalues of the Cauchy-Green strain tensor field. Component-wise, the Hessian matrix of a general, smooth, scalar-valued function f is defined as

$$(\mathbf{H}_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}, \tag{2.29}$$

which, for our three-dimensional flow, reduces to

$$\mathbf{H}_f = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{pmatrix}. \tag{2.30}$$

Condition (2.28a) ensures that the normal repulsion rate is larger than the tangential stretch arising due to shear strain along the LCS, in accordance with definition 5. Conditions (2.28c) and (2.28d) suffice to enforce that the normal repulsion rate attains a local extremum along the LCS, relative to all nearby material surfaces. Lastly, condition (2.28b) ensures that this is a strict local maximum.

From condition (2.28c) and the orthormality of the Cauchy-Green strain eigenvectors (cf. equation (2.25)), it follows that any initial (that is, time-t_0) image of an LCS surface is everywhere tangent to the planes locally spanned by $\boldsymbol{\xi}_1(\mathbf{x}_0)$ and $\boldsymbol{\xi}_2(\mathbf{x}_0)$. Thus, an integral curve of any (normalized) linear combination of the $\boldsymbol{\xi}_1$- and $\boldsymbol{\xi}_2$-direction fields, launched from an arbitrary point of the surface $\mathcal{M}(t_0)$, will never leave $\mathcal{M}(t_0)$. Hence:

Remark 1 *(Invariance of time-t_0 images of repelling LCSs).* The time-t_0 image $\mathcal{M}(t_0)$ of any repelling LCS (definition 6) is an invariant manifold of the autonomous dynamical system

$$\dot{\mathbf{x}} = a\boldsymbol{\xi}_1(\mathbf{x}) + b\boldsymbol{\xi}_2(\mathbf{x}), \quad a^2 + b^2 = 1. \tag{2.31}$$

Note that the converse of remark 1 does not hold. That is, a material surface $\Xi(t_0)$ which is an invariant manifold of all (normalized) linear combinations of ξ_1 and ξ_2 does not necessarily correspond to a repelling LCS $\mathcal{M}(t_0)$ — unless $\Xi(t_0)$ also satisfies conditions (2.28a), (2.28b) and (2.28d) (Oettinger and Haller 2016).

3 Method

This chapter contains a complete description of our method for computing repelling LCSs. Details regarding the specific flow systems we used as test cases will be presented in sections 3.1 and 3.2. Our way of extracting the Cauchy-Green strain eigenvalues and -vectors (cf. equation (2.25)) from the aforementioned flow systems then follows in sections 3.3 and 3.4. As will be outlined in sections 3.5–3.11, the Cauchy-Green strain eigenvalues and -vectors were then used to compute manifolds as three-dimensional surfaces — parametrized in terms of points organized in *geodesic level sets* — from which repelling LCSs were extracted as subsets. Lastly, section 3.12 contains a succinct description of some optimization tweaks we introduced in order to limit the consumption of computational resources.

Note that we present two variants of the method of geodesic level sets for expanding a manifold by the addition of mesh points. Section 3.6 contains the framework for a level set approach which closely resembles the method introduced by Krauskopf, Osinga, et al. (2005) (see also Krauskopf and Osinga (2003)). However, by making use of the properties defining repelling LCSs (see theorem 1 in section 2.3.1), we were able to simplify the method of generating new mesh points considerably, resulting in a great speedup in terms of computation runtimes. Our adaption of the method of geodesic level sets for the specific case of repelling LCSs will hereafter be referred to as the *revised* approach to computing new mesh points, and is described in detail in section 3.7.

3.1 Flow systems defined by analytical velocity fields

Within all branches of computational science — perhaps particularly for the analysis of nonlinear systems of the form suggested in section 2.2 — analytical test cases are very useful. Especially as far as reproducibility is concerned. Hence, we chose two variants of the three-dimensional flow system commonly referred to as the *Arnold-Beltrami-Childress flow*, which has previously been subject to Lagrangian analysis (Blazevski and Haller 2014; Oettinger and Haller 2016), as our base cases. We present a steady variant in section 3.1.1, and an unsteady one in section 3.1.2, with the intention of investigating to what extent the introduction of time dependence results in altered repelling LCSs (more to follow in sections 3.3–3.11 and chapter 4).

3.1.1 *Steady Arnold-Beltrami-Childress flow*

The Arnold-Beltrami-Childress (henceforth abbreviated to ABC) flow is a three-dimensional, incompressible velocity field which solves the Euler equations exactly. It is a simple example of a fluid flow which can exhibit chaotic behaviour (Frisch 1995, p.204). In terms of the Cartesian coordinate vector $\mathbf{x} = (x, y, z)$, the system can be expressed mathematically as

$$\dot{\mathbf{x}} = \mathbf{v}(t, \mathbf{x}) = \begin{pmatrix} A\sin(z) + C\cos(y) \\ B\sin(x) + A\cos(z) \\ C\sin(y) + B\cos(x) \end{pmatrix}, \tag{3.1}$$

where A, B, and C are parameters which dictate the nature of the flow pattern. The inherent periodicity with regards to the Cartesian axes naturally leads to a domain of interest $\mathcal{U} = [0, 2\pi]^3$, with periodic boundary conditions imposed in x, y and z.

Here, the parameter values

$$A = \sqrt{3}, \quad B = \sqrt{2}, \quad C = 1, \tag{3.2}$$

were used, as has been common in litterature (e.g. by Oettinger and Haller (2016)); these values are known to result in chaotic tracer trajectories (Zhao et al. 1993). The time interval of interest for this system was $\mathcal{I} = [0, 5]$.

3.1.2 *Unsteady Arnold-Beltrami-Childress flow*

Inspired by Oettinger and Haller (2016), a temporally aperiodic modification of the ABC flow (equation (3.1)) was made by the replacements

$$\begin{aligned} B &\longrightarrow \widetilde{B}(t) = B + B \cdot k_0 \tanh(k_1 t) \cos\big((k_2 t)^2\big), \\ C &\longrightarrow \widetilde{C}(t) = C + C \cdot k_0 \tanh(k_1 t) \sin\big((k_3 t)^2\big), \end{aligned} \tag{3.3}$$

with A, B, and C given by equation (3.2), where the parameters values

$$k_0 = 0.3, \quad k_1 = 0.5, \quad k_2 = 1.5, \quad k_3 = 1.8, \tag{3.4}$$

were used. The fundamental idea of this modification is to further enhance the chaotic nature of the resulting flow patterns. Similarly modified ABC flow has previously been at the centre of other three-dimensional transport barrier investigations — including hyperbolic LCSs — such as the work of Blazevski and Haller (2014); albeit with quite different methods of computing said LCSs than the one considered here. Like for its stationary sibling, the time interval of interest for this system was $\mathcal{I} = [0, 5]$. The time dependence of the $\widetilde{B}$ and $\widetilde{C}$ coefficients is illustrated in figure 3.1.

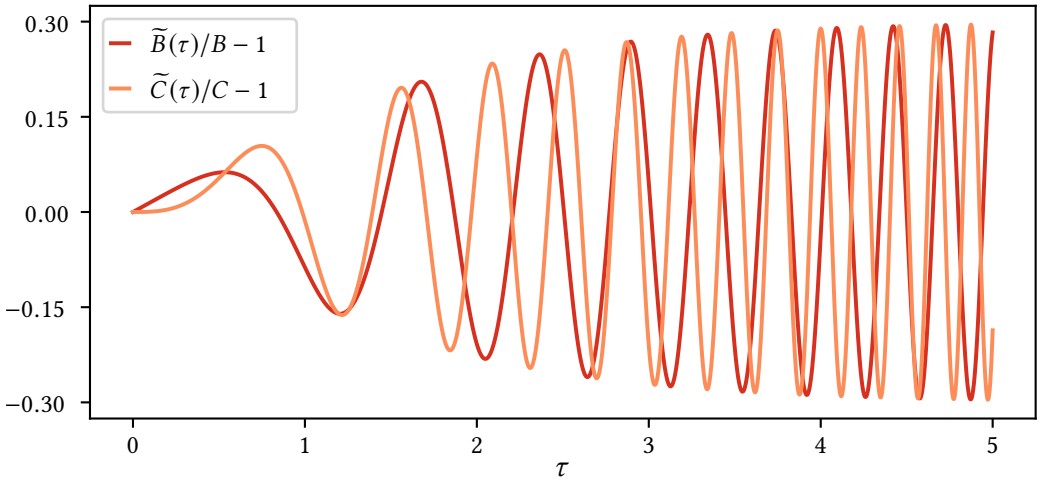


Figure 3.1: Time dependence of the coefficient functions for unsteady ABC flow, defined in equations (3.2)–(3.4).

3.2 Flow systems defined by gridded velocity data

As suggested in section 2.1.2, most kinds of computational science pertaining to the simulation of natural phenomena rely on some physical model for the generation of data, which can then be used to predict future states by solving appropriate differential equations. In order to demonstrate the applicability of Lagrangian analysis to real-life systems, we thus considered particle transport by oceanic currents in the Førde fjord. Section 3.2.1 contains a brief description of the relevance of transport in the Førde fjord in particular — in light of recent legislations and regulations — in addition to showcasing our domain of interest within said fjord. Then, in section 3.2.2, we present our way of interpolating the discrete model data in order to solve the set of transport equations pertaining to Lagrangian flow analysis (more to follow in sections 3.3 and 3.4).

3.2.1 *Oceanic currents in the Førde fjord*

In 2016, the mining company Nordic Mining ASA received permission from the Norwegian Ministry of Climate and Environment to extract rutile from the Engebø mountain in Naustdal, Norway (Garvik 2017; Haugan 2015). Furthermore, the company was authorized to deposit the mining waste into the nearby Førde fjord; a legislation which has been debated fiercely, and heavily protested against, ever since the original application was submitted in 2008. Early estimates suggest that, when operating at full scale, the mining operation will result in yearly oceanic mine tailings deposits in excess of five million tonnes (Garvik 2017).

Several centres of technical expertise — such as the Norwegian Institute of Marine Research — have publically advised against depositing mine tailings into the fjord, emphasizing the potentially severe negative consequences for marine life (Haugan 2015). Not only is the surrounding area a significant spawning ground for cod, there is always a possibility of particles being transported by the water currents such that they contaminate the outer edges of the fjord, or even the ocean. Accordingly, the use of LCSs in order to predict possible flow patterns for contaminants resulting from the deposit of mine tailings would be of great environmental interest.

To this end, gridded three-dimensional velocity data, modelling oceanic currents in the (depths of the) Førde Fjord, was made available by SINTEF Fisheries and Aquaculture, based on the SINMOD oceanic model (Slagstad and McClimans 2005). The data set considered here contains velocity data for the time period between June 1 2013, 00:00 and June 2 2013, 23:40, sampled in intervals of 20 minutes, with a horizontal resolution of of approximately 50 m and a vertical resolution varying from 25 m in the fjord depths, to 1 m near the surface. For our simulations, the time interval of interest was the 12 hour time window between 00:00 and 12:00 on June 1 2013 — practically ensuring the encapsulation of a tidal cycle.

We concentrated our analysis on the depths of the fjord. Therefore, we looked for LCSs in a region of water which was neither particularly close to the oceanic surface, nor reached the coastline when advected for the 12 hour duration of the time interval of interest. This limited our research to a $500\,\text{m} \times 500\,\text{m} \times 250\,\text{m}$ region, with depths ranging from 50 m to

300 m below the surface. A bird's-eye view of the region is shown in white in figure 3.2, which also contains a map view of the geographical surroundings.

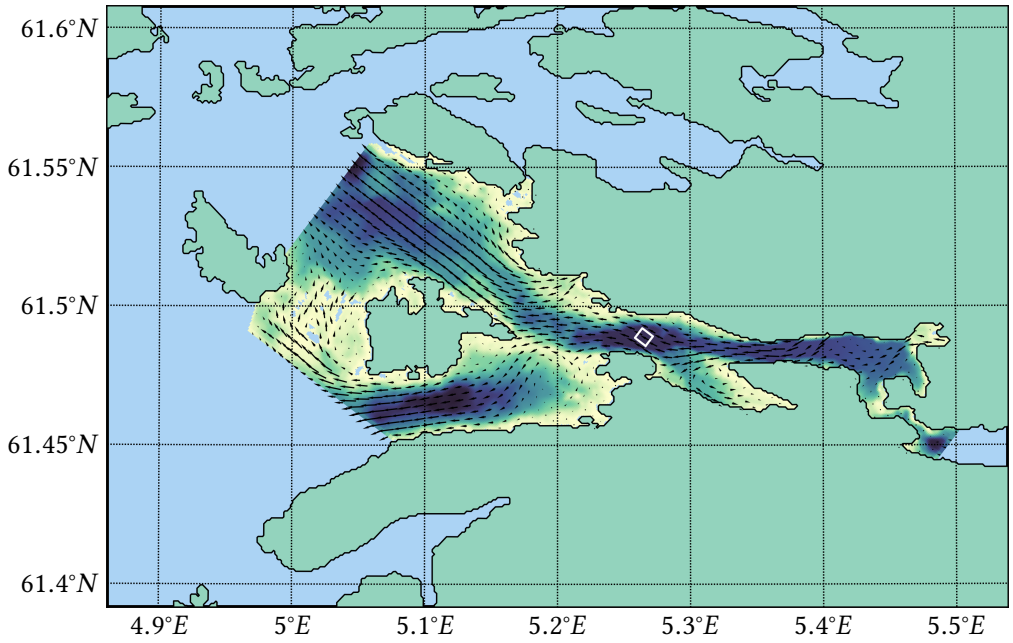


Figure 3.2: Stereographical map projection of the Førde fjord and its surroundings. The local fjord depths are indicated by a varying background color. A subset of the gridded velocity vectors indicates the macroscopic trends of the oceanic currents at a depth of 3 m below the surface. Outlined in white is a bird's-eye view of the main region of interest; namely, a region of water which was neither particularly close to the oceanic surface, nor struck the coastline when transported by the currents for the duration of the 12 hour time interval of interest.

3.2.2 *Interpolating gridded velocity data*

In order to describe transport phenomena in the Førde fjord, interpolating the discretely sampled velocity field becomes necessary. Based upon the considerations presented in section 2.1.2, we elected to do so by means of cubic B-splines in time and space. Thus, each of the velocity field's three components was considered to be a quadrivariate function of time and the three spatial coordinates.

Several multidimensional B-spline interpolation libraries are publically available under open source licensing. For this project, we elected to use the Bspline-Fortran library, partly motivated by its extensive documentation (Williams 2018). In particular, we made use of its `bspline_4d` derived type, which we, along with a subset of its type-bound procedures, made available in C by means of the Fortran standard interoperability with C-languages — that is, the `iso_c_binding` module, which is shipped with most modern Fortran compilers. From there, we wrote a thin wrapper class in C++, which was exposed to Python via Cython.

The choice of Python as our main programming language was made partly due to its relatively low development costs, in addition to its beneficial properties as a "glue language" — as examplified by the above — and the ease of parallelization by means of e.g. MPI

(the use of which will be outlined in greater detail in section 3.12). Moreover, by utilizing Fortran's reference-based subroutine call structure, in addition to pointers at C-level (typed memoryviews in Cython), we were able to minimize memory duplication. This could otherwise have been a significant bottleneck.

3.3 Computing the flow map and its directional derivatives

Computing the flow map Jacobian field is crucial, as it is used in our definition of the Cauchy-Green strain tensor field (cf. equation (2.24)) — whose eigenvalues and -vectors, in turn, form the basis of the LCS existence criteria given in equation (2.28). An outline of how we solved equation (2.16) in conjunction with equation (2.22) in order to obtain the final state of the flow map Jacobian field is presented in section 3.3.1. Furthermore, section 3.3.2 contains a detailed description of how the dynamic Runge-Kutta solver step size (see section 2.1.1) was implemented. How we then extracted the Cauchy-Green strain eigenvalues and -vectors from the final state flow map Jacobian field is the topic of section 3.4.

3.3.1 *Advecting a set of tracers*

The variational framework for computing LCSs is based upon the advection of non-interacting tracers, as described in section 2.2, by the systems mentioned in sections 3.1 and 3.2. The computational domains $\mathcal{U}$ were discretized by a set equidistant tracers, effectively creating a uniform grid with tracers placed on and within the domain boundaries of $\mathcal{U}$. The grid parameters are summarized in table 3.1.

Table 3.1: Grid parameters for advection in the considered flow systems. For the fjord system, the domain extents and grid spacings are given in units of metre. Also note that the grid spacings have been truncated to two significant decimal digits.

	Analytical ABC flow	Fjord model data
Computational domain	$[0, 2\pi]^3$	$[0, 500] \times [0, 500] \times [50, 300]$
N_x, N_y, N_z	256, 256, 256	200, 200, 100
$\Delta x = \Delta y = \Delta z$	$2.5 \cdot 10^{-2}$	2.5

In order to increase the precision of the computed Cauchy-Green strain tensor field, it is necessary to increase the accuracy with which one computes the Jacobian of the flow map, as their accuracies are intrinsically linked. This follows from equation (2.24). Accordingly, the flow map Jacobian was computed directly, by means of simultaneously solving the twelve coupled ODEs given by equations (2.16) and (2.22), letting the underlying velocity field transport the tracers. All twelve ODEs were solved simultaneously, using the Dormand-Prince 8(7) method (see section 2.1.1 and, in particular, table 2.4). The dynamic step length adjustment procedure will be outlined in detail in section 3.3.2.

In this framework, the tracer advection takes second stage to the "advection" of the components of the flow map Jacobian. As it turns out, the increase in mathematical complexity

which the coupling terms introduces is a small price to pay for the increased precision compared to the straightforward approach of applying a finite diffference scheme to the advected flow map (Oettinger and Haller 2016). This is also evident from previous "finite difference-based" LCS computing endeavors, in which the use of several grids of tracers was necessitated in order to accurately compute the flow map Jacobian (Løken 2017; Farazmand and Haller 2012a).

3.3.2 *The implementation of dynamic Runge-Kutta step size*

In order to implement automatic step size control, the procedure suggested by Hairer, Nørsett, and Wanner (1993, pp.167–168) was followed closely. A starting step size h needs to be prescribed; this generally differs based upon the (pseudo-)time scale of the underlying system. For the first solution step, the embedded Dormand-Prince 8(7) method, as described in section 2.1.1 and table 2.4, yields the two approximations x_1 and $\widehat{x}_1$, from which the difference $x_1 - \widehat{x}_1$ can be used as an estimate of the error of the less precise result. The idea is to force the error of the numerical solution to satisfy, componentwise:

$$\left|x_{1,i} - \widehat{x}_{1,i}\right| \leq \mathrm{sc}_i, \quad \mathrm{sc}_i = \mathrm{Atol}_i + \max\left(\left|x_{1,i}\right|, \left|\widehat{x}_{1,i}\right|\right) \cdot \mathrm{Rtol}_i, \tag{3.5}$$

where Atol_i and Rtol_i are the desired absolute and relative tolerances. For this project, the tolerance values

$$\mathrm{Atol}_i = 10^{-7}, \quad \mathrm{Rtol}_i = 10^{-7} \tag{3.6}$$

were used throughout.

As a measure of the numerical error,

$$\mathrm{err} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_{1,i} - \widehat{x}_{1,i}}{\mathrm{sc}_i}\right)^2} \tag{3.7}$$

is used. Then, err is compared to unity in order to find an optimal step size. From definition 2, it follows that err scales like h^{q+1}, where $q = \min(p, \widehat{p})$. Thus, under the expected scaling $\mathrm{err} \approx Kh^{q+1}$, and the assumption $1 \approx Kh_{\mathrm{opt}}^{q+1}$, one finds the optimal step size according to

$$h_{\mathrm{opt}} = h \cdot \left(\frac{1}{\mathrm{err}}\right)^{\frac{1}{q+1}}. \tag{3.8}$$

If $\mathrm{err} \leq 1$, the suggested solution step is accepted, the (pseudo-)time variable t is increased by h, and the step length is modified according to equations (3.8) and (3.9). Which of the two approximations x_{n+1} or $\widehat{x}_{n+1}$ is used to continue the integration generally depends on the embedded Runge-Kutta method in question. Continuing the integration with the higher order result is commonly referred to as *local extrapolation*. The Dormand-Prince 8(7) method is tuned in order to minimize the error of the higher order result; accordingly, local extrapolation was used throughout. If $\mathrm{err} > 1$, the solution step is rejected, and the step length decreased before attempting another step. The procedure for updating the time step can be summarized as follows:

$$h_{\mathrm{new}} = \begin{cases} \min(\mathrm{fac}_{\max} \cdot h, \mathrm{fac} \cdot h_{\mathrm{opt}}) & \text{if the solution step is accepted,} \\ \mathrm{fac} \cdot h_{\mathrm{opt}}, & \text{if the solution step is rejected,} \end{cases} \tag{3.9}$$

where fac and $\text{fac}_{\max}$ are numerical safety factors, intended to prevent increasing the step size *too* much, in order to make it more likely that the next step is accepted. Here, the parameter values

$$\text{fac} = 0.8, \quad \text{fac}_{\max} = 2.0, \tag{3.10}$$

were used throughout.

3.4 Computing Cauchy-Green strain eigenvalues and -vectors

Computing the Cauchy-Green strain tensor field directly, by performing a series of matrix products per its definition in equation (2.24), and then solving for its eigenvalues and -vectors turns out to be numerically disadvantageous (Oettinger and Haller 2016). In particular, this method leaves the smallest eigenvalues quite susceptible to numerical round-off error. A fully equivalent, more numerically sound way of identifying the Cauchy-Green strain eigenvalues and -vectors is based on performing an SVD decomposition of the Jacobian field of the flow map, i.e.,

$$\nabla \boldsymbol{\phi}_{t_0}^{t}(\mathbf{x}_0) = \mathbf{U}\Sigma\mathbf{V}^*, \tag{3.11}$$

where the asterisk refers to the adjoint operation, $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices, and Σ is a diagonal matrix with nonnegative real numbers — the *singular values* of $\nabla\boldsymbol{\phi}$ — on the diagonal. Because the flow map Jacobian is square, so too are the matrices $\mathbf{U}$, Σ, and $\mathbf{V}$. Moreover, as the flow map Jacobian is real-valued, so too are the matrices $\mathbf{U}$ and $\mathbf{V}$. The eigenvalues of the right Cauchy-Green strain tensor (cf. equations (2.24) and (2.25)) are given by the squares of the singular values, that is, $\lambda_i(\mathbf{x}_0) = \left(\sigma_i(\mathbf{x}_0)\right)^2$, and the corresponding orthonormal eigenvectors are found in the columns of $\mathbf{V}$.

Interpolating the Cauchy-Green strain eigenvalues

For computing LCSs, the Cauchy-Green strain eigenvalues frequently need to be evaluated inbetween the grid points. Moreover, as suggested by the existence criterion given in equation (2.28b), all of the second derivatives of $\lambda_3(\mathbf{x}_0)$ are also needed. Accordingly, the eigenvalues were interpolated by means of cubic trivariate B-splines, in order to ensure continuous second derivatives. For this purpose, the `bspline_3d` derived type from the Bspline-Fortran library (Williams 2018) was ported to Python using the techniques described in section 3.2.2.

Interpolating the Cauchy-Green strain eigenvectors

Just like the eigenvalues of the Cauchy-Green strain tensor field, its eigenvectors frequently need to be evaluated between the grid points in order to compute LCSs. Like the strain eigenvalues, the strain eigenvectors were interpolated by means of cubic trivariate B-splines, through the `bspline_3d` derived type from the Bspline-Fortran library (Williams 2018) — albeit with a twist, in order to remove local orientational discontinuities. In particular, the local stretch is equal in magnitude along any given negative ξ_i axis as that of its positive counterpart, and there is no *a priori* reason to expect the SVD decomposition (cf.

equation (3.11)) to follow any particular convention regarding the "sign" of the computed eigenvectors.

The interpolation routine outlined here is a generalization of a similar special-purpose linear interpolation routine which has previously been utilized to compute LCSs in two spatial dimensions (Onu, Huhn, and Haller 2015; Løken 2017). Our routine is based upon careful monitoring and local reorientation prior to cubic interpolation, and its two-dimensional equivalent is illustrated in figure 3.3 — the principles are similar in three dimensions, but illustrating a two-dimensional projection of the three-dimensional case simply became cluttered beyond comprehension.

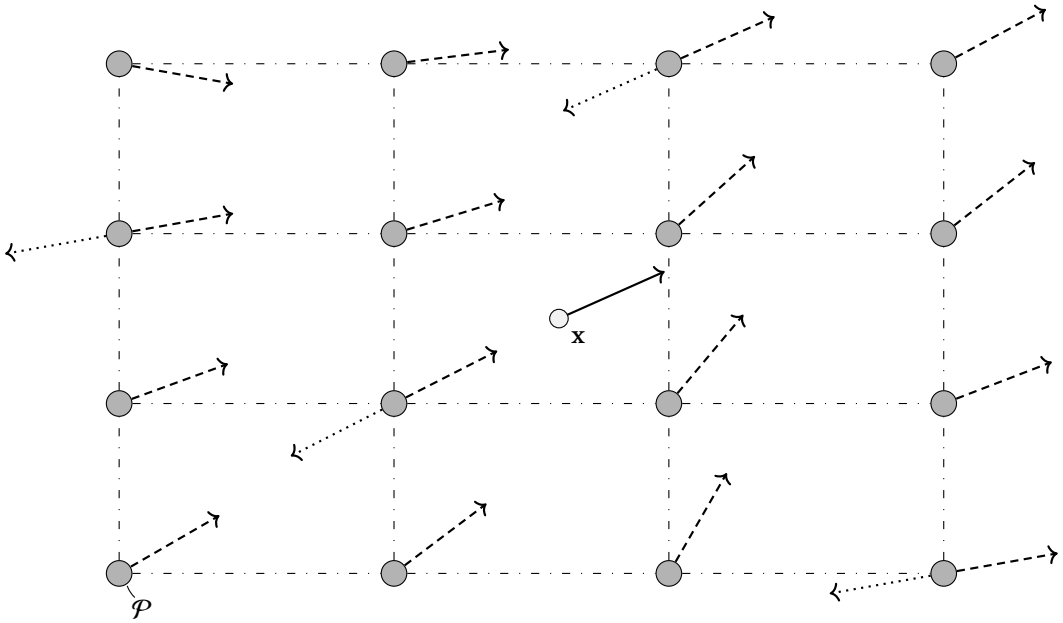


Figure 3.3: Conceptual illustration of the special-purpose cubic interpolation routine for the Cauchy-Green strain eigenvectors. The 64 nearest grid points (16 in two dimensions; here shown in gray) to any given coordinate $\mathbf{x}$ are identified. As the local stretch is equal in magnitude along the negative ξ_i axis as that of its positive counterpart, we are free to reverse any ξ_i vector which is rotated more than 90° with regards to the chosen pivot vector (at the grid point denoted by $\mathcal{P}$) prior to interpolating the components of ξ_i, making use of cubic B-splines. The vectors used in the local cubic interpolation are dashed, whereas the vectors which had to be reversed are dotted.

First, the 64 (in two dimensions: 16) nearest neighboring grid points corresponding to any given coordinate $\mathbf{x}$ are identified. Choosing a pivot vector at a corner of this local interpolation voxel, orientational discontinuities between the grid elements are found by inspecting the inner products of the ξ_i vectors of the remaining grid points with the pivot. Rotations exceeding 90° are identified by inner products with the pivot vector being negative, labelled as orientational discontinuities, and then corrected by reversing the direction of the corresponding vectors. For each of ξ_i's three components, cubic B-spline interpolation is used within the interpolation voxel in order to find $\xi_i(\mathbf{x})$, which is then normalized, like the ξ_i vectors defined at the grid points are per their definition, cf. equation (2.25).

3.5 Preliminaries for computing repelling LCSs in 3D flow by means of geodesic level sets

Repelling LCSs in three spatial dimensions are quite challenging to compute. Straightforward numerical integration of the flow in a strain eigendirection field suffices for Lagrangian analysis of two-dimensional systems (Farazmand and Haller 2012a; Løken 2017). In three dimensions, however, this is not the case; LCS existence criterion (2.28c) implies that three-dimensional repelling LCSs are everywhere simultaneously tangent to the ξ_1- *and* ξ_2-direction fields (see remark 1). Another way to interpret this extra degree of freedom — compared to the two-dimensional case — is that everywhere within three-dimensional repelling LCS, one is allowed to move "freely" within a plane which is orthogonal to $\xi_3(\mathbf{x})$. Thus, more sophisticated algorithms are needed in order to compute three-dimensional LCSs, than their two-dimensional counterparts. Here, we consider a variation of the method of geodesic level sets for computing repelling LCSs as invariant manifolds of the ξ_1 and ξ_2 direction fields (cf. remark 1), as presented by Krauskopf, Osinga, et al. (2005). The short presentation to follow in the next paragraph will be explained further in depth in the subsequent sections.

The method is based on the concept of developing an unstable manifold from a local neighborhood of an initial condition $\mathbf{x}_0$ (how the initial conditions are selected will be described in section 3.5.1). In particular, a small, closed curve C_1 forming a geodesic circle, consisting of mesh points which are all located in the tangent plane defined by the coordinate $\mathbf{x}_0$ and the unit normal $\xi_3(\mathbf{x}_0)$, separated from $\mathbf{x}_0$ by a distance δ_{init}, is assumed to be a part of the same manifold as $\mathbf{x}_0$. The idea is then to compute the next geodesic circle in a local, dynamic coordinate system, defined by hyperplanes which are orthogonal to the most recently computed geodesic circle. A set of accuracy parameters governs the number of next points by which the next geodesic circle is approximated, in solving a set of boundary value problems. During the computation, the interpolation error stays limited by the density of mesh points, so that the overall quality of the mesh is preserved (Krauskopf and Osinga 2003).

3.5.1 *Identifying suitable initial conditions for developing LCSs*

Inspired by the two-dimensional approach of Farazmand and Haller (2012a), in order to identify repelling LCSs, the first step was to identify the subdomain $\mathcal{U}_0 \subset \mathcal{U}$ in which existence conditions (2.28a), (2.28b) and (2.28d) are satisfied — as these conditions can be verified for individual points, unlike criterion (2.28c). All grid points in $\mathcal{U}_0$ would then be valid initial conditions for repelling LCSs. Of the aforementioned criteria, condition (2.28d) is the least straightforward to implement numerically, as identifying the zeros of inner products is prone to numerical round-off error. Our approach is based on comparing the value of λ_3 at a given grid point $\mathbf{x}_0$ to the values of λ_3 at the two points $\mathbf{x}_0 \pm \varepsilon\xi_3(\mathbf{x}_0)$, where ε is a number one order of magnitude smaller than the grid spacing. Should $\lambda_3(\mathbf{x}_0)$ be the largest of the three, signifying that $\mathbf{x}_0$ could be an approximate maximum of the λ_3 field along the (local) ξ_3 direction (which compliance with condition (2.28b) would confirm), the point $\mathbf{x}_0$ would be flagged as satisfying criterion (2.28d).

Using all of the points in $\mathcal{U}_0$ would invariably involve computing a lot of LCSs several times over — in particular, if two neighboring grid points are both part of $\mathcal{U}_0$, then they likely belong to the same manifold. In order to reduce the number of redundant calculations, the set of considered initial conditions was further reduced, by only checking whether every ν^{th} grid point along each axis belonged to $\mathcal{U}_0$, that is, only considering one in every ν^3 grid points in the entire domain as possible initial conditions. Because the number of grid points was different for the different types of flow (cf. table 3.1), so too was the pseudo-sampling frequency ν. The values for ε, ν, and the resulting number of initial conditions are given in table 3.2. Note that, using the given filtering parameters, the initial conditions reduced to a far more manageable number of grid points, than all of the grid points which satisfy the LCS conditions (2.28a), (2.28b) and (2.28d).

Table 3.2: Parameter choices for selecting LCS initial conditions. ε was made to be one order of magnitude smaller than the grid spacing, and ν was selected to be a common divisor of the number of grid points along each Cartesian abscissa, cf. table 3.1. Note that ε for the fjord model data is given in units of metre. Observe how the reduced set of initial conditions contains orders of magnitude fewer points than the total number of grid points which satisfy the LCS existence criteria (2.28a), (2.28b) and (2.28d).

	Analytical ABC flow	Fjord model data
ε	$5 \cdot 10^{\cdot -3}$	10^{-1}
ν	8	5
# initial conditions without ν-filtering	340 951 (steady) 361 461 (unsteady)	209 945
# initial conditions with ν-filtering	618 (steady) 676 (unsteady)	1 631

3.5.2 *Parametrizing the innermost level set*

For an initial condition $\mathbf{x}_0$, identified by means of the method outlined in section 3.5.1, the corresponding LCS must locally be tangent to the plane with unit normal given by $\xi_3(\mathbf{x}_0)$, as a consequence of LCS existence criterion (2.28c). Accordingly, the first geodesic level set is approximated by a set of n mesh points $\{\mathcal{M}_{1,j}\}_{j=1}^{n}$, placed in the aforementioned tangent plane, evenly distributed along a circle centered at $\mathbf{x}_0$ with radius δ_{init}. All of these points are assumed to be contained within the same manifold. Figure 3.4 shows where the points in the innermost level set are located, in relation to $\mathbf{x}_0$. The parameter δ_{init} was chosen small compared to the grid spacing, in order to limit the inherent errors of this linearization.

An interpolation curve C_1 was then made, with a view to representing the innermost level set in a smoother fashion. In particular, this interpolation curve was designed as a parametric spline. To this end, the points $\{\mathcal{M}_{1,j}\}_{j=1}^{n+1}$, where $\mathcal{M}_{1,n+1} = \mathcal{M}_{1,1}$, were ordered in clockwise or counterclockwise — merely a matter of perspective — fashion. Then, each mesh point was assigned an independent variable s based upon the cumulative interpoint distance along the ordered list of points, starting at $j = 1$; estimated by means of the Euclidean norm, then normalized by dividing through by the total interpoint distance around the entire initial

level set. Consequently, there was a one-to-one correspondence between the s-values and the mesh points, aside from $\mathcal{M}_{1,1}$, to which both $s = 0$ and $s = 1$ were mapped.

Next, we made lists containing the coordinates of the ordered mesh points $\{\mathcal{M}_{i,j}\}_{j=1}^{n+1}$; i.e., one list for each of the three Cartesian coordinates. Considering each of the lists of the mesh points' Cartesian coordinates as univariate functions of the pseudo-arclength parameter s, separate cubic B-splines were then made for each set of coordinates, making use of the `bspline_1d` extension type from the Bspline-Fortran library (Williams 2018), which was ported to Python as outlined in section 3.2.2. The constructed innermost level set and its associated interpolation curve is illustrated in figure 3.4. Interpolation curves for all subsequent level sets (the computation of which will be described in detail in the sections to follow) were made completely analogously to C_1.

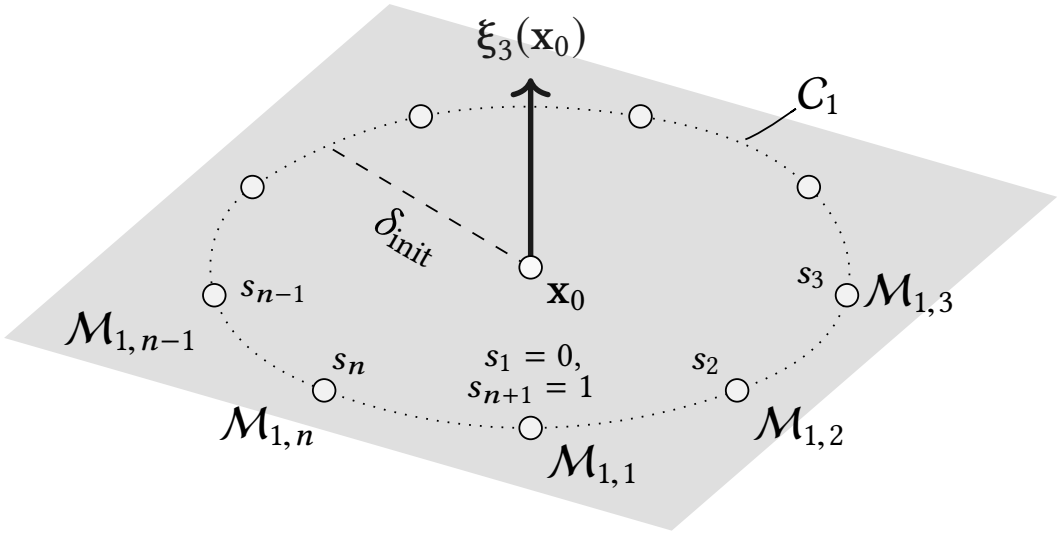


Figure 3.4: The construction of the innermost geodesic level set. An initial condition $\mathbf{x}_0$ is found by means of the method outlined in section 3.5.1. A set of n mesh points $\{\mathcal{M}_{1,j}\}_{j=1}^{n}$ is then evenly distributed within the plane defined by the point $\mathbf{x}_0$ and the unit normal $\xi_3(\mathbf{x}_0)$, which is shaded. Each mesh point is separated from $\mathbf{x}_0$ by a small distance δ_{init} (dashed). Using a normalized pseudo-arclength parameter s, the mesh point coordinates are interpolated using cubic B-splines, forming the smooth curve C_1 (dotted).

In the following, let $\mathbf{x}_{i,j}$ denote the location of mesh point $\mathcal{M}_{i,j}$. As suggested by Krauskopf, Osinga, et al. (2005), we next sought to develop a new level set, parametrized by a new set of points $\{\mathcal{M}_{2,j}\}$ located in the family of half-planes $\{\mathcal{H}_{1,j}\}_{j=1}^{n}$, extending radially outwards from the corresponding points $\{\mathcal{M}_{1,j}\}_{j=1}^{n}$ in the initial level set while being orthogonal to C_1. These half-planes are generally defined by the points $\mathbf{x}_{i,j}$ and the (unit) tangent vectors $\mathbf{t}_{i,j}$. For the innermost level set, these tangent vectors were computed as

$$\mathbf{t}_{1,j} = \frac{\xi_3(\mathbf{x}_0) \times (\mathbf{x}_{1,j} - \mathbf{x}_0)}{\left\| \xi_3(\mathbf{x}_0) \times (\mathbf{x}_{1,j} - \mathbf{x}_0) \right\|}. \tag{3.12}$$

For the subsequent level sets, Krauskopf, Osinga, et al. (2005) suggest determining the tangents $\mathbf{t}_{i,j}$ using the interpolation curve C_i, by drawing a vector between two points

equidistant to $\mathcal{M}_{i,j}$ in either direction along C_i. However, an inheritance-based approach was found to yield more smoothly parametrized manifolds, which were less sensitive to numerical noise. This approach, along with the treatment of special cases, will be explained in greater detail in the sections to follow (in particular, section 3.8.1).

A *guidance* vector $\boldsymbol{\rho}_{i,j}$ was computed for each of the mesh points $\{\mathcal{M}_{i,j}\}$, in order to keep track of the local (quasi-)radial direction. The guidance vectors for the innermost level set were computed as

$$\boldsymbol{\rho}_{1,j} = \frac{\mathbf{x}_{1,j} - \mathbf{x}_0}{\left\|\mathbf{x}_{1,j} - \mathbf{x}_0\right\|}. \tag{3.13}$$

For each mesh point in all ensuing level sets, the guidance vectors were computed relative to the coordinates of the point in the immediately preceding level set, from which the new point was computed. That is,

$$\boldsymbol{\rho}_{i,j} = \frac{\mathbf{x}_{i,j} - \mathbf{x}_{i-1,\hat{\jmath}}}{\left\|\mathbf{x}_{i,j} - \mathbf{x}_{i-1,\hat{\jmath}}\right\|}, \tag{3.14}$$

where the indices j and $\hat{\jmath}$ generally need not be the same, as there is generally not a one-to-one correspondence between points in subsequent level sets; see section 3.8.1 for details.

Note that, in computing new mesh points, organized in level sets, a descendant point $\mathcal{M}_{i+1,j}$ has to be computed for each ancestor point $\mathcal{M}_{i,j}$. This is due to the method being based on parametrizing manifolds as a series of smooth topological circles. Should this prove not to be possible, given a set of tolerance parameters which will be described in greater detail in the sections to come, the computation is stopped abruptly, leaving the manifold parametrized by however many of its geodesic level sets were successfully completed. Further details on the stopping criteria for the generation of new geodesic level sets will be presented in section 3.10.

3.6 Legacy approach to computing new mesh points

As tentatively suggested in section 3.5.2, each of the points in the first level set $\mathcal{M}_1 = \{\mathcal{M}_{1,j}\}_{j=1}^{n}$ is used to compute a point in the ensuing level set $\mathcal{M}_2$. This notion extends to all of the subsequent level sets; namely, the points in level set $\mathcal{M}_{i+1}$ are computed from the points in the prior level set $\mathcal{M}_i$. For reasons of brevity in the discussion to follow, we denote the points $\{\mathcal{M}_{i,j}\}$ and $\{\mathcal{M}_{i+1,j}\}$ as *ancestor* and *descendant points*, respectively. Furthermore, the set of mesh points which can be traced backwards to a single, common ancestor, is referred to as a *point strand*. The considerations to follow rely on each mesh point $\mathcal{M}_{i,j}$ inheriting its tangential vector from its direct ancestor; that is, $\mathbf{t}_{i,j} := \mathbf{t}_{i-1,j}$. The treatment of the special cases of this inheritance-based approach will be described in greater detail in section 3.8.1.

From the mesh point $\mathcal{M}_{i,j}$, we wish to place a new mesh point $\mathcal{M}_{i+1,j}$ at an intersection of the manifold $\mathcal{M}$ and the half-plane $\mathcal{H}_{i,j}$ located a distance Δ_i from $\mathcal{M}_{i,j}$. The aforementioned half-plane is defined by the coordinate $\mathbf{x}_{i,j}$, the unit normal $\mathbf{t}_{i,j}$ and the guidance vector $\boldsymbol{\rho}_{i,j}$ (cf. equations (3.13) and (3.14)). Note that this intersection may occur anywhere on the half-circle within $\mathcal{H}_{i,j}$ of radius Δ_i, centered at $\mathbf{x}_{i,j}$. The search for a new mesh point is conducted by defining an aim point $\mathbf{x}_{\text{aim}}$ within $\mathcal{H}_{i,j}$, computed by performing a single,

classical, 4^{th}-order Runge-Kutta step (cf. table 2.3) of length Δ_i in the vector field locally defined as

$$\boldsymbol{\psi}(\mathbf{x}) = \frac{\boldsymbol{\xi}_3(\mathbf{x}) \times \mathbf{t}_{i,j}}{\left\| \boldsymbol{\xi}_3(\mathbf{x}) \times \mathbf{t}_{i,j} \right\|}, \tag{3.15}$$

starting at $\mathbf{x}_{i,j}$. Moreover, all of the vectors of the intermediary Runge-Kutta evaluations of $\boldsymbol{\psi}(\mathbf{x})$ were corrected, if necessary, by continuous comparison with $\boldsymbol{\rho}_{i,j}$ and sign-reversion if an intermediary vector was directed radially inwards. Finally, the computed aim point was projected into the half-plane $\mathcal{H}_{i,j}$ as follows:

$$\mathbf{x}_{\text{aim}} := \mathbf{x}_{\text{aim}} - \langle \mathbf{t}_{i,j}, \mathbf{x}_{\text{aim}} - \mathbf{x} \rangle \mathbf{t}_{i,j}. \tag{3.16}$$

The idea is then to look for a new position within $\mathcal{H}_{i,j}$, in the vicinity of $\mathbf{x}_{\text{aim}}$, at a distance Δ_i from $\mathbf{x}_{i,j}$, by moving within the constraints of the manifold. Motivated by remark 1 — on the invariance of the time-t_0 image of repelling LCSs under perturbations in the local $\boldsymbol{\xi}_1$- and $\boldsymbol{\xi}_2$-direction fields — this involves computing trajectories which everywhere l2ay within the plane spanned by the local $\boldsymbol{\xi}_1$- and $\boldsymbol{\xi}_2$-vectors. This was done by defining a local, normalized direction field as

$$\boldsymbol{\mu}(\mathbf{x}, \mathbf{x}_{\text{aim}}) = \frac{\mathbf{x}_{\text{aim}} - \mathbf{x} - \langle \boldsymbol{\xi}_3(\mathbf{x}), \mathbf{x}_{\text{aim}} - \mathbf{x} \rangle \boldsymbol{\xi}_3(\mathbf{x})}{\left\| \mathbf{x}_{\text{aim}} - \mathbf{x} - \langle \boldsymbol{\xi}_3(\mathbf{x}), \mathbf{x}_{\text{aim}} - \mathbf{x} \rangle \boldsymbol{\xi}_3(\mathbf{x}) \right\|}, \tag{3.17}$$

that is, the normalized projection of the vector separating $\mathbf{x}$ and $\mathbf{x}_{\text{aim}}$ into the plane orthogonal to the local $\boldsymbol{\xi}_3$ vector, in accordance with LCS existence criterion (2.28c). A visual representation of this direction field is given in figure 3.5. The choice of initial conditions for computing trajectories within the manifold, with a view to expanding it, is the topic of (the immediately forthcoming) section 3.6.1.

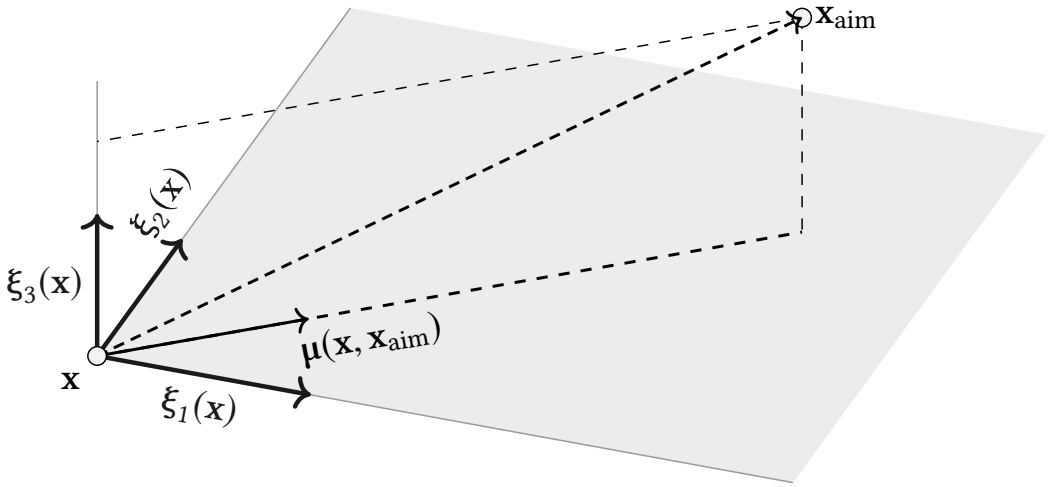


Figure 3.5: Visualization of the direction field used to compute trajectories within a manifold, using the legacy approach. The direction $\boldsymbol{\mu}(\mathbf{x}, \mathbf{x}_{\text{aim}})$ (see equation (3.17)) is found by normalizing the projection of $\mathbf{x}_{\text{aim}} - \mathbf{x}$ into the plane spanned by the local $\boldsymbol{\xi}_1$- and $\boldsymbol{\xi}_2$ vectors (shaded).

3.6.1 *Selecting initial conditions from which to compute new mesh points*

As suggested by Krauskopf, Osinga, et al. (2005), the starting point for all trajectories intended to reach $\mathbf{x}_{\text{aim}}$ could be chosen as any point not equal to $\mathbf{x}_{i,j}$ along the parametrized curve C_i. However, trajectories starting out at points along C_i which are far removed from $\mathbf{x}_{i,j}$ are likely to require a long integration path, which would result in an increase in the accumulated numerical error. Subject to this kind of error, these trajectories might not even get close to $\mathbf{x}_{\text{aim}}$ with a reasonable computational resource consumption. Accordingly, we limited the potential number of trajectories to compute by only considering a subset of the interpolation curve C_i as initial conditions, as follows:

$$\mathbf{x}_{\text{init}} = C_i(\check{s}), \quad \check{s} \in \big\{[s_j - \varsigma, s_j) \cup (s_j, s_j + \varsigma]\big\}, \quad 0 < \varsigma \leq \frac{1}{2}, \tag{3.18}$$

where the inherent periodicity of the (quasi-)arclength parametrization of C_i is implicitly applied. Here, ς was set to 0.1, ensuring that 20 % of all possible initial conditions along C_i were available for consideration.

For computing trajectories whose initial conditions are given by equation (3.18), and direction fields are given by equation (3.17), the Dormand-Prince 8(7) adaptive ODE solver (cf. table 2.4 and section 3.3.2) was the method of choice. In particular, the dynamic integration step size adjustment meant that we did not need to treat the integration step itself as a degree of freedom (of which there are many, as will be revealed shortly). However, in order to ensure that any trajectory did not overstep the half-plane $\mathcal{H}_{i,j}$ in passing, the step length was continuously limited from above by $\|\mathbf{x}_{\text{aim}} - \mathbf{x}\|$. Moreover, in order to avoid spending unreasonable computational resources on trajectories which for practical purposes never would result in acceptable, new mesh points, the total allowed integration arclength was limited by a scalar multiple γ_{arc} of the initial separation $\|\mathbf{x}_{\text{aim}} - \mathbf{x}_{\text{init}}\|$. In particular, this limitation meant that trajectories which ended in stable orbits around $\mathbf{x}_{\text{aim}}$ were not allowed to keep going indefinitely.

If any trajectory terminated in a point $\mathbf{x}_{\text{fin}}$ located in the half-plane $\mathcal{H}_{i,j}$ at a distance Δ_i from $\mathbf{x}_{i,j}$, then the new mesh point $\mathcal{M}_{i+1,j}$ was placed at $\mathbf{x}_{\text{fin}}$. Numerically, these conditions were implemented by means of tolerance parameters, as comparing floating-point numbers for equality is prone to numerical round-off error. More precisely, a point $\mathbf{x}$ was said to lay within $\mathcal{H}_{i,j}$ provided that

$$\boldsymbol{\eta} := \frac{\mathbf{x} - \mathbf{x}_{i,j}}{\big\|\mathbf{x} - \mathbf{x}_{i,j}\big\|}; \quad \big|\langle \boldsymbol{\eta}, \mathbf{t}_{i,j} \rangle\big| < \gamma_{\mathcal{H}}, \tag{3.19}$$

whereas

$$\left| \frac{\big\|\mathbf{x} - \mathbf{x}_{i,j}\big\|}{\Delta_i} - 1 \right| < \gamma_{\Delta} \tag{3.20}$$

sufficed for it to be flagged as laying a distance Δ_i from $\mathbf{x}_{i,j}$, with $\gamma_{\mathcal{H}}$ and γ_{Δ} chosen as small numbers. When a trajectory first intersected $\mathcal{H}_{i,j}$, the integration was stopped abrubtly, leaving its endpoint $\mathbf{x}_{\text{fin}}$ as its suggested coordinates for the new mesh point. As briefly mentioned in section 3.5.2, the unit tangent vectors $\mathbf{t}_{i,j}$ were generally inherited — the

treatment of special cases will be explained in greater detail in section 3.8.1. Figure 3.6 depicts a few characteristic trajectory patterns which commonly occured when searching for new mesh points in the fashion discussed in the above.

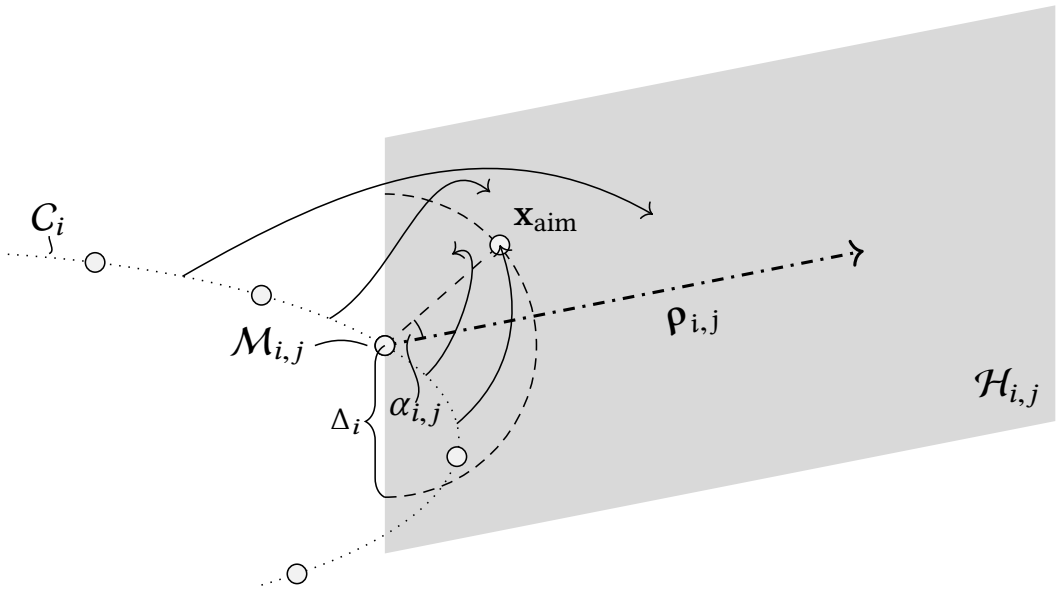


Figure 3.6: Visualization of typical trajectories used to compute a new mesh point, using the legacy approach. The aim point $\mathbf{x}_{\text{aim}}$ is used to guide trajectories which are locally orthogonal to the ξ_3 direction field (cf. equation (3.17)) towards the intersection between the manifold $\mathcal{M}$ and the half-plane $\mathcal{H}_{i,j}$ (shaded), in order to find a new mesh point $\mathcal{M}_{i+1,j}$ located a distance Δ_i from $\mathcal{M}_{i,j}$. $\mathcal{H}_{i,j}$ is defined by the point $\mathbf{x}_{i,j}$ (that is, the coordinates corresponding to the mesh point $\mathcal{M}_{i,j}$), its unit normal $\mathbf{t}_{i,j}$ (not shown) and the (quasi-)radial unit vector $\boldsymbol{\rho}_{i,j}$ (dashdotted). The half-circle of radius Δ_i, centered in $\mathbf{x}_{i,j}$ and laying within $\mathcal{H}_{i,j}$, on which we seek to find a new mesh point, is dashed. All permitted trajectory initial positions lie along the smooth parametrized curve C_i (dotted), and are given by equation (3.18). A select few trajectories are shown, where the arrowheads indicate the first intersection with $\mathcal{H}_{i,j}$ (as per equation (3.19)), at which point the integration was terminated.

3.6.2 *Choosing new trajectory start points by an algorithm with memory*

Our method of choosing parameter values $\check{s}$ corresponding to points along C_i (cf. equation (3.18)), from which to compute trajectories of the direction field given by equation (3.17) — with the intention of computing new mesh points — rests on the assumption that

$$\Delta(\check{s}) := \left\|\mathbf{x}_{\text{fin}}(\check{s}) - \mathbf{x}_{i,j}\right\|, \quad \mathbf{x}_{\text{fin}} \in \mathcal{H}_{i,j} \tag{3.21}$$

is a continuous function of $\check{s}$. To this end, we keep track of why each computed trajectory is terminated. In particular, we first note whether or not each trajectory, corresponding to a start point $\mathbf{x}(\check{s})$, ends up at some point $\mathbf{x}_{\text{fin}} \in \mathcal{H}_{i,j}$. If this is the case, we also note whether the corresponding separation $\Delta(\check{s})$ (defined in equation (3.21)) was an over- or undershoot with regards to the desired separation Δ_i.

Based on the premises outlined above, we then make use of the intermediate value theorem; specifically, if we have

$$\Delta(\check{s}_1) < \Delta_i, \quad \Delta(\check{s}_2) > \Delta_i \tag{3.22a}$$

for $\check{s}_1 < \check{s}_2$, then the intermediate value theorem implies that there must exist an $\check{s}$, such that

$$\Delta(\check{s}) = \Delta_i, \quad \check{s}_1 < \check{s} < \check{s}_2, \tag{3.22b}$$

under the assumption that $\Delta(\check{s})$ is a continuous function. In order to optimize our use of computational resources, we thus endeavor to take large steps when moving along C_i whenever the computed intersections with $\mathcal{H}_{i,j}$ are far from fulfilling $\Delta(\check{s}) = \Delta_i$. However, when a subinterval of C_i is identified, within which the intermediate value theorem suggests that a trajectory may fulfill our requirements, we decrease the (quasi-)arclength increment $\delta\check{s}$ in order to increase our odds of finding said trajectory. While the purpose of $\delta\check{s}_{\text{min}}$ was to manage resource requirements, $\delta\check{s}_{\text{max}}$ was used in order to avoid bypassing subsets of C_i from which two or more trajectories satisfy $\Delta(\check{s}) = \Delta_i$. Overstepping a region containing an even number of such intersections could render it undetectable using our algorithm (see figure 3.7), as no change in trajectory termination status need be detected.

The feedback received by tracking why each trajectory is terminated, allows us to dynamically select new trajectory start points along C_i. We do so by increasing the (quasi-)arclength increment $\delta\check{s}$ as long as there is no change in trajectory termination status, and, conversely, backtracking and reducing $\delta\check{s}$ when a status change is detected. This process is shown schematically in figure 3.7. As we assume asymptotic behaviour close to any regions in which $\Delta(\check{s})$ is not defined (that is, regions where no trajectories reach the half-plane $\mathcal{H}_{i,j}$, cf. equation (3.21)), the adjustment of $\delta\check{s}$ is treated in the same fashion there.

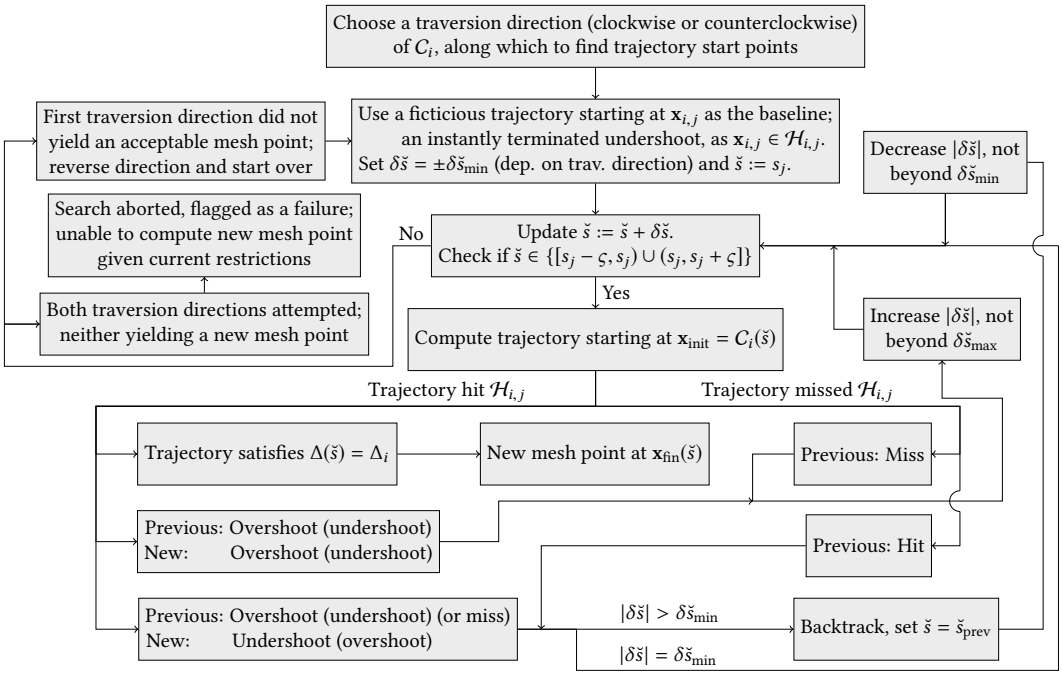


Figure 3.7: Flowchart illustrating the algorithm for iteratively choosing new trajectory start points based on the termination status of the preceding trajectories, using the legacy approach. All possible trajectory start points are contained within a subset of C_i, per equation (3.18). Whether or not a given trajectory intersected with the half-plane $\mathcal{H}_{i,j}$, and satisfied $\Delta(\check{s}) = \Delta_i$, was determined using equations (3.19) and (3.20).

3.6.3 *Handling failures to compute satisfactory mesh points*

As mentioned in section 3.6.1, it can never be guaranteed that any trajectories which start out at some point along the smooth curve C_i will be able to generate a new mesh point located within $\mathcal{H}_{i,j}$ which simultaneously satisfies all of our defined constraints. Missing out on points in any freshly generated level set prohibits the generation of more level sets — in particular, partly dependent on the number of successfully computed points, the smoothness of the interpolation curve C_{i+1} exhibits a critical dependence on *all* of the points used for its creation. Accordingly, handling tricky mesh points is crucial.

Although it remains impossible to *ensure* that the trajectory-based approach to computing new mesh points is successful, the success rate can be increased significantly by responding appropriately to an initially failed search. Our strategy of choice is based upon adjusting the computed aim point incrementally. Specifically, given the initial angular offset $\alpha_{i,j}$ of $\mathbf{x}_{\text{aim}}$ along the semicircle of radius Δ_i, with regards to the guidance vector $\boldsymbol{\rho}_{i,j}$ (see figure 3.6), we perturb $\mathbf{x}_{\text{aim}}$ along said semicircle by the forced alteration

$$\alpha_{i,j} := \alpha_{i,j} + \delta\alpha, \tag{3.23a}$$

with

$$|\delta\alpha| \leq \delta\alpha_{\max}. \tag{3.23b}$$

Note that the range of offsets — determined by $\delta\alpha_{\max}$ — should be chosen based on the expected geometry of the manifold as a whole; in contrast to the number of attempted angular offsets, which should be guided by considerations pertaining to the availability of computational resources.

Note that reattempting the iterative point search algorithm outlined in the entirety of the current section (that is, section 3.6) for all possible perturbation configurations of $\mathbf{x}_{\text{aim}}$ (given by equation (3.23)) does not *guarantee* that an acceptable mesh point is found. In an attempt to find particularly elusive mesh points, the entire algorithm — including angular perturbations as outlined in the above — was then rerun with simultaneously and progressively relaxed point acceptance criteria. That is, the numerical tolerance parameters $\gamma_{\mathcal{H}}$ and γ_{Δ} (cf. equations (3.19) and (3.20)) were gradually increased up to pre-set maximum values $\gamma_{\mathcal{H}}^{\max}$ and $\gamma_{\Delta}^{\max}$.

If, after having increased said tolerance parameters to their maximal permitted values, we were unable able to find an "acceptable" mesh point, the incomplete level set was promptly discarded, and attempted to be computed anew with Δ_i reduced to $\Delta_{\min}$ (more on the general adjustment procedure for Δ_i to follow in section 3.8.3). Should an entire, new geodesic level set not be computable even with the minimum permitted step length, attempts at expanding the computed manifold further were abandoned; constrained by the given (maximal) tolerance parameters, the method simply would not be able to expand the computed manifold any further. The various other stopping criteria for the generation of manifolds will be described in detail in section 3.10.

3.7 Revised approach to computing new mesh points

The version of the method of geodesic level sets presented by Krauskopf, Osinga, et al. (2005) is centered around computing manifolds defined by being everywhere tangent to some three-dimensional vector field (as a concrete example of application, Krauskopf, Osinga, et al. seek to compute the strange attractor of the Lorenz system). As noted by Oettinger and Haller (2016), repelling LCSs in 3D consist of subsets of manifolds defined by being everywhere orthgonal to the ξ_3 direction field — which is reflected in existence criterion (2.28c). Thus, compared to the type of systems considered by Krauskopf, Osinga, et al., we have an extra degree of freedom when seeking to identify repelling LCSs in 3D; in particular, moving along a manifold in arbitrary directions within a plane orthogonal to the local ξ_3 vector is allowed, in contrast to being constrained to moving back and forth along a curve. The paragraphs (and sections) to follow will reveal how we utilized the additional degree of freedom to reduce the number of calculations necessary to compute new mesh points.

The overarching principles, nevertheless, remain the same. Like for the legacy approach presented in section 3.6, mesh points in level set $\mathcal{M}_{i+1}$ are computed from the points in the prior level set $\mathcal{M}_i$. Similarly, the considerations to follow rely on each mesh point $\mathcal{M}_{i,j}$ having inherited its tangential vector from its direct ancestor; namely, $\mathbf{t}_{i,j} := \mathbf{t}_{i-1,j}$. How the special cases of this inheritance-based approach were treated, will be described in greater detail in section 3.8.1. From each mesh point $\mathcal{M}_{i,j}$, we wish to place a new mesh point $\mathcal{M}_{i+1,j}$ at an intersection of the manifold $\mathcal{M}$ and the half-plane $\mathcal{H}_{i,j}$ located a distance Δ_i from $\mathcal{M}_{i,j}$. As usual, $\mathcal{H}_{i,j}$ is defined by the coordinate $\mathbf{x}_{i,j}$, the unit normal $\mathbf{t}_{i,j}$ and the guidance vector $\boldsymbol{\rho}_{i,j}$ (where the latter of which is defined in equations (3.13) and (3.14)).

3.7.1 *Computing pseudoradial trajectories directly*

Just like in the legacy approach outlined in section 3.6, we define a local direction field as

$$\boldsymbol{\psi}(\mathbf{x}) = \frac{\boldsymbol{\xi}_3(\mathbf{x}) \times \mathbf{t}_{i,j}}{\left\| \boldsymbol{\xi}_3(\mathbf{x}) \times \mathbf{t}_{i,j} \right\|}, \tag{3.24}$$

where $\mathbf{t}_{i,j}$ is the unit tangent associated with the mesh point $\mathcal{M}_{i,j}$. Here, however, we make explicit use of our previously mentioned additional degree of freedom — namely that trajectories within the parametrized manifold are allowed arbitrary movements within the planes which are locally orthogonal to the ξ_3 direction field, rather than being constrained to moving along a three-dimensional curve — by computing the coordinates of the new mesh point $\mathcal{M}_{i+1,j}$ as the end point of a *single* trajectory.

Any trajectory starting out within the half-plane $\mathcal{H}_{i,j}$ and moving in the direction field given by equation (3.24) is certain to remain within the half-plane, as the direction field is everywhere orthogonal to its unit normal $\mathbf{t}_{i,j}$. That is, as long as the direction field used in computing said trajectory is everywhere oriented radially outwards. Accordingly, we computed a single trajectory in the aforementioned direction field, starting out at $\mathbf{x}_{\text{init}} = \mathbf{x}_{i,j}$, using the Dormand-Prince 8(7) adaptive ODE solver (see table 2.4 and section 3.3.2), where all of the vectors of the intermediary Runge-Kutta evaulations were corrected, if necessary,

by continuous comparison with $\boldsymbol{\rho}_{i,j}$ and direction-reversion if an intermediary vector was pointing radially inwards.

Should the ξ_3 direction be parallel to the unit tangent $\mathbf{t}_{i,j}$ locally along the trajectory, the direction field equation (3.24) would become undefined. In such regions, we allowed the Runge-Kutta solver to step in the direction used for the immediately preceding step. Numerically, such regions were recognized by

$$\left\|\xi_3(\mathbf{x}) \times \mathbf{t}_{i,j}\right\| < \gamma_{\|}, \tag{3.25}$$

where $\gamma_{\|}$ is a small tolerance parameter. Like for the legacy approach (outlined in section 3.6), the self-correcting integration step length meant that we did not treat the integration step as a degree of freedom, and, in order to avoid overstepping, the step length of the Dormand-Prince solver was continuously limited from above by $\Delta_i - \left\|\mathbf{x} - \mathbf{x}_{i,j}\right\|$. Moreover, the total allowed integration arclength was limited to an integer multiple γ_{arc} of the interset step Δ_i, allowing for the termination of any (hypothetical) trajectory which would end up in a stable orbit.

The trajectory integration was immediately interrupted upon reaching a point $\mathbf{x}_{\text{fin}}$ separated from $\mathbf{x}_{i,j}$ by a distance Δ_i. Like for the legacy approach, this criterion was checked by means of a tolerance parameter, seeing as directly comparing floating-point numbers for equality is prone to numerical round-off error. In particular, if a point $\mathbf{x}$ satisfied

$$\left|\frac{\left\|\mathbf{x} - \mathbf{x}_{i,j}\right\|}{\Delta_i} - 1\right| < \gamma_{\Delta}, \tag{3.26}$$

with γ_{Δ} being a small number, the point was flagged as laying a distance Δ_i from $\mathbf{x}_{i,j}$. Thus, upon reaching a point satisfying equation (3.26), the trajectory was terminated, and the new mesh point $\mathcal{M}_{i+1,j}$ was placed at the trajectory end point $\mathbf{x}_{\text{fin}}$. Figure 3.8 depicts a typical trajectory used to compute new mesh points in the fashion discussed in the above.

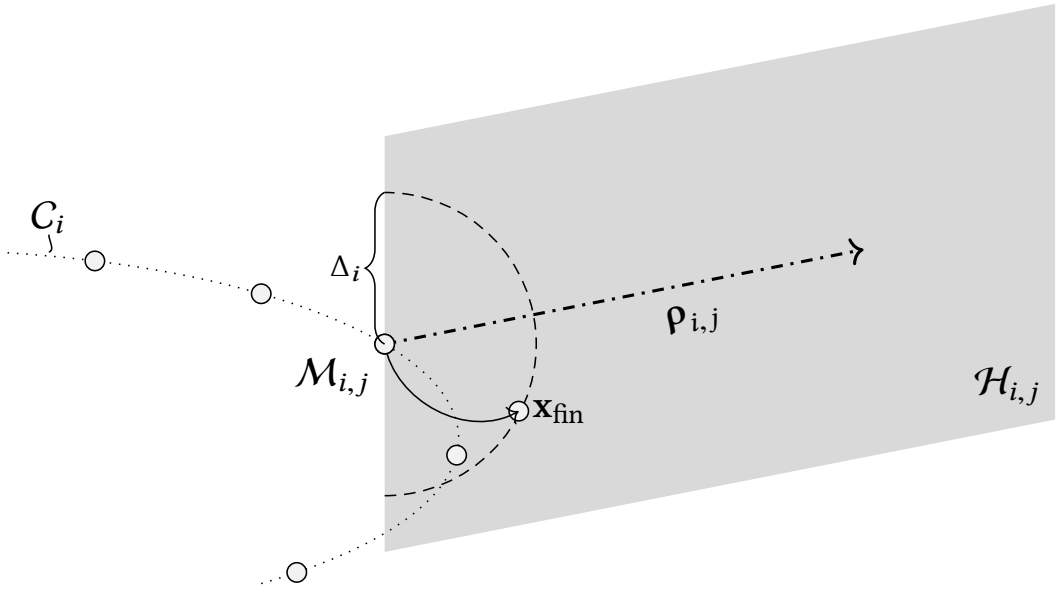


Figure 3.8: Visualization of a typical trajectory used to compute a new mesh point, using the revised approach. A trajectory is computed, starting at $\mathbf{x}_{i,j}$ (that is, the coordinates corresponding to the mesh point $\mathcal{M}_{i,j}$) and moving in the direction field given by equation (3.24), in order to find a new mesh point at the intersection between the manifold $\mathcal{M}$ and the half-plane $\mathcal{H}_{i,j}$ (shaded), located a distance Δ_i from $\mathcal{M}_{i,j}$. $\mathcal{H}_{i,j}$ is defined by the point $\mathbf{x}_{i,j}$, its unit normal $\mathbf{t}_{i,j}$ (not shown) and the (quasi-)radial unit vector $\boldsymbol{\rho}_{i,j}$ (dashdotted). The half-circle of radius Δ_i, centered in $\mathbf{x}_{i,j}$ and laying within $\mathcal{H}_{i,j}$, on which we seek to find a new mesh point, is dashed. As soon as a trajectory reaches a point separated from $\mathcal{M}_{i,j}$ by a distance Δ_i (per equation (3.26)), the integration is stopped, and a new mesh point $\mathcal{M}_{i+1,j}$ (not shown) is placed at the trajectory end point $\mathbf{x}_{\text{fin}}$.

3.7.2 *Handling failures to compute satisfactory mesh points*

Much like for the legacy approach outlined in section 3.6, it can never be guaranteed that all of the computed trajectories will yield new, acceptable mesh points — moreover, missing out on points in a level set prohibits the generation of further level sets. In particular, our procedure for maintaining mesh accuracy (which will be described in section 3.8.1) makes extensive use of the interpolation curves $\{C_i\}$. As the smoothness of the interpolation curve C_{i+1} depends on *all* of the points used for its creation, proper handling of tricky mesh points remains critically important.

Seeing as the only tolerance parameter involved in this method pertains to achieving the desired separation between a meshpoint and its direct descendant — see equation (3.26) — we elected not to progressively relax this constraint. Instead, we interpreted the failure of any trajectory to reach a point sufficiently far away, as the trajectory being coiled such that the required integration arc length to yield a point sufficiently far away from the ancestor mesh point, exceeded the maximum allowed integration path length (governed by γ_{arc}, cf. section 3.7.1). Accordingly, the incomplete level set was discarded, and attempted to be recomputed with Δ_i reduced to $\Delta_{\min}$ (our general dynamic adjustment procedure for Δ_i will be described in detail in section 3.8.3). Should an entire, new geodesic level set prove incomputable even at the minimum permitted step length, attempts at further expansion of

the computed manifold were abandoned. In such cases, this variant of the method of geodesic level sets simply did not suffice, for the given set of development parameters. Details on other stopping criteria for the generation of manifolds will be presented in section 3.10.

3.7.3 *Key improvements of the revised algorithm for computing new mesh points*

When compared to the convoluted way of computing new mesh points presented in section 3.6, it is readily apparent that the revised approach is significantly less complex. In particular, note how launching a single trajectory starting at the ancestor mesh point results in the legacy algorithm for computing new trajectory start points iteratively (cf. figure 3.7) being rendered entirely superfluous. Furthermore, as all computed trajectories necessarily remain within the target half-planes, no tolerance parameter for detecting intersections between trajectories and half-planes is needed. In short, the revised algorithm is conceptually simpler, and involves a lesser number of free (tolerance) parameters.

Simple numerical experiments confirmed that, given the same initial conditions and underlying direction fields, the two approaches yielded the same mesh points — at least, within rounding error. Moreover, these tests also revealed that the revised algorithm is usually two orders of magnitude faster (in terms of computational runtime). Unsurprisingly, the main time expenditure of the legacy approach turned out to be the computation of elusive mesh points; often, several thousand computed trajectories were needed before a satisfactory new point was found. In particular, the probability of finding an acceptable mesh point depends sensitively on choosing an appropriate aim point — leading to significant slowdowns in regions wherein the underlying manifold behaves erratically.

As a consequence of its superior speed and simplicity, the revised approach of forced pseudoradial trajectories became our method of choice. Accordingly, all of our results (which will be presented in chapter 4) were generated with this method. Note that, while deemed inferior in the context of identifying repelling LCSs in three-dimensional flow, the legacy approach of guided trajectories remains a valid way of computing three-dimensional manifolds. Thus, it can be used as a baseline for computing other kinds of three-dimensional manifolds defined in a different manner than repelling LCSs — which remains beyond the scope of this project.

3.8 Managing mesh accuracy

As will be described extensively in section 3.9, the manifold $\mathcal{M}$ was extracted from its set of parametrization points $\{\mathcal{M}_{i,j}\}$ by means of linear interpolation. In order to keep the interpolation error in check, we sought to limit the distance separating neighboring mesh points by means of pre-set upper and lower boundaries — in the following denoted $\Delta_{\min}$ and $\Delta_{\max}$, respectively. Considering a set of mesh points as a holistic approximation of a manifold, the separations between the mesh points in each level set, combined with the interset step lengths $\{\Delta_i\}$, determine the overall accuracy. Our algorithmic approach to maintaining the overall mesh quality will be explained in the sections to follow.

3.8.1 *Maintaining mesh point density*

When expanding a manifold by computing new geodesic level sets, the distance separating neighboring mesh points within each subsequent level set generally increases. This is due to the mesh points being a parametrization of what is essentially an expanding topological circle, for which an increasing amount of sampling points are needed in order to maintain the point density. This concept is illustrated in figure 3.9. Thus, having successfully computed a new geodesic level set — that is, having found a descendant point $\mathcal{M}_{i+1,j}$ for each of the ancestor points $\{\mathcal{M}_{i,j}\}$ — by use of the method outlined in section 3.7, we then inspected all of the distances separating nearest neighbors.

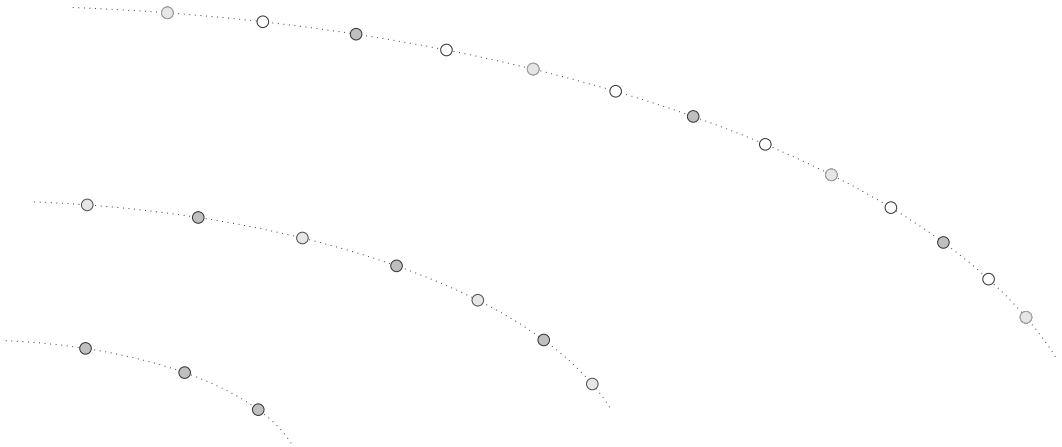
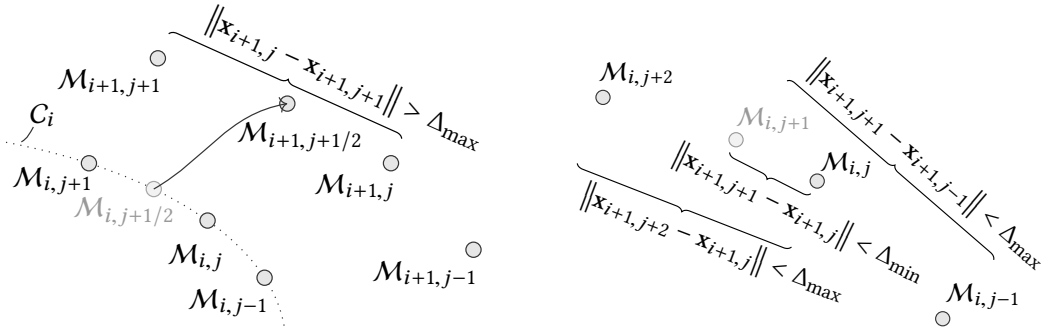


Figure 3.9: Conceptual illustration of the rationale behind the insertion of new mesh points as the geodesic level sets expand. The shown subset of the innermost topological circle is parametrized by three mesh points, shown as dark gray circles. As the topological circle is expanded, an increasing amount of mesh points must be inserted in order to maintain the (approximate) mesh point density. In the intermediate-sized circle shown in the figure, the mesh points with no direct analogue in the smallest circle are shown in a lighter shade of gray. Similarly, in the largest circle shown, the mesh points with no direct analogue in the intermediate-sized circle are drawn without fill.

Specifically, if any of the separations between nearest neighbors exceeded $\Delta_{\max}$, we sought to insert a new mesh point between them. That is, if $\left\|\mathbf{x}_{i+1,j} - \mathbf{x}_{i+1,j+1}\right\| > \Delta_{\max}$, a new mesh point $\mathcal{M}_{i+1,j+1/2}$ was computed using the method described in section 3.7, by launching a trajectory starting from a ficticious ancestor point $\mathcal{M}_{i,j+1/2}$, located midway inbetween $\mathcal{M}_{i,j}$ and $\mathcal{M}_{i,j+1}$ along the interpolation curve C_i. As the ficticious mesh point $\mathcal{M}_{i,j+1/2}$ does not itself have a direct ancestor from which to inherit a unit tangent, $\mathbf{t}_{i,j+1/2}$ was instead constructed by normalizing the arithmetic average of $\mathbf{t}_{i,j}$ and $\mathbf{t}_{i,j+1}$, and passed on to $\mathcal{M}_{i,j+1/2}$. The ficticious mesh point's guidance vector $\boldsymbol{\rho}_{i,j+1/2}$ was constructed in similar fashion (the guidance vectors' role in the dynamic update of the interset separation Δ_i will be described in detail in section 3.8.3). This way, the interpolation error is limited; no new mesh points are generated using interpolations (in intermediary computations) over intervals of length exceeding $\Delta_{\max}$.

Conversely, if any of the nearest neighbor separations became smaller than $\Delta_{\min}$, we sought to remove one of the points, as long as the distance separating mesh points which would

then become nearest neighbors did not exceed $\Delta_{\max}$. Accordingly, if any one of a pair of neighboring mesh points was to be removed, we chose to discard the one which would result in the smallest separation between the ensuing, *new* pair of nearest neighbors. In our experience, the removal of mesh points rarely occurred; however, it was crucial for generating the spherical LCS surface which will be presented in section 4.2.1. Our principles for inserting new mesh points inbetween others, and removing grid points which are too close together, are illustrated in figure 3.10.



(a) Inserting a new mesh point inbetween a pair of mesh points which are too far apart

(b) Removing a mesh point too close to another, if the ensuing separations are acceptable

Figure 3.10: Our approach to inserting new, or removing, mesh points to maintain mesh point density. When two neighboring mesh points in a freshly computed level set are too far apart with regards to the given mesh parameter $\Delta_{\max}$, we attempt to insert a new mesh point between them. As shown in (a), this is done by the method described in section 3.7, using a ficticious initial condition midway inbetween the two ancestor mesh points $\mathcal{M}_{i,j}$ and $\mathcal{M}_{i,j+1}$ along the interpolated curve C_i, indicated in the figure by a lighter shade of gray. Should two neighboring mesh points be too close together, and one of the two can be removed without the resulting sets of neighboring points being too far apart, we remove the one which results in the shortest interpoint separation — as shown in (b), where the point which is removed is indicated with a lighter shade of gray.

Having completed the process of inserting and removing mesh points, the points in the level set were assigned consecutive integer indices. Specifically, if a mesh point $\mathcal{M}_{i,j+1/2}$ was inserted inbetween $\mathcal{M}_{i,j}$ and $\mathcal{M}_{i,j+1}$, the respective indices were updated as $\mathcal{M}_{i,j+1/2} \to \mathcal{M}_{i,j+1}$, $\mathcal{M}_{i,j+1} \to \mathcal{M}_{i,j+2}$ and so on. Conversely, if mesh point $\mathcal{M}_{i,j+1}$ was removed, then $\mathcal{M}_{i,j+2} \to \mathcal{M}_{i,j+1}$, $\mathcal{M}_{i,j+3} \to \mathcal{M}_{i,j+2}$ and so forth.

3.8.2 *Limiting the accumulation of numerical noise*

Early tests with regards to the generation of manifolds revealed that the compound numerical error over the course of many level sets often resulted in irregular behaviour. Specifically, small bulges in the mesh easily became amplified in the subsequent level sets, which frequently caused unwanted loops in the interpolation curves $\{C_i\}$ which extended far from the manifold epicentre (namely $\mathbf{x}_0$, cf. section 3.5.1). Occasionally, this would lead the generated manifold to fold into itself. Per their definition, this is never permitted for repelling LCSs. In particular, self-intersecting manifolds indicate that, in a small neighborhood of any

intersection, there would not be a well-defined direction of strongest repulsion, which would violate LCS existence criterion (2.28a).

As a countermeasure against the formation of undesired loops within a level set, we reviewed a recently computed geodesic level set, as follows: Consider a point $\mathcal{M}_{i+1,j}$ located at $\mathbf{x}_{i+1,j}$. If, for any point $\mathcal{M}_{i+1,j+k}$ with $k > 1$, the separation between the mesh points $\mathcal{M}_{i+1,j}$ and $\mathcal{M}_{i+1,j+k}$ satisfies the pre-set bounds for the mesh point density (cf. section 3.8.1), and is significantly smaller than the cumulative nearest neighbor separations in the mesh point sequence $\{\mathcal{M}_{i+1,j+\kappa}\}_{\kappa=0}^{k}$, we would remove the intermediate mesh points. In more mathematical terms; if the interpoint distances satisfy

$$\Delta_{\min} < \left\|\mathbf{x}_{i+1,j+k} - \mathbf{x}_{i+1,j}\right\| < \Delta_{\max} \tag{3.27a}$$

and

$$\left\|\mathbf{x}_{i+1,j+k} - \mathbf{x}_{i+1,j}\right\| < \gamma_{\cup} \sum_{\kappa=0}^{k-1} \left\|\mathbf{x}_{i+1,j+\kappa+1} - \mathbf{x}_{i+1,j+\kappa}\right\|, \tag{3.27b}$$

then all mesh points $\{\mathcal{M}_{i+1,j+\kappa}\}_{\kappa=1}^{k-1}$ are removed. Here, $0 \leq \gamma_{\cup} \leq 1$ is a bulge tolerance parameter, which essentially determines an upper limit for the extent (measured in cumulative arclength) of loop-like segments of any given interpolation curve C_i. Specifically, a large $\gamma_{\cup}$ facilitates removal of rounded loops, whereas a small $\gamma_{\cup}$ restricts the removal process to sharp bulges.

A characteristic example demonstrating this method of removing unwanted loops is shown in figure 3.11. While possibly sacrificing some resolution of the manifold as a whole, adhering to criterion (3.27) prevents the removal of any *reasonable* bulge formations. The removal of several consecutive mesh points in the manner outlined may cause some triangle elements in the reconstruction of manifold surfaces from the resulting point meshes (see section 3.9) to be somewhat larger than their neighbors, and, in some cases, some triangles may partly overlap. However, as no new mesh points are *added* as a direct consequence of this loop-removal algorithm, it does not introduce new errors. In our experience, the removal of loop-forming mesh points was rarely required.

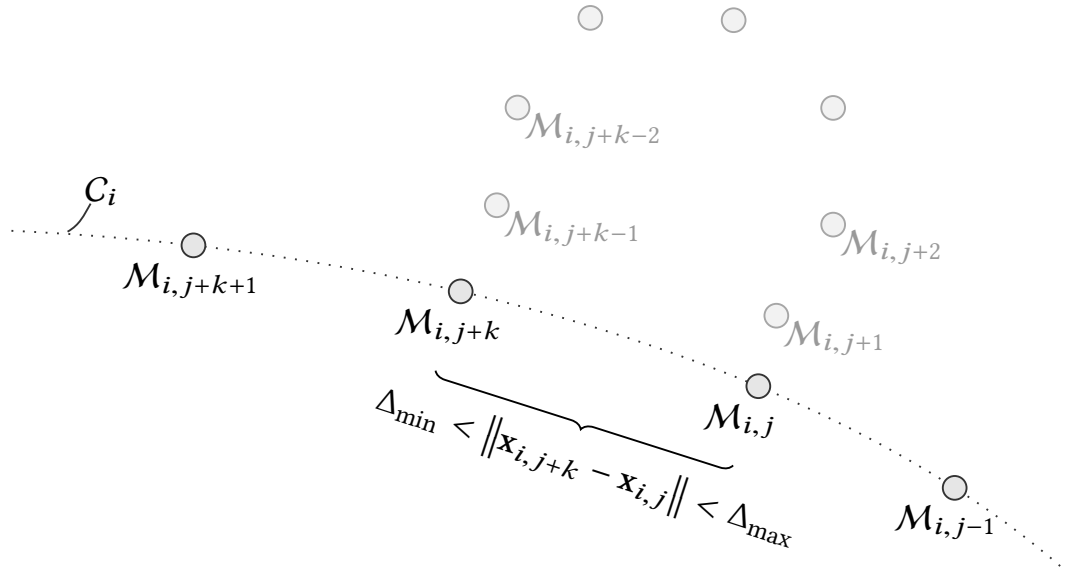


Figure 3.11: Our approach to limit the compound numerical error, by continuously removing unwanted loops in the computed level sets. As the separation between the mesh points $\mathcal{M}_{i,j}$ and $\mathcal{M}_{i,j+k}$ satisfies the restrictions regarding mesh point density (cf. section 3.8.1), the mesh points $\{\mathcal{M}_{i,j+\kappa}\}_{\kappa=1}^{k-1}$ (shown in a lighter shade of gray) are removed, provided that the cumulative neighbor separation around the bulge is sufficiently large (per the conditions presented in equation (3.27)) — which is the case in the above. The resulting interpolation curve $\mathcal{C}_i$ becomes significantly more well-behaved without the superfluous mesh points.

3.8.3 *A curvature-based approach to determining interset separations*

Given the interpoint separation constraints described in section 3.8.1, we have some flexibility regarding the choice of interset step length Δ_i. In an approach closely mirroring that of Krauskopf, Osinga, et al. (2005), we used the (approximate) local curvatures along each point strand (i.e., the set of mesh points which can be traced back to a common ancestor) in order to determine whether or not the local manifold dynamics were resolved to a satisfactory level of detail. Starting out with an initial interset separation $\Delta_1 = 2\Delta_{\min}$ for the second innermost level set (i.e., the first level set which was computed using the method described in section 3.7), we sought to ensure that the subsequent Δ_i resulted in the encapsulation of the finer details of the growing manifolds.

Specifically, once all mesh points which constitute a geodesic level set $\mathcal{M}_{i+1}$ have been identified, all a distance Δ_i away from their direct ancestor points in the previous level set $\mathcal{M}_i$, we computed the angular offsets $\alpha_{i,j}$ between each pair of guidance vectors $\boldsymbol{\rho}_{i,j}$ and $\boldsymbol{\rho}_{i+1,j}$ (as defined in equations (3.13) and (3.14)). This is sketched in figure 3.12. Note, however, that angular offsets for mesh points computed from ficticious ancestors in order to maintain the mesh point density (see section 3.8.1), were not computed. If

$$\alpha_{i,j} > \alpha_{\downarrow} \quad \text{or} \quad \Delta_i \cdot \alpha_{i,j} > (\Delta\alpha)_{\downarrow} \quad \text{for at least one } j, \tag{3.28}$$

where $\alpha_{\downarrow}$ and $(\Delta\alpha)_{\downarrow}$ are upper curvature tolerance parameters, was satisfied, the level set $\mathcal{M}_{i+1}$ was discarded and recomputed with reduced Δ_i. However, Δ_i was never reduced below

$\Delta_{\min}$. Conversely, if

$$\alpha_{i,j} < \alpha_{\uparrow} \quad \text{and} \quad \Delta_i \cdot \alpha_{i,j} < (\Delta\alpha)_{\uparrow} \quad \text{for all } j, \tag{3.29}$$

where $\alpha_{\uparrow}$ and $(\Delta\alpha)_{\uparrow}$ are lower curvature tolerance parameters, was satisfied, the interset distance for computing the *next* level set, Δ_{i+1}, was made bigger than Δ_i (although never beyond the pre-set $\Delta_{\max}$).

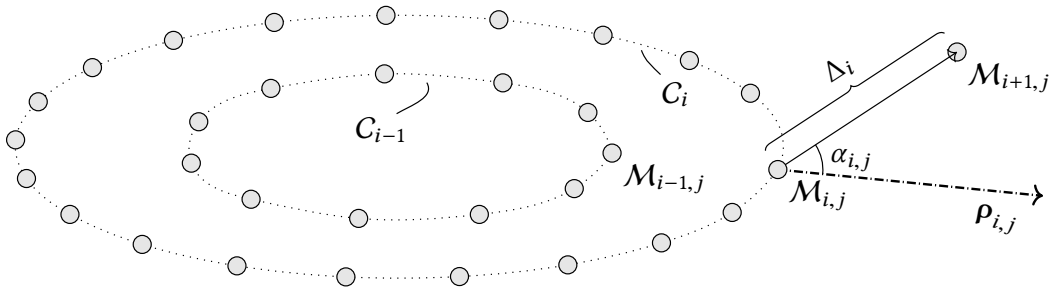


Figure 3.12: The principles of curvature-guided interset step length adjustment. For each of the mesh points $\{\mathcal{M}_{i+1,j}\}$ constituting the level set $\mathcal{M}_{i+1}$ which were computed from the mesh points constituting the most recently completed level set $\mathcal{M}_i$ (i.e., all but the mesh points which were computed from ficticious ancestors, cf. section 3.8.1), the angular offsets $\alpha_{i,j}$ between the guidance vectors $\boldsymbol{\rho}_{i,j}$ (dashdotted) and $\boldsymbol{\rho}_{i+1,j}$ (not shown, but parallel to the vector separating $\mathcal{M}_{i,j}$ and $\mathcal{M}_{i+1,j}$, shown in solid) were computed. These were used in conjunction with Δ_i, the interset step used to compute the new mesh points, in order to determine whether or not the suggested level set would have to be discarded due to the local curvature along at least one point strand being sufficiently large to comply with criterion (3.28). If the level set was deemed acceptable, local curvature estimates $\{\alpha_{i,j}\}$ and $\{\Delta_i \cdot \alpha_{i,j}\}$ were then used to determine if the *subsequent* level set $\mathcal{M}_{i+2}$ could be computed using $\Delta_{i+1} > \Delta_i$; namely, if criterion (3.29) was satisfied.

As is evident from equations (3.28) and (3.29), the parameters $\alpha_{\downarrow}$, $\alpha_{\uparrow}$, $(\Delta\alpha)_{\downarrow}$ and $(\Delta\alpha)_{\uparrow}$ determine the mesh adaption along point strands. The bounds for the offsets $\{\Delta_i \cdot \alpha_{i,j}\}$ enforce stricter requirements on angular offsets for large interstep lengths. Conversely, level sets computed using small interset lengths are generally allowed to exhibit (comparatively) larger angular offsets. In our experience, the interset step lengths were rarely *increased* — typically, there would be one or more subsets of the mesh points constituting any given level set which underwent sufficient curvature such that condition (3.29) did not hold.

3.9 Continuously reconstructing manifold surfaces from point meshes in 3D

While expanding the point mesh parametrization of a manifold $\mathcal{M}$ in bundles of mesh points constituting geodesic level sets, we attempted to reproduce its fully three-dimensional structure by simultaneously interpolating inbetween the mesh points. Our main objective for representing manifolds as continuous interpolation objects is visual representation (in addition to the detection of self-intersections, as will be outlined in section 3.10.1) — accordingly, we do not provide any form of analytical expression for $\mathcal{M}$'s surface. Moreover, as high order interpolation schemes are complicated considerably by the irregular structure

inherent to the parametrization of $\mathcal{M}$ as a sequence of level sets, linear interpolation became our method of choice.

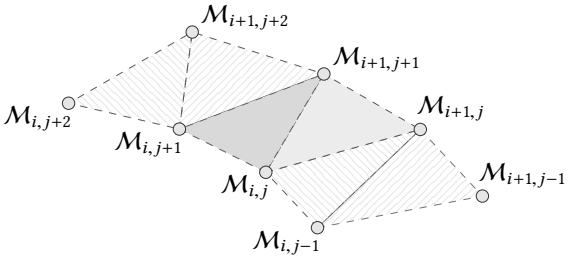
To our knowledge, all three-dimensional plotting algorithms rely on some sort of triangulation method to regularize a pointwise parametrized surface. General-purpose routines for triangulation generation were found to be unsuitable, due to the specific mesh structure of $\mathcal{M}$, arising from the parametrization by geodesic level sets. For instance, Delaunay triangulation not only resulted in omitting triangles which, to the naked eye, were crucial for the overall manifold structure, but also generated a lot of undesirable surface triangles — a problem which became increasingly prominent near creases. Accordingly, we made use of the fact that the `plot_trisurf` routine from the Python plotting library `Matplotlib` accepts an optional input argument specifying a list of triangles to plot, given by their vertices, and made our own triangulation scheme based on the specific structure of the mesh point parametrization of the computed manifolds.

We begin by fixing a traversion direction, specifying the order in which the triangles are created. Starting with the innermost level set, we then specify the vertices for a set of triangles which together cover the (approximate) surface between the manifold epicentre $\mathbf{x}_0$ and C_1 (see figure 3.4). When a new geodesic level set $\mathcal{M}_{i+1}$ satisfies the accuracy constraints outlined in section 3.8, we move along the mesh points constituting C_{i+1} in the selected direction, adding new triangles covering the (approximate) surface area between the interpolation curves C_i and C_{i+1}.

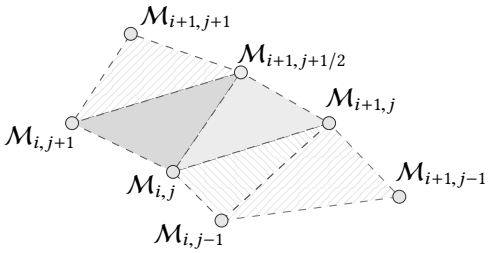
The surface area enclosed by the innermost level set was simply reconstructed by forming the triangles whose vertices are given as $\{\mathbf{x}_0, \mathbf{x}_{1,j}, \mathbf{x}_{1,j+1}\}$. Our treatment of the ensuing level sets is best described in terms of the local triangles formed around a single mesh point $\mathcal{M}_{i,j}$. The base case — that is, when no (nearby) points in the level set $\mathcal{M}_{i+1}$ have been removed nor added inbetween direct descendants in order to maintain the overall mesh point quality (cf. sections 3.8.1 and 3.8.2) — is handled by demanding that the triangles associated with $\mathcal{M}_{i,j}$ cover the tetragonal surface element with vertices given by $\{\mathbf{x}_{i,j}, \mathbf{x}_{i,j+1}, \mathbf{x}_{i+1,j+1}, \mathbf{x}_{i+1,j}\}$. Accordingly, we add two triangles with vertices given by $\{\mathbf{x}_{i,j}, \mathbf{x}_{i+1,j}, \mathbf{x}_{i+1,j+1}\}$ and $\{\mathbf{x}_{i,j}, \mathbf{x}_{i,j+1}, \mathbf{x}_{i+1,j}\}$, respectively. This is shown in figure 3.13a.

The cases not involving a one-to-one correspondence between the mesh points in $\mathcal{M}_i$ and $\mathcal{M}_{i+1}$ require special attention. This occurs whenever mesh points are inserted by making use of ficticious ancestors, or when mesh points are removed, in order to maintain the mesh point density (cf. section 3.8.1) — alternatively, when removing unwanted bulges in order to dampen the effects of compound numerical noise (as described in section 3.8.2). The treatment of these special cases is shown in figures 3.13b and 3.13c, and will be outlined in greater detail in the upcoming paragraph. In particular, note how all of $\mathcal{M}_{i,j}$'s nearest neighbors in the surrounding level set $\mathcal{M}_{i+1}$ are used in the triangulations — regardless of whether these are computed from the mesh points in level set $\mathcal{M}_i$, or have ficticious ancestors. This ensures (approximate) coverage of the entire surface area inbetween each level set, and thus the computed manifold as a whole.

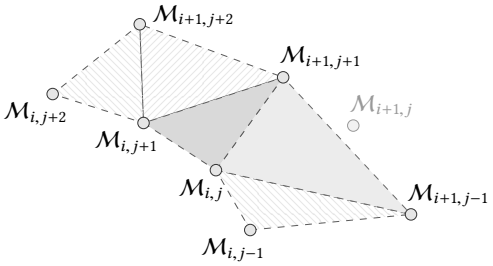
When an extra mesh point $\mathcal{M}_{i+1,j+1/2}$ is inserted inbetween $\mathcal{M}_{i+1,j}$ and $\mathcal{M}_{i+1,j+1}$, the tetragonal surface element whose vertices are located at $\{\mathbf{x}_{i,j}, \mathbf{x}_{i,j+1}, \mathbf{x}_{i+1,j}, \mathbf{x}_{i+1,j+1/2}\}$ is approximated by means of two triangles. Again expressed in terms of their vertices, these are $\{\mathbf{x}_{i,j}, \mathbf{x}_{i+1,j}, \mathbf{x}_{i+1,j+1/2}\}$ and $\{\mathbf{x}_{i,j}, \mathbf{x}_{i,j+1}, \mathbf{x}_{i+1,j+1/2}\}$. In the event that the mesh point $\mathcal{M}_{i+1,j}$ was removed, either as part of an undesired bulge or in order to preserve mesh density, the tetragonal surface with vertices at $\{\mathbf{x}_{i,j}, \mathbf{x}_{i,j+1}, \mathbf{x}_{i+1,j-1}, \mathbf{x}_{i+1,j+1}\}$ is constructed using two triangles, with vertices at $\{\mathbf{x}_{i,j}, \mathbf{x}_{i+1,j-1}, \mathbf{x}_{i+1,j+1}\}$ and $\{\mathbf{x}_{i,j}, \mathbf{x}_{i,j+1}, \mathbf{x}_{i+1,j+1}\}$, respectively. If more than one intermediate mesh point is removed, the treatment is completely analogous, occasionally resulting in some triangle elements being significantly larger than their neighbors.



(a) The base case for our triangulation scheme



(b) Including mesh points with ficticious ancestors in the triangulation



(c) Triangulating with the resulting nearest neighbors, if a mesh point is removed

Figure 3.13: Conceptual illustrations of our triangulation algorithm. The standard approach, which applies when there is a one-to-one correspondence between the nearest neighbors of the point $\mathcal{M}_{i,j}$ in the level set $\mathcal{M}_i$, and the and $\mathcal{M}_{i,j}$'s nearest neighbors in the level set $\mathcal{M}_{i+1}$, is shown in (a). Meanwhile, (b) and (c) illustrate our handling of the special cases which arise when the one-to-one correspondence between points in subsequent level sets is broken. In particular, (b) shows how we include a mesh point inserted inbetween the points $\mathcal{M}_{i+1,j}$ and $\mathcal{M}_{i+1,j+1}$ in the triangulation, while (c) demonstrates how the removal of mesh point $\mathcal{M}_{i+1,j}$ affects the triangulation. Both of these kinds of adjustments were made in order to maintain the density of mesh points (cf. section 3.8.1). In all of the illustrations, the pair of triangles which arise when triangulating outwards from $\mathcal{M}_{i,j}$, are shaded. The remaining triangles (patterned) originate from the triangulation of other points in level set $\mathcal{M}_i$. Note that the removal of bundles of mesh points in order to dampen the effect of numerical noise (which introduces bulges, as described in section 3.8.2) is treated analogously to the case of single missing points.

3.10 Macroscale stopping criteria for the expansion of computed manifolds

In principle, the process of developing manifold approximations by adding more and more mesh points, organized in geodesic level sets, would continue as long as the overall mesh quality was conserved (cf. section 3.8). The enforced (pseudo-)uniform expansion (quasi-)radially outwards, inherent to our take on the method of geodesic level sets, would then yield a mesh providing a conservative estimate of the extent of the *actual* manifold, as there is no particular reason to expect the manifold to appear homogeneous, when viewed from the epicentre of the computed level sets (i.e. the initial position $\mathbf{x}_0$, cf. section 3.5.1). With reasonable parameter choices, we became able to generate large manifolds quite quickly. This lead us to enforce two additional types of stopping criteria, based on what would happen if the computed manifolds reached the edges of the computational domain, or folded into themselves — where the latter of the two will be described in section 3.10.1.

The trajectories which are computed in order to identify new mesh points (per the method described in section 3.7) frequently overstep the domains within which the Cauchy-Green strain eigenvalues and -vectors are defined in order to compute mesh points on or near the domain boundaries. Thus, in order to resolve the behaviour of manifolds near the edges of the domain of interest, the aforementioned strain characteristics need to be computed in a region which *contains* the domain of interest, expanding beyond it in all directions. This is how we treated the case of tidal flow in the Førde fjord (as outlined in section 3.2). For perfectly periodic flow systems, such as (either variant of) the ABC flow — described in section 3.1 — this reduces to the trivial exercise of utilizing the inherent periodicity (that is, provided that the computational domain is sufficiently large to encompass at least one cycle along each direction).

3.10.1 *Continuous self-intersection checks*

Per LCS existence criterion (2.28a), there must be a uniquely defined direction of strongest repulsion everywhere along a repelling LCS. Furthermore, our method of expanding manifolds by adding mesh points organized in geodesic level sets (cf. section 3.7) is based on continuous expansions (quasi-)radially outwards from the manifold centre. Accordingly, the intersection of any manifold with itself was interpreted as a nonphysical artefact of accumulated numerical error. For that reason, we sought to terminate the expansion of a manifold when self-intersections were detected.

Our method of detecting manifold self-intersections is based on comparing the continuously computed interpolation triangles (as described in section 3.9) — in particular, we compared each triangle which was added with the most recently computed level set, to all of the triangles which had been added with the preceding level sets. If at least one pair of triangles intersected, the newest level set was flagged as self-intersecting. Our way of determining whether or not two triangles intersect, is based on the Möller-Trumbore ray-triangle intersection algorithm, with a detection sensitivity parameter $\epsilon = 10^{-8}$ (Möller and Trumbore 1997). A visual representation of our triangle-intersection detection algorithm is available in figure 3.14.

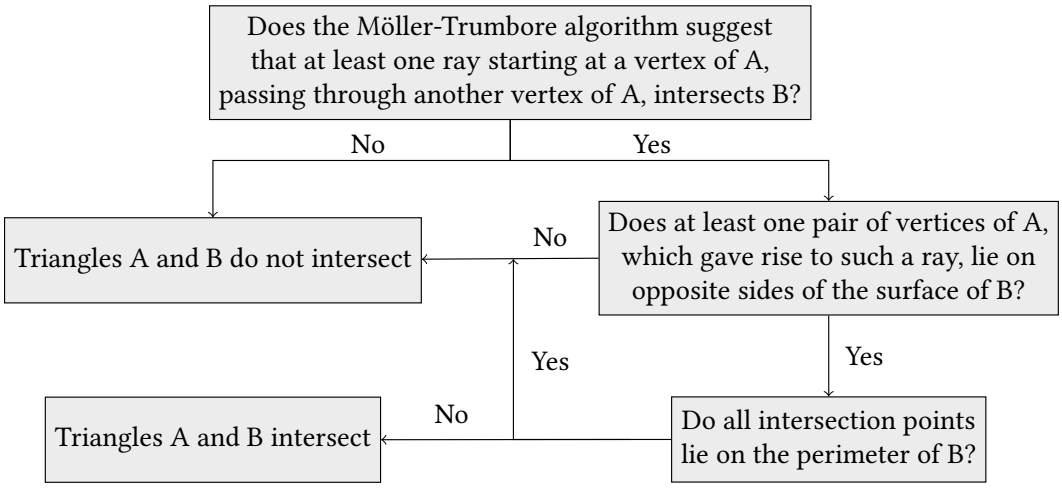
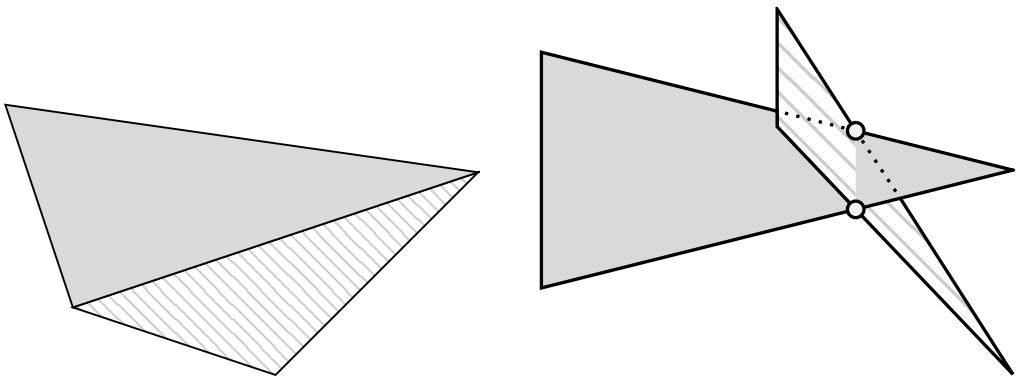


Figure 3.14: Flowchart illustrating the algorithm for detecting self-intersections. In order to ensure that no self-intersection went unnoticed, the indicated procedure was carried out by comparing each of the interpolation triangles A in the most recently computed level set, to all of the triangles B in the previously computed level sets. Provided that any of the new triangles intersected any of the old ones, the new level set as a whole was flagged as self-intersecting. How this intersection-detection approach handled a set of special cases is illustrated in figure 3.15.



(a) Two triangles sharing an edge, normally treated as a continuous intersection

(b) Two triangles whose edges intersect in exactly two unique points

Figure 3.15: How the intersection-detection algorithm handles special cases. Triangles which share a common edge, as illustrated in (a), form the premise of our triangulation algorithm; accordingly, this scenario does not get flagged as an intersection. Neither does the case when two triangles are identical (or one is contained within the other) — which is not shown here — nor the case when the edges of two triangles intersect in exactly two unique points, shown in (b). The latter is a very marginal case which hardly ever occurs, and, for our purposes, would invariably coincide with another (nearby) pair of triangles intersecting in a different manner than these special cases; ensuring that the computed level set as a whole would be flagged as self-intersecting.

Some cases of intersecting triangles warrant special treatment. In particular, as our triangulation method (outlined in section 3.9) is based on generating triangles which share sides with its neighbors, we decided to allow triangles to intersect along the edges. Similarly, we allowed for two triangles to be identical — which might happen if a manifold folds onto itself perfectly. Our algorithmic way of treating these cases resulted in a theoretical false negative; namely, the case of two triangles intersecting in exactly two points, laying along the edges of *both* triangles. In our experience, however, this never proved problematic — should one pair of triangles happen to intersect in this exact fashion (which is a rarity in itself due to numerical round-off errors), another pair of triangles would intersect in such a way that the most recently added set as a whole would be flagged as self-intersecting. Two of the aforementioned special cases — that is, two triangles sharing an edge, and two triangles intersecting in exactly two points laying along the edges of both triangles — are shown in figure 3.15.

Initial tests revealed that some self-intersections were quite innocuous, in that, if the computed manifold was expanded by an additional level set, the newly added triangulations need not necessarily intersect with any of the preceding ones. This kind of insipid intersection could be a consequence of the linear nature of our triangulation method, possibly compounded by round-off error. Accordingly, we decided to stop the manifold expansion process if *several consecutive* geodesic level sets introduced intersection triangulations. This was done by computing the sum of the interset distances $\{\Delta_i\}$ for each consecutive level set $\{\mathcal{M}_i\}$ which introduced new intersections. Whenever this pseudo-intersection length exceeded a scalar multiple $\gamma_\cap$ of $\Delta_{\min}$ (cf. section 3.8.1), we terminated the manifold computation process; empirical trials suggested that the intersection issue would then only worsen if more mesh points were added.

3.11 Identifying LCSs as subsets of computed manifolds

The collection of manifolds computed by means of the method outlined in the preceding sections, starting from an approximately even distribution of points in the $\mathcal{U}_0$ domain (cf. section 3.5 and, in particular, table 3.2), are all surfaces which satisfy LCS existence criterion (2.28c) — that is, they are everywhere perpendicular to the local direction of maximal repulsion. In order to extract repelling LCSs from these parametrized surfaces, we then identified the regions of the manifolds — represented as a subset of the mesh points in their parametrization — which also satisfy the remaining existence criteria; namely, (2.28a), (2.28b) and (2.28d). This was done completely analogously to how we identified the grid points belonging within the $\mathcal{U}_0$ domain, as outlined in section 3.5.1. In particular, each mesh point $\mathcal{M}_{i,j}$ of a computed manifold $\mathcal{M}$ was flagged as to whether or not it satsified all of the aforementioned existence criteria.

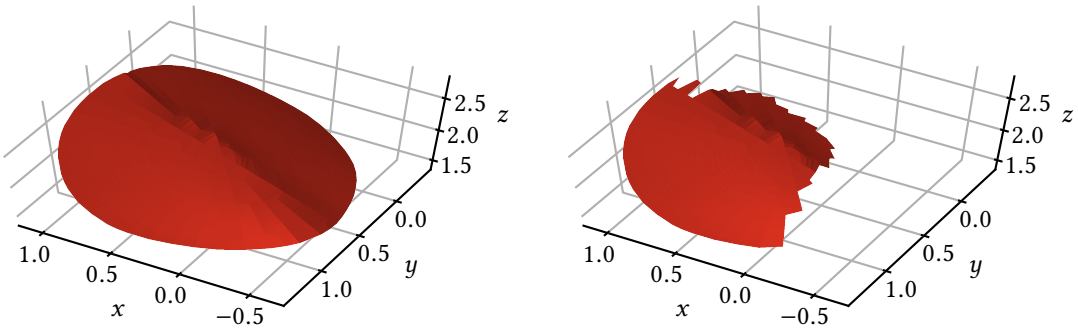
We then proceded to construct a repelling LCS $\mathcal{L}$ from the mesh points of $\mathcal{M}$. Per section 3.5.1, the mesh point at the centre of any given manifold always satisfies all the LCS existence criteria; accordingly, it was added as the first mesh point $\mathcal{L}_0$ of the extracted LCS. Going through the list of the remanining mesh points $\mathcal{M}_{i,j}$ which satisfied all LCS criteria, traversing each level set in the order in which their mesh points were added, we added a manifold mesh

point to the set of LCS points provided that

$$\left\|\mathbf{x}_{i,j} - \tilde{\mathbf{x}}_\kappa\right\| < \gamma_\square \Delta_{\max} \tag{3.30}$$

holds for at least one κ, where $\mathbf{x}_{i,j}$ and $\tilde{\mathbf{x}}_\kappa$ denote the coordinates of mesh point $\mathcal{M}_{i,j}$ and the already accepted LCS point $\mathcal{L}_\kappa \in \{\mathcal{L}_k\}$, respectively. $\Delta_{\max}$ is the maximum allowed mesh point separation used for computing the manifold (cf. section 3.8.1), while the scalar parameter $\gamma_\square \geq 1$ allows for extracting smoother LCSs, more well-suited for visualization purposes — as will be made clear shortly. Subsequently, the remaining mesh points of $\mathcal{M}$ (i.e., the ones which did *not* satisfy all of the LCS existence criteria) were added to the set of LCS points — provided that they comply with a similar distance threshold as given in equation (3.30); where we only computed mesh point separations relative to the LCS points which satisfied all of the existence criteria (thus not to any point added to the LCS purely for the purpose of enhanced visual representation).

The tolerance parameter $\gamma_\square$ thus allowed us to mitigate possible numerical error; in particular, if any given mesh point was slightly perturbed away from the underlying manifold, it could still end up being a part of the LCS. Finally, we looked at all of the surface elements pertaining to the triangulation of the manifold $\mathcal{M}$ (as described in section 3.9) in conjunction with the set of mesh points which had been recognized as belonging to the LCS $\mathcal{L}$. If mesh points corresponding to two of the three vertices defining a triangular surface element had been recognized as part of $\mathcal{L}$, we then added the mesh point corresponding to the last remaining vertex to the set of LCS points $\{\mathcal{L}_k\}$. Accordingly, the triangulations of the $\mathcal{M}$ were reused for $\mathcal{L}$. These slight relaxations of the LCS existence criteria facilitate the extraction of smoother LCS surfaces and are favorable for the visual representation of LCSs. Moreover, they mitigate the possible effects of numerical error perturbing any given mesh point $\mathcal{M}_{i,j}$ away from the *actual* manifold — leaving it more than sufficiently close to the manifold for visualization purposes — by possibly allowing it to be included as part of the LCS after all. Figure 3.16 shows an example of extracting a repelling LCS from a computed manifold.



(a) A computed manifold in its entirety **(b)** The extracted repelling LCS

Figure 3.16: An example of a repelling LCS extracted as a subset of a computed manifold. In (a), a sample manifold for the steady ABC flow is shown, whereas (b) shows the subset of the manifold which satisfies the LCS existence criteria given in equation (2.28) (or is sufficiently close to any point satisfying these criteria, meaning that their inclusion facilitates triangulations which provide an overall enhanced visual representation).

The extracted LCS surfaces $\mathcal{L}$, parametrized as a set of mesh points $\{\mathcal{L}_k\}$, represent three-dimensional surfaces which — allowing for a little numerical error — comply with all of the existence criteria for repelling LCSs given in equation (2.28), as originally proposed by Haller (2011). Inspired by the work of Farazmand and Haller (2012a), we then sought to dispose of the smallest among the computed LCSs, as these are expected to be the least significant in terms of influencing the overall flow within the system.

In order to obtain a measure of the size of our three-dimensional surfaces, to each LCS point $\mathcal{L}_k$, we assigned a weighting given by the surface area approximating the region of the underlying manifold $\mathcal{M}$ that is closer to the corresponding mesh point $\mathcal{M}_{i,j}$ than any others. To the mesh point located at the manifold epicentre $\mathbf{x}_0$, we assigned the weight $\mathcal{W}_0 = \pi(\delta_{\text{init}}/2)^2$. The weights of all other mesh points were computed as

$$\mathcal{W}_k := \mathcal{A}_{i,j} \approx \frac{\Delta_i + \Delta_{i-1}}{2} \cdot \frac{\left\|\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j}\right\| + \left\|\mathbf{x}_{i,j} - \mathbf{x}_{i,j-1}\right\|}{2}, \tag{3.31}$$

where, as always, $\mathbf{x}_{i,j}$ denotes the coordinates of mesh point $\mathcal{M}_{i,j}$. This surface approximation is illustrated in figure 3.17. These weights were also used to compute a repulsion average $\overline{\lambda}_3$; in particular,

$$\mathcal{W} = \sum_k \mathcal{W}_k, \quad \overline{\lambda}_3 = \frac{1}{\mathcal{W}} \sum_k \lambda_3(\tilde{\mathbf{x}}_k)\mathcal{W}_k, \tag{3.32}$$

where the summation is over all mesh points in the parametrization of $\mathcal{L}$, and $\tilde{\mathbf{x}}_k$ denotes the coordinates of mesh point $\mathcal{L}_k$.

To limit biasing from outlier mesh points (measured in terms of their λ_3 value) — due to spurious interpolation artefaacts — in the computed repulsion average, we alternately removed the least and most repulsive mesh points from the averaging process, as long as this altered the resulting repulsion average significantly. In particular, denoting the adjusted repulsion average by $\widehat{\lambda}_3$, we iteratively removed outlier extrema provided that

$$\left|1 - \frac{\widehat{\lambda}_3}{\overline{\lambda}_3}\right| > 0.1, \tag{3.33a}$$

whereupon we continuously accepted the most recently computed adjusted repulsion average as the new reference repulsion average — that is,

$$\overline{\lambda}_3 := \widehat{\lambda}_3. \tag{3.33b}$$

This process was repeated until the removal of the least or most strongly repelling mesh points did not alter the resulting repulsion average enough to trigger condition (3.33a).

Any LCS for which the computed total weight $\mathcal{W}$ (a measure of its surface area) was smaller than some pre-set limit $\mathcal{W}_{\text{min}}$ or $\overline{\lambda}_3 < 1$ — the latter as a sanity check to ensure overall repulsion — per existence criterion (2.28a), were discarded. As an aside, note that this method of identifying repelling LCSs can easily be adapted to the identification of *attracting* LCSs, by computing Cauchy-Green strain eigenvalues and -vectors for the *reversed* time interval $[t_1, t_0]$ (see definition 7) and otherwise proceeding as discussed.

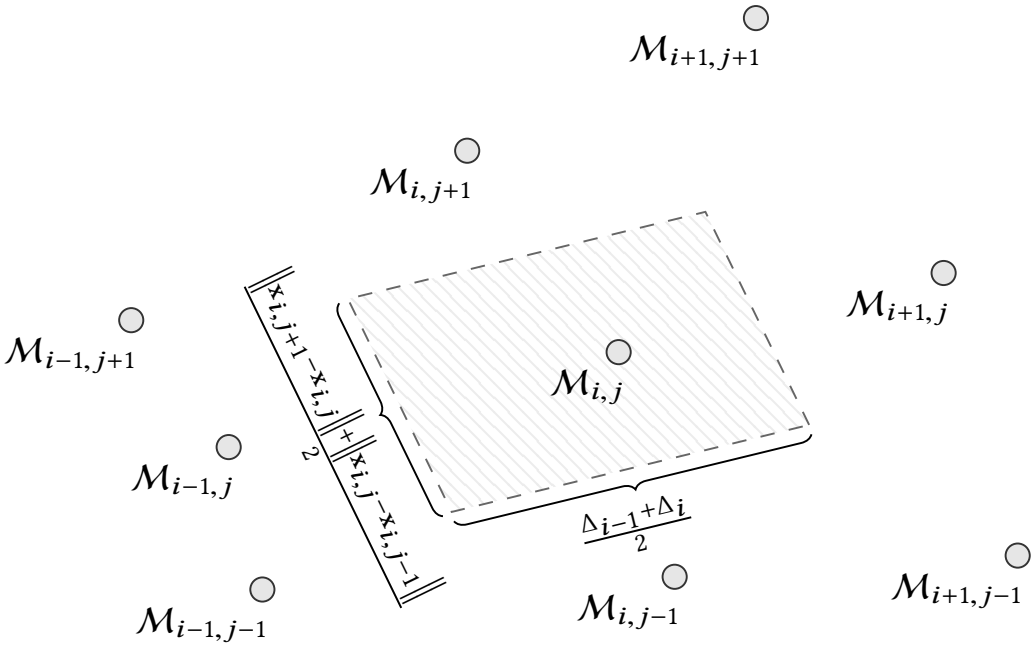


Figure 3.17: Our way of assigning weights to mesh points in computed LCSs. To a given LCS point $\mathcal{L}_k$, we assigned a weight given by a rectangular approximation of the surface area closer to the corresponding manifold mesh point $\mathcal{M}_{i,j}$ than all other points in the parametrization of the manifold $\mathcal{M}$ (patterned). Here, Δ_i corresponds to the interset step length used to create level set $\mathcal{M}_{i+1}$, based on level set $\mathcal{M}_i$, (see section 3.7), while $\mathbf{x}_{i,j}$ denotes the coordinates of mesh point $\mathcal{M}_{i,j}$.

3.12 Making the most of the available computational resources

The simultaneous solution of twelve coupled ODEs involved in the advection of tracers in order to computed the (directional deriatives of the) flow map (cf. section 3.3) quickly proved an unreasonably strenuous task for the author's own personal laptop. Seeing as the available memory was the main limitation, we parallelized this computation by means of MPI, and ran it on NTNU's supercomputer, Vilje. In spite of the tracers being independent, we elected to utilize MPI over alternative multiprocessing tools in order to access multiple nodes within the Vilje cluster. Due to the problem's pleasingly parallel nature, the parallelization process consisted of distributing an approximately even amount of tracers across all ranks, whereupon each rank advected (that is, simultaneously solved the twelve coupled ODEs for all of) its allocated tracers. In the end, all of the final state flow map Jacobians were collected by the designated main process (i.e., rank = 0), whereupon the Cauchy-Green strain eigenvalues and -vectors were extracted by means of a SVD decomposition (as outlined in section 3.4).

Regarding the generation of manifolds, code profiling (unsurprisingly) revealed that the generation of new mesh points by computing (quasi-)radial trajectories orthogonal to the ξ_3-direction field was a great source of time expenditure. Accordingly, we rewrote all numerical routines pertaining to the generation of new mesh points in Cython. In particular, we made

use of highly optimized, low-level BLAS[1] routines whenever possible. Because of the ever increasing number of necessary triangle comparisons in our algorithm of detectng manifolds which self-intersect (as described in section 3.10.1), all of the accompanying numerical methods were expressed in Cython. Similarly to how we exposed the Bspline-Fortran library to Python (cf. section 3.2.2), we also consistently used calls by reference in order to avoid unneccesary memory duplication. Overall, the transition from (NumPy-based) Python to Cython for the most significant tasks reduced the overall runtime by two orders of magnitude.

Analogously to the advection of tracers, we made use of the mutual independence of the computed manifolds to accelerate their computation by means of MPI parallelization across the Vilje cluster. We elected to make a one-to-one correspondence between the number of MPI threads and the number of manifolds to generate (as described in section 3.5.1) such that each manifold was allotted as much working memory as possible, facilitating each manifold to grow as large as possible before being stopped due to the one of the criteria proposed in section 3.10. Compared to the advection of tracers, or the expansion of manifolds, the extraction of repelling LCSs as subsets of the computed manifolds (cf. section 3.11) was not a particularly laborious task. Thus, we chose to parallelize this selection process by making use of the Python `multiprocessing` library, as access to a single node in the Vilje cluster sufficed to complete it in less than a minute.

[1] See www.netlib.org/blas and https://docs.scipy.org/doc/scipy/reference/linalg.cython_blas.html

4 Results

This chapter contains plots which illustrate some key properties of manifolds (and subsequently, repelling LCSs) computed using the method outlined in chapter 3. In section 4.1, we present geodesic level set approximations to a couple of three-dimensional surfaces, and how these were used in order to guide our parameter choices pertaining to the generation of mesh points in the general case. Section 4.2 describes how we verified our method of extracting *repelling* LCSs from the computed manifolds. Lastly, sections 4.3 and 4.4 contain the repelling LCSs we computed for flow in (either variant of) the ABC flow and the Førde fjord, respectively.

4.1 Verifying our method of generating manifolds

To start things off, section 4.1.1 outlines a series of geodesic level set approximations of an analytically known three-dimensional surface. From these, we extract key insight regarding which mesh point configurations might be reasonable for general application. In section 4.1.2, we illustrate how a sample manifold from the steady ABC flow (see section 3.1) depends on the mesh point parameters; which we then used to guide our parameter choices for the generation of manifolds in the general case. Lastly, we illustrate that the computed manifolds are in fact invariant for trajectories everywhere tangent to (arbitrary linear combinations of) the ξ_1- and ξ_2-direction fields (see remark 1) — and, consequently, everywhere orthogonal to the ξ_3-field, in compliance with existence criterion (2.28c).

4.1.1 *An analytical test case*

As a verification test case for (our variant of) the method of geodesic level sets to compute three-dimensional manifolds, we sought to reproduce a three-dimensional surface defined by

$$z = g(x, y) = A \sin(\omega_x x) \sin\big(\omega_y y\big) + z_0, \tag{4.1a}$$

which can be expressed as the zeros of the scalar function

$$f(\mathbf{x}) = g(x, y) - z, \tag{4.1b}$$

where $\mathbf{x}$ denotes the Cartesian coordinate vector (x, y, z). In order to do so, we computed its unit normal vector field as

$$\mathbf{n}(\mathbf{x}) = \frac{\boldsymbol{\nabla} f(\mathbf{x})}{\|\boldsymbol{\nabla} f(\mathbf{x})\|}, \tag{4.2}$$

which we then substituted for the ξ_3-direction field in equation (3.24). Here, we chose the parameters

$$A = 1, \quad \omega_x = \omega_y = 2, \quad z_0 = \pi \tag{4.3}$$

and used a single initial position $\mathbf{x}_0 = (\pi, \pi, \pi)$ from which to develop the surface, by subsequently adding mesh points organized in level sets, as outlined in sections 3.7–3.10.

Specifically, we computed a total of seven surface approximations, using the parameter values provided in table 4.1 with different values of $\Delta_{\min}$, thereby using different mesh

point densities. Four of the resulting manifolds are shown in figure 4.1. Although all of the presented manifolds successfully encapsulate the macroscale behaviour of the underlying surface (given by equations (4.1) and (4.3)), increasing the mesh point density clearly facilitates more accurate approximations. In particular, some of the visual discrepancies of figure 4.1 can be attributed to the linear interpolation inherent to our triangulation scheme (see section 3.9).

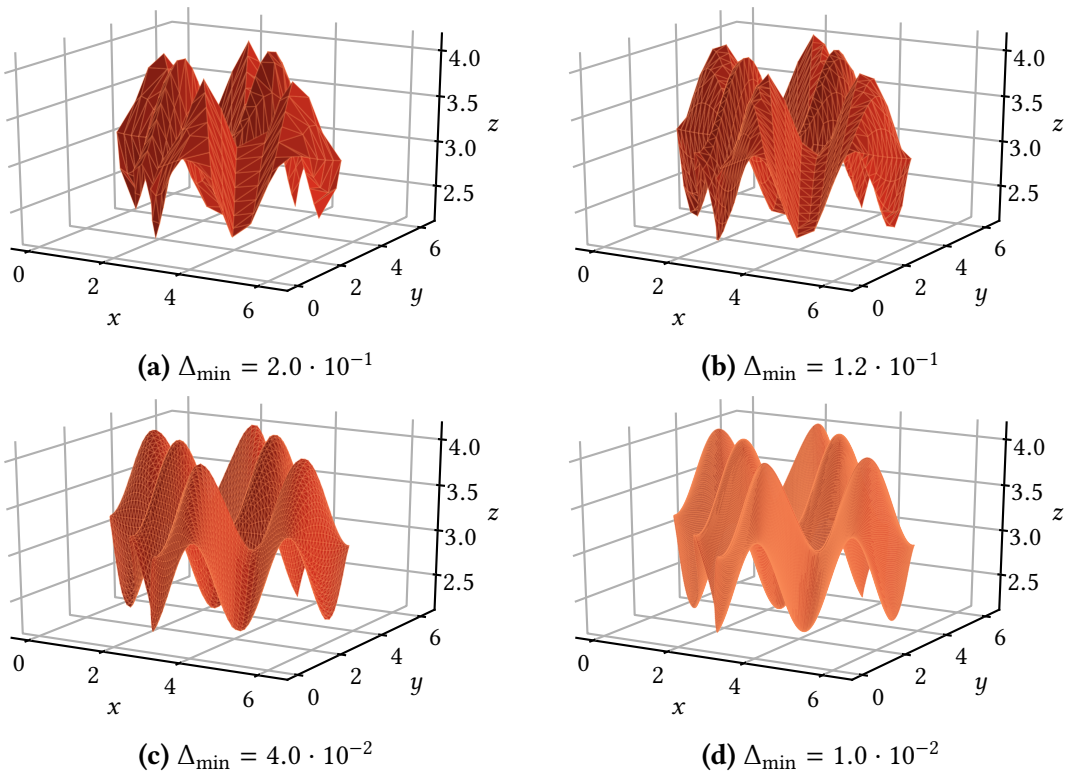
Table 4.1: Parameter values used to approximate an analytically known three-dimensional surface (see equations (4.1) and (4.3)) by the method of geodesic level sets. Note that the interset distances Δ_i were dynamically altered, as described in section 3.8.3. Accordingly, only the *first* interset distance, Δ_1, was set explicitly. Moreover, while we varied the overall mesh point density by altering Δ_{min} and Δ_{max}, we kept the ratio between them constant.

Parameter	Value	Description
δ_{init}	10^{-3}	Separation of innermost geodesic level set from the manifold epicentre, cf. figure 3.4
Δ_{min}, Δ_{max}	$\frac{\Delta_{\text{max}}}{\Delta_{\text{min}}} = 4$	(Variable) boundaries for interpoint separations (details found in section 3.8.1)
Δ_1	$2\Delta_{\text{min}}$	Interset distance used to compute the second geodesic level set (see sections 3.7 and 3.8.3)
$\gamma_{\parallel}$	10^{-4}	Tolerance for detecting regions in which ξ_3 is (anti-)parallel to $\mathbf{t}_{i,j}$ (see equation (3.25))
γ_{Δ}	$5 \cdot 10^{-3}$	Tolerance for the separation of a mesh point from its ancestor (per equation (3.26))
γ_{arc}	5	Sets an upper limit to trajectory lengths as $\gamma_{\text{arc}}\Delta_i$ (briefly mentioned in section 3.7.1)
$\gamma_{\cup}$	$7 \cdot 10^{-1}$	Sets an upper limit to the extent of loop-like segments of any level set (see section 3.8.2)
$\alpha_{\uparrow}$ $\alpha_{\downarrow}$ $(\delta\alpha)_{\uparrow}$, $(\delta\alpha)_{\downarrow}$	$8.7 \cdot 10^{-2}$ rad (5°) $4.4 \cdot 10^{-1}$ rad (25°) $2\delta_{\text{min}}\alpha_{\uparrow}$, $2\delta_{\text{min}}\alpha_{\downarrow}$	Used in a curvature-based approach to adjust interset distances (outlined in section 3.8.3)
$\gamma_{\cap}$	5	Used for terminating the expansion of self-intersecting manifolds (cf. section 3.10.1)

In order to obtain a quantitative measure of how an increase in the mesh point density impacts the accuracy of the overall approximation, we computed the root mean square (hereafter abbreviated to RMS) error of each mesh point as

$$\text{err}_{\text{RMS}} = \sqrt{\frac{1}{N}\sum_{i,j}\left|z_{i,j} - g(x_i, y_j)\right|^2}, \tag{4.4}$$

where we sum over all of the N computed points, with $g(x, y)$ given by equations (4.1) and (4.3). The RMS error is shown as a function of the mesh point density in figure 4.2. It appears to scale quadratically with Δ_{min}, the smallest permitted separation between neighboring mesh points — which doubles as a measurement of the overall mesh density.



(a) $\Delta_{\min} = 2.0 \cdot 10^{-1}$ **(b)** $\Delta_{\min} = 1.2 \cdot 10^{-1}$

(c) $\Delta_{\min} = 4.0 \cdot 10^{-2}$ **(d)** $\Delta_{\min} = 1.0 \cdot 10^{-2}$

Figure 4.1: Geodesic level set approximation to an analytically known three-dimensional surface. The underlying sinusoidal surface is given by equations (4.1) and (4.3). The manifolds were all computed using the parameters given in table 4.1, with varying mesh point densities (given by the value of $\Delta_{\min}$), where the mesh outlines are shown explicitly for clarity. Each of the approximations were grown from the initial position $\mathbf{x}_0 = (\pi, \pi, \pi)$. Note how all of the manifolds are able to capture the macroscale behaviour of the underlying surface, although the microscale behaviour is increasingly well-resolved when the mesh point density increases.

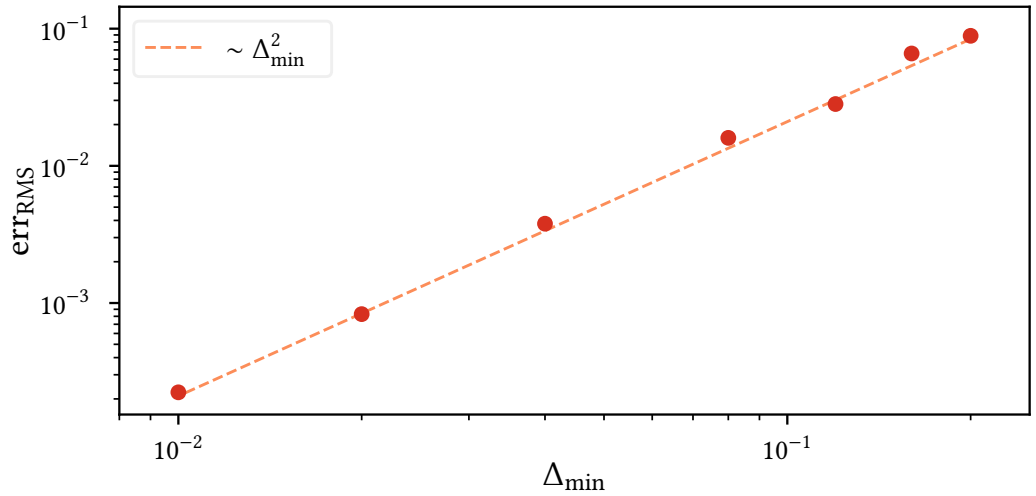


Figure 4.2: RMS error of geodesic level set approximations to an analytically known three-dimensional surface, as a function of mesh point density. Using seven different mesh point densities (here referred to by their $\Delta_{\min}$), we computed approximations of the sinusoidal surface defined by equations (4.1) and (4.3) — four of which are shown in figure 4.1 — whose RMS error (as defined in equation (4.4)) are shown as solid dots. Note how the RMS error appears to increase quadratically as a function of $\Delta_{\min}$.

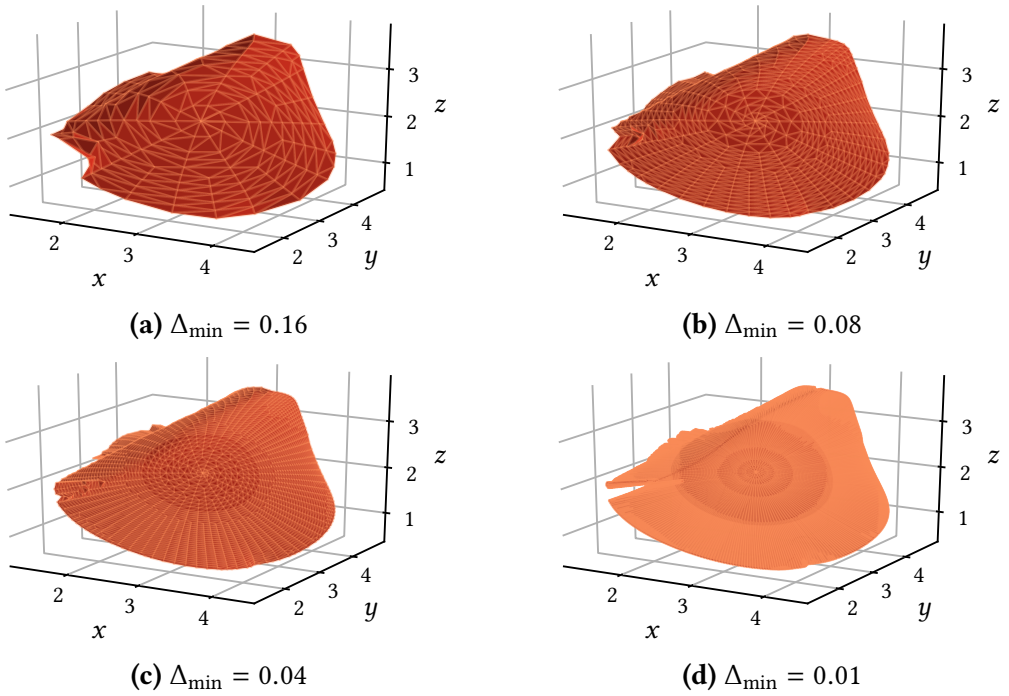
Note that, due to the local (pseudo-)planar nature of the parametrization of manifolds by means of points organized in level sets, the number of mesh points in the parametrization of a manifold *increases* quadratically as Δ_{min} decreases. Thus, the decrease in numerical error comes at the cost of an increased consumption of computational resources. For instance, the method's most computationally costly operations (namely, generating trajectories to identify new mesh points and checking for self-intersections, cf. sections 3.7 and 3.10.1, respectively) need to be performed far more frequently when the number of mesh points increases. Perhaps more crucially, an increased number of mesh points leads to an increase in the required memory. These considerations, together with the ones to follow in the immediately forthcoming section, became the foundation on which we based our choice of mesh point density for the computation of LCSs (which will be elaborated upon in sections 4.3 and 4.4).

4.1.2 *Sample manifolds computed from the steady ABC flow*

Figure 4.3 shows a sample manifold obtained for the steady ABC flow (cf. section 3.1), computed for a few different mesh point densities, and otherwise similar parameters as were used in order to approximate the sinusoidal surface (see table 4.1 and figure 4.1). From figure 4.3, it is apparent that increasing the mesh point density generally results in increased resolution. However, there appears to be some density threshold, beneath which unwanted numerical artefacts start to emerge. This is particularly evident in figure 4.3d, where the sharp indentation appears to be out of place. Such oddities could be due to accumulation of numerical error when computing new mesh points from ficticious ancestor points (as described in section 3.8.1) — which naturally occurs more often with increased mesh point density. Accordingly, our choice of mesh point density for (both variants of) the ABC flow (which will be presented in table 4.2) was guided by a desire to limit the amount of such oddities, in addition to the memory required in order to store each manifold (as mentioned in section 4.1.1).

Motivated by remark 1, we sought to verify that our computed manifolds, by virtue of containing repelling LCSs, act as invariant manifolds for arbitrary linear combinations of the ξ_1- and ξ_2-direction fields. Figure 4.4 shows a sample manifold obtained for the steady ABC flow (described in detail in section 3.1), using the parameter values provided in table 4.2 — the very same that we used in order to compute LCSs in said flow (more to follow in section 4.3). It also shows 200 different trajectories launched from (a small circle laying within the manifold, centered in) the manifold epicentre $\mathbf{x}_0$ and computed using the Dormand-Prince 8(7) adaptive ODE solver (see table 2.4 and section 3.3.2). In figure 4.4a, the trajectories are solution curves of equation (3.24), where $\mathbf{t}_{i,j}$ was kept constant along each trajectory. Meanwhile, figure 4.4b shows solution curves of equation (2.31), for 200 different pairs of weights (a, b) such that the initial trajectory directions were evenly distributed in the plane defined by the coordinate $\mathbf{x}_0$ and the unit normal $\xi_3(\mathbf{x}_0)$.

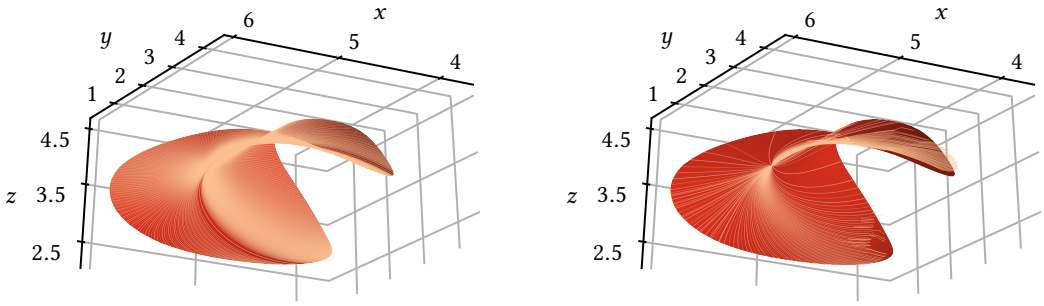
Unsurprisingly, the curves in figure 4.4a all stay within the computed manifold, as all of its constituent mesh points were computed as endpoints of trajectories in a similarly defined direction field (see section 3.7). The same applies for the curves shown in figure 4.4b; however, there seems to be some regions of the manifold which are particularly challenging



(a) $\Delta_{\min} = 0.16$ **(b)** $\Delta_{\min} = 0.08$

(c) $\Delta_{\min} = 0.04$ **(d)** $\Delta_{\min} = 0.01$

Figure 4.3: Geodesic level set approximations with varying mesh point densities, for a manifold in the steady ABC flow (see section 3.1). The manifolds were all computed using the parameters given in table 4.1, with varying mesh point densities (here noted by their $\Delta_{\min}$), where the mesh outlines are shown explicitly for clarity. Each of the approximations were grown from the same initial position $\mathbf{x}_0$. Note how, although the microscale behaviour generally is more well-resolved as the mesh point density increases, there appears to be a threshold beneath which undesired numerical artefacts — perhaps most notably near the left edge of the manifold shown in (d) — manifest.



(a) Trajectories computed in the direction field given by equation (3.24)

(b) Trajectories of the direction field given by equation (2.31), for various weights (a, b)

Figure 4.4: Trajectories orthogonal to the ξ_3-direction field, superimposed onto a computed manifold surface in the steady ABC flow. The manifold was computed using the parameters given in table 4.2. The trajectories were computed as solution curves of equations (3.24) and (2.31), respectively, starting at (a small circle laying within the manifold, centered at) the manifold epicentre $\mathbf{x}_0$. Note how none of the trajectories ever leave the manifold. The trajectories shown in (a) cover the entire manifold, which is as expected, as the manifold mesh points were computed as end points of trajectories in a similar direction field (cf. section 3.7). The trajectories shown in (b) never leave the manifold, but face difficulties reaching certain regions of it, as can be seen from locally reduced density of trajectories. For trajectories passing through $\mathbf{x}_0$, such regions are likely only accessible for very particular weights (a, b).

to hit — signified by a decreased trajectory density — it appears that very specific weights (a, b) are required to reach them. Regardless, none of the computed trajectories ever leave the computed surface, which confirms that the surface *is* in fact an invariant manifold of the ξ_1- and ξ_2-direction fields, in compliance with LCS existence criterion (2.28c) (see also remark 1).

4.2 Verifying our method of extracting LCSs from the computed manifolds

In section 4.2.1, we present an analytical flow field defined to exhibit a single repelling LCS, and how we may use the accompanying λ_3 field (i.e., the largest Cauchy-Green strain eigenvalues) to determine its location. We then show that, using the method outlined in chapter 3, we reproduce it exactly. Section 4.2.2 contains a description of how we verified that the computed LCSs are, in fact, repelling.

4.2.1 *An analytical test case*

As a verification test case for our way of extracting repelling LCSs from the computed manifolds (which is described in detail in section 3.11), we defined the purely radial velocity field

$$\dot{\mathbf{x}} = \mathbf{v}(\mathbf{x}) = \frac{\mathbf{x}}{\|\mathbf{x}\|} \sin\big(\pi(\|\mathbf{x}\| - r)\big), \tag{4.5}$$

which is changes from being directed radially inwards, to pointing radially outwards, on the sphere $\|\mathbf{x}\| = r$ (on which the velocity field is zero). We then computed strain eigenvalues and -vectors over the time interval $\mathcal{I} = [0, 1]$ for an equidistant grid of $200 \times 200 \times 200$ tracers in the domain $\mathcal{U} = [-2, 2]^3$, as outlined in sections 3.3.1 and 3.4, for $r = 1$. Moreover, we computed the arithmetic average of $\lambda_3(\mathbf{x}_0)$ across all solid angles in order to approximate it as a function of radius alone. The dependence of λ_3 as a function of radius is shown in figure 4.5, which reveals a sharp repulsion peak at $\|\mathbf{x}\| = r = 1$. This agrees well with the underlying velocity field (equation (4.5)); in particular, in passing through $\|\mathbf{x}\| = r$, the flow direction changes from radially inwards to radially outwards (or vice versa). Accordingly, we expect to find a single repelling LCS, forming a unit sphere.

Using $\varepsilon = 5 \cdot 10^{-3}$, a filtering frequency $\nu = 20$ (see section 3.5.1 and table 3.2), and otherwise the same parameters as given in table 4.2, we developed manifolds from the set of 291 grid points in the reduced $\mathcal{U}_0$ domain by the method described in sections 3.5 and 3.7–3.10. Then, by means of the method outlined in section 3.11, we extracted repelling LCSs from the computed manifolds, using a tolerance parameter $\gamma_\square = 1.2$, keeping only the LCSs with (pseudo-)surface area greater than or equal to $\mathcal{W}_{\min} = 1$. The result was a total of three identical (to numerical precision) spherical LCSs of radius 1. These are shown in figure 4.6. Seeing as the expected LCS was successfully reproduced without false positives, we concluded that our LCS extraction routine functions as intended.

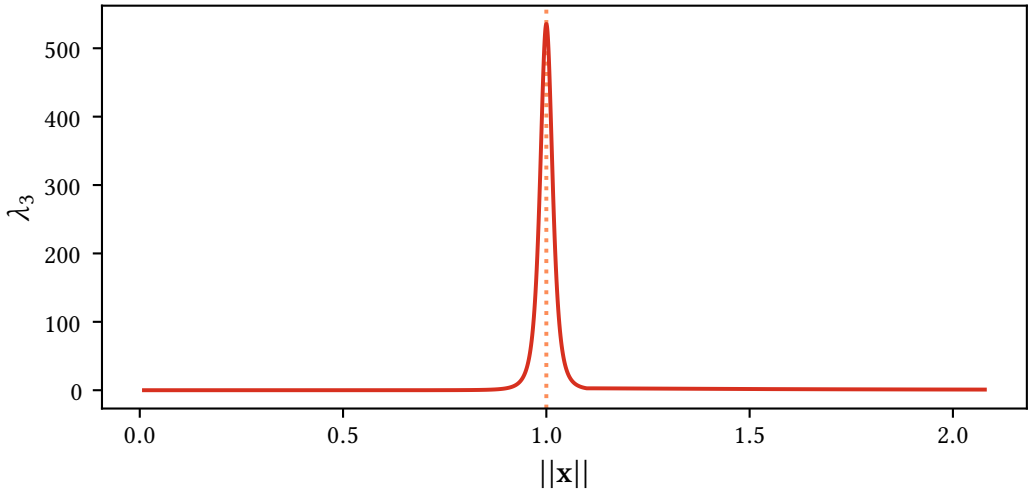


Figure 4.5: Arithmetic average of λ_3 across all solid angles, for the purely radial velocity field given by equation (4.5), using $r = 1$. Note in particular the sharp repulsion peak at $\|\mathbf{x}\| = 1$, which is a strong indicator for the presence of a repelling LCS.

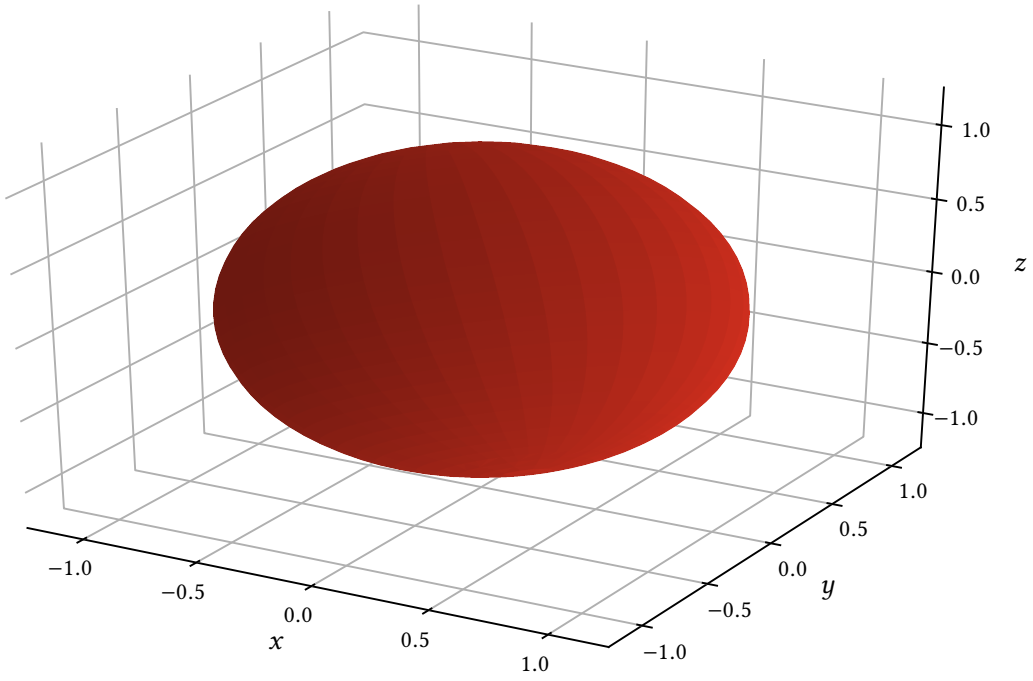
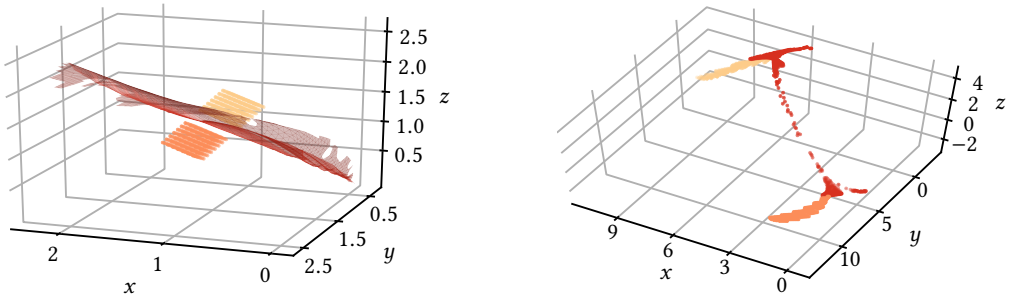


Figure 4.6: The single repelling LCS present in the purely radial velocity field given by equation (4.5) with $r = 1$. In computing the underlying manifolds, we used the filtering parameters $\varepsilon = 10^{-3}$ and $\nu = 20$ (cf. section 3.5.1 and table 3.2), and otherwise the same parameter values as given in table 4.2. To extract the LCSs, we used the tolerance parameter $\gamma_\square = 1.2$, keeping only the LCSs whose (pseudo-)surface area was greater than or equal to $\mathcal{W}_{\text{min}} = 1$ (see section 3.11). This resulted in 3 identical (to numerical precision) copies of a single, spherical, repelling LCS of unit radius — which is exactly as expected, given the sharp repulsion maximum at $\|\mathbf{x}\| = 1$, as shown in figure 4.5.

4.2.2 *Verifying that the computed LCSs are in fact repelling*

Per definitions 5 and 6, we expect lumps of particles which start out at opposite sides of a repelling LCS to quickly diverge under transport in the underlying flow system. Moreover, courtesy of being material surfaces, no particle may ever cross a repelling LCS (see section 2.3). In order to verify the impenetrable and repelling nature of the computed LCSs, we used a setup of two blobs of initial conditions, situated at opposite sides of an identified LCS surface in the steady ABC flow (more on which to follow in section 4.3). We then advected the particle blobs and the points in the parametrization of the LCS for five units of time, in the velocity field given by equations (3.1) and (3.2), using the Dormand-Prince 8(7) ODE solver in similar fashion to how we "advected" the flow map Jacobian field in the first place (see section 3.3). The initial and final states are shown in figure 4.7.



(a) Initial state, at $t = 0$

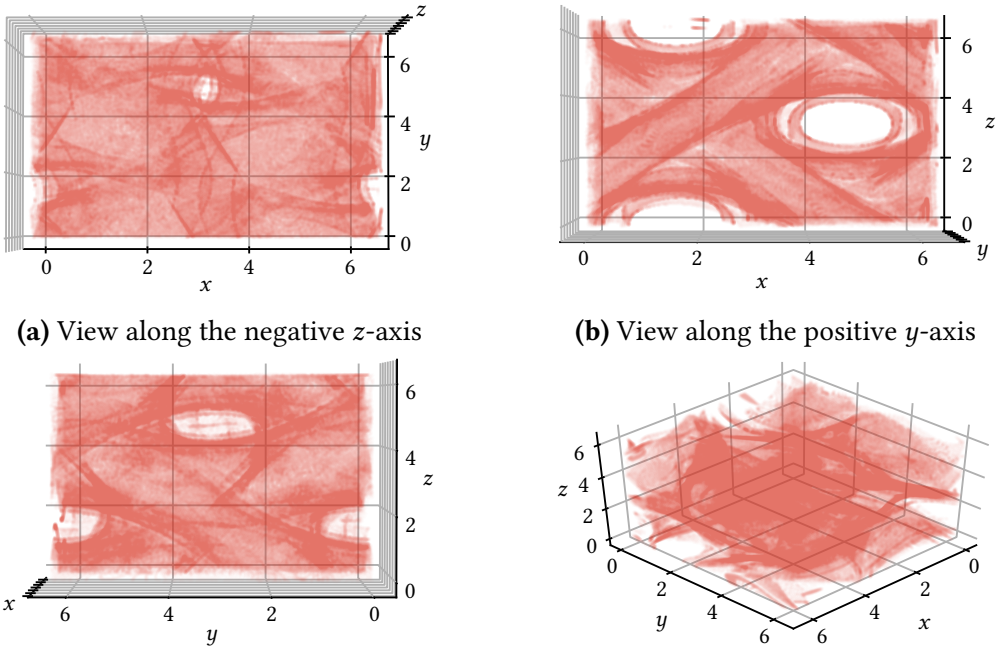
(b) Final state, at $t = 5$

Figure 4.7: Advection of tracers in order to verify the repelling nature of the computed LCSs. At $t = 0$, two blobs of initial conditions are placed at opposite sides of a repelling LCS identified in the steady ABC flow — more on which to follow in section 4.3 — as shown in (a). The two blobs, and the mesh points in the parametrization of the LCS, are then advected by the velocity field given by equations (3.1) and (3.2) using the Dormand-Prince 8(7) ODE solver until $t = 5$, for which the corresponding state is shown in (b). Note that, although the LCS triangulation breaks down under the advection, the two blobs of particles remain on fairly compact, and remain on the opposite sides of the LCS, never crossing the LCS surface. This indicates that the local centre of strongest repulsion remains located *inbetween* the two blobs of particles throughout — which is the exact behaviour we expect for a repelling LCS.

Figure 4.7 shows that, while the triangulated structure of the LCS breaks down, the two blobs of particles have been far removed from each other in the transition from figure 4.7a to figure 4.7b. Furthermore, particles belonging to a single blob remain reasonably compact, indicating that the local repulsion centre was situated between the two blobs throughout — i.e., along the LCS. One possible explanation for the relatively large amount of stretching of the mesh points in the parametrization of the LCS under the aforementioned advection, could be that (several of) the mesh points being slightly perturbed away from the *actual* LCS surface, due to round-off errors (which are practically unavoidable, as LCSs are of infinitesimal width per existence criterion (2.28b)). Finally, the fact that none of the particles belonging to either of the two blobs ever appear to move across the LCS mesh points, supports the notion that the computed LCSs do in fact act as repelling material surfaces, and thus barriers to transport.

4.3 Computed LCSs in the ABC flow

Having computed Cauchy-Green strain eigenvalues and -vectors for both variants of the ABC flow (as presented in section 3.1), we used the $\mathcal{U}_0$ domain — i.e., the grid points satisfying the LCS existence criteria (2.28a), (2.28b) and (2.28d) (the implementations of which are described in section 3.5.1) — as a first approximation to where we may reasonably expect to find repelling LCSs. Four different views of the $\mathcal{U}_0$ domains for the steady and unsteady ABC flows are shown in figures 4.8 and 4.9, respectively.



(a) View along the negative z-axis

(b) View along the positive y-axis

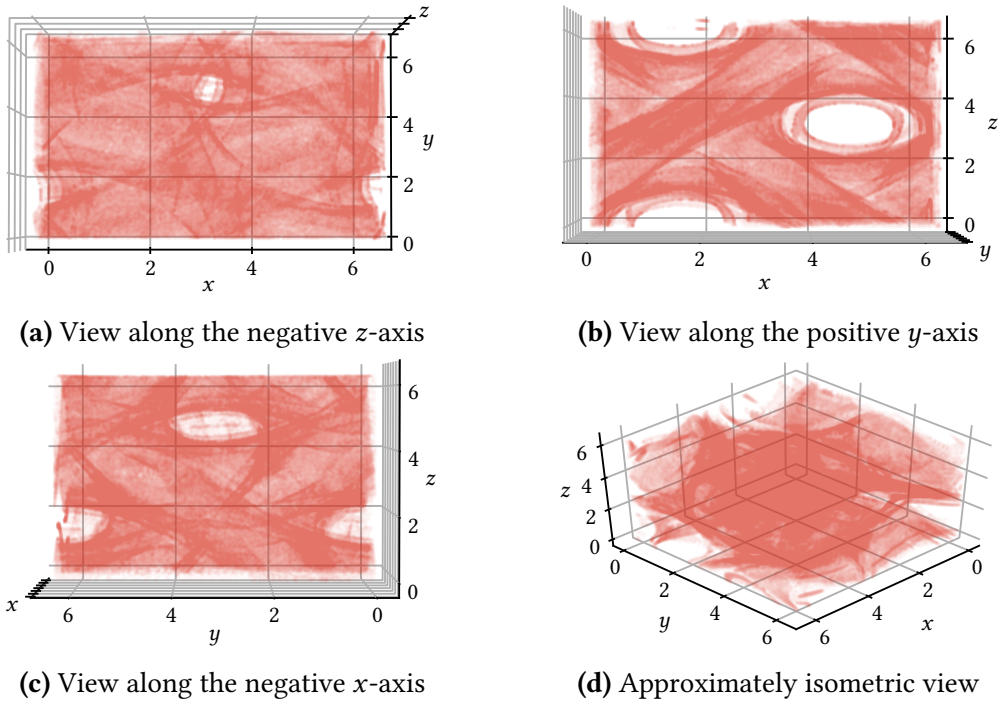
(c) View along the negative x-axis

(d) Approximately isometric view

Figure 4.8: Four views of the $\mathcal{U}_0$ domain obtained for transport in the steady ABC flow, for the time interval $\mathcal{I} = [0, 5]$, identified as the grid points which satisfy the LCS criteria (2.28a), (2.28b) and (2.28d) (details on the implementation are available in section 3.5.1). In total, the domain consists of 340 951 different points (cf. table 3.2).

Note that although the $\mathcal{U}_0$ domains for the two flow variants do not contain the same number of points (cf. table 3.2), the macroscopic trends remain the same. The differences consist of minute details, such as the topmost cavities in figures 4.8a and 4.9a being of slightly different sizes, or the point densities along the north east "bands" in figures 4.8b and 4.9b being somewhat dissimilar. This is as expected, seeing as the two underlying transport systems are lightly perturbed versions of one another.

Using the filtering parameters provided in table 3.2, we identified initial conditions for the development of manifolds as a subsets of the $\mathcal{U}_0$ domains — yielding a total of 618 and 676 points for the steady and unsteady variants of the ABC flow, respectively. Then, we computed manifolds and extracted LCSs using the parameters in tables 3.2 and 4.2 and the method outlined in sections 3.5 and 3.7–3.11. This resulted in a total of 22 LCS surfaces for the steady flow, and 31 surfaces for the unsteady flow.



(a) View along the negative z-axis

(b) View along the positive y-axis

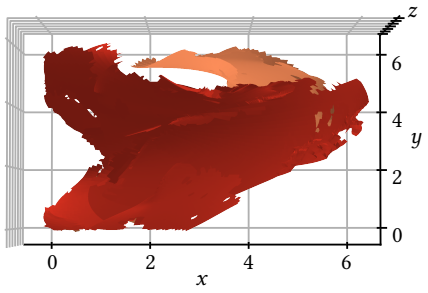
(c) View along the negative x-axis

(d) Approximately isometric view

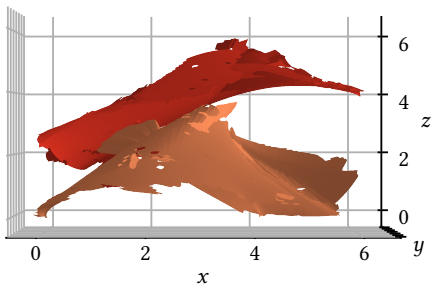
Figure 4.9: Four views of the $\mathcal{U}_0$ domain obtained for transport in the unsteady ABC flow, for the time interval $\mathcal{I} = [0, 5]$, identified as the grid points which satisfy the LCS criteria (2.28a), (2.28b) and (2.28d) (details on the implementation are available in section 3.5.1). In total, the domain consists of 361 461 different points (cf. table 3.2).

The LCSs present in the steady flow turn out to form two distinct, fairly smooth and coherent structures, which are shown in figure 4.10. Completely analogous tests to that which is outlined in section 4.2.2 verify that the computed LCSs do, in fact, act as repulsive transport barriers. The structures lie close together, yet appear not to be connected — accordingly, the two structures are highlighted by different colors for the purpose of facilitating visual comparisons. Note in particular the correspondence between the $\mathcal{U}_0$ domain, shown in figure 4.8, and the computed LCSs, shown in figure 4.10 — where the perspective of each subfigure is the same as that of the corresponding subfigure in figure 4.8. Two prominent similarities are the tunnel-like structure apparent in the middle right of figures 4.8b and 4.10b, and the indent which manifests near the bottom right corner of figures 4.8c and 4.10c.

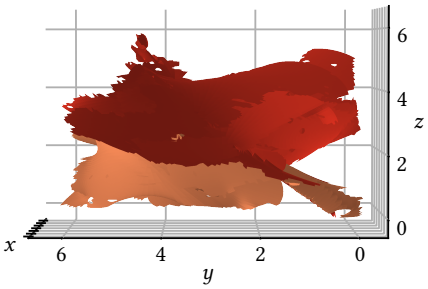
The computed LCSs in the unsteady flow constitute three distinct, smooth and coherent structures, which are shown in figure 4.11. Although the structures lie adjacent to each other, they do not seem to be connected. Thus, the different structures are indicated using disparate colors, yet again to facilitate visual comparisons. Just like for the LCSs in the steady flow, the computed LCSs strongly resemble subsets of the $\mathcal{U}_0$, shown in figure 4.9 — where the viewing angle of each $\mathcal{U}_0$ domain subfigure is the same as that of the corresponding subfigure in figure 4.11. The two largest structures apparent in figure 4.11 harmonize with the two dominant structures found for the steady flow (see figure 4.10). The smallest structure appears reasonable when considered together with the $\mathcal{U}_0$ domain — shown in figure 4.9 — in particular, it further enhances the tubular structures manifesting in the middle right of figures 4.9b and 4.11b, and the middle top of figures 4.9c and 4.11c.



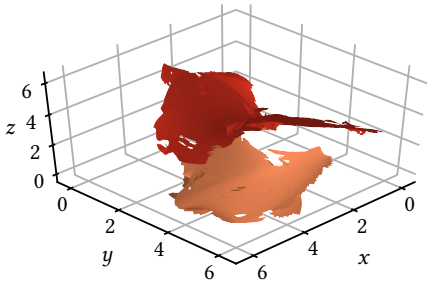
(a) View along the negative z-axis



(b) View along the positive y-axis

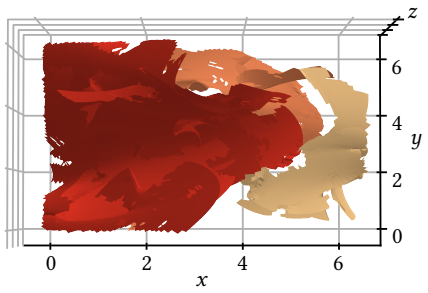


(c) View along the negative x-axis

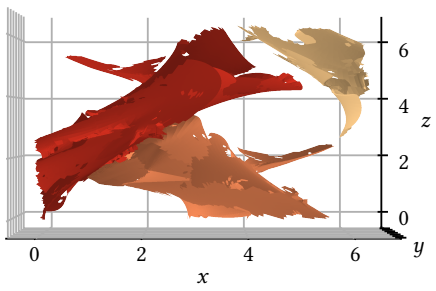


(d) Approximately isometric view

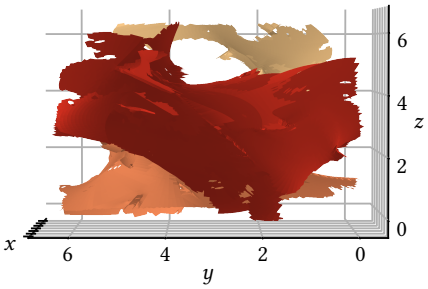
Figure 4.10: Four views of the repelling LCSs obtained for transport in the steady ABC flow, for the time interval $\mathcal{I} = [0, 5]$ (see section 3.1.1). A total of 22 surfaces constitute two distinct, fairly smooth and coherent structures, which are shown in different colors. We provide four different viewing angles (the same as the ones used in figure 4.8, which shows the computed $\mathcal{U}_0$ domain), chosen in order convey the three-dimensional structures in as great detail as possible.



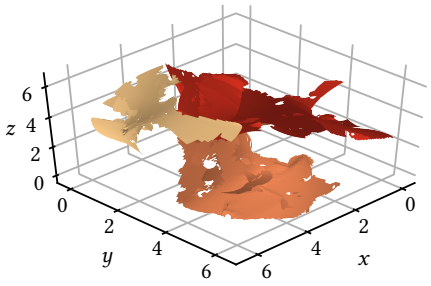
(a) View along the negative z-axis



(b) View along the positive y-axis



(c) View along the negative x-axis



(d) Approximately isometric view

Figure 4.11: Four views of the repelling LCSs obtained for transport in the unsteady ABC flow, for the time interval $\mathcal{I} = [0, 5]$ (see section 3.1.2). A total of 31 surfaces constitute three distinct, reasonably smooth and coherent structures, which are shown in different colors. We provide four different viewing angles (the same as the ones used in figure 4.9, which shows the computed $\mathcal{U}_0$ domain), chosen in order convey the three-dimensional structures in as great detail as possible.

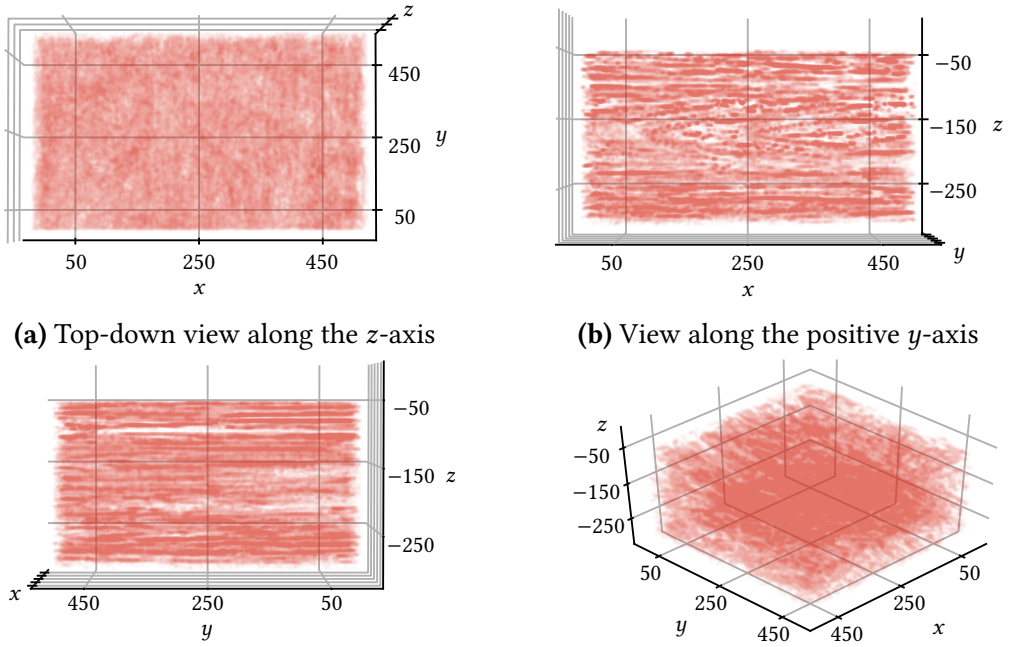
Table 4.2: Parameters used to compute manifolds, and subsequently repelling LCSs, in both variants of the ABC flow (see section 3.1). Note that the interset distances Δ_i were dynamically altered, as described in section 3.8.3. Accordingly, only the *first* interset distance, Δ_1, was set explicitly.

Parameter	Value	Description
δ_{init}	10^{-3}	Separation of innermost geodesic level set from the manifold epicentre, cf. figure 3.4
$\Delta_{\min}$, $\Delta_{\max}$	0.04, 0.16	Boundaries for interpoint separations (details found in section 3.8.1)
Δ_1	$2\Delta_{\min}$	Interset distance used to compute the second geodesic level set (see sections 3.7 and 3.8.3)
$\gamma_{\parallel}$	10^{-4}	Tolerance for detecting regions in which $\boldsymbol{\xi}_3$ is (anti-)parallel to $\mathbf{t}_{i,j}$ (see equation (3.25))
γ_{Δ}	$5 \cdot 10^{-3}$	Tolerance for the separation of a mesh point from its ancestor (per equation (3.26))
γ_{arc}	5	Sets an upper limit to trajectory lengths as $\gamma_{\text{arc}}\Delta_i$ (briefly mentioned in section 3.7.1)
$\gamma_{\cup}$	$7 \cdot 10^{-1}$	Sets an upper limit to the extent of loop-like segments of any level set (see section 3.8.2)
$\alpha_{\uparrow}$ $\alpha_{\downarrow}$ $(\Delta\alpha)_{\uparrow}$, $(\Delta\alpha)_{\downarrow}$	$8.7 \cdot 10^{-2}$ rad (5°) $4.4 \cdot 10^{-1}$ rad (25°) $2\Delta_{\min}\alpha_{\uparrow}$, $2\Delta_{\min}\alpha_{\downarrow}$	Used in a curvature-based approach to adjust interset distances (outlined in section 3.8.3)
$\gamma_{\cap}$	5	Used for terminating the expansion of self-intersecting manifolds (cf. section 3.10.1)
$\gamma_{\square}$	1.75	Relaxation parameter for extracting LCSs from the computed manifolds (see section 3.11)
$\mathcal{W}_{\min}$	6.0	Filters away the smallest LCSs measured in (pseudo-)surface area (see section 3.11)

4.4 Computed LCSs in the Førde fjord

Just like for the ABC flow, we used the $\mathcal{U}_0$ domain — i.e., the grid points satisfying the LCS existence criteria (2.28a), (2.28b) and (2.28d) (the implementations of which are described in section 3.5.1) — obtained for the transport system governed by the oceanic currents in the Førde fjord as a first approximation to where repelling LCSs can reasonably be expected to be found. Four different views of the $\mathcal{U}_0$ domain are shown in figure 4.12. Compared to the corresponding domains for the two variants of the ABC flow (as shown in figures 4.8 and 4.9), the flow in the Førde fjord appears a lot more chaotic, with fewer discernible macroscopic trends. Figures 4.12b and 4.12c do, however, indicate that the points in the oceanic $\mathcal{U}_0$ domain are loosely organized in horizontal layers.

Using the filtering parameters provided in table 3.2, we identified initial conditions for the development of manifolds as a subset of the $\mathcal{U}_0$ domain; yielding a total of 1 631 points. Then we computed manifolds and extracted LCSs using the parameters in tables 3.2 and 4.3 and the method outlined in sections 3.5 and 3.7–3.11. This resulted in a total of 110 LCS



(a) Top-down view along the z-axis

(b) View along the positive y-axis

(c) View along the negative x-axis

(d) Approximately isometric view

Figure 4.12: Four views of the $\mathcal{U}_0$ domain obtained for transport in the Førde fjord, over a time interval of 12 hours (see section 3.2), identified as the grid points which satisfy the LCS criteria (2.28a), (2.28b) and (2.28d) (details on the implementation are available in section 3.5.1). Note that the axes are given in metres, where $z = 0$ corresponds to the surface level, increasing downwards. In total, the domain consists of 1 631 points (cf. table 3.2).

surfaces. As figure 4.13 indicates, these largely appear to be organized in a sequence of horizontal layers. Furthermore, all of the LCSs are sufficiently large to warrant treating them as individual entities. Thus, in contrast to the LCSs obtained in (either version of) the ABC flow (see section 4.3), we elected to assign each LCS a numerical value

$$Q_i = \frac{\log\left(\overline{\lambda}_3\right)_i}{\max\limits_i \left\{\log\left(\overline{\lambda}_3\right)_i\right\}}, \tag{4.6}$$

where $\left(\overline{\lambda}_3\right)_i$, the repulsion average of LCS surface i, is defined in equation (3.32). We then used the (unit normalized) set of numbers $\{Q_i\}$ to select a color for the corresponding LCSs, drawn from a perceptually uniform colormap.

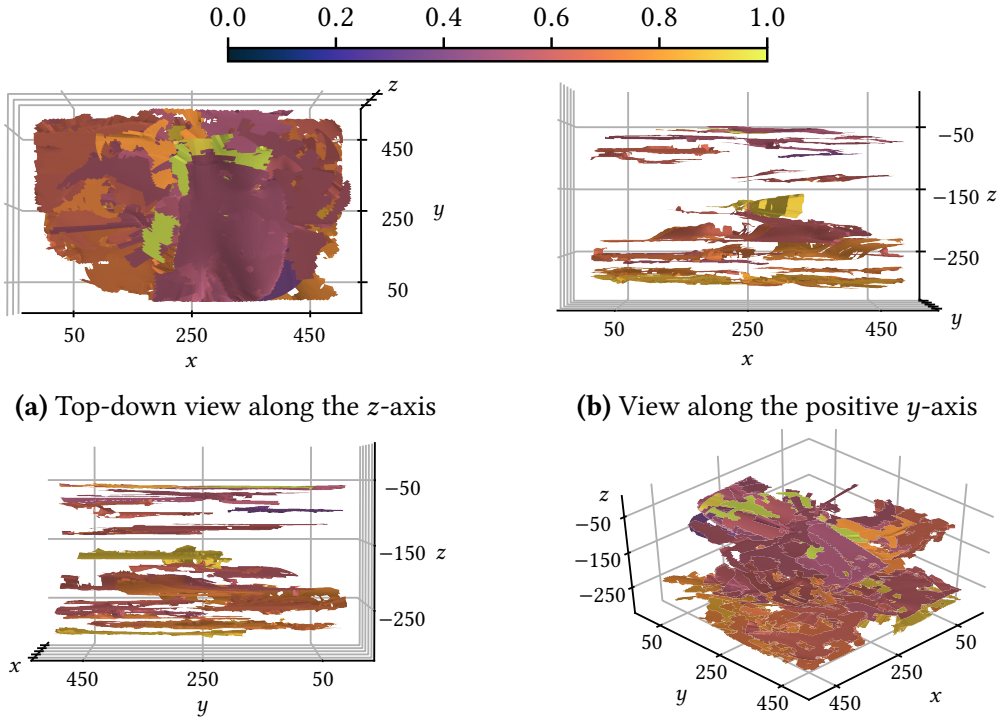
When comparing the computed LCSs (figure 4.13) to the corresponding $\mathcal{U}_0$ domain (figure 4.12), the organization of the LCSs in horizontal layers seems reasonable. This is particularly apparent from inspecting figures 4.12b and 4.13b, and figures 4.12c and 4.13c. This indicates that the oceanic flow undergoes the most stretching in the vertical direction. Moreover, the repulsion appears to be largely uniform within any given depth layer.

Similarly to our treatment of an LCS surface in the steady ABC flow in section 4.2.2, we placed two blobs of particles on either side of the most strongly repulsive LCS present in the fjord subdomain — perhaps most easily spotted in the middle of figures 4.13b and 4.13c — and allowed the oceanic currents to transport the particles as well as the computed LCS for

Table 4.3: Parameters used to compute manifolds, and subsequently repelling LCSs, in the Førde fjord (see section 3.2). Note that the interset distances Δ_i were dynamically altered, as described in section 3.8.3. Accordingly, only the *first* interset distance, Δ_1, was set explicitly. Moreover, as the domain of interest is scaled in units of metre, so too are δ_{init}, $\Delta_{\min}$, $\Delta_{\max}$ and Δ_1; whereas $\mathcal{W}_{\min}$ is given in square metre.

Parameter	Value	Description
δ_{init}	10^{-1}	Separation of innermost geodesic level set from the manifold epicentre, cf. figure 3.4
$\Delta_{\min}$, $\Delta_{\max}$	2, 8	Boundaries for interpoint separations (details found in section 3.8.1)
Δ_1	$2\Delta_{\min}$	Interset distance used to compute the second geodesic level set (see sections 3.7 and 3.8.3)
$\gamma_{\parallel}$	10^{-4}	Tolerance for detecting regions in which $\boldsymbol{\xi}_3$ is (anti-)parallel to $\mathbf{t}_{i,j}$ (see equation (3.25))
γ_{Δ}	$5 \cdot 10^{-3}$	Tolerance for the separation of a mesh point from its ancestor (per equation (3.26))
γ_{arc}	5	Sets an upper limit to trajectory lengths as $\gamma_{\text{arc}}\Delta_i$ (briefly mentioned in section 3.7.1)
$\gamma_{\cup}$	$7 \cdot 10^{-1}$	Sets an upper limit to the extent of loop-like segments of any level set (see section 3.8.2)
$\alpha_{\uparrow}$ $\alpha_{\downarrow}$ $(\Delta\alpha)_{\uparrow}$, $(\Delta\alpha)_{\downarrow}$	$8.7 \cdot 10^{-2}$ rad (5°) $4.4 \cdot 10^{-1}$ rad (25°) $2\Delta_{\min}\alpha_{\uparrow}$, $2\Delta_{\min}\alpha_{\downarrow}$	Used in a curvature-based approach to adjust interset distances (outlined in section 3.8.3)
$\gamma_{\cap}$	5	Used for terminating the expansion of self-intersecting manifolds (cf. section 3.10.1)
$\gamma_{\square}$	1.2	Relaxation parameter for extracting LCSs from the computed manifolds (see section 3.11)
$\mathcal{W}_{\min}$	20 000	Filters away the smallest LCSs measured in (pseudo-)surface area (see section 3.11)

the entirety of our 12 hour time interval of interest. The initial and final states are shown in figure 4.14. Like for the ABC flow case, the triangulated structure of the LCS breaks down, yet the two blobs of particles diverge from each other in the transition from figure 4.14a to figure 4.14b. The particles belonging to either blob remain close together, and none of them appear to move across the LCS mesh point. This suggests that, like the ones computed for the ABC flow, the LCSs obtained for flow in the Førde fjord act as repelling material surfaces, and thereby also as transport barriers.



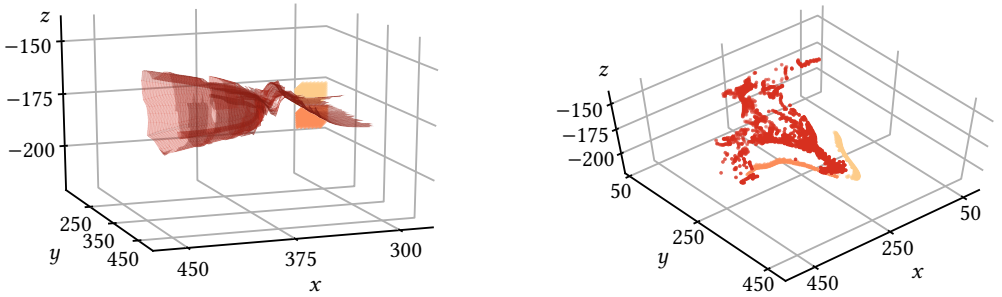
(a) Top-down view along the z-axis

(b) View along the positive y-axis

(c) View along the negative x-axis

(d) Approximately isometric view

Figure 4.13: Four views of the repelling LCSs obtained for transport in the Førde fjord, over a time interval of 12 hours (see section 3.2). A total of 110 distinct surfaces are shown, which are colored according to their relative repulsion averages (per equation (4.6)) using the perceptually uniform colormap shown at the top. We provide four different viewing angles (the same as the ones used in figure 4.12, which shows the computed $\mathcal{U}_0$ domain), chosen in order to convey the three-dimensional structures in as great detail as possible.



(a) Initial state, June 1 2013, 00:00

(b) Final state, June 1 2013, 12:00

Figure 4.14: Advection of tracers in order to verify the repelling nature of the computed LCSs in the Førde fjord. At midnight June 1, 2013, two blobs of initial conditions are placed at opposite sides of the most strongly repulsive LCS identified for flow in the Førde fjord (see figure 4.13) as shown in (a). The two blobs, and the mesh points in the parametrization of the LCS, are then advected in the model data for the oceanic currents briefly described in section 3.2.1, using the Dormand-Prince 8(7) ODE solver, for the 12 hour time interval of interest, where the final state is shown in (b). Note that, although the LCS triangulation breaks down under the advection, the two blobs of particles remain on fairly compact, and remain on the opposite sides of the LCS, never crossing the LCS surface. This indicates that the local centre of strongest repulsion remains located *inbetween* the two blobs of particles throughout — which is the exact behaviour we expect for a repelling LCS.

5 Discussion

The main focus of this chapter is the analysis and critique of our method for computing LCSs in three-dimensional flows (presented in chapter 3). Sections 5.1–5.3 contain thorough examinations of our various choices and assumptions made in order to compute invariant manifolds everywhere orthogonal to the direction of strongest repulsion. In sections 5.4 and 5.5, we review our method of extracting repelling LCSs from the aforementioned manifolds. Lastly, section 5.6 assesses the overall relevance of our method for computing three-dimensional transport barriers. Throughout this chapter, we present potential topics of further research, mainly in terms of further method refinement.

5.1 Comments on the method of geodesic level sets

As mentioned in chapter 3, our take on the method of geodesic level sets (see section 3.7) hinges on characteristic properties of hyperbolic LCSs (cf. definition 8). For *repelling* LCSs (see definition 6), which we concentrated on for this project, the existence criterion given in equation (2.28c) states that these are everywhere *orthogonal* to the local direction of strongest repulsion. The extra degree of freedom compared to manifolds defined as being everywhere tangent to some direction field (such as the strange attractor in the Lorenz system, as was considered by Krauskopf, Osinga, et al. (2005)) facilitated a more effective (in terms of computational runtime) and conceptually simpler method of generating such manifolds, than the more direct adaption of Krauskopf, Osinga, et al.'s (2005) method (which is outlined in section 3.6).

Although useful for managing the mesh accuracy (see section 3.8), and central to our triangulation algorithm (outlined in detail in section 3.9), exclusively arranging mesh points in closed topological circles has irrefutable weaknesses. In systems for which periodic boundary conditions are not applicable, the addition of further level sets is promptly terminated when one or more of the trajectories used to compute new mesh points (see section 3.7) exits the computational domain. This impedes the ability to resolve LCS behaviour near the domain boundaries. Although this issue could be managed by computing strain eigenvalues and -vectors in a domain *containing* the domain of interest and expanding from it in all directions — which is how we were able to resolve the boundary behaviour for the LCSs in the Førde fjord (presented in section 4.4) — this workaround is computationally demanding. Depending on to what extent the underlying flow system is known (or modelled), and the location of the domain of interest, it might not even be possible.

On a related note, demanding that a new geodesic level set is computed using a mesh point descending from *each* of the mesh points in the preceding level set renders computing the underlying manifold in its entirety from a *single* focal point $\mathbf{x}_0$ (see section 3.5) quite difficult. If a single point strand (that is, the set of mesh points which can be traced back to a single, common ancestor) is terminated — either due to reaching the domain edges, or failure to compute a new mesh point (see section 3.7.2) — so too is the addition of further level sets. Thus, unless the manifold as a whole expands as a perfectly planar, circular surface, as

seen from the focal point $\mathbf{x}_0$, encapsulating it in its entirety by means of geodesic level sets becomes impossible — even when ignoring the possibility of numerical error.

As mentioned in section 3.7, we let each computed mesh point *inherit* its unit tangent $\mathbf{t}$ from its direct ancestor, rather than computing new unit tangents using the interpolation curve C_i. This was a conscious choice. In our experience, the computed level sets quickly started to form fully three-dimensional topological circles, leading to notable local curvature along the interpolation curves. This rendered selecting how far to either side of a given mesh point to move, in order to approximate the local tangent vector by a finite coordinate difference, hard to do in a consistent manner. Simply using the coordinates of the mesh point's nearest neighbors (as originaly suggested by Krauskopf, Osinga, et al. (2005)) for this purpose was also found to be inconsistent. Failure to compute tangent vectors consistently often lead neighboring point strands to form intersections, yielding disorderly meshes which in turn frequently lead the points constituting a level set to *not* form a topological circle. These issues are not present in our aforementioned inheritance-based approach. However, if a computed manifold were to twist itself in such a way that a unit tangent $\mathbf{t}$ laid *within* it, this could result in failure to compute one or more new mesh points, which would then lead to terminating the process of adding new level sets (alternatively, terminating the process of expanding one or more point strands, cf. the preceding paragraphs) prematurely. We addressed this issue using the tolerance parameter $\gamma_{\|}$; see section 3.7 and, in particular, equation (3.25) for details.

Our variation of the method of geodesic level sets contains many degrees of freedom (see e.g. tables 3.2 and 4.2). Some of these parameters, mainly those governing the minimum and maximum allowed separations between neighboring mesh points (see section 3.8.1), could reasonably be chosen based on considerations pertaining to the spatial extent of the computational domain; alternatively, to what extent the small-scale details of the LCSs are to be resolved. How to determine several other undeniably key parameter values — such as the tolerances for the detection of intersecting manifolds (see section 3.10), and the removal of mesh points which form undesired bulges (which is outlined in section 3.8.2) — remains less obvious.

That being said, the parameters related to the curvature-guided approach to dynamically adjust the interset separations were, in our experience, of less importance; as briefly mentioned in section 3.8.3, the interset step length was rarely *increased*. More often than not, the interset step length was quickly reduced to its lower limit, and remained at that level for the generation of all subsequent level sets. This is not entirely unexpected, as sufficiently large curvature within a *single* region of any given level set sufficed to lower the step length (compare equations (3.28) and (3.29)). Moreover, as the geodesic level sets continuously expand, encountering such a region becomes increasingly likely. As the accuracy of the computed mesh points is independent of the density of mesh points, the main use of the interset step size is to manage the interpolation error inherent to our linear triangulation scheme (see section 3.9 and Krauskopf and Osinga (2003)). Thus, it seems reasonable to forego the dynamic interset step length in favor of a fixed one, in cases where the minute

details of LCS surfaces are unimportant. As tentatively suggested in the above, doing so reduces the overall complexity of our method for generating mesh points, in addition to reducing the number of free parameters.

Another way of organizing the mesh points, which could circumvent some of the aforementioned limitations of the present approach, would be as a group of point strands, each associated with a particular unit tangent $\mathbf{t}$ — determined using C_1, the interpolation curve of the innermost level set, in similar fashion to that which is described in section 3.5.2. Treating the expansion along each point strand independently would then permit further expansion of a manifold even when one or more point strands would reach the domain boundaries; this would also solve the possible issue of computed trajectories along any given strand failing to yield acceptable mesh points (see section 3.7.2) — in which case only the strands in question need to be terminated, rather than prohibiting the addition of further geodesic level sets entirely.

Aside from taking a step further away from the method of geodesic level set as originally proposed by Krauskopf, Osinga, et al. (2005), organizing mesh points as a group of point strands would necessitate developing the mesh accuracy management method outlined in section 3.8 further. In particular, the present approach utilizes the interpolation curve C_i in order to insert mesh points inbetween nearest neighbor mesh points in level set $\mathcal{M}_{i+1}$ which are deemed to lie too far away from each other (see section 3.8.1). A reasonable approach for the case of point strands could be to, having identified two strands inbetween which a new mesh point is needed, retrieve their respective ancestor points in the innermost level set, and then compute (the trajectory of a) new point strand starting out at a point on C_1, midway inbetween said ancestors, keeping only the first point along the strand which is required to maintain the point density.

Depending on the extent of the initial level set — that is, the circumference of its interpolation curve C_1 — and the required mesh point density, however, this approach could be prone to errors arising from numerical round-off errors in computing the start points of new point strands. In order to maintain an overall mesh structure suitable for triangulation purposes, the steps along each point strand should be equal; that is, mesh points i and $i + 1$ along all point strands should be separated by the same distance Δ_i, whereas all interpoint step sizes $\{\Delta_i\}$ need not be equal. This way, a quasi-circular structure is maintained as the point strands expand — that is, subject to one or more of them reaching the domain boundaries or being terminated due to ending up in (approximately) closed orbits (see section 3.7.1).

Lastly, several other methods of computing invariant manifolds of vector fields exist; of whom some might be well-suited in the context of LCSs. For instance, the method of geodesic level sets is one of five methods presented by Krauskopf, Osinga, et al. (2005); none of the others were pursued as part of this work. Exploring the strengths and weaknesses of other approaches to computing three-dimensional hyperbolic LCSs remains beyond the scope of this project.

5.2 On our approach to computing the Cauchy-Green strain characteristics

We used an SVD decomposition of the flow map Jacobian to find the Cauchy-Green strain eigenvalues and -vectors, rather than computing these directly from the Cauchy-Green strain tensor field — as described in sections 3.3 and 3.4. This approach, suggested by Miron et al. (2012) and endorsed by Oettinger and Haller (2016), boasts superior accuracy compared to the more conventional approach of approximating the directional derivatives of the flow map (i.e., the components of the flow map Jacobian) by applying a finite difference method and then explicitly computing the Cauchy-Green strain tensor field (which Farazmand and Haller (2012a) did in order to find LCSs in two-dimensional flow). As a consequence of the increased mathematical complexity, directly transporting the flow map Jacobian field is considerably more expensive in terms of computational resources, compared to methods based upon the application of finite differences. Moreover, our approach relies on bounded first spatial derivatives of the underlying velocity field, as is evident from inspecting equation (2.22). This should, however, not be an issue when considering smooth analytical test cases, or when using a high (quadratic or higher, cf. section 2.1.2) order interpolation method for gridded data. Alternatively, the derivatives can be approximated by e.g. a finite difference method. Should any of these approaches prove impractical, the method of Farazmand and Haller (2012a) could be sufficient.

Note that the resolution of the grid of tracers, on which the Cauchy-Green strain eigenvalues and -vectors are computed (see sections 3.3 and 3.4), plays a critically important role in the successful detection of LCSs. For instance, difficulties arose when using excessively sparse grids of tracers such that no initial conditions for the generation of manifolds — that is, points satisfying the LCS existence criteria (2.28a), (2.28b) and (2.28d) — in the transport system governed by equation (4.5) were sufficiently close to the strongly repulsive unit sphere (see section 4.2.1 and figure 4.6). To our knowledge, there is no way to determine *a priori* what density of tracers will suffice for any given flow system. Thus, even though educated guesses based on the scale at which one is interested in the microscopic behaviour in the system might be prudent, we recommend to use as fine a grid of tracers as possible, within the constraints set by the available computational resources.

As mentioned in section 3.2, we interpolated the model velocity field using quadrivariate, cubic B-splines — that is, cubic spline interpolation in time and all spatial directions. This involved us having to keep the model data pertaining to the region of interest (that is, the model data for a domain expanding beyond said region in all directions, in order to resolve the behaviour near the boundaries, cf. section 3.10) for the entirety of the considered time interval. Because of our data set's disparate resolution in the horizontal and vertical directions (as mentioned in section 3.2), in addition to the small spatial region of interest (in comparison to the entire fjord), the use of quadrivariate interpolation was unproblematic regarding the consumption of working memory. For other applications, however, this is not necessarily the case, depending on the problem's scale (temporal and spatial) and the resolution of the model data.

Should memory consumption be an issue for a discrete dataset, it is possible to forego temporal interpolation entirely (provided that the sampling rate is adequate), and instead opt for trivariate interpolation in space, generating an interpolation object for each time instance. This renders the use of ODE solvers with adaptive step size — like the Dormand-Prince 8(7) method we wound up choosing (more on that to follow in section 5.3) — moot, as the solution time steps would then have to coincide with the time levels of the dataset. Using a lower order ODE solution method, such as the explicit trapezoidal rule (which does not require intermediary samples when moving from one time level to the next), does, however, yield inferior performance, as high-order embedded Runge-Kutta solvers are generally much more efficient (Løken 2017).

5.3 Regarding our choice of numerical ODE solver

As previously mentioned, the Dormand-Prince 8(7) method was used for both the tracer advection used to compute the Cauchy-Green strain eigenvalues and -vectors, and computing new mesh points in the expansion of computed manifolds (as described in sections 3.3 and 3.4, and section 3.7, respectively). In contrast to traditional singlestep ODE solvers — such as the classical 4^{th}-order Runge-Kutta method (see section 2.1) — which would typically require different step sizes for the two cases (as the scales at which the dynamics occurs in the two systems can reasonably be presumed to be disparate), using an embedded method with a single set of numerical tolerance parameters for the integration step adjustment (see section 3.3.2) means that the propagation of numerical round-off errors can reasonably be expected to have occurred in a consistent manner throughout.

Furthermore, the aforementioned tolerance levels can be selected independently of the scales of the system; the results obtained by Løken (2017) suggest that using the Dormand-Prince 8(7) method with tolerance levels ranging from 10^{-10} to 10^{-5} are sufficient in order for numerical round-off error to be the main concern, compared to pure integration error, when computing hyperbolic LCSs (albeit in two-dimensional systems). In addition, using an embedded ODE solver meant that explicitly defining integration step sizes for the two aforementioned transport processes (for the computation of strain eigenvalues and -vectors, and for obtaining new mesh points, respectively) was not required. This is significant, because the present approach already involves several free parameters (see section 5.1).

Løken (2017) showed that the Dormand-Prince 8(7) method yields very accurate numerical approximations at a very low computational cost — at least, that is, for smooth flow systems (see section 2.2). As is apparent from definition 2, the accuracy of numerical solutions obtained by using Runge-Kutta solvers depend on the order of the method itself, in addition to the smoothness of the underlying function. Although generally more accurate, higher-order ODE solvers yield increasingly diminishing returns compared to their lower-order siblings when the order of the ODE solver exceeds the function's number of smooth derivatives.

Thus, for gridded model data — both regarding the oceanic currents of the Førde fjord (section 3.2.1), and the discretely sampled Cauchy-Green strain eigenvalue and -vector

fields (sections 3.3 and 3.4) — the interpolation routine sets an upper bound in terms of the accuracy with which LCSs can be computed. For more complex systems than the ones investigated here, the interaction between the integration and interpolation schemes could be critical; both in terms of numerical precision and computational resource consumption. Independently of the scales at which well-resolved LCSs are sought in a given transport system, the aforementioned effects warrant further investigation, yet remain beyond the scope of this project.

5.4 Reflections upon the process of identifying locally most repelling material surfaces

To our knowledge, standardized algorithms for the detection of points which satisfy LCS existence criterion (2.28d) — which is used to identify points that might be local repulsion maxima — have not yet been found. In particular, numerical round-off error makes detecting the zeros of inner products, like the one in condition (2.28d), challenging. The conditions given in equations (2.28a)–(2.28c) are quite unambiguous, in comparison. Moreover, while the concept of *local* maxima for the normal repulsion is well-defined for analytical systems, this is not the case for numerical simulations. In contrast to the infinitesimal neighborhoods one may consider for analytical flow, the discrete nature of numerics — coupled with possible numerical round-off error — means that the regions within which one looks for repulsion maxima must have finite extent. Accordingly, the scale at which one performs *local* comparisons becomes significant. Our approach to finding points which satisfy LCS existence condition (2.28d) is outlined in section 3.5.1, where we used a small perturbation parameter ε to define the extent of the nearby regions within which we sought local repulsion maxima. In particular, ε was chosen to be an order of magnitude smaller than the grid spacing, in order for the local neighborhoods to be of the same approximate scale as the smallest level of detail which the cubic interpolation schemes (see sections 3.2 and 3.4) can reasonably be expected to resolve.

An alternative way of checking if a point satisfies condition (2.28d) could be extending the work of Farazmand and Haller (2012a) from two to three dimensions. A direct adaption would amount to finding all intersections between the computed surfaces and a family of planes, then, having organized the surfaces in bundles based on their intersections with any given plane being sufficiently close, flagging the most repulsive surface within each bundle as a local strain maximizer. This would not, however, *fully* solve the challenge of translating the concept of locality to numerics. Furthermore, there does not appear to be an unambiguous way of selecting the aforementioned family of planes — a notion which is supported by Farazmand and Haller (2012a) failing to mention any details on the set of lines (the two-dimensional equivalent of the family of planes) they used for their applications. Lastly, as the intersection between any material surface and a plane generally forms a curve, rather than a unique intersection point, the process of identifying a material surface whose intersections with any given plane lie sufficiently close to those of any other material surface could easily become expensive in terms of computational resources.

Another option, somewhat similar to the approach of Farazmand and Haller (2012a), would be to simply divide the computational domain into a set of smaller domains, identifying the computed surfaces which (partially) lie within each such region and then flagging the most strongly repelling surface within each subdomain as a local repulsion maximizer. Like the selection of planes in the aforementioned adaption of the method of Farazmand and Haller, however, there does not (to our knowledge) exist an objective way to select the size nor locations of these subdomains. Furthermore, the approach of comparisons within smaller sets of the computational domain does not take the orientation of the material surfaces into account — a weakness which is shared with the previously mentioned adaption of Farazmand and Haller's method. Conceptually, using direct comparisons of material surfaces in order to detect the surfaces which form local repulsion maxima should not involve comparing surfaces with disparate orientations. Neither should two surfaces for which only a small subset of one lies anywhere near the other; such material surfaces would likely influence the overall flow patterns quite differently. Highly optimized algorithms would likely be needed in order to check such extra comparison criteria without excess consumption of the available computational resources.

To our knowledge, computing LCSs in three-dimensional flow has not been attempted particularly frequently, rendering us without reliable reference cases. Blazevski and Haller (2014) construct three-dimensional LCSs by dividing their computational domain into a set of planes. After computing LCSs within each plane as locally most repelling material lines, they consider these LCS curves as the projections of three-dimensional structures onto the plane family, whereupon they apply a curve fitting algorithm to connect the LCS curves and form three-dimensional structures. This approach is not, however, fully three-dimensional, as it ignores transport orthogonal to the planes; moreover, Blazevski and Haller do not provide any evidence as to whether or not their approach is robust with regards to the orientation (or density) of the plane family. Oettinger and Haller (2016) seemingly do not even attempt to identify local repulsion or attraction maxima in their considerations of hyperbolic LCSs (see definitions 6–8). Notably, Oettinger and Haller appear to be content with identifying invariant manifolds of the ξ_2- and ξ_1-direction fields (in the case of attracting LCSs, the ξ_1-direction field is replaced with the ξ_3-direction field) as regions where LCSs may reasonably be expected to exist (see remark 1).

Although outside of the scope of this project, yet another alternative approach would be to identify all material surfaces which lie reasonably close to each other, having similar spatial orientation and (preferably) size, by means of some numerical clustering algorithm. Then, the most strongly repellent surface segments within each cluster could reasonably be considered as the local repulsion maximizer. Possibly geared towards a project pertaining to machine learning, this sort of approach would benefit greatly from reliable reference cases. Furthermore, basing the selection process solely on the computed manifolds' repulsion averages (as defined in equation (3.32)), or other macroscale quantities, need not necessarily be the best possible approach in terms of extracting the most significant LCSs. In particular, subsets of large LCSs could exhibit significant repulsion, without necessarily resulting in large repulsion averages. Similarly, relatively small yet strongly repelling LCSs need not be

particularly significant for the overall flow pattern.

Compared with the aforementioned alternatives, one could argue that our approach of checking whether or not each *point* in a computed material surface satisfies existence criterion (2.28d) by considering a small neighborhood around them (whose extent is defined by the perturbation parameter ε, cf. section 3.5.1) is more faithful to the underlying theory. Note in particular that our approach is based on the local repulsion of small subsets (namely, the points constituting the parametrization) of the computed manifolds, in contrast to the global comparison of repulsion averages (or similar quantities) inherent to the other methods. However, put simply, there is certainly room for further research with regards to the numerical implementation of LCS existence criterion (2.28d).

5.5 Thoughts on the extraction of LCSs as subsets of the computed manifolds

While the perturbation parameter ε — used in order to identify local repulsion maxima (discussed in section 5.4) — may reasonably be chosen based on the density of advected tracers (see section 3.3), how to determine suitable values for the relaxation parameter $\gamma_\square$ and the filtering weight $\mathcal{W}_{\text{min}}$ used to extract repelling LCSs from our computed manifolds (see section 3.11) is less self-evident. Specifically, these might be selected based on user-determined, subjective determinations, such as to modify the *number* of ensuing LCSs in addition to their sizes. This might be useful in some settings, but is hard to reconcile with the otherwise objective nature of our LCS generation routines.

Two main regimes exist regarding the choices of $\gamma_\square$ and $\mathcal{W}_{\text{min}}$. Our preference — in the following referred to as the *clustering* approach — involves selecting (relatively) small values for both $\gamma_\square$ and $\mathcal{W}_{\text{min}}$, and then treating clusters of overlapping LCS surface elements as single entities. This necessitates a way of sorting the surface elements into bundles of interconnected surfaces (manual or otherwise), yet yields LCSs which are not particularly dependent on the parameter values. The other way — in the following referred to as the *carve-out* approach — involves the use of large values for both $\gamma_\square$ and $\mathcal{W}_{\text{min}}$, and considering each resulting surface element as a standalone LCS. While this approach would generally yield smoother and more aesthetically pleasing LCS surfaces, it may easily involve significant bias towards the largest material surfaces — without there being an obvious way of determining whether or not these form the most significant barriers to transport *a priori* — and depends sensitively on the parameter choices.

Whether the clustering approach is the better choice in the general case, or if an intermediate approach might be more sensible, remains to be seen. Moreover, for some applications, the assumption that only sufficiently large repelling LCSs influence the overall flow patterns significantly (originally suggested by Farazmand and Haller (2012a)) might not hold; in particular where small, yet very strongly repelling material surfaces are concerned. In such cases, the use $\mathcal{W}_{\text{min}}$ to filter away the supposed least coherent (i.e., those we believe to be most affected by numerical noise, which might just not be LCSs at all) LCS surfaces is

problematic. Investigations pertaining to the general case, however, remain beyond the scope of this project.

5.6 Remarks on the overall computation of three-dimensional repelling LCSs

As briefly alluded to in section 5.5, all of the LCS surfaces presented in chapter 4 were generated using the clustering approach, favored over the carve-out approach due to being less reliant on large underlying manifolds (in addition to the conceptual advantages outlined in section 5.5). This proved important for LCS analysis in both variants of the ABC flow, for which the computed manifolds often terminated due to the detection of unphysical self-intersections (see section 3.10). Possibly caused by small numerical errors perturbing a small number of mesh points onto an adjacent manifold — accompanied by the subsequent interpolation curves C_i being distorted, yielding compound errors when inserting mesh points from ficticious ancestor points (see section 3.8) — this issue might be mitigated by organizing mesh points as bundles of point strands rather than geodesic circles (as outlined in section 5.1).

Following manual bundling of (partly) overlapping surface elements, the LCSs in either variant of the ABC flow (see figures 4.10 and 4.11), were obtained as structures consisting of between 3 and 23 unique surface elements. Similarly, LCSs obtained in the Førde fjord (see figure 4.13), were largely grouped in a sequence of horizontal layers. We elected to color each surface element constituting the LCSs in the Førde fjord according to their relative repulsion average, rather than assigning a single color to each layer, in order to investigate whether or not the repulsion within a given horizontal layer is uniform. From inspection of figure 4.13, this seems to be a reasonable conclusion.

Demonstrated in section 4.2, our method for computing repelling LCSs appears to work as intended. This notion is further supported by the LCSs obtained for the ABC flows and flow in the Førde fjord conforming well with the computed $\mathcal{U}_0$ domains (see sections 4.3 and 4.4). Interestingly, the conformity between the $\mathcal{U}_0$ domains and LCSs obtained for the steady and unsteady ABC flows (see section 4.3) indicates that, although substantial, the time perturbation in the case of the unsteady flow (illustrated in figure 3.1) did not significantly alter its underlying stretch and strain properties. This result is indicative of the robustness believed to be characteristic to LCSs. Moreover, considering the model velocity data to provide a reasonable approximation of the *actual* oceanic circulation, this also suggests that our computed LCSs provide a good estimate of (some of) the underlying transport barriers in the Førde fjord — a conjecture which warrants further investigation.

Seeing as computing LCSs in three dimensions is a considerably more convoluted process than for two-dimensional flows (compare our method, outlined in chapter 3, to that of e.g. Løken (2017)), investigating whether or not the conceptually simpler quasi-three-dimensional approach of e.g. Blazevski and Haller (2014) (and, to a lesser extent, Oettinger and Haller (2016)) — elaborated upon in greater detail in section 5.4 — yields similar LCSs could certainly

be worthwile. This could reduce the need for computational resources, which in turn would make the detection of LCSs in three-dimensional flow systems more readily available. That being said, the quasi-three-dimensional approaches will likely never be fully capable of encapsulating the peculiarities of three-dimensional surfaces, regardless of which curve fitting algorithm is used in order to extract them.

Overall, strong situational arguments in terms of accuracy requirements or a desire for insight in the fully three-dimensional structure of transport barriers in a given system are needed in order to justify computing LCSs in three dimensions rather than two. Generally, we would advise computing LCSs in three dimensions only for systems in which all three dimensions are of similar relevance; flow in the Førde fjord (a subset of which was considered here) constitutes an example of such a system, as its depth and width are typically similar in magnitude, and significant vertical transport can be observed. Conversely, transport along oceanic surface currents — within which contaminations such as garbage patches or oil spill remnants are frequently transferred — can reasonably be approximated as being two-dimensional.

Pilot studies pertaining to the use of LCSs as predictors for transport by oceanic currents have recently been conducted. For instance, Filippi et al. (2018) performed field experiments on repelling and attracting transport barriers along the Scott Reef in Western Australia, while Peacock et al. (2018) did similar exercises in the vicinity of Martha's Vineyard. Interestingly, Peacock et al. investigated the fully three-dimensional transport characteristics of offshore currents, finding complementary results to that suggested by Lagrangian analysis. This suggests that designing field experiments in order to verify the repelling LCSs computed by the method of chapter 3 is not beyond the realms of possibility — then again, doing so remains beyond the scope of this project.

6 Conclusions

Our method for computing repelling LCSs in three-dimensional flow systems seems to yield reasonable results. Analytically constructed test cases indicate that our method of generating three-dimensional surfaces, in addition to extracting repelling LCSs as subsets of computed manifolds which satisfy all the necessary and sufficient existence criteria, works as intended. Moreover, the robustness of the LCSs obtained in the two variations of the Arnold-Beltrami-Childress flow considered here, suggests that the computed LCSs are not particularly sensitive to imperfect model data — which is a characteristic property of *Lagrangian* transport barriers. Accordingly, our computed LCSs for flow in the Førde fjord likely form reasonable approximations of the actual LCSs contained within.

There is certainly room for further research with regards to the identification of locally most repelling material surfaces. To our knowledge, a general, robust numerical routine for this purpose is yet to be described in the literature. For instance, in a recent study conducted by Oettinger and Haller (2016), who set out to compute quasi-three-dimensional hyperbolic LCSs, the authors seemingly did not attempt to implement the aforementioned criteria numerically, appearing to be satisfied with reasonable suggestions as to where such LCSs *might* be located. Our approach, based on identifying which of the individual points constituting each manifold was locally repelling, involved one parameter which was determined by use of the initial grid spacing, and another which filtered away the smallest LCS surfaces — as these were assumed not to impact the overall circulation significantly (as originally suggested by Farazmand and Haller (2012a)). A suggested alternative approach, which was not investigated as part of this project, would be to utilize a sort of numerical clustering algorithm to extract LCSs as the most repelling material surfaces in small neighborhoods, where each of the considered sets of surfaces ideally would be of similar size and orientation.

Seeing as our method of computing fully three-dimensional LCSs is significantly more complex and consumes more computational resources than well established routines for generating their two-dimensional counterparts (see e.g. Onu, Huhn, and Haller (2015)), strong arguments regarding the additional insight from three-dimensional analysis are needed in order to substantiate a change in practices. Moreover, computing fully three-dimensional LCSs might not be necessary, depending on the type of transport phenomenon under consideration. For instance, regarding the spread of debris and contaminations such as garbage patches or oil spill remnants across the ocean surface, the underlying transport system can reasonably be considered two-dimensional. Meanwhile, for circulation in rivers or fjords, where the depth is of the same scale as the width and vertical transport is often significant, fully three-dimensional analysis might be more suitable. Simply put, for systems in which all three dimensions are of similar relevance — i.e., systems which cannot reasonably be regarded as two-dimensional — computing fully three-dimensional LCS surfaces might be prudent; in contrast to the quasi-three-dimensional approach of Blazevski and Haller (2014), which loses out on the minute details of the extra dimension, and does not yield coherent surfaces favorable for further inquiries.

Suggestions for further work

Experimental methods for the verification of Lagrangian flow analysis have gained traction lately. Investigations are currently being conducted as to the usefulness of LCSs as predictors for diffusive flow patterns, for which analyzing tracer trajectories of the overall circulation might be insufficient (Haller and Karrasch 2018). Moreover, recent pilot studies indicate that Lagrangian analysis yields valid predictions for transport by oceanic currents (Filippi et al. 2018) — including the fully three-dimensional circulation patterns arising in off-shore currents (Peacock et al. 2018). Thus, empirical studies pertaining to the validity of the LCSs computed by use of our variation of the geodesic level set method remain within the realm of feasibility.

In addition to the aforementioned issue of implementing the LCS existence criterion which identifies LCSs as surfaces which are locally most repelling, further research aiming to enhance our approach for computing repelling LCSs could result in increased transparency and efficiency. Specifically, abandoning the strict ordering of mesh points in level sets forming topological circles could allow for an increase in the resolution of LCS behaviour near domain boundaries, and potentially yield more accurate surface approximations overall. Then again, other options for computing invariant manifolds of three-dimensional vector fields exist. Aside from the method of geodesic level sets, Krauskopf, Osinga, et al. (2005) present four others; investigating each of them in the context of computing hyperbolic LCSs could provide valuable insights. Moreover, a method less demanding in terms of resource consumption than the present approach would render the computation of three-dimensional LCSs a feasible task without the need of supercomputers. Courtesy of facilitating three-dimensional Lagrangian analysis for a wider audience, this could, in turn, accelerate the further development of LCS computing tools as a whole.

Lastly, to our knowledge, little research has been conducted regarding the choice of interpolation and integration methods in order to simulate transport phenomena in real-world systems. High-order adaptive step size integration methods are generally used in conjunction with interpolated velocity fields, seemingly with little awareness as to whether or not the use of higher-order methods are warranted. For instance, van Sebille et al. (2018) provide a rigorous discussion of the currently available tools for computer simulated tracer advection, yet do not treat integration methods in detail. The significance of the choice of interpolation scheme is generally recognized, though left largely unexplored. Whereas Lekien and Marsden (2005) developed a (locally) tricubic interpolation method intended for use in simulating three-dimensional flow (and computing three-dimensional LCSs) — motivated by the observation that linear interpolation is largely insufficient in terms of yielding smooth velocity fields from model data — the authors provide no arguments for preferring cubic methods over even higher order alternatives. A third example is the article by Gough et al. (2017), in which the authors report having used a high-order adaptive step size Runge-Kutta solver together with a cubic interpolation routine — much like what was done for this project — in order to investigate oceanic transport patterns in the northwestern Gulf of Mexico; yet fail to motivate their choices of integration and interpolation methods. Based on the above, investigations with regards to the interaction between integration and

interpolation schemes in the context of computing LCSs are appealing, due to their innate relation to application.

References

Ali, S. and Shah, M. (2007). "A Lagrangian Particle Dynamics Approach for Crowd Flow Segmentation and Stability Analysis". In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–6. DOI: 10.1109/CVPR.2007.382977. ISSN: 1063-6919.

Blazevski, D. and Haller, G. (2014). "Hyperbolic and elliptic transport barriers in three-dimensional unsteady flows". In: *Physica D: Nonlinear Phenomena* 273–274, pp. 46–62. ISSN: 0167-2789. DOI: 10.1016/j.physd.2014.01.007.

de Boor, C. (1978). *A Practical Guide to Splines*. 1st ed. Springer-Verlag New York. ISBN: 978-0-387-95366-3.

Farazmand, M. and Haller, G. (2012a). "Computing Lagrangian coherent structures from their variational theory". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 22.1, p. 013128. ISSN: 1054-1500. DOI: 10.1063/1.3690153.

Farazmand, M. and Haller, G. (2012b). "Erratum and addendum to 'A variational theory of hyperbolic Lagrangian coherent structures' [Physica D 240 (2011) 547–598]". In: *Physica D: Nonlinear Phenomena* 241.4, pp. 439–441. ISSN: 0167-2789. DOI: 10.1016/j.physd.2011.09.013.

Filippi, M. et al. (2018). *The detection of Lagrangian Transport Structures in a coral reef atoll*. Paper presented at the Ocean Sciences Meeting, Portland, OR, February 12–16.

Frisch, U. (1995). *Turbulence, the legacy of A.N. Kolmogorov*. 1st ed. Cambridge University Press. ISBN: 978-0-521-45713-2.

Garvik, O. (Dec. 12, 2017). "Gruvekonflikten i Førdefjorden". In: *Store norske leksikon*. Available at https://snl.no/Gruvekonflikten_i_Førdefjorden (accessed May 11, 2018).

Gough, M. K. et al. (2017). "Persistent Lagrangian transport patterns in the northwestern Gulf of Mexico". Unpublished. Preprint available at https://arxiv.org/abs/1710.04027 (retrieved December 19, 2017).

Hairer, E., Nørsett, S. P., and Wanner, G. (1993). *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2nd ed. Springer-Verlag Berlin Heidelberg. ISBN: 978-3-540-56670-0. Corrected 3rd printing, 2008.

Hairer, E. and Wanner, G. (1996). *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. 2nd ed. Springer-Verlag Berlin Heidelberg. ISBN: 978-3-642-05221-7. Corrected 2nd printing, 2002.

Haller, G. and Yuan, G. (2000). "Lagrangian coherent structures and mixing in two-dimensional turbulence". In: *Physica D: Nonlinear Phenomena* 147.3, pp. 352–370. ISSN: 0167-2789. DOI: 10.1016/S0167-2789(00)00142-1.

Haller, G. (2011). "A variational theory of hyperbolic Lagrangian Coherent Structures". In: *Physica D: Nonlinear Phenomena* 240.7, pp. 547–598. ISSN: 0167-2789. DOI: 10.1016/j.physd.2010.11.010.

Haller, G. and Karrasch, D. (2018). *Material Barriers to Diffusive Mixing: Theory*. Paper presented at the Ocean Sciences Meeting, Portland, OR, February 12–16.

Haugan, I. (May 20, 2015). "– Sjødeponi kan være den beste løsningen". In: *forskning.no*. Available at https://forskning.no/forurensning-bergfag-geofag-kjemi/2015/05/sjodeponi-kan-vaere-den-beste-losningen (accessed May 11, 2018).

Karrasch, D. (2012). "Comment on 'A variational theory of hyperbolic Lagrangian coherent structures, Physica D 240 (2011) 574–598'". In: *Physica D: Nonlinear Phenomena* 241.17, pp. 1470–1473. ISSN: 0167-2789. DOI: 10.1016/j.physd.2012.05.008.

Krauskopf, B. and Osinga, H. M. (2003). "Computing Geodesic Level Sets on Global (Un)stable Manifolds of Vector Fields". In: *SIAM Journal on Applied Dynamical Systems* 2.4, pp. 546–569. ISSN: 1536-0040. DOI: 10.1137/030600180.

Krauskopf, B., Osinga, H. M., et al. (2005). "A Survey of Methods for Computing (un)Stable Manifolds of Vector Fields". In: *International Journal of Bifurcation and Chaos* 15.3, pp. 763–791. ISSN: 1793-6551. DOI: 10.1142/S0218127405012533.

Lekien, F. and Marsden, J. E. (2005). "Tricubic interpolation in three dimensions". In: *International Journal for Numerical Methods in Engineering* 63.3, pp. 455–471. ISSN: 0029-5981. DOI: 10.1002/nme.1296.

Løken, A. M. T. (2017). "Sensitivity to Numerical Integration Scheme in Calculation of Lagrangian Coherent Structures". Unpublished specialization project. Available at http://folk.ntnu.no/amloken/sensitivity_to_numerical_integration_scheme_loken.pdf.

Miron, P. et al. (2012). "Anisotropic mesh adaption on Lagrangian Coherent Structures". In: *Journal of Computational Physics* 231.19, pp. 6419–6437. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2012.06.015.

Möller, T. and Trumbore, B. (1997). "Fast, Minimum Storage Ray-Triangle Intersection". In: *Journal of Graphics Tools* 2.1, pp. 21–28. DOI: 10.1080/10867651.1997.10487468.

Oettinger, D. and Haller, G. (2016). "An autonomous dynamical system captures all LCSs in three-dimensional unsteady flows". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 26.10. ISSN: 1054-1500. DOI: 10.1063/1.4965026.

Olascoaga, M. et al. (2008). "Tracing the early development of harmful algal blooms on the West Florida Shelf with the aid of Lagrangian coherent structures". In: *Journal of Geophysical Research: Oceans* 113.C12. ISSN: 0148-0227. DOI: 10.1029/2007JC004533.

Onu, K., Huhn, F., and Haller, G. (2015). "LCS Tool: A computational platform for Lagrangian coherent structures". In: *Journal of Computational Science* 7, pp. 26–36. ISSN: 1877-7503. DOI: 10.1016/j.jocs.2014.12.002.

Peacock, T. et al. (2018). *Targeted drifter deployments around Martha's Vineyard to uncover Lagrangian transport structures.* Paper presented at the Ocean Sciences Meeting, Portland, OR, February 12–16.

Prince, P. and Dormand, J. (1981). "High order embedded Runge-Kutta formulae". In: *Journal of Computational and Applied Mathematics* 7.1, pp. 67–75. ISSN: 0377-0427. DOI: 10.1016/0771-050X(81)90010-3.

Slagstad, D. and McClimans, T. A. (2005). "Modeling the ecosystem dynamics of the Barents sea including the marginal ice zone: I. Physical and chemical oceanography". In: *Journal of Marine Systems* 58.1, pp. 1–18. ISSN: 0924-7963. DOI: 10.1016/j.jmarsys.2005.05.005.

Stoer, J. and Bulirsch, R. (2002). *Introduction to Numerical Analysis.* 3rd ed. Springer-Verlag New York. ISBN: 978-1-4419-3006-4.

Strogatz, S. H. (2014). *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering.* 2nd ed. Westview press, Colorado. ISBN: 978-0-813-34910-7.

van Sebille, E. et al. (2018). "Lagrangian ocean analysis: Fundamentals and practices". In: *Ocean Modelling* 121, pp. 47–75. ISSN: 1463-5003. DOI: 10.1016/j.ocemod.2017.11.008.

Williams, J. (2018). *Bspline-Fortran: Multidimensional B-Spline Interpolation of Data on a Regular Grid*. Zenodo. DOI: 10.5281/zenodo.1215289.

Zhao, X.-H. et al. (1993). "Chaotic and Resonant Streamlines in the ABC flow". In: *SIAM Journal on Applied Mathematics* 53.1, pp. 71–77. ISSN: 0036-1399. DOI: 10.1137/0153005.