Preface

The following is my master's thesis, written at the NTNU Faculty of Natural Sciences as a part of my master's degree in Applied Physics and Mathematics. Starting in January and spanning over the 2018 spring semester, the accompanying project work was carried out for SINTEF Ocean and supervised by Associate Professor Tor Nordam.

Having previously cooperated with Assoc. Prof. Nordam, considering interpolator performance in relation to identification of hyperbolic Lagrangian coherent structures (LCSs) in two dimensions, Tor approached me and fellow student Arne Magnus Tveita Løken with the present problem statement. Acknowledging the complexities of LCS identification in three dimensions, this project has been carried out in cooperation with Arne. Although writing separate theses, it should be noted that we devised all methods and computed all results collaboratively. The guidance and support, as well as enthusiastic disposition, of Associate Professor Nordam has also been a constant source of help, inspiration, and motivation throughout the process

The intended audience of this report possesses a level of knowledge of physics and mathematics corresponding to that expected at a master's level physics program. While in-depth knowledge of finite strain theory and numerical analysis could prove helpful, all advanced methods and concepts are explained prior to application. The code used to produce the results presented in this thesis is available at github.[1]

Trondheim, 2018-06-11

Simon Nordgreen

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

Acknowledgments

First, I would like to extend my sincerest thanks to my supervisor Tor Nordam who provided me with this interesting problem, as well as competent and enthusiastic guidance throughout the project. Also of great importance has been the collaboration, discussion, and help I have received from fellow student Arne Magnus Tveita Løken. Your contributions have been invaluable. Finally, I would like to thank my friends and family who continue to support me in everything I do.

S.N.

Abstract

Lagrangian coherent structures (LCSs) are dynamic surfaces that shape the flow patterns in complex transport systems. Accurate identification of LCSs has applications in particle transport, including dissemination of oil in water following major oil spills and airborne ash after volcanic eruptions. While methods exist for identifying LCSs by use of their variational theory in two-dimensional systems, limited effort has previously been given to extending these methods to three dimensions. Where some systems are reasonably approximated as two-dimensional, LCSs in other firmly three-dimensional systems have commonly been computed by combining two-dimensional cross sections.

Aiming to develop a dedicated method for computing three-dimensional hyperbolic LCSs, this study combines existing LCS theory with recognized methods for computing three-dimensional manifolds. This approach is outlined with respect to its theoretical foundation, as well as resource management and accuracy concerns. Moreover, reference test cases are used to confirm method accuracy in terms of reproducing simple manifolds and LCSs. Inquiring as to the robustness of these LCSs, we also apply the method to a well-known three-dimensional steady velocity field, using a time-perturbed variation of the same field for comparison. The high degree of similarity between the resulting LCSs indicate a high degree of LCS robustness. This robustness is considered desirable, as it suggests low sensitivity to the velocity field inaccuracies associated with commonly used gridded data models for flow fields.

While these results seem promising, it remains to be seen whether the three-dimensional approach is sufficiently advantageous in terms of accuracy and descriptive power to justify the corresponding disadvantages with respect to complexity and resource requirements.

Contents

Chapter 1

Introduction

Consider a highly complex transport system such as the oceanic currents or atmospheric winds. Although the fundamental dynamics of these systems are well-known, their immense complexity impedes accurate description. The main obstacles to our understanding of complex transport systems are usually enormous computational requirements and limited data availability. This prompts us to approach these transport problems in a less ambitious manner, simply aiming to understand the macro-level structures of the underlying system. In many cases, such an approach could prove sufficient, as the idiosyncrasies of individual particle flow trajectories are insignificant in the context of most practical applications.

Over the course of the past two decades, the concept of Lagrangian coherent structures (LCSs) has been proposed and developed as a tool for understanding complex flow systems on a macro-level. LCSs may be seen as the overarching structures, or skeletons, governing the macro-level behavior of transport systems. Specifically, LCSs are the most repelling, attracting or shearing formations that shape tracer trajectory patterns in unsteady dynamical systems. Furthermore, LCSs may be divided into three main categories, namely: hyperbolic, parabolic, and elliptic. Where parabolic and elliptic LCSs respectively describe maximally shearing material lines and Lagrangian vortices, hyperbolic LCSs represent maximally repelling or attracting material surfaces (Haller, 2015).

Hyperbolic LCSs are of particular interest with respect to application. This is because transport barriers — often identified as hyperbolic LCSs through which no particle may move within a specific timespan — are considered particularly useful in terms of anticipating key system characteristics. Likely areas of application include predicting the spread of oil spills in oceanic currents to aid cleanup efforts. Similarly, forecasting dissemination of volcanic ash could provide airline companies with an opportunity to divert exposed flights.

Although significant work has been done detailing methods for LCS identification for two-dimensional transport systems, little work has been dedicated to extending these methods to three dimensions. It has been common practice to identify LCSs in three-dimensional systems

by computing a set of two-dimensional LCSs, merging them into coherent surfaces by use of some interpolation scheme (Oettinger and Haller, 2016). This approach is outlined by for example Blazevski and Haller (2014). Although allowing us to investigate system dynamics in three dimensions, this method ignores transport orthogonal to these two-dimensional system slices.

While an adequate theoretical foundation for three-dimensional LCS theory exists, implementing three-dimensional methods for identification of hyperbolic LCSs is impeded both by increased complexity and hardware requirements. Moreover, many important transport systems may reasonably be approximated as two-dimensional, neglecting dynamics of a subordinate axis. The extent to which it is beneficial to replace two-dimensional methods for identification of LCSs with their three-dimensional counterparts is therefore still unknown and probably case dependent.

This study aims to utilize the descriptions of LCSs provided by Haller (2011) and Oettinger and Haller (2016), as well as the method of geodesic levelsets first described by Krauskopf and Osinga (2003), to compute three-dimensional hyperbolic LCSs. Hoping to demonstrate the efficacy of this approach, several reference results are computed, as well as a case based on modelled ocean currents.

1.1 Background

Coined in 2000 by Haller and Yuan, the term "Lagrangian coherent structure" (LCS) refers to the overarching structures framing chaotic flow system behavior. LCS theory has over the past two decades been developed as an alternative to traditional transport system modelling for systems exhibiting high sensitivity to initial conditions. Early on, identification of hyperbolic LCSs was focused on utilizing finite-time Lyapunov exponent (FTLE) fields to identify surfaces forming local extrema with regard to repulsion or attraction. A thorough and rigorous description of the FTLE approach to identification of hyperbolic LCSs may be found in Shadden et al. (2005). However, Haller (2011) demonstrates that this approach produces both false positives and false negatives, making the case for a variational method. A numerical implementation of this method for two-dimensional systems is outlined in Farazmand and Haller (2012a).

Noting that LCSs in three-dimensional systems have so far commonly been computed by combining LCSs in two-dimensional domain cross sections, Oettinger and Haller (2016) argue for using an autonomous dynamical system to compute trajectories within the target LCS surfaces. While improving upon existing methods in terms of truly acknowledging the three-dimensional dynamics of these systems, Oettinger and Haller (2016) seem content with simply computing surfaces within which LCSs may exist. Inspired by Oettinger and

Haller (2016), this study aims to combine the use of this autonomous dynamical system with the ideas of Haller (2011), as well as dedicated methods of computing manifolds in three dimensions. Hoping to develop a method for identifying three-dimensional hyperbolic LCSs, this investigation is intended as a reasonable extrapolation of Farazmand and Haller (2012a)'s method to three-dimensional systems.

Chapter 2

Theoretical Background

Providing the needed background for all subsequent considerations, the current chapter outlines the concepts of numerical analysis and LCS theory used throughout this study. In particular, this pertains to solution of ordinary differential equations by use of Runge-Kutta methods, spline interpolation, finite strain theory, and the theoretical foundation of hyperbolic LCSs. Finally, the latter concepts will be used to pose existence criteria, as well as some key characteristics for hyperbolic LCSs.

2.1 Numerically solving ordinary differential equations

Particle transport problems are often governed by sets of ordinary differential equations (ODEs) of the form

$$\dot{\mathbf{x}} = \mathbf{v}(t, \mathbf{x}), \tag{2.1}$$

where $\mathbf{v}$ is the velocity field function, t is time, and $\mathbf{x}$ and $\dot{\mathbf{x}}$ are the particle position and corresponding time derivative, respectively. Solving equation (2.1) amounts to finding $\mathbf{x}(t)$. As only a few exceptional ODE systems may be solved analytically, most cases require numerical solution by use of a numerical ODE solver. The simplest and most intuitive of these is the Euler method

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{v}(t_n, \mathbf{x}_n)\Delta t, \tag{2.2}$$

where t_n is the n^{th} time step and $\mathbf{x}_n$ is an approximation of $\mathbf{x}(t_n)$; the exact solution of equation (2.1) at time t_n. Note that $t_n = t_0 + n\Delta t$, where t_0 is the start time and Δt is the time step length. Solving an ODE such as equation (2.1) numerically to find an approximation of $\mathbf{x}(t)$, is known as numerical integration of the ODE.

2.1.1 Runge-Kutta iterative ordinary differential equation solvers

Several of the most common numerical integrators are members of the Runge-Kutta family of explicit iterative methods. In the two-variable case (e.g. t, x), these methods are of the form

$$x_{n+1} = x_n + \Delta t(b_1 k_1 + ... + b_s k_s). \tag{2.3}$$

Here, x_n and x_{n+1} are the current and coming iterations of the function value, respectively. Intermediate time step slope evaluations are denoted by k_i, while b_i are the corresponding weighting coefficients. The number of stages and order of the method are denoted s and p, respectively. Note that in the case of ODEs, the two-variable case may easily be extended by treating multiple dependent variables separately.

The Runge-Kutta explicit iterative methods may be considered as a generalization of the classical 4^{th}-order Runge-Kutta method to any order. The 4^{th}-order classical Runge-Kutta method, for an ODE $\dot{x} = f(t, x)$, is given by

$$\begin{aligned}
x_{n+1} &= x_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\
t_{n+1} &= t_n + \Delta t, \\
k_1 &= f(t_n, x_n), \\
k_2 &= f(t_n + \frac{\Delta t}{2}, x_n + \frac{k_1}{2}\Delta t), \\
k_3 &= f(t_n + \frac{\Delta t}{2}, x_n + \frac{k_2}{2}\Delta t), \\
k_4 &= f(t_n + \Delta t, x_n + k_3 \Delta t),
\end{aligned} \tag{2.4}$$

and is commonly used for a wide range of applications due to its 4^{th}-order accuracy, ease of implementation, and moderate computational requirements. Specifically, Runge-Kutta methods of order $p > 4$ require a larger number of stages s than their order, yielding a reduced accuracy gain per additional function evaluation (Hairer et al., 2008).

2.1.2 Runge-Kutta method error bounds

Explicit Runge-Kutta methods of order p are defined by their partial derivatives matching those of the underlying analytical solution up to and including order p (Hairer et al., 2008). This attribute has implications for their error bounds in a single step, usually referred to as local error. The local error $e(\Delta t)$ is defined as

$$e(\Delta t) = x(t_0 + \Delta t) - x_1, \tag{2.5}$$

where t_0 may be identified as the time corresponding to the previous function value; $x(t_0) = x_0$. As outlined in Bieberbach (1951), this may be analyzed by substituting equation (2.3) into equation (2.5)

$$e(\Delta t) = x(t_0 + \Delta t) - x_0 - \Delta t \sum_{i=1}^{s} b_i k_i \tag{2.6}$$

and Taylor expanding

$$x(t_0 + \Delta t) = x_0 + x'(t_0)\Delta t + x''(t_0)\frac{\Delta t^2}{2!} + \ldots + x^{(p+1)}(t_0 + q\Delta t)\frac{\Delta t^{p+1}}{(p+1)!}, \tag{2.7}$$

$$k_i(\Delta t) = k_i(0) + k_i'(0)\Delta t + \ldots + k_i^{(p)}(q_i\Delta t)\frac{\Delta t^p}{p!}, \tag{2.8}$$

with $0 < q < 1$ and $0 < q_i < 1$. Note that due to the partial derivatives up to and including order p being identical, we are left with the terms in the Taylor expansions corresponding to x and k of orders including and exceeding $p+1$ and p, respectively.

Consequently, if a Runge-Kutta method of order p is applied to a function $f(t, x)$ with existing and continuous partial derivatives up to order p, then the local error is strictly bounded by

$$|x(t_0 + \Delta t) - x_1| \leq \Delta t^{p+1} \left(\frac{1}{(p+1)!} \max_{q\in[0,1]} \left|x^{(p+1)}(t_0 + q\Delta t)\right| + \frac{1}{p!} \sum_{i=1}^{s} |b_i| \max_{q\in[0,1]} \left|k_i^{(p)}(q\Delta t)\right| \right), \tag{2.9}$$

which gives

$$|e(\Delta t)| = |x(t_0 + \Delta t) - x_1| \leq C\Delta t^{p+1} \tag{2.10}$$

for the order of the Runge-Kutta method local error, where C is some constant.

Given the iterative nature of Runge-Kutta methods, the way in which their local error compounds into global error over n consecutive time steps is of great interest. The global error of a numerical solution is the deviation from the analytical solution after several steps given by

$$E = x(t_n) - x_n, \tag{2.11}$$

where x_n is obtained by n successive iterative steps from x_0 and $x(t_n)$ is the analytical solution evaluated at t_n. As described in detail in Hairer et al. (2008); if the local error of an iteration of a Runge-Kutta method satisfies equation (2.10), then we have for the global error

$$|E| \leq \tilde{C}\Delta t^p, \tag{2.12}$$

where $\tilde{C}$ again is a constant, in general different from C.

2.1.3 Adaptive step Runge-Kutta methods

The previously described ODE solvers are all constant step length methods. That is, we choose a step length, manually or otherwise, balancing accuracy requirements with performance restrictions. This step length is then used throughout the entire iterative solution, unless a step length change is explicitly specified. This is problematic as the local behavior of our target solution may require varying step lengths at different points in our computation. As we have no *a priori* knowledge of this possibly changing local behavior, manually selecting an optimal step length is impractical.

The idea of adaptive step methods is to constantly estimate the local error of our chosen iterative solver. By defining a local error tolerance level ϵ_{tol}, we can then evaluate whether the chosen step length was in fact adequate for this specific phase of our calculation. If the local error is estimated to exceed our tolerance level, we reject and recompute the current step with a shorter time step, while a comparatively small error prompts the solver to accept the step and increase the step length for the subsequent iteration.

An error estimate may be computed by using the same discretization method for two different step lengths. Alternatively, we may use one step length and two different Runge-Kutta discretization methods of orders p and $p+1$. These are called *Runge-Kutta-Fehlberg methods* (Stoer and Bulirsch, 1996) and have the form

$$\begin{aligned} \widehat{x}_{n+1} &= \bar{x}_n + \Delta t \phi_I(t_n, \bar{x}_n; \Delta t), \\ \bar{x}_{n+1} &= \bar{x}_n + \Delta t \phi_{II}(t_n, \bar{x}_n; \Delta t), \end{aligned} \tag{2.13}$$

where $\phi_I(t_n, \bar{x}_n; \Delta t)$ and $\phi_{II}(t_n, \bar{x}_n; \Delta t)$ are Runge-Kutta methods of order p and $p+1$, respectively. We implement step length control by considering the difference

$$\bar{x}_{n+1} - \widehat{x}_{n+1} = \Delta t\big(\phi_{II}(t_n, \bar{x}_n; \Delta t) - \phi_I(t_n, \bar{x}_n; \Delta t)\big). \tag{2.14}$$

It follows from equation (2.10) that ϕ_I and ϕ_{II} have local errors of order p and $p+1$, respectively. We may therefore, for small Δt, express the difference in equation (2.14) as

$$\bar{x}_{n+1} - \widehat{x}_{n+1} \approx C(t_n)\Delta t^{p+1}, \tag{2.15}$$

where $C(t)$ is some function. Neglecting higher order error terms, we now use equation (2.15) as an approximation for the local error associated with using $\phi_I(t_n, \bar{x}_n; \Delta t)$.

Suppose that we just completed a successful step, that is, we have

$$|\bar{x}_{n+1} - \widehat{x}_{n+1}| \approx \left|C(t_n)\Delta t^{p+1}\right| \leq \epsilon_{\text{tol}}. \tag{2.16}$$

Now, by assuming

$$C(t_n) \approx C(t_{n+1}) \approx \frac{|\bar{x}_{n+1} - \widehat{x}_{n+1}|}{|\Delta t^{p+1}|}, \tag{2.17}$$

we can approximate our condition (2.16) by

$$|\bar{x}_{n+1} - \widehat{x}_{n+1}| \left|\frac{\Delta t_{\text{new}}}{\Delta t}\right|^{p+1} \leq \epsilon_{\text{tol}}. \tag{2.18}$$

Finally, we estimate the new step length by isolating Δt_{new} according to

$$\Delta t_{\text{new}} = \Delta t \left|\frac{\epsilon_{\text{tol}}}{\bar{x}_{n+1} - \widehat{x}_{n+1}}\right|^{1/(p+1)}. \tag{2.19}$$

Whether $\bar{x}$ or $\widehat{x}$ is used for the actual solution step depends on the particular method implementation. Several adaptive step methods are available in literature. The widely used Dormand-Prince method of orders $p = 4$ and 5 is outlined by for example Stoer and Bulirsch (1996) and Dormand and Prince (1980). A higher order alternative, corresponding to orders $p = 7$ and 8, is described by Prince and Dormand (1981).

2.1.4 Interpolation

Given that real world transport systems are known only by partial measurement or grid based model output, considering the trajectories of particles moving between these sampling or grid points necessitates use of interpolation. A two-dimensional interpolation problem may be described by considering the family of functions

$$\Phi(x, y; a_0, \ldots, a_n), \tag{2.20}$$

each characterized by the $n + 1$ parameters $a_0, \ldots, a_n$. Having been given a set of $n + 1$ coordinates and corresponding function values (x_i, y_i, f_i), where $x_i \neq x_k$ for $i \neq k$ and $f_i = f(x_i, y_i)$, the interpolation problem amounts to determining the set of parameters $\{a_i\}_{i=0}^n$ as to make Φ satisfy

$$\Phi(x_i, y_i; a_0, \ldots, a_n) = f_i, \quad i = 0, ..., n. \tag{2.21}$$

Here, we name the coordinates (x_i, y_i), function values (f_i), and points (x_i, y_i, f_i) support abscissas, support ordinates, and support points, respectively. Moreover, as long as Φ depends linearly on the set of parameters a_i, and may be written in the form

$$\Phi(x, y; a_0, ..., a_n) = a_0\Phi_0(x, y) + a_1\Phi_1(x, y) + ... + a_n\Phi_n(x, y), \tag{2.22}$$

this may be classified as a linear interpolation problem. According to Stoer and Bulirsch (1996), the linear class of interpolation problems includes among others polynomial interpolation, trigonometric interpolation, and spline interpolation.

An interpolation problem is solved through spline interpolation by determining the set of parameters $\{a_i\}_{i=0}^n$ in equation (2.22) with the set of corresponding functions $\{\Phi_i\}_{i=0}^n$ limited to spline functions. These spline functions, also simply referred to as splines, are connected by use of a partition. Considering the one-dimensional case for simplicity, the partition

$$\Delta : \quad a = x_0 < x_1 < ... < x_n = b \tag{2.23}$$

of the interval $[a, b]$ gives the domains of the piecewise polynomial spline functions S in the set S_Δ. These spline functions are connected at support abscissas, in the context of splines called knots. Order k spline functions S_k may be defined as piecewise polynomial functions of order k that are $k-1$ times differentiable at all interior knots of Δ (that is, x_i for $1 \leq i \leq n-1$) (Stoer and Bulirsch, 1996). These spline functions are entirely defined by their $k+1$ coefficients, determined by the $k-1$ derivatives and the function value at their left bordering knot, as well as the function value at their right bordering knot. In order to properly define splines in the exterior subdomains $[x_0, x_1]$ and $[x_{n-1}, x_n]$, we impose boundary conditions. As the splines are composed of polynomials, the $k-1$ times differentiability condition at each knot ensures the resulting spline interpolation Φ is $k-1$ times differentiable at all points.

A special class of these piecewise polynomial functions, B-splines have some very useful properties. That is, B-splines are nonnegative and evaluate to zero everywhere except the contiguous interval $[x_i, x_{i+1}]$ (Stoer and Bulirsch, 1996). This makes B-splines ideal for spline interpolation.

A selection of spline interpolators applied to a discretely sampled 5^{th}-order polynomial is displayed in figure 2.1, highlighting the increased accuracy of higher order splines when applied to continuous functions. Higher order spline interpolation schemes do however require more input. Specifically, a k^{th} order spline requires at least $k+1$ input coordinates. According to Stoer and Bulirsch (1996), spline functions have seen increasing use in numerical methods

due to yielding smooth interpolating curves with a limited prevalence of oscillations for higher order polynomials.

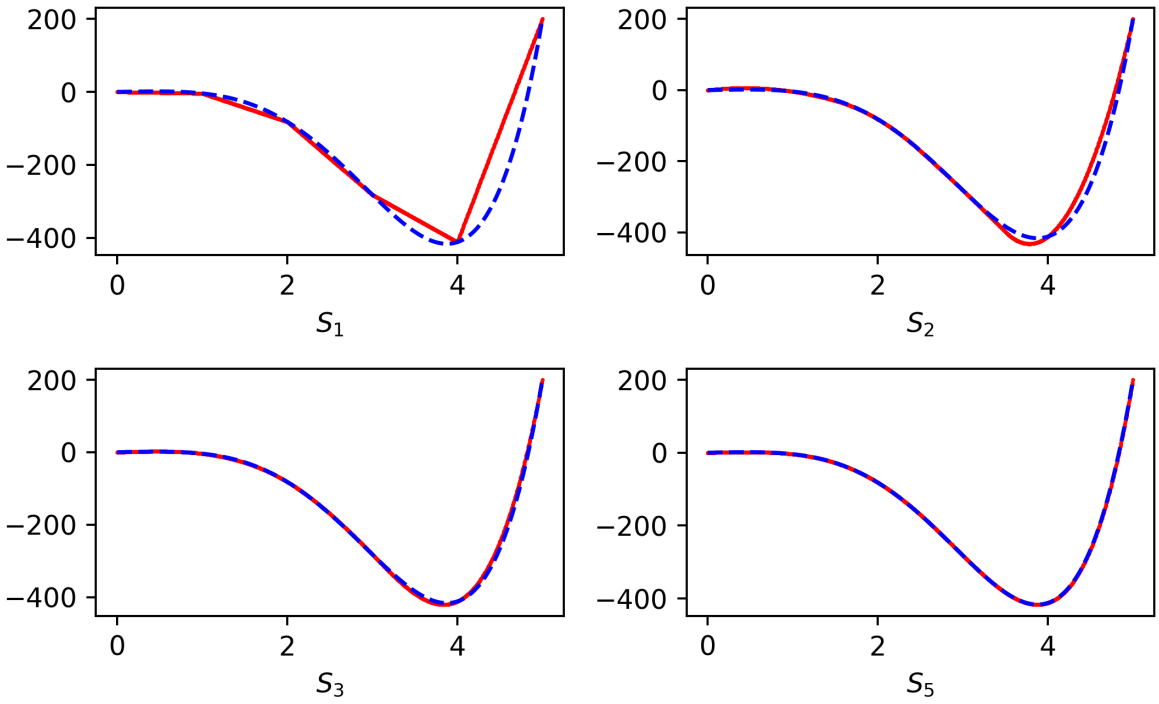


Figure 2.1: Spline interpolation of orders 1, 2, 3, and 5 in solid red applied to a discretely sampled fifth order polynomial (blue dashed line). We observe that higher order splines yield more accurate interpolations.

2.2 Transport systems

LCS theory may be seen as a set of tools developed to gain useful insight into finite-time transport systems that may be too sensitive to initial conditions to be accurately described by conventional numerical modelling. Correspondingly, LCS theory deals with the transport of passive tracers in velocity fields. Tracers are infinitesimal particles that follow the currents of the system without influencing it, or each other, in any way. The underlying velocity field is usually given, as the method by which this is obtained is inconsequential to the characteristics of the system. Although the scope of the current investigation primarily treats the three-dimensional case, considering the more well-known two-dimensional case is useful in terms of outlining theoretical concepts.

2.2.1 Transport system description and notation

Consider a transport problem in a finite-time three-dimensional unsteady velocity field of the form

$$\dot{\mathbf{x}} = \mathbf{v}(t, \mathbf{x}), \quad \mathbf{x} \in U, \quad t \in [t_0, t_0 + T], \tag{2.24}$$

where $\mathbf{x}$ is position, t denotes time, $\mathbf{v}$ is the velocity field function, U is the function domain, and t_0 and $t_0 + T$ are the start and stop times, respectively. Considering a particle in the system (2.24) of initial position $\mathbf{x}(t_0) = \mathbf{x}_0$, we denote its subsequent trajectory by

$$\mathbf{x}(t; t_0, \mathbf{x}_0). \tag{2.25}$$

We then define the flow map $\mathbf{F}_{t_0}^{t}$ as

$$\mathbf{F}_{t_0}^{t}(\mathbf{x}_0) \coloneqq \mathbf{x}(t; t_0, \mathbf{x}_0), \tag{2.26}$$

mapping the set of initial positions in U at time t_0 to the corresponding positions at time t. As noted in Haller (2015), the flow map retains the smoothness of the underlying velocity field.

2.2.2 Deformation

As a set of passive tracer particles are transported, neighboring particles are likely to either diverge or converge at various rates based on the local properties of the velocity field. These local rates of repulsion or attraction may be described and quantified by considering the flow maps of particles with nearly identical initial conditions. Specifically, we consider the flow map three-dimensional Jacobian, or flow gradient, given by

$$\nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial x}{\partial x_0} & \frac{\partial x}{\partial y_0} & \frac{\partial x}{\partial z_0} \\ \frac{\partial y}{\partial x_0} & \frac{\partial y}{\partial y_0} & \frac{\partial y}{\partial z_0} \\ \frac{\partial z}{\partial x_0} & \frac{\partial z}{\partial y_0} & \frac{\partial z}{\partial z_0} \end{bmatrix}, \tag{2.27}$$

where $\frac{\partial}{\partial x_0}$ etc. denotes differentiation with respect to initial position. The flow map Jacobian provides a measure for the local strain of the set of tracer initial positions.

Consider a small perturbation or initial position deviation $\boldsymbol{\epsilon}(t) = \mathbf{x}_2(t) - \mathbf{x}_1(t)$. Taking the time derivative given by equation (2.24) with positions expressed by equation (2.25) and using the Jacobian of the velocity field, we get

$$\dot{\boldsymbol{\epsilon}} = \nabla \mathbf{v}(t, \mathbf{x}(t; t_0, \mathbf{x}_0))\boldsymbol{\epsilon}. \tag{2.28}$$

According to Haller (2015), this perturbation may be expressed as

$$\boldsymbol{\epsilon}(t) = \nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)\boldsymbol{\epsilon}(t_0), \tag{2.29}$$

seeing as $\nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)$ is the fundamental matrix solution of equation (2.28). We therefore obtain, for the squared magnitude of the perturbation, at time t

$$|\boldsymbol{\epsilon}(t)|^2 = \langle \nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)\boldsymbol{\epsilon}(t_0), \nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)\boldsymbol{\epsilon}(t_0) \rangle, \tag{2.30}$$

where $\langle \mathbf{A}, \mathbf{B} \rangle$ signifies taking the inner product of $\mathbf{A}$ and $\mathbf{B}$. This allows us to define the right Cauchy-Green strain tensor $\mathbf{C}_{t_0}^{t}(\mathbf{x}_0)$ as (Truesdell and Noll, 2004)

$$\mathbf{C}_{t_0}^{t}(\mathbf{x}_0) = \left[\nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)\right]^* \nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0), \tag{2.31}$$

yielding

$$|\boldsymbol{\epsilon}(t)|^2 = \left\langle \boldsymbol{\epsilon}(t_0), \mathbf{C}_{t_0}^{t}(\mathbf{x}_0)\boldsymbol{\epsilon}(t_0) \right\rangle, \tag{2.32}$$

where $[...]^*$ signifies matrix transposition. Hence, the Cauchy-Green strain tensor maps position perturbations at time t_0 to their magnitudes at a later time t. Due to the invertibility of $\nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)$, the Cauchy-Green strain tensor is positive definite (Haller, 2015). Furthermore, in three dimensions, it satisfies the following eigenvector and eigenvalue relations:

$$\begin{aligned} \mathbf{C}_{t_0}^{t}(\mathbf{x}_0)\boldsymbol{\xi}_i(\mathbf{x}_0) = \lambda_i(\mathbf{x}_0)\boldsymbol{\xi}_i(\mathbf{x}_0), \quad |\boldsymbol{\xi}_i(\mathbf{x}_0)| = 1, \quad i = 1, 2, 3, \\ 0 < \lambda_1(\mathbf{x}_0) \leq \lambda_2(\mathbf{x}_0) \leq \lambda_3(\mathbf{x}_0), \\ \boldsymbol{\xi}_1(\mathbf{x}_0) \perp \boldsymbol{\xi}_2(\mathbf{x}_0), \\ \boldsymbol{\xi}_1(\mathbf{x}_0) \perp \boldsymbol{\xi}_3(\mathbf{x}_0), \\ \boldsymbol{\xi}_2(\mathbf{x}_0) \perp \boldsymbol{\xi}_3(\mathbf{x}_0), \end{aligned} \tag{2.33}$$

where λ_i and $\boldsymbol{\xi}_i$ are the eigenvalues and eigenvectors of $\mathbf{C}_{t_0}^{t}$, respectively (Haller, 2011). The dependence of λ_i and $\boldsymbol{\xi}_i$ on t and t_0 has been omitted in the interest of notational simplicity. Note that these quantities describing material deformation satisfy objectivity (see section 2.3). Moreover, for incompressible flow systems ($\mathrm{div}(\mathbf{v}) \equiv 0$), the eigenvalues of $\mathbf{C}_{t_0}^{t}(\mathbf{x}_0)$ also satisfy

$$\lambda_1(\mathbf{x}_0)\lambda_2(\mathbf{x}_0)\lambda_3(\mathbf{x}_0) = 1 \tag{2.34}$$

for all $\mathbf{x}_0$ (Haller, 2015).

2.2.3 Flow map dynamics

The flow map $\mathbf{F}_{t_0}^t(\mathbf{x}_0)$ obeys

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{F}_{t_0}^t(\mathbf{x}_0) = \mathbf{v}\left(t, \mathbf{F}_{t_0}^t(\mathbf{x}_0)\right), \tag{2.35}$$

as tracer positions move in the velocity field $\mathbf{v}$. Useful for analyzing local deformation, the time development of the Jacobian of the flow map $\nabla\mathbf{F}_{t_0}^t(\mathbf{x_0})$ (see equation (2.27)) is governed by (Miron et al., 2012)

$$\frac{\mathrm{d}}{\mathrm{d}t}\nabla\mathbf{F}_{t_0}^t(\mathbf{x_0}) = \nabla\mathbf{v}\left(t, \mathbf{F}_{t_0}^t(\mathbf{x_0})\right)\nabla\mathbf{F}_{t_0}^t(\mathbf{x_0}). \tag{2.36}$$

In Cartesian coordinates, equation (2.36) constitutes 9 equations, each coupled with the three equations corresponding to equation (2.35). Note that equation (2.36) may be expressed in Cartesian form as

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial F_i}{\partial x_j}\right) = \sum_k \frac{\partial v_i}{\partial x_k}\frac{\partial F_k}{\partial x_j}. \tag{2.37}$$

Combined, equations (2.35) and (2.37) form a system of 12 coupled equations, solvable by ordinary ODE methods.

2.2.4 Singular-value decomposition

Computing the Cauchy-Green strain tensor $\mathbf{C}_{t_0}^t(\mathbf{x}_0)$ eigenvectors and eigenvalues constitutes a problem of the form

$$\mathbf{B}^*\mathbf{B}\mathbf{Q} = \mathbf{Q}\mathbf{\Lambda}, \tag{2.38}$$

where $\mathbf{\Lambda}$ is the diagonal matrix

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}. \tag{2.39}$$

Here, λ_i are the eigenvalues of $\mathbf{B}^*\mathbf{B}$, where $\mathbf{B}$ is some 3×3 matrix. Moreover, $\mathbf{Q}$ is another 3×3 matrix, its i^{th} column corresponding to the i^{th} eigenvector of $\mathbf{B}^*\mathbf{B}$, denoted $\boldsymbol{\xi}_i$. This problem may be solved directly by computing $\mathbf{B}^*\mathbf{B}$ and finding its eigenvalues and eigenvectors. However, according to Watkins (2005), some information about the smaller eigenvalues is

lost when computing $\mathbf{B}^*\mathbf{B}$ in floating-point arithmetic. Instead, Watkins (2005) suggests utilizing singular-value decomposition.

Singular-value decomposition may be seen as a generalization of eigendecomposition. That is, as the eigendecomposition

$$\mathbf{B}^*\mathbf{B} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}, \tag{2.40}$$

follows from (2.38), the more general singular-value decomposition follows from the relation

$$\mathbf{B}\mathbf{v}_i = \sigma_i \mathbf{u}_i. \tag{2.41}$$

Here, σ_i is the i^{th} singular value of $\mathbf{B}$, while $\mathbf{v}_i$ and $\mathbf{u}_i$ are the corresponding normalized right and left singular vectors. This relation may also be rewritten as

$$\mathbf{B}\mathbf{V} = \mathbf{U}\mathbf{\Sigma}, \tag{2.42}$$

where $\mathbf{v}_i$ and $\mathbf{u}_i$ form the i^{th} columns of the orthogonal matrices $\mathbf{V}$ and $\mathbf{U}$, respectively. Note that for any orthogonal matrix $\mathbf{A}$, we have $\mathbf{A}^* = \mathbf{A}^{-1}$. Moreover, $\mathbf{\Sigma}$ is of the form

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}, \tag{2.43}$$

where σ_i are the positive and real singular values of $\mathbf{B}$ (Trefethen and Bau, 1997). As $\mathbf{V}$ is orthogonal, we may multiply equation (2.42) by its inverse $\mathbf{V}^*$ from the right to obtain

$$\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*. \tag{2.44}$$

This factorization corresponds to the most commonly used form of singular-value decomposition (SVD) (Trefethen and Bau, 1997). Now, consider our problem of computing the eigenvalues and eigenvectors of $\mathbf{B}^*\mathbf{B}$. By insertion of equation (2.44), we get

$$\mathbf{B}^*\mathbf{B} = \mathbf{V}\mathbf{\Sigma}^*\mathbf{U}^*\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \mathbf{V}(\mathbf{\Sigma}^*\mathbf{\Sigma})\mathbf{V}^*, \tag{2.45}$$

where we used that $\mathbf{U}^*\mathbf{U} = \mathbf{I}$ for the orthogonal matrix $\mathbf{U}$. Consequently, we may identify the eigenvalues λ_i of $\mathbf{B}^*\mathbf{B}$ as σ_i^2. Moreover, by comparison of equation (2.45) with equations (2.40) and (2.38), we notice that the columns of $\mathbf{V}$ correspond to the eigenvectors of $\mathbf{B}^*\mathbf{B}$.

Therefore, by performing singular-value decomposition on the matrix $\mathbf{B}$, we are able to implicitly compute the eigenvalues and eigenvectors of $\mathbf{B}^*\mathbf{B}$. As this is done without actually computing $\mathbf{B}^*\mathbf{B}$, we limit numerical error due to use of floating-point arithmetic. In

the context of deformation, this means that we may use $\nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)$ to compute the eigenvalues and eigenvectors of $\mathbf{C}_{t_0}^{t}(\mathbf{x}_0) = \left[\nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)\right]^* \nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)$ directly.

2.3 Hyperbolic Lagrangian coherent structures

Inspired by coherent tracer patterns observed both experimentally and in nature, Lagrangian coherent structure (LCS) theory emerged in the interface between nonlinear dynamics and fluid mechanics as an alternative way of approaching and understanding transport in complex fluid flow systems. Coined by Haller and Yuan (2000), the term Lagrangian coherent structure refers to the most repelling, attracting or shearing material surfaces that guide the principal structures of particle transport dynamics (Haller, 2015). Here, material surfaces refer to time-evolving structures in a flow system, traceable by a set of particles following the set of point trajectories defining the surface. As necessitated by their material surface nature, LCS theory takes a Lagrangian perspective where the identities of individual particles are tracked as they are advected in a velocity field. This is opposed to the Eulerian perspective, where we consider the behavior of the flow field at stationary points in space. Here, particles are simply viewed as realizations of the velocity field as they pass points in space, their identities therefore irrelevant.

The concept of objectivity is also central to the development of LCS theory. Objectivity is one of the fundamental axioms of mechanics and requires that any material response is independent of the perspective of the observer. This makes it necessary to for example take centrifugal forces and the Coriolis effect into account when calculating flow velocity fields on a rotating planet.

Motivated by application in real world systems, LCS theory deals exclusively with finite-time systems. Here, asymptotic concepts from nonlinear dynamics such as stable and unstable manifolds lose their meaning, necessitating the use of finite-time tools. The three main types of LCSs; hyperbolic, elliptic and parabolic, are examples of such finite-time tools used to gain insights into overarching flow patterns. According to Onu et al. (2015), hyperbolic LCSs are defined as the locally most repelling or attracting material surfaces in a specific time interval $[t_0, t]$. One useful property of these attracting and repelling LCSs is forward-backward time duality. As described by Haller (2015), this permits us to compute attracting and repelling LCSs using the same time interval $[t_0, t]$. In the case of repelling LCSs, this is done by ordinary advection, while attracting LCSs are computed by use of a reversed flow map attained from backward-time advection.

Unlike hyperbolic LCSs, elliptic LCSs and parabolic LCSs are described as coherent Lagrangian vortex boundaries and Lagrangian jet cores, respectively (Onu et al., 2015). How-

ever, all LCSs may be seen as consisting of a set of particles evolving with the particle transport in time, like any other material surface.

2.3.1 Defining most repelling material surfaces

Focusing on hyperbolic LCSs, we now follow the argument of Farazmand and Haller (2012a) endeavoring to quantify normal attraction and repulsion for a smooth curve $\mathcal{M}(t_0)$ at time t_0 in a two-dimensional system. This is motivated by the previously discussed definition of hyperbolic LCSs as locally most repelling or attracting material surfaces. While the main focus of our discussion pertains to three-dimensional systems, the two-dimensional case is useful in terms of coherently outlining this underlying argument.

As the system evolves, the material line $\mathcal{M}(t_0)$ is transported according to the flow map forming a time-dependent material line $\mathcal{M}(t) = \mathbf{F}_{t_0}^t(\mathcal{M}(t_0))$. Now, choose a normal vector $\mathbf{n}_0$ for each initial position $\mathbf{x}_0 \in \mathcal{M}(t_0)$, as visualized in figure 2.2a.

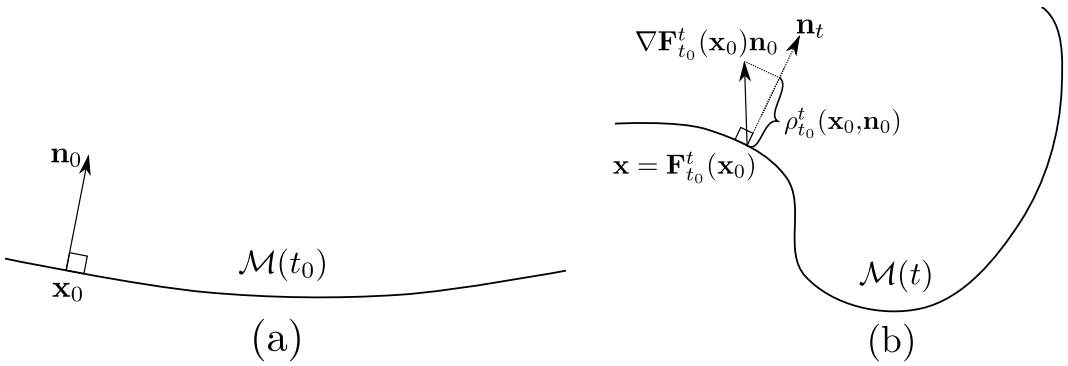


Figure 2.2: Visualization of the repulsion rate $\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0)$ for $\mathbf{x}_0$ on the material line $\mathcal{M}(t_0)$. (a) $\mathbf{n}_0$ denotes the normal vector of the material line $\mathcal{M}(t_0)$ at the point $\mathbf{x}_0$. (b) The vector $\nabla\mathbf{F}_{t_0}^t(\mathbf{x}_0)\mathbf{n}_0$ indicates how $\mathbf{n}_0$ has evolved in the time interval $[t_0, t]$, as defined by the advection of the particles initially situated at its endpoints. The normal repulsion rate $\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0)$ is then defined by the component of this time-evolved vector that is normal to the advected material line $\mathcal{M}(t)$. The normal of $\mathcal{M}(t)$ is denoted $\mathbf{n}_t$.

By considering the separation of the particles with initial positions $\mathbf{x}_0$ and $\mathbf{x}_0 + \mathbf{n}_0$, the time evolution of $\mathbf{n}_0$ is given by $\nabla\mathbf{F}_{t_0}^t(\mathbf{x}_0)(\mathbf{x}_0 + \mathbf{n}_0 - \mathbf{x}_0) = \nabla\mathbf{F}_{t_0}^t(\mathbf{x}_0)\mathbf{n}_0$ (see equation (2.29)). In order to measure the normal repulsion rate of the material line $\mathcal{M}(t_0)$, we are then interested in the component of this time-evolved normal vector that is perpendicular to $\mathcal{M}(t)$. Denoting this component $\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0)$, we may express the normal repulsion rate of $\mathcal{M}(t)$ on the trajectory $\mathbf{x}(t; t_0, \mathbf{x}_0)$ as

$$\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0) = \frac{\left\langle \nabla\mathbf{F}_{t_0}^t(\mathbf{x}_0)\mathbf{n}_0, \mathbf{n}_t \right\rangle}{|\mathbf{n}_0|}, \tag{2.46}$$

where $\mathbf{n}_t$ is normal to $\mathcal{M}(t)$ (see figure 2.2b). The normal repulsion rate $\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0)$ may then be used to classify material lines as either repelling or attracting. Specifically, if $\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0) > 1$ is satisfied, then $\mathcal{M}(t)$ is classified as normally repelling in the time interval $[t_0, t]$. Conversely, if $\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0) < 1$ is satisfied, $\mathcal{M}(t)$ is classified as normally attracting in the same time interval.

As outlined by Farazmand and Haller (2012a), the normal repulsion rate may be computed using the Cauchy-Green strain tensor

$$\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0) = \frac{1}{\sqrt{\left\langle \mathbf{n}_0, \left[\mathbf{C}_{t_0}^t(\mathbf{x}_0)\right]^{-1} \mathbf{n}_0 \right\rangle}}, \tag{2.47}$$

where $[...]^{-1}$ signifies taking the inverse. Using this description, Farazmand and Haller (2012a) define a normally repelling material line in the time interval $[t_0, t]$ as a compact material line segment $\mathcal{M}(t)$ on which

$$\rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0) > 1, \quad \rho_{t_0}^t(\mathbf{x}_0, \mathbf{n}_0) > |\nabla \mathbf{F}_{t_0}^t(\mathbf{x}_0)\mathbf{e}_0| \tag{2.48}$$

holds for any initial position $\mathbf{x}_0 \in \mathcal{M}(t_0)$. Here, $\mathbf{e}_0$ is a tangential unit vector of $\mathcal{M}(t_0)$. Moreover, compactness ensures that $\mathcal{M}(t)$ is bounded. Satisfying equation (2.48) therefore implies that $\mathcal{M}(t)$ is repelling and that this repulsion is greater than tangential stretching for all $\mathbf{x}$ along $\mathcal{M}(t)$. Note that the above argument is easily transferable to material surfaces in three dimensions. We do this by simply using the maximum of $|\nabla \mathbf{F}_{t_0}^t(\mathbf{x}_0)\mathbf{e}_0|$ for any tangent vector $\mathbf{e}_0$ within the material surface in equation (2.48).

This now enables us to further specify our definition of hyperbolic LCSs as locally most repelling or attracting material surfaces within a specific time interval. Dividing hyperbolic LCSs into attracting and repelling types, repelling types may only exist on normally repelling material lines $\mathcal{M}(t)$. Moreover, these material lines $\mathcal{M}(t)$ must serve as pointwise non-degenerate local maxima in terms of normal repulsion rate over the time interval $[t_0, t]$ (Farazmand and Haller, 2012a). Here, the neighborhood in which $\mathcal{M}(t)$ forms a local maximum consists of all sufficiently close continuously differentiable material surfaces. Conversely, utilizing the forward-backward time duality, attracting LCSs may be defined as repelling LCSs over the reversed time interval $[t, t_0]$.

Given that an LCS is invariably associated with the time interval over which it was computed, there is no guarantee for the continued existence of the same LCS over preceding or following time intervals. However, Farazmand and Haller (2012a) point out that as LCSs are computed as a time interval aggregate, a small time interval perturbation causing small changes to the repulsion rates of material surfaces is unlikely to materialize as notable changes to the LCSs.

2.4 Identifying repelling hyperbolic LCSs from their variational theory

Using the definition of hyperbolic LCSs in terms of normally attracting or repelling material lines described in section 2.3.1, Farazmand and Haller (2012b) derive a set of sufficient and necessary existence criteria. Considering a compact material surface $\mathcal{M}(t) \subset U$ evolving over the time interval $[t_0, t]$, $\mathcal{M}(t)$ is a repelling LCS over $[t_0, t]$ if and only if the following holds for all initial positions $\mathbf{x}_0 \in \mathcal{M}(t_0)$:

$$\begin{aligned} &1. \quad \lambda_2(\mathbf{x}_0) \neq \lambda_3(\mathbf{x}_0) > 1, \\ &2. \quad \langle \boldsymbol{\xi}_3(\mathbf{x}_0), H_{\lambda_3}(\mathbf{x}_0)\boldsymbol{\xi}_3(\mathbf{x}_0)\rangle < 0, \\ &3. \quad \boldsymbol{\xi}_3(\mathbf{x}_0) \perp \mathbf{T}_{\mathbf{x}_0}\mathcal{M}(t_0), \\ &4. \quad \langle \nabla\lambda_3(\mathbf{x}_0), \boldsymbol{\xi}_3(\mathbf{x}_0)\rangle = 0. \end{aligned} \tag{2.49}$$

Here λ_2, λ_3, and $\boldsymbol{\xi}_3$ are eigenvalues and eigenvectors of the Cauchy-Green strain tensor (see section 2.2.2), while $\mathbf{T}_{\mathbf{x}_0}\mathcal{M}(t_0)$ is the tangent space of $\mathcal{M}(t_0)$. The Hessian of the λ_3-field (see equation (2.50)) is denoted H_{λ_3}. By reference to the forward-backward time duality property, $\mathcal{M}(t)$ is an attracting hyperbolic LCS if the same criteria hold over the reversed time interval $[t, t_0]$. Note that condition 2 in (2.49) is derived from the general result in Farazmand and Haller (2012b). This computation is shown in appendix A.

Continuing the use of objective deformation theory (see section 2.2.2), condition 3 establishes that the Cauchy-Green strain tensor eigenvector $\boldsymbol{\xi}_3(\mathbf{x}_0)$, associated with the largest eigenvalue $\lambda_3(\mathbf{x}_0)$, is at all points perpendicular to $\mathcal{M}(t)$. Combined with condition 1, this ensures that $\mathcal{M}$ is normally repelling. This is because ensuring $\boldsymbol{\xi}_3(\mathbf{x}_0) \perp \mathcal{M}(t_0)$ makes the unit vector $\boldsymbol{\xi}_3(\mathbf{x}_0)$ coincide with $\mathbf{n}_0$. Note that we drop the tangent space notation for the sake of simplicity. Therefore, $\lambda_3 > 1$ corresponds to $\rho^t_{t_0}(\mathbf{x}_0, \mathbf{n}_0) > 1$. Moreover, requiring $\lambda_2(\mathbf{x}_0) < \lambda_3(\mathbf{x}_0)$ assures that the second inequality in equation (2.48) holds, even when adapted to the three-dimensional case.

Conditions 2 and 4 (see equation (2.49)) ensure that $\mathcal{M}(t)$ is a local maximum in terms of normal repulsion quantified by λ_3. Specifically, condition 4 establishes that the gradient of λ_3 at $\mathbf{x}_0$ along the perpendicular vector $\boldsymbol{\xi}_3(\mathbf{x}_0)$ is 0, indicating that $\mathcal{M}(t)$ is either a ridge, a valley, or a saddle point with respect to λ_3. The Hessian test imposed in condition 2, analogous to the second derivative test in one dimension, then assures that this local extremum is in fact a maximum. In three dimensions, the Hessian is given by

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{bmatrix}. \tag{2.50}$$

2.4.1 Autonomous dynamical system captures all LCSs in three-dimensional flows

As may be inferred from the preceding discussion, repelling hyperbolic LCSs are at all points normal to the eigenvector $\boldsymbol{\xi}_3(\mathbf{x}_0)$, corresponding to the largest Cauchy-Green strain tensor eigenvalue. According to Oettinger and Haller (2016), any hyperbolic or elliptic LCS initial position $\mathcal{M}(t_0)$ is at all points normal to a linear combination of $\boldsymbol{\xi}_1(\mathbf{x}_0)$ and $\boldsymbol{\xi}_3(\mathbf{x}_0)$. That is, material surfaces $\mathcal{M}(t_0)$ within which hyperbolic or elliptic LCSs may exist, are necessarily everywhere perpendicular to a vector field given by

$$\mathbf{n} = a\boldsymbol{\xi}_1(\mathbf{x}_0) + b\boldsymbol{\xi}_3(\mathbf{x}_0). \tag{2.51}$$

Specifically, where attracting hyperbolic LCSs are at all points perpendicular to $\boldsymbol{\xi}_1(\mathbf{x}_0)$, repelling hyperbolic LCSs are perpendicular to $\boldsymbol{\xi}_3(\mathbf{x}_0)$. Moreover, elliptic LCSs may be obtained from surfaces that are perpendicular to certain linear combinations of $\boldsymbol{\xi}_1(\mathbf{x}_0)$ and $\boldsymbol{\xi}_3(\mathbf{x}_0)$ (Oettinger and Haller, 2016). All hyperbolic or elliptic LCSs must therefore necessarily be subsets of manifolds defined as invariant manifolds of the system

$$\mathbf{x}_0' = (1-p)\boldsymbol{\xi}_2(\mathbf{x}_0) + p\boldsymbol{\xi}_i(\mathbf{x}_0), \quad p \in [0,1], \tag{2.52}$$

where $i = 1$ for repelling LCSs and $i = 3$ for attracting ones, while p is some scalar. This means that any trajectory in a system given by a linear combination of $\boldsymbol{\xi}_2(\mathbf{x}_0)$ and $\boldsymbol{\xi}_i(\mathbf{x}_0)$ starting out within $\mathcal{M}(t_0)$, is bound to remain within $\mathcal{M}(t_0)$. We use $\mathbf{x}'$ here instead of $\dot{\mathbf{x}}$ to distinguish the pseudo-time derivative of manifold defining ODEs from the actual time dependence of transport system ODEs.

Note that as these LCS candidate surfaces are all orthogonal to linear combinations of $\boldsymbol{\xi}_1(\mathbf{x}_0)$ and $\boldsymbol{\xi}_3(\mathbf{x}_0)$, they are all invariant manifolds of the autonomous dynamical system

$$\mathbf{x}_0' = \boldsymbol{\xi}_2(\mathbf{x}_0). \tag{2.53}$$

That is, any integral curve following the $\boldsymbol{\xi}_2$-field starting from a point within $\mathcal{M}(t_0)$, remains confined to $\mathcal{M}(t_0)$ (Oettinger and Haller, 2016). Constituting the main finding of Oettinger

and Haller (2016), this allows us to identify possible hyperbolic or elliptic LCS surfaces by computing long trajectories defined by the ODE (2.53).

Chapter 3

Method

Outlining the various choices, motivations, and approaches used during the current project, this chapter details our method for identifying three-dimensional hyperbolic LCSs. Starting out by summarizing the numerical treatment of system velocity fields and resulting strain characteristics, we describe methods for efficiently performing the necessary field interpolation and solving ODEs. The hyperbolic LCS existence criteria outlined in chapter 2 are then adapted as to facilitate numerical analysis. Prompted by these criteria, we then adapt the method of geodesic levelsets first described by Krauskopf and Osinga (2003) to compute LCS candidate manifolds. Two different approaches to adapting the method of geodesic levelsets are presented and compared with respect to accuracy, performance, and clarity. We finally devise a method for extracting hyperbolic LCSs from these candidate manifolds.

3.1 Tracer advection

Aiming to describe the overarching structures of complex flow systems in a finite-time interval, efforts of LCS identification are usually developed from flow maps. As discussed in section 2.2, the flow map $\mathbf{F}_{t_0}^{t}(\mathbf{x}_0)$ maps the initial position at t_0 of each particle in the system to its position at time t. In numerical analysis, a flow map is developed by advecting a set of particles in a velocity field by use of an iterative ODE solver. The velocity field for a fluid flow system may for example be attained by use of the Navier-Stokes equations, or in principle — if not necessarily in practice — by direct measurements. Attainment of the velocity field is however beyond the scope of this investigation.

Forming the basis of all further analysis, the flow map $\mathbf{F}_{t_0}^{t}(\mathbf{x}_0)$ is computed for a regular grid of $N = N_x N_y N_z$ initial positions on and within the boundaries of the region of interest U. This is done by advecting tracer particles initially placed at the initial positions $\mathbf{x}_0$ in the governing velocity field. As outlined in section 2.2.3, the flow map is developed in time while simultaneously solving the variational equation (2.36), also providing us with the flow

map Jacobian $\nabla \mathbf{F}_{t_0}^{t}(\mathbf{x}_0)$. This corresponds to simultaneously solving the set of 12 coupled equations given by the Cartesian forms of equation (2.35) and equation (2.36). We do this by use of the Dormand-Prince adaptive step method of orders 7 and 8. See sections 2.1 and 3.1.1 for further details.

Performing this computation is quite costly for large grids consisting of millions of tracers, requiring the use of parallel computing. As each tracer particle is independent of the remaining grid, this is easily done, for example by use of MPI and the *mpi4py* Python library. Specifically, the tracer particles are distributed evenly among all available processing cores, where the variational equations are solved for the considered time interval. Finally, the results are passed to the main process for further analysis.

The system of 12 coupled equations carries an added computational load compared to simply advecting the tracer particles. Consider however that the most notable alternative approach, proposed by Farazmand and Haller (2012a), for computing the flow map Jacobian entails advecting an additional auxiliary grid of six tracers per main grid tracer. The number of equations needed to be solved per time step is therefore $21N$ for Farazmand and Haller (2012a)'s approach, and only $12N$ for the current one. While adding some complexity to the computation, the method of variational equations is therefore preferred both due to its superior accuracy (Oettinger and Haller, 2016) and efficiency.

3.1.1 Implementation of automatic step control

The flow map advection described in section 3.1 and the point search trajectories to be described in section 3.4.3 are both computed by use of the Dormand-Prince method of orders 7 and 8. As noted in section 2.1.3, this method uses the difference between a 7^{th}- and an 8^{th}-order Runge-Kutta solution, $\bar{\mathbf{x}}_{n+1} - \widehat{\mathbf{x}}_{n+1}$, to automatically control step length. Here, $\widehat{\mathbf{x}}_{n+1}$ is the 7^{th}-order $(n+1)^{\text{th}}$ Runge-Kutta ODE solution step, while $\bar{\mathbf{x}}_{n+1}$ is the corresponding 8^{th}-order solution. Inspired by Hairer et al. (2008), we control the step length by limiting the componentwise error according to

$$|\bar{x}_i - \widehat{x}_i| \leq sc_i, \tag{3.1}$$

where

$$sc_i = \text{Atol}_i + \max\left(|\bar{x}_i|, |\widehat{x}_i|\right) \cdot \text{Rtol}_i, \tag{3.2}$$

is our componentwise error limit and the parameters

$$\text{Atol}_i = \text{Rtol}_i = 10^{-7} \tag{3.3}$$

are used consistently. This allows us to define our error measure

$$err = \sqrt{\frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \left(\frac{\bar{x}_{n+1,i} - \widehat{x}_{n+1,i}}{sc_i} \right)^2}, \tag{3.4}$$

(where $\mathcal{N}$ is the number of simultaneously computed variables) and use $\epsilon_{\text{tol}} = 1$ in equation (2.16), yielding the step acceptance criterion

$$err \leq 1. \tag{3.5}$$

Finally, the ideal step length Δt_{opt} is determined by inserting $\epsilon_{\text{tol}} = 1$ into equation (2.19), yielding

$$\Delta t_{\text{opt}} = \Delta t \left(\frac{1}{err} \right)^{\frac{1}{p+1}}, \tag{3.6}$$

where $p = 7$ is the order of the least accurate Runge-Kutta method.

Now, all steps that satisfy equation (3.5) are accepted, while steps that do not satisfy (3.5) are rejected. That is, as long as equation (3.5) is satisfied, we adopt the approximation $\mathbf{x}_{n+1} = \bar{\mathbf{x}}_{n+1}$ for next next time step $t_{n+1} = t_n + \Delta t$. The new step lengths are computed according to

$$\Delta t_{\text{new}} = \begin{cases} \min\left(\text{fac}_{\max} \cdot \Delta t, \text{fac} \cdot \Delta t_{\text{opt}}\right) & \text{if step is accepted} \\ \text{fac} \cdot \Delta t_{\text{opt}} & \text{if step is rejected.} \end{cases} \tag{3.7}$$

Here, $\text{fac} = 0.8$ and $\text{fac}_{\max} = 2.0$ are numerical safety factors, preventing excessive increases in step length and increasing the likelihood of the next step being accepted.

Note that while each coordinate in the three-dimensional system is managed independently, the corresponding time step lengths must be identical, prompting us to — for each step — apply the largest Δt_{new} to all coordinate ODEs.

Also note that the Dormand-Prince ODE solver, used to compute manifold points (see section 3.4.3), was implemented in Cython as to improve performance. This was done in order to retain the ease of development, as well as comprehensibility, of Python, while maximizing performance in computationally heavy operations.

3.1.2 Velocity field interpolation

Unlike investigations of three-dimensional analytical velocity fields, analysis of time-varying gridded model data in three dimensions requires the use of a quadrivariate interpolator. Specifically, this is needed to compute the accompanying flow map. Considering each velocity

field component separately, they are interpolated in time as well as the three spatial coordinates. As suggested by van Hinsberg et al. (2013), this was done by use of cubic B-splines (see section 2.1.4). Several multidimensional B-spline interpolation libraries are currently available. Choosing the *Bspline-Fortran* library for its performance and comprehensive documentation (Williams, 2018), quadrivariate velocity field interpolation was carried out using the `bspline_4d` derived type. This Fortran tool was made available in Python through the Fortran standard C interoperability. By writing a C++ wrapper class, the solver was made available in Python via Cython. Note that memory duplication was minimized by use of the Fortran reference-based subroutine call structures, as well as pointers at the C-level. This is critical, as large memory requirements could limit the efficiency of parallelization. Specifically, memory-intensive tasks could require us to use several computing cores per parallel process in order to pool memory, dramatically increasing resource usage.

3.2 Computing Cauchy-Green strain tensor eigenvalue and eigenvector fields

Enabling further deformation analysis and identification of LCSs, the Cauchy-Green eigenvectors $\boldsymbol{\xi}_i(\mathbf{x}_0)$ and eigenvalues $\lambda_i(\mathbf{x}_0)$ were computed at the tracer initial position grid by use of singular-value decomposition. As outlined in section 2.2.4, this allows us to implicitly determine $\boldsymbol{\xi}_i(\mathbf{x}_0)$ and $\lambda_i(\mathbf{x}_0)$ without having to compute the Cauchy-Green strain tensor $\mathbf{C}_{t_0}^{t}(\mathbf{x}_0)$. In this way, we limit the error introduced by floating point arithmetic (Watkins, 2005). Singular-value decomposition algorithms are widely available, for example in the *NumPy* library (Oliphant, 2015).

3.2.1 Cauchy-Green eigenvalue field interpolation

Identifying LCSs in the previously described discrete eigenvector and eigenvalue fields necessitates use of interpolation. That is, as we analyze off-grid points in U with regard to the existence conditions 2.49, we will need interpolated values for the eigenvalue fields $\lambda_2(\mathbf{x}_0)$ and $\lambda_3(\mathbf{x}_0)$.

These eigenvalue fields were interpolated by use of cubic trivariate B-splines, ensuring continuous first and second derivatives. Note that this use of cubic B-splines, ensuring continuous second derivatives, is necessary for evaluating the Hessian of $\lambda_3(\mathbf{x}_0)$ in existence condition 2 (see equation (2.49)). This was done by use of the `bspline_3d` derived type from the *Bspline-Fortran* library. Like previously described in section 3.1.2, this method was made available in Python by use of Cython.

3.2.2 Cauchy-Green eigenvector field interpolation

Accompanying the interpolated eigenvalue fields, LCSs identification requires a continuous $\boldsymbol{\xi}_3$-field. That is, as we construct surfaces according to condition 3 in equation (2.49), it becomes necessary to evaluate the $\boldsymbol{\xi}_3$-field between grid points.

While ultimately treated in the same way as the eigenvalue field — using the `bspline_3d` derived type from the *Bspline-Fortran* library — the eigenvector field requires some additional treatment. Specifically, as the stretching along $\boldsymbol{\xi}_i(\mathbf{x}_0)$ and $-\boldsymbol{\xi}_i(\mathbf{x}_0)$ is the same, there is no *a priori* reason to assume the singular-value decomposition consistently chose any specific orientation. As our cubic B-spline interpolation routine imposes continuity of the eigenvector components, it is critical that the underlying field exhibits this trait. Otherwise, it is likely that the resulting B-spline interpolation will be inaccurate. Continuity of eigenvector components was ensured by use of a three-dimensional local orientation correction scheme, adapted from the two-dimensional case outlined by Onu et al. (2015).

Local orientation continuity is ensured by considering the set of 64 nearest neighbor grid points of the considered coordinate. Note that $4^3 = 64$ is the smallest number of nearest neighbor grid eigenvector values required to compute a local trivariate cubic spline interpolation. Assuming that the extent of this neighborhood is small compared to the scale over which significant orientational changes may occur in the eigenvector field, we ensure that no input grid vectors are oriented in opposing directions. This is done by selecting the eigenvector of a neighborhood corner as a reference direction. All remaining grid point eigenvectors are then compared to this reference direction by means of computing their inner product. Whenever this inner product evaluates to less than zero, the current grid point eigenvector is reversed. That is, we exchange $\boldsymbol{\xi}_3(\mathbf{x}_0)$ for $-\boldsymbol{\xi}_3(\mathbf{x}_0)$. The two-dimensional equivalent of this algorithm is demonstrated in figure 3.1. Note that while unsuited for representation in a two-dimensional figure, the three-dimensional scheme remains completely analogous.

Following this local orientation correction, the neighborhood grid is passed to the *Bspline-Fortran* interpolator method. The three-component output is finally normalized, conforming with the convention of normalized eigenvectors.

3.3 Limit search to LCS existence subdomain

As previously discussed in section 2.4, three-dimensional hyperbolic LCSs may be identified as material surfaces forming local repulsion maxima. In order to limit the search for LCSs to the regions of the domain in which such maxima may exist, a subdomain of U is defined as the set of points where conditions 1, 2 and 4, given by equation (2.49), hold. Here, condition 1 establishes that a direction of maximum repulsion exists, while condition 2 excludes all

regions where no local maximum of λ_3 may occur. Finally, compliance with condition 4 ensures that any given point $\mathbf{x}_0$ is a local maximum of λ_3 along the direction defined by $\boldsymbol{\xi}_3(\mathbf{x}_0)$. Farazmand and Haller (2012a) note that condition 2 may be made more appropriate for numerical analysis by also allowing for equality with zero. Moreover, compliance with condition 4 may be checked by comparing $\lambda_3(\mathbf{x}_0)$ to $\lambda_3(\mathbf{x}_0 + \epsilon\boldsymbol{\xi}_3(\mathbf{x}_0))$ and $\lambda_3(\mathbf{x}_0 - \epsilon\boldsymbol{\xi}_3(\mathbf{x}_0))$, where ϵ is some appropriately chosen scalar. The resulting conditions used to obtain an "ABD subdomain" U_{ABD} are

$$\text{(A)} \quad \lambda_2(\mathbf{x}_0) \neq \lambda_3(\mathbf{x}_0), \quad \lambda_3(\mathbf{x}_0) > 1, \tag{3.8}$$

$$\text{(B)} \quad \langle \boldsymbol{\xi}_3(\mathbf{x}_0), H_{\lambda_3}(\mathbf{x}_0)\boldsymbol{\xi}_3(\mathbf{x}_0)\rangle \leq 0, \tag{3.9}$$

$$\text{(D)} \quad \lambda_3(\mathbf{x}_0) > \max(\lambda_3(\mathbf{x}_0 - \epsilon\boldsymbol{\xi}_3(\mathbf{x}_0)), \lambda_3(\mathbf{x}_0 + \epsilon\boldsymbol{\xi}_3(\mathbf{x}_0))), \tag{3.10}$$

where $H_{\lambda_3}(\mathbf{x}_0)$ is the Hessian matrix (see equation (2.50)) of the largest Cauchy-Green eigenvalue. $H_{\lambda_3}(\mathbf{x}_0)$ was computed by taking the partial derivatives of the cubic spline interpolation of $\lambda_3(\mathbf{x}_0)$.

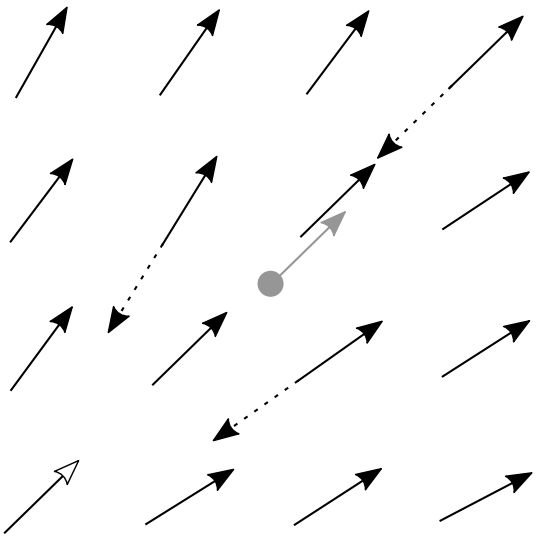


Figure 3.1: Visualization of the chosen local orientation correction scheme in two dimensions. Allowing for cubic interpolation, the $4^2 = 16$ nearest neighbor eigenvector grid points of the evaluated point (in gray) are extracted and corrected using the lower left corner eigenvector (no arrow fill) as a reference. Considering each extracted vector, its inner product with the lower left corner eigenvector is evaluated. If this inner product evaluates to less than zero, the eigenvector is turned 180° before passing to the interpolation routine. Note the dashed arrows indicating original eigenvector orientations where corrections have been performed. The resulting interpolated vector is presented in gray. This algorithm is generalized to three dimensions by using a grid of $4^3 = 64$ nearest neighbor eigenvectors.

Subdomains U_{ABD} obtained by applying conditions A, B, and D to the ABC flow test cases, as well as the fjord gridded model data set, described in sections 4.3-4.5, are presented in figures 4.7, 4.11, and 4.14.

3.4 Computing manifolds in Cauchy-Green eigenvector field by use of geodesic levelsets

As shown by Oettinger and Haller (2016) and described in section 2.4.1, hyperbolic LCSs in three-dimensional systems are invariant manifolds of the autonomous dynamical system described by equation (2.52). This suggests an approach to three-dimensional repelling LCS identification centered around computing these manifolds, producing surfaces that satisfy condition 3 in (2.49), alternatively expressed as

$$\text{(C)} \quad \mathcal{M}(\mathbf{x}_0, t_0) \parallel a\boldsymbol{\xi}_1(\mathbf{x}_0) + b\boldsymbol{\xi}_2(\mathbf{x}_0), \tag{3.11}$$

where a and b are scalars. In other words, the manifold $\mathcal{M}$ is at any point $\mathbf{x}_0$ tangent to the plane spanned by the vectors $\boldsymbol{\xi}_1(\mathbf{x}_0)$ and $\boldsymbol{\xi}_2(\mathbf{x}_0)$. As already noted in section 2.4, this is equivalent to $\mathcal{M}$ being perpendicular to $\boldsymbol{\xi}_3(\mathbf{x}_0)$ at all points. Also accounting for conditions A, B and D, presented in equations (3.8), (3.9) and (3.10), the locally most repelling surfaces are identified as hyperbolic LCSs.

As noted by Krauskopf et al. (2005), computing accurate multidimensional invariant manifolds is challenging, necessitating dedicated algorithms. Among other alternatives, Krauskopf et al. (2005) suggest approximation by geodesic levelsets, originally described by Krauskopf and Osinga (2003). Although initially intended for investigating manifolds described by functions of the form $\mathbf{x}' = \mathbf{f}(\mathbf{x})$, the method of geodesic levelsets may reasonably be adapted to identify manifolds defined by equation (2.52). This method is based on computing successive sets of points forming topological circles, each point computed by developing a trajectory in the underlying direction field from the preceding levelset. Points are then inserted or removed as to keep the level of mesh detail consistent. The result is a mesh M comprising radial strands of points along the invariant manifold, emanating from the initial position from which the manifold is developed. Some of the main terminology used to describe the method of geodesic levelsets is illustrated in figure 3.2.

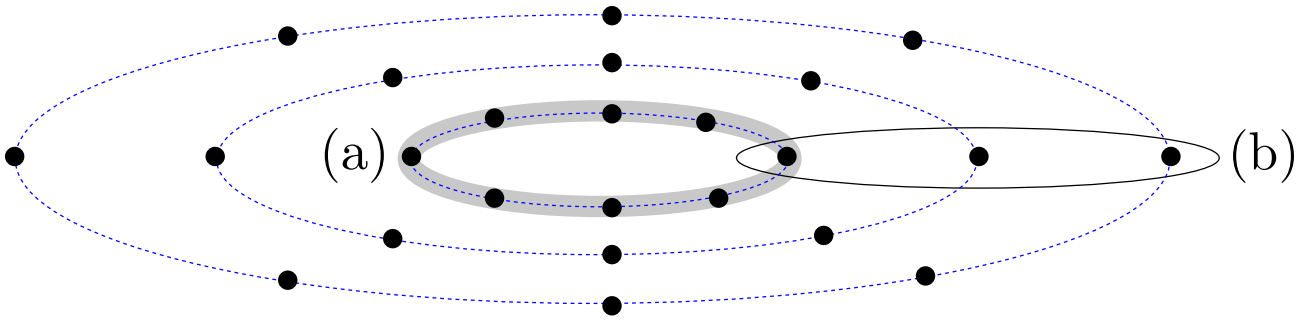


Figure 3.2: Illustration of some of the main terminology used to describe the method of geodesic levelsets. (a) The set of mesh points covered by the shaded area form a geodesic levelset M_i. The corresponding dashed line is referred to as a topological circle C_i. (b) The outlined set of points emanating radially outwards is referred to as a point strand.

3.4.1 Selecting initial positions and computing subsequent mesh points

Having identified an initial position $\mathbf{r}_0$ in the target invariant manifold $\mathcal{M}$ described by equation (2.52), a set of points are selected forming a small circle of radius r_{init} centered at $\mathbf{r}_0$ in the plane spanned by $\boldsymbol{\xi}_1(\mathbf{r}_0)$ and $\boldsymbol{\xi}_2(\mathbf{r}_0)$. This is shown in figure 3.3. The selected points form the initial levelset M_1, which is subsequently interpolated as to form a continuous circle C_1. This was carried out by use of a cubic spline interpolation scheme. A more detailed description of this topological circle interpolation scheme may be found in section 3.4.2.

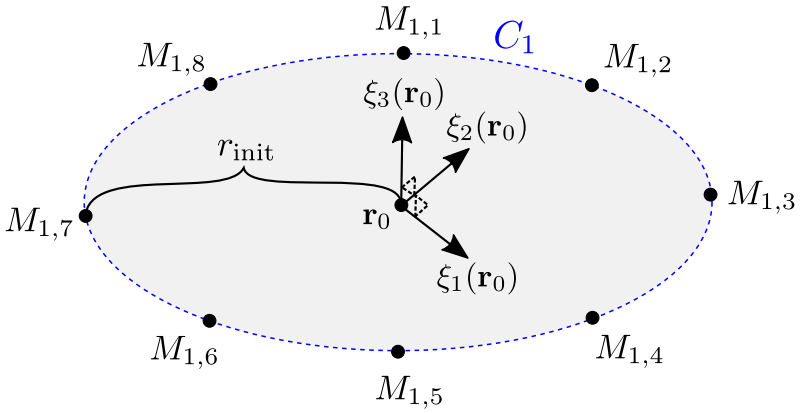


Figure 3.3: The initial levelset M_1 approximating a circle of radius r_{init} around the initial position $\mathbf{r}_0$. The points $M_{1,j}$ are computed as various linear combinations of $\boldsymbol{\xi}_1(\mathbf{r}_0)$ and $\boldsymbol{\xi}_2(\mathbf{r}_0)$ added to the initial position $\mathbf{r}_0$. Choosing a sufficiently small radius r_{init} and noting that as both $\boldsymbol{\xi}_1(\mathbf{r}_0)$ and $\boldsymbol{\xi}_2(\mathbf{r}_0)$ are perpendicular to $\boldsymbol{\xi}_3(\mathbf{r}_0)$, we assume that the set of points $\{M_{1,j}\}$ are all very close to the target manifold $\mathcal{M}$. The dashed topological circle C_1, approximated by interpolation of M_1, (see section 3.4.2) is highlighted in blue.

Each point in the first levelset M_1 is then used to compute a point in the subsequent levelset M_2. In the same way, all subsequent levelsets M_{i+1} are initially computed starting with the prior levelset M_i. For the sake of brevity and intuitiveness in the coming discussion,

we here denote the starting points in M_i ,and new points in M_{i+1}, ancestor and descendant points, respectively. Moreover, the set of points consisting of an initial levelset point $M_{1,j}$ and and all its descendants $\{M_{i,j}\}_{i=2}^{m}$, where m is the total number of levelsets, are referred to as a point strand. Here, point j in levelset i is denoted $M_{i,j}$. This is illustrated in figure 3.2.

In order to identify a new mesh point, consider the j^{th} point $M_{i,j}$ in the i^{th} levelset M_i. We search for a new point from the target manifold $\mathcal{M}$ in the half-plane $\mathcal{F}_r$ through $M_{i,j}$. This half-plane is orthogonal to C_i at $M_{i,j}$ and stretches outwards, as defined by the vector $M_{i,j} - M_{i-1,j}$. As may be seen in figure 3.4, this restricts us to searching for points in the outward radial direction, while allowing for local curvature in the underlying manifold. Krauskopf et al. (2005) suggest computing the normalized half-plane defining tangent vector at $M_{i,j}$ as

$$\mathbf{T}(M_{i,j}) = \frac{M_{i,j+1} - M_{i,j-1}}{|M_{i,j+1} - M_{i,j-1}|}. \tag{3.12}$$

However, in practice, it was found that simply inheriting the tangent vector from $M_{1,j}$ yielded smoother manifolds that are less exposed to accumulation of numerical noise. That is, we define the initial normalized tangent vector according to

$$\mathbf{T}_j = \mathbf{T}(M_{1,j}) = \frac{\boldsymbol{\xi}_3(M_{1,j}) \times (M_{1,j} - \mathbf{r}_0)}{|\boldsymbol{\xi}_3(M_{1,j}) \times (M_{1,j} - \mathbf{r}_0)|}, \tag{3.13}$$

subsequently passing it on to all following points along the same strand.

We start the search for a new point $M_{i+1,j}$ within the target half-plane $\mathcal{F}_r$ by defining a position "guess" $\mathbf{r}_{\text{aim}}$, presumed to be close to the intersection of $\mathcal{M}$ with $\mathcal{F}_r$. In order to choose $\mathbf{r}_{\text{aim}}$, we begin by computing a single 4^{th}-order Runge-Kutta step of length Δ_i from $M_{i,j}$. This is done by computing the cross product of the normalized tangent vector $\mathbf{T}$ and the underlying $\boldsymbol{\xi}_3$-field (see section 3.5 for details). Here, Δ_i is the prescribed inter-levelset step length, detailed in section 3.4.5.

We find the point $M_{i+1,j}$ by computing trajectories along linear combinations of $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$ from the prior topological circle C_i, aimed at $\mathbf{r}_{\text{aim}}$. A point along one of these trajectories is accepted as $M_{i+1,j}$ if it is both found to be within $\mathcal{F}_r$ and appropriately distanced from $M_{i,j}$. That is,

$$||M_{i+1,j} - M_{i,j}| - \Delta_i| < \Gamma_\Delta \Delta_i \tag{3.14}$$

is satisfied, where Γ_Δ is an inter-levelset separation tolerance factor. Moreover, sufficient proximity to $\mathcal{F}_r$ is defined as satisfying

$$\frac{\mathbf{r} - M_{i,j}}{|\mathbf{r} - M_{i,j}|} \cdot \mathbf{T}_j < \Gamma_{\mathcal{F}_r}, \tag{3.15}$$

where $\mathbf{r}$ is the current trajectory position and $\Gamma_{\mathcal{F}_r}$ is some scalar tolerance parameter. Note that as no given trajectory is certain to produce an acceptable point, this process may have to be repeated for several initial positions on C_i. This trial and error process is outlined in section 3.4.3. Also notice that by construction, $M_{i+1,j}$ may only be situated on — or very close to — the half-circle within $\mathcal{F}_r$, centered at $M_{i,j}$, of radius Δ_i. Refer to figure 3.4 for a visual representation of this approach.

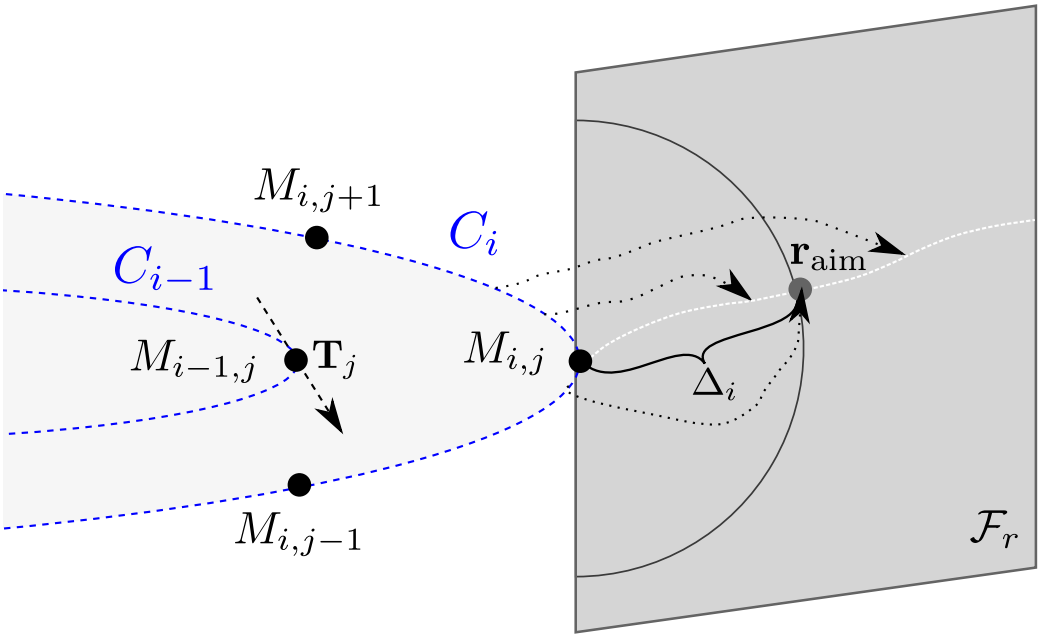


Figure 3.4: Computing a new mesh point $M_{i+1,j}$ within the half-plane $\mathcal{F}_r$. Using the inherited and normalized tangent vector $\mathbf{T}_j$ to define $\mathcal{F}_r$, the aim point $\mathbf{r}_{\text{aim}}$ is determined by taking a single 4^{th}-order Runge-Kutta step of length Δ_i within $\mathcal{F}_r$. This 4^{th}-order Runge-Kutta step is computed using the cross products of $\mathbf{T}_j$ and $\boldsymbol{\xi}_3(\mathbf{r})$, oriented along the vector $M_{i,j} - M_{i-1,j}$, where $\mathbf{r}$ is the current trajectory position. The trajectories from C_i (black dashed curves) are then computed by projecting the vector $\mathbf{r}_{\text{aim}} - \mathbf{r}$ into the local plane characterized by orthogonality to $\boldsymbol{\xi}_3(\mathbf{r})$ (see section 3.4.3). These trajectories are terminated whenever they reach $\mathcal{F}_r$, as defined by equation (3.15). If the concluding point of the trajectory $\mathbf{r}_{\text{end}}$ is approximately Δ_i away from $M_{i,j}$ (see equation (3.14)), it is accepted as $M_{i+1,j}$. Otherwise, a new trajectory is computed from a different initial position on C_i. This choice of initial positions is outlined in section 3.4.3. The white dashed line represents the intersection of the target manifold $\mathcal{M}$ with the half-plane $\mathcal{F}_r$. Note that imposing the inter-levelset step length Δ_i ensures that the selected point is located at the displayed half-circle in the half-plane $\mathcal{F}_r$.

3.4.2 Constructing topological circles from geodesic levelsets

Providing an approximation of the topological circle corresponding to levelset i, C_i is computed as a parametric spline interpolation. Considering the levelset M_i, the set of points $\{M_{i,j}\}_{j=1}^{n}$ are ordered clockwise. Point $M_{i,l}$ is then given a normalized parameter value $s_{j=l}$ based on the cumulative interpoint Euclidean distance along the list. Specifically,

$$s_{j=l} = \frac{\sum_{j=1}^{l-1} |M_{i,j+1} - M_{i,j}|}{\sum_{j=1}^{n-1} |M_{i,j+1} - M_{i,j}|}, \tag{3.16}$$

where n is the number of points in the levelset M_i. As to smooth out the junction at $M_{i,1}$, $M_{i,1}$ is added to the end of the ordered list of levelset points $\{M_{i,j}\}_{j=1}^{n}$. Now, the strictly increasing list of parameters $\{s\}$ corresponding to $\{M_{i,j}\}_{j=1}^{n} + M_{i,1}$ is given by

$$\{s\} = [0, s_{j=2}, s_{j=3}, ..., s_{j=n}, 1], \tag{3.17}$$

where $s_{j=l}$ by definition is equal to 0 and 1 for the first and last entries, respectively. Considering each of the Cartesian coordinates of the point $M_{i,l}$ as a univariate function of the parameter $s_{j=l}$, cubic B-splines were made for each set of coordinates by use of the `bspline_1d` derived type from the *Bspline-Fortran* library. This method was made available in Python, by use of Cython, as previously described in section 3.1.2.

3.4.3 Computing trajectories in the Cauchy-Green strain tensor eigenvector field

As outlined in section 3.4.1, new descendant points $M_{i+1,j}$ are found by computing trajectories along $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$, starting in C_i and continuing until a point is found that satisfies equations (3.14) and (3.15). Consider an initial position $C_i(s_k)$ somewhere along C_i. A trajectory is then computed along $\mathcal{M}$ by use of a Runge-Kutta iterative solver, specifically the Dormand-Prince method of orders 7 and 8. As $\mathcal{M}$ is everywhere orthogonal to $\boldsymbol{\xi}_3(\mathbf{r})$, the local direction of this trajectory may be chosen as any linear combination of $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$. That is, these trajectories could be computed according to

$$\mathbf{r}' = a\boldsymbol{\xi}_1(\mathbf{r}) + b\boldsymbol{\xi}_2(\mathbf{r}), \tag{3.18}$$

where a and b are scalars.

In order to limit the number of operations needed for the trajectory to reach $\mathcal{F}_r$, this linear combination should be chosen as to make $\mathbf{r}'$ as similar to $\mathbf{r}'_{\text{aim}} = \mathbf{r}_{\text{aim}} - \mathbf{r}$ as possible. That is, $\mathbf{r}'$ is continuously chosen within the local plane spanning $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$ as to guide the trajectory towards $\mathbf{r}_{\text{aim}}$. In order to reduce memory requirements, this was done by removing

the components of $\mathbf{r}'_{\text{aim}}$ parallel to $\boldsymbol{\xi}_3(\mathbf{r})$, leaving only the orthogonal components. In this way, we are able to construct trajectories in $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$ with no need for retaining the corresponding $\boldsymbol{\xi}_1$- and $\boldsymbol{\xi}_2$-interpolations in memory. Specifically, the trajectory is computed according to the ODE

$$\mathbf{r}' = \frac{\mathbf{r}'_{\text{aim}} - (\mathbf{r}'_{\text{aim}} \cdot \boldsymbol{\xi}_3(\mathbf{r}))\,\boldsymbol{\xi}_3(\mathbf{r})}{|\mathbf{r}'_{\text{aim}} - (\mathbf{r}'_{\text{aim}} \cdot \boldsymbol{\xi}_3(\mathbf{r}))\,\boldsymbol{\xi}_3(\mathbf{r})|}, \tag{3.19}$$

where the convention of unit length eigenvectors is implicit.

That is, $\mathbf{r}'$ is computed as the component of the direction towards $\mathbf{r}_{\text{aim}}$ that is orthogonal to the eigenvector $\boldsymbol{\xi}_3(\mathbf{r})$. As this limits $\mathbf{r}'$ to the local plane spanned by $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$ (see equation (2.33)), there is no need to compute $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$ explicitly. This approach is illustrated in figure 3.5.

Notice that while the Dormand-Prince method is a variable step ordinary differential equation solver, a maximum step length should be specified as to prevent the trajectory from overstepping the half-plane $\mathcal{F}_r$. This was done by setting a maximum trajectory step length of $|\mathbf{r}_{\text{aim}} - \mathbf{r}|$. Also note that while simpler in terms of implementation, choosing a constant step iterative solver is problematic in terms of choosing step length. While it seems natural to choose a fraction of the inter-levelset step Δ_i, such an approach would couple trajectory accuracy to the prescribed level of mesh detail. This kind of dependency is undesired, especially as Δ_i is changed dynamically over the course of computing a single manifold. Alternatively, the user could specify a constant step length which may or may not be appropriate for the considered eigenvector field.

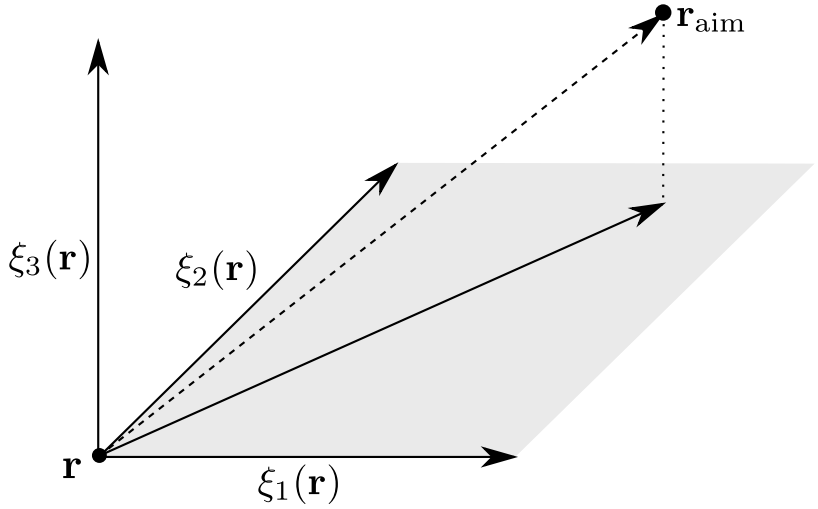


Figure 3.5: Illustration of algorithm for guiding trajectories in $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$. The vector $\mathbf{r}_{\text{aim}} - \mathbf{r}$ is projected into the plane spanned by $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$. The resulting vector is normalized and passed to the Runge-Kutta iterative solver.

The trajectory is terminated and the final point $\mathbf{r}_{\text{end}}$ returned if $\mathbf{r}_{\text{end}}$ satisfies equations (3.14) and (3.15). If no such point is found, the trajectory is terminated when it accumulates a length exceeding $l_{\max}\,|\mathbf{r}_{\text{aim}} - C_i(s_k)|$, where $l_{\max}$ is a scalar parameter greater than 1. Here, trajectory length is computed as the sum of all constituent steps $\mathrm{d}r$. As there is no guarantee for finding a point that satisfies both conditions (3.14) and (3.15) with any given trajectory originating from C_i, several attempts may be needed from different initial positions.

As to limit the number of trajectories we compute in order to find any single point, we track how each trajectory is terminated. Specifically, we note whether each trajectory, corresponding to a specific start point $C_i(s_k)$ along C_i, reaches $\mathcal{F}_r$. If the trajectory hit $\mathcal{F}_r$, we also track whether the distance $|\mathbf{r}_{\text{end}} - M_{i,j}|$ overshot or undershot with regard to Δ_i. For example, if we find that $|\mathbf{r}_{\text{end}} - M_{i,j}| < \Delta_i$, we consider the corresponding trajectory to undershoot with respect to the target point $M_{i+1,j}$. Conversely, if $|\mathbf{r}_{\text{end}} - M_{i,j}| > \Delta_i$, we consider the corresponding trajectory to overshoot. This feedback information allows us to dynamically choose the next initial position along C_i. We do this by increasing step length as long as there is no output status change, and conversely backtracking and reducing step length when a status change is detected.

Consider the search for a new descendant point $M_{i+1,j}$ from the ancestor point $M_{i,j}$. We use an imagined trajectory originating in $M_{i,j}$ to set the initial conditions in terms of trajectory termination status. Specifically, as $M_{i,j}$ is part of $\mathcal{F}_r$, this imaginary trajectory is immediately terminated and considered as undershooting with respect to $M_{i+1,j}$. We then move by one small step $\mathrm{d}s_{\min}$ clockwise along $C_i(s)$ and compute a trajectory according to the previously outlined method. If this trajectory also reaches $\mathcal{F}_r$, undershooting with respect to Δ_i, the step length $\mathrm{d}s$ is increased by a factor 10 before moving on. A third trajectory will then be computed using $s = s_{\text{start}} + 11\mathrm{d}s_{\min}$. In general, as long as the feedback status — that is, success at reaching $\mathcal{F}_r$ and under or overshoot with respect to Δ_i — does not change, the step length $\mathrm{d}s$ is increased until reaching a maximum value $\mathrm{d}s_{\max}$. However, if this feedback status changes, we backtrack to the last initial position before the status change. Then we attempt to move forward by one tenth of the last step forward. The step length is reduced by a factor of 10 until either no status change is detected, or the minimum step length $\mathrm{d}s_{\min}$ is reached. In both of these cases we move on, refraining from increasing the step length unless the status change was overstepped using $\mathrm{d}s_{\min}$. This process is illustrated in figure 3.6, where we proceed from undershooting with respect to Δ_i to subsequently overshooting. Note that status changes going from overshooting to undershooting are treated in exactly the same way. This is also the case for status change regarding whether or not the trajectory reached the half-plane $\mathcal{F}_r$ before being terminated due to exceeding the maximum trajectory length $l_{\max}\,|\mathbf{r}_{\text{aim}} - C_i(s_k)|$.

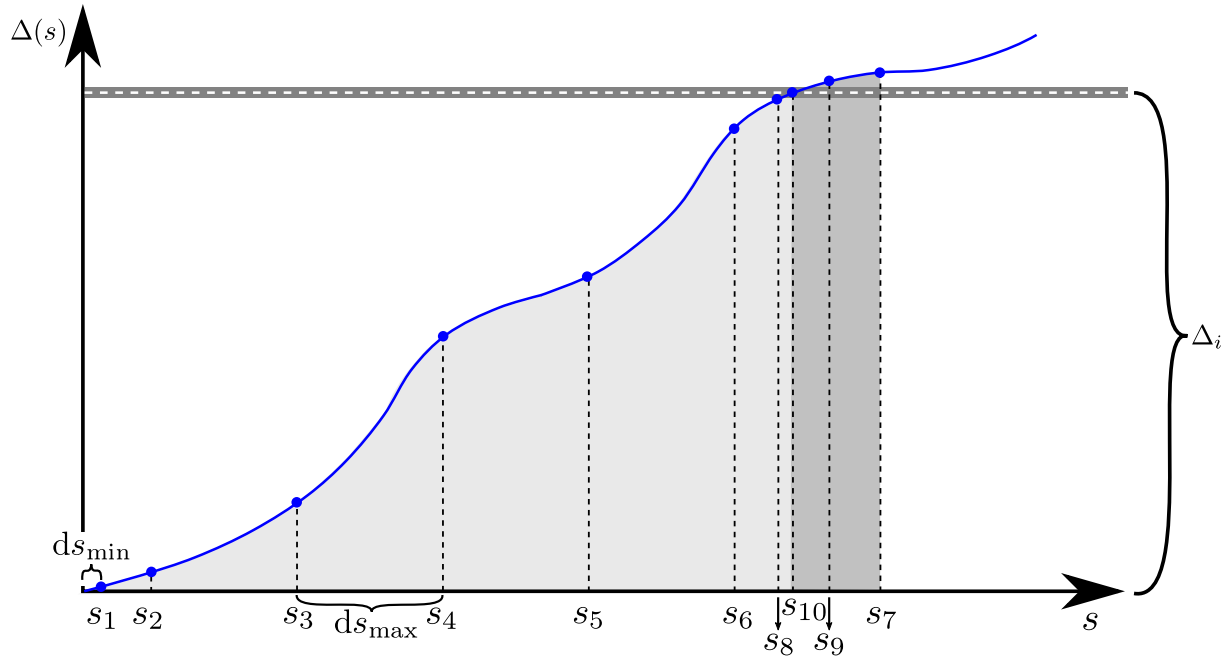


Figure 3.6: Illustration of algorithm for determining trajectory initial positions along the topological circle $C_i(s)$, parameterized by s. Along the horizontal axis is the parameter s of the interpolated topological circle $C_i(s)$ on which trajectories to find new grid points are started. On the horizontal axis we correspondingly have the distance from the resulting trial point $\mathbf{r}_{\text{end}}$ to the ancestor point $M_{i,j}$. This distance $|\mathbf{r}_{\text{end}} - M_{i,j}|$ is denoted $\Delta(s)$ and assumed to be a continuous function of s. The target inter-levelset step length Δ_i is signified with a dashed white line within a grey field indicating the accompanying numerical tolerance (see equation (3.14)). Note that $\Delta(s_{\text{start}} = 0) = 0$, as $M_{i,j}$ is part of the half-plane $\mathcal{F}_r$. Starting at the initial position $C_i(s_1 = s_{\min})$, a trajectory is developed in the ODE given by equation (3.19) until reaching the target half-plane $\mathcal{F}_r$. As the resulting distance of separation $\Delta(s_1)$, like $\Delta(s_0)$, is considered too small, $\mathrm{d}s$ is increased by a factor of 10 proceeding to $C_i(s_2)$. As we also have $\Delta(s_2) < \Delta_i$, $\mathrm{d}s$ is increased once more, reaching the imposed maximum $\mathrm{d}s_{\max}$. Sucessive steps of $\mathrm{d}s_{\max}$ are then made along $C_i(s)$ until we encounter $\Delta(s_7) > \Delta_i$. Backtracking to $C_i(s_6 + \mathrm{d}s_{\max}/10)$, we get $\Delta(s_8) < \Delta_i$. Reusing the same step length, we get $\Delta(s_9) > \Delta_i$, prompting the algorithm to backtrack again to $C_i(s_8 + \mathrm{d}s_{\max}/100)$. This initial position $C_i(s_{10})$ yields an acceptable point, as defined by equation (3.14). Note that this dynamic step management treats any endpoint status change, including failure to reach $\mathcal{F}_r$, in the same way.

This method for choosing initial positions $C_i(s)$ along the levelset topological circle rests on the assumption that the distance $\Delta(s) = |\mathbf{r}_{\text{end}}(s) - M_{i,j}|$ behaves like a continuous function, possibly exhibiting asymptotic behavior near regions in which it is undefined. That is, initial positions for which the corresponding trajectories fail to reach the half-plane $\mathcal{F}_r$ within the allotted trajectory length. Based on this premise, we use the intermediate value theorem to conclude that if $\Delta(s_1) < \Delta_i$ and $\Delta(s_2) > \Delta_i$, then $\Delta(s) = \Delta_i$ for some s such that $s_1 < s < s_2$. In order to limit the number of necessary trajectory attempts, we therefore endeavor to move along C_i with large steps wherever we are far from any intersections $\Delta(s) = \Delta_i$. However, whenever we may locate an intersection within a subdomain of $C_i(s)$,

we decrease the step length $\mathrm{d}s$ as to maximize our probability of finding this intersection. In order to manage resource requirements, we do not allow the step length to decrease beyond the minimum step length $\mathrm{d}s_{\min}$. Conversely, we limit the step from being increased beyond a maximum $\mathrm{d}s_{\max}$ as to avoid stepping over regions of two or more intersections $\Delta(s) = \Delta_i$. Note that moving past an even number of such intersections would render these intersections invisible to our method based on the intermediate value theorem, as no change in trajectory termination status would be detected.

As we anticipate asymptotic behavior close to any regions in which $\Delta(s)$ is undefined, these regions are also of great interest. We therefore manage $\mathrm{d}s$ in the same way here as for the neighborhoods that are close to intersections $\Delta(s) = \Delta_i$. This is the case both when moving from a region in which $\Delta(s)$ is undefined and when we move into such a region. In both cases, we endeavor to find an intersection within the aforementioned possible asymptotic behavior.

In principle, all initial positions along $C_i(s)$ could be tried in order to find the new point $M_{i+1,j}$. However, trajectories from points that are far removed from $M_{i,j}$ are likely to require a long integration path, resulting in increased numerical error. Moreover, this accumulated numerical error may decrease the likelihood of reaching $\mathcal{F}_r$. In order to further limit the number of computed trajectories, we therefore restrict the choice of initial positions to points within a certain parameter interval $s_{\text{start}} \pm s_{\text{lim}}$ around the starting position. Specifically, we start by moving clockwise around $C_i(s)$ until reaching $s_{\text{start}} + s_{\text{lim}}$. We then move back to the start position $M_{i,j} = C_i(s_{\text{start}})$ and proceed counter-clockwise in the same way until reaching $s_{\text{start}} - s_{\text{lim}}$. If no new point $M_{i+1,j}$ has been found after completing this search, we prompt the exception management algorithm described in section 3.4.6.

3.4.4 Managing mesh point density and accuracy

As new levelsets M_i are constructed, the distance separating neighboring points usually increases. Combined with the inter-levelset step length Δ_i, these nearest-neighbor distances determine the mesh density of the computed manifold. As the point mesh M is ultimately converted into a continuous manifold by use of linear interpolation (see section 3.7), we note that the accuracy of this resulting manifold depends on the mesh density of M. We manage this mesh density by specifying upper and lower boundaries for nearest neighbor separation, denoting these $\Delta_{\mathcal{F}}$ and $\delta_{\mathcal{F}}$, respectively. By requiring

$$\delta_{\mathcal{F}} \leq \Delta_i \leq \Delta_{\mathcal{F}} \tag{3.20}$$

for the inter-levelset step length and

$$\delta_{\mathcal{F}} \leq |M_{i,j+1} - M_{i,j}| \leq \Delta_{\mathcal{F}} \tag{3.21}$$

for levelset neighbor points, we hence limit the error of the following manifold interpolation (see section 3.7).

After a set of descendant points $\{M_{i+1,j}\}_{j=1}^{n}$ have been computed from the ancestor points $\{M_{i,j}\}_{j=1}^{n}$, according to the previously discussed method, all nearest neighbor separation distances are reviewed. If any of these exceed $\Delta_{\mathcal{F}}$, new points are inserted inbetween. We do this by use of the same algorithm as before, except that we use ghost ancestor points, placed on C_i halfway between the two nearest points. Specifically, if we have $|M_{i+1,j+1} - M_{i+1,j}| > \Delta_{\mathcal{F}}$, we use a ghost ancestor point $M_{i,j+1/2}$ between $M_{i,j}$ and $M_{i,j+1}$ to compute $M_{i+1,j+1/2}$. In this way, no new points are inserted using interpolations over larger separations than $\Delta_{\mathcal{F}}$. Note that as the ghost ancestor point does not itself have an ancestor to inherit a tangent vector from, $\mathbf{T}_{j+1/2}$ is computed as a weighted average of $\mathbf{T}_j$ and $\mathbf{T}_{j+1}$. This weighting is given by the distance from their respective mesh points to the ghost ancestor point.

In some cases, notably the spherical LCS described in section 4.2, nearest neighbor separation may instead decrease below $\delta_{\mathcal{F}}$. In this case, a point should be removed as long as doing so does not result in any nearest neighbor separations exceeding $\Delta_{\mathcal{F}}$. As may be seen in figure 3.7, if any of the two neighboring points may be removed without violating the upper limit of condition (3.21), the one resulting in the shortest interpoint separation is removed. If no point may be removed without violating the upper limit of condition (3.21), no action is carried out, as the minimum mesh point density threshold imposed by condition (3.21) is critical in terms of limiting interpolation error.

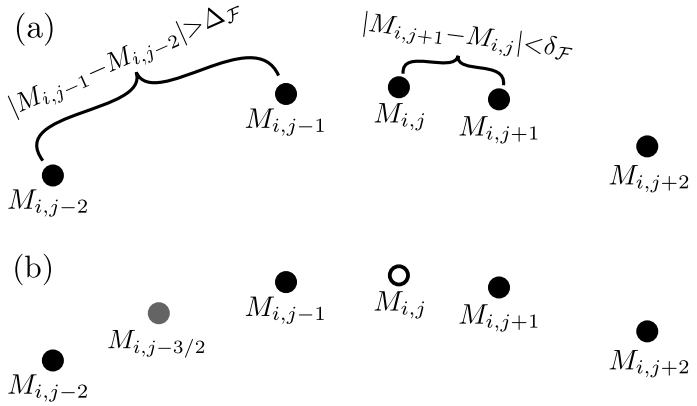


Figure 3.7: Mesh density management by insertion or removal of points. (a) The distance $|M_{i,j-1} - M_{i,j-2}|$ is identfied as greater than the maximum nearest neighbor distance $\Delta_{\mathcal{F}}$. Conversely, the distance $|M_{i,j+1} - M_{i,j}|$ is found to be smaller than the minimum nearest neighbor distance $\delta_{\mathcal{F}}$. (b) A new point $M_{i,j-3/2}$ is inserted between $M_{i,j-2}$ and $M_{i,j-1}$, as to keep nearest neighbor distance smaller than $\Delta_{\mathcal{F}}$. Noting that we have $|M_{i,j+1} - M_{i,j-1}| < |M_{i,j+2} - M_{i,j}|$ and $\delta_{\mathcal{F}} \leq |M_{i,j+1} - M_{i,j-1}| \leq \Delta_{\mathcal{F}}$, $M_{i,j}$ is removed to keep nearest neighbor distance larger than $\delta_{\mathcal{F}}$.

3.4.5 Curvature-guided step management

As detailed in section 3.4.4, all nearest neighbor manifold points are required to be separated by a distance in the range specified by conditions (3.20) and (3.21). This leaves us with some flexibility with regard to choice of the inter-levelset step length Δ_i. As suggested by Krauskopf et al. (2005), this is managed by monitoring local curvature. Aiming to represent local manifold behavior with appropriate detail, we adjust Δ_i dynamically. We start by defining minimum and maximum angular offset thresholds, denoting these $\alpha_{\min}$ and $\alpha_{\max}$, respectively. The local curvature boundary parameters $\alpha_{\min}$ and $\alpha_{\max}$ specify an interval of acceptable axial angle offsets α. As illustrated in figure 3.8, $\alpha_{i,j}$ is defined as the angular offset between the vectors $M_{i,j} - M_{i-1,j}$ and $M_{i+1,j} - M_{i,j}$, both projected into $\mathcal{F}_r$. A second analogous criterion compares the product $\alpha\Delta_i$ with corresponding upper and lower boundaries $(\Delta\alpha)_{\min}$ and $(\Delta\alpha)_{\max}$. By choosing appropriate bounds $(\Delta\alpha)_{\min}$ and $(\Delta\alpha)_{\max}$, this criterion allows us to set stricter axial angle offset requirements for large values of Δ_i. Conversely, small inter-levelset step lengths Δ_i are allowed larger α-values. In the same way, large Δ_i steps are allowed smaller α-values. That is, we require that for each point $M_{i+1,j}$ in a new levelset we have

$$\alpha_{\min} < \alpha_{i,j} < \alpha_{\max} \tag{3.22}$$

and

$$(\Delta\alpha)_{\min} < \Delta_i\alpha_{i,j} < (\Delta\alpha)_{\max}. \tag{3.23}$$

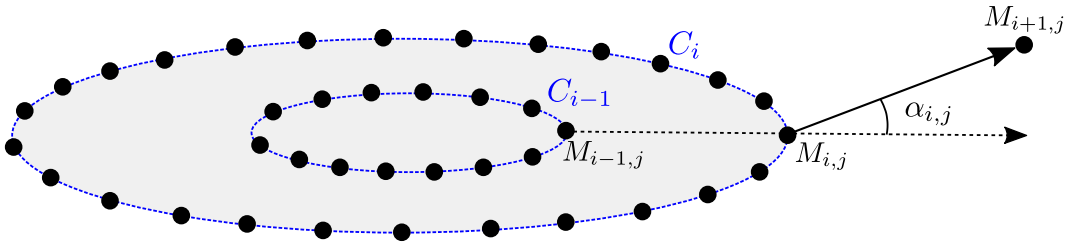


Figure 3.8: Illustration of the axial angle offset α. We define $\alpha_{i,j}$ as the angle between the vectors $M_{i,j} - M_{i-1,j}$ and $M_{i+1,j} - M_{i,j}$.

Having completed a levelset M_{i+1} of descendant points, using the step length Δ_i, we review all angles $\{\alpha_{i,j}\}_{j=1}^{n}$. If any of these angles violate the upper thresholds (see equations (3.22) and (3.23)) and $\Delta_i \geq 2\delta_{\mathcal{F}}$, M_{i+1} is discarded and recomputed using half the initial step-length. Conversely, if for all $\alpha_{i,j}$, both lower thresholds are violated and $2\Delta_i \leq \Delta_{\mathcal{F}}$, the levelset is kept, but the inter-levelset step length Δ_i is doubled for the next levelset. Otherwise, Δ_i is unaltered. Note that these local curvature tests are not used for inserted points. This is because an inserted point ghost ancestor does not itself have an ancestor, nor

a vector $M_{i,j+1/2} - M_{i-1,j+1/2}$ from which to compute an angular offset. However, when using the inserted point $M_{i+1,j+1/2}$ as an ancestor, local curvature $\alpha_{i+1,j+1/2}$ is estimated by use of the vector $M_{i+1,j+1/2} - M_{i,j+1/2}$. That is, by use of the ghost ancestor point.

3.4.6 Handling failure to identify satisfactory point

As noted in section 3.4.3, no given trajectory from the topological circle C_i is certain to produce an acceptable point in the half-plane $\mathcal{F}_r$. It is therefore possible that none of the attempted initial positions $\{C_i(s_k)\}$ along C_i yield acceptable points. Since any single missing point prohibits us from computing further levelsets, proper handling of elusive points is critical.

Although impossible to guarantee, the probability of finding any given point may be significantly increased by taking appropriate measures in response to an initially failed point search. The chosen strategy was centered around introducing incremental adjustments to $\mathbf{r}_{\text{aim}}$, while progressively relaxing the conditions for accepting candidate points.

Specifically, $\mathbf{r}_{\text{aim}}$ is adjusted by introducing an angular offset $\Delta\phi$ along the half-circle of radius Δ_i in $\mathcal{F}_r$ (see figure 3.9). Note that while the range of such angular offsets should be guided by the chosen maximal axial angle offset $\alpha_{\max}$, the number of trial offsets should be guided by runtime considerations. Specifically, the angular offsets $\Delta\phi$ should roughly range from $-\alpha_{\max}$ to $\alpha_{\max}$, ranging most or all acceptable axial angle offsets. In this way, we attempt to correct poor initial choices of $\mathbf{r}_{\text{aim}}$.

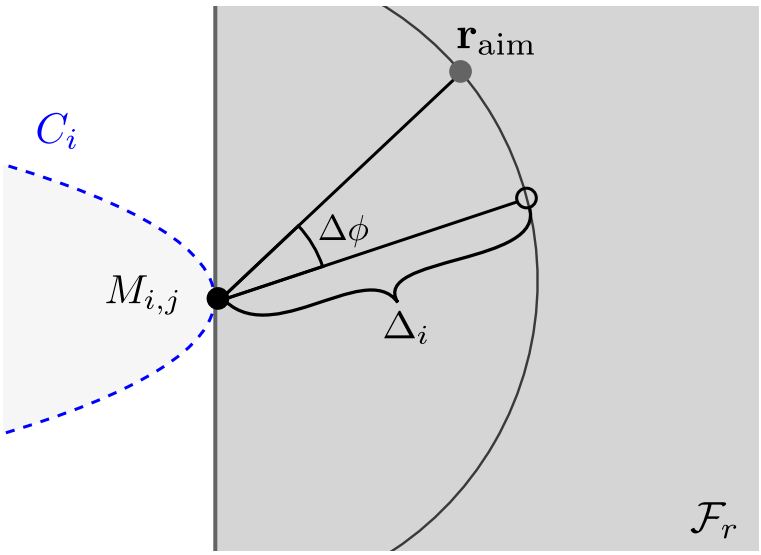


Figure 3.9: Adjustment of $\mathbf{r}_{\text{aim}}$ by introducing an angular offset $\Delta\phi$ along the half-circle of radius Δ_i in $\mathcal{F}_r$. This adjustment is introduced as to account for poor initial choices of $\mathbf{r}_{\text{aim}}$.

Having repeated the point search algorithm described in sections 3.4.3 using all $\mathbf{r}_{\text{aim}}$ defined with all offsets $\{\Delta\phi\}$ without obtaining an acceptable manifold point, the conditions

for point acceptance are relaxed. This is done progressively by increasing the tolerances $\Gamma_{\mathcal{F}_r}$ and Γ_Δ up to predetermined maxima $\Gamma_{\mathcal{F}_r}^{\max}$ and $\Gamma_\Delta^{\max}$. Having increased the point acceptance tolerances to their maximum values without finding an acceptable point, the incomplete levelset is discarded and recomputed with $\Delta_i = \delta_\mathcal{F}$. If the step length Δ_i was already set to its minimum $\delta_\mathcal{F}$, the procedure terminates.

3.4.7 Limiting accumulation of numerical noise

Early tests of the method of geodesic levelsets indicated that error buildup over many levelsets would result in unexpected behavior for some non-analytical test fields. Specifically, small ridges in the mesh would grow for every levelset, eventually resulting in seemingly chaotic manifold structures. In some cases, this would cause loops in C_i, or otherwise make the manifold fold into itself. This behavior is highly problematic as manifolds described by equation (2.52) are by definition unable to self-intersect in any way.

In order to curb this accumulation of noise, we review each completed levelset M_i. Consider a point $M_{i,j}$ on this completed levelset. If any non-nearest neighbor $M_{i,j+k}$, $k > 1$ is sufficiently close to $M_{i,j}$, the intervening levelset points are removed. That is, we cut C_i short whenever $|M_{i,j+k} - M_{i,j}|$ is sufficiently small compared to both the maximum nearest neighbor separation $\Delta_\mathcal{F}$ and the accumulated Euclidean nearest neighbor separations along the intervening points. Specifically, as we move along C_i, if for any point $M_{i,j+k}$, where $k > 1$, we have

$$|M_{i,j+k} - M_{i,j}| < \Delta_\mathcal{F} \tag{3.24}$$

and

$$|M_{i,j+k} - M_{i,j}| < c_{\text{arc}} \sum_{l=0}^{k-1} |M_{i,j+l+1} - M_{i,j+l}|\,, \tag{3.25}$$

all points $M_{i,j+l}$ for $l = 1, 2, ..., k-2, k-1$ are removed. Here, c_{arc} is some constant $0 \leq c_{\text{arc}} \leq 1$ used to control bulge tolerance. Specifically, c_{arc} determines by how much arc length, represented by cumulative nearest neighbor Euclidean norm, must be reduced in order to justify point removal. That is, a large c_{arc} allows for removal of blunt bulges in C_i, while a small c_{arc} only allows for removal of sharp bulges or loops. An example demonstrating this method is displayed in figure 3.10. Note that while possibly causing some loss of manifold detail, requiring compliance with conditions (3.24) and (3.25) prevents deletion of most conceivable large bulge formations. Also notice that since no new points are added, no error is introduced by this correction algorithm.

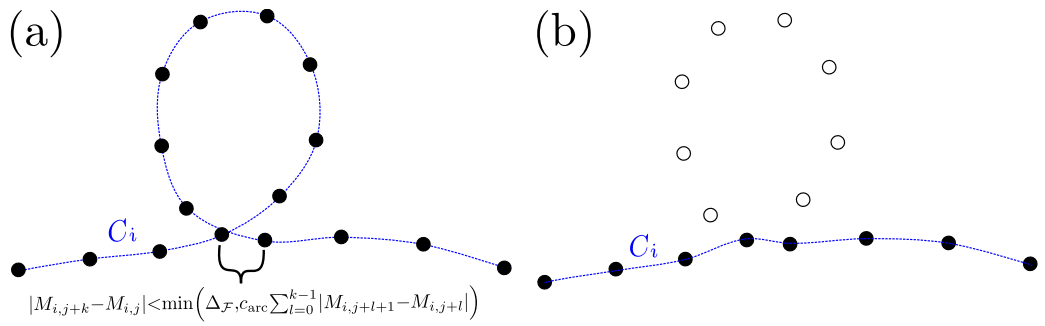


Figure 3.10: Visualization of presently described noise removal algorithm. (a) The interpoint distance $|M_{i,j+k} - M_{i,j}|$ is found to be smaller than the minimum of $\Delta_{\mathcal{F}}$ and the parameter c_{arc} times the accumulated Euclidean distance between nearest neighbor points from $M_{i,j}$ to $M_{i,j+k}$. (b) The levelset points intervening between $M_{i,j}$ and $M_{i,j+k}$ have been discarded, smoothing out the corresponding topological circle C_i.

3.4.8 Termination criteria

As to avoid computing needlessly large manifolds, we define a maximum manifold size. We use geodesic distance — the smallest distance along the manifold from $\mathbf{r}_0$ to the outer levelset — as a proxy variable indicating size. This geodesic distance $r_{\mathcal{M}}$ is approximated by the sum of all completed inter-levelset steps according to

$$r_{\mathcal{M}} = \sum_i \Delta_i. \tag{3.26}$$

To limit manifold size, we define a maximum geodesic distance $r_{\max}$, requiring the method to terminate whenever $r_{\mathcal{M}}$ exceeds $r_{\max}$. This was done in order to ensure that no manifolds would become unnecessarily large, possibly straining the available memory.

A second criterion for immediate termination is implemented in order to prevent manifold self-intersections. As already noted in section 3.4.7, this behavior should by definition not occur and could therefore be considered an indication of failure. Evaluating whether every new topological circle C_i intersects any previous topological circle $C_{k<i}$, we define the self-intersection geodesic distance tracker variable q. Initially set to zero, q is incremented by Δ_i whenever a new levelset M_i is found to cause a self-intersection, and conversely reset to zero whenever no new self-intersections are detected. In order allow for some numerical error, we define the tolerance parameter $q_{\max}$, requiring the manifold generation method to terminate whenever q exceeds $q_{\max}$.

Self-intersections are detected by use of triangle interpolations. As will be described in detail in section 3.7, the complete mesh M is interpolated into a continuous surface by forming triangular surface elements between neighboring mesh points. This is essentially done by trilinear interpolation. Having computed a new levelset M_i, the area between C_{i-1} and C_i

is covered by triangles, whereupon each new triangle is compared with all previously added triangles. If one or more of these triangles are found to intersect with any of the previously added triangles, the levelset M_i is flagged as self-intersecting. Intersections between new and previously added triangles were detected with the Möller-Trumbore ray-triangle intersection algorithm, using a detection sensitivity parameter $\epsilon_{MT} = 10^{-8}$ (Möller and Trumbore, 1997). Our algorithm for detecting triangle intersections is outlined in figure 3.11.

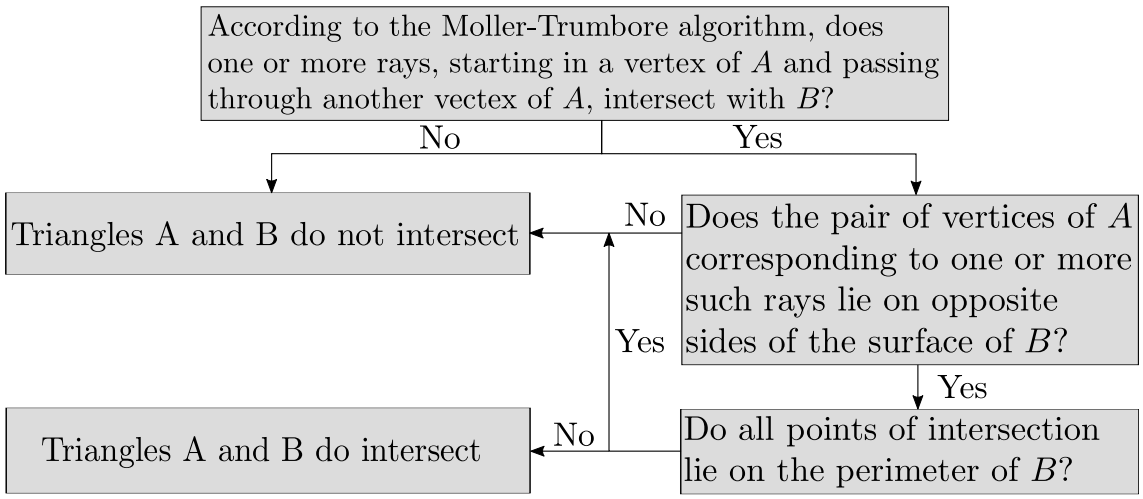


Figure 3.11: Flowchart describing our approach to identifying manifold self-intersections. This was done by applying the above algorithm to all interpolation triangles A (see section 3.7) in a pending levelset, each compared to all previously added interpolation triangles B. If one or more intersections between any new triangle A and and existing triangle B is identified, the pending levelset is flagged as self-intersecting.

As neighboring triangles defined with the chosen triangulation approach (see section 3.7) share sides, intersections along sides were allowed. Although unlikely to occur, a side effect of allowing shared sides is that two or more triangles are also allowed to coincide perfectly. Note that this exception also allows for self-intersections, as long as the corresponding intersecting triangles do so in exactly two points, each on the sides of both triangles. This case, as well as the case of neighboring triangles, are both shown in figure 3.12. As could be expected when using the double-precision number representation, these issues were found to be insignificant. Even if a single triangle intersection is missed due to these inconsistencies, neighboring triangles are very likely to be flagged as intersecting.

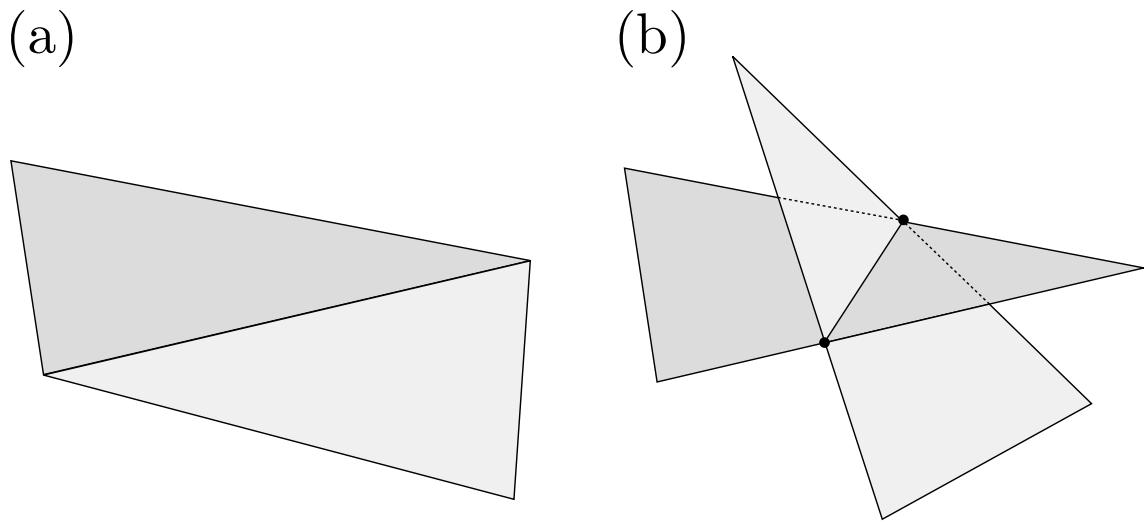


Figure 3.12: Illustration of triangle intersection special cases. (a) Neighboring triangles sharing a single side are accepted as these are a natural part of our triangulation algorithm (see section 3.7). (b) Unintended, but unproblematic, case of triangle intersections evading detection from the algorithm presented in figure 3.11. As the triangle sides intersect in exactly two points, this case is not recognized as a self-intersection.

3.4.9 Boundary treatment

The method of geodesic levelsets requires adding new points in complete levelsets forming topological circles. In addition to prohibiting addition of further points after failing to complete a single levelset, this forbids us from adding new levelsets after exceeding the domain boundaries within which the Cauchy-Green eigenvalues and eigenvectors are defined. That is, reaching these domain boundaries at a single point prohibits further development of the manifold. In order to consistently detect the manifold behavior near the boundaries of the region of interest U, the domain in which the Cauchy-Green eigenvalues and eigenvectors are defined should extend some distance beyond U. In the case of periodic systems, such as the ABC flow (see section 4.3), this is simply an exercise of utilizing this periodicity.

In aperiodic flow fields however, this requires us to perform advection and compute eigenvalues and -vectors for an extended domain enclosing the region of interest U. However, in the absence of such a padding region, imposing periodic boundary conditions on the interpolated eigenvector field allows us to continue the computation — even as parts of the manifold have left the defined domain. Note that as long as no points outside U are selected as parts of the resulting LCS (see section 3.8), this treatment will not introduce any error unless a point strand returns to U after previously having left the domain of interest.

It should also be noted that the trajectories used to identify new mesh points are likely to reach these domain boundaries before the actual mesh points. This unpredictability may be controlled by limiting the maximal trajectory length parameter $l_{\max}$ (see section 3.4.3).

3.5 Simplifying the method of geodesic levelsets by use of radial trajectories

The method of geodesic levelsets was initially developed to compute manifolds defined by being everywhere tangent to a single direction field $\mathbf{r}' = \mathbf{f}(\mathbf{r})$. That is, an initial position in $\mathcal{M}$, transported by $\mathbf{f}(\mathbf{r})$ for any time t remains within and spans the manifold $\mathcal{M}$. Assuming continuous trajectory changes while moving continuously along the topological circle C_i, new points are found by simply advecting these initial positions in $\mathbf{f}(\mathbf{r})$. As noted by Oettinger and Haller (2016), hyperbolic repelling LCSs are subsets of manifolds defined by their orthogonality to the $\boldsymbol{\xi}_3$-field. In contrast to the trajectories used by Krauskopf and Osinga (2003) and Krauskopf et al. (2005), trajectories in these manifolds are free to move within the local $\boldsymbol{\xi}_1$-$\boldsymbol{\xi}_2$-plane. That is, these trajectories have an additional degree of freedom. As described in section 3.4.3, this allows us to minimize trajectory travel distance by aiming each trajectory towards a position expected to be close to the next mesh point.

A more direct approach than the previously described adaptation was implemented by utilizing radial trajectories within the half-planes $\mathcal{F}_r$. These trajectories are defined by the cross product of the $\boldsymbol{\xi}_3$-field and the local tangent vector of C according to

$$\mathbf{r}' = \boldsymbol{\xi}_3(\mathbf{r}) \times \mathbf{T}_j. \tag{3.27}$$

The resulting direction $\mathbf{r}'$ is then compared with $|M_{i,j} - M_{i-1,j}|$, reversing $\mathbf{r}'$ if their inner product evaluates to less than zero. This approach is further illustrated in figure 3.13.

Using the cross product of $\boldsymbol{\xi}_3$ and the $\mathcal{F}_r$ plane normal vector $\mathbf{T}_j$ to compute trajectories requires special handling of the case where $\boldsymbol{\xi}_3(\mathbf{r}) \parallel \mathbf{T}_j$. This is because $\boldsymbol{\xi}_3(\mathbf{r}) \times \mathbf{T}_j$ goes to zero as $\boldsymbol{\xi}_3$ is orthogonal to $\mathcal{F}_r$. Specifically, whenever the condition

$$|\boldsymbol{\xi}_3(\mathbf{r}) \times \mathbf{T}_j| \geq \Gamma_\perp \tag{3.28}$$

is violated, $\mathbf{r}'$ is copied from the last step. Here, $\Gamma_\perp$ is some scalar input parameter. Note that condition (3.28) also serves the purpose of preventing numerical round-off error from significantly altering the computation as $\boldsymbol{\xi}_3(\mathbf{r}) \times \mathbf{T}(\mathbf{r})$ becomes small. Since trajectories defined by equation (3.27) necessarily remain within the half-plane $\mathcal{F}_r$, equation (3.14) constitutes the only remaining acceptance criterion for trajectory points. In other words, the trajectory originating from the ancestor point $M_{i,j}$ is developed according to equation (3.27) until equation (3.14) is satisfied. In order avoid overstepping, the Dormand-Prince method step length $\mathrm{d}r$ is continuously limited from above according to

$$\mathrm{d}r \leq \Delta_i \Gamma_\Delta. \tag{3.29}$$

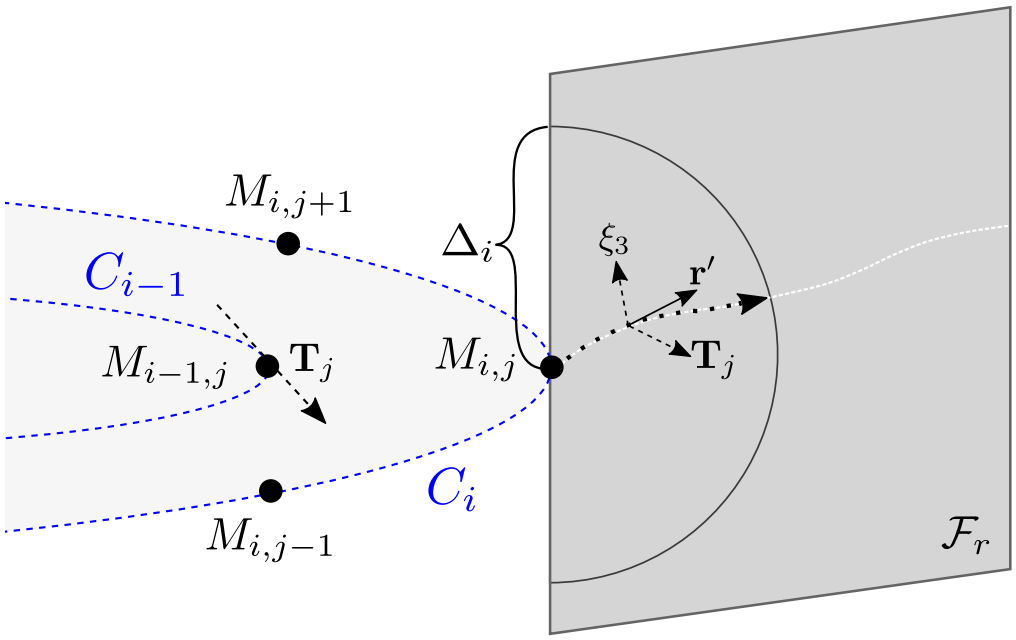


Figure 3.13: Illustration of point search trajectories being generated by use of forced radial trajectories within the half-plane $\mathcal{F}_r$. A single trajectory is computed from the ancestor point $M_{i,j}$ by use of an adaptive step Runge-Kutta iterative ODE solver (see section 3.1.1). The input directional vectors $\mathbf{r}'$ are computed as the cross product of the inherited tangent vector $\mathbf{T}_j$ and $\boldsymbol{\xi}_3(\mathbf{r})$, oriented along $M_{i,j} - M_{i-1,j}$. That is, if the inner product $\mathbf{r}' \cdot (M_{i,j} - M_{i-1,j})$ evaluates to less than zero, $\mathbf{r}'$ is reversed. The trajectory is terminated and the concluding point returned if $|\mathbf{r} - M_{i,j}|$ is sufficiently close to Δ_i (see equation (3.14)). The white dashed line represents the intersection of the target manifold with the half-plane $\mathcal{F}_r$.

That is, we impose a maximum step length given by the inter-levelset step tolerance. Finally, the trajectory is terminated if no acceptable point has been found after accumulating an arc-length exceeding $l_{\max}\Delta_i$, where, again, $l_{\max}$ is a scalar input parameter greater than 1. In this case, the algorithm attempts to decrease the inter-levelset step length Δ_i to $\delta_{\mathcal{F}}$ and recompute the levelset. If the failed levelset was computed using $\Delta_i = \delta_{\mathcal{F}}$, the computation is terminated.

Note that while altering trajectory development, this simplified approach of radial trajectories otherwise follows the same process as the previously described adaptation of the method of geodesic levelsets. This includes construction of the initial levelset, management of mesh density, elimination of numerical noise, and self-intersection control (see sections 3.4.1, 3.4.4, 3.4.7, and 3.4.8, respectively).

3.6 Comparing adaptation approaches for the method of geodesic levelsets

Sections 3.4 and 3.5 outline two related approaches to adapting the method of geodesic levelsets for manifolds defined by orthogonality to a vector field $\boldsymbol{\xi}_3(\mathbf{r})$. The first approach relies on estimating the location of new points in order to guide trajectories in $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$. In contrast, the latter simply computes trajectories by forming the cross product of the vector $\boldsymbol{\xi}_3(\mathbf{r})$, to which the manifold is orthogonal, with a levelset tangent vector, inherited from the initial circular levelset M_1.

While we in both approaches follow trajectories in the $\boldsymbol{\xi}_3$-orthogonal manifold and add points in a mesh of topological circles, they differ significantly both in terms of speed and consistency. The approach of forced radial trajectories within $\mathcal{F}_r$ was found to be approximately two orders of magnitude faster than the alternative method. This is because each radial trajectory is bound to the half-plane $\mathcal{F}_r$, essentially guaranteeing that it produces an acceptable mesh point as long we choose a reasonable maximum arc length $l_{\max}\Delta_i$. That is, we only compute a single trajectory per new mesh point. This one-to-one relationship between computed trajectories and new mesh points is contrasted by the method of guided trajectories, where thousands of trajectories may be needed to add a single mesh point. As the probability of trajectories terminating with an acceptable mesh point is highly dependent on having an appropriate aim point, the first method also suffers heavily in terms of speed from erratic manifold behavior. Moreover, as all radial trajectory points are necessarily sufficiently close to $\mathcal{F}_r$, we only require a single acceptance criterion in the alternative approach, given by equation (3.14).

In addition to superior performance, the method of forced radial trajectories has the advantage of being significantly simpler in terms of implementation. The added complexity associated with the approach of guided trajectories is mostly due to the added task of selecting appropriate trajectory initial positions along the circle interpolation C_i. Moreover, the not insignificant probability of being unsuccessful in locating a specific mesh point, even after searching from initial positions along most of C_i, necessitates elaborate exception handling. As outlined in section 3.4.6, this includes adjusting the aim point while gradually relaxing our point acceptance criteria. The differences in complexity between the two considered adaptation approaches may be seen by comparing the implementation structure overviews presented in figures 3.14 and 3.15.

Like performance, the actual mesh point positions in the approach of guided trajectories were found to be sensitive to the choice of aim points. Specifically, the accuracy of this approach, in terms of reproducing reference manifolds, was found to be highly sensitive to the algorithm for selecting aim points. For example, when replacing identification of the aim

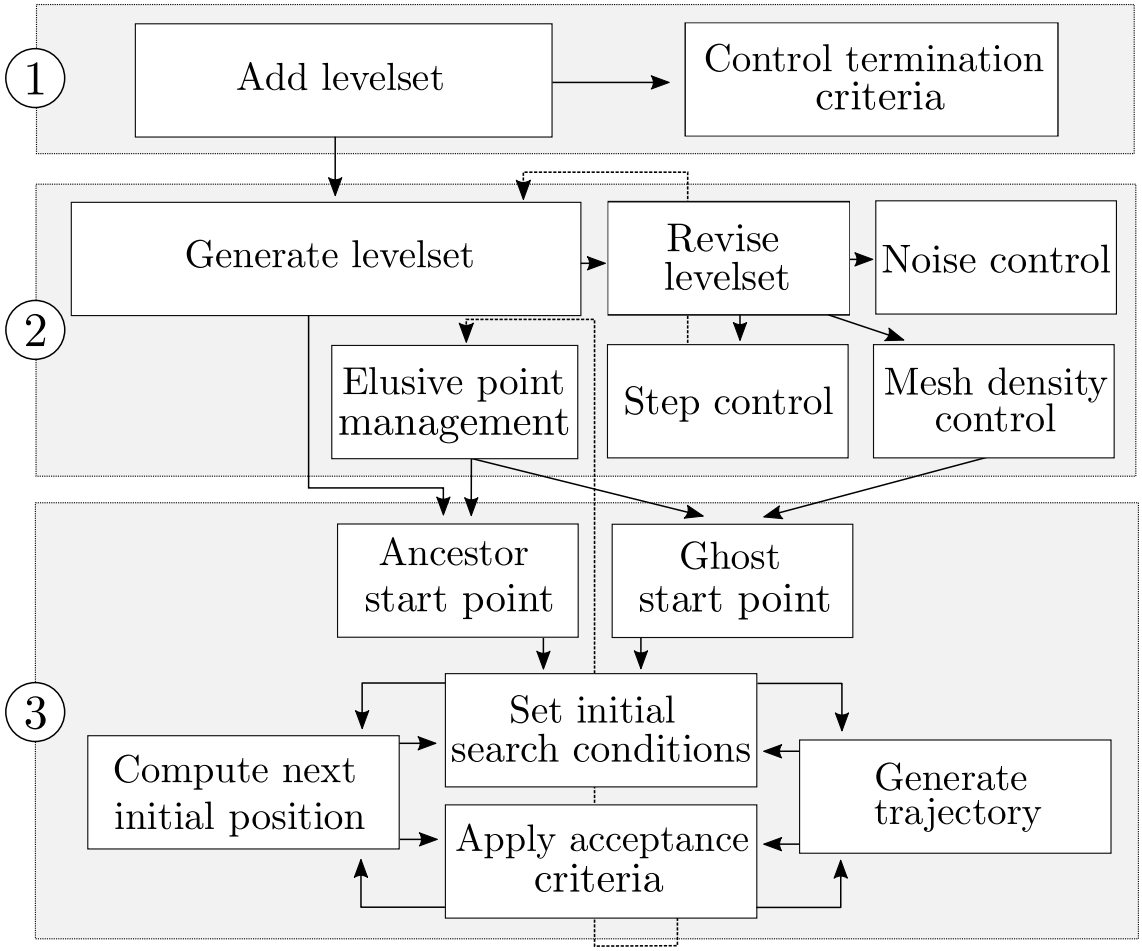


Figure 3.14: Outline of interaction between algorithm elements in the approach of guided trajectories. Indicating our approach to object orientation, the numbered gray fields represent our organization of the algorithm into separate layers, actualized as Python objects. (1) In the manifold layer we combine levelsets into the resulting manifold, constantly monitoring termination criteria and handling exceptions raised in the lower layer algorithm components. Specifically, this pertains to failure with regard to identifying points, prompting us to reduce inter-levelset step length Δ_i, or ultimately terminate the process. (2) At the geodesic levelset layer, we combine points into sets, handling exceptions raised in the point search algorithm by calling the elusive point management algorithm (see section 3.4.6). This is also where suggested sets are revised by controlling mesh density, axial angle offset, and removing unnecessary bulges and loops (see sections 3.4.4, 3.4.5, and 3.4.7, respectively). Finally, the corresponding interpolation triangles are added (see section 3.7). (3) At the point layer, new mesh points are computed either by use of an existing ancestor point, or by a ghost ancestor point chosen from the previous topological circle C_{i-1}. This start point is then used to compute a trajectory in $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$ toward the aim point $\mathbf{r}_{\text{aim}}$ within $\mathcal{F}_r$. This is done by first setting the initial conditions for our dynamic search for initial positions $C_i(s)$. We do this by imagining a trajectory from the start point $M_{i,j}$, immediately terminating as $M_{i,j}$ is in $\mathcal{F}_r$. Subsequent initial positions are then computed and used to generate trajectories providing feedback to the initial position selection algorithm. Whenever an acceptable point is found, it is returned to the geodesic levelset layer. Alternatively, if no such point is found an exception is raised.

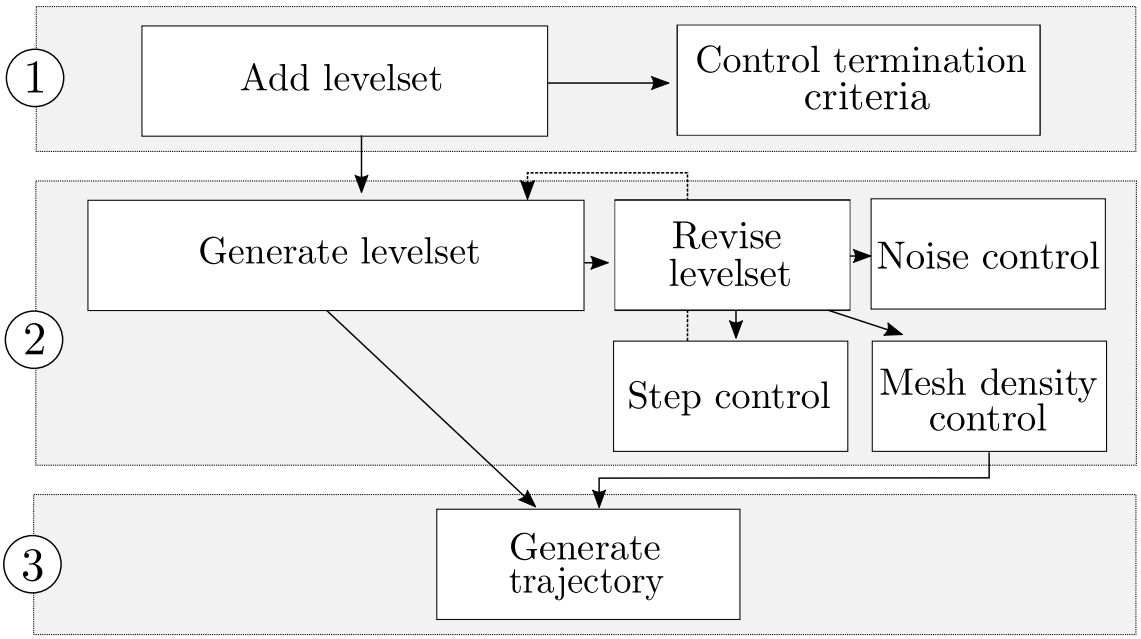


Figure 3.15: Outline of interaction between algorithm elements in the approach of radial trajectories. Indicating our approach to object orientation, the numbered gray fields represent our organization of the algorithm into separate layers, actualized as Python objects. (1) In the manifold layer we combine levelsets into the resulting manifold, constantly monitoring termination criteria and handling exceptions raised in the lower layer algorithm components. Specifically, this pertains to failure with regard to identifying points, prompting us to reduce inter-levelset step length Δ_i, or ultimately terminate the process. (2) At the geodesic levelset layer we combine points into levelsets. Note that no elusive point management routine is necessary, as we are virtually guaranteed to find an acceptable point with each attempted trajectory. This is also where suggested sets are revised by controlling mesh density, axial angle offset, and removing unnecessary bulges and loops (see sections 3.4.4, 3.4.5, and 3.4.7, respectively). Finally, the corresponding interpolation triangles are added (see section 3.7). (3) The point layer simply consists of trajectory generation, returning acceptable points to the geodesic levelset layer. Alternatively, if no such point is found, an exception is raised. Note that failure to identify an acceptable point is very rare when using forced radial trajectories.

point $\mathbf{r}_{\text{aim}}$ by linear extrapolation of $M_{i,j} - M_{i-1,j}$ with the Runge-Kutta step described in section 3.4.1, we experienced large gains both in terms of performance and accuracy. This accuracy discrepancy is surprising, as all mesh candidate points are computed by developing trajectories in $\boldsymbol{\xi}_1(\mathbf{r})$ and $\boldsymbol{\xi}_2(\mathbf{r})$, regardless of choice of aim point. Consequently, all mesh point position error should originate either from the Runge-Kutta iterative ODE solver, the initial levelset approximation (see section 3.4.1), or the topological circle interpolations C_i used to insert ghost ancestor points. Although it is conceivable that these observations could be accounted for by shortening of trajectory arc lengths, hence reducing accumulated error, this behavior lessens the credibility of the approach of guided trajectories.

Due to its superior speed, consistency, and simplicity, the approach of forced radial trajectories was found preferable. All results presented in this report are generated by use of

this method. It should however be noted that while deemed inferior for the purposes of this project, the approach of guided trajectories with appropriate choice of aim points, yielded practically identical results to those of forced radial trajectories in selected test cases.

3.7 Constructing manifold surfaces from point meshes

Having identified a point mesh M sampled from the target manifold $\mathcal{M}$, we attempt to reproduce $\mathcal{M}$ by use of a linear interpolation scheme. Linear interpolation was chosen as implementation of higher order interpolation schemes is complicated considerably by the irregular structure of M.

Our primary objective for reproducing continuous manifold approximations is visual representation, rather than providing analytical expressions for $\mathcal{M}$. Therefore, no such approximated analytical expressions are computed. Instead, three-dimensional plotting algorithms, such as the triangulated surface plotting routine `plot_trisurf` from the Python *matplotlib* library, may be used to produce visual representations of manifolds and LCSs.

These surface plotting schemes require the use of some triangulation algorithm to define the triangular surface elements constituting a manifold. Standard triangulation algorithms such as Delaunay triangulation (Berg et al., 2008) were found unsuitable for this purpose, as these do not take the specific mesh structure of M into account. For instance, Delaunay triangulation was not only found to omit necessary surface triangles, but also included undesirable surface triangles, especially close to manifold creases. A custom triangulation method was therefore devised based on the method of geodesic levelsets and the corresponding mesh structure. Like in the method of geodesic levelsets, new triangles are added by starting in the manifold initial position $\mathbf{r}_0$ and moving progressively outwards through the levelsets M_i. Within a levelset M_i, we move clockwise around C_i, adding triangles covering the surface intervening between C_i and C_{i+1}.

As illustrated in figure 3.16, this triangulation scheme primarily handles four main cases. In order to outline these cases, we consider a single point $M_{i,j}$. The first of these, displayed in figure 3.16a, is the base case where no points have been added or removed as to manage mesh density, or eliminate numerical noise. Following the convention that triangles associated with $M_{i,j}$ should cover the surface approximating the quadrilateral $M_{i,j}M_{i,j+1}M_{i+1,j}M_{i+1,j+1}$, two triangles are added. Expressed by their vertices, these are $M_{i,j}M_{i+1,j}M_{i+1,j+1}$ and $M_{i,j}M_{i,j+1}M_{i+1,j+1}$ (see figure 3.16). Notice how, when adding quadrilaterals according to this algorithm for each point, surfaces are formed between all neighboring points within M. The exception to this is the surface between the manifold initial position $\mathbf{r}_0$ and the first topological circle C_1. This surface is simply reconstructed by forming triangle surfaces $\{M_0M_{1,j}M_{1,j+1}\}_{j=1}^{n}$, where M_0 denotes the manifold initial position $\mathbf{r}_0$ and n is the number

of points in the initial levelset. Also note the implicit convention of periodic intra-levelset numbering, that is, $M_{i,j+n} = M_{i,j}$.

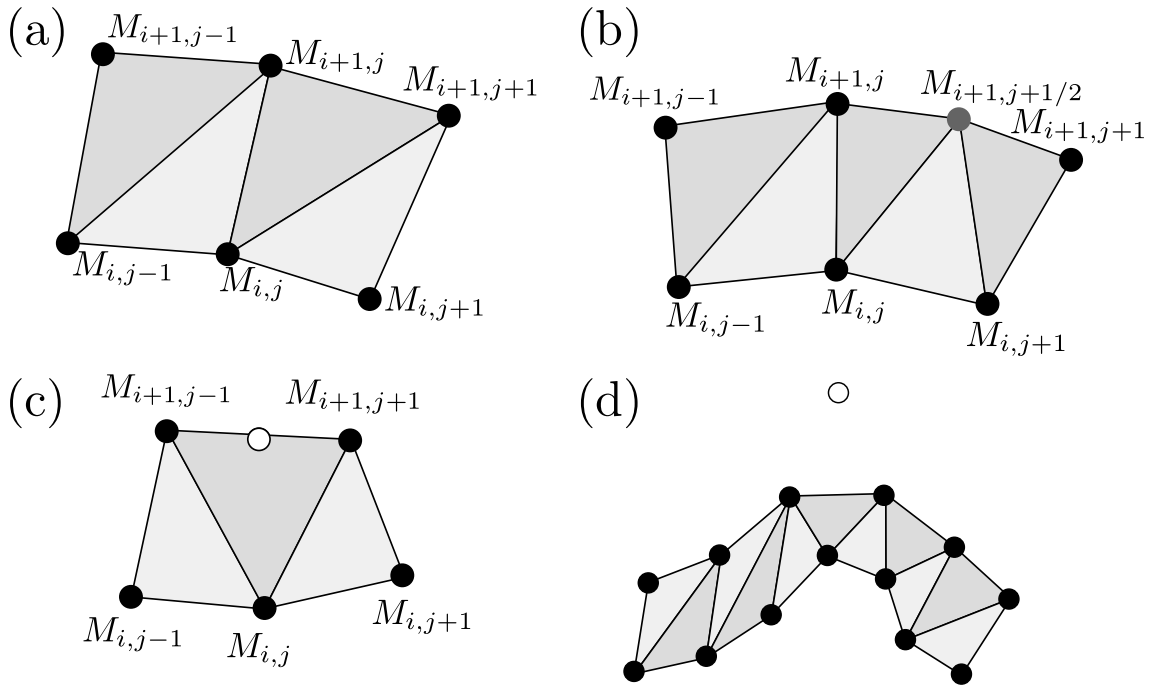


Figure 3.16: Illustration of our dedicated triangulation algorithm. (a) The fundamental case where each new point has an ancestor and no points have been removed as to preserve mesh density. Associated with point $M_{i,j}$, we then insert the triangles $M_{i,j}M_{i+1,j}M_{i+1,j+1}$ and $M_{i,j}M_{i+1,j+1}M_{i,j+1}$, each expressed by their vertices. (b) The extra point $M_{i+1,j+1/2}$ in gray has been inserted as to preserve the prescribed mesh density. Replacing the righmost quadrilateral in case (a), we now use the three triangles $M_{i,j}M_{i+1,j}M_{i+1,j+1/2}$, $M_{i,j}M_{i+1,j+1/2}M_{i,j+1}$, and $M_{i,j+1}M_{i+1,j+1/2}M_{i+1,j+1}$. (c) The point $M_{i+1,j}$ has been removed as to retain the prescribed mesh density. Here, the mesh point $M_{i,j-1}$ is merely associated with the triangle $M_{i,j-1}M_{i,j}M_{i+1,j-1}$, while $M_{i,j}$ is associated with $M_{i,j}M_{i+1,j-1}M_{i+1,j+1}$, and $M_{i,j}M_{i,j+1}M_{i+1,j+1}$. (d) A bulge has been removed. Note that while this case is handled in exactly the same way as situation (c), we could possibly be required to remove several points forming a larger bulge or loop. This instance may cause some overlap as several points in M_{i-1} form triangles with the bulge-bordering points $M_{i,j}$ and $M_{i,j+k}$. However, the uncommon nature of such sudden bulges prompted us to accept this problematic case.

The two remaining cases are necessary to handle point insertion and removal. As described in sections 3.4.4 and 3.4.7, this is necessary to preserve the specified mesh density and limit accumulation of numerical noise. When an extra point $M_{i+1,j+1/2}$ is inserted between $M_{i+1,j}$ and $M_{i+1,j+1}$, the surface approximating the quadrilateral $M_{i,j}M_{i,j+1}M_{i+1,j}M_{i+1,j+1}$ is reconstructed by use of three triangular surfaces. Again expressed by their vertices, these are $M_{i,j}M_{i+1,j}M_{i+1,j+1/2}$, $M_{i,j}M_{i,j+1}M_{i+1,j+1/2}$, and $M_{i,j+1}M_{i+1,j+1/2}M_{i+1,j+1}$ (see figure 3.16b).

Now, instead consider the final case where the point $M_{i+1,j}$ has been removed, either to preserve the desired mesh density, or to remove a bulge or loop. In this case, the surface approximating the two quadrilaterals $M_{i,j-1}M_{i,j}M_{i+1,j-1}M_{i+1,j}$ and $M_{i,j}M_{i,j+1}M_{i+1,j}M_{i+1,j+1}$

are reconstructed using the three triangular surfaces $M_{i,j-1}M_{i,j}M_{i+1,j-1}$, $M_{i,j}M_{i+1,j-1}M_{i+1,j+1}$, and $M_{i,j}M_{i,j+1}M_{i+1,j+1}$. This may be seen in figure 3.16c. Note that the nearest neighbors of the the removed mesh point $M_{i+1,j}$ are used in these triangulations, regardless of whether these are descendant points or inserted points. As may be seen in figure 3.16d, removal of loops or bulges consisting of one or more points are handled in the exact same way. Note that if several consecutive points are removed like this, the resulting triangle surface elements may be uncharacteristically large, or even partly overlap. This was however found to be a very unusual occurrence with insignificant effects on the computed LCSs.

3.8 Identifying repelling hyperbolic LCSs as manifold subsets

Starting by identifying a large number of manifolds defined by equation (2.52), developed from initial positions within U_{ABD}, we aim to identify a sufficiently comprehensive set of surfaces that satisfy LCS condition C. That is, we identify surfaces in our domain of interest that are everywhere perpendicular to the direction of maximal repulsion. For the sake of simplicity, these initial positions are selected among the grid of tracer initial positions used to compute the flow map and its derived properties. In other words, we select the subset of these tracer initial positions that fall within U_{ABD}. The number of selected manifold initial positions may be controlled by superimposing a mask onto the tracer initial position grid points, evaluating only every $n_f{}^{\text{th}}$ point in each direction. In this way, we effectively select initial positions within U_{ABD} from a grid of prescribed density. This is done in order to reduce the number of redundant initial positions, that is, initial positions that are members of the same manifold.

As the repelling hyperbolic LCSs of the system should be a subset of the resulting surfaces, we then proceed by applying existence criteria A, B and D to the computed manifolds. Consider a computed manifold mesh M. Each point $M_{i,j}$ in M is reviewed and categorized based on its inclusion in U_{ABD}, or lack thereof. This is done by evaluating equations (3.8), (3.9), and (3.10), using the interpolated eigenvalue and eigenvector fields (see section 3.2), yielding the mesh point sets $\{M_{\text{ABD}}\}$ and $\{M_{\cancel{\text{ABD}}}\}$. Note that while conditions A and B are independent of our parameter choices, D is sensitive to our choice of ϵ (see equation (3.10)). This choice determines our tolerance to offsets from the actual LCS. That is, as we compare the considered point $M_{i,j}$ with neighboring points, separated by a distance ϵ in each $\boldsymbol{\xi}_3$-direction, we could detect a peak in terms of λ_3 as long it is within the interval $[M_{i,j} - \epsilon\boldsymbol{\xi}_3(M_{i,j}), M_{i,j} + \epsilon\boldsymbol{\xi}_3(M_{i,j})]$ along $\boldsymbol{\xi}_3(M_{i,j})$.

It seems natural to choose ϵ based on the tracer initial position grid spacing; it being indicative of the smallest scale of eigenvalue field detail. That is, if we choose ϵ larger than

this grid spacing, we risk overstepping significant field behavior. Conversely, if chosen too small, we risk missing LCSs that do not pass sufficiently close by any mesh points. This is particularly crucial when selecting manifold initial positions in U_{ABD} from the tracer initial position grid points, as there is no *a priori* reason to assume that these points are close to the underlying LCSs. In conclusion, it seems reasonable to choose ϵ approximately one order of magnitude smaller than the tracer initial position grid spacing.

Starting from the manifold initial position — assumed to be part of the LCS candidate — and moving clockwise around levelsets in the order they were added, the mesh points $\{M_{\mathrm{ABD}}\}$ are included if for any of the previously added points $\{L_k\}$

$$|M_{\mathrm{ABD}} - L_k| < \Gamma_{\mathrm{ABD}} \Delta_{\mathcal{F}} \tag{3.30}$$

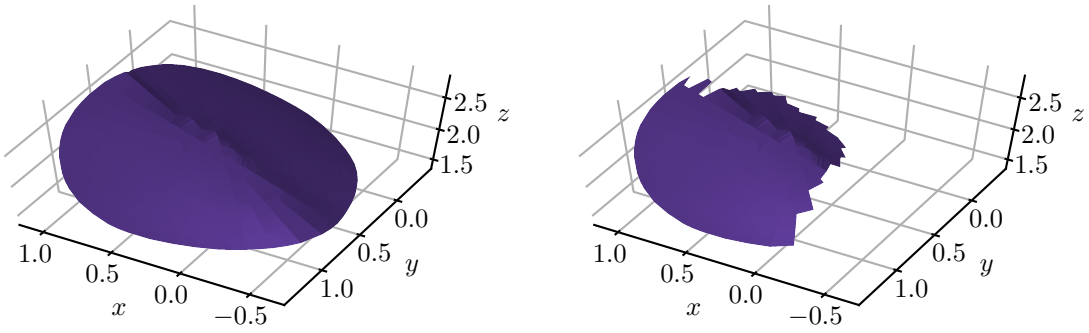
holds. That is, mesh points found to be part of U_{ABD} are added if they are sufficiently close to any previously added point. This distance threshold is defined by the maximal nearest neighbor mesh point separation $\Delta_{\mathcal{F}}$ and the scalar input parameter Γ_{ABD}. In this way, we ensure that LCSs extracted from a single manifold are in fact coherent by avoiding the addition of isolated points.

Subsequently, we add all mesh points outside U_{ABD}, $\{M_{\cancel{\mathrm{ABD}}}\}$, that satisfy

$$|M_{\cancel{\mathrm{ABD}}} - L_k| < \Gamma_{\mathrm{ABD}} \Delta_{\mathcal{F}} \tag{3.31}$$

for any k. Note that $\{L_k\}$ now consists of — and is limited to — all the previously added points from $\{M_{\mathrm{ABD}}\}$. These new points are included in order to account for numerical error with respect to the Cauchy-Green eigenvalue and eigenvector interpolations, used to determine ABD subdomain inclusion. Moreover, they support development of continuous LCS candidate surfaces that are more conducive to analysis by inspection. Note that along with the added mesh points, we also add all the corresponding interpolation triangles for which all vertices are part of the LCS. Finally, as to smoothen out our LCS boundaries, all manifold triangle surface elements for which at least two out of three vertex points have been accepted, are added to the LCS candidate visual representation. An example of this extraction process is displayed in figure 3.17.

Having extracted the manifold subsets on which conditions A, B, and D (see equations (3.8), (3.9), and (3.10)) are satisfied, we are left with surfaces that, when allowing for some numerical error, have been determined to comply with all the LCS acceptance criteria proposed by Farazmand and Haller (2012b). In the interest of clarity, the smallest LCS candidates are discarded, as these may be expected to have limited impact on the flow system (Farazmand and Haller, 2012a). This is done by computing LCS area according to equation 3.34 and defining a minimum area threshold A_{min}. Finally, as a sanity check, any LCS candi-



(a) Sample manifold prior to LCS extraction.

(b) Extracted LCS from sample manifold displayed in (a).

Figure 3.17: LCS extraction from sample manifold. The sample manifold was computed using the steady ABC flow system described in section 4.3.

dates with average repulsion $\overline{\lambda}_3$ smaller than 1, are removed. This is done to ensure that all identified repelling hyperbolic LCS are in fact repelling, as established by condition A. The remaining LCS candidates are then accepted as repelling hyperbolic LCSs.

The repulsion average $\overline{\lambda}_3$ was determined for each LCS candidate surface by a weighted average of λ_3, evaluated at all its constituent points. Note that, as the interpolated eigenvalue field may exhibit oscillatory behavior, some points were found to exhibit λ_3-values several orders of magnitude different from all neighboring points. In order to prevent small numbers of outlier points from severely skewing $\overline{\lambda}_3$, some data points were neglected. Specifically, the adjusted repulsion average $\widehat{\lambda}_3$ was computed iteratively by alternately removing $\max(\lambda_3)$ and $\min(\lambda_3)$. Each adjustment was then accepted if

$$\left| \frac{\overline{\lambda}_3}{\widehat{\lambda}_3} - 1 \right| > 0.1, \tag{3.32}$$

prompting us to accept $\widehat{\lambda}_3$ as the new baseline average $\overline{\lambda}_3$. These adjustments were repeated until neither $\max(\lambda_3)$ nor $\min(\lambda_3)$ could be removed without violating condition (3.32).

The weighting of this repulsion average was chosen to approximate the surface area represented by each sampled point $M_{i,j}$. Specifically, the weight of $M_{i,j}$, $W_{i,j}$, was computed as

$$W_{i,j} = \frac{\Delta_i + \Delta_{i-1}}{2} \frac{|M_{i,j} - M_{i,j-1}| + |M_{i,j+1} - M_{i,j}|}{2}. \tag{3.33}$$

That is, the weight of each manifold point $M_{i,j}$ is defined as to approximate the region of $\mathcal{M}$ that is closer to $M_{i,j}$ than to any other mesh point in M. The physical analogue to this weighting is illustrated in figure 3.18. Accordingly, we define the weight of the initial position point M_0 as $\pi(r_{\text{init}}/2)^2$. Moreover, the pseudo-surface area $A_{\mathcal{M}}$, corresponding to the mesh

accumulated total of these point weights, is used as an estimate for the LCS surface area. We compute $A_{\mathcal{M}}$ as

$$A_{\mathcal{M}} = \sum_{i,j} W_{i,j}. \tag{3.34}$$

Note that all weights are computed prior to extraction of LCS points. The pseudo-surface area $A_{\mathcal{M}}$ is then computed by summing the weightings corresponding to its constituent mesh points.

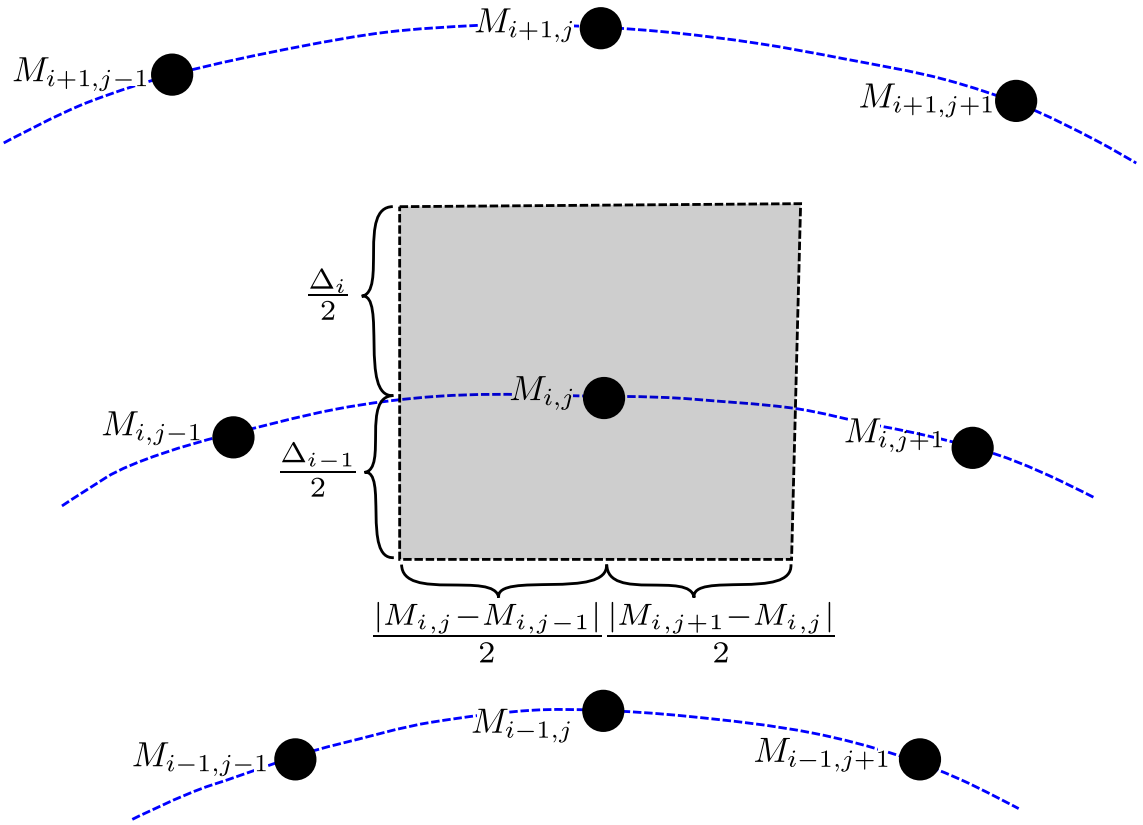


Figure 3.18: Visualization of the weighting used to compute surface average repulsion $\bar{\lambda}_3$ and surface pseudo-surface area $A_{\mathcal{M}}$. The weight of $M_{i,j}$ is an approximation of the region of $\mathcal{M}$ that is closer to $M_{i,j}$ than any other mesh point (shaded in gray). This is done by computing the area of the rectangle of sides equal to the average of $M_{i,j}$ nearest neighbor distances in the radial direction, as well as along C_i.

Note that while the preceding discussion focuses on identification of repelling hyperbolic LCSs, the method is very easily adapted as to instead identify attracting hyperbolic LCSs. This may be done by advecting tracer particles in the time-reversed interval $[t, t_0]$ and thereafter proceeding as discussed.

3.9 Managing computing time and resource requirements

Given that large numbers of manifolds have to be computed from initial positions in order to ensure sufficient coverage of potential LCSs, significant performance gains may be attained from dividing manifold development into separate processes. As each manifold may be computed independent of all other manifolds, no communication is required between the various processes. We disperse the workload among several computation cores by use of MPI. Specifically, this was done by use of the *mpi4py* library in Python. Given the levelset based structure exhibited by the method of geodesic levelsets, it seems highly impractical to parallelize the development of individual manifolds. In particular, the dynamic inter-levelset step length described in section 3.4.5 and setwise monitoring of numerical noise and self-intersections described in sections 3.4.7 and 3.4.8, would not only require significant overhead in terms of inter-process communication, but also prohibit rapidly progressing point strands from progressing ahead of slower ones.

Like development of manifolds, the flow map and flow map Jacobian "advection" is easily parallelizable with minimal needs for inter-process communication. This was done by evenly dispersing the grid particles among all available processing units and subsequently gathering the results after completing each computation. In this way, performance for computing the flow map and flow map Jacobian on large grids may be increased greatly. The potential for speedup is usually limited by the available number of processing units. Representing a much smaller fraction of the total computational load, preparation of manifold initial positions was distributed within a single cluster node by use of the Python *multiprocessing* library.

The majority of code used in this project was written in Python as to promote accessibility, as well as ease of development. However, as manifolds could consist of tens of thousands of points, each iteratively computed using a Runge-Kutta ODE solver, performance issues may quickly arise. Unsurprisingly, systematic line profiling of the code implementation revealed that time expenditure was largely concentrated in computation of point search trajectories, self-intersection checks, and noise removal.

Most prominent of these, computation of point search trajectories is performed for each new mesh point, possibly requiring a large number of Dormand-Prince method iterations. In order to improve performance, the Dormand-Prince iterative solver, as well as acceptance criteria controls, were reimplemented as C-functions. This was done by use of Cython, an optimizing static compiler that allows for calling C-code from Python, as well as tuning Python code to C performance. Specifically, the Runge-Kutta solver, as well as frequently used functions such as vector normalization and computing Euclidean norms were optimized in C and called directly from Python. This yielded large performance gains.

Similar treatment was given to the self-intersection check and noise reduction implementations, yielding major performance benefits. However, as these modules were comparatively less demanding in terms of workload, the associated absolute performance benefits were moderate.

Chapter 4

Results

Highlighting the strengths and limitations of the previously outlined method, the present chapter includes results presented as plots and charts, key insights, and accompanying considerations with regard to further application. Starting out by demonstrating the efficacy of our adapted method of geodesic levels in terms of reproducing a reference manifold, we subsequently consider a reference LCS before finally displaying some case examples. In addition to an analytical flow test example, we also investigate the oceanic currents found in a Norwegian fjord, supplied by a gridded velocity field model.

4.1 Manifold identification reference test case

In order to test the performance of the adapted method of geodesic levelsets in terms of computing manifolds defined by equation (2.52), an analytical test case was defined according to

$$\boldsymbol{\xi}_3(x, y, z) = \begin{pmatrix} 2\cos(2x)\sin(2y) \\ 2\sin(2x)\cos(2y) \\ -1 \end{pmatrix}. \tag{4.1}$$

That is, we define a $\boldsymbol{\xi}_3$-field by equation (4.1) and use that $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ are both orthogonal to $\boldsymbol{\xi}_3$ to compute a manifold defined by equation (2.52). Starting from the initial position $[\pi, \pi, \pi]$, the target manifold is analytically given by

$$z = f(x, y) = \sin(2x)\sin(2y) + \pi. \tag{4.2}$$

As to investigate convergence, 7 manifolds were computed using different mesh densities. Four of the resulting manifolds are presented in figure 4.1, while the input parameters corre-

sponding to figure 4.1c may be found in table 4.1. Note that only the mesh density defining parameters $\delta_{\mathcal{F}}$ and $\Delta_{\mathcal{F}}$, were varied, leaving the remaining input parameters constant.

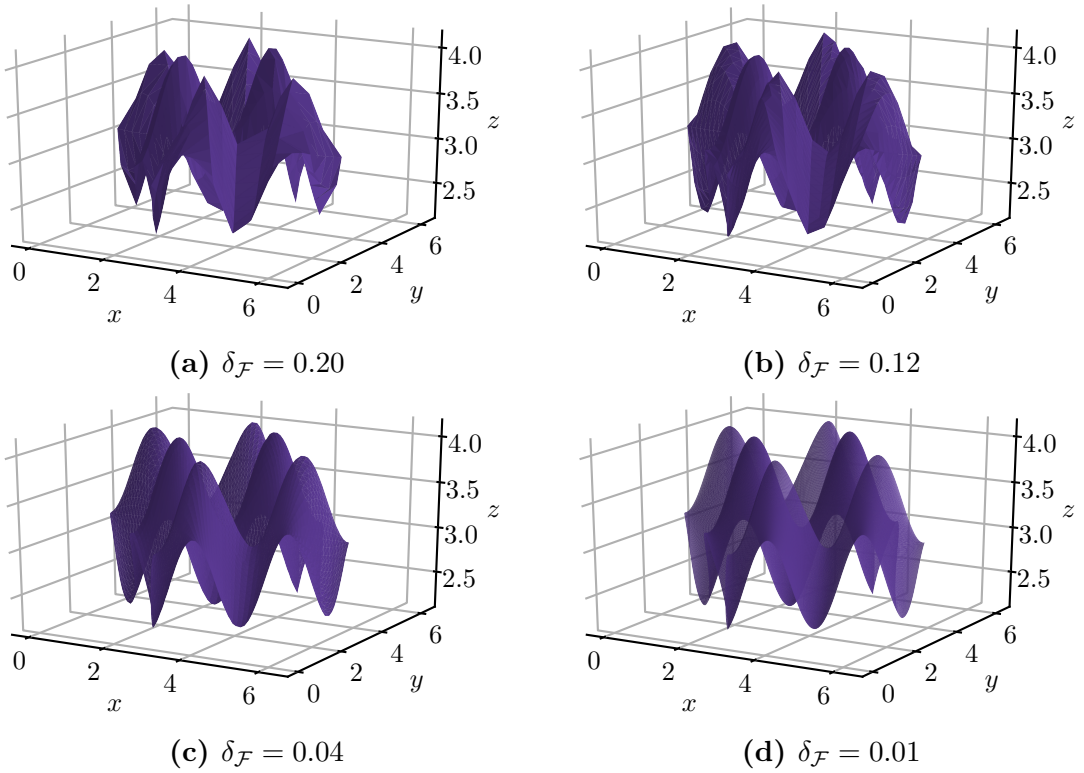
Table 4.1: Manifold computation input parameters applied to the sinusoidal field, spherical LCS, and ABC flow test cases. Note that for sinusoidal field and ABC flow convergence tests, the mesh density parameters $\delta_{\mathcal{F}}$ and $\Delta_{\mathcal{F}}$ were both varied simultaneously. That is, a factor K change in $\delta_{\mathcal{F}}$ was matched with a corresponding factor K change in $\Delta_{\mathcal{F}}$. Parameters expressed in terms of $\delta_{\mathcal{F}}$ were also changed accordingly.

Parameter	Value	Description
r_{init}	0.001	Radius of initial levelset
Δ_1	$2\delta_{\mathcal{F}}$	Initial inter-levelset step length
$\delta_{\mathcal{F}}$	0.04	Minimum nearest neighbor separation
$\Delta_{\mathcal{F}}$	0.16	Maximum nearest neighbor separation
Γ_Δ	0.005	Inter-levelset separation tolerance factor
$\alpha_{\min}$	5	Minimum axial angular offset (°)
$\alpha_{\max}$	25	Maximum axial angular offset (°)
$(\Delta\alpha)_{\min}$	$2\alpha_{\min}\delta_{\mathcal{F}}$	Lower axial angular offset parameter
$(\Delta\alpha)_{\max}$	$2\alpha_{\max}\delta_{\mathcal{F}}$	Upper axial angular offset parameter
$l_{\max}$	5	Maximum arc length to initial separation factor
$q_{\max}$	$5\delta_{\mathcal{F}}$	Maximal distance of continuous self-intersections
$r_{\max}$	2π	Maximum cumulative geodesic distance
$\Gamma_\perp$	10^{-4}	Eigenvector field orthogonality tolerance
c_{arc}	0.7	Noise removal arc length tolerance

As may be seen in figure 4.1, the manifolds computed using minimum inter-levelset separations $\delta_{\mathcal{F}}$ of 0.12, 0.08, 0.04, and 0.01 all succeed in terms of capturing the large-scale behavior of the target manifold. It is however clear that increased mesh density contributes towards increased accuracy. The relation between mesh density and error for this test case is presented in figure 4.2. Here, the average absolute value mesh point error $\bar{E}$ is computed as the average of absolute value z-offsets between mesh points and the function value of (4.1) at the corresponding xy-coordinate. Denoting the coordinates of the mesh point $M_{i,j}$ as $x_{i,j}$, $y_{i,j}$, and $z_{i,j}$, respectively, $\bar{E}$ is computed as the LCS mesh point average of

$$E_{i,j} = \left| z_{i,j} - \sin(2x_{i,j}) \sin(2y_{i,j}) - \pi \right| . \tag{4.3}$$

It seems that, in the investigated mesh density interval, the average position error of computed manifold points scales quadratically with the minimum nearest neighbor separation. This is likely primarily due to the decreased interpolation spans necessary to insert ghost ancestor points (see section 3.4.4). Moreover, as the mesh density is increased, the linear mesh interpolation becomes more accurate, accounting for the main visual discrepancies in figure 4.1.



(a) $\delta_{\mathcal{F}} = 0.20$ **(b)** $\delta_{\mathcal{F}} = 0.12$

(c) $\delta_{\mathcal{F}} = 0.04$ **(d)** $\delta_{\mathcal{F}} = 0.01$

Figure 4.1: Convergence test for the sinusoidal field target manifold described in equation (4.2). Note that all non-mesh density parameters are kept constant (see table 4.1). We notice that, while lacking the apparent smoothness of higher mesh density manifolds, subfigures (a) and (b) display all the appropriate principal characteristics of the target manifold. Moreover, while using approximately a factor 1/16 as many mesh points as the highest density alternative, the manifold representation in (c) is visually indistinguishable from (d).

Like manifold point position accuracy, the number of points necessary to constitute a manifold of a given size increases quadratically as minimum nearest neighbor separation is decreased. From this we may infer an approximately reciprocal function relationship between average mesh point error $\bar{E}$ and mesh density. Note that this increased mesh density is expensive both in terms of computation time and working memory. This is because our most expensive operations — computing trajectories to identify new points and performing self-intersection checks — scale linearly and quadratically with respect to number of manifold points, respectively. Moreover, our memory requirements naturally increase as the number of mesh points increases.

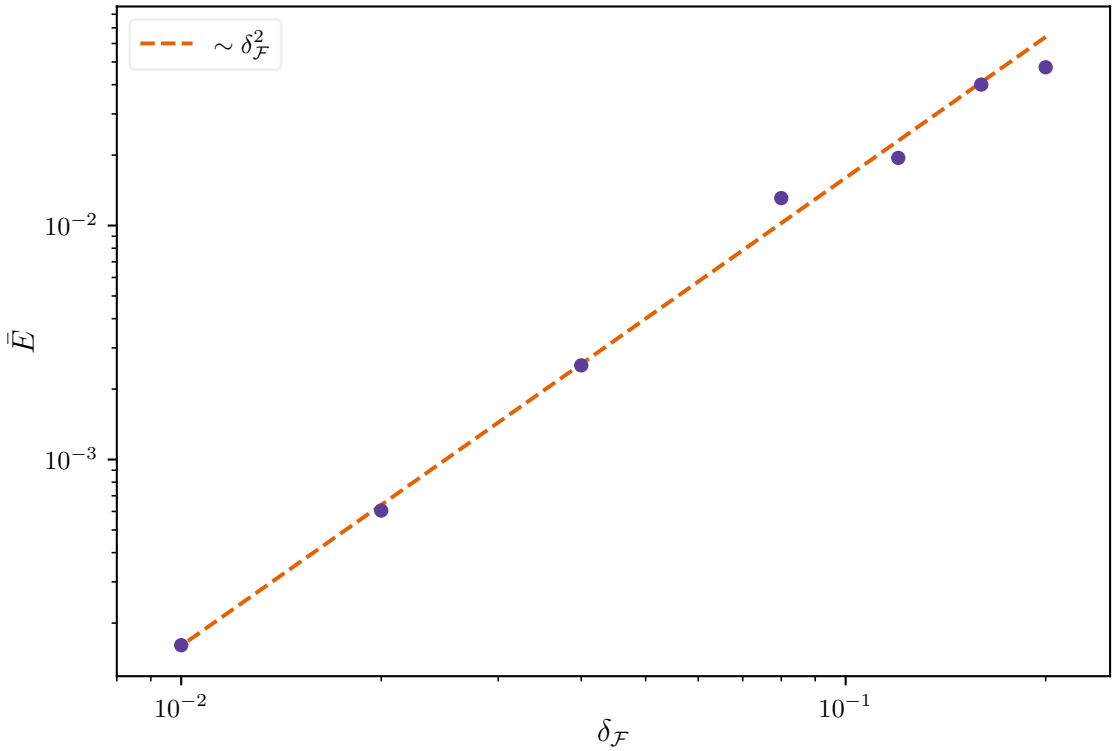


Figure 4.2: Error scaling of the sinusoidal field (see equation (4.2)) convergence test. Denoting the average deviation of mesh points from the target manifold $\bar{E}$, this average error is plotted as a function of the minimum nearest neighbor separation $\delta_{\mathcal{F}}$. Using $\delta_{\mathcal{F}}$ as an indicator variable for mesh density, we uncover a quadratic relation $\bar{E} \approx \delta_{\mathcal{F}}^2$. Note that as grid density, that is points per area, is expected to scale with $1/\delta_{\mathcal{F}}^2$ for this test, we anticipate a reciprocal function relationship between average mesh point error and mesh density.

4.2 Lagrangian coherent structure identification reference test case

Verifying the efficacy of our method for identifying repelling hyperbolic LCSs requires a reference test example. An analytical flow system displaying a single spherical repelling hyperbolic LCS for short time intervals is given by

$$\dot{\mathbf{x}} = \begin{pmatrix} x \sin(\pi(|\mathbf{x}| - k))/|\mathbf{x}| \\ y \sin(\pi(|\mathbf{x}| - k))/|\mathbf{x}| \\ z \sin(\pi(|\mathbf{x}| - k))/|\mathbf{x}| \end{pmatrix}, \tag{4.4}$$

where k is the radius of the spherical LCS and $|\mathbf{x}| = \sqrt{x^2 + y^2 + z^2}$. As may be seen in figure 4.3, this system exhibits a sharp peak in terms of radial repulsion at $|\mathbf{x}| = k$. This was assessed by transporting an evenly distributed $200 \times 200 \times 200$ tracer particle grid covering the domain $U = (x, y, z) \in [-2, 2]^3$ for 1 time unit with $k = 1$. The corresponding Cauchy-Green eigenvalues and eigenvectors were then computed as described in chapter 3. The resulting

average $\bar{\lambda}_3(|\mathbf{x}|)$, obtained by use of SVD (see section 2.2.4), was then plotted as a function of radius. Given that we have $\dot{\mathbf{x}}(|\mathbf{x}| = 1) = 0$, there is no non-radial repulsion here. We therefore expect to find a repelling hyperbolic LCS forming a sphere of radius $k = 1$. Note that these considerations only hold for short time intervals, as trajectories in equation (4.4) eventually converge at stationary points. The rather short time interval of 1 time unit was chosen to account for this system feature.

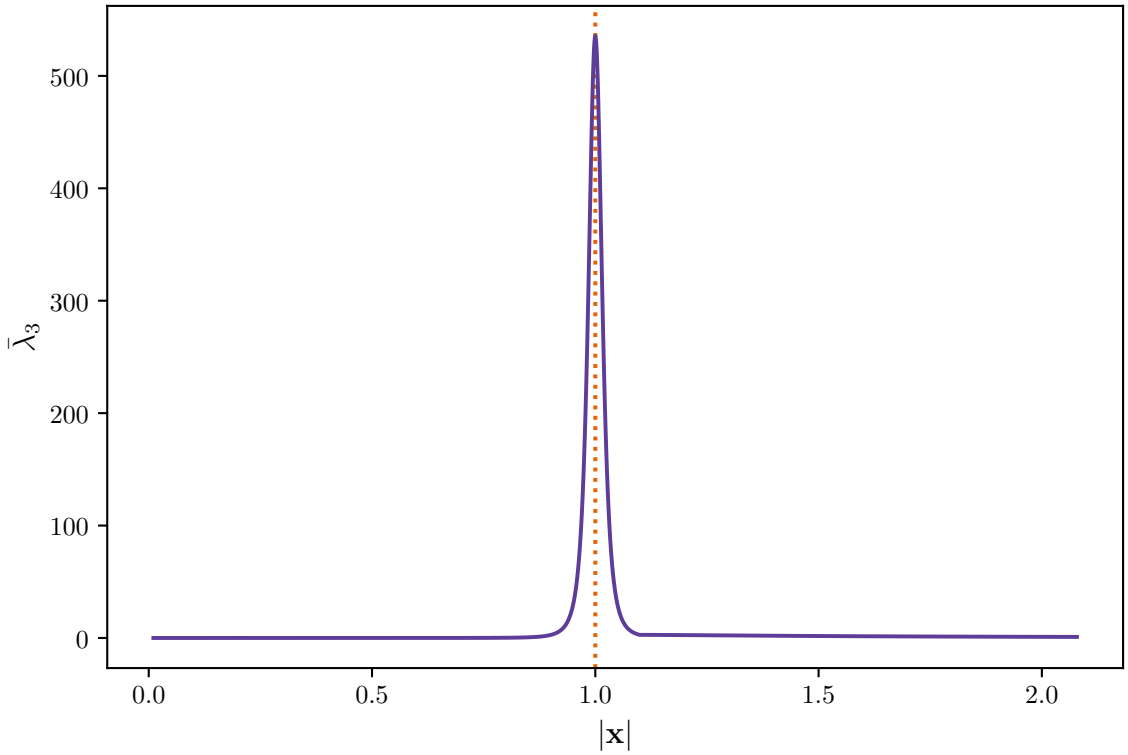


Figure 4.3: Domain average λ_3 as a function of distance from the origin. We notice that, while repulsion remains very modest for most of the interval, we have a sharp peak near $|\mathbf{x}| = 1$. This result indicates that any manifold following the sphere of radius $|\mathbf{x}| = 1$ is exceedingly likely to be a repelling hyperbolic LCS, as such a surface would function as a local maximizer in terms of λ_3.

Employing the same particle grid covering the domain $U = (x, y, z) \in [-2, 2]^3$, 291 initial positions within U_{ABD} (see equations (3.8), (3.9), and (3.10)) were computed and used to identify candidate manifolds. Note that these initial positions were selected from a total of 2329833 candidate positions using $n_f = 20$ (see section 3.8). The resulting manifolds were computed using the parameters displayed in table 4.1. Repelling hyperbolic LCSs were then extracted from the candidate manifolds using the method described in section 3.8, employing a smoothing tolerance of $\Gamma_{\text{ABD}} = 1.2$, $A_{\min} = 1$, and $\epsilon = 0.005$ (see equation 3.10). Note that this ϵ is one fourth of the tracer initial position grid spacing of 0.02. The result, displayed in figure 4.4, was a total of 3 LCSs, all clustered at the sphere of radius $k = 1$. As the target

LCS was successfully reproduced, avoiding false positives, this test example was found to substantiate the efficacy of our method.

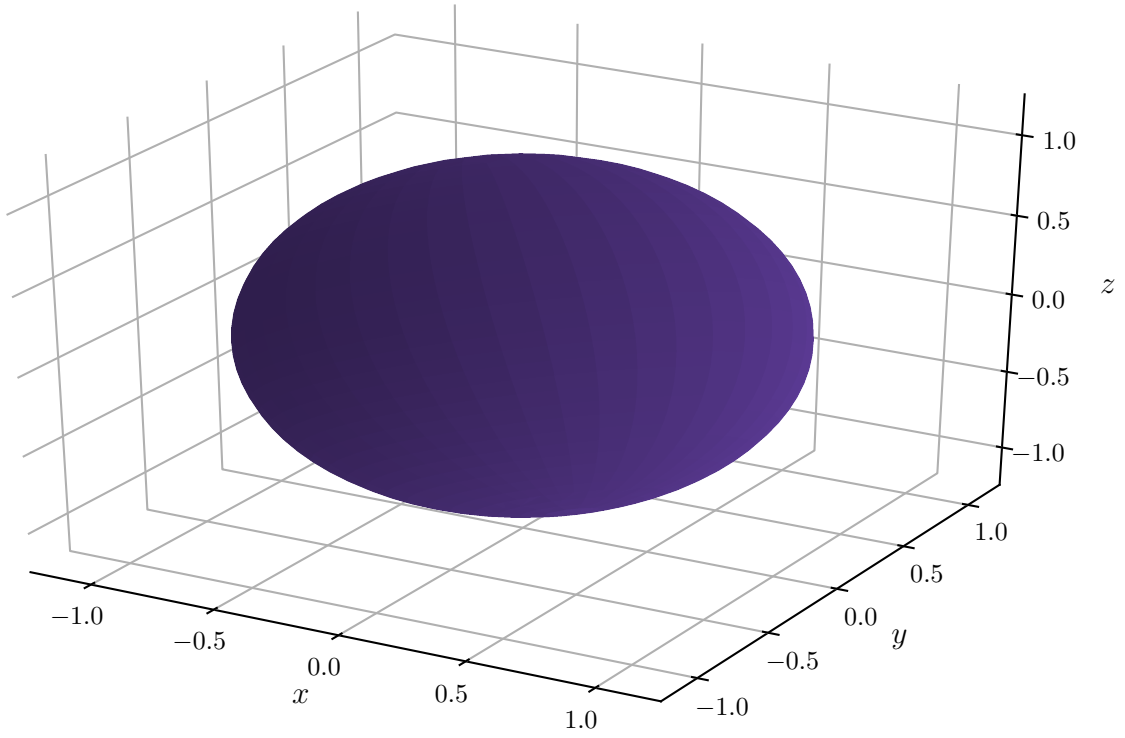


Figure 4.4: Identified LCSs in the system described by equation (4.4), considering the time interval $t \in [t_0, t_0 + 1]$. The resulting LCS structure consists of 3 component structures, each overlapping onto the sphere of unit radius. Noting the sharp peak in repulsion observed in figure 4.3, this result seems reasonable. Moreover, we note from the velocity field definition (see equation (4.4)) that $\dot{\mathbf{x}}(\mathbf{x}) = 0$, as long as $|\mathbf{x}| = 1$. Therefore, at $|\mathbf{x}| = 1$, all repulsion must necessarily be radial. These considerations prompt us to expect to find this spherical LCS of unit radius.

4.3 Steady ABC flow

Previously targeted for investigated with regard to LCS theory by for example Oettinger and Haller (2016), the Arnold-Beltrami-Childress flow (ABC flow) is a three-dimensional solution to the Euler equations given by

$$\begin{aligned} \dot{x} &= A\sin(x) + C\cos(y) \\ \dot{y} &= B\sin(x) + A\cos(z) \\ \dot{z} &= C\sin(y) + B\cos(x). \end{aligned} \tag{4.5}$$

Here, A, B, and C are scalars. Note that the ABC flow is divergence free and displays 2π-periodicity in x, y, and z.

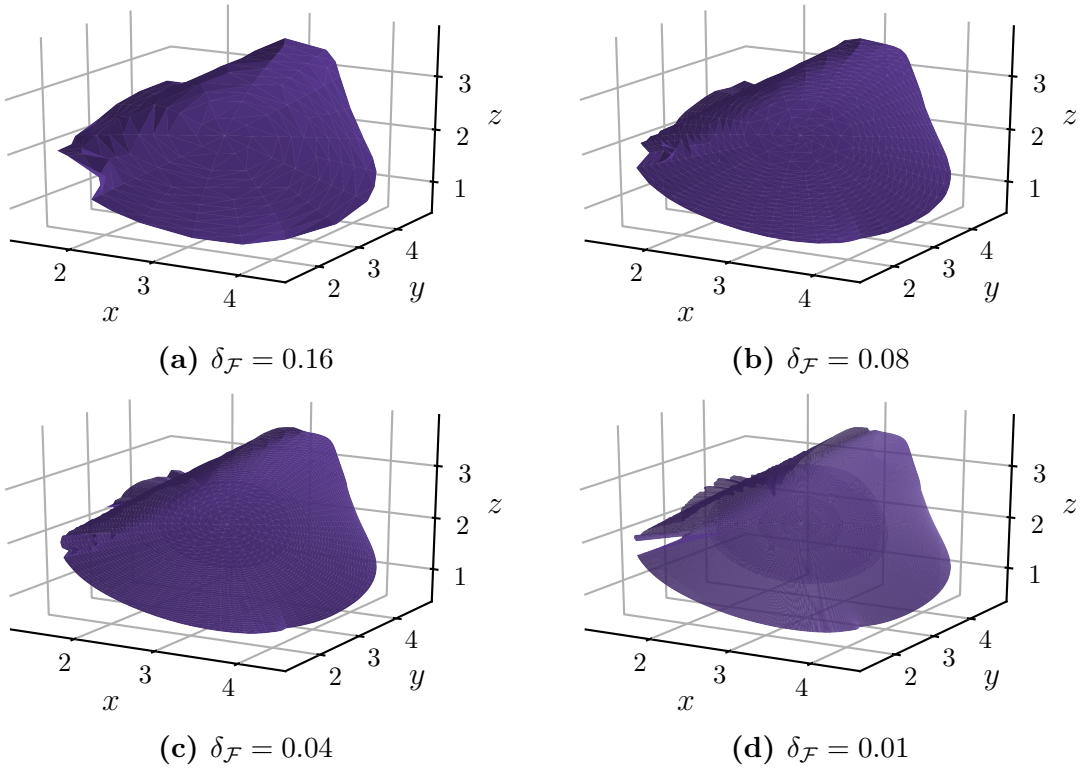
Using the same scalar parameters as Oettinger and Haller (2016); $A = \sqrt{3}$, $B = \sqrt{2}$, and $C = 1$, a grid of $256 \times 256 \times 256$ particles evenly distributed over the domain U given by $(x, y, z) \in [0, 2\pi]^3$ was advected according to the description in section 3.1 over the time interval $t \in [0, 5]$. Having computed the corresponding Cauchy-Green eigenvalues and eigenvectors by use of SVD, continuous $\boldsymbol{\xi}_3$-, λ_2-, and λ_3-fields were reconstructed by use of periodic tricubic interpolation (see sections 3.2.1 and 3.2.2).

With the purpose of limiting memory requirements, as well as computation time — while still preserving reasonable manifold accuracy — convergence tests were conducted in order to determine adequate mesh density for this system. A representative result, focusing on a single manifold, is displayed in figure 4.5. We notice that more details are distinguished as mesh density is increased. It is however clear that all displayed manifold reproductions capture the same principal structures. Moreover, we observe that the manifold reproduction presented in figure 4.5d exhibits some undesirable behavior. Specifically, the bulge visible at approximately $x = 2$, $y = 2$, $z = 1.5$ may be the result of numerical noise, as this structure seems uncharacteristic of the surrounding neighborhood.

Noting our previous considerations with respect to performance and accuracy highlighted in section 4.1, as well as the noise found in very high density meshes, it seems reasonable to continue using the input parameters displayed in table 4.1. Limiting mesh density according to these parameters allows us to compute sufficiently large manifolds without exhausting available memory.

Given that all trajectories defined by equation (3.18) should be entirely contained in a single manifold, such trajectories are useful in terms of confirming manifold behavior. As can be seen in the example case displayed in figure 4.6a, developing 200 trajectories from the same initial position on the manifold substantiates that our selected manifold successfully captures the dynamics of the interpolated Cauchy-Green eigenvector field. This was generally found to be the case. Note that these trajectories are each defined as a unique linear combination of $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$.

While displaying convincing agreement with our manifold representation, we observe that these trajectories in $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ tend to converge on some parts of the manifold, while evading other regions. In order to confirm the behavior of the manifold in these evaded regions, a second set of tests were carried out using radially forced trajectories (see equation (3.27)). Still employing the test case from figure 4.6a, these trajectories are presented in figure 4.6b. The highly convincing agreement between these trajectories and our manifold representation is however expected, as all manifold points are computed using trajectories defined by equation (3.27).



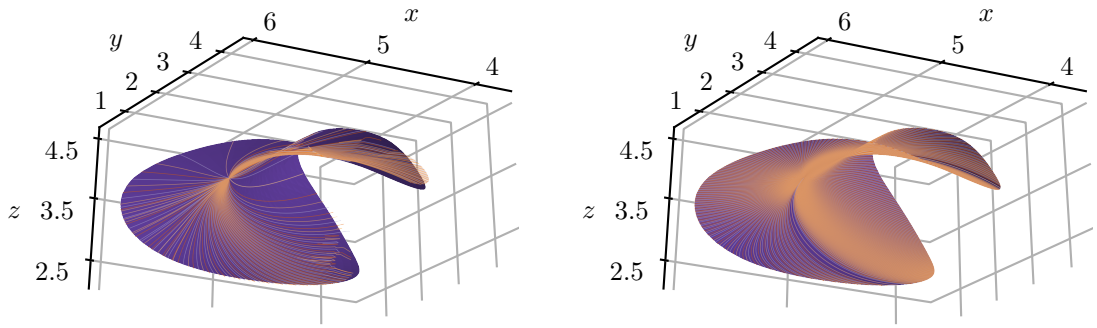
(a) $\delta_{\mathcal{F}} = 0.16$ **(b)** $\delta_{\mathcal{F}} = 0.08$

(c) $\delta_{\mathcal{F}} = 0.04$ **(d)** $\delta_{\mathcal{F}} = 0.01$

Figure 4.5: Convergence test for a selected steady ABC flow manifold. Note that all non-mesh density parameters are kept constant (see table 4.1). We also notice that while lacking the apparent smoothness of higher mesh density manifolds, figures (a) and (b) display all the appropriate principal characteristics of the higher density representations. Another interesting aspect of this convergence test is the noise increase observed for $\delta_{\mathcal{F}} = 0.01$. That is, we observe oscillatory behavior that is uncharacteristic of the surrounding domain.

Using the Cauchy-Green eigenvector and eigenvalue fields with equations (3.8), (3.9), and (3.10), we evaluate the previously described $256 \times 256 \times 256$ particle grid according to the method described in section 3.8. The grid points that satisfy conditions A, B, and D, not only form our set of manifold initial positions, but also approximate the underlying ABD subdomain U_{ABD}, where LCSs may exist. This ABD subdomain representation corresponding to the steady ABC flow of equation 4.5 is displayed in figure 4.7.

Selecting 618 initial positions from the gridded steady ABC flow ABD subdomain of 340951 candidate positions using $n_f = 8$, manifolds were developed and LCSs extracted according to the method described in chapter 3. These LCSs were extracted using an ABD subdomain tolerance Γ_{ABD} of 1.75 (see section 3.8), a minimum pseudo-surface area of $A_{\text{min}} = 6$, and $\epsilon = 0.005$. Note that this ϵ corresponds to approximately one fifth of the tracer initial position grid spacing. The resulting LCS structure, consisting of 22 LCS element surfaces, is highlighted in figure 4.8. We observe that although 22 LCS surfaces were identified, these form two largely smooth and coherent structures. While these structures are partly



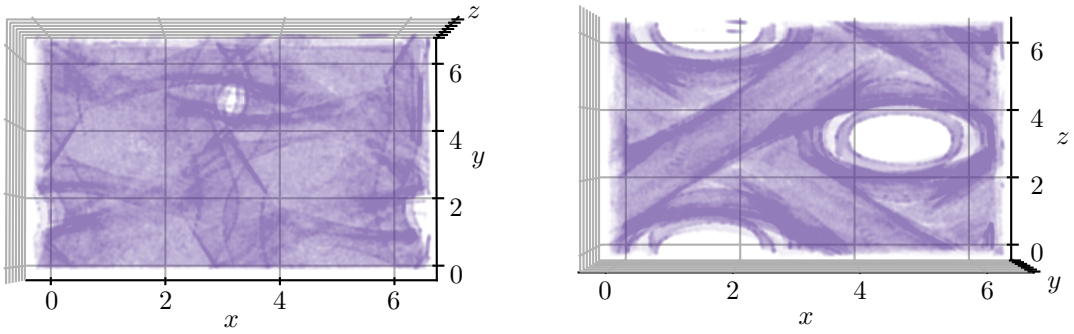
(a) Trajectories as linear combinations of $\boldsymbol{\xi}_1(\mathbf{x}_0)$ and $\boldsymbol{\xi}_2(\mathbf{x}_0)$ superimposed onto a sample manifold.

(b) Forced radial trajectories computed according to equation (3.27) superimposed onto a sample manifold.

Figure 4.6: Trajectories within the target manifold defined by equation (2.52) superimposed onto the corresponding sample computed manifold. Notice that while the trajectories in (a) correspond very well to the computed manifold, they converge and diverge according to the local nature of the underlying field, making certain regions hard to investigate. In order to propely verify the manifold behavior in these regions, a second test was devised by use of the forced radial trajectories described in section 3.5. Note that the highly convincing agreement displayed in (b) is unsurprising, as these trajectories are computed in the same way as the manifold mesh points.

connected, they are clearly distinct and therefore highlighted in different colors. Note the clear correspondence between our ABD subdomain and the accompanying LCSs. This is particularly evident when inspecting figure 4.7b and figure 4.8b, noticing the middle-right side tubular structure. Note that the viewing angles of figures 4.7a, 4.7b, 4.7c, and 4.7d correspond to those of figures 4.8a, 4.8b, 4.8c, and 4.8d, respectively.

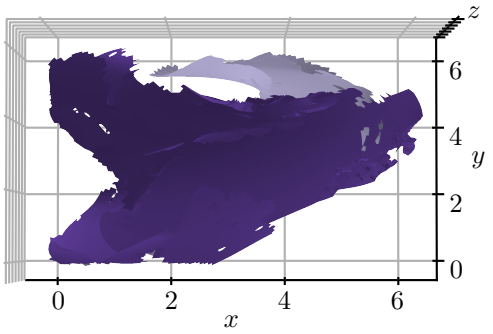
Having extracted repelling hyperbolic LCSs from manifold candidates, we expect particles initially situated on opposite sides of the LCS to diverge rapidly under advection in the original velocity field over the considered time interval. As highlighted in figure 4.9, this was tested by advecting two sets of initial positions, each situated on opposite sides of an identified LCS over the considered time interval $t \in [0, 5]$. We observe that while the LCS triangulation breaks down, the opposite-side particle groups have diverged notably from figure 4.9a to 4.9b. Moreover, each particle group remains fairly compact, indicating that the region of large stretching is situated between them. Note the large stretching observable between the LCS mesh points. From this we infer that the target LCS crosses our LCS reproduction, leaving significant portions of the mesh points on each side. This behavior is not unexpected, as LCSs have infinitesimal width. Due to this infinitesimal width, any numerical errors are expected to leave LCS mesh points slightly off the target LCS.



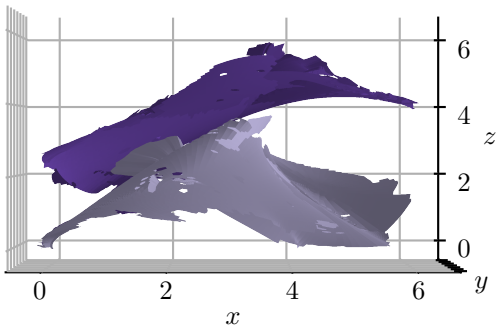
(a) ABD subdomain viewed along the z-axis. **(b)** ABD subdomain viewed along the y-axis.

(c) ABD subdomain viewed along the x-axis. **(d)** ABD subdomain isometric view.

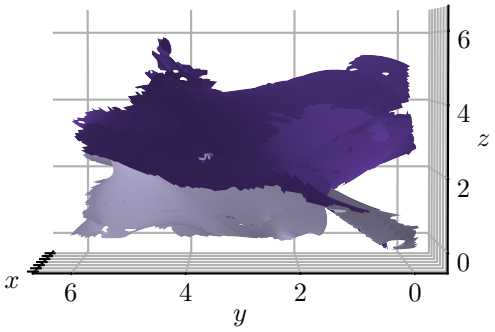
Figure 4.7: Steady ABC flow subdomain U_{ABD} from various viewing angles. The ABD subdomain is here represented as a partly transparent scatter plot of the constituent tracer initial position grid points. That is, degree of coloring indicates ABD subdomain concentration along the respective viewing angles. Note the clearly distinct tubular structures, particularly visible in (b) and (c).



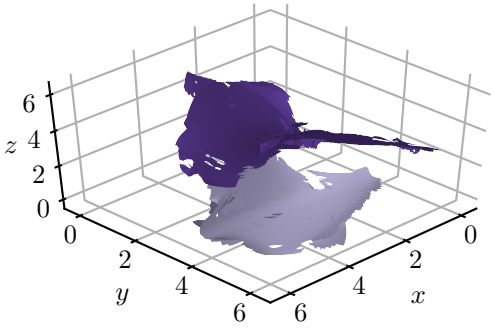
(a) LCSs viewed along the z-axis.



(b) LCSs viewed along the y-axis.

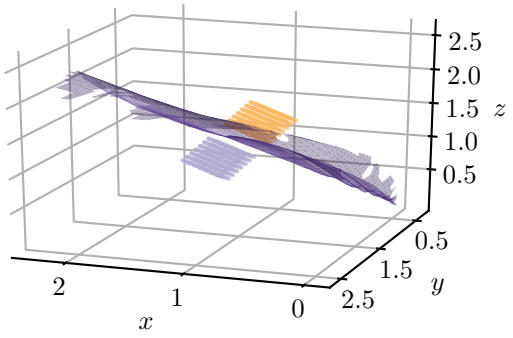


(c) LCSs viewed along the x-axis.

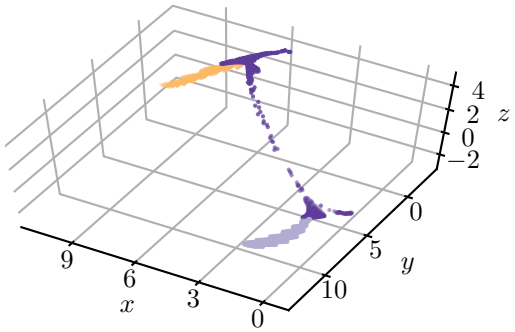


(d) LCS isometric view.

Figure 4.8: Steady ABC flow repelling hyperbolic LCSs from various viewing angles, each main structure indicated by distinct coloring. Note the agreement between the ABD subdomain displayed in 4.7 and the corresponding LCSs.



(a) Sample LCS and selected tracers prior to advection ($t = t_0$).



(b) Sample LCS points, as well as the selected tracer points after advection ($t = t_0 + 5$).

Figure 4.9: LCS fragment along with selected tracer particles prior to and after advection over the selected time interval $t \in [0, 5]$. Note that while the coherent LCS surface structure breaks down after advection, the selected tracer particles are clearly separated based on initial placement. Also notice the change in scale between (a) and (b), witnessing the large repulsion we expect to find at the LCS. While the breakdown of the LCS surface structure is undesirable, it is not unexpected. This is because the infinitesimal width of the target LCS structure makes mesh point placement error unavoidable. An accurate LCS identification algorithm is therefore still likely to place similar portions of the constituent mesh points on each side of the actual LCS, likely resulting in extensive stretching like we observe in (b).

4.4 Unsteady ABC flow

As demonstrated by Oettinger and Haller (2016), we may modify the ABC flow system by perturbing the scalar parameters A, B, and C, producing a similar unsteady flow example. This system is defined by

$$\begin{aligned} \dot{x} &= A\sin(x) + \tilde{C}(t)\cos(y) \\ \dot{y} &= \tilde{B}(t)\sin(x) + A\cos(z) \\ \dot{z} &= \tilde{C}(t)\sin(y) + \tilde{B}(t)\cos(x) \end{aligned} \quad . \tag{4.6}$$

where

$$\begin{aligned} \tilde{B}(t) &= B + B \cdot k_0 \tanh(k_1 t)\cos((k_2 t)^2) \\ \tilde{C}(t) &= C + C \cdot k_0 \tanh(k_1 t)\sin((k_3 t)^2) \end{aligned} \quad . \tag{4.7}$$

Like Oettinger and Haller (2016), we choose $k_0 = 0.3$, $k_1 = 0.5$, $k_2 = 1.5$, and $k_3 = 1.8$. The effect of introducing this time dependence can be seen in figure 4.10. We observe that the range of the oscillations in $\tilde{B}$ and $\tilde{C}$ is over half the magnitude of B and C, respectively. We therefore expect significant differences in terms of field behavior, although the principal field behavior remains.

Treating this unsteady flow system in the same way as its steady counterpart (see section 4.3), we arrive at the ABD subdomain displayed in figure 4.11. This result is surprisingly similar to its steady flow equivalent, displaying all the same principal structures. There are however significant differences, both in terms of the sizes of the ABD subdomain openings, as well as the general distribution density of ABD subdomain points. For example, by closely inspecting figures 4.7a and 4.11a, we notice that the upper middle gap structure is somewhat larger in the unsteady case. We also notice that the rather sharp outlines visible in figure 4.7b are accompanied by larger dark patches, indicating high density of ABD subdomain points in figure 4.11b. Again, note that the viewing angles of figures 4.7a, 4.7b, 4.7c, and 4.7d correspond to those of figures 4.11a, 4.11b, 4.11c, and 4.11d, respectively.

Once again, we treat this unsteady ABC flow case in the same way as its steady equivalent. Now with 676 initial positions extracted from 361461 candidate positions using $n_f = 8$, we develop manifolds using the parameters displayed in table 4.1. LCSs are then extracted using $\Gamma_{\text{ABD}} = 1.75$, $A_{\min} = 6$ and $\epsilon = 0.005$. Note that this ϵ again corresponds to approximately one fifth of the tracer initial position grid spacing. The resulting LCS formations are presented in figure 4.12. These structures consist of 31 LCS surface elements organized into three separate formations indicated by coloring. Although displaying larger differences from its

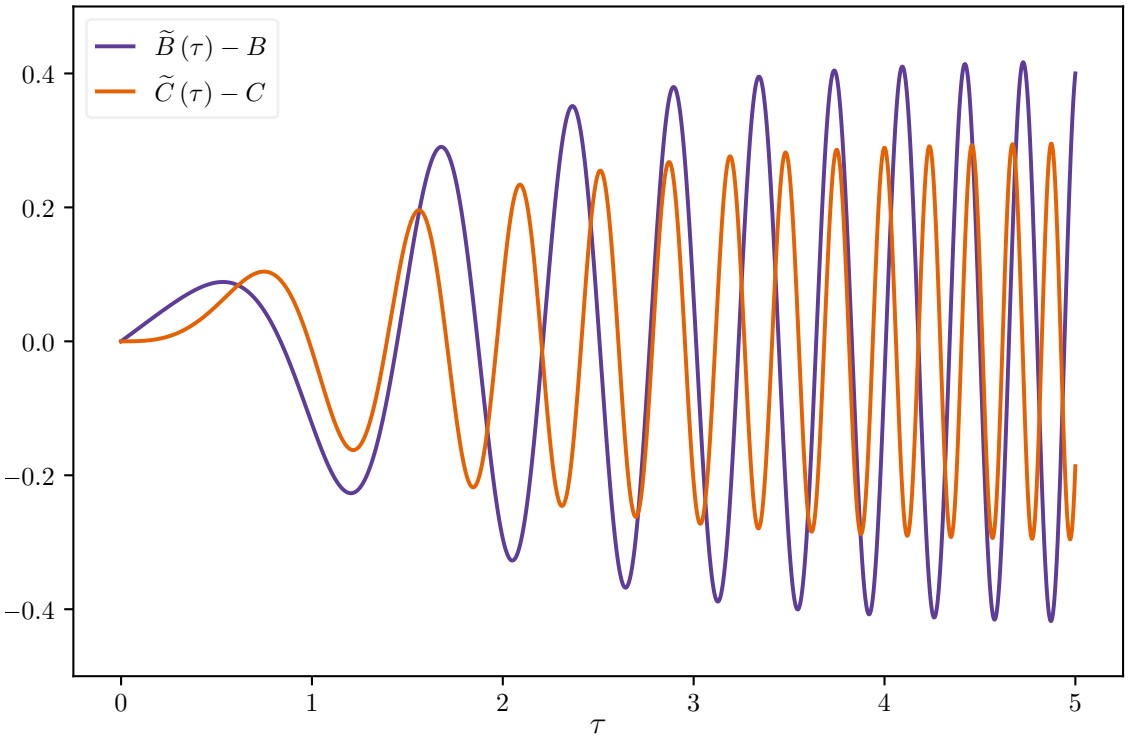
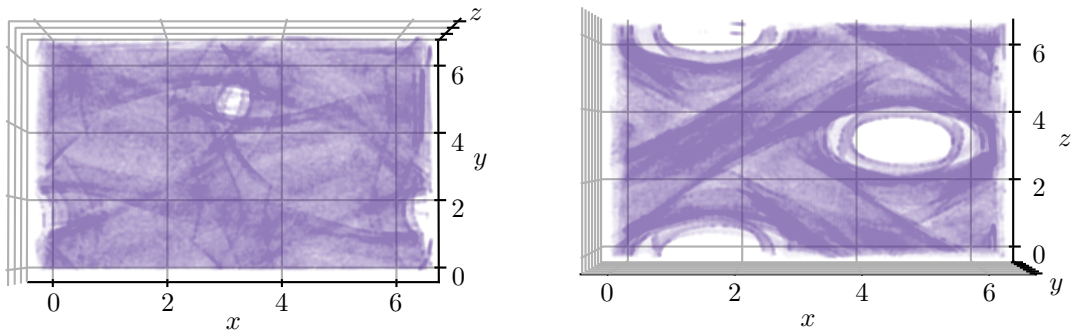


Figure 4.10: ABC flow coefficient perturbations over time τ. We note that the perturbations to $B = \sqrt{2}$ and $C = 1$ involve oscillations of range approximately equal to half the unperturbed coefficients.

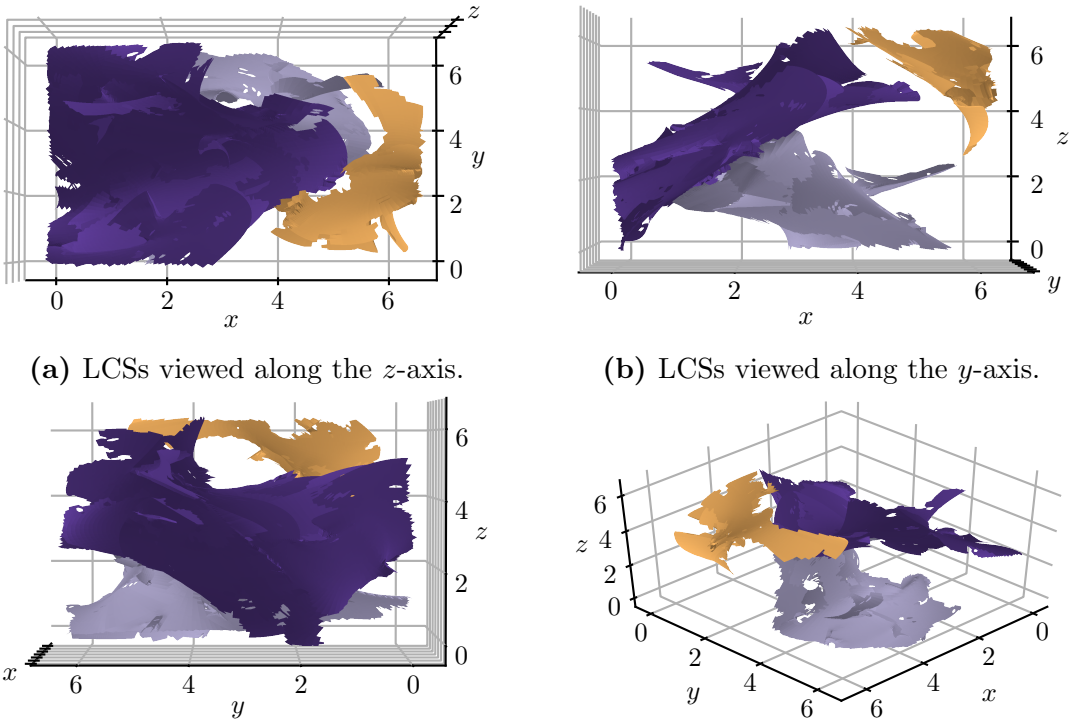
steady counterpart (see figure 4.8) than seen for the steady and unsteady ABD subdomains, there are clear similarities. This is particularly clear when inspecting figures 4.7b and 4.11b, as well as figures 4.7d and 4.11d. We notice that the two largest structures in the unsteady case correspond very well to the structures found in the steady case. In addition to these we have gained a third formation, separate from the original two. This new addition seems reasonable by inspection of the accompanying ABD subdomain, for instance completing the middle-left tubular structure now visible in both figures 4.11b and 4.12b.



(a) ABD subdomain viewed along the z-axis. **(b)** ABD subdomain viewed along the y-axis.

(c) ABD subdomain viewed along the x-axis. **(d)** ABD subdomain isometric view.

Figure 4.11: Unsteady ABC flow subdomain U_{ABD} from various viewing angles. The ABD subdomain is here represented as a partly transparent scatter plot of the constituent tracer initial position grid points. That is, degree of coloring indicates ABD subdomain concentration along the respective viewing angles. Note the clearly distinct tubular structures, particularly visible in (b) and (c), as well as the apparent similarities to the steady ABC flow ABD subdomain (see figure 4.7).



(a) LCSs viewed along the z-axis.

(b) LCSs viewed along the y-axis.

(c) LCSs viewed along the x-axis.

(d) LCS isometric view.

Figure 4.12: Unsteady ABC flow repelling hyperbolic LCSs from various viewing angles, each main formation indicated by distinct coloring. Note the agreement between the ABD subdomain displayed in 4.7 and the corresponding LCSs. Moreover, two of the three identified LCSs are very similar to the steady ABC flow LCSs displayed in figure 4.8.

4.5 Gridded ocean model data example

More typical of real world application, we finally apply our method to a gridded model data set describing the oceanic currents of the Førde fjord in western Norway. This case is of particular interest due to current plans for using the fjord as a mine tailings deposit. The data set covers large parts of the fjord over six days, using time steps of one hour. However, as tracer behavior at the coastline is ill-defined, tracer initial positions were selected within a 500×500 m region in the middle of the fjord. In this way we ensure that very few tracers reach the coastline. The exact location of this domain U within the larger Førde fjord area may be found highlighted by a white square frame in figure 4.13.

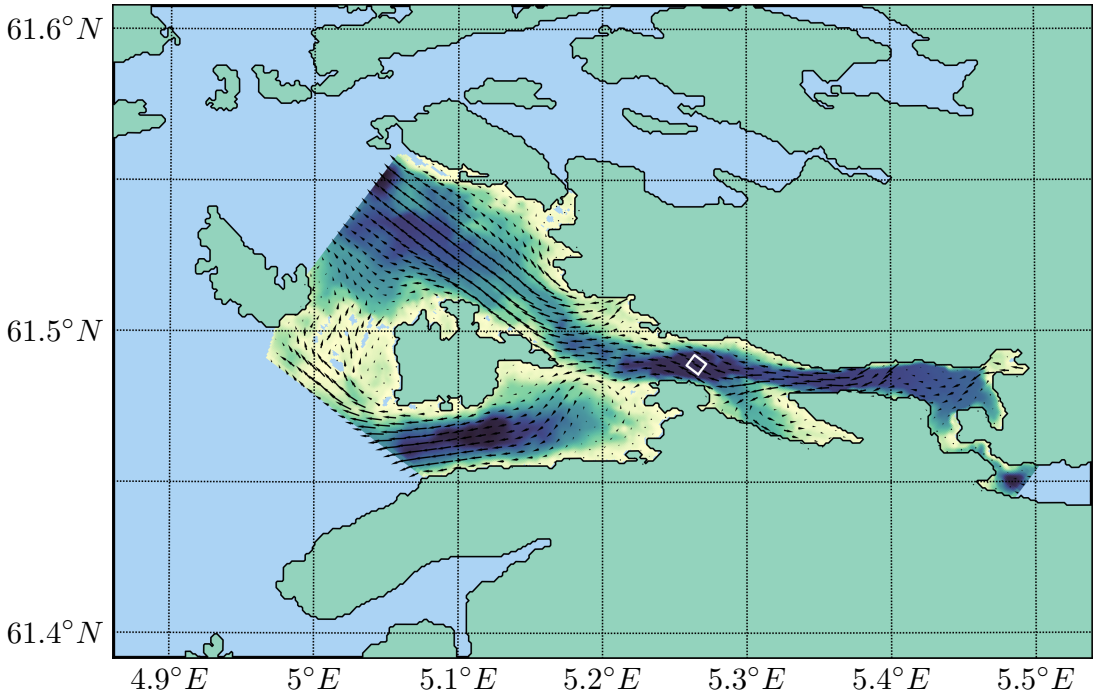
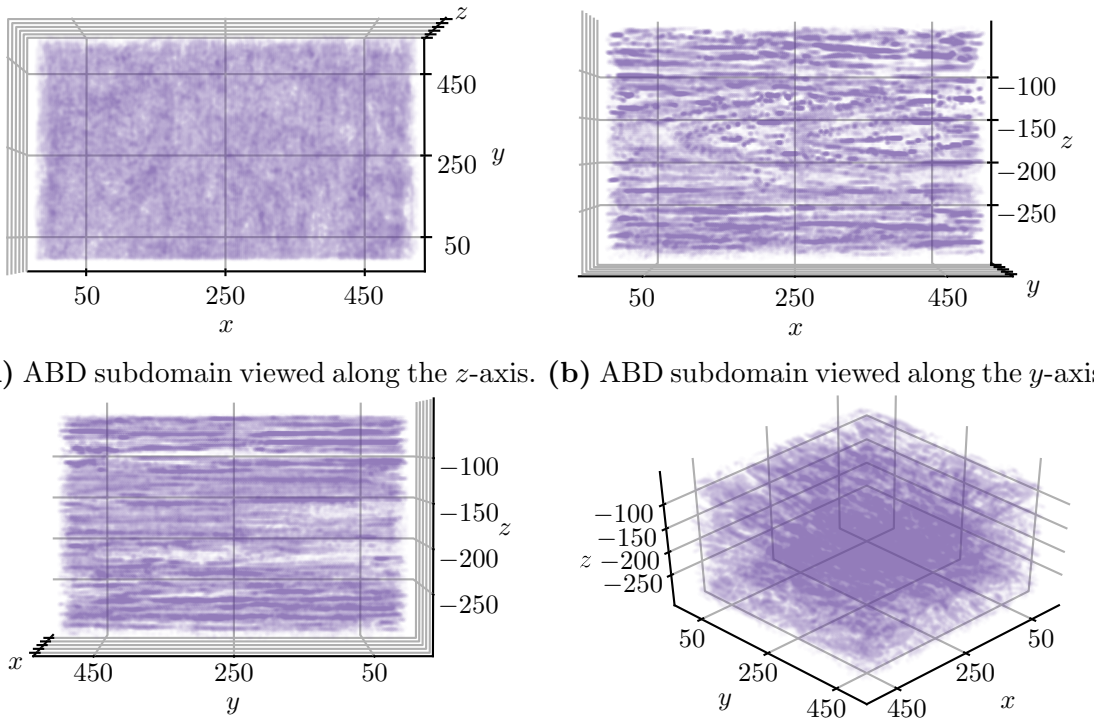


Figure 4.13: Overview of the outer Førde fjord area. Note the white square frame specifying the investigated domain within the larger gridded model domain, indicated by current vectors and dynamic coloring. Here, darker colors signify larger depths. The surrounding area is shaded in light blue and green to indicate water bodies and land area, respectively.

Consisting of a rectangular velocity field grid, the Førde fjord data set has a constant nearest neighbor grid point separation within the horizontal plane of 53.3 m. Vertical resolution is however variable, with grid points separated by 5-25 m from 50 m depth to 300 m. Using the entire fjord velocity data set, a $200 \times 200 \times 100$ regular tracer grid, covering the domain U, was advected over a 12-hour period, using the velocity field interpolation described in section 3.1.2. Again computing continuous Cauchy-Green eigenvalue and eigenvector fields, from gridded data (see sections sections 2.2.4, 3.2.1, and 3.2.2), 1631 manifold

initial positions were identified from 209945 candidate positions using $n_f = 5$. This ABD subdomain is presented in figure 4.14.



(a) ABD subdomain viewed along the z-axis. **(b)** ABD subdomain viewed along the y-axis.

(c) ABD subdomain viewed along the x-axis. **(d)** ABD subdomain isometric view.

Figure 4.14: Fjord current ABD subdomain from various viewing angles, $z = 0$ corresponding to the fjord surface. The ABD subdomain is here represented as a partly transparent scatter plot of the constituent tracer initial position grid points. That is, degree of coloring indicates ABD subdomain concentration along the respective viewing angles. Note the clearly distinct horizontal layers, particularly visible in (b) and (c).

Having computed the corresponding manifolds with the parameters summarized in table 4.2, LCSs were extracted using $\Gamma_{\text{ABD}} = 1.2$, $A_{\min} = 20000$ m^2, and $\epsilon = 0.1$ m. Note that this ϵ corresponds to approximately 1/25 of the tracer initial position grid spacing. The resulting LCSs are displayed in figure 4.15, where LCS surface segments have been shaded according to relative average normal repulsion (measured by $\bar{\lambda}_3$). The more strongly repelling LCS segment surfaces are colored bright yellow while less repelling ones are indicated by darker colors. Specifically, coloring was determined according to

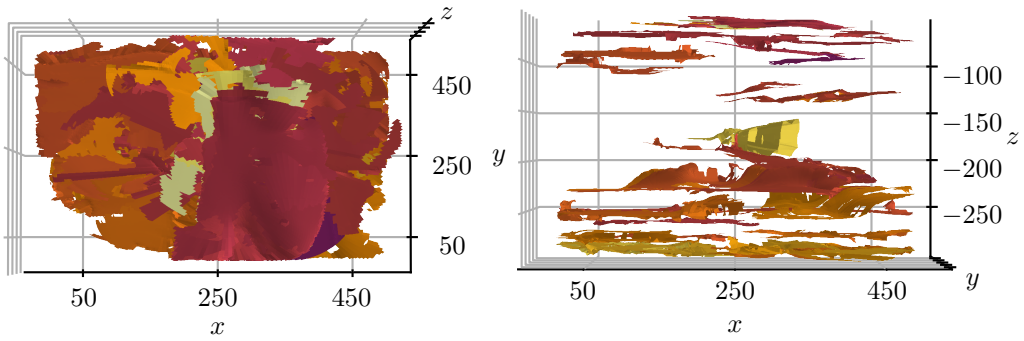
$$\gamma = \frac{\log \bar{\lambda}_3}{\max(\log \bar{\lambda}_3)}, \tag{4.8}$$

where γ corresponds to the color scale displayed in figure 4.15e. Notice how the domain boundary LCS behavior is captured in figure 4.15. This boundary description was enabled by imposing periodic boundary conditions on the eigenvector interpolator (see section 3.4.9).

Note how the nearly horizontal layers observable in figures 4.14b and 4.14c are recovered in the LCS structure, visible in figures 4.15b and 4.15c. This indicates limited vertical mixing of horizontal water layers in the investigated fjord region. An LCS neighbor tracer advection test, similar to that of figure 4.9, for the most repelling fjord LCS is displayed in figure 4.16. The clear separation of opposite-side neighboring tracers seems to confirm the transport barrier characteristics of the intervening LCS surface.

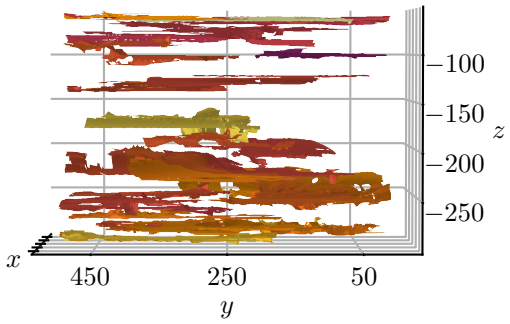
Table 4.2: Manifold computation input parameters applied to the Førde fjord test cases. Note that the units of derived input parameters correspond to those of their constituent base parameters.

Parameter	Value	Description
r_{init}	0.1	Radius of initial levelset (m)
Δ_1	4	Initial inter-levelset step length (m)
$\delta_{\mathcal{F}}$	2	Minimum nearest neighbor separation (m)
$\Delta_{\mathcal{F}}$	8	Maximum nearest neighbor separation (m)
Γ_Δ	0.005	Inter-levelset separation tolerance factor
$\alpha_{\min}$	5	Minimum axial angular offset (°)
$\alpha_{\max}$	25	Maximum axial angular offset (°)
$(\Delta\alpha)_{\min}$	$2\alpha_{\min}\delta_{\mathcal{F}}$	Lower axial angular offset parameter
$(\Delta\alpha)_{\max}$	$2\alpha_{\max}\delta_{\mathcal{F}}$	Upper axial angular offset parameter
$l_{\max}$	5	Maximum arc length to initial separation factor
$q_{\max}$	$5\delta_{\mathcal{F}}$	Maximal distance of continuous self-intersections
$r_{\max}$	20000	Maximum cumulative geodesic distance (m)
$\Gamma_\perp$	10^{-4}	Eigenvector field orthogonality tolerance
c_{arc}	0.7	Noise removal arc length tolerance

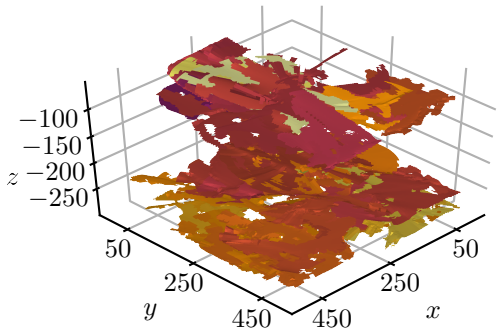


(a) LCSs viewed along the z-axis.

(b) LCSs viewed along the y-axis.



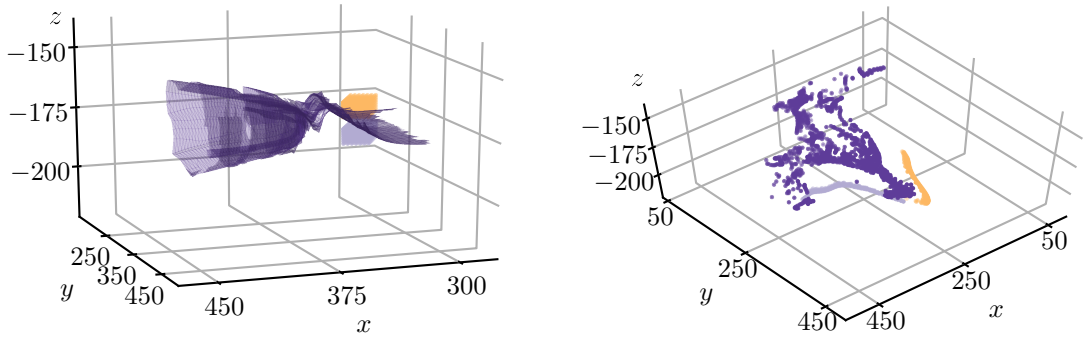
(c) LCSs viewed along the x-axis.



(d) LCS isometric view.

(e) LCS surface element repulsion color scale. LCS surface element color value is computed as the log of its average repulsion (see section 3.8) divided by the log of the largest observed average repulsion.

Figure 4.15: Fjord current repelling hyperbolic LCSs from various viewing angles, each surface element's repulsion rate indicated by its color. Note the agreement between the ABD subdomain displayed in 4.14 and the corresponding LCSs. Like in figure 4.14, $z = 0$ indicates the fjord surface.



(a) Sample LCS and selected tracers prior to advection (June 13 2013, 00:00).

(b) Sample LCS points and selected tracer points after advection (June 1 2013, 12:00).

Figure 4.16: LCS fragment along with selected tracer particles prior to and after advection over the selected time interval of 12 hours. Note that while the coherent LCS surface structure breaks down after advection, the selected tracer particles are clearly separated based on initial placement. Although the partial breakdown of the LCS surface structure is undesirable, it not unexpected. This is because the infinitesimal width of the target LCS structure makes mesh point placement error unavoidable. An accurate LCS identification algorithm is therefore still likely to place similar portions of the constituent mesh points on each side of the actual LCS, likely resulting in some stretching like we observe in (b).

Chapter 5

Discussion and Conclusions

A thorough discussion is warranted in order to properly contextualize and scrutinize the various choices, experiences and results that have been utilized or obtained throughout this study. Most prominently, this pertains to the set of assumptions and choices that were made in absence of an adequate preexisting method for LCS identification in three dimensions. Furthermore, the results presented in chapter 4 warrant some consideration. Finally, exploring the successes and shortcomings of this exercise yields suggestions for further work in the field.

5.1 Computing Cauchy-Green eigenvalue and eigenvector fields

As outlined in section 3.1, tracer advection and calculation of the flow map and flow map Jacobian was performed by use of the variational equations (2.35) and (2.36). This approach, suggested by Oettinger and Haller (2016), was chosen due to its superior accuracy. While some complexity is added by solving a set of twelve equations as opposed to three, we avoid having to compute auxiliary grids (Farazmand and Haller, 2012a) and introducing the accompanying error from finite differencing. It should however be noted that this method requires bounded first order velocity field partial derivatives with respect to the spatial coordinates (see equation (2.37)). This requires us to use differentiable analytical test cases and higher order interpolation methods for gridded data models. Cases in which this is impossible or impractical should therefore be handled by use of auxiliary grids and finite differencing, as outlined by Farazmand and Haller (2012a).

Also pertaining to velocity field interpolation is the use of cubic quadrivariate interpolation. Requiring us to keep the entire velocity field data set in working memory while the interpolation is being used, this approach may be prohibitively expensive for some applica-

tions. Consider for example a data set of 100 time steps, over a $500 \times 500 \times 100$ grid of three component double precision velocity vectors. Using quadrivariate interpolation for this data set requires us to keep in excess of 60 GB in memory while advecting tracer particles, which is impractical. This issue may be resolved by dividing the time interval into several segments, each computed with separate interpolations that are passed from memory whenever we leave the corresponding time segment. Note that the number of time steps included in each segment must allow the desired order of time dimension interpolation. In the case of cubic interpolation, this corresponds to four time steps or more.

Alternatively, we may drop the quadrivariate interpolation in favor of trivariate interpolation in the three spatial dimensions. Using a Runge-Kutta method without intermediate step slope approximations, such as the trapezoidal rule (Hairer et al., 2008), allows us to solve the system of (2.35) and (2.36) without time interpolation. It should however be noted that this approach prohibits us from detaching our solution steps from the time step of the data set. Moreover, using a lower order integrator such as the second order trapezoidal rule carries significant disadvantages, as such methods are inferior to higher order adaptive step methods in terms of efficiency.

It should be noted that tracer initial position grid density could be critical in terms of identifying some LCSs. Specifically, the spherical test case LCS described in section 4.2 was indiscernible when using a lower tracer initial position grid density. This is reasonable, as no manifold initial positions were placed sufficiently close to the sphere of heightened repulsion with these configurations. It is hard to tell *a priori* what tracer density is sufficient for any given application. We therefore recommend to run the highest tracer initial position grid density that is practical on the available hardware setup.

5.2 Adapting the method of geodesic levelsets

Managing the added degree of freedom gained when defining manifolds according to equation (2.51), compared to manifolds defined by ODEs of the form $\mathbf{x}' = \mathbf{f}(\mathbf{x})$, the method of geodesic levelsets described by Krauskopf and Osinga (2003) was modified by forcing radial trajectories. As previously discussed in section 3.5, this approach was preferred over that of guided trajectories due to superior speed, clarity, and accuracy. The approach of radially forced trajectories with inherited target half-planes $\mathcal{F}_r$ also permits a departure from the structuring of points into topological circles. Although useful in terms of managing mesh grid density, as well as guiding our triangulation algorithm, this hierarchical structure carries major disadvantages.

As highlighted by the fjord current LCS case (see section 4.5), the organization of points into levelsets, approximating topological circles, requires us to use periodic eigenvector field

boundary conditions in order to consistently determine the boundary behavior of the LCS system. In section 3.4.9, we noted that this treatment may introduce false positive LCS fragments if mesh point strands return to U after initially having moved beyond the boundaries of the domain of interest.

This issue can be managed by use of an extended initial position tracer grid domain. That is, by defining eigenvalue and eigenvector fields extending outside U, boundary behavior may be more adequately explored. However, this approach is computationally demanding and in some cases unworkable. An alternative, and much less costly, solution involves partly departing from the convention of adding manifold points only as complete levelsets representing topological circles. This may be done by organizing points into strands of descendants, each associated with a certain tangent line on the initial circular levelset curve C_1. This allows us to continue adding points even after parts of a levelset has left the region of interest U. Moreover, this partly bypasses the problem of point search trajectories potentially never reaching an acceptable new point. The strand in question would stop, but all other active strands could continue irrespective of this local error.

Organizing points into descendant strands instead of levelsets forming topological circles raises the question of how to manage grid density. This is because the topological circles $\{C_i\}$ are useful for inserting ghost ancestor points wherever inserting an additional point is necessary. We could handle this by simply computing a new strand using an intervening target half-plane $\mathcal{F}_r$. Discarding all unnecessary points, we would then be left with the required point. Note that in order to maintain a similar point structure conducive to proper mesh triangulation, the constant inter-levelset — or in this case inter-strand point — step length should be kept equal. That is, strand step n in strand i, should be of the same length as strand step length n in strand $j \neq i$. In this way, the same circular structure of the method of geodesic levelsets is maintained unless a domain boundary or local error intervenes.

Although the same curvature-guided step management could be maintained in this approach, it is not necessarily desirable. While seemingly useful in terms of adapting mesh density to the local neighborhood of the target manifold, this approach was found to add very little in terms of value to the overall method. Specifically, the curvature-guided step management method was found to exclusively decrease inter-levelset step length. In practice, the manifold generator would continue using the default step length until encountering any rough neighborhood. It would then immediately decrease step length to the minimum allowed amount, never to be increased. This is unsurprising, as a rough neighborhood intersecting anywhere with the current levelset would prohibit us from decreasing the inter-levelset step length. As geodesic distance — and consequently topological circle C_i arc length — is increased, the likelihood of encountering at least one such neighborhood becomes very large. Given that point placement accuracy is disconnected from inter-levelset step length, the value

of this step guidance method is limited to managing the interpolation error associated with mesh triangulation. While this may be critical for some applications, it seems reasonable that an appropriately chosen constant inter-levelset step length may be sufficient.

Departing from the choices of Krauskopf and Osinga (2003), we decided to refrain from recomputing the half-plane defining tangent vector $\mathbf{T}$ for every added point. That is, where Krauskopf and Osinga (2003) computed $\mathbf{T}$ as the vector spanning between the ancestor point nearest neighbors, we elected to compute $\mathbf{T}$ in the initial levelset, passing it along the corresponding mesh point strand. This modification was prompted as the recomputed tangent vectors of Krauskopf and Osinga (2003) were found to cause mesh structure inconsistencies. Specifically, mesh point strands were found to occasionally intersect. Moreover, this approach to defining $\mathbf{T}$ causes it to be highly sensitive to the small-scale behavior of C near the considered ancestor point. In some cases, the tangent vector of Krauskopf and Osinga (2003) was found to be poor in terms of representing the larger scale structure of the topological circle. Resulting in irregular nearest neighbor separations and even strand intersections, this behavior prompted us to redefine grid point placement. That is, points are placed along a family of planes defined by constant angles $\{\theta\}$ in the cylinder coordinate system centered on the manifold initial position $\mathbf{r}_0$ with, the z-axis along $\boldsymbol{\xi}_3(\mathbf{r}_0)$. Although not explored in this investigation, there are multiple alternatives to the method of geodesic levelsets that could be investigated with regard to LCS theory application. Several such approaches are summarized by Krauskopf et al. (2005).

5.3 Determining repelling hyperbolic LCSs from manifolds in the Cauchy-Green eigenvector field

Unlike our approach to manifold identification, the construction of repelling hyperbolic LCSs from candidate manifolds was found to be heavily dependent on parameter choice. While the point comparison distance ϵ of condition D may be chosen based on tracer grid point separation, choosing ABD subdomain tolerance Γ_{ABD} and minimum surface area A_{min} is largely left up to personal judgment. This leaves room for the user to define these parameters to tailor the result with regard to the number and sizes of LCSs. Although possibly useful for application, this is unsatisfactory from a scientific point of view.

User choice of Γ_{ABD} and A_{min} should ideally fall into one of two approaches. The first of these, referred to as the cluster approach, uses small values for both Γ_{ABD} and A_{min} with the intent of treating clusters of overlapping LCS surface elements as single LCSs. While requiring the user to manually, or otherwise, sort these LCS surface elements by structure association, this approach is convincing in terms of low reliance on our tolerance parameters.

As opposed to the cluster approach, what may be termed the carve-out approach utilizes large values for Γ_{ABD} and A_{min} to extract LCSs from single manifolds. Specifically, we attempt to choose Γ_{ABD} and A_{min} such that we get a small number of large LCS surfaces. This approach yields smoother, more coherent LCSs by relying heavily on our tolerance parameters. The result is often a more visually appealing set of LCSs, albeit less convincing in terms of accuracy.

The results presented in chapter 4 were all computed using the cluster approach, most identified LCS structures consisting of several surface segments. This is particularly evident for the ABC flow data where each identified LCS structure consists of anywhere between 3 and 23 LCS surface segments. This approach was found preferable, not only due to its presumed superior accuracy, but also because the carve-out approach was found impractical. Most prominent for the ABC flow test cases, this was because most computed manifolds were too small to extract sufficiently large LCSs. That is, our manifold identification method was often unable to produce sufficiently large manifolds with the allotted time. This issue could be alleviated by implementing some of the aforementioned method alterations, or simply allocating more computation time. Based on practical considerations, manifold construction was limited to 1 hour of wall clock time, while allowing the process an extra hour to terminate. This margin of safety was introduced as no manifold would be saved in the case of failure to terminate within the reserved cluster wall time. This strategy should ideally be combined with an accompanying reduction of Runge-Kutta method error tolerance parameters, reducing error accumulation as to decrease the prevalence of procedure termination prompted by continuous self-intersections.

5.3.1 Implementing criterion of requiring LCSs to be locally most repelling

Although simple in mathematical terms, the LCS existence conditions suggested by Farazmand and Haller (2012a) are not entirely clear-cut in terms of implementation. This specifically pertains to condition 4 in (2.49), intended to establish the LCS as a local repulsion maximum. It could be argued that simply noting the presence of a local repulsion maximum is of limited value in terms of application. That is, by following Haller (2011)'s criteria, we might be selecting large numbers of inconsequential LCSs, cluttering a system interpretation intended to highlight only major system-defining structures. This despite requiring a minimum LCS surface area. We could view this as a weakness of the present approach, it being a rather close adaptation of Farazmand and Haller (2012a)'s criteria. This shortcoming could possibly be alleviated by either requiring a certain LCS repulsion rate, or by only selecting

a single LCS from any given LCS grouping. Note that the latter of these requirements may not be paired with the previously mentioned cluster approach to choosing Γ_{ABD} and A_{min}.

5.3.2 Alternative methods

There are several alternative approaches to LCS selection from manifold candidates, each with accompanying advantages and disadvantages, as compared to the present method. A very simple approach could be devised by defining LCS neighborhoods as rectangular cuboids within the domain of interest U. LCS selection could then be carried out by selecting the most repelling of the LCS candidates present in a single neighborhood. Note that as condition 4 in (2.49) is approximated by this act of comparison and selection, we substitute the ABD subdomain described in section 3.3 with an analogous AB subdomain. That is, we drop condition D (see equation (3.10)), performing mesh point selection with criteria A and B (see equations (3.8) and (3.9)). While simple in terms of implementation and cheap in terms of computational resources, this method is quite far removed from the theoretical foundation given by condition 4 in equation (2.49). This method does however provide a large amount of flexibility in terms of neighborhood selection, requiring the user to exercise personal judgment.

A second possible alternative can be construed by generalizing the approach of Farazmand and Haller (2012a) to three dimensions. This approach entails computing LCS candidate surfaces using the aforementioned AB subdomain. Intersections of the resulting LCS candidates with a selection of horizontal and vertical planes are then identified and used to define locally neighboring LCS candidates. The most repelling LCS candidate surface within each such plane intersection neighborhood is then selected as an LCS. This approach has the advantage of inherently choosing the most repelling among nearby LCS candidates. It should however be noted that two LCS candidates, identified as neighbors at a plane intersection, may rapidly diverge elsewhere in the domain. This is indeed also a weakness of the method as originally proposed in the two-dimensional case.

Note that while approximating condition 4 in equation (2.49) more closely than the method of neighborhood volumes, the likely divergence of identified neighbor surfaces may cause major discrepancies from the theoretical definition. More robust algorithms for identifying neighboring LCS candidate surfaces could be conceived by use of various clustering algorithms.

5.4 Results and prospects for application

Having applied our method for identification of repelling hyperbolic LCSs to various distinct cases, we are now able to draw some conclusions regarding these systems, as well as the

prospects for further use of the method itself. Specifically, the efficacy — or lack thereof — of our method in terms of unveiling useful system insights could guide us with regard to further development and utilization.

5.4.1 ABC flow system

As previously discussed in section 5.3, manifold selection in the ABC flow systems was done according to the cluster approach. In addition to allowing limited use of numerical tolerance, this approach is less dependent on large input manifolds. This latter advantage was found to be crucial in the case of the ABC flow system. That is, manifold development was frequently stunted by complicated eigenvector field regions. Specifically, for the steady and unsteady ABC flow cases, manifolds in the Cauchy-Green eigenvector fields were found to frequently display self-intersections, causing the algorithm to terminate. This behavior was likely caused by small numerical errors making one or more mesh points jump onto a neighboring manifold. In these chaotic regions, even closely neighboring manifolds would diverge significantly from the target manifold, possibly causing the unphysical self-intersections. These issues could possibly be mitigated, either by increasing the accuracy requirements of the selected adaptive step Runge-Kutta method, or eliminating our use of topological circle curve interpolations for inserting intervening mesh points (see section 5.2).

The small differences observed between the results of the steady and unsteady ABC flow systems are also interesting. This indicates that although considerable, the added perturbation in the unsteady flow field did not significantly alter the principal system characteristics. Although certainly requiring a more rigorous investigative approach, this result is suggestive of the robustness thought to be characteristic to LCSs. That is, the small differences observed in terms of ABD subdomains and resulting repelling hyperbolic LCSs caused by rather large changes to the underlying flow system, are indicative of LCSs in fact being largely independent of the idiosyncrasies of small scale flow patterns.

Forming large and largely coherent particle transport boundaries, the LCSs detected in the steady and unsteady ABC flow systems exhibit many desirable attributes with regard to possible application. Specifically, the formation of clear boundaries, effectively separating the investigated domain into sections, is exactly the kind of useful insight sought by application of LCS theory. It should however be noted that the observed breakdown of the LCS surface is problematic in terms of some applications. While computing initial position LCSs is sufficient in terms of investigating large scale tracer mixing patterns, we gain little insight with regard to the actual transport barriers. That is, unless we are able to accurately transport our LCS surfaces in time, we are unable to track the transport barriers deemed so desirable in terms of application. Techniques for increasing the accuracy of LCS transport in the two-dimensional case are described by Farazmand and Haller (2012a).

5.4.2 Førde fjord system

Less difficult in terms of manifold development, the Førde fjord data set was successfully analyzed with comparatively small ABD subdomain tolerance Γ_{ABD}. The resulting, largely horizontal, LCSs are interesting in that they suggest minimal vertical mixing across horizontal water layers. If possible to reproduce for larger parts of the fjord area, this is a useful insight with regard to the present issue of using the Førde fjord as a deposit for mine tailings.

It could be speculated that this behavior is due to sudden steps in terms of water layer density caused by sharp changes in either salinity or temperature. The exact causes for the observed LCSs is however beyond the scope of this investigation.

5.4.3 Prospects for application

Based on our test cases, it seems plausible that the present method — either after minor modifications, or in its current form — could be useful in terms of application to real world flow systems. This view is substantiated both from the seemingly valuable insights gained from application to the Førde fjord data set, as well as the seeming robustness of our LCSs to flow field perturbations. The latter argument is particularly interesting as it suggests imperfect gridded data models could be sufficient in order to accurately reproduce real world LCSs.

The most significant detriment to further application of our method of three-dimensional LCS identification is likely its complexity. As computing LCSs in three dimensions is significantly more costly — both with respect to development and resources — than its two-dimensional counterpart, strong arguments in terms of result accuracy and insights are needed to justify a change in practices. It is intuitively clear that the argument for using three-dimensional LCS identification is stronger for systems where all three dimensions are of similar importance to overall system characteristics. Fjord flow systems, such as the one exhibited in section 4.5, may be seen as examples of such systems, their depth often being of similar magnitude to their width and their vertical transport significantly altering system dynamics. Conversely, the surface layer of the oceanic within which buoyant materials, such as some types of plastic, are often contained, is an example of a nearly two-dimensional system.

Supplementing considerations of system dimensions, comparative studies of two- and three-dimensional LCS identification could be used to guide approach selection. This could be done numerically, or in the field by deploying tracers in well-knowns flow systems. Such studies could be very useful in terms of determining the applicability of three-dimensional LCS identification as a part of a larger toolbox consisting of existing and coming methods.

5.5 Recommendations for further work

As described by Oettinger and Haller (2016), identification of LCSs in three-dimensional systems has traditionally been done by identifying two-dimensional LCS layers, combining them to surfaces by use of interpolation. This approach, outlined by Blazevski and Haller (2014), is clearly problematic as it completely ignores all 3rd-dimension system dynamics. Moreover, the method proposed by Oettinger and Haller (2016), while flexible, does not yield coherent surfaces conducive to further analysis. However, as suggested by Oettinger and Haller (2016), combining LCS theory with dedicated manifold identification methods yields a more practical approach to identification of three-dimensional hyperbolic LCSs.

Further inquiries as to LCS identification in three-dimensional systems could focus on different methods for reconstructing manifolds. Several such dedicated methods, including the method of geodesic levelsets utilized in this investigation, are outlined by Krauskopf et al. (2005). An investigation of the efficacy of these various methods with regard to LCS identification in three-dimensional systems could be of great value with respect to application. This also pertains to the various LCS extraction approaches outlined in section 5.3.

More specific to the current approach, exploring the previously described method alterations could potentially yield improvements in terms of comprehensibility, speed, and accuracy. Particularly, departing from the strict ordering of mesh points into levelsets could allow better boundary treatment, as well as general improvements in terms of accuracy (see section 5.2). Sensitivity analysis with regard to ODE solver error tolerances, as well as interpolation order and tracer grid density, could also prove valuable in terms of identifying sources of error.

Furthermore, as many flow systems may reasonably be approximated as two-dimensional, comparative studies of two-dimensional and three-dimensional approaches to LCS identification could be very useful as to determine proper application of each tool. That is, for what applications are the comparatively simpler two-dimensional approaches sufficient to capture the most critical system dynamics.

Finally, combining available gridded data models with physical tracers deployed in real world flow systems could allow us to verify the robustness of our LCS identification tools to the inaccuracies of these data models. One such pilot experiment, known as the *NSF-ALPHA Sea Experiment 2017* was conducted by Filippi et al. (2018). A similar approach was applied to the Scott Reef atoll in Western Australia by Ross et al. (2018). Further experiments are planned by the *NSF-ALPHA* project in the near future.

Appendices

Appendix A

Existence criteria for repelling hyperbolic LCSs in three dimensions

According to Farazmand and Haller (2012b), a compact material surface $\mathcal{M}(t) \subset U$ is a repelling LCS over the time interval $[t_0, t]$ if and only if the following conditions hold:

$$\begin{aligned} &1. \quad \lambda_{n-1}(\mathbf{x}_0) \neq \lambda_n(\mathbf{x}_0) > 1, \\ &2. \quad \text{The matrix } \mathbf{L}(\mathbf{x}_0) \text{ is positive definite for all } \mathbf{x}_0 \in \mathcal{M}(\mathrm{t}_0), \\ &3. \quad \boldsymbol{\xi}_n(\mathbf{x}_0) \perp \mathbf{T}_{\mathbf{x}_0}\mathcal{M}(t_0), \\ &4. \quad \langle \nabla\lambda_n(\mathbf{x}_0), \boldsymbol{\xi}_n(\mathbf{x}_0)\rangle = 0. \end{aligned} \tag{A.1}$$

Here, n is the number of dimensions of the system. In three dimensions, the matrix $\mathbf{L}(\mathbf{x}_0)$ takes the form

$$\mathbf{L}(\mathbf{x}_0) = \begin{bmatrix} \nabla^2\mathbf{C}^{-1}[\boldsymbol{\xi}_3,\boldsymbol{\xi}_3,\boldsymbol{\xi}_3,\boldsymbol{\xi}_3] & 2\frac{\lambda_3-\lambda_1}{\lambda_1\lambda_3}\langle\boldsymbol{\xi}_1, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\rangle & 2\frac{\lambda_3-\lambda_2}{\lambda_2\lambda_3}\langle\boldsymbol{\xi}_2, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\rangle \\ 2\frac{\lambda_3-\lambda_1}{\lambda_1\lambda_3}\langle\boldsymbol{\xi}_1, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\rangle & 2\frac{\lambda_3-\lambda_1}{\lambda_1\lambda_3} & 0 \\ 2\frac{\lambda_3-\lambda_2}{\lambda_2\lambda_3}\langle\boldsymbol{\xi}_2, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\rangle & 0 & 2\frac{\lambda_3-\lambda_2}{\lambda_2\lambda_3} \end{bmatrix}. \tag{A.2}$$

Note that we have dropped dependence on t_0 and t for notational simplicity and adhere to Haller (2011)'s convention of ∇^2 denoting the Hessian. In order to simplify condition 2, we need the following relation given by Haller (2011) (for $n = 3$)

$$\nabla^2\mathbf{C}^{-1}[\boldsymbol{\xi}_3,\boldsymbol{\xi}_3,\boldsymbol{\xi}_3,\boldsymbol{\xi}_3] = -\frac{1}{\lambda_3^2}\langle\boldsymbol{\xi}_3, \nabla^2\lambda_3\boldsymbol{\xi}_3\rangle + 2\sum_{q=1}^{2}\frac{\lambda_3-\lambda_q}{\lambda_3\lambda_q}\langle\boldsymbol{\xi}_q, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\rangle^2. \tag{A.3}$$

Now, according to Sylvester's theorem (Gilbert, 1991), the matrix $\mathbf{L}(\mathbf{x}_0)$ is positive definite if and only if all its leading principal minors are positive. For the three-dimensional case ($n = 3$), this corresponds to the following requirements

$$\nabla^2\mathbf{C}^{-1}\left[\boldsymbol{\xi}_3, \boldsymbol{\xi}_3, \boldsymbol{\xi}_3, \boldsymbol{\xi}_3\right] > 0, \tag{A.4}$$

$$2\nabla^2\mathbf{C}^{-1}\left[\boldsymbol{\xi}_3, \boldsymbol{\xi}_3, \boldsymbol{\xi}_3, \boldsymbol{\xi}_3\right]\frac{\lambda_3 - \lambda_1}{\lambda_1\lambda_3} - 4\frac{(\lambda_3 - \lambda_1)^2}{(\lambda_1\lambda_3)^2}\left\langle \boldsymbol{\xi}_1, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\right\rangle^2 > 0, \tag{A.5}$$

and

$$\begin{aligned} &4\nabla^2\mathbf{C}^{-1}\left[\boldsymbol{\xi}_3, \boldsymbol{\xi}_3, \boldsymbol{\xi}_3, \boldsymbol{\xi}_3\right]\frac{(\lambda_3 - \lambda_1)(\lambda_3 - \lambda_2)}{\lambda_1\lambda_2\lambda_3^2} \\ &- 8\frac{(\lambda_3 - \lambda_1)^2(\lambda_3 - \lambda_2)}{\lambda_1^2\lambda_2\lambda_3^3}\left\langle \boldsymbol{\xi}_1, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\right\rangle^2 \\ &- 8\frac{(\lambda_3 - \lambda_1)(\lambda_3 - \lambda_2)^2}{\lambda_1\lambda_2^2\lambda_3^3}\left\langle \boldsymbol{\xi}_2, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\right\rangle^2 > 0 \end{aligned}. \tag{A.6}$$

Inserting equation (A.3) into equations (A.4), (A.5), and (A.6) and performing basic algebraic manipulations yields the simplified conditions

$$\frac{\lambda_3 - \lambda_1}{\lambda_1}\left\langle \boldsymbol{\xi}_1, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\right\rangle^2 + \frac{\lambda_3 - \lambda_2}{\lambda_2}\left\langle \boldsymbol{\xi}_2, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\right\rangle^2 - \frac{1}{2\lambda_3}\left\langle \boldsymbol{\xi}_3, \nabla^2\lambda_3\boldsymbol{\xi}_3\right\rangle > 0, \tag{A.7}$$

$$\frac{\lambda_3 - \lambda_2}{\lambda_2}\left\langle \boldsymbol{\xi}_2, \nabla\boldsymbol{\xi}_3\boldsymbol{\xi}_3\right\rangle^2 - \frac{1}{2\lambda_3}\left\langle \boldsymbol{\xi}_3, \nabla^2\lambda_3\boldsymbol{\xi}_3\right\rangle > 0, \tag{A.8}$$

and

$$\left\langle \boldsymbol{\xi}_3, \nabla^2\lambda_3\boldsymbol{\xi}_3\right\rangle < 0. \tag{A.9}$$

Now, note that $\lambda_3 \geq \lambda_2 \geq \lambda_1 > 0$ such that all the scalar prefactors in equations (A.7)-(A.9) are nonnegative. Moreover, the square of the real inner products must be nonnegative. Therefore, whenever condition (A.9) is satisfied, so are equation (A.7) and (A.8). We are therefore left with equation (A.9) as the simplified condition 2 in (A.1).

Bibliography

Berg, M. d., Cheong, O., Kreveld, M. v., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition.

Bieberbach, L. (1951). On the remainder of the Runge-Kutta formula in the theory of ordinary differential equations. *Zeitschrift fur angewandte Mathematik und Physik ZAMP*, 2(4):233–248.

Blazevski, D. and Haller, G. (2014). Hyperbolic and elliptic transport barriers in three-dimensional unsteady flows. *Physica D: Nonlinear Phenomena*, 273-274:46 – 62.

Dormand, J. and Prince, P. (1980). A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19 – 26.

Farazmand, M. and Haller, G. (2012a). Computing Lagrangian coherent structures from their variational theory. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(1):013128.

Farazmand, M. and Haller, G. (2012b). Erratum and addendum to "A variational theory of hyperbolic Lagrangian coherent structures" [Physica D 240 (2011) 574–598]. *Physica D: Nonlinear Phenomena*, 241(4):439 – 441.

Filippi, M., Rayson, M., Hadjighasem, A., Ivey, G. N., Lowe, R., Gilmour, J., and Peacock, T. (2018 (accessed May 28, 2018)). The detection of Lagrangian transport structures in a coral reef atoll. https://agu.confex.com/agu/os18/meetingapp.cgi/Paper/317787.

Gilbert, G. T. (1991). Positive definite matrices and Sylvester's criterion. *The American Mathematical Monthly*, 98(1):44–46.

Hairer, E., Nørsett, S. P., and Wanner, G. (2008). *Solving Ordinary Differential Equations 1*. Springer, Berlin, second revised edition.

Haller, G. (2011). A variational theory of hyperbolic Lagrangian coherent structures. *Physica D: Nonlinear Phenomena*, 240(7):574 – 598.

Haller, G. (2015). Lagrangian coherent structures. *Annual Review of Fluid Mechanics*, 47(1):137–162.

Haller, G. and Yuan, G. (2000). Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3):352 – 370.

Krauskopf, B., Osinga, H., Doedel, E., Henderson, M., Guckenheimer, J., Vladimirsky, A., Dellnitz, M., and Junge, O. (2005). A survey of method's for computing (un)stable manifold of vector fields. *International Journal of Bifurcation and Chaos*, 15:763 – 791.

Krauskopf, B. and Osinga, H. M. (2003). Computing geodesic level sets on global (un)stable manifolds of vector fields. *SIAM Journal on Applied Dynamical Systems*, 2(4):546–569.

Miron, P., Vétel, J., Garon, A., Delfour, M., and Hassan, M. E. (2012). Anisotropic mesh adaptation on Lagrangian coherent structures. *Journal of Computational Physics*, 231(19):6419 – 6437.

Möller, T. and Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28.

Oettinger, D. and Haller, G. (2016). An autonomous dynamical system captures all LCSs in three-dimensional unsteady flows. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 26(10):103111.

Oliphant, T. E. (2015). *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition.

Onu, K., Huhn, F., and Haller, G. (2015). LCS Tool: A computational platform for Lagrangian coherent structures. *Journal of Computational Science*, 7(Supplement C):26 – 36.

Prince, P. and Dormand, J. (1981). High order embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 7(1):67 – 75.

Ross, S. D., Rypina, I., Shadden, S., and Peacock, T. (2018 (accessed May 28, 2018)). Targeted drifter deployments around Martha's Vineyard to uncover Lagrangian transport structures. https://doi.org/10.5281/zenodo.1215290.

Shadden, S. C., Lekien, F., and Marsden, J. E. (2005). Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3):271 – 304.

Stoer, J. and Bulirsch, R. (1996). *Introduction to Numerical Analysis*. Springer, Berlin, third edition.

Trefethen, L. N. and Bau, D. (1997). *Numerical Linear Algebra*. SIAM, Philadelphia, PA, first edition.

Truesdell, C. and Noll, W. (2004). *The Non-Linear Field Theories of Mechanics*. Springer, Berlin, third edition.

van Hinsberg, M. A. T., Boonkkamp, J. H. M. t. T., Toschi, F., and Clercx, H. J. H. (2013). Optimal interpolation schemes for particle tracking in turbulence. *Phys. Rev. E*, 87:043307.

Watkins, D. S. (2005). Product eigenvalue problems. *SIAM Review*, 47(1):3–40.

Williams, J. (2018 (accessed May 11, 2018)). Bspline-Fortran: Multidimensional B-spline interpolation of data on a regular grid. https://doi.org/10.5281/zenodo.1215290.