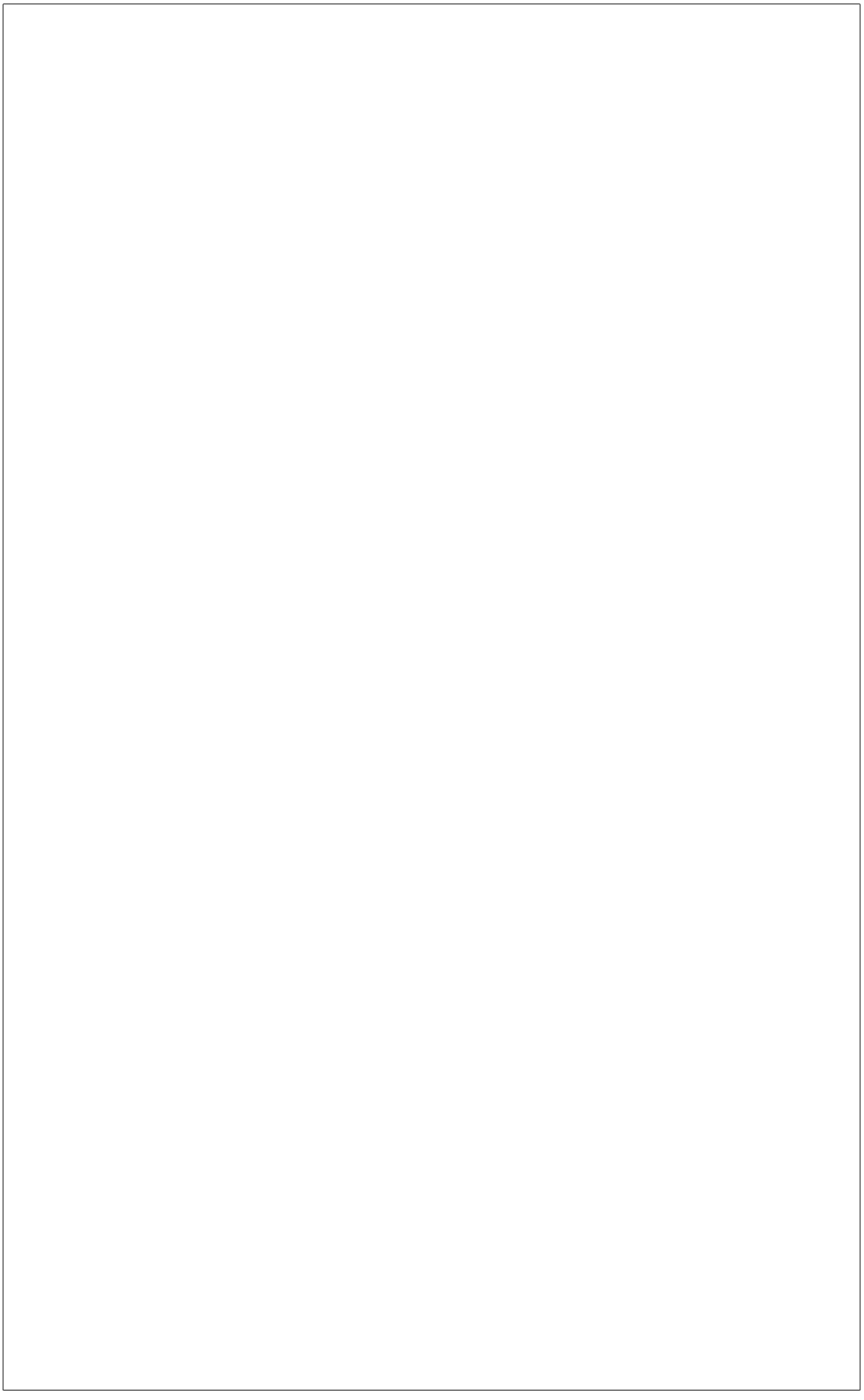
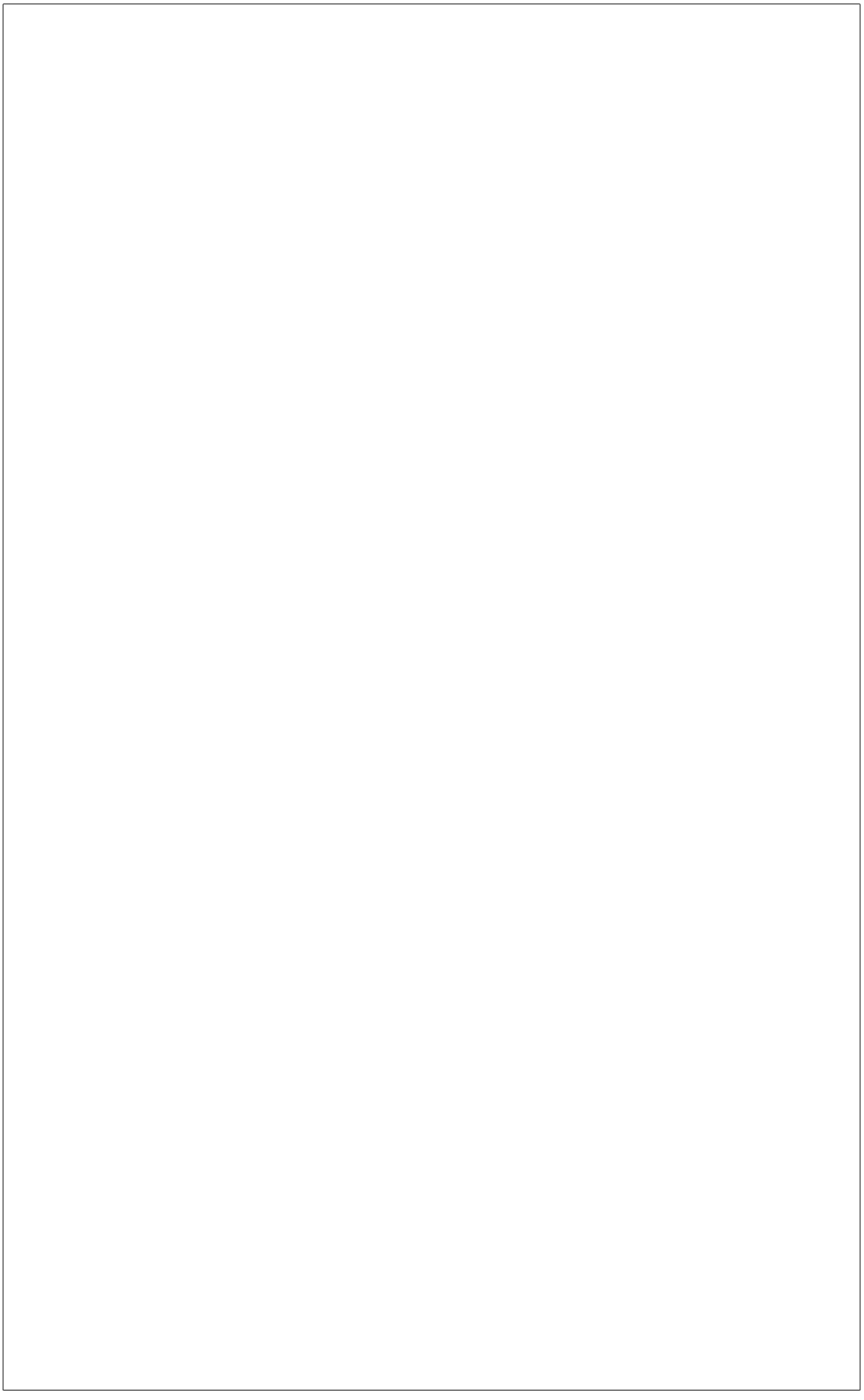
Abstract



Sammendrag



Preface

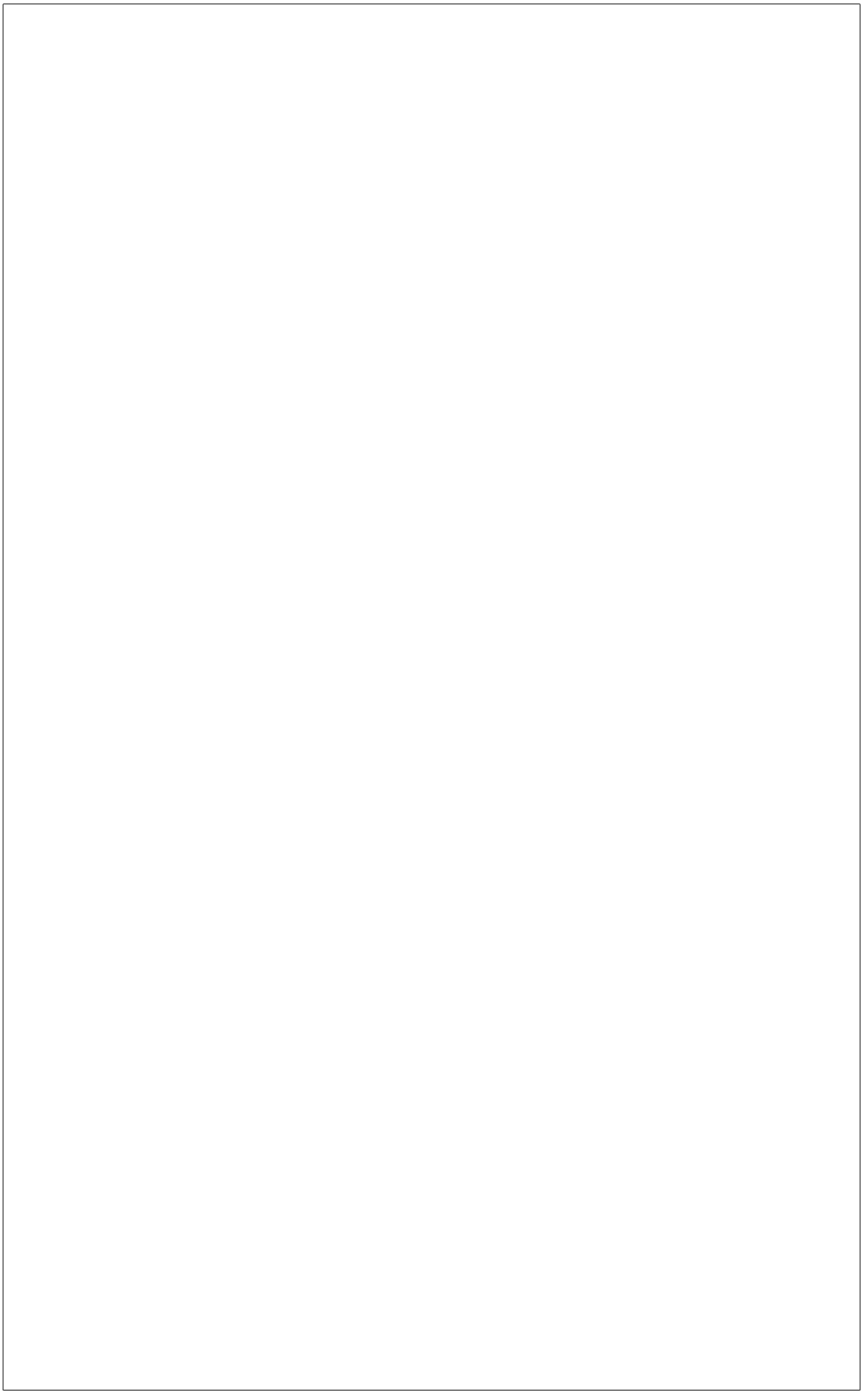
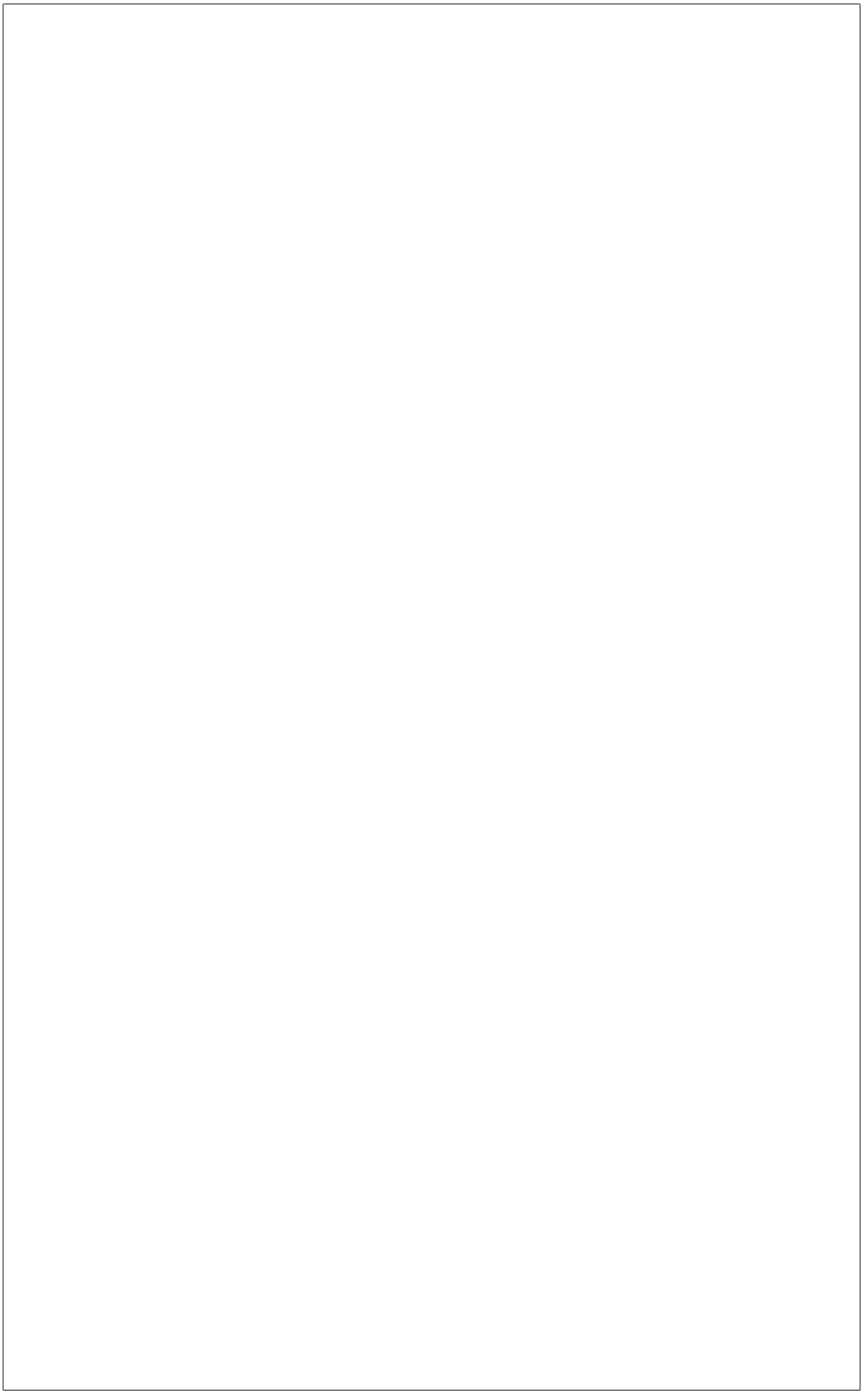
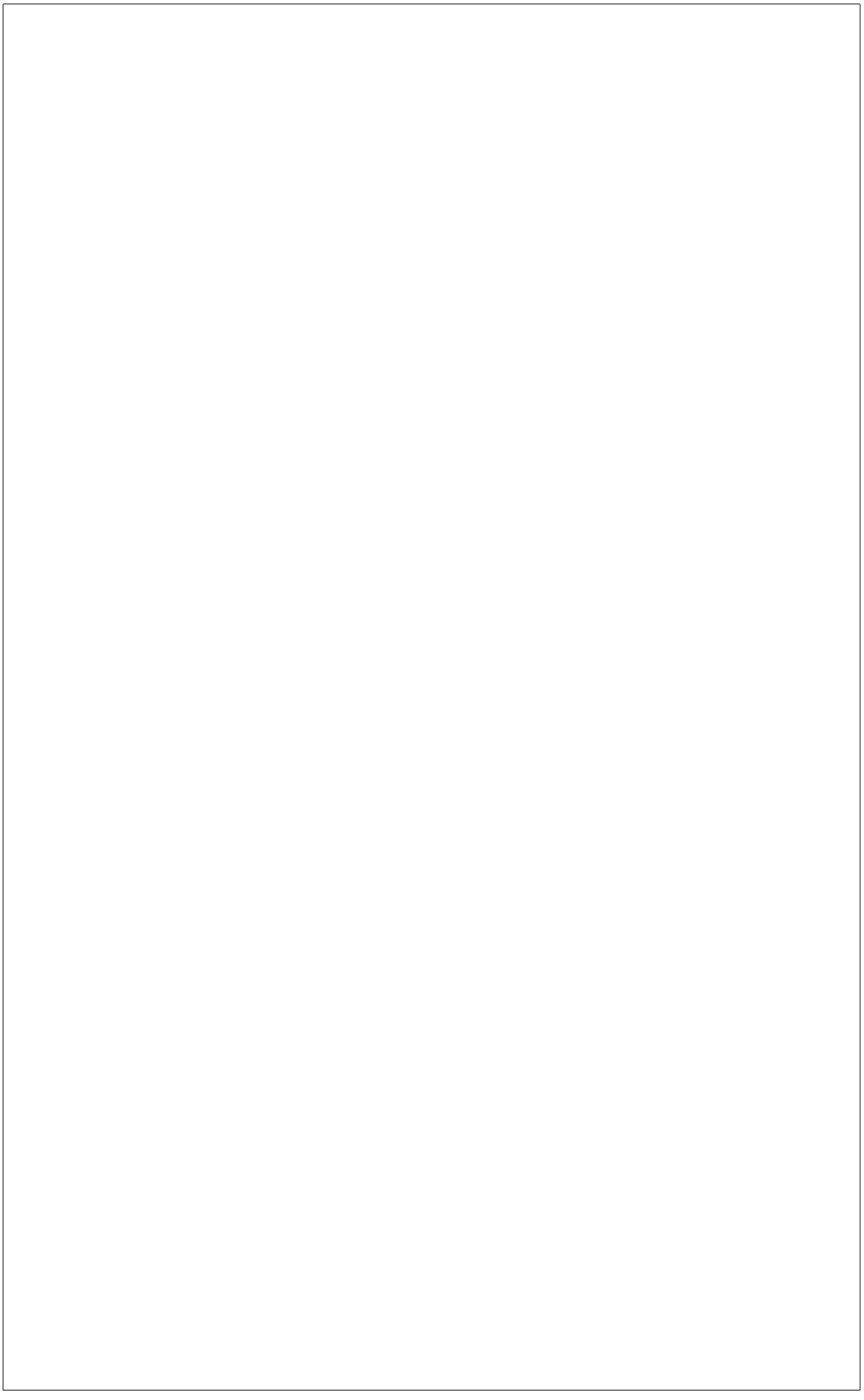


Table of Contents

List of Figures



List of Tables



Notation

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

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

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

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

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

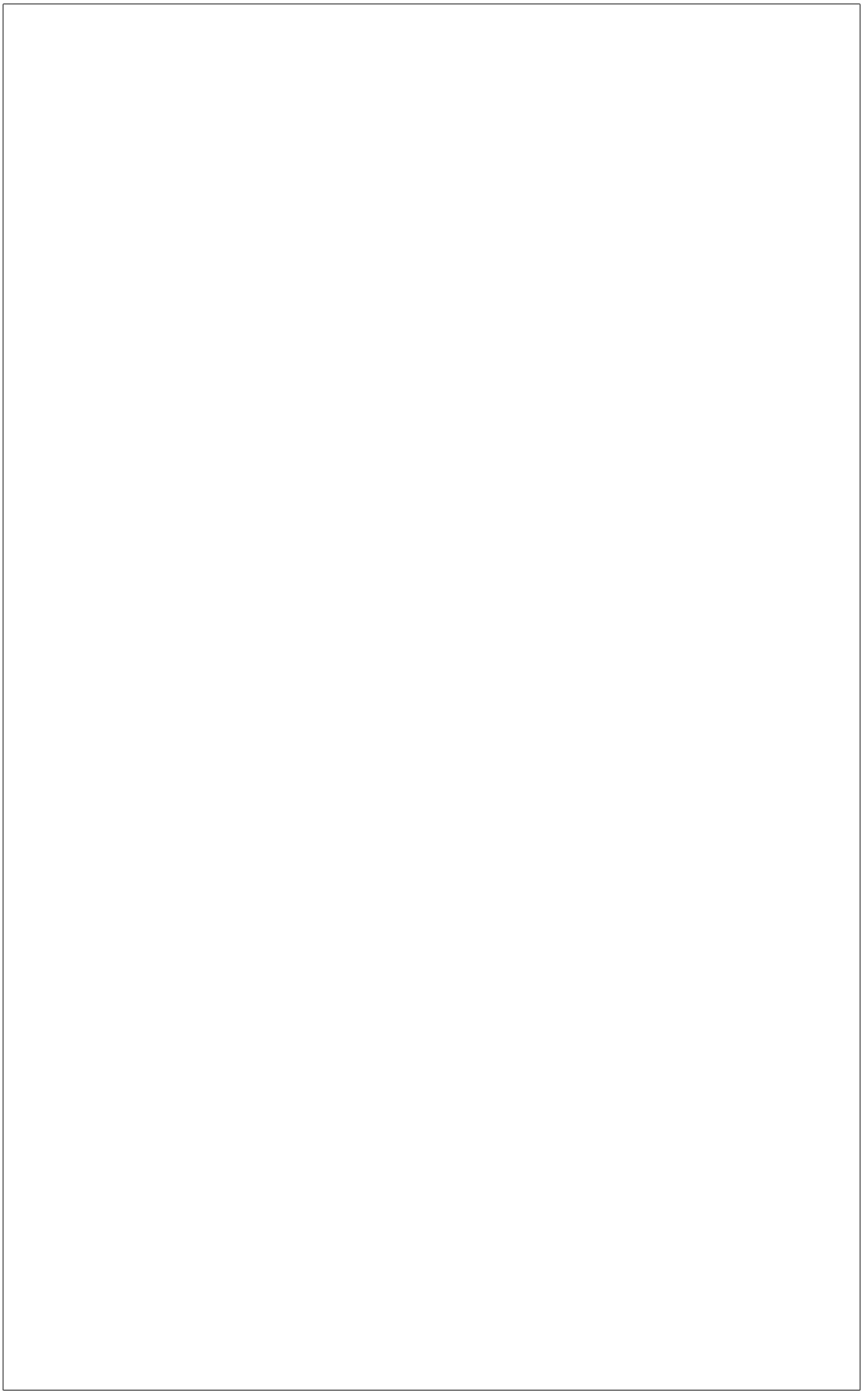
$$\langle \boldsymbol{\xi}, \boldsymbol{\psi} \rangle = \sum_{i=1}^{n} \xi_i \psi_i.$$

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

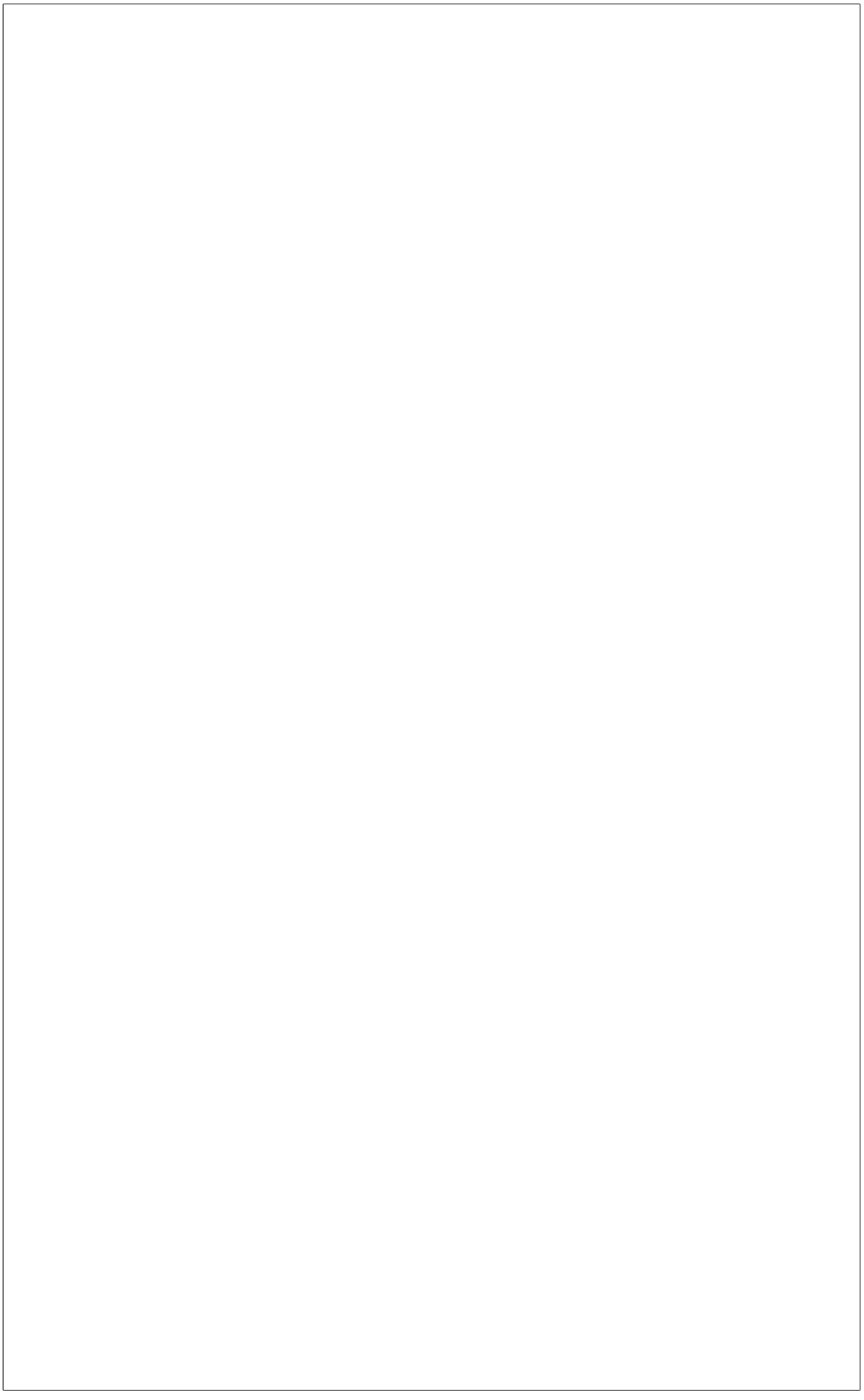
$$\|\boldsymbol{\xi}\| = \sqrt{\langle \boldsymbol{\xi}, \boldsymbol{\xi} \rangle}.$$

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

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



1 Introduction



2 Theory

2.1 Solving systems of ordinary differential equations

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

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

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

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

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

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

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

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

All numerical integration schemes fall into one of two categories; explicit and implicit methods. Explicit methods are characterized by computing the state of the system at a later time, based on the state of the system at the current time (in some cases, the state at earlier times are also considered). Implicit methods, however, involve the solution of an equation in which both the current and the later state of the system are involved. Thus, a generic, explicit method for computing the state of the system at time $t + h$, given its state at t, can be expressed as

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

while, for implicit methods, an equation of the sort

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

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

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

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

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

where $x(t)$ is the exact solution at time t. The accuracy of the approximation, however, depends on both the numerical integration method and the time step length h used for the temporal discretization. One way of obtaining approximations of the true solution *inbetween* the discrete time levels is by means of interpolation — a numerical technique which will be elaborated upon in section 2.1.2. For nonlinear systems, analytical solutions usually do not exist. Thus, such systems are often analyzed by means of numerical methods.

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

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

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

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

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

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

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

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

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

The *global* error

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

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

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

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

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

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

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

During the first half of the twentieth century, a substantial amount of research was conducted in order to develop numerically robust, high-order, explicit Runge-Kutta methods. The idea

Table 2.1: Butcher tableau representing a generic s-stage Runge-Kutta method.

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

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

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

For embedded methods, the coefficients are tuned such that

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

is of order p, and

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

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

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

Table 2.2: Butcher tableau representation a generic, embedded, explicit Runge-Kutta method.

0					
c_2	$a_{2,1}$				
c_3	$a_{3,1}$	$a_{3,2}$			
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
c_s	$a_{s,1}$	$a_{s,2}$	$\dots$	$a_{s,s-1}$	
	b_1	b_2	$\dots$	b_{s-1}	b_s
	$\widehat{b}_1$	$\widehat{b}_2$	$\dots$	$\widehat{b}_{s-1}$	$\widehat{b}_s$

of an eighth order solution with a seventh order interpolant — was chosen as the single, multipurpose, numerical ODE solver for this project.

Note that the concept of *order* is less well-defined for embedded methods than for fixed stepsize methods, as a direct consequence of the adaptive time step. Although the *local* errors of each integration step scale as per equation (2.7), the bound on the *global* (i.e., observable) error suggested in equation (2.10) is invalid, as the time step is, in principle, different for each integration step. Butcher tableau representations of the classical 4^{th}-order Runge-Kutta method and the embedded Dormand-Prince 8(7) method are available in is available in tables 2.3 and 2.4; where the latter of which has been typeset in landscape mode for the reader's convenience. Details on how the dynamic time step of the Dormand-Prince 8(7) method was implemented will be presented in section 3.3.2.

Table 2.3: Butcher tableau representation of the explicit, classical, 4^{th}-order Runge-Kutta method.

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

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

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

2.1.2 *Spline interpolation of discrete data*

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

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

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

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

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

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

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

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

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

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

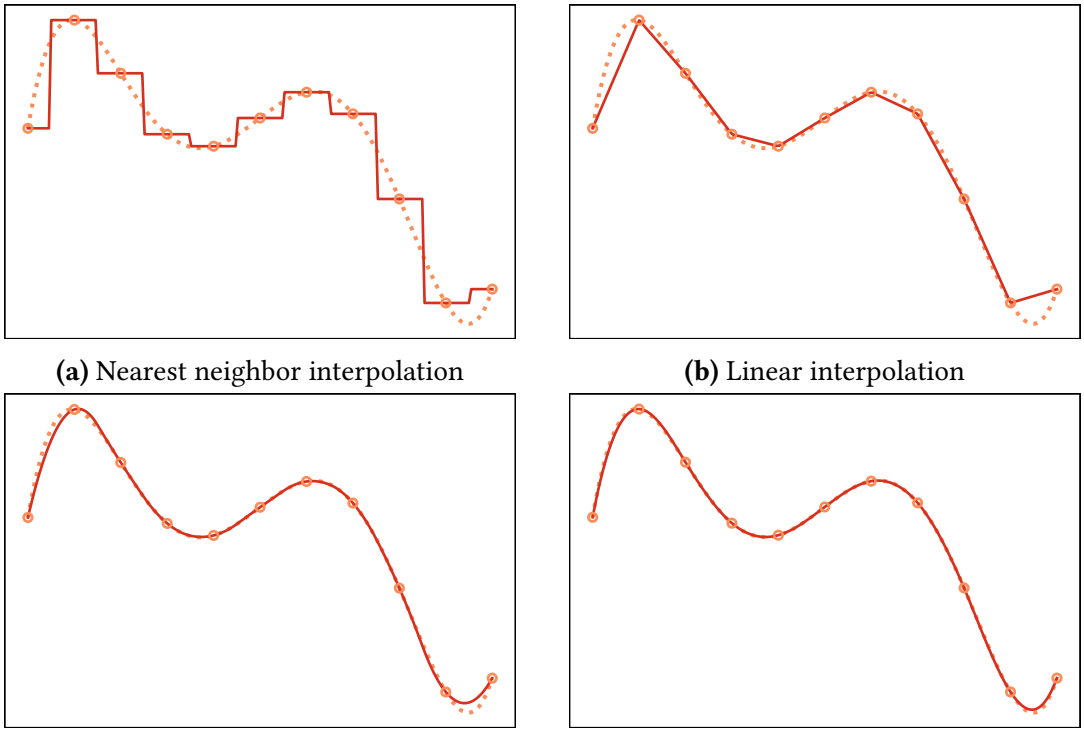
of the closed interval $[a, b]$ determines the domains of the piecewise polynomial spline functions $\mathcal{S}$ in the set $\mathcal{S}_\Delta$. These spline functions are joined at support abscissas, which, in the context of splines, are called *knots*.

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

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

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

Accordingly, spline functions $\mathcal{S}_\Delta$ of degree k are polynomial functions of degree k which are $(k-1)$ times continuously differentiable at the interior knots, that is, $\{x_i\}_{i=1}^{n-1}$, of the partition Δ. These k^{th}-order polynomials are uniquely determined by $k+1$ coefficients, of which k are given by the $(k-1)$ derivatives and function values at their left bordering knot.



(a) Nearest neighbor interpolation

(b) Linear interpolation

(c) Quadratic B-spline interpolation

(d) Cubic B-spline interpolation

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

B-splines are a family of nonnegative spline functions which have minimal support for any given degree, smoothness and domain partition. Furthermore, any spline function of a given degree can be expressed as a linear combination of B-splines of the same degree

(Stoer and Bulirsch 2002, pp.107–110). Accordingly, B-splines provide the foundation of efficient and numerically stable computations of splines. A selection of commonly used interpolation methods applied to a discretely sampled, high order polynomial is shown in figure 2.1. From the figure, the increased precision of higher order splines when applied to continuous functions is readily apparent. Note, however, that higher order interpolation methods require more sampling points than lower order methods. In particular, at least $k + 1$ samples are required in order to construct a k^{th} order spline. In higher dimensions, that is, with data which depends on several other (independent) variables, the required amount of samples increases rapidly with the interpolation order. Subsequently, the use of cubic B-splines constitutes a popular method for general-purpose interpolation, providing a good balance between numerical accuracy and computational complexity.

2.2 The type of flow systems considered

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

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

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

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

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

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

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

which, for our three-dimensional flow, reduces to

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

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

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

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

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

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

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

where the Kronecker delta is defined as

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

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

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

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

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

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

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

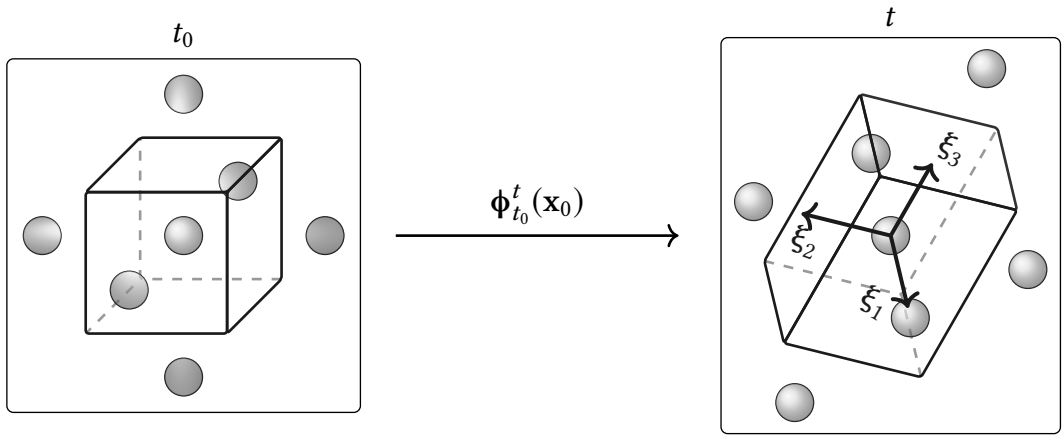


Figure 2.2: Geometrix interpretation of the eigenvectors of the Cauchy-Green strain tensor. The central unit cell is stretched and deformed under the flow map $\boldsymbol{\phi}_{t_0}^{t}(\mathbf{x}_0)$. The local stretching is the largest in the direction of ξ_3, the eigenvector which corresponds to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

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

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

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

2.3 Definition of Lagrangian coherent structures for three-dimensional flows

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

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

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

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

In the following, the entire material surface will be referred to by the notation $\mathcal{M}(t)$. Although no material surfaces can be crossed by tracers, only special material surfaces create coherence in the phase space $\mathcal{U}$ and thus act as observable transport barriers. Such material surfaces are generally referred to as *Lagrangian coherent structures* (henceforth abbreviated to LCSs).

Subsequently, LCSs can be described as time-evolving surfaces which shape coherent trajectory patterns in dynamical systems, defined over a finite time interval (Haller 2011).

There are three main types of LCSs, namely *elliptic*, *hyperbolic* and *parabolic*. Rougly speaking, parabolic LCSs outline cores of jet-like trajectories, elliptic LCSs describe vortex boundaries, whereas hyperbolic LCSs are comprised of overall attractive or repelling manifolds. As such, hyperbolic LCSs practically act as organizing centers of observable tracer patterns (Onu, Huhn, and Haller 2015). Because hyperbolic LCSs provide the most readily applicable insight in terms of forecasting flow in e.g. oceanic currents, such structures have been the focus of this project.

2.3.1 *Hyperbolic LCSs in three dimensions*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

which, for our three-dimensional flow, reduces to

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

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

From condition (2.28c) and the orthormality of the Cauchy-Green strain eigenvectors (cf. equation (2.25)), it follows that any initial LCS surface is tangent to the planes locally spanned by $\boldsymbol{\xi}_1(\mathbf{x}_0)$ and $\boldsymbol{\xi}_2(\mathbf{x}_0)$. Thus, an integral curve of any (normalized) linear combination of the $\boldsymbol{\xi}_1$- and $\boldsymbol{\xi}_2$-direction fields, launched from an arbitrary point of the surface $\mathcal{M}(t_0)$, will lever leave $\mathcal{M}(t_0)$. Hence:

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

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

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

3 Method

3.1 Flow systems defined by analytical velocity fields

3.1.1 *Steady Arnold-Beltrami-Childress flow*

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

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

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

Here, the parameter values

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

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

3.1.2 *Unsteady Arnold-Beltrami-Childress flow*

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

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

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

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

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

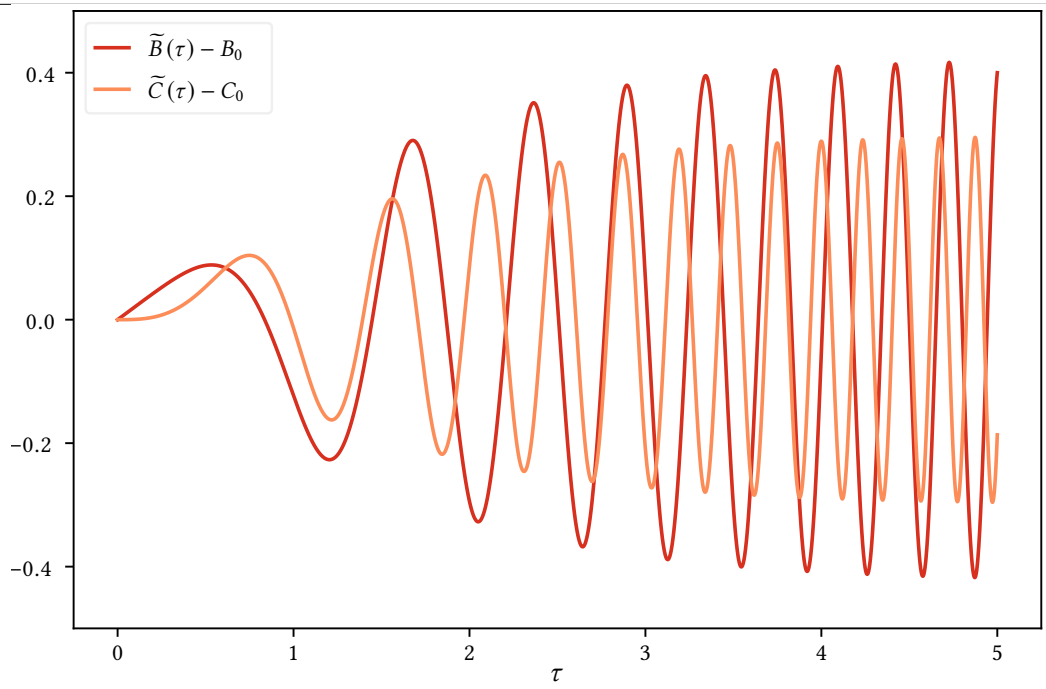


Figure 3.1: Time dependence of the coefficient functions for unsteady ABC flow, defined in equations (3.2)–(3.4).

3.2 Flow systems defined by gridded velocity data

3.2.1 *Oceanic currents in the Førde fjord*

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

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

To this end, gridded three-dimensional velocity data, modelling oceanic currents in the (depths of the) Førde Fjord, was made available by SINTEF Fisheries and Aquaculture. The data set considered here contains velocity data for the time period between June 1 2013, 00:00 and June 2 2013, 23:40, sampled in intervals of 20 minutes. For our simulations, the

time interval of interest was the 12 hour time window between 00:00 and 12:00 on June 1 2013 — practically ensuring the encapsulation of a tidal cycle.

Seeing as some of the most harmful contaminants found within the mine tailings are heavy metals, we concentrated our analysis on the depths of the fjord. Accordingly, we looked for LCSs in a region of water which was neither particularly close to the oceanic surface, nor hit the coastline when advected for the 12 hour duration of the time interval of interest. This limited our research to a 500 m × 500 m × 250 m region, with depths ranging from 50 m to 300 m below the surface. A bird's-eye view of the region is shown in white in figure 3.2, which also contains a map view of the geographical region.

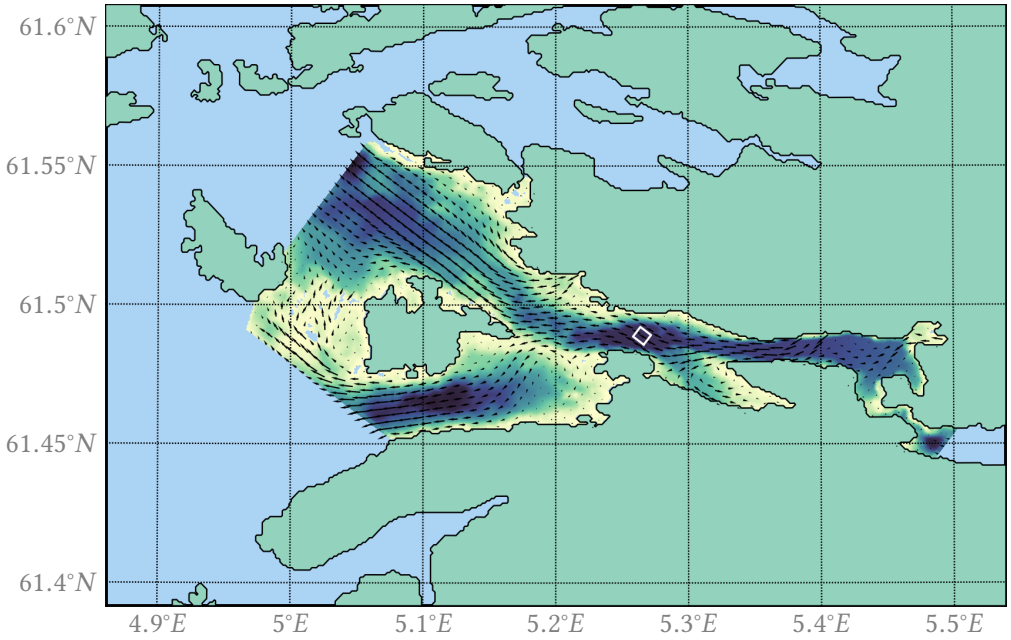


Figure 3.2: Stereographical map projection of the Førde fjord and its surroundings. The local fjord depths are indicated by a varying background color. A subset of the gridded velocity vectors indicates the macroscopic trends of the oceanic currents at a depth of 50 m below the surface. Outlined in white is a bird's-eye view of the main region of interest; namely, a region of water which was neither particularly close to the oceanic surface, nor struck the coastline when transported by the currents for the duration of the 12 hour time interval of interest.

3.2.2 *Interpolating gridded velocity data*

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

Several multidimensional B-spline interpolation libraries are publically available under open access licensing. For this project, we elected to use the Bspline-Fortran library, partly

motivated by its extensive documentation (Williams 2018). In particular, we made use of the `bspline_4d` derived type, which we, along with a subset of its type-bound procedures, made available in C making use of the Fortran standard interoperability with C-languages, that is, the `iso_c_binding` module — which is shipped with most modern Fortran compilers. From there, we wrote a thin wrapper class in C++, which was exposed to Python via Cython.

The choice of Python as our main programming language was made partly due to its relatively low development costs, in addition to its beneficial properties as a 'glue language' — as examplified by the above, and the ease of parallelization by means of e.g. MPI (the use of which will be outlined in greater detail in section 3.3.1 and ***SETT INN REF TIL AVSN OM BEREGNING AV MANGFOLDIGHETER***). Moreover, by utilizing Fortran's reference-based subroutine call structure, in addition to pointers at C-level (typed memoryviews in Cython), we were able to minimize memory duplication. This could otherwise have been a significant bottleneck.

3.3 Computing the flow map and its directional derivatives

3.3.1 *Advecting a set of tracers*

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

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

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

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

In this framework, the tracer advection takes second stage to the 'advection' of the components of the flow map Jacobian. As it turns out, the increase in mathematical complexity which the coupling terms introduces is a small price to pay for the increased precision compared to the straightforward approach of applying a finite diffference scheme to the advected flow map (Oettinger and Haller 2016). This is also evident from previous 'finite difference-based' LCS computing endeavors, in which the use of several grids of tracers was necessitated in order to accurately compute the flow map Jacobian (Løken 2017; Farazmand and Haller 2012a).

Nevertheless, simultaneously solving twelve coupled ODEs for the millions of initial conditions as suggested in table 3.1 quickly proved an unreasonably strenuous task for the author's own personal laptop. In order to accelerate the computational process, this computation was parallelized by means of MPI and run on NTNU's supercomputer, Vilje. The choice of MPI over alternative multiprocessing tools was mainly motivated by MPI facilitating access to multiple nodes within the Vilje cluster. Because the trajectories are independent, the parallelization process consisted of distributing an approximately even amount of tracers across all ranks, whereupon each rank advected (that is, simultaneously solved the twelve coupled ODEs for all of) its allocated tracers. In the end, all of the final state flow map Jacobians were collected by the designated main process (i.e., rank = 0), whereupon the Cauchy-Green strain eigenvalues and -vectors were computed — which will be elaborated upon in section 3.4. The associated speedup (and increase in available memory) was crucial for this project.

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

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

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

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

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

were used throughout.

As a measure of the numerical error,

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

is used. Then, err is compared to unity in order to find an optimal step size. From definition 2, it follows that err scales like h^{q+1}, where $q = \min(p, \widehat{p})$. Under the assumption $1 \approx K h_{\mathrm{opt}}^{q+1}$,

one finds the optimal step size from

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

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

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

where fac and $\text{fac}_{\max}$ are numerical safety factors, intended to prevent increasing the step size *too* much. Here, the parameter values

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

were used throughout.

3.4 Computing Cauchy-Green strain eigenvalues and -vectors

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

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

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

Interpolating the Cauchy-Green strain eigenvalues

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

Interpolating the Cauchy-Green strain eigenvectors

Just like the eigenvalues of the Cauchy-Green strain tensor field, its eigenvectors frequently need to be evaluated inbetween the grid points in order to compute LCSs. Like the strain eigenvalues, the strain eigenvectors were interpolated by means of cubic trivariate B-splines, through the `bspline_3d` derived type from the Bspline-Fortran library (Williams 2018) — albeit with a twist, in order to remove local orientational discontinuities. In particular, the local stretch is equal in magnitude along any given negative $\boldsymbol{\xi}_i$ axis as that of its positive counterpart, and there is no a priori reason to expect the SVD decomposition (cf. equation (3.11)) to follow any particular convention regarding the 'sign' of the computed eigenvectors.

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

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

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

Repelling LCSs in three spatial dimensions are quite challenging to compute. Straightforward numerical integration of the flow is insufficient — in three dimensions, the existence

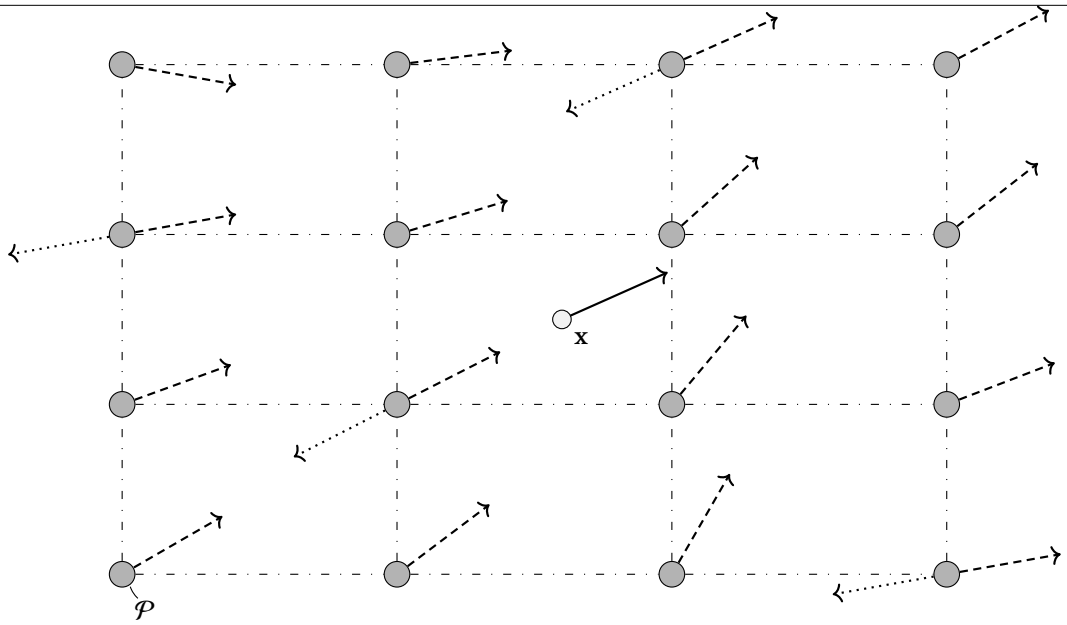


Figure 3.3: Conceptual illustration of the special-purpose cubic interpolation routine for the Cauchy-Green strain eigenvectors. The 64 nearest grid points (16 in two dimensions; here shown in gray) to any given coordinate $\mathbf{x}$ are identified. As the local stretch is equal in magnitude along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

criterion (2.28c) provides another degree of freedom, as, everywhere within such an LCS, one is allowed to move 'freely' within a plane which is orthogonal to $\xi_3(\mathbf{x})$. Dedicated algorithms are needed. Here, we consider a (variation of) the method of geodesic level sets for computing repelling LCSs as invariant manifolds of the ξ_1 and ξ_2 direction fields (cf. remark 1), as presented by Krauskopf et al. (2005). The short presentation to follow in the next paragraph will be explained further in depth in the subsequent subsections.

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

3.5.1 *Identifying suitable initial conditions for developing LCSs*

Inspired by the two-dimensional approach of Farazmand and Haller (2012a), in order to identify repelling LCSs, the first step was to identify the subdomain $\mathcal{U}_0 \subset \mathcal{U}$ in which the existence conditions (2.28a), (2.28b) and (2.28d) are satisfied — as these conditions can be verified for individual points, unlike criterion (2.28c). All grid points which lie within $\mathcal{U}_0$ would then be valid initial conditions for repelling LCSs. Of the aforementioned criteria, condition (2.28d) is the least straightforward to implement numerically, as identifying the zeros of inner products is prone to numerical round-off error. Our approach is based on comparing the value of λ_3 at a given grid point $\mathbf{x}_0$ to the values of λ_3 at the two points $\mathbf{x}_0 \pm \varepsilon\boldsymbol{\xi}_3(\mathbf{x}_0)$, where ε is a number one order of magnitude smaller than the grid spacing. Should $\lambda_3(\mathbf{x}_0)$ be the largest of the three, the point $\mathbf{x}_0$ would be flagged as satisfying criterion (2.28d).

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

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

	Analytical ABC flow	Fjord model data
ε	$5 \cdot 10^{\cdot -3}$	10^{-1}
ν	8	5
# initial conditions without ν-filtering	340951 (steady) 361461 (unsteady)	209945
# initial conditions with ν-filtering	618 (steady) 676 (unsteady)	1631

3.5.2 *Parametrizing the innermost level set*

For an initial condition $\mathbf{x}_0$ identified by means of the method outlined in section 3.5.1, the corresponding LCS must locally be tangent to the plane with unit normal given by $\boldsymbol{\xi}_3(\mathbf{x}_0)$,

as a consequence of LCS existence criterion (2.28c). Accordingly, the first geodesic level set is approximated by a set of n points $\{\mathcal{M}_{1,j}\}_{j=1}^{n}$, which all lie within the aforementioned tangent plane, separated from $\mathbf{x}_0$ by a distance δ_{init}; all of which assumed to lie within the same manifold. The parameter δ_{init} was chosen small compared to the grid spacing, in order to limit the inherent errors of this linearization.

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

In an intermediary step, the coordinates of the mesh point $\mathcal{M}_{1,1}$ was appended to a list containing the ordered coordinates of the mesh points $\{\mathcal{M}_{1,j}\}_{j=1}^{n}$, such that, aside from $\mathcal{M}_{1,1}$, which corresponded to both $s = 0$ and $s = 1$, there was a one-to-one correspondence between the s-values and the coordinates of the mesh points. Then, considering each of the lists containing the Cartesian coordinates of the mesh points as univariate functions of the pseudo-arclength parameter s, separate cubic B-splines were made for each set of coordinates, making use of the `bspline_1d` extension type from the Bspline-Fortran library (Williams 2018), which was ported to Python as outlined in section 3.2.2. The constructed innermost level set and its associated interpolation curve is illustrated in figure 3.4.

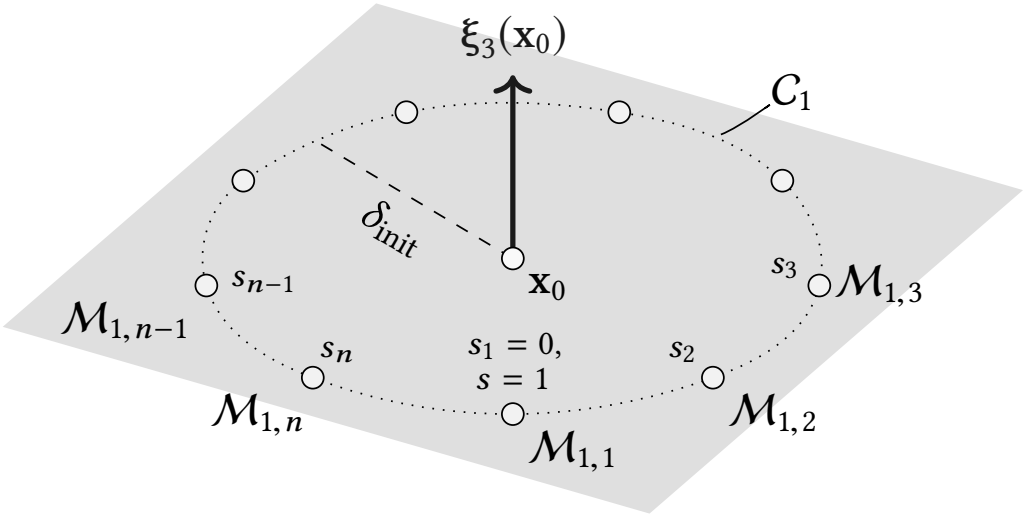


Figure 3.4: The construction of the innermost geodesic level set. An initial condition $\mathbf{x}_0$ is found by means of the method outlined in section 3.5.1. A set of n meshpoints $\{\mathcal{M}_{1,j}\}_{j=1}^{n}$ is then evenly distributed within the plane defined by the point $\mathbf{x}_0$ and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

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

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

For the subsequent level sets, Krauskopf et al. (2005) suggest determining the tangents $\mathbf{t}_{i,j}$ using the interpolation curve C_i, by drawing a vector between two points equidistant to $\mathcal{M}_{i,j}$ in either direction along C_i. However, an inheritance-based approach was found to yield more smoothly parametrized manifolds, which were less sensitive to numerical noise. This approach, along with the treatment of special cases, will be explained in greater detail in the sections to follow (in particular, section 3.8.1).

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

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

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

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

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

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

3.6 Legacy approach to computing new mesh points

As tentatively suggested in section 3.5.2, each of the points in the first level set $\mathcal{M}_1 = \{\mathcal{M}_{1,j}\}_{j=1}^{n}$ is used to compute a point in the ensuing level set $\mathcal{M}_2$. This notion extends to all of the subsequent level sets; namely, the points in level set $\mathcal{M}_{i+1}$ are computed from the points in the prior level set $\mathcal{M}_i$. For reasons of brevity in the discussion to follow, denote the points

$\{\mathcal{M}_{i,j}\}$ and $\{\mathcal{M}_{i+1,j}\}$ as *ancestor* and *descendant points*, respectively. Furthermore, the set of mesh points which can be traced backwards to a single, common ancestor, is referred to as a *point strain*. The considerations to follow rely on each mesh point $\mathcal{M}_{i,j}$ inheriting its tangential vector from its direct ancestor; that is, $\mathbf{t}_{i,j} := \mathbf{t}_{i-1,j}$. The treatment of the special cases of this inheritance-based approach will be described in greater detail in section 3.8.1.

From the mesh point $\mathcal{M}_{i,j}$, we wish to place a new mesh point $\mathcal{M}_{i+1,j}$ at an intersection of the manifold $\mathcal{M}$ and the half-plane $\mathcal{H}_{i,j}$ located a distance Δ_i from $\mathcal{M}_{i,j}$. The aforementioned half-plane is defined by the coordinate $\mathbf{x}_{i,j}$, the unit normal $\mathbf{t}_{i,j}$ and the guidance vector $\boldsymbol{\rho}_{i,j}$ (cf. equations (3.13) and (3.14)). Note that this intersection may occur anywhere on the half-circle within $\mathcal{H}_{i,j}$ of radius Δ_i, centered at $\mathbf{x}_{i,j}$. The search for a new mesh point is conducted by defining an aim point $\mathbf{x}_{\text{aim}}$ within $\mathcal{H}_{i,j}$, computed by performing a single, classical, 4^{th}-order Runge-Kutta step (cf. table 2.3) of length Δ_i in the vector field locally defined as

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

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

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

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

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

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

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

As suggested by Krauskopf et al. (2005), the starting point for all trajectories intended to reach $\mathbf{x}_{\text{aim}}$ could be chosen as any point not equal to $\mathbf{x}_{i,j}$ along the parametrized curve C_i. However, trajectories starting out at points along C_i which are far removed from $\mathbf{x}_{i,j}$ are likely to require a long integration path, which would result in an increase in the accumulated numerical round-off error. Subject to such errors, these trajectories might not even get close

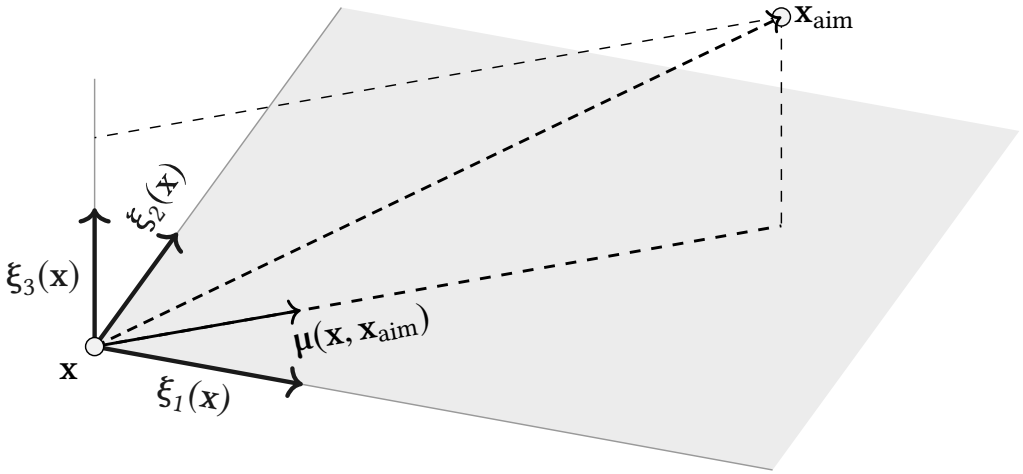


Figure 3.5: Visualization of the direction field used to compute trajectories within a manifold, using the legacy approach. The normalized direction $\boldsymbol{\zeta}(\mathbf{x}, \mathbf{x}_{\text{aim}})$ is found by normalizing the projection of $\mathbf{x}_{\text{aim}} - \mathbf{x}$ into the plane spanned by the local ξ_1- and ξ_2 vectors (shaded).

to $\mathbf{x}_{\text{aim}}$ with a reasonable computational resource consumption. Accordingly, we limited the potential number of trajectories to compute by only considering a subset of the interpolation curve C_i as initial conditions, as follows:

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

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

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

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

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

whereas

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

sufficed for it to be flagged as lying a distance Δ_i from $\mathbf{x}_{i,j}$, with $\gamma_\mathcal{H}$ and γ_Δ small numbers. When a trajectory first intersected $\mathcal{H}_{i,j}$, the integration was stopped abrubtly, leaving its endpoint $\mathbf{x}_{\text{fin}}$ as its suggested coordinates for the new mesh point. As briefly mentioned in section 3.5.2, the unit tangent vectors $\mathbf{t}_{i,j}$ were generally inherited — the treatment of special cases will be explained in greater detail in section 3.8.1. Figure 3.6 depicts a few characteristic trajectory patterns which commonly occured when searching for new mesh points in the fashion discussed in the above.

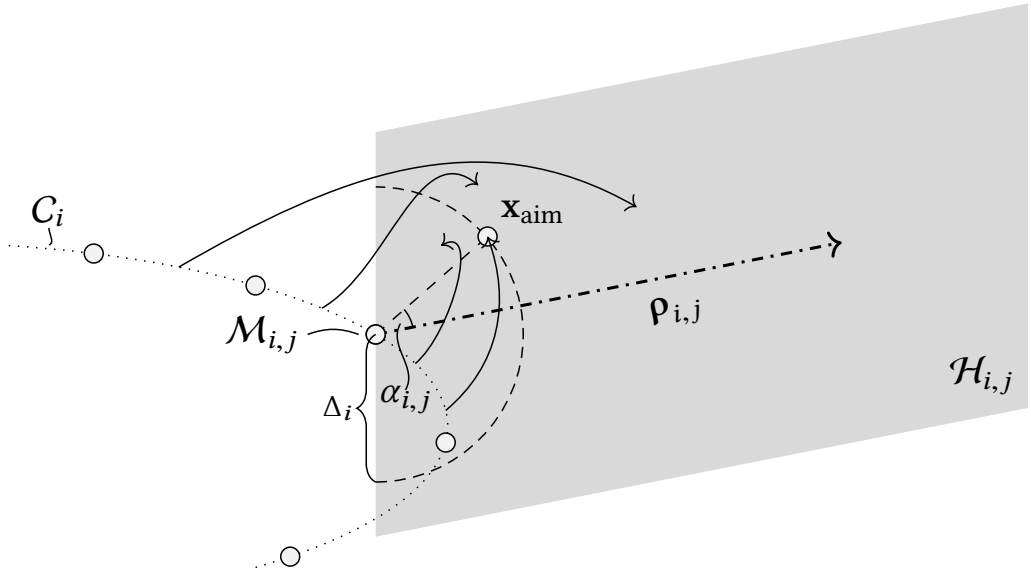


Figure 3.6: Visualization of typical trajectories used to compute a new mesh point, using the legacy approach. The aim point $\mathbf{x}_{\text{aim}}$ is used to guide trajectories which are locally orthogonal to the ξ_3 direction field (cf. equation (3.17)) towards the intersection between the manifold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

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

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

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

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

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

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

for $\check{s}_1 < \check{s}_2$, then the intermediate value theorem implies that we must have

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

under the assumption that $\Delta(\check{s})$ is a continuous function. In order to optimize our use of computational resources, we thus endeavor to take large steps when moving along C_i whenever the computed intersections with $\mathcal{H}_{i,j}$ are far from fulfilling $\Delta(\check{s}) = \Delta_i$. However, when a subinterval of C_i within which the intermediate value theorem suggests that a trajectory may fulfill our requirements is identified, we decrease the (quasi-)arclength increment $\delta\check{s}$ in order to increase our odds of finding said trajectory. The increment $\delta\check{s}$ was not allowed to decrease beyond a pre-set $\delta\check{s}_{\text{min}}$, nor to increase beyond a pre-set $\delta\check{s}_{\text{max}}$. The latter limitation was made in order to avoid bypassing subsets of C_i from which two or more trajectories satisfy $\Delta(\check{s}) = \Delta_i$. Overstepping any region containing an even number of such intersections could render it undetectable using our intermediate value theorem-based approach, as no change in trajectory termination status need be detected.

The feedback received by tracking why each trajectory is terminated, allows us to dynamically select new trajectory start points along C_i. We do so by increasing the (quasi-)arclength increment $\delta\check{s}$ as long as there is no change in trajectory termination status, and, conversely, backtracking and reducing $\delta\check{s}$ when a status change is detected. This process is shown schematically in figure 3.7. Note that our algorithm is not perfect; theoretical closed loops exist (left for the interested reader to find, as an exercise[1]). However, in our experience, the iterative search process never became stuck beyond recovery; likely due to the direction field (see equation (3.17)) yielding well-behaved trajectories (and, possibly, numerical round-off error). As we assume asymptotic behaviour close to any regions in which $\Delta(\check{s})$ is not defined (that is, regions where no trajectories reach the half-plane $\mathcal{H}_{i,j}$, cf. equation (3.21)), the adjustment of $\delta\check{s}$ is treated in the same fashion there.

3.6.3 *Handling failures to compute satisfactory mesh points*

As mentioned in section 3.6.1, it can never be guaranteed that any trajectories which start out at some point along the smooth curve C_i will be able to generate a new mesh point located within $\mathcal{H}_{i,j}$ which simultaneously satisfies all of our defined constraints. Missing out on points in any freshly generated level set prohibits the generation of more level sets — in

[1] Hint: What would happen if the very first computed trajectory failed to intersect with $\mathcal{H}_{i,j}$?

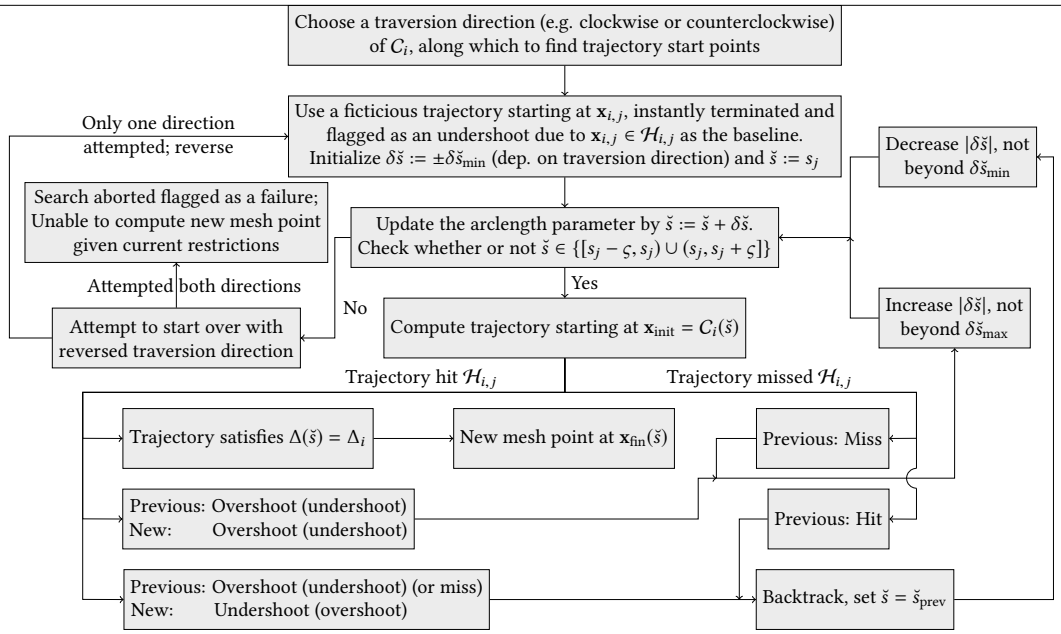


Figure 3.7: Flowchart illustrating the algorithm for iteratively choosing new trajectory start points based on the termination status of the preceding trajectories, using the legacy approach. All possible trajectory start points are contained within a subset of C_i, per equation (3.18). Whether or not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

particular, partly dependent on the number of successfully computed points, the smoothness of the interpolation curve C_{i+1} exhibits a critical dependence on *all* of the points used for its creation. Accordingly, handling tricky mesh points is crucial.

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

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

with

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

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

Repeating the iterative point search algorithm outlined in the entirety of the current section (that is, section 3.6) for all possible perturbation configurations of $\mathbf{x}_{\text{aim}}$, given by equation (3.23), however, still does not guarantee that an acceptable mesh point is found.

Should that be the case, the entire algorithm — including angular perturbations as outlined in the above — is then rerun with simultaneously and progressively relaxed point acceptance criteria. That is, the numerical tolerance parameters $\gamma_{\mathcal{H}}$ and γ_{Δ} (cf. equations (3.19) and (3.20)) are gradually increased up to pre-set maximum values $\gamma_{\mathcal{H}}^{\max}$ and $\gamma_{\Delta}^{\max}$.

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

3.7 Revised approach to computing new mesh points

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

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

3.7.1 *Computing pseudoradial trajectories directly*

Completely analogously to the legacy approach outlined in section 3.6, we define a local direction field as

$$\boldsymbol{\psi}(\mathbf{x}) = \frac{\boldsymbol{\xi}_3(\mathbf{x}) \times \mathbf{t}_{i,j}}{\left\| \boldsymbol{\xi}_3(\mathbf{x}_0) \times \mathbf{t}_{i,j} \right\|}, \tag{3.24}$$

where $\mathbf{t}_{i,j}$ is the unit tangent associated with the mesh point $\mathcal{M}_{i,j}$. Here, however, we make explicit use of our previously mentioned additional degree of freedom — namely that trajectories within the parametrized manifold are allowed arbitrary movements within the planes which are locally orthogonal to the $\boldsymbol{\xi}_3$ direction field, rather than being constrained to moving along a three-dimensional curve — by computing the coordinates of the new mesh point $\mathcal{M}_{i+1,j}$ as the end point of a *single* trajectory.

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

Should the $\boldsymbol{\xi}_3$ direction be parallel to the unit tangent $\mathbf{t}_{i,j}$ locally along the trajectory, the direction field equation (3.24) would become undefined. In such regions, we allowed the Runge-Kutta solver to take a step in the previous direction as was used for the immediately preceding step. Numerically, such regions were recognized by

$$\left\| \boldsymbol{\xi}_3(\mathbf{x}) \times \mathbf{t}_{i,j} \right\| < \gamma_{\|}, \tag{3.25}$$

where $\gamma_{\|}$ is a small tolerance parameter. In order to avoid overstepping, the step length of the Dormand-Prince solver was continuously limited from above by $\left\| \mathbf{x} - \mathbf{x}_{i,j} \right\| - \Delta_i$. Moreover, the total allowed integration arclength was limited to a scalar multiple of Δ_i, allowing for the termination of any (hypothetical) trajectory which would end up in a stable orbit.

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

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

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

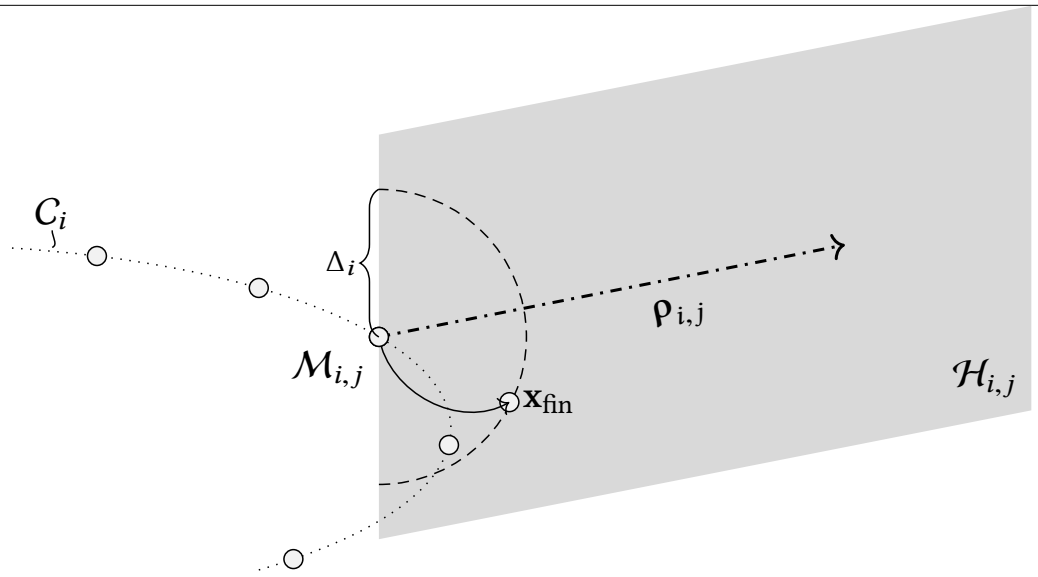


Figure 3.8: Visualization of a typical trajectory used to compute a new mesh point, using the revised approach. A trajectory is computed, starting at $\mathbf{x}_{i,j}$ (that is, the coordinates corresponding to the mesh point $\mathcal{M}_{i,j}$) and moving in the direction field given by equation (3.24),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

3.7.2 *Handling failures to compute satisfactory mesh points*

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

Seeing as the only tolerance parameter involved in this method pertains to achieving the desired separation between a meshpoint and its direct descendant — see equation (3.26) — we elected not to progressively relax this constraint. Instead, we interpreted the failure of any trajectory to reach a point sufficiently far away, as the trajectory being coiled such that the required integration arc length to yield a point sufficiently far away from the ancestor mesh point pre-set allowed path length. Accordingly, the incomplete level set was discarded, and attempted to be recomputed with reduced Δ_i (for which the dynamic adjustment procedure will be described in detail in section 3.8.3). Should an entire, new geodesic level set prove incomputable even at the minimum permitted step length, attempts at further expansion of the computed manifold were abandoned. In such cases, this variant of the method of geodesic

level sets simply did not suffice, for the given set of development parameters. Details on other stopping criteria for the generation of manifolds will be presented in section 3.10.

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

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

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

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

3.8 Managing mesh accuracy

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

3.8.1 *Maintaining mesh point density*

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

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

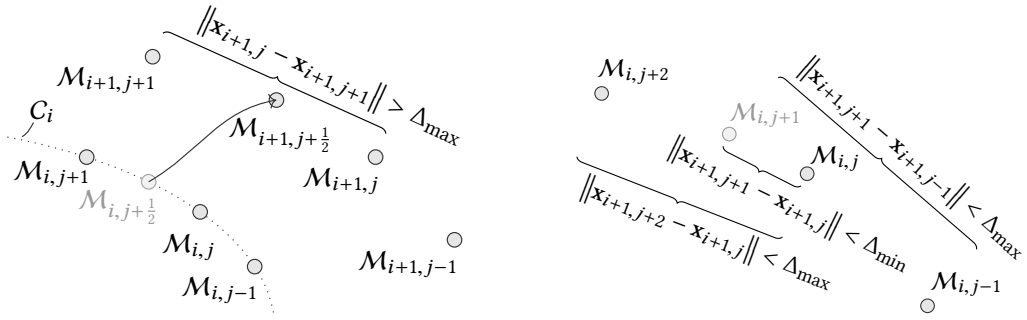
Conversely, if any of the nearest neighbor separations became smaller than $\Delta_{\min}$, we sought to remove one of them, as long as the distance separating mesh points which would then become nearest neighbors did not exceed $\Delta_{\max}$. Accordingly, if any one of a pair of neighboring mesh points was to be removed, we chose to discard the one which would result in the smallest separation between the ensuing, *new* pair of nearest neighbors. Our principles for inserting new mesh points inbetween others, and removing grid points which are too close together, are illustrated in figure 3.9.

3.8.2 *Limiting the accumulation of numerical noise*

Early tests with regards to the generation of manifolds revealed that the compound numerical error over the course of many level sets often resulted in irregular behaviour. Specifically, small bulges in the mesh easily became amplified in the subsequent level sets, which frequently caused unwanted loops in the interpolation curves $\{C_i\}$ which extended far from the manifold epicentre. Occasionally, this would lead to the generated manifold to fold into itself. This *cannot* happen in repelling LCSs. In particular, self-intersecting manifolds indicate that, in a small neighborhood of any intersection, there would not be a well-defined direction of strongest repulsion, which would violate LCS existence criterion (2.28a).

As a countermeasure against the formation of undesired loops within a level set, we reviewed a recently computed (suggestion for a) geodesic level set, as follows: Consider a point $\mathcal{M}_{i+1,j}$ located at $\mathbf{x}_{i+1,j}$. If, for any point $\mathcal{M}_{i+1,j+k}$ with $k > 1$, the interpoint distances satisfy

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



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

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

Figure 3.9: Our approach to inserting new, or removing, mesh points to maintain mesh point density. When two neighboring mesh points in a freshly computed level set are too far apart with regards to the given mesh parameter $\Delta_{\max}$, we attempt to insert a new mesh point between them. As shown in (a), this is done by using the method described in section 3.7, using a ficticious initial condition midway inbetween the two ancestor mesh points $\mathcal{M}_{i,j}$ and $\mathcal{M}_{i,j+1}$ along the interpolated curve C_i, indicated in the figure by a lighter shade of gray. Should two neighboring mesh points be too close together, and one of the two can be removed without the resulting sets of neighboring points being too far apart, we remove the one which results in the shortest interpoint separation (as shown in (b), where the point which is removed is indicated with a lighter shade of gray).

and

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

then all mesh points $\{\mathcal{M}_{i+1,j+\kappa}\}_{\kappa=1}^{k-1}$ are removed. Here, $0 \leq \gamma_{\cup} \leq 1$ is a bulge tolerance parameter, which essentially determines an upper limit for the extent (measured in cumulative arclength) of loop-like segments of any given interpolation curve C_i. Specifically, a large $\gamma_{\cup}$ facilitates removal of rounded loops, whereas a small $\gamma_{\cup}$ restricts the removal process to sharp bulges. A characteristic example demonstrating this method of removing unwanted loops is shown in figure 3.10. While possibly sacrificing some resolution of the manifold as a whole, adhering to criterion (3.27) prevents the removal of any *reasonable* bulge formations. As no new mesh points are *added* as a direct consequence of this loop-removal algorithm, it does not introduce new errors. It is also worth mentioning that some of the removed mesh points were recovered in the ensuing level sets, then as constituents of an overall smoother level set.

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

Given the interpoint separation constraints described in section 3.8.1, we have some flexibility regarding the choice of interset step length Δ_i. In an approach closely mirroring that of Krauskopf et al. (2005), we used the (approximate) local curvatures along each point strain (i.e., the set of mesh points which can be traced back to a common ancestor) in order to determine whether or not the local manifold dynamics were resolved to a satisfactory

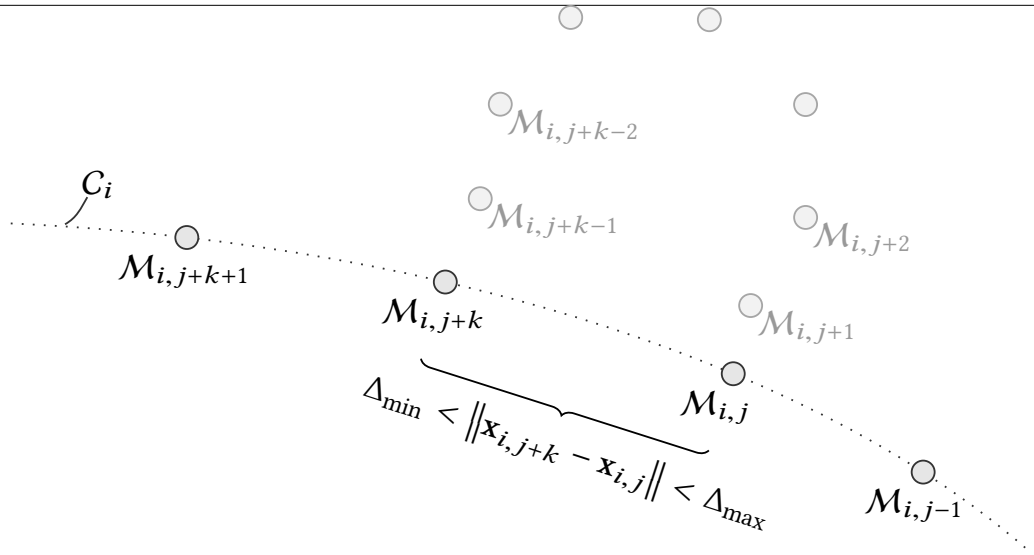


Figure 3.10: Our approach to limit the compound numerical error, by continuously removing unwanted loops in the computed level sets. As the separation between the mesh points $\mathcal{M}_{i,j}$ and $\mathcal{M}_{i,j+k}$ satisfies the restrictions regarding mesh point density (cf. section 3.8.1), the mesh points $\{\mathcal{M}_{i,j+\kappa}\}_{\kappa=1}^{k-1}$ (shown in a lighter shade of gray) are removed, provided that the cumulative neighbor separation around the bulge is sufficiently large (per the conditions presented in equation (3.27)) — which is the case in the above. The resulting interpolation curve C_i becomes significantly more well-behaved without the superfluous mesh points.

level of detail. Starting out with an initial interset separation $\Delta_1 = 2\Delta_{\min}$ for the second innermost level set (i.e., the first level set which was computed using the method described in section 3.7), we sought to ensure that the subsequent Δ_i resulted in the encapsulation of the finer details of the growing manifolds.

Specifically, once all mesh points which constitute a geodesic level set $\mathcal{M}_{i+1}$ have been identified, all a distance Δ_i away from their direct ancestor points in the previous level set $\mathcal{M}_i$, we computed the angular offsets $\alpha_{i,j}$ of each pair of guidance vectors $\boldsymbol{\rho}_{i,j}$ and $\boldsymbol{\rho}_{i+1,j}$ (as defined in equations (3.13) and (3.14)). This is sketched in figure 3.11. If

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

where $\alpha_{\downarrow}$ and $(\Delta\alpha)_{\downarrow}$ are upper curvature tolerance parameters, was satisfied, the level set $\mathcal{M}_{i+1}$ was discarded and recomputed with reduced Δ_i. If Δ_i was already as low as the pre-set $\Delta_{\min}$, the manifold generation process was terminated, as a new level set $\mathcal{M}_{i+1}$ could not be computed under the given constraints on interpoint separation (cf. section 3.8.1). Conversely, if

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

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

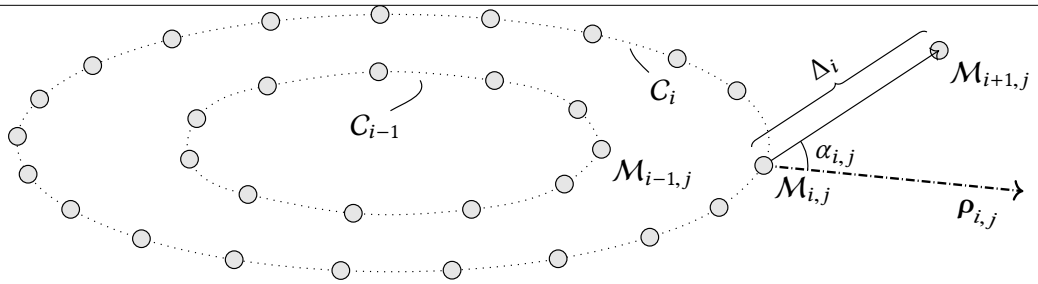


Figure 3.11: The principles of curvature-guided interset step length adjustment. For each of the mesh points $\{\mathcal{M}_{i+1,j}\}$ constituting the level set $\mathcal{M}_{i+1}$, the angular offsets $\alpha_{i,j}$ between the guidance vectors $\boldsymbol{\rho}_{i,j}$ (dashdotted) and $\boldsymbol{\rho}_{i+1,j}$ (not shown, but parallel to the vector separating $\mathcal{M}_{i,j}$ and $\mathcal{M}_{i+1,j}$, shown in solid) were computed. These were used in conjunction with Δ_i, the interset step used to compute the new mesh points, in order to determine whether or not the suggested level set would have to be discarded due to the local curvature along at least one point strain being sufficiently large to comply with criterion (3.28). If the level set was deemed acceptable, local curvature estimates $\{\alpha_{i,j}\}$ and $\{\Delta_i \cdot \alpha_{i,j}\}$ were then used to determine if the *subsequent* level set $\mathcal{M}_{i+2}$ could be computed using $\Delta_{i+1} > \Delta_i$; namely, if criterion (3.29) was satisfied.

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

3.9 Continuously reconstructing manifold surfaces from point meshes in 3D

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

To our knowledge, all three-dimensional plotting algorithms rely on some sort of triangulation method to regularize a pointwise parametrized surface. General-purpose routines for triangulation generation were found to be unsuitable, due to the specific mesh structure of $\mathcal{M}$, arising from the parametrization by geodesic level sets. For instance, Delaunay triangulation not only resulted in omitting triangles which, to the naked eye, were crucial

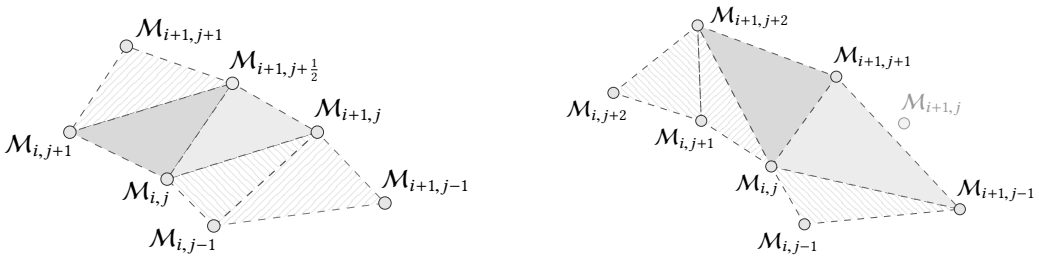
for the overall manifold structure, but also generated a lot of undesirable surface triangles — a problem which became increasingly prominent near creases. Accordingly, we made use of the fact that the `plot_trisurf` routine from the Python plotting library `Matplotlib` accepts an optional input argument specifying a list of triangles to plot, given by their vertices, and made our own triangulation scheme based on the specific structure of the mesh point parametrization of the computed manifolds.

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

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

The special cases not involving a one-to-one correspondence between the mesh points in $\mathcal{M}_i$ and $\mathcal{M}_{i+1}$ require special attention. This occurs whenever mesh points are inserted by making use of ficticious ancestors, or when mesh points are removed, in order to maintain the mesh point density (cf. section 3.8.1) — or, when removing unwanted bulges in order to dampen the effects of compound numerical noise (as described in section 3.8.2). The treatment of these special cases is shown in figure 3.12, and will be outlined in greater detail in the upcoming paragraph. In particular, note how all of $\mathcal{M}_{i,j}$'s nearest neighbors in the surrounding level set $\mathcal{M}_{i+1}$ are used in the triangulations.

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



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

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

Figure 3.12: How our triangulation algorithm handled the special cases arising when the one-to-one correspondence between the points in subsequent level sets was broken. In (a), a mesh point is inserted inbetween the points $\mathcal{M}_{i+1,j}$ and $\mathcal{M}_{i+1,j+1}$, while, in (b), the mesh point $\mathcal{M}_{i+1,j}$ is removed. Both of these adjustments were made in order to maintain the density of mesh points (cf. section 3.8.1). The ensuing pair of triangles which are computed when considering the triangulations from $\mathcal{M}_{i,j}$, are shaded. The remaining triangles (patterned) arise when triangulating from different points in level set $\mathcal{M}_i$. Note that the removal of bundles of mesh points in order to dampen the effect of numerical noise (which introduces bulges, as described in section 3.8.2) is treated analogously to the case of single missing points.

3.10 Macroscale stopping criteria for the expansion of computed manifolds

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

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

along each direction).

3.10.1 *Continuous self-intersection checks*

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

Our method of detecting manifold self-intersections is based on comparing the continuously computed interpolation triangles (as described in section 3.9) — in particular, we compared each triangle which was added together with the most recently computed level set, to all of the triangles which had been added with the preceding level sets. If at least one pair of triangles intersected, the newest level set was flagged as self-intersecting. Our way of determining whether or not two triangles intersect, is based on the Möller-Trumbore ray-triangle intersection algorithm, with a detection sensitivity parameter $\epsilon = 10^{-8}$ (Möller and Trumbore 1997). A visual representation of our triangle-intersection algorithm is available in figure 3.13.

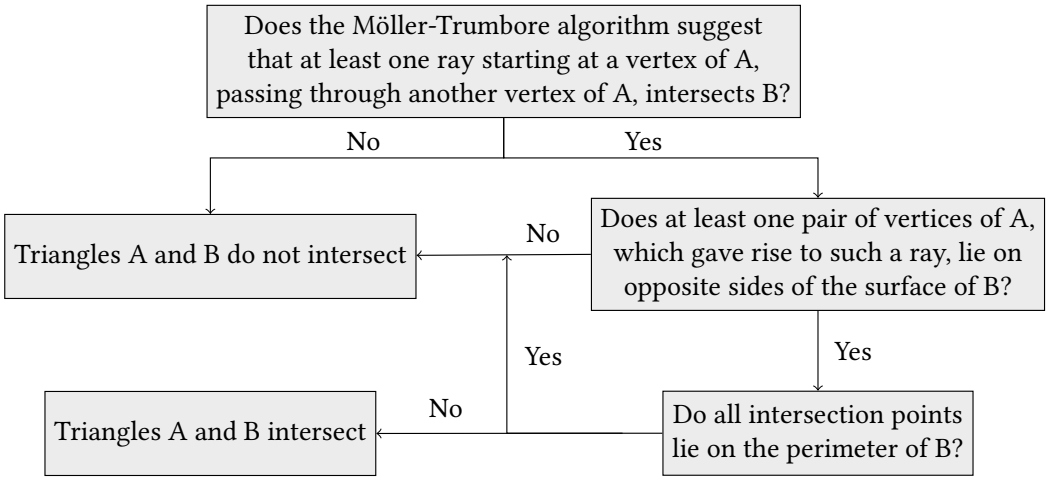
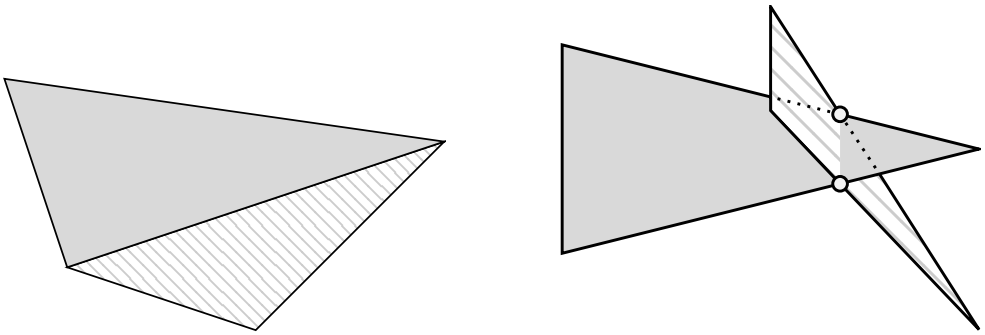


Figure 3.13: Flowchart illustrating the algorithm for detecting self-intersections. In order to ensure that no self-intersection went unnoticed, the indicated procedure was carried out by comparing each of the interpolation triangles A in the most recently computed level set, to all of the triangles B in the previously computed level sets. Provided that any of the new triangles intersected any of the old ones, the new level set as a whole was flagged as self-intersecting. How this intersection-detection approach handled a set of special cases is illustrated in figure 3.14.

Some special cases of intersecting triangles warrant special treatment. In particular, as our triangulation method (outlined in section 3.9) is based on generating triangles which share sides with its neighbors, we decided to allow triangles to intersect along the edges. Similarly,

we allowed for two triangles to be identical — which might happen if a manifold folds onto itself perfectly. Our algorithmic way of treating these cases resulted in a theoretical false negative; namely, the case of two triangles intersecting in exactly two points, laying along the edges of *both* triangles. In our experience, however, this never proved problematic — should one pair of triangles happen to intersect in this exact fashion (which is a rarity in itself due to numerical round-off errors), another pair of triangles would intersect in such a way that the most recently added set as a whole would be flagged as self-intersecting. Two of the aforementioned special cases — that is, two triangles sharing an edge, and two triangles intersecting in exactly two points laying along the edges of both triangles — are shown in figure 3.14.



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

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

Figure 3.14: How the intersection-detection algorithm handles special cases. Triangles which share a common edge, as illustrated in (a), form the premise of our triangulation algorithm; accordingly, this scenario does not get flagged as an intersection. Neither does the case when two triangles are identical (or one is contained within the other) — which is not shown here — nor the case when the edges of two triangles intersect in exactly two unique points, shown in (b). The latter is a very marginal case which hardly ever occurs, and, for our purposes, would invariably coincide with another (nearby) pair of triangles intersecting in a different manner than these special cases; ensuring that the computed level set as a whole would be flagged as self-intersecting.

Initial tests revealed that some self-intersections were quite innocuous, in that, if the computed manifold was expanded by an additional level set, the newly added triangulations need not necessarily intersect with any of the preceding ones. This kind of insipid intersections could be a consequence of the linear nature of our triangulation method, possibly compounded by round-off error. Accordingly, we decided to stop the manifold expansion process if *several consecutive* geodesic level sets introduced intersection triangulations. This was done by computing the sum of the interset distances $\{\Delta_i\}$ for each consecutive level set $\{\mathcal{M}_i\}$ which introduced new intersections. Whenever this pseudo-intersection length exceeded $5 \cdot \Delta_{\text{min}}$ (cf. section 3.8.1), we terminated the manifold computation process; empirical trials suggested that the intersection issue would then only worsen if more mesh points were added.

3.11 Identifying LCSs as subsets of computed manifolds

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

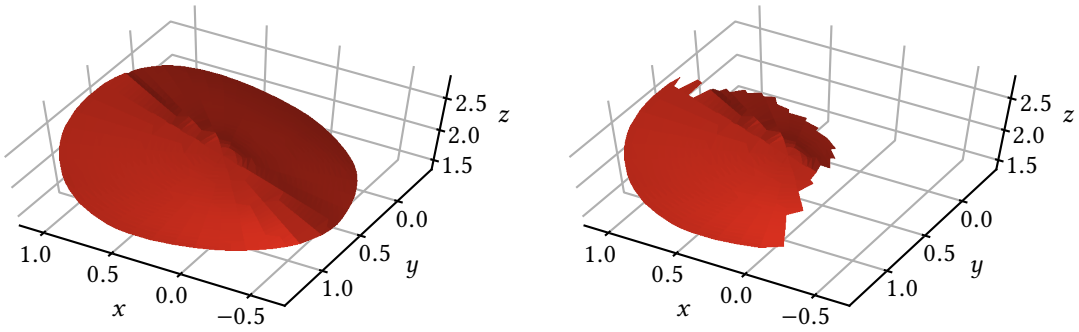
We then proceded to construct a repelling LCS $\mathcal{L}$ from the mesh points of $\mathcal{M}$. Per section 3.5.1, the mesh point at the centre of any given manifold always satisfies all the LCS existence criteria; accordingly, it was added as the first mesh point $\mathcal{L}_0$ of the extracted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

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

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

The tolerance parameter $\gamma_\square$ thus allowed us to mitigate possible numerical error; in particular, if any given mesh point was slightly perturbed away from the underlying manifold, it could still end up being a part of the LCS. Finally, we looked at all of the surface elements pertaining to the triangulation of the manifold $\mathcal{M}$ (as described in section 3.9) in conjunction with the set of mesh points which had been recognized as belongning to the LCS $\mathcal{L}$. If mesh points corresponding to two of the three vertices defining a triangular surface element had been recognized as part of $\mathcal{L}$, we then added the mesh point corresponding to the last remaining vertex to the set of LCS points $\{\mathcal{L}_k\}$. Accordingly, the triangulations of the $\mathcal{M}$ were reused for $\mathcal{L}$. These slight relaxations of the LCS existence criteria facilitate the extraction of smoother LCS surfaces are favorable for the visual representation of LCSs. Moreover, they mitigate the possible effects of numerical error perturbing any given mesh point $\mathcal{M}_{i,j}$ away from the *actual* manifold — leaving it more than sufficiently close to the manifold for visualization purposes — by possibly allowing it to be included as part of the LCS after all. Figure 3.15 shows an example of extracting a repelling LCS from a computed

manifold.



(a) A computed manifold in its entirety

(b) The extracted repelling LCS

Figure 3.15: An example of a repelling LCS extracted as a subset of a computed manifold. In (a), a sample manifold for the steady ABC flow is shown, whereas (b) shows the subset of the manifold which satisfies the LCS existence criteria given in equation (2.28) (or lies sufficiently close to any point satisfying these criteria, meaning that their inclusion facilitates triangulations which provide an overall enhanced visual representation).

The extracted LCS surfaces $\mathcal{L}$, parametrized as a set of mesh points $\{\mathcal{L}_k\}$, represent three-dimensional surfaces which — allowing for a little numerical error — comply with all of the existence criteria for repelling LCSs, as originally proposed by Haller (2011). Inspired by the work of Farazmand and Haller (2012a), we then sought to dispose of the smallest among the computed LCSs, as these are expected to be the least significant in terms of the overall flow within the system. In order to obtain a measure of the size of our three-dimensional surfaces, to each LCS point $\mathcal{L}_k$, we assigned a weighting given by the surface area approximating the region of the underlying manifold $\mathcal{M}$ that is closer to the corresponding mesh point $\mathcal{M}_{i,j}$ than any others. That is,

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

where, as always, $\mathbf{x}_{i,j}$ denotes the coordinates of mesh point $\mathcal{M}_{i,j}$. This surface approximation is illustrated in figure 3.16. These weights were also used to compute a repulsion average $\overline{\lambda}_3$; in particular,

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

where the summation is over all mesh points in the parametrization of $\mathcal{L}$, and $\widetilde{\mathbf{x}}_k$ denotes the coordinates of mesh point $\mathcal{L}_k$. Any LCS for which the computed total weight $\mathcal{W}$ was smaller than some pre-set limit $\mathcal{W}_{\min}$ or $\overline{\lambda}_3 < 1$, the latter as a sanity check to ensure overall repulsion, per existence criterion (2.28a), were discarded.

3.12 Making the most of the available computational resources

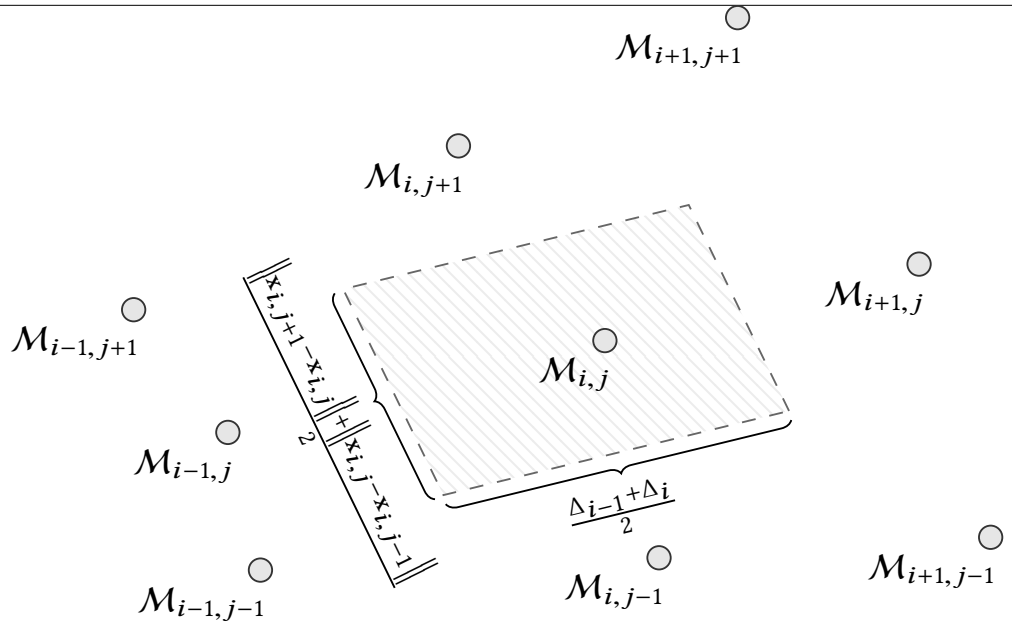


Figure 3.16: Our way of assigning weights to mesh points in computed LCSs. To a given LCS point $\mathcal{L}_k$, we assigned a weight given by a rectangular approximation of the surface area closer to the corresponding manifold mesh point $\mathcal{M}_{i,j}$ than all other point in the parametrization of the manifold $\mathcal{M}$ (shaded). Here, Δ_i corresponds to the interset step length used to create level set $\mathcal{M}_{i+1}$, based on level set $\mathcal{M}_i$, (see section 3.7), while $\mathbf{x}_{i,j}$ denotes the coordinates of mesh point $\mathcal{M}_{i,j}$.

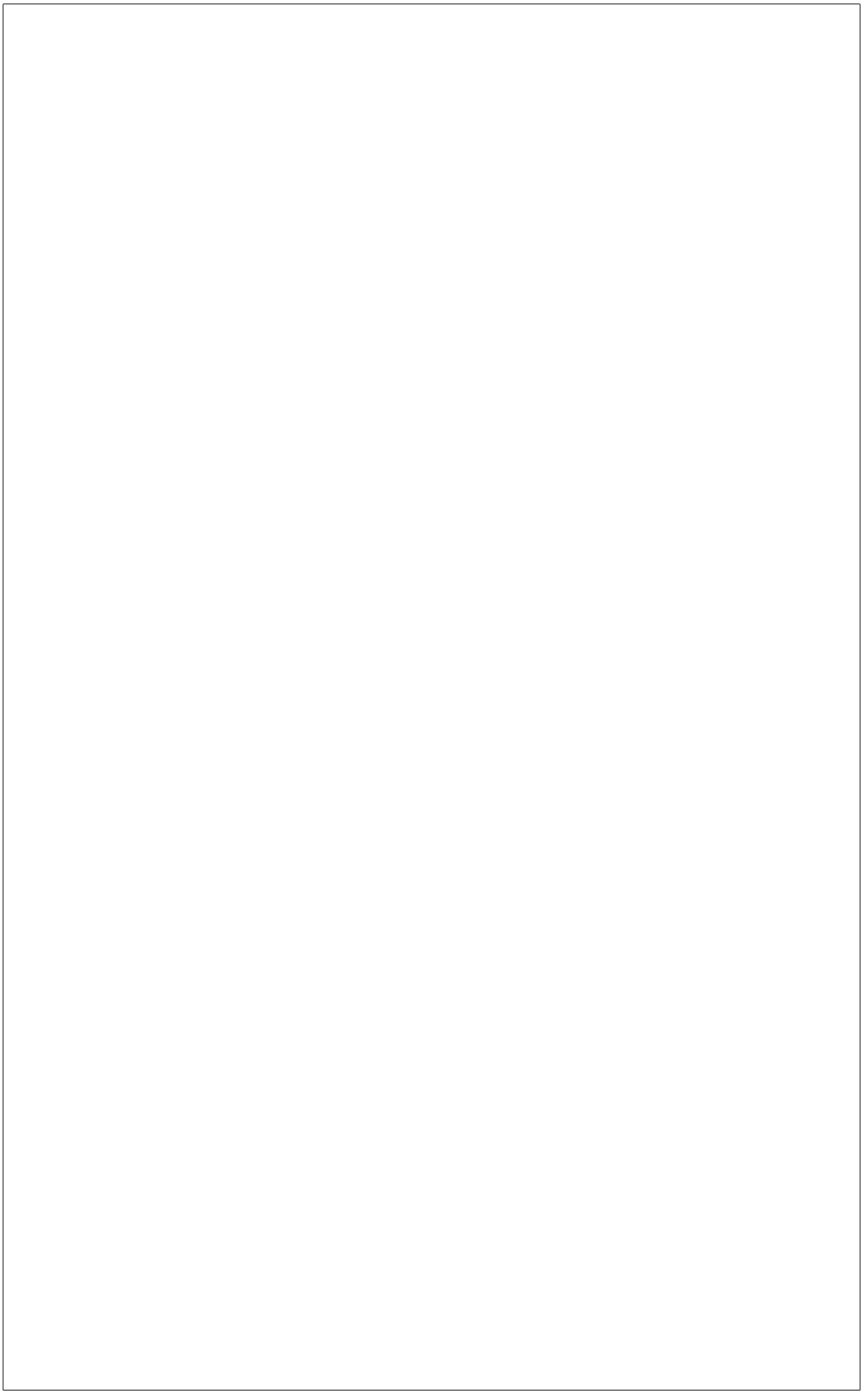
Poengter bruk av MPI for (triviell) parallellisering av mangfoldighetsgenerering, og videre Pythons `multiprocessing` for seleksjon.

'Kodeprofilering avslørte at ... var den store tidstyven, derfor ...'

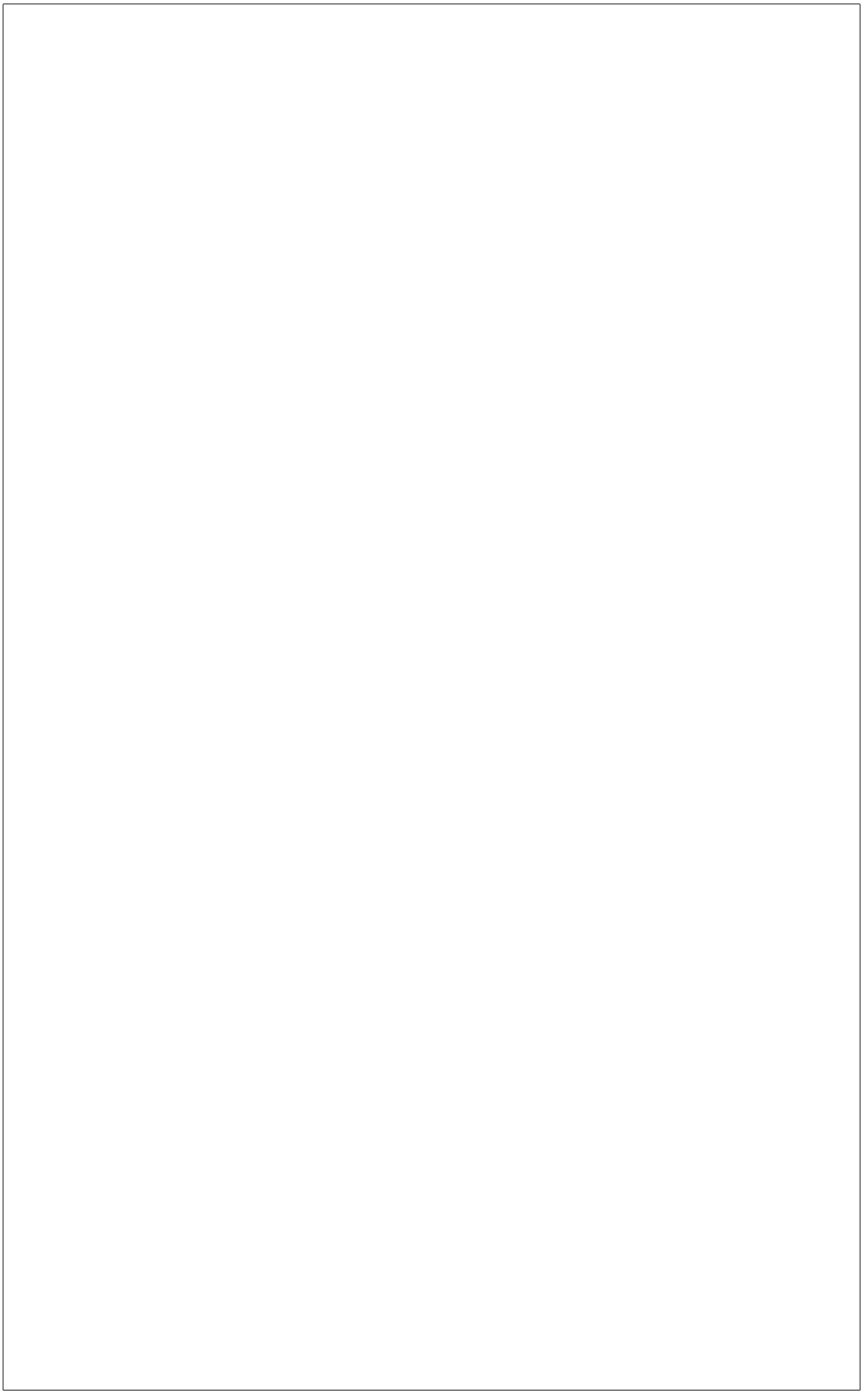
Understrek at alle numeriske rutiner som har med generering av nye punkter å gjøre, ble skrevet i Cython for å minimere kjøretid. Spesifikt benyttet vi høyt optimerte, lavnivå BLAS-rutiner når dette lot seg gjøre.

Likeså progget vi intersection-deteksjonsalgoritmen i samme språk, pga jevnt økende behov for sammenligninger når mangfoldigheten spredte seg utover.

Totalt sett medførte omskrivingen til Cython to størrelsesordeners reduksjon i kjøretid.



- ~~ABC-strømning (begge former), fjordstrømning (vist v/ kartprojeksjon(er)), oppsett av grids og beregning av λ, ξ v/ SVD~~
- ~~Interpolasjon av skalare- og vektorstørrelser~~
- Fullstendig beskrivelse av opprinnelig metode (lett modifisert GLS; tanvec, prevvec, levelsets etc.), muligens med flowchart
- Vis/beskriv eksplisitt noen av problemene/utfordringene med opprinnelig metode
  - Veldig mange frihetsgrader
  - (Tidvis) tilfeldighetspreget oppførsel i regioner med rasktvarierende ξ_3, hvor godt vi traff avh. av hvilket vinkeloffset vi traff med først (opp eller ned)
  - Langsom metode; Trenger gjerne mange forsøk om det først går galt
- Utfyllende meskrivelse av ny metode, med hovedfokus på forskjellene fra opprinnelig versjon. Flowchart?
  - Kan tillate os dette fordi vi har en ekstra frihetsgrad sml med system fra levelset-paper
  - Utvikling langs «strains» er en tredimensjonal videreføring av strainlines (siter Løken/Nordgreen 2017; F&H 2012)
  - Mer konsekvent oppførsel nær sterke diskontinuiteter, god konvergens
- Nytt av versjon 2 er kontinuerlige self-intersection-checks
  - Beskriv bruk av Möller-Trumbore osv.
- Seleksjon av lokalt sterkest repulsive materialoverflater v/ lokal sjekk
  - Sjekk hvert punkt på mangfoldigheten ift ABD
  - Behold også punkt som ikke er i ABD, så fremt de er nærme nok minst et annet punkt i ABD
  - Filtrer vekk LCS-kandidater som er for små



4 Results

4.1 Verifiying our method of generating manifolds

- Beskrivelse av tredimensjonal overflate og hvordan vi fant et normalvektorfelt

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

$$f(\mathbf{x}) = g(x, y) - z \tag{4.2}$$

$$\boldsymbol{\nabla} f(\mathbf{x}) = \begin{pmatrix} A\omega_x \cos(\omega_x x) \sin\left(\omega_y y\right) \\ A\omega_y \sin(\omega_x x) \cos\left(\omega_y y\right) \\ 1 \end{pmatrix} \tag{4.3}$$

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

- Tabell som viser *alle* parameterne vi brukte for å generer sinusfelt
- Ligning som beskriver hvordan RMS-feil ble beregnet

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

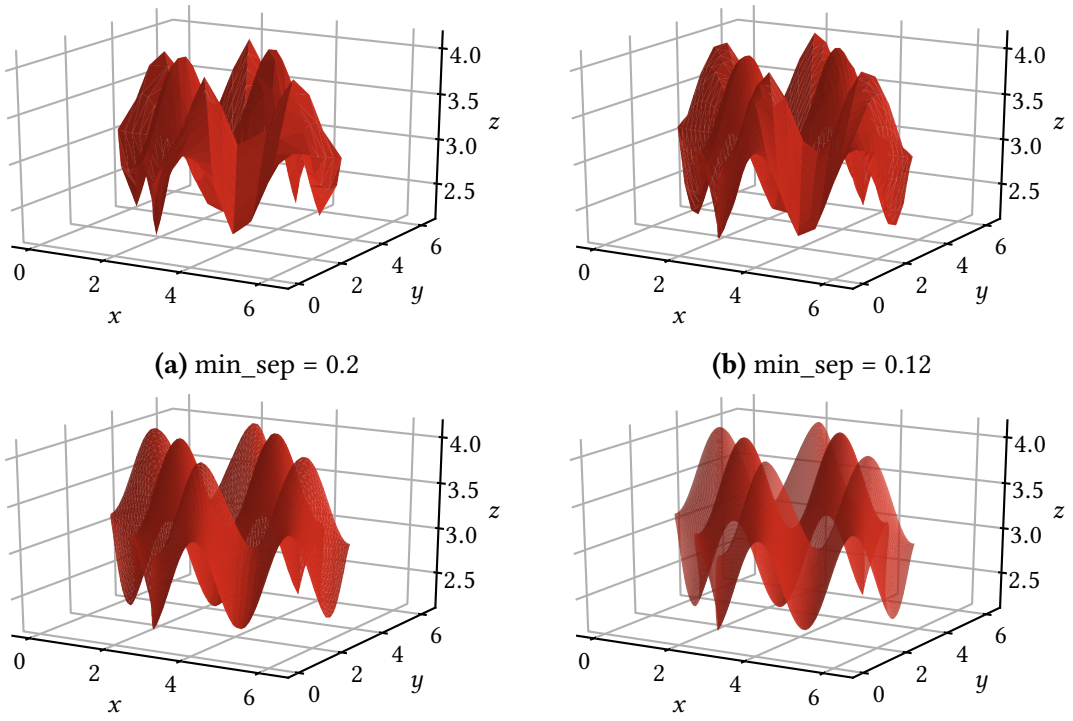
- Henvisning til tabell i avsnitt om ABC-strømning som viser *alle* parameterne vi brukte på ABC-feltene
- Henvisning til remark 1 om invarians for baner i vilkårlige lineærkombinasjoner av $\xi_{1,2}$.

4.2 Verification case for the extraction of repelling LCSs from manifolds

- Ligning som beskriver det gitte hastighetsfeltet

$$\dot{\mathbf{x}}(t) = \frac{\mathbf{x}}{\|\mathbf{x}\|} \sin(\pi(\|\mathbf{x}\| - 1)) \tag{4.5}$$

- Henvis til tabell som oppgir parametre for sinusfelted, i en ny tabell som inneholder alle de endrede parameterne
- Figur som viser hvorfor vi forventer å finne en LCS i $\|\mathbf{x}\| = 1$



(a) min_sep = 0.2

(b) min_sep = 0.12

(c) min_sep = 0.04

(d) min_sep = 0.01

Figure 4.1: Manifolds generated from analytically determined vector field.

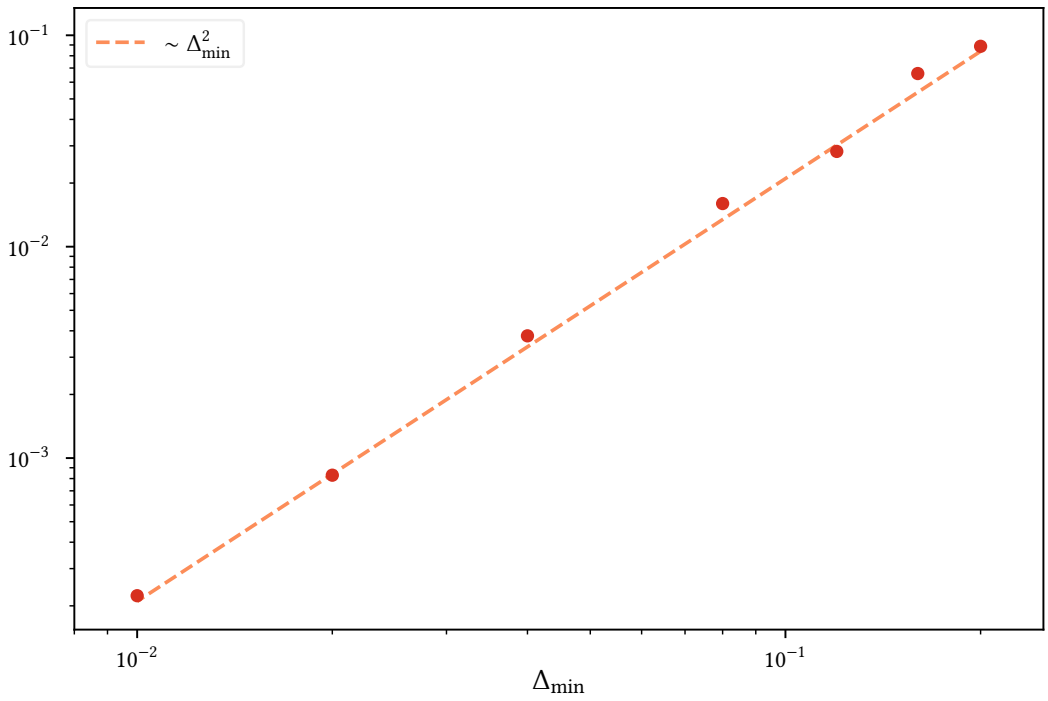
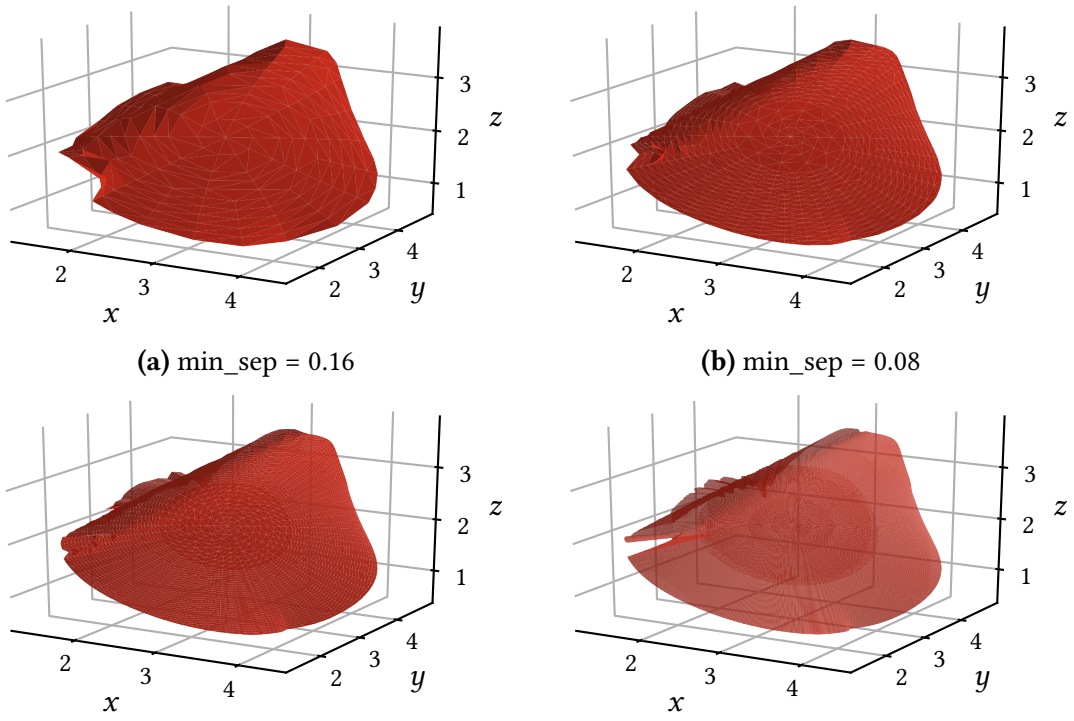


Figure 4.2: RMS error of sinusoidal manifolds



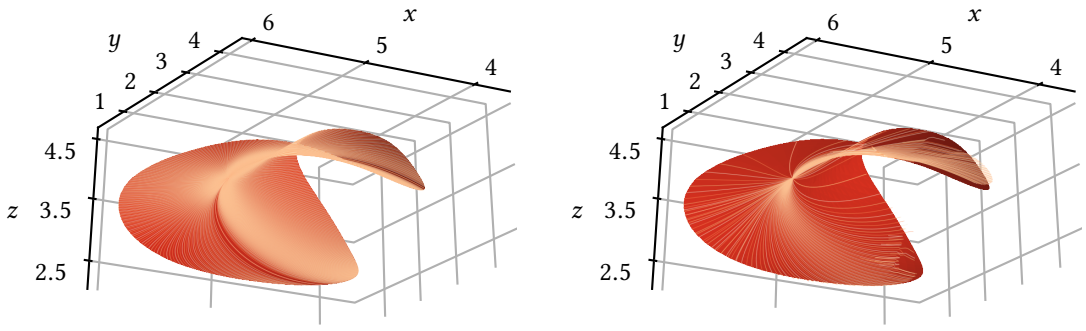
(a) min_sep = 0.16

(b) min_sep = 0.08

(c) min_sep = 0.04

(d) min_sep = 0.01

Figure 4.3: Numerical convergence as min_sep decreases



(a) Trajectories forced radially outwards

(b) Pure linear combinations of ξ_1 and ξ_2

Figure 4.4: Trajectories with local directionality given by linear cominations of ξ_1 and ξ_2.

- Figur som viser LCSen vi fant
- Beskrivelse av blobtest for å verifisere at vi finner *frastøtente* LCSer

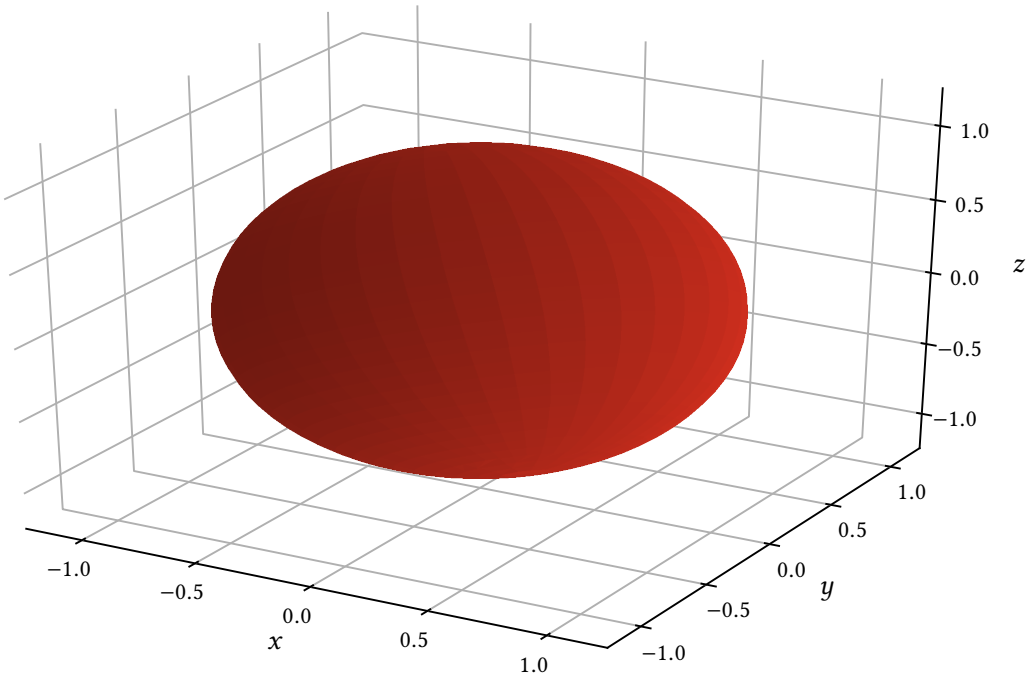


Figure 4.5: The identified LCS for the spherical flow

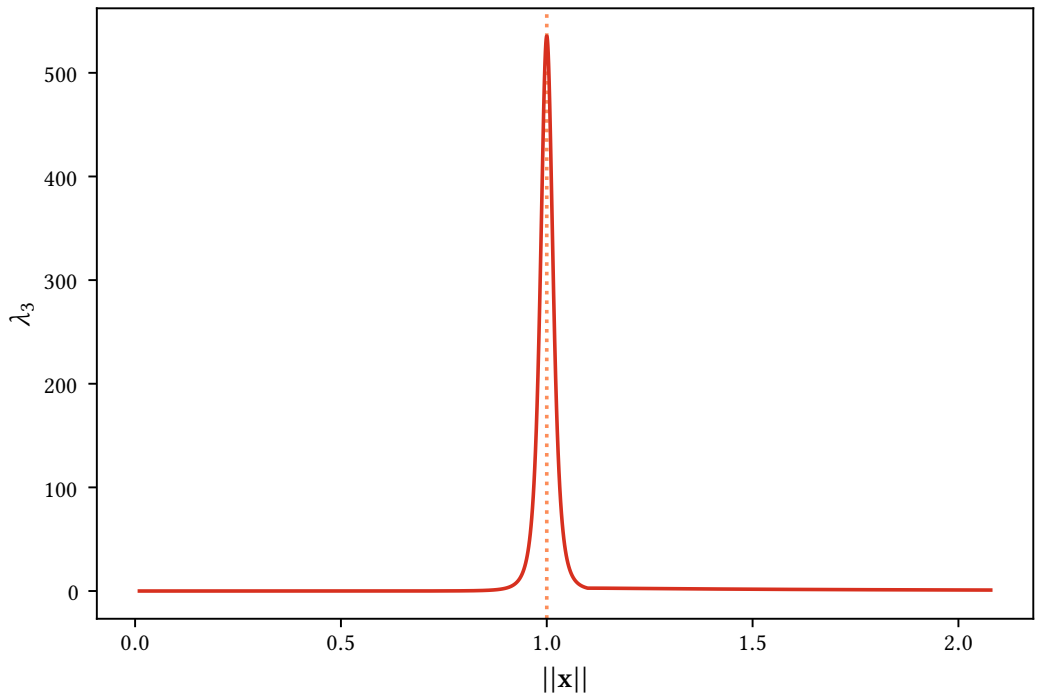
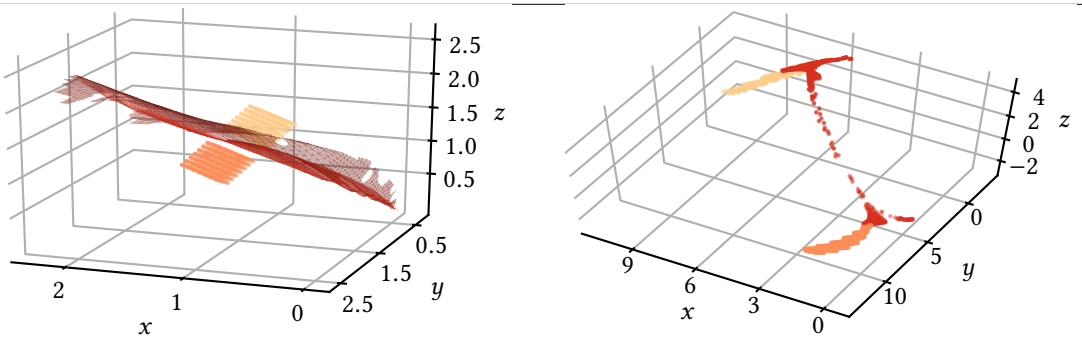


Figure 4.6: λ_3 as a function of radius, for the spherical flow. Displayed here is the arithmetic average across all solid angles.



(a) $t = 0$ **(b)** $t = 5$

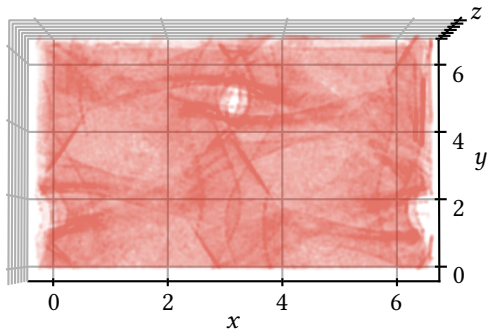
Figure 4.7: Blob test for verifying expected LCS behaviour

4.3 Computed LCSs in the ABC flow

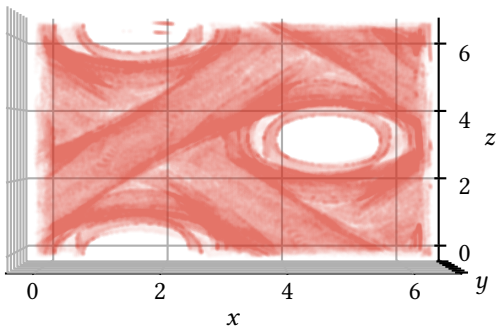
- Henvis til tidligere tabeller i ny tabell som oppgir alle nye parameterverdier — spesielt minste vekting og toleranseparameter for aksept av punkt
- Henvis til tabell som oppgir antall punkter i $\mathcal{U}_0$ (opprinnelig så vel som redusert) (table 3.2)
- Tydeliggjør tidlig at samme innsynsvinkler er benyttet
- Understrek hvordan både LCSer og $\mathcal{U}_0$-domener ligner på hverandre — vi kjenner igjen de to tilfellene som varianter av den samme underliggende strømningen

4.4 Computed LCSs in the Førde fjord

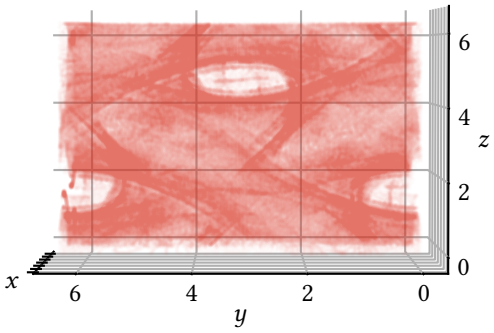
- Lag tabell som oppgir alle parameterverdier
- Henvis til tabell som oppgir antall punkter i $\mathcal{U}_0$ (opprinnelig så vel som redusert) (table 3.2)
- Understrek at vi bruker samme innsynsvinkler på domener, så vel som LCSer, som for ABC-strømning
- Påpek at de beregnede LCSene i stor grad er orientert i distinkte horisontalplan, som indikerer at det er størst strekk i dybderetningen



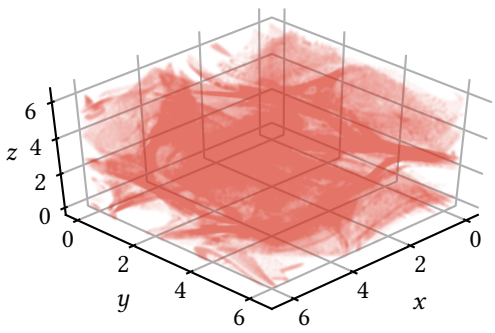
(a) View along the negative z-axis



(b) View along the positive y-axis

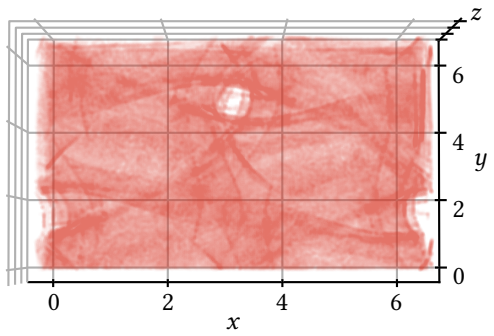


(c) View along the positive x-axis

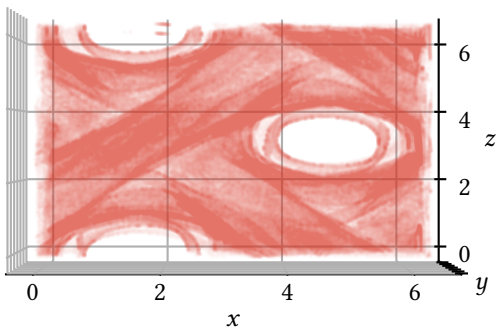


(d) Approximately isometric view

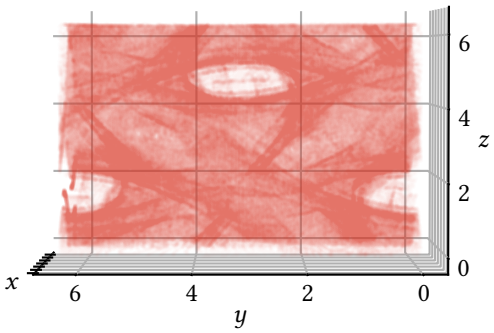
Figure 4.8: ABD domain obtained for stationary flow. $\epsilon = 0.005$.



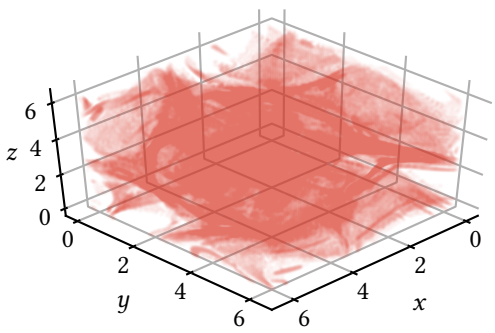
(a) View along the negative z-axis



(b) View along the positive y-axis

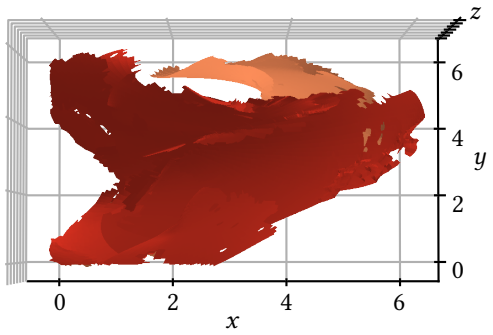


(c) View along the positive x-axis

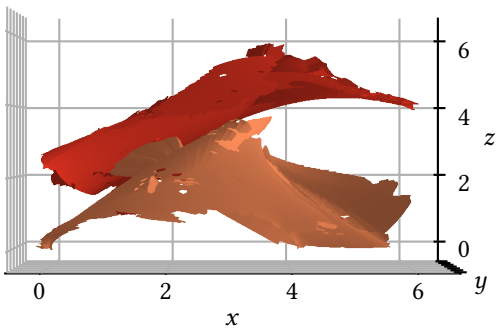


(d) Approximately isometric view

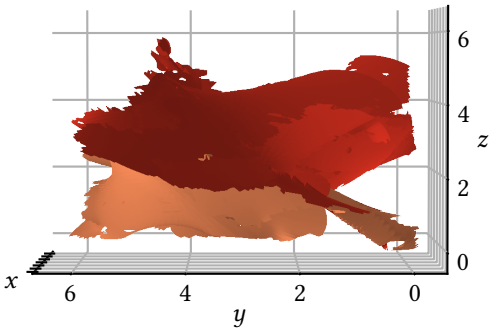
Figure 4.9: ABD domain obtained for dynamic flow. $\epsilon = 0.005$.



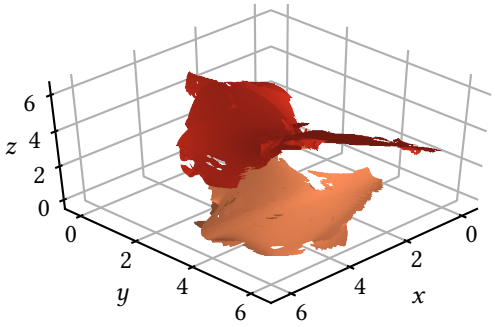
(a) View along the negative z-axis



(b) View along the positive y-axis

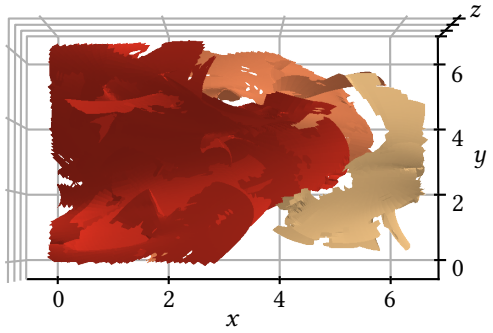


(c) View along the positive x-axis

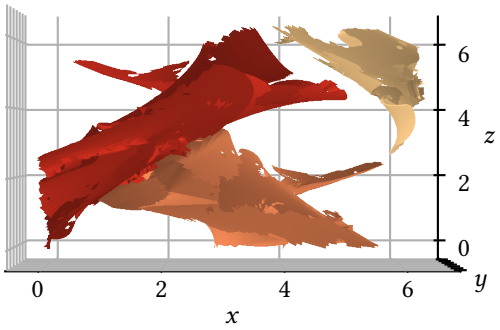


(d) Approximately isometric view

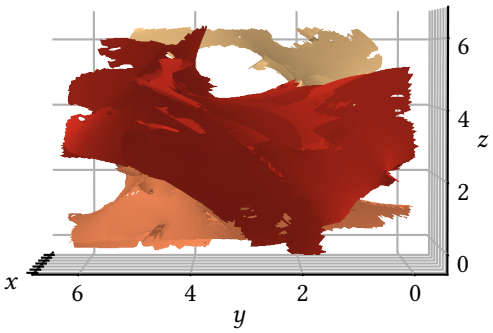
Figure 4.10: LCSs obtained for stationary ABC flow



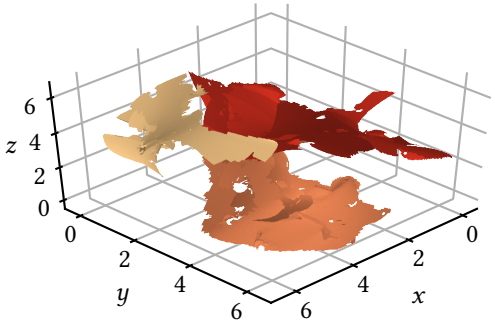
(a) View along the negative z-axis



(b) View along the positive y-axis

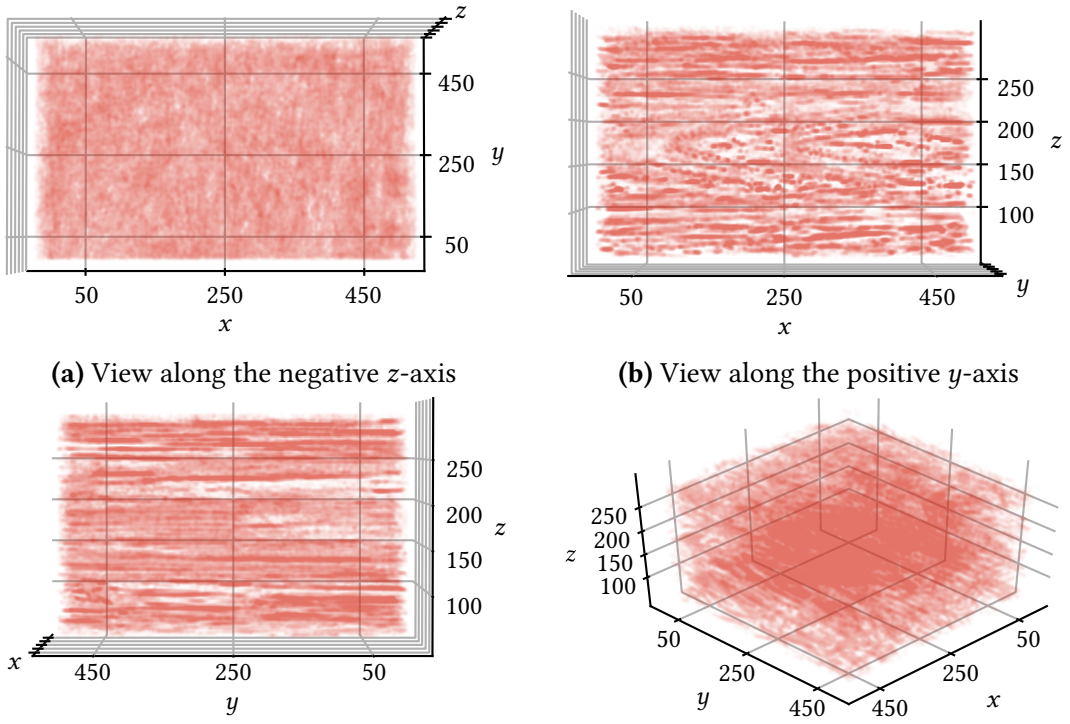


(c) View along the positive x-axis



(d) Approximately isometric view

Figure 4.11: LCSs obtained for dynamic ABC flow



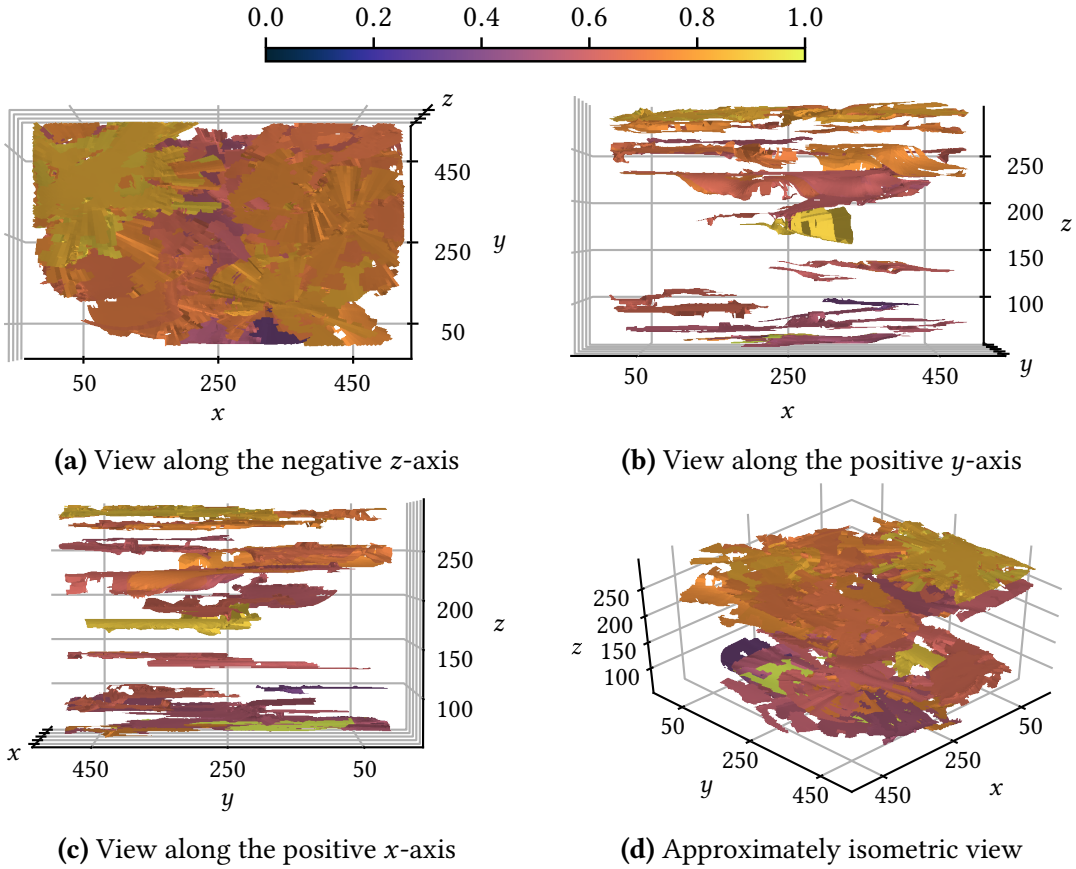
(a) View along the negative z-axis

(b) View along the positive y-axis

(c) View along the positive x-axis

(d) Approximately isometric view

Figure 4.12: ABD domain obtained for gridded data. The axes have dimension meters. z = 0 corresponds to the sea level, increasing downwards. $\epsilon = 0.1$.



(a) View along the negative z-axis

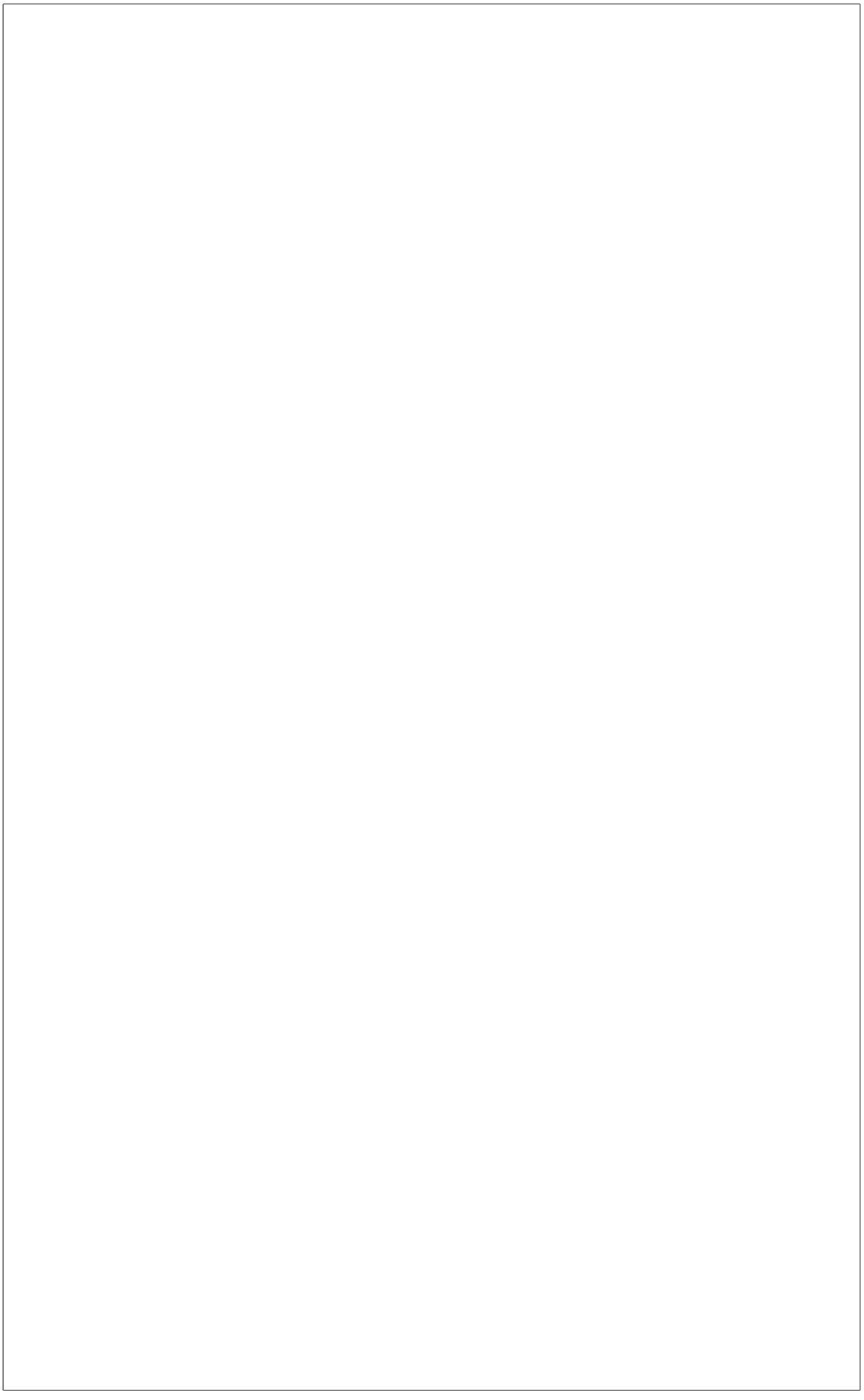
(b) View along the positive y-axis

(c) View along the positive x-axis

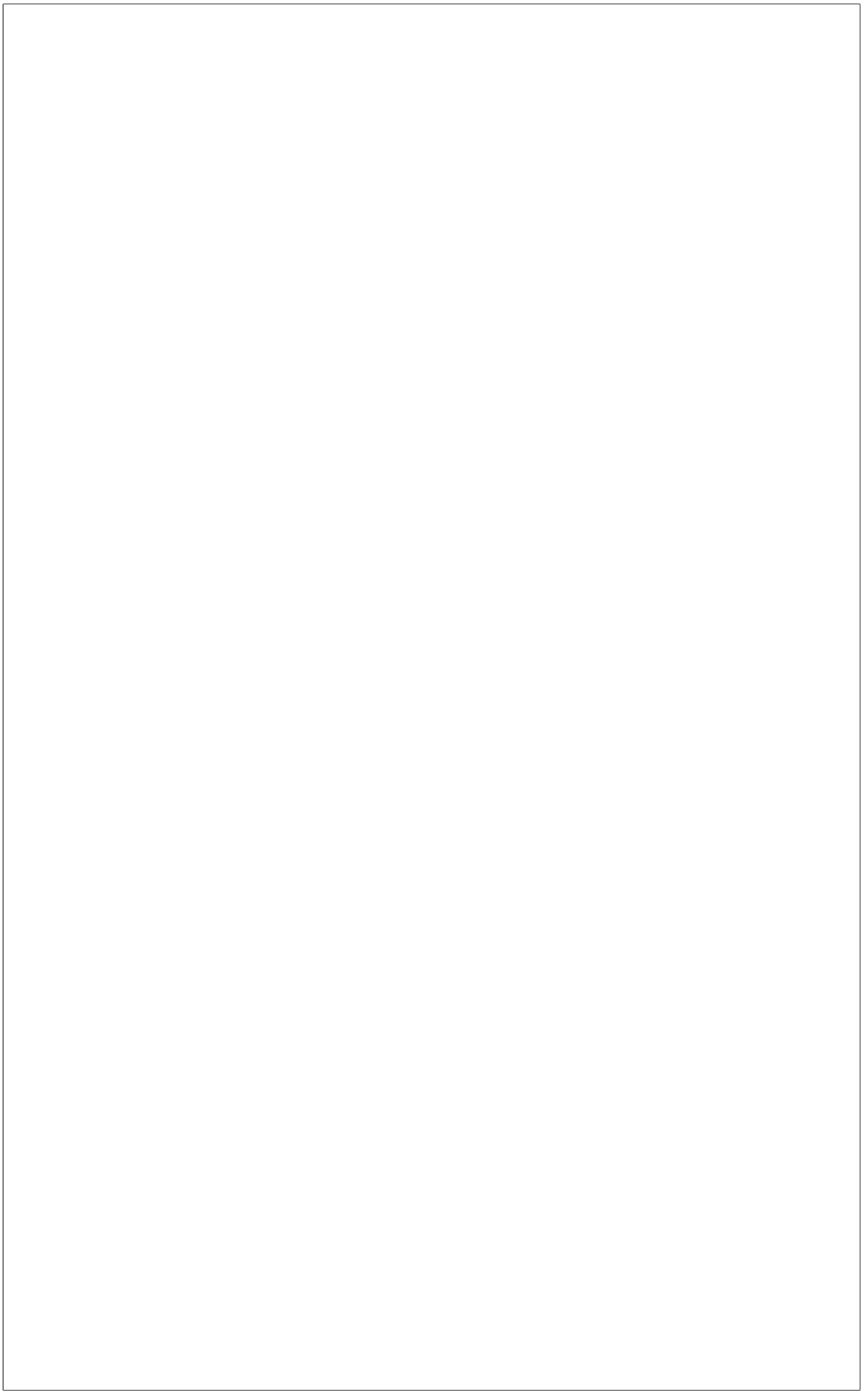
(d) Approximately isometric view

Figure 4.13: LCSs obtained for flow in the Førde fjord

5 Discussion



6 Conclusions



References

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

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

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

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

Frisch, U. (1995). *Turbulence, the legacy of A.N. Kolmogorov*. 1st ed. Cambridge University Press. ISBN: 0-521-45103-5.

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

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

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

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

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

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

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

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

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

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

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

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

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

Williams, J. (2018). *Bspline-Fortran: Multidimensional B-Spline Interpolation of Data on a Regular Grid*. Zenodo. DOI: 10.5281/zenodo.1215290. Open Access Software.

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

A Appendix A