An Algorithmic Introduction to Lagrangian Coherent Structures

Kristjan Onu Florian Huhn George Haller

January 24, 2014

Contents

Abstract

We give an applied introduction to Lagrangian coherent structures (LCSs.) We review recent results in variational LCS theory, present algorithms based on the theory and implement the algorithms in a computational engine called LCS Tool. We use LCS Tool to analyse flow examples.

1 Introduction

LCSs are used in fluid dynamics to capture essential flow features and thereby simplify the description of fluid flows. More specifically, LCSs

have been particularly well adapted to analyze two dimensional finite time turbulent flows. LCSs capture dominant transport barriers, repelling and attracting structures and LCSs are used to identify vortex boundaries.

LCSs have been used to analyse oceanic and atmospheric flows, biological applications[1, 2], aeronautics[3] and mechanical systems[4].

The finite-time Lyapunov exponent (FTLE) gives a mathematical definition for LCSs. Although the FTLE has been applied to a variety a situations, it is possible to find examples where it fails to identify LCSs correctly, giving false positives and false negatives[5, 6]. To remedy these problems, variational principles have been used to present new approaches to identifying LCSs[5, 7, 8, 9, 10].

This article presents an algorithmic introduction LCSs using a computational engine for LCSs called LCS Tool[1].

LCS Tool is a library of functions that perform the computations to identify LCSs in two dimensional finite time flows. LCS Tool aims to initiate scientists to LCSs. Existing LCS software programs are:

- *MANGEN*[11], calculates stability manifolds adaptively in two dimensional finite time velocity fields. Includes a graphical user interface and uses MPI for parallel calculations.
- *LCS MATLAB Kit*[12] calculates the FTLE from velocity datasets. Includes a graphical user interface.
- *Newman*[13] calculates the FTLE in N dimensions. Assists ridge extraction of FTLE fields. Supports analytic and dataset defined velocity definitions.
- *FlowVC*[14] is a general purpose LCS platform for two and three dimensional datasets. Parallel calculations are supported with OpenMP, CUDA and OpenCL.

These packages are based on the FTLE; the objective of this article and LCS Tool is to replace FTLE methods.

2 Theory

The fluid flows considered are defined in two dimensions by a velocity field:

$$\dot{\boldsymbol{x}} = \boldsymbol{v}(\boldsymbol{x}, t), \qquad \boldsymbol{x} \in U \subset \mathbf{R}^2, \qquad t \in [t_-, t_+].$$

Integration yields the flow map, defined between an initial time t_0 and a final time t within $[t_-, t_+]$:

$$F_{t_0}^{t}(\boldsymbol{x}_0) := \boldsymbol{x}(t, t_0, \boldsymbol{x}_0).$$

The time interval used to detect LCSs is left as a choice; it can be the complete interval for which the flow is defined, or data is available, or it can be a chosen subset.

[1]LCS Tool is free MATLAB software available for download at: https://github.com/jeixav/LCS-Tool

The right Cauchy-Green strain tensor measures Lagrangian strain in the velocity field and is defined:

$$C_{t_0}^t(\boldsymbol{x}_0) = \left[\nabla F_{t_0}^t(\boldsymbol{x}_0)\right]^T \nabla F_{t_0}^t(\boldsymbol{x}_0).$$

The eigenvalues of the Cauchy-Green strain tensor are denoted by $\lambda_{1,2}$ and the eigenvectors by $\boldsymbol{\xi}_{1,2}$. The eigenvalues satisfy $0 < \lambda_1 \leq \lambda_2$ and the eigenvectors satisfy $\boldsymbol{\xi}_1 \perp \boldsymbol{\xi}_2$.

2.1 Elliptic LCS

We find coherent Lagrangian vortices as closed orbits of one of the two vector fields

$$\boldsymbol{\eta}_{\pm}^{\lambda} = \sqrt{\frac{\lambda_2 - \lambda^2}{\lambda_2 - \lambda_1}}\boldsymbol{\xi}_1 \pm \sqrt{\frac{\lambda^2 - \lambda_1}{\lambda_2 - \lambda_1}}\boldsymbol{\xi}_2, \tag{2.1}$$

which are a function of the eigenvalues and eigenvectors of $C_{t_0}^t(\boldsymbol{x}_0)$. Curves γ that are trajectories of $\boldsymbol{\eta}_{\pm}^{\lambda}$ can be parametrized as $\boldsymbol{r}(s)$ with their arclength $s \in [0, \sigma]$. They are given by integrating

$$\boldsymbol{r}' \equiv \frac{d\boldsymbol{r}}{ds} = \boldsymbol{\eta}_{\pm}^{\lambda}(\boldsymbol{r}). \tag{2.2}$$

λ is a free parameter close to unity that indicates the uniform stretching of the curve γ, i.e., each tangent vector $\boldsymbol{r}'(s)$ with length l_{t_0} stretches by a factor λ to length l_t under the flow map:

$$\begin{aligned} l_{t_0}(s) &= \sqrt{\langle \boldsymbol{r}'(s), \boldsymbol{r}'(s)\rangle}, \\ l_t(s) &= \sqrt{\langle \boldsymbol{r}'(s), C_{t_0}^t(\boldsymbol{r}(s))\boldsymbol{r}'(s)\rangle}, \\ l_t(s) &= \lambda\, l_{t_0}(s). \end{aligned}$$

For $\lambda = 1$, $\boldsymbol{\eta}_{\pm}^{\lambda}$ points in the direction of maximal Lagrangian shear and we obtain uniformly non-stretching curves over the finite time interval $[t_0, t]$. In summary, the boundaries of coherent Lagrangian vortices are closed curves that uniformly stretch by a factor λ close to unity when advected by the flow. We call them lambda-lines. The outermost lambda-line is the physically observable boundary of an eddy, generating the tracer pattern of an isolated rotating fluid region.

Note that $\boldsymbol{\eta}_{\pm}^{\lambda}$ follows from the variational principle that the first variation of the averaged Lagrangian strain along the searched closed curve vanishes (see [10] for full derivations):

$$Q(\gamma) = \frac{1}{\sigma}\int_0^{\sigma} \frac{l_t(s)}{l_{t_0}(s)}\mathrm{d}s \tag{2.3}$$

$$\delta Q(\gamma) = 0. \tag{2.4}$$

This coherence principle means that the searched curve γ, the coherent eddy boundary, is a stationary point of the averaged Lagrangian strain functional and therefore nearby fluid elements cannot break away from the curve γ.

2.2 Hyperbolic LCS

Hyperbolic LCS are trajectories of the two eigenvector fields $\boldsymbol{\xi}_{1,2}$ of the Cauchy-Green strain tensor $C_{t_0}^t$. Repelling hyperbolic LCSs, denoted as strainlines, are obtained as solutions of the differential equation

$$\boldsymbol{r}_1' = \boldsymbol{\xi}_1(\boldsymbol{r}),$$

and attracting hyperbolic LCSs, denoted as stretchlines, are obtained as a solution of the differential equation

$$\boldsymbol{r}_2' = \boldsymbol{\xi}_2(\boldsymbol{r}).$$

A normal perturbation to the strainline segment $\boldsymbol{r}_1'$ grows under the flow map by a factor λ_2, i.e., fluid elements are repelled by that rate in normal direction. Equivalently, a normal perturbation to the stretchline segment $\boldsymbol{r}_2'$ shrinks with factor λ_1 and fluid elements are attracted.

Similarly to ellitpic LCS, hyperbolic LCS are also based on a variational principle, in this case for the Lagrangian shear

$$p(s) = \frac{\langle \boldsymbol{r}'(s), D(\boldsymbol{r}(s))\boldsymbol{r}'(s)\rangle}{\sqrt{\langle \boldsymbol{r}'(s), C(\boldsymbol{r}(s))\boldsymbol{r}'(s)\rangle \langle \boldsymbol{r}'(s), \boldsymbol{r}'(s)\rangle}}$$

where

$$D(\boldsymbol{x}_0) = \frac{1}{2}[C(\boldsymbol{x}_0)\Omega - \Omega C(\boldsymbol{x}_0)], \quad \Omega = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

The first variation of the average Lagrangian shear along a hyperbolic LCS vanishes

$$P(\gamma) = \frac{1}{\sigma}\int_0^\sigma p(s)\mathrm{d}s.$$
$$\delta P(\gamma) = 0.$$

They are one type of shearless LCS. For details of the derivation of hyperbolic LCS, see [15].

1. Setup a Cartesian grid and auxiliary grid in the (rectangular) domain of the flow.
2. Integrate the flow velocity equations at each grid point and auxiliary grid point. This yields the flow map, $F_{t_0}^t(\boldsymbol{x}_0)$.
3. Use finite differences to approximate the derivative of the flow map, $DF_{t_0}^t(\boldsymbol{x}_0)$.
4. Compute the Cauchy-Green strain tensor, $C_{t_0}^t = \left(DF_{t_0}^t\right)^T DF_{t_0}^t$ and its eigenvalues, $\lambda_{1,2}$ and eigenvectors, $\xi_{1,2}$.

Table 3.1: Algorithm to calculate the Cauchy-Green strain tensor eigenvalues and eigenvectors

3 Methods

This section presents the numerical methods used to compute LCS and explains them step by step. In other words, the algorithms that constitute LCS Tool are presented.

3.1 The Cauchy-Green strain tensor

To obtain Lagrangian coherent structures the Cauchy-Green strain tensor must be calculated first. More specifically, the eigenvalues and eigenvectors of the Cauchy-Green strain tensor are needed. The function associated with this calculation in LCS Tool is eig_cgStrain. The main steps of this function are enumerated in Table 3.1 and are described in detail in the following part. Table 3.2 summarises the function syntax.

We setup a Cartesian grid according to user-defined resolutions. We can find the optimal resolution by an iterative procedure in which we start with a low resolution and double it until convergence of LCSs is observed visually. If the chosen domain only comprises few expected LCS, e.g., one vortex, a resolution of about 500 grid point along the longest axis usually gives good results. Otherwise, for larger domains, a higher resolution must be chosen. Furthermore an auxiliary grid is setup. This is illustrated in Figure 3.1. The improvements the auxiliary grid yields compared to using only the main grid were reported in Farazmand and Haller [7]. Experience indicates that optimal results are obtained when the auxiliary grid spacing is between 1% and 10% of the main grid spacing.

The function eig_cgStrain provides the option to calculate the eigenvector from the auxiliary grid, but the eigenvalues from the main grid. We have found that for flows defined analytically the eigenvalues should be calculated from the main grid whereas for the flows defined by datasets, using the auxiliary grid gives better results.

As stated, typically we set the main grid resolution of the Cauchy-Green strain tensor to around 500 grid points. For a square domain, this implies the velocity equations must be integrated, after including 4 auxiliary grid points around every main grid point, at 1.25 million grid points.

[cgEigenvector,cgEigenvalue] = eig_cgStrain(derivative,domain,timespan,resolution)	
derivative	function handle for flow velocity equations
domain	2×2 array to define flow domain
timespan	1×2 array to define flow timespan
resolution	1×2 array to define Cauchy-Green strain main grid resolution
auxGridRelDelta	optional scalar between 0 and 0.5 to specify auxiliary grid spacing. Default is 10^{-2}.
eigenvalueFromMainGrid	optional logical to control whether eigenvalues of Cauchy-Green strain are calculated from main or auxiliary grid. Default is true.
incompressible	optional logical to specify if incompressibility is imposed. Default is false.
odeSolverOptions	optional odeset structure to specify flow map integration options

Table 3.2: Syntax of the function eig_cgStrain

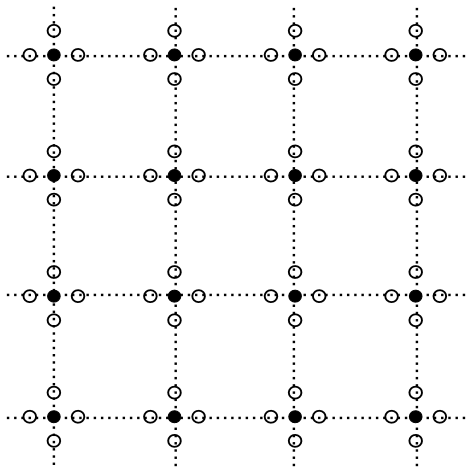

Figure 3.1: Illustration of main grid (filled circles) and auxiliary grid (empty circles) used to compute the Cauchy-Green strain. Solid circles are main grid points and hollow circles are auxiliary grid points. The variable auxGridRelDelta specifies the grid spacing of the auxiliary grid relative to the main grid spacing.

The "natural" way to perform these integration calculations would be to solve a two dimensional first-order ODE system at every grid point and iterate over every grid point within a for-loop. With 1.25 million grid points this calculation can take an excessively long time. To alleviate this problem, we solve the flow equations in a vector form where the equations at all grid points are combined into a single system of equations, even though the system is composed of independent blocks of two dimensional first-order ODE systems. MATLAB's ode45 function is used to perform the integration and the flow integration typically takes five to ten minutes with default error tolerances. The drawback of vector form integration is its memory requirements can become excessive at high resolutions. Furthermore, writing the velocity function in vector form is more error-prone than the intrinsic two dimensional form.

The Cauchy-Green strain tensor can have singularities, i.e., $C_{t_0}^t(\boldsymbol{x}_0) = I$. Such points are isolated in general[16], therefore no special accommodations for are made in computations for singularities of the Cauchy-Green strain tensor.

3.1.1 Special case: incompressible velocity fields

Flows that are incompressible, meaning $\nabla \cdot \boldsymbol{v} = 0$, satisfy the relation $\lambda_1(\boldsymbol{x}_0)\lambda_2(\boldsymbol{x}_0) = 1, \forall\, \boldsymbol{x}_0 \in U$[17]. This gives the possibility of imposing incompressibility when calculating the eigenvalues of the Cauchy-Green strain tensor by first calculating λ_2 and then setting $\lambda_1(\boldsymbol{x}_0) = 1/\lambda_2(\boldsymbol{x}_0)$. This procedure should be applied carefully since by changing λ_1 but not ξ_1, the error in satisfying the eigenvalue equation for the Cauchy-Green strain tensor may increase.

At some grid points it may occur that $\lambda_2 < 1$ due to numerical integration errors of the flow velocity equations. By setting integration tolerances to smaller values the number of grid points with $\lambda_2 < 1$ can be reduced. Nonetheless, some grid points can be so close to singularities of the Cauchy-Green strain tensor that satisfying $\lambda_2 \geq 1$ everywhere could be excessively computationally costly. Therefore the function eig_cgStrain records the number of points at which incompressibility cannot be imposed and this measure can also help to set appropriate integration tolerances.

3.1.2 Special case: data-defined velocity fields

Velocity fields defined by datasets require pre-processing prior to performing the integration that yields the flow map. An interpolation function must be written to enable evaluating the velocity function at arbitrary points in U and at arbitrary times between t_- and t_+. MATLAB's interpolation functions can be integrated into the velocity function. As with analytic cases, care should be taken to ensure the interpolation embedded in the flow velocity integration function works correctly when performing integration in vector form. In section 4.3, we present an example how LCS are computed for an ocean dataset and details of the interpolation function are given.

Beyond the pre-processing needed to obtain the Cauchy-Green strain tensor eigenvalues and eigenvectors, no special manipulations are needed

1. Position Poincare sections in flow domain as initial positions of lambda-lines
2. Integrate lambda-lines tangent to $\boldsymbol{\eta}_{\pm}$
3. Calculate Poincare return map
4. Find closed orbits in Poincare return map
5. Identify outermost closed orbit

Table 3.3: Algorithm to calculate Shear LCS

to treat data-defined flow. It may be however, that parameter values giving good results with analytically defined flows need adjustment prior to processing data-defined velocity fields.

3.2 Shear LCS

Shear LCSs, or coherent Lagrangian vortices, are closed orbits in the Lagrangian shear vector fields $\boldsymbol{\eta}_{\pm}$ defined in Equation (2.2). Since $\lambda = 1$, the closed curves are non-stretching and area preserving. We find closed orbits by integrating lambda-lines starting from a Poincare section and evaluating the first return map. A lambda-line closing onto itself in the starting point is a shear LCS. The main steps of the algorithm used to calculate shear LCS are enumerated in Table 3.3 and described in detail in the following part. The syntax of shear LCS functions in LCS Tool can be seen in Table 3.4.

The first step is to set the position of Poincare sections poincareSection.endPosition that are straight lines in regions where closed lambda-lines are expected. The Poincare section must be oriented in a way that the first endpoint is close to the centre of the expected eddy and the second endpoint is outside the expected eddy region. There is no predetermined procedure for deciding where the Poincare sections should be positioned. Possibilities are an examination of the finite-time Lyapunov exponent field of the flow, or some prior knowledge of where coherent eddies could be expected. Additionally, the number of lambda-lines launched from the Poincare section poincareSection.numPoints must be defined. A good default value is 100.

The second step is to integrate the lambda-lines to obtain the Poincare return map. Integration of lambda-lines is performed with the η_+ and η_- vector fields defined on the main grid. These vector fields are derived from eigenvalues and eigenvectors as described in Farazmand and Haller [7]. The eigenvector fields have orientation discontinuities, therefore the vector field must be locally reoriented for the integration. The algorithm used to perform variable step-size integration in vector fields with orientation discontinuities is enumerated in Table 3.5 and illustrated in Figure 3.2. Linear interpolation is used when interpolating $\boldsymbol{\eta}_{\pm}$ within a grid element, since using higher order interpolation would necessitate verifying that there are no orientation discontinuities at more than only the four nearest grid points. Detection of orientation discontinuities is

[shearline.etaPos,shearline.etaNeg] = lambda_line(cgEigenvector,cgEigenvalue, lambda)	
cgEigenvector	array of Cauchy-Green strain eigenvectors
cgEigenvalue	array of Cauchy-Green strain eigenvalues
lambda	scalar lambda value in Equation (2.1)
[closedOrbits,orbits] = poincare_closed_orbit_multi(domain,resolution,shearline, PSList)	
domain	array to define flow domain
resolution	1×2 array to define Cauchy-Green strain main grid resolution
shearline	structure of arrays of η_+ and η_- values on main grid
PSList	structure for Poincare section end-points, number of lambda-lines launched from Poincare section, and maximum closed lambda-line length
nBisection	optional positive integer. Default is 5. Number of bisection steps to refine zero crossings of Poincare return map.
dThresh	optional scalar. Default is 10^{-2}. Threshold to discard discontinuous zero crossings of Poincare return map.
odeSolverOptions	optional odeset structure to specify lambda-line integration options
periodicBc	optional 1×2 logical array to specify periodic boundary conditions. Default is [false,false].

Table 3.4: LCS Tool shear LCS functions syntax

1. Linearly interpolate vector field orientation at initial position
2. At next position, check whether vector field has rotated by over 90°, if yes, flip the vector field orientation by 180°.
3. Stop integration when lambda-line returns to Poincare section, lambda-line hits the domain boundary, or maximum integration length has been reached.

Table 3.5: Algorithm used for variable time step integration of lambda-lines.

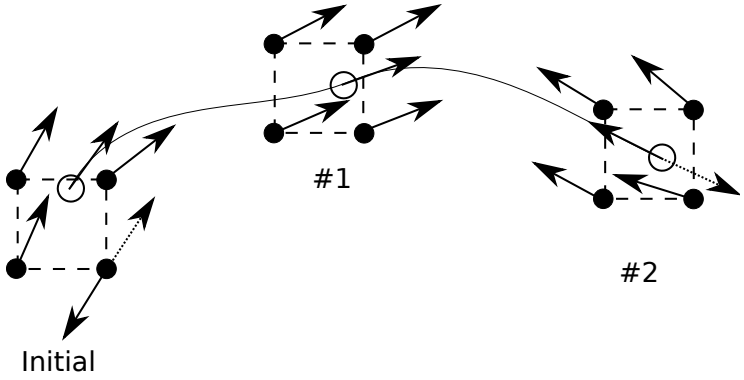

Figure 3.2: Schematic illustration of lambda-line variable time step integration. At the initial point, there is an orientation discontinuity at the lower right grid point that must be corrected prior to linear interpolation. At point #1, no orientation discontinuities are present. At point #2 the interpolated $\eta_{\pm}$ vector must be rotated by 180° to match the orientation of the trajectory.

achieved by checking the inner product of the vector fields at adjacent grid points. Then, rotations exceeding 90° are classified as orientation discontinuities and are corrected before linear interpolation. Thus when setting the Cauchy-Green strain tensor main grid resolution, a histogram of eigenvector field rotations may be produced to ensure that all rotations are below 90° or close to 180°.

An example of a Poincare return map produced from integration of the $\boldsymbol{\eta}_{\pm}$ field is shown in Figure 3.3. Well behaved orbits will return to the Poincare section and their integration will be stopped using an ODE event detection function. Some orbits may however deviate far from the Poincare section and their integration could take inordinately long. To prevent this behaviour, a maximum orbit length, poincareSection.orbitMaxLength , is specified. In practice, viewing the Poincare section as the radius of a circle and setting the maximum lambda-line integration length to twice the circumference gives acceptable results. In Figure 3.3, circle markers indicate closed orbit positions. Closed orbits exist where the Poincare return map, the distance between the final and the initial point of the orbit

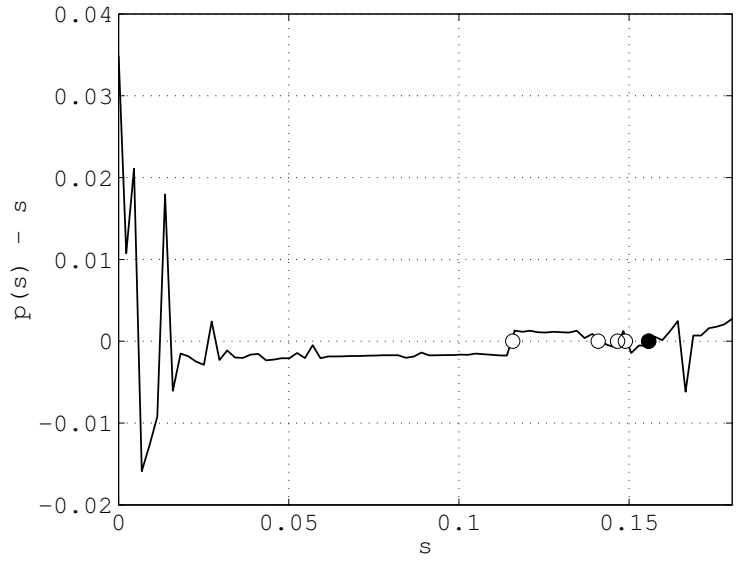

Figure 3.3: Example of a Poincare return map obtained for shear LCS. Circle markers indicate closed orbit positions. The filled circle indicates the outermost closed orbit and shear LCS.

$P(s) - s$, is zero. The function poincare_closed_orbit performs the computations. In Figure 3.3 not all zero crossings have circles. This is because a threshold parameter dThresh is used to discard those zero crossings that appear in regions with high numerical noise. Its default value is 1% of the length of the Poincare section. The position of each detected zero crossing is refined by iteratively bisecting the interval of the zero crossing. If after a predetermined number of iterations, nBisection whose default value is five, the two points around the zero crossing have absolute values of more dThresh, the zero crossing is discarded. In Figure 3.3, discarded zero crossings are seen where $0 < s < 0.05$. Once all valid closed lambda-line orbits have been located, the outermost orbit of every Poincare section is deemed a shear LCS.

3.3 Hyperbolic LCS

The main steps of the algorithm used to calculate hyperbolic LCS are enumerated in Table 3.6. The main function to compute hyperbolic LCS in LCS Tool is seed_curves_from_lambda_max and its syntax is given in Table 3.7.

Strainlines, trajectories of the $\boldsymbol{\xi}_1$ vector field, are integrated starting from local maxima of λ_2 on the main grid. The local maximisation distance distance is defined. This distance is set to a value between two and five grid points, since it represents the number of grid points over which the Cauchy-Green strain tensor is susceptible to numerical noise

1. Define a local maximisation distance.
2. Find all local maximums of λ_2 on the main grid such that they are maximums within a circle with a radius equal to the local minimisation distance.
3. Define a maximum strainline length
4. Integrate a strainline forward and backward, using the largest λ_2 local maximum as the initial position. Integrate until the strainline has attained the maximum strainline length, or until it has reached the domain boundary.
5. Flag any remaining local maximums of λ_2 within the minimisation distance of the strainline as ineligible initial positions for subsequent strainlines
6. Continue integrating strainlines using local maximums of λ_2 as initial positions until no eligible local maximums of λ_2 remain.
7. Remove any strainline segment contained within (closed) shear LCS

Table 3.6: Algorithm to calculate hyperbolic LCS

[curvePosition,curveInitialPosition] = seed_curves_from_lambda_max(distance, cgEigenvalue,cgEigenvector,flowDomain,flowResolution)	
distance	threshold distance for placement of lambda maximums
cgEigenvalue	array of Cauchy-Green strain eigenvalues
cgEigenvector	array of Cauchy-Green strain eigenvectors
flowDomain	2×2 array to define flow domain
flowResolution	1×2 array to define Cauchy-Green strain main grid resolution
periodicBc	optional 1×2 logical array to specify periodic boundary conditions. Default is [false,false].
nMaxCurves	optional maximum number of curves (i.e. strainlines of stretchlines) to generate. Default is numel(cgEigenvalue).
odeSolverOptions	optional odeset structure to specify flow map integration options

Table 3.7: Syntax of the function seed_curves_from_lambda_max

from integration. Thus, the local maximisation distance is expected to be a function of variable time step integration tolerances. A record is made of all grid points where λ_2 is maximal within a circle whose radius equals the defined maximisation distance. Then the first strainline is integrated from the largest of all the local maxima in both directions, i.e., integrating $\boldsymbol{\xi}_1$ and $-\boldsymbol{\xi}_1$. Similarly to lambda-lines, this integration must take into account eigenvector field orientation discontinuities. Prior to initiating strainline integration, a maximum length is defined. Strainlines are then integrated until reaching the domain boundary or the maximum length. The maximum length may be seen as safety factor to prevent highly sinuous strainlines leading to inordinately long computations; the maximum length should be set such that most strainline stop at the domain boundary. Once a strainline position has been calculated, local maxima of λ_2 within the minimisation distance are flagged as ineligible initial positions for subsequent strainlines. After the first strainline has been obtained and any ineligible local maximums of λ_2 have been flagged, the next strainline is integrated. The process continues until all eligible local maxima of λ_2 have been used to initialise strainline integration. Finally, any strainline segment contained within (closed) shear LCS are removed. These calculations are performed by the functions seed_curves_from_lambda_max and remove_strain_in_shear.

Note that once a strainline integration has begun, it may progress so as to be within the local minimisation distance of an existing strainline. In other words, the local minimisation distance controls where strainlines begin, but not where they stop.

Hyperbolic LCS inside ellitpic LCS Explain, figure?

4 Example

In this section we present three examples of how to use LCS-Tool to obtain LCSs in different flows, namley the double gyre flow, the Bickley jet flow, and an oceanic geostrophic flow. All three examples are available as demo files in the demo folder of LCS-Tool and the user can follow the computation of LCS step by step.

4.1 The double gyre

The double gyre is a model for a time-dependent two gyre system observed in geophysical flows [18]. It consists of two counterrotating sinusoidal vortices with a harmonically oscillating line in-between. The velocity equations are:

$$\begin{aligned} \dot{x} &= -\pi A \sin[\pi f(x,t)] \cos(\pi y) \\ \dot{y} &= \pi A \cos[\pi f(x,t)] \sin(\pi y) \frac{\partial f(x,t)}{\partial x} \\ f(x,t) &= \epsilon \sin(\omega t) x^2 + [1 - 2\epsilon \sin(\omega t)] x. \end{aligned} \tag{4.1}$$

The MATLAB function corresponding to these equations is given in Listing 4.1 and shows how to specify a derivative function that supports vector form integration.

Listing 4.1: Double gyre derivative function corresponding to Equations 4.1.

```
function derivative_ = derivative(t,x,epsilon,amplitude,omega)

idx1 = 1:2:numel(x)-1;
idx2 = 2:2:numel(x);

a = epsilon*sin(omega*t);
b = 1 - 2*epsilon*sin(omega*t);
forcing = a*x(idx1).^2 + b*x(idx1);

derivative_ = nan(size(x));

derivative_(idx1) = -pi*amplitude*sin(pi*forcing).*cos(pi*x(idx2));
derivative_(idx2) = pi*amplitude*cos(pi*forcing).*sin(pi*x(idx2)).
    *(2*a*x(idx1) + b);
```

In what follows, the parameter values are: $A = 0.1$, $\epsilon = 0.1$, $\omega = \pi/5$. The flow timespan is $t \in [0, 5]$ and the domain is $x \in [0, 2]$, $y \in [0, 1]$. In Figure 4.1, we show the FTLE at a resolution of 500×250.

By examining the FTLE, we can position Poincare sections to detect lambda-line LCSs. An LCS Tool script to perform this operation is given in Listing 4.2 where two Poincare sections are defined. In the script we have set error tolerances for integration of the Cauchy-Green strain tensor and lambda lines. These numerical parameters help achieve convergence.

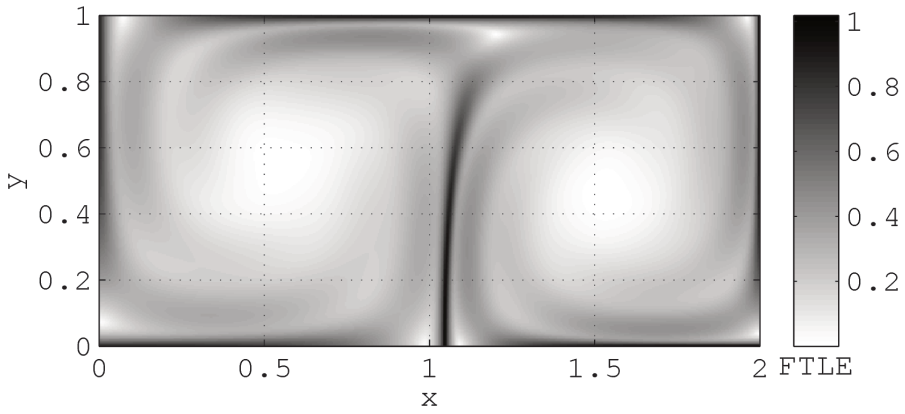

Figure 4.1: Double gyre FTLE.

Listing 4.2: Double gyre script for lambda-line LCSs

```
%% Input parameters
epsilon = .1;
amplitude = .1;
omega = pi/5;
domain = [0,2;0,1];
resolution = [500,250];
timespan = [0,5];

%% Velocity definition
lDerivative = @(t,x,~)derivative(t,x,false,epsilon,amplitude,omega);
incompressible = true;

%% LCS parameters
cgStrainOdeSolverOptions = odeset('relTol',1e-5);

% Lambda-lines
lambda = 1;
lambdaLineOdeSolverOptions = odeset('relTol',1e-6);
poincareSection = struct('endPosition',{},'numPoints',{},'
    orbitMaxLength',{});
poincareSection(1).endPosition = [.55,.55;.2,.5];
poincareSection(2).endPosition = [1.53,.45;1.9,.5];

% Number of orbit seed points along each Poincare section
[poincareSection.numPoints] = deal(100);

% Set maximum orbit length to twice the expected circumference
nPoincareSection = numel(poincareSection);
for i = 1:nPoincareSection
```

```
        rOrbit = hypot(diff(poincareSection(i).endPosition(:,1)),diff(
            poincareSection(i).endPosition(:,2)));
        poincareSection(i).orbitMaxLength = 2*(2*pi*rOrbit);
    end

    %% Cauchy-Green strain eigenvalues and eigenvectors
    [cgEigenvector,cgEigenvalue] = eig_cgStrain(lDerivative,domain,
        resolution,timespan,'incompressible',incompressible,'
        odeSolverOptions',cgStrainOdeSolverOptions);

    %% Lambda-line LCSs
    [shearline.etaPos,shearline.etaNeg] = lambda_line(cgEigenvector,
        cgEigenvalue,lambda);
    closedLambdaLine = poincare_closed_orbit_multi(domain,resolution,
        shearline,poincareSection,'odeSolverOptions',
        lambdaLineOdeSolverOptions);
```

In Figure 4.2 the resolution of the Cauchy-Green strain tensor is varied from 500×250 to 1000×500. The location of the outermost closed lambda-line changes little, demonstrating convergence.

We continue the example by adding the calculation of hyperbolic strainline and stretchline LCSs. Figures 4.3 and 4.4 show the results and Listing 4.3 gives the LCS Tool commands. The local maximisation distance is set larger for stretchlines than for strainlines usually. This is because strainlines follow ridges of λ_2 maximums whereas stretchlines do not follow ridges of λ_1 minimums.

Listing 4.3: LCS Tool commands for strainline and stretchline LCSs

```
    %% Strainlines LCSs
    strainlineMaxLength = 20;
    gridSpace = diff(domain(1,:))/(double(resolution(1))-1);
    strainlineLocalMaxDistance = 2*gridSpace;

    strainlineLcs = seed_curves_from_lambda_max(
        strainlineLocalMaxDistance,strainlineMaxLength,cgEigenvalue
        (:,2),cgEigenvector(:,1:2),domain,resolution);

    %% Stretchlines LCSs
    stretchlineMaxLength = 20;
    stretchlineLocalMaxDistance = 10*gridSpace;

    stretchlineLcs = seed_curves_from_lambda_max(
        stretchlineLocalMaxDistance,stretchlineMaxLength,-
        cgEigenvalue(:,1),cgEigenvector(:,3:4),domain,resolution);
```

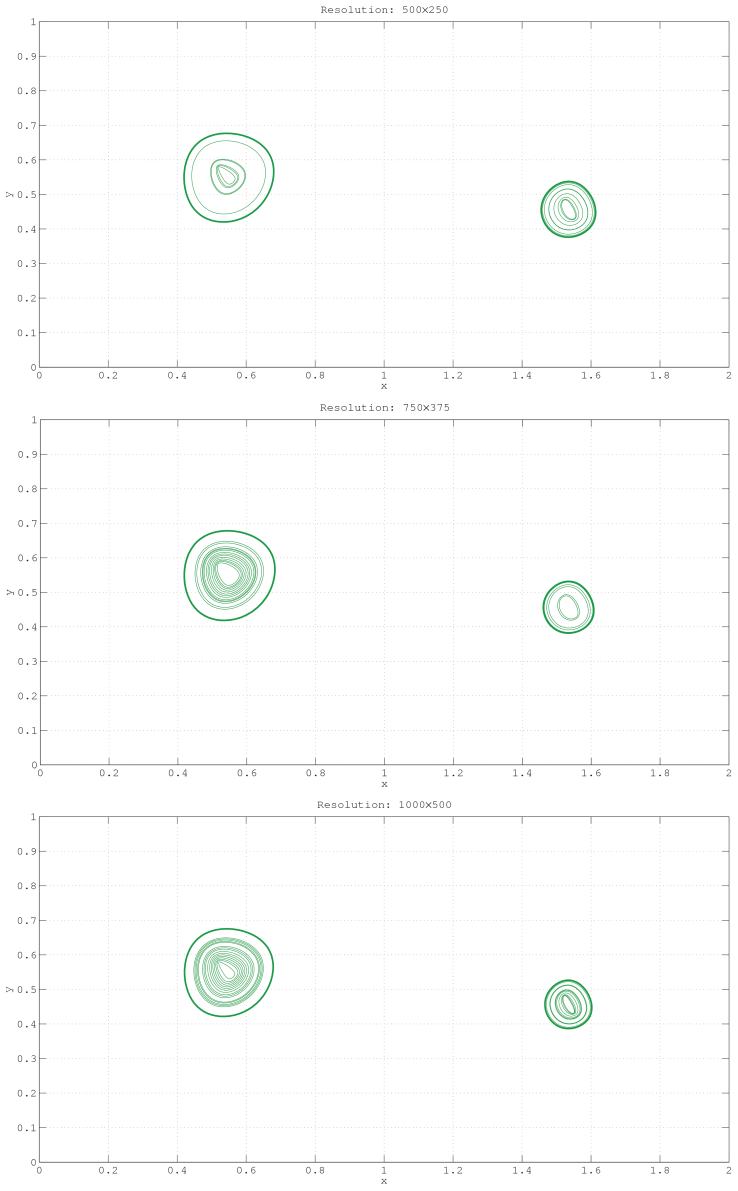

Figure 4.2: Double gyre closed lambda-line at varying resolutions

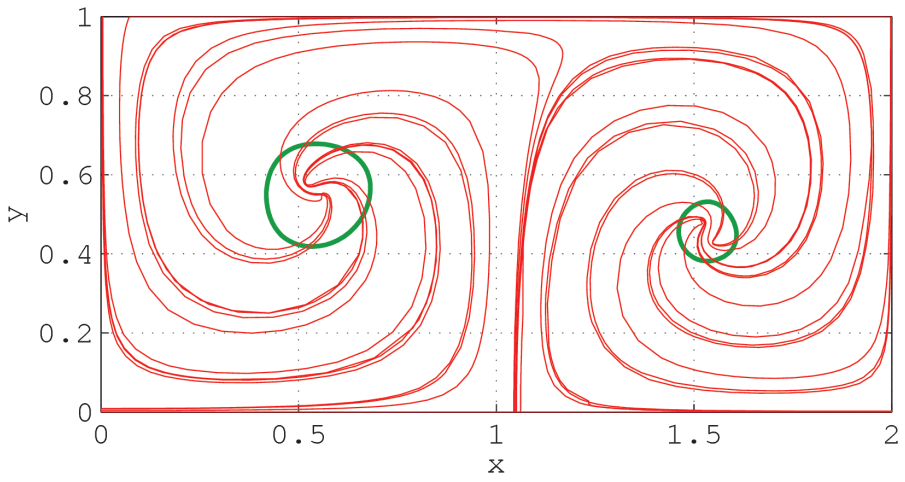

Figure 4.3: Double gyre lambda-line and strainline LCSs

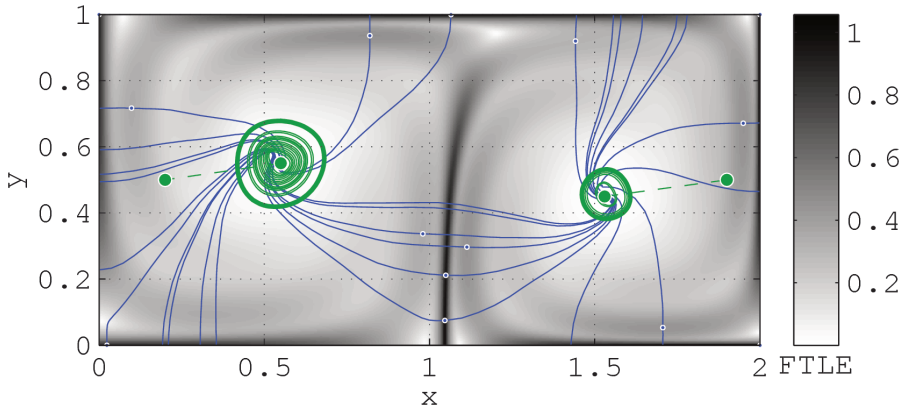

Figure 4.4: Double gyre lambda-line and stretchline LCSs

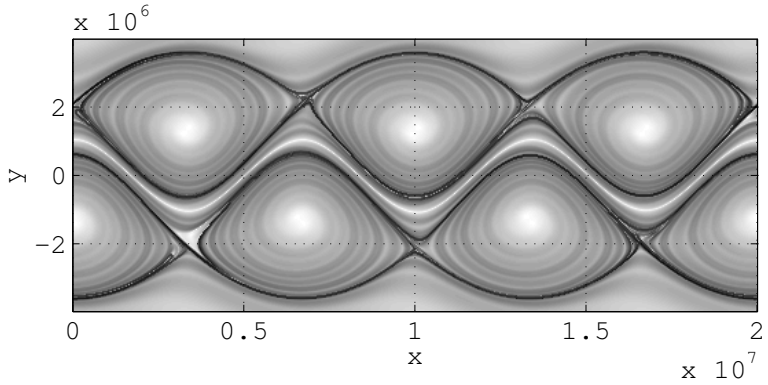

Figure 4.5: Bickley jet FTLE

4.2 Bickley jet

The Bickley jet models a meandering zonal jet flanked above and below by counterrotating vortices. This is an idealised model of geophysical flows such as the polar night jet perturbed by a Rossby wave[8, 19].

The velocity is given by $\boldsymbol{v}(x,y,t) = (-\partial_y\psi, \partial_x\psi)$ where

$$\psi(x,y,t) = \psi_0(x,y) + \psi_1(x,y,t),$$
$$\psi_0(x,y) = c_3 y - UL_y \tanh\frac{y}{L_y} + \epsilon_3 UL_y \operatorname{sech}^2\frac{y}{L_y}\cos k_3 x,$$
$$\psi_1(x,y,t) = UL_y \operatorname{sech}^2\frac{y}{L_y}\operatorname{Re}\left[\sum_{n=1}^{2}\epsilon_n f_n(t) e^{ik_n x}\right].$$

As a forcing function we choose a chaotic solution of the Duffing oscillator, specifically

$$\frac{d\phi_1}{dt} = \phi_2,$$
$$\frac{d\phi_1}{dt} = -0.1\phi_2 - \phi_1^3 + 11\cos(t),$$
$$f_{1,2}(t) = 2.625\times 10^{-2}\phi_1(t/6.238\times 10^5)$$

The parameter values we use are: $U = 62.66$, $c_2 = 0.205U$, $c_3 = 0.461U$, $L_y = 1.77\times 10^6$, $\epsilon_1 = 0.0075$, $\epsilon_2 = 0.04$, $\epsilon_3 = 0.3$, $L_x = 6.371\times 10^6\pi$, $k_n = 2n\pi/L_x$, $\sigma_1 = 0.5k_2(c_2 - c_3)$, $\sigma_2 = 2\sigma_1$.

Figure 4.5 shows the overall appearance of the Bickley jet and in Figure 4.6 lambda-line LCSs are shown. The latter figure demonstrates a poor result since the LCSs change shape significantly from one eddy to the next. To address this problem we could try to adjust numerical parameters such as the resolution, but with our hardware memory availability was reached before achieving convergence. To circumvent this problem, we analyse lambda-line closed orbits of one vortex for a short timespan.

The script used to analyse lambda-line LCS is given in Listing 4.4. The script produces the data shown in Figure 4.7. Parameter values used

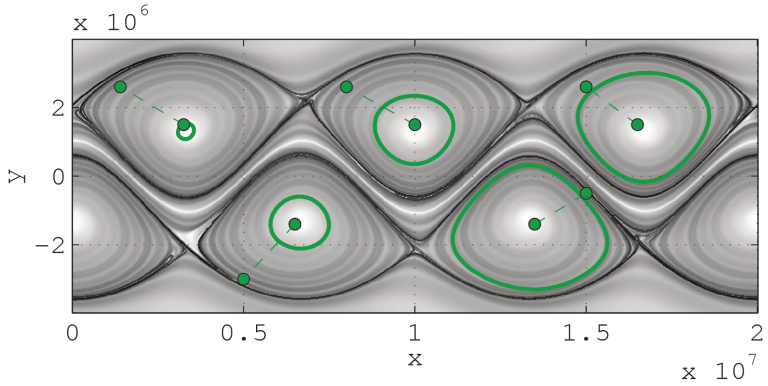

Figure 4.6: Bickley jet lambda-line LCSs

may read from the script. It is worth pointing out that in order to obtain closed orbits, $\lambda = 0.995$; with $\lambda = 1$ no closed orbits are found.

Listing 4.4: Bickley jet script for lambda-line LCSs

```
%% Input parameters
u = 62.66;
lengthX = pi*earthRadius;
lengthY = 1.77e6;
epsilon = [.075,.4,.3];
domain = [2e6,lengthX/2;[-1,.25]*2.25*lengthY];
resolutionX = 400:100:600;
timespan = [0,2*lengthX/u];

%% Velocity definition
perturbationCase = 3;
phiTimespan = [0,25];
phiInitial = [0,0];
phiSol = ode45(@d_phi,phiTimespan,phiInitial);
timeResolution = 1e5;
phi1 = deval(phiSol,linspace(phiTimespan(1),phiTimespan(2),
    timeResolution),1);
phi1Max = max(phi1);
lDerivative = @(t,x,~)derivative(t,x,false,u,lengthX,lengthY,epsilon,
    perturbationCase,phiSol,phi1Max);
incompressible = true;

%% LCS parameters
% Cauchy-Green strain
cgStrainOdeSolverOptions = odeset('relTol',1e-4);

% Lambda-lines
```

```
lambda = .995;
lambdaLineOdeSolverOptions = odeset('relTol',1e-6);
poincareSection.endPosition = [6.5,-1.4;4.5,-3.5]*1e6;
[poincareSection.numPoints] = deal(100);
rOrbit = hypot(diff(poincareSection.endPosition(:,1)),diff(
    poincareSection.endPosition(:,2)));
poincareSection.orbitMaxLength = 2*(2*pi*rOrbit);
lambdaLineColor = [0,.6,0];
dThresh = 1e-3;

for m = 1:numel(resolutionX)
    % Make x and y grid spacing as equal as possible
    lResolutionX = resolutionX(m);
    gridSpace = diff(domain(1,:))/(double(lResolutionX)-1);
    resolutionY = round(diff(domain(2,:))/gridSpace) + 1;
    resolution = [lResolutionX,resolutionY];

    %% Cauchy-Green strain eigenvalues and eigenvectors
    [cgEigenvector,cgEigenvalue] = eig_cgStrain(lDerivative,domain,
        resolution,timespan,'incompressible',incompressible,'
        odeSolverOptions',cgStrainOdeSolverOptions);

    %% Lambda-line LCSs
    [shearline.etaPos,shearline.etaNeg] = lambda_line(cgEigenvector
        ,cgEigenvalue,lambda);
    [closedLambdaLine,~,hFigure] = poincare_closed_orbit_multi(
        domain,resolution,shearline,poincareSection,'
        odeSolverOptions',lambdaLineOdeSolverOptions,'dThresh',
        dThresh,'showGraph',true);
end
```

To calculate strainline and stretchline LCSs we set the timespan to $t \in [0, 2L_x/U]$. The script used to detect strainlines is given in Listing 4.5 and the results are shown in Figure 4.8. Stretchline LCSs are shown in Figure 4.9.

Listing 4.5: LCS Tool commands for Bickley jet strainline LCSs

```
%% Input parameters
u = 62.66;
lengthX = pi*earthRadius;
lengthY = 1.77e6;
epsilon = [.075,.4,.3];
domain = [2e6,.55*lengthX;[-1,.25]*2.25*lengthY];
resolutionX = 500;
timespan = [0,2*lengthX/u];

%% Velocity definition
perturbationCase = 3;
phiTimespan = [0,25];
```

```
phiInitial = [0,0];
phiSol = ode45(@d_phi,phiTimespan,phiInitial);
timeResolution = 1e5;
phi1 = deval(phiSol,linspace(phiTimespan(1),phiTimespan(2),
    timeResolution),1);
phi1Max = max(phi1);
lDerivative = @(t,x,~)derivative(t,x,false,u,lengthX,lengthY,epsilon,
    perturbationCase,phiSol,phi1Max);
incompressible = true;

%% LCS parameters
% Cauchy-Green strain
cgStrainOdeSolverOptions = odeset('relTol',1e-4);

% Strainlines
strainlineMaxLength = 1e8;
strainlineOdeSolverOptions = odeset('relTol',1e-4);

% Make x and y grid spacing as equal as possible
gridSpace = diff(domain(1,:))/(double(resolutionX)-1);
resolutionY = round(diff(domain(2,:))/gridSpace) + 1;
resolution = [resolutionX,resolutionY];

strainlineLocalMaxDistance = 4*gridSpace;

%% Cauchy-Green strain eigenvalues and eigenvectors
[cgEigenvector,cgEigenvalue] = eig_cgStrain(lDerivative,domain,
    resolution,timespan,'incompressible',incompressible,'
    odeSolverOptions',cgStrainOdeSolverOptions);

%% Strainline LCSs
strainlineLcs = seed_curves_from_lambda_max(
    strainlineLocalMaxDistance,strainlineMaxLength,cgEigenvalue
    (:,2),cgEigenvector(:,1:2),domain,resolution,'odeSolverOptions',
    strainlineOdeSolverOptions);
```

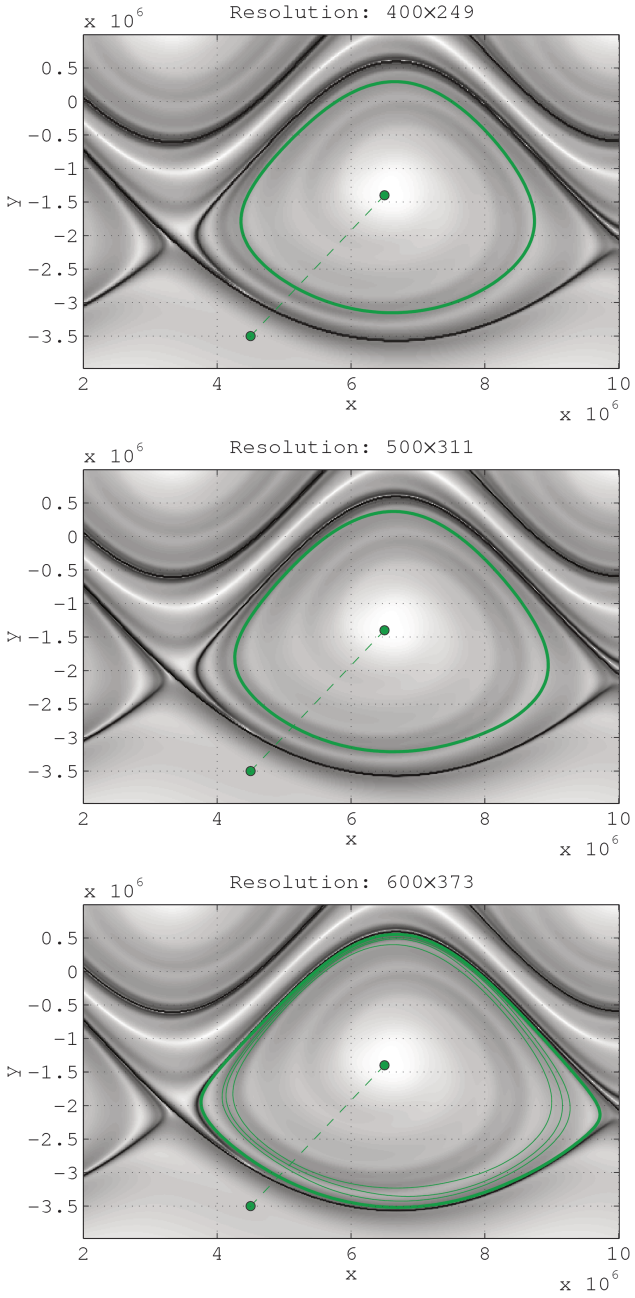

Figure 4.7: Bickley jet closed lambda-line at varying resolutions

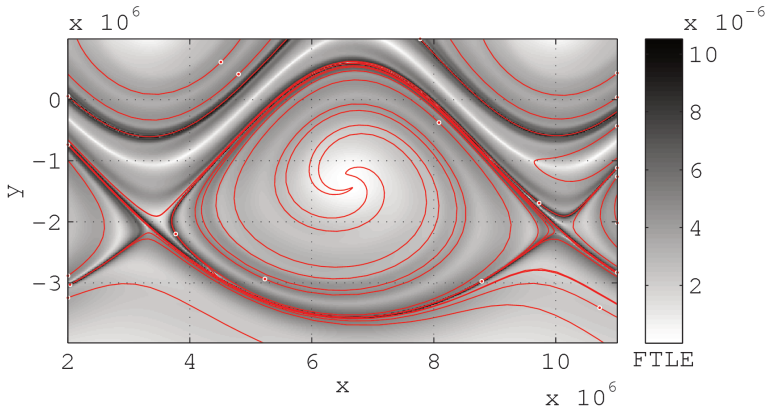

Figure 4.8: Bickley jet strainline LCSs

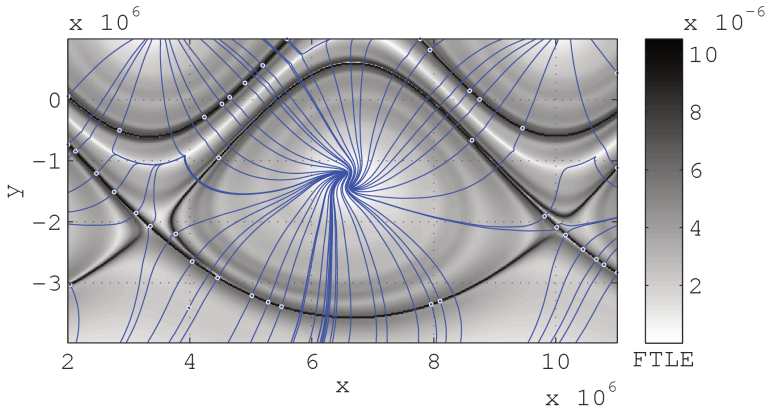

Figure 4.9: Bickley jet stretchline LCSs

4.3 Ocean dataset

For the last example, we choose a measured geophysical flow where, in contrast to the previous analytical examples, the two dimensional incompressible velocity field is presented as a dataset with discrete temporal and spatial resolution. Our region of interest is a small domain in the South Atlantic Ocean, where exceptionally coherent eddies, Agulhas rings, were recently found using methods provided by LCS Tool [10]. The oceanic velocity field is obtained from altimetric satelite measurements of the sea surface height under the assumption that the flow is geostrophic. Under this assumption the sea surface height η serves as a streamfunction and in a longitude-latitude (φ, θ) coordinate system the evolution of a fluid particle is given by

$$\dot{\varphi}(\varphi,\theta,t) = -\frac{g}{R^2 f(\theta)\cos\theta}\partial_\theta \eta(\varphi,\theta,t) \tag{4.2}$$

$$\dot{\theta}(\varphi,\theta,t) = +\frac{g}{R^2 f(\theta)\cos\theta}\partial_\varphi \eta(\varphi,\theta,t) \tag{4.3}$$

where g is the gravitational acceleration, R is the mean radius of Earth, and $f(\theta) := 2\Omega\sin\theta$ is the Coriolis parameter with Ω being the Earth's mean angular velocity.

The data is given at a spatial resolution of $1/4°$ and temporal resolution of 7 days. Due to the discrete data, defining the right hand side of Eq. (4.2) and (4.3) involves interpolation in space and time with a spline interpolation scheme. An interpolant is generated first, then the function flowdata_derivative evaluates the interpolants for the zonal and meridional velocity at the needed coordinates. Listing 4.6 shows the corresponding commands.

Listing 4.6: Generate interpolant of ocean velocity data set.

```
% Define right hand side of ODE, ocean.flow.derivative
interpMethod = 'spline';
vlon_interpolant = griddedInterpolant({time,lat,lon},vlon,
    interpMethod);
vlat_interpolant = griddedInterpolant({time,lat,lon},vlat,
    interpMethod);
ocean.flow.derivative = @(t,y,~)flowdata_derivative(t,y,
    vlon_interpolant,vlat_interpolant);
```

For the ocean data, the integration time was chosen to be $T = 30$ days, a time larger than the turn over time of the mesoscale eddies in the domain. The resolution of the grid of initial conditions is set to 400×400 which corresponds to a resolution of roughly $0.015°$ and gives good results. With this choice the resolution of the tracer grid is of the order of 15 times higher than the resolution of the velocity field. Figures 4.10 and 4.11 show elliptic and hyperbolic LCS on 22 November 2006, i.e., the same time instant shown in Haller and Beron-Vera [10], Beron-Vera et al. [20]. The analysis with LCS Tool reveals a large coherent eddy bounded by a closed lambda-line at $(3, -32)$ and four further smaller coherent eddy cores.

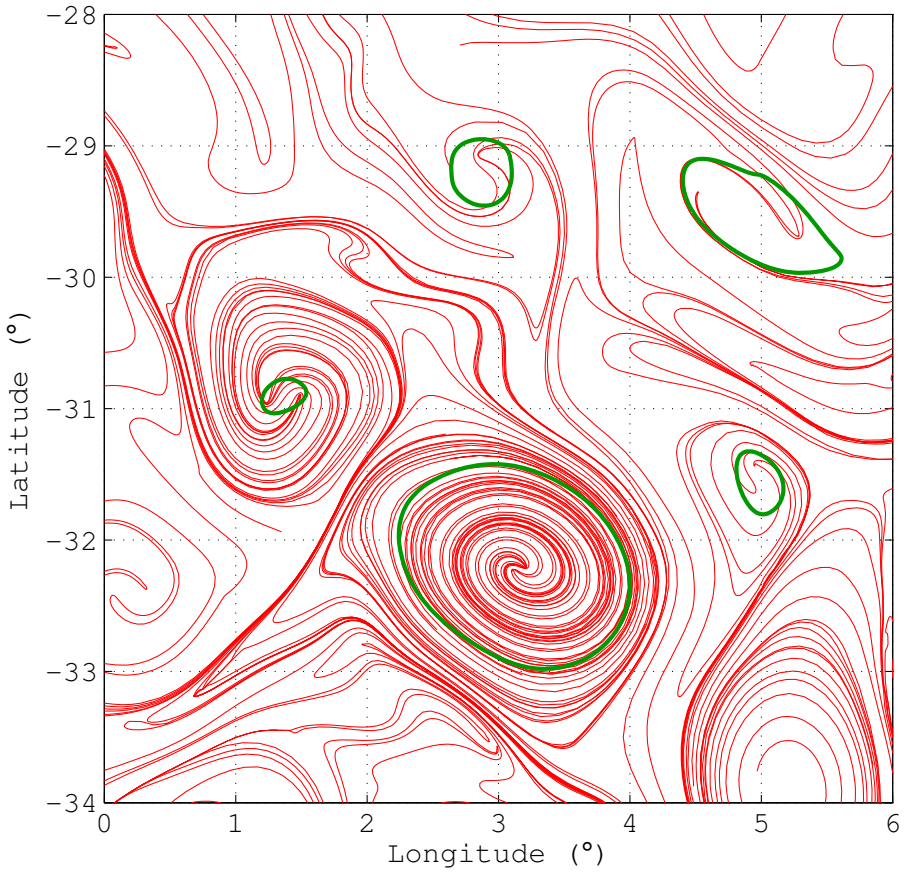

Figure 4.10: Ocean dataset lambda-line (green) and strainline (red) LCS.

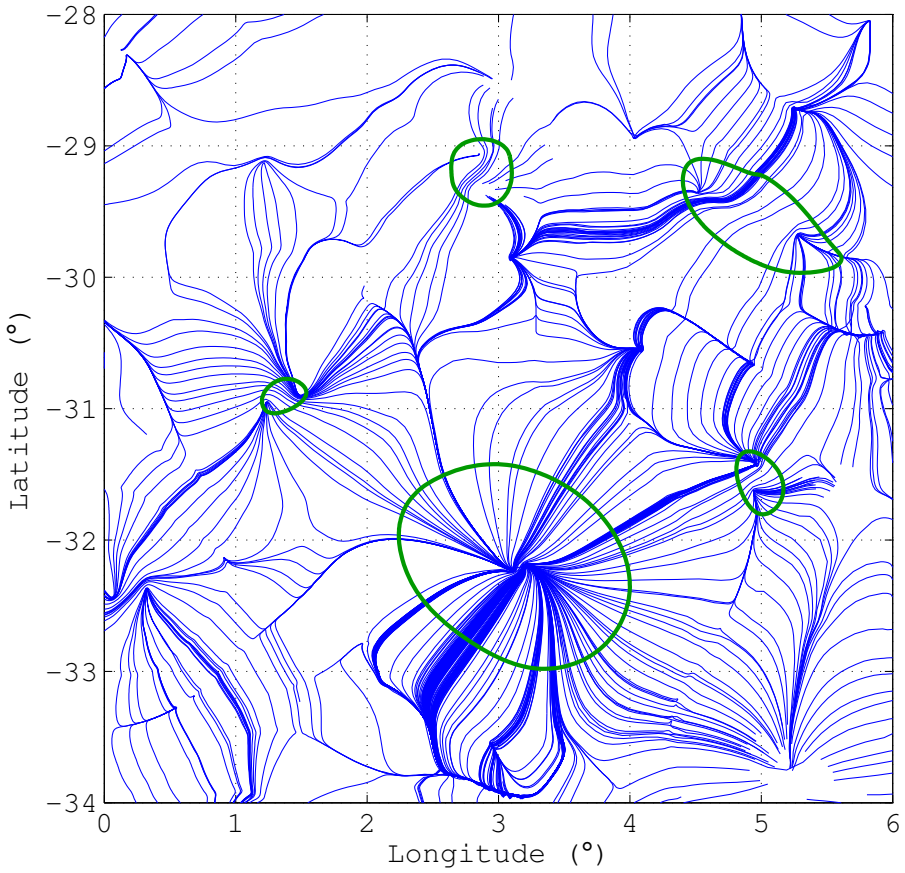

Figure 4.11: Ocean dataset lambda-line (green) and stretchline (blue) LCSs.

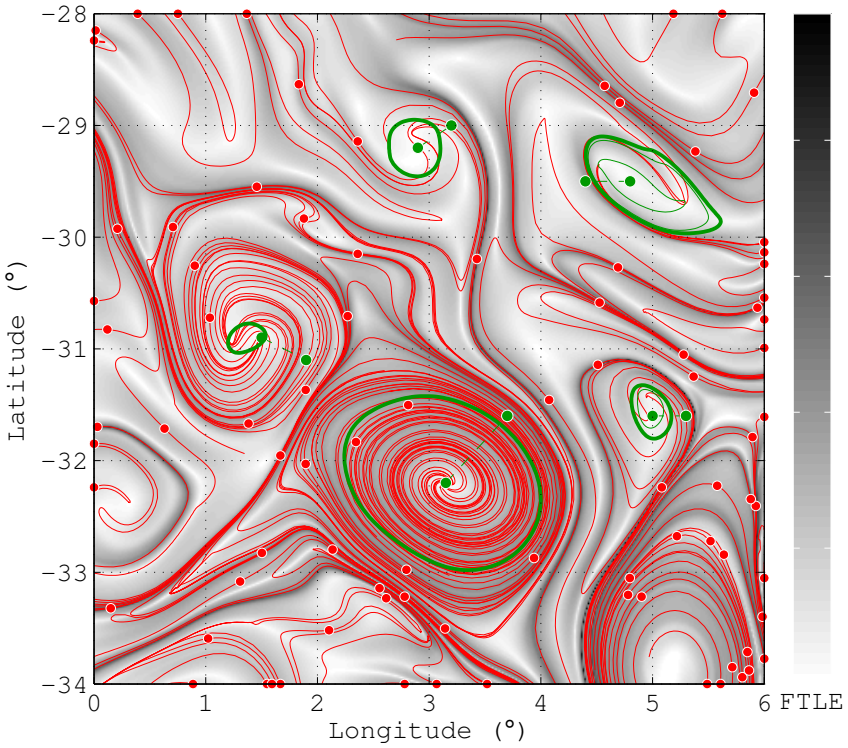

Figure 4.12: Same LCS as in Figure 4.10, but with additional details. Background gray-scale is the finite-time Lyapunov exponent. Dashed green lines are Poincare sections. Green curves are closed lambda-lines with thick green curves being outermost closed lambda-lines, i.e. lambda-line LCSs. White dots indicate λ_2 local maximums used for strainline integration initial positions.

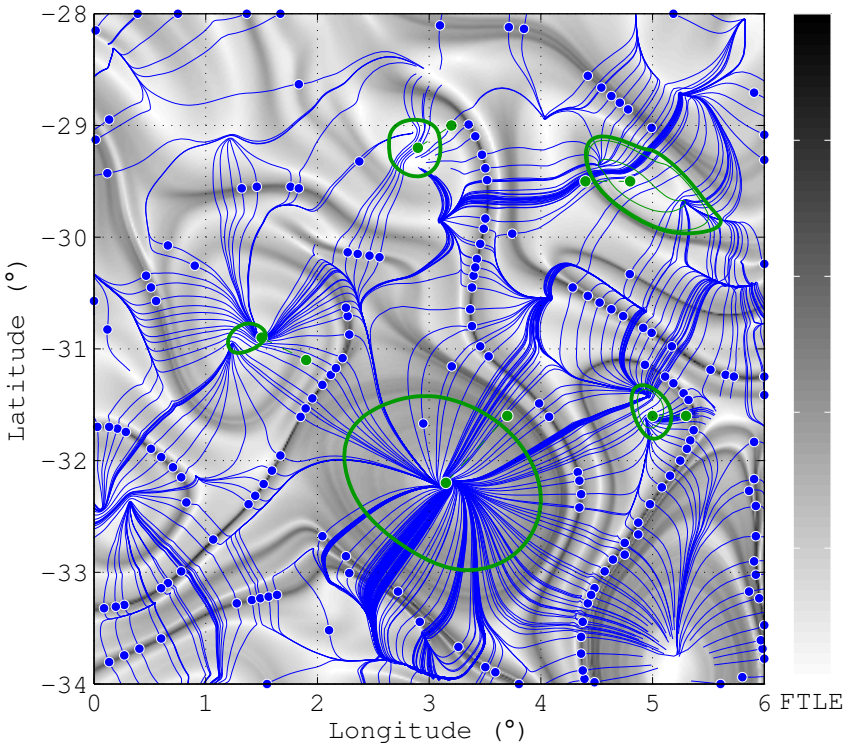

Figure 4.13: Same LCS as in Figure 4.11, but with additional details. Background gray-scale is the finite-time Lyapunov exponent. Dashed green lines are Poincare sections. Green curves are closed lambda-lines and thick green curves are outermost closed lambda-lines, i.e. lambda-line LCSs. White dots indicate λ_1 local minimums used for stretchline integration intial positions.

5 Conclusions

Algorithms to identify LCSs using variational principles have been presented, and a software library, LCS Tool, that implements these algorithms has been demonstrated on geophysical flow models and a dataset. The software library provides functions to enable exploring variational LCS methods without requiring detailed knowledge of variational LCS theory. These functions make it possible to write compact programs identify and plot LCSs.

LCS Tool is built atop MATLAB[21] and leverages that software's capabilities extensively. With FTLE-based LCS identification, computational performance has received considerable attention[22, 23]. LCS Tool is free software and we hope it will evolve to incorporate similar advances. Optimising computational performance will likely help apply variational LCS methods to large-scale real-time forecasting applications, for example to allow tracking environmental contaminants.

In addition to improving computational performance, we hope LCS Tool will be the foundation to integrate new theoretical advances, for example the variational LCS jet core identification method[15] and the analysis of LCSs in higher dimensions[24].

Acknowledgements

The altimeter products used in this work are produced by SSALTO/DUACS and distributed by AVISO, with support from CNES http://www.aviso.oceanobs.com/duacs.

References

[1] M. M. Wilson et al. "Lagrangian coherent structures in low Reynolds number swimming". In: *Journal of Physics: Condensed Matter* 21.20 (2009), p. 204105. DOI: 10.1088/0953-8984/21/20/204105.

[2] P. Tallapragada, S. D. Ross, and D. G. Schmale Burton III. "Lagrangian coherent structures are associated with fluctuations in airborne microbial populations". In: *Chaos* 21.3 (2011), p. 033122. DOI: 10.1063/1.3624930.

[3] W. Tang, P. W. Chan, and G. Haller. "Accurate extraction of Lagrangian coherent structures over finite domains with application to flight data analysis over Hong Kong International Airport". In: *Chaos* 20.1 (2010), p. 017502. DOI: 10.1063/1.3276061.

[4] A. Hadjighasem, M. M. Farazmand, and G. Haller. "Detecting invariant manifolds, attractors, and generalized KAM tori in aperiodically forced mechanical systems". In: *Nonlinear Dynamics* 73.1-2 (2013), pp. 689–704. ISSN: 0924-090X. DOI: 10.1007/s11071-013-0823-x.

[5] G. Haller. "A variational theory of hyperbolic Lagrangian Coherent Structures". In: *Physica D: Nonlinear Phenomena* 240.7 (Mar. 2011), pp. 574–598. DOI: 10.1016/j.physd.2010.11.010.

[6] G. Norgard and P.-T. Bremer. "Second derivative ridges are straight lines and the implications for computing Lagrangian Coherent Structures". In: *Physica D: Nonlinear Phenomena* 241.18 (Sept. 2012), pp. 1475–1476. ISSN: 0167-2789. DOI: 10.1016/j.physd.2012.05.006.

[7] M. M. Farazmand and G. Haller. "Computing Lagrangian coherent structures from their variational theory". In: *Chaos* 22.1 (2012). DOI: 10.1063/1.3690153.

[8] G. Haller and F. J. Beron-Vera. "Geodesic Theory of Transport Barriers in Two-Dimensional Flows". In: *Physica D: Nonlinear Phenomena* 241.20 (Oct. 2012), pp. 1680–1702. DOI: 10.1016/j.physd.2012.06.012.

[9] M. M. Farazmand and G. Haller. "Attracting and repelling Lagrangian coherent structures from a single computation". In: *Chaos* 23.2 (2013). DOI: 10.1063/1.4800210.

[10] G. Haller and F. J. Beron-Vera. "Coherent Lagrangian vortices: the black holes of turbulence". In: *Journal of Fluid Mechanics* 731 (Sept. 2013), R4. DOI: 10.1017/jfm.2013.391.

[11] F. Lekien. "Time-dependent dynamical systems and geophysical flows". PhD thesis. California Institute of Technology, 2003. URL: http://resolver.caltech.edu/CaltechETD:etd-04082003-180353.

[12] J. O. Dabiri. *LCS MATLAB Kit*. 2009. URL: http://dabiri.caltech.edu/software.html.

[13] P. C. du Toit. "Transport and separatrices in time-dependent flows". PhD thesis. California Institute of Technology, 2010. URL: http://resolver.caltech.edu/CaltechTHESIS:10072009-165901284.

[14] S. C. Shadden. *FlowVC*. 2010. URL: http://shaddenlab.berkeley.edu/software.html.

[15] M. M. Farazmand, D. Blazevski, and G. Haller. *Shearless transport barriers in unsteady two-dimensional flows and maps*. Submited to Physica D: Nonlinear Phenomena. Aug. 28, 2013. arXiv:1308.6136 [math.DS].

[16] T. Delmarcelle and L. Hesselink. "The topology of symmetric, second-order tensor fields". In: *Visualization, 1994., Visualization '94, Proceedings., IEEE Conference on*. Oct. 1994, pp. 140–147. DOI: 10.1109/VISUAL.1994.346326.

[17] V. I. Arnol'd. *Mathematical methods of classical mechanics*. Springer-Verlag, 1978.

[18] S. C. Shadden, F. Lekien, and J. E. Marsden. "Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows". In: *Physica D: Nonlinear Phenomena* 212.3-4 (Dec. 2005), pp. 271–304. DOI: 10.1016/j.physd.2005.10.007.

[19] F. J. Beron-Vera et al. "Invariant-tori-like Lagrangian coherent structures in geophysical flows". In: *Chaos* 20 (2010), p. 017514. DOI: 10.1063/1.3271342.

[20] F. J. Beron-Vera et al. "Objective detection of oceanic eddies and the Agulhas leakage". In: *Journal of Physical Oceanography* (2013). DOI: 10.1175/JPO-D-12-0171.1.

[21] MathWorks. *MATLAB*. 2013. URL: http://www.mathworks.com/products/matlab/.

[22] C. Conti, D. Rossinelli, and P. Koumoutsakos. "GPU and APU computations of Finite Time Lyapunov Exponent fields". In: *Journal of Computational Physics* 231.5 (Mar. 2012), pp. 2229–2244. DOI: 10.1016/j.jcp.2011.10.032.

[23] P. Miron et al. "Anisotropic mesh adaptation on Lagrangian Coherent Structures". In: *Journal of Computational Physics* 231.19 (Aug. 2012), pp. 6419–6437. DOI: 10.1016/j.jcp.2012.06.015.

[24] D. Blazevski and G. Haller. *Hyperbolic and Elliptic Transport Barriers in Three-Dimensional Unsteady Flows*. Submited to Physica D: Nonlinear Phenomena. July 4, 2013. arXiv:1306.6497 [math.DS].