AMSC 663 Project Proposal: Lagrangian Analysis of Two- and Three-Dimensional Oceanic Flows from Eulerian Velocity Data

David Russell
Second-year Ph.D. student, Applied Math and Scientific Computing
russelld@umd.edu

Project Advisor: Kayo Ide
Department of Atmospheric and Oceanic Science
Center for Scientific Computation and Mathematical Modeling
Earth System Science Interdisciplinary Center
Institute for Physical Science and Technology
ide@umd.edu

October 15, 2015

Abstract

In this project, we will design the tools to conduct a Lagrangian analysis of an oceanic flow whose velocity is proscribed on a spatio-temporal grid. Our main tools in this analysis will be the so-called M-function (arc-length over a fixed time interval) and the maximal finite-time Lyapunov exponent, both of which help elucidate the underlying coherent structures of the flow. After validating them, we will test these tools on a dataset coming from a model flow of the Chesapeake Bay.

1 Introduction

Ocean currents have all sorts of large-scale coherent structures that are generally invisible to the naked eye. By coherent structure, we mean a blob of fluid that moves as one—eddies or jets, for example [1]. Finding ways to unveil these structures is of general interest to those who study mixing and transport. For example, if a pollutant enters the water, it tends to stay within a single coherent structure, so the boundaries (or manifolds) separating different coherent structures serve as barriers to its transport (neglecting molecular diffusion). The classification of these structures borrows heavily from dynamical systems theory. Of particular interest are the notions of a distinguished hyperbolic trajectory, or DHT (the equivalent of a fixed point in a changing flow field) and stable and

unstable manifolds (structural boundaries on which the flow is toward or away from the DHT, respectively).

Velocity data from ocean models give us the raw material to bring out these structures. To do so, it is useful to move from the grid-based (Eulerian) viewpoint to a flow-following (Lagrangian) one. That is, set up a vast network of fluid "particles" to be tracked through the flow, simulate their trajectories as the flow evolves, and then analyze those trajectories to get structural information. In Lagrangian terms, a coherent structure is nothing more than a group of particles whose trajectories "go together" in some sense. The challenge is to use precise quantitative tools to define what this means.

2 Approach

One way to delineate boundaries between coherent structures is to look for places where velocity changes abruptly in a certain direction. Wherever a region of fast-moving particles abuts a slow-moving region, a structural boundary is evident. This will also be true if one of the regions *will be* moving or *has been* moving faster at a time not far from the current one. In other words, if we color each particle by how far it has traveled or will travel within a certain fixed time interval (for example, the last two days), then color boundaries will tend to align with structural boundaries. Thus, letting $\mathbf{X}(\mathbf{X}_0, t)$ be the position at time t of the particle which began at position $\mathbf{X}_0$ at time t_0, we introduce the so-called M-function [2]:

$$M_{\mathbf{u},\tau}(\mathbf{X}_0^*, t^*) = \int_{t^*-\tau}^{t^*+\tau} \left(\sum_{i=1}^{2 \text{ or } 3} \left(\frac{dX_i(t)}{dt} \right)^2 \right)^{\frac{1}{2}} dt,$$

which is simply the distance traveled by the particle that began at position $\mathbf{X}_0^*$ over the time interval spanning forward and backward time τ from the current time t^*.

Another way to look for transport boundaries is to look for places where a flow bifurcates, or splits apart. If a small parcel of fluid experiences relatively little stretching and squishing as it moves through the flow, chances are that it stays within one coherent structure. Conversely, if it finds itself getting massively stretched (as time runs either forward or backward), chances are that it was on the boundary between two coherent structures. Thus we need a way to quantify the degree to which two nearby trajectories diverge in time. To that end, we introduce the maximal finite-time Lyapunov exponent (FTLE):

$$\lambda = \frac{1}{2t} \ln \left(\lambda_{\max} \left(L^T L \right) \right)$$

where

$$L = \frac{\partial \mathbf{X}(\mathbf{X}_0, t)}{\partial \mathbf{X}_0}$$

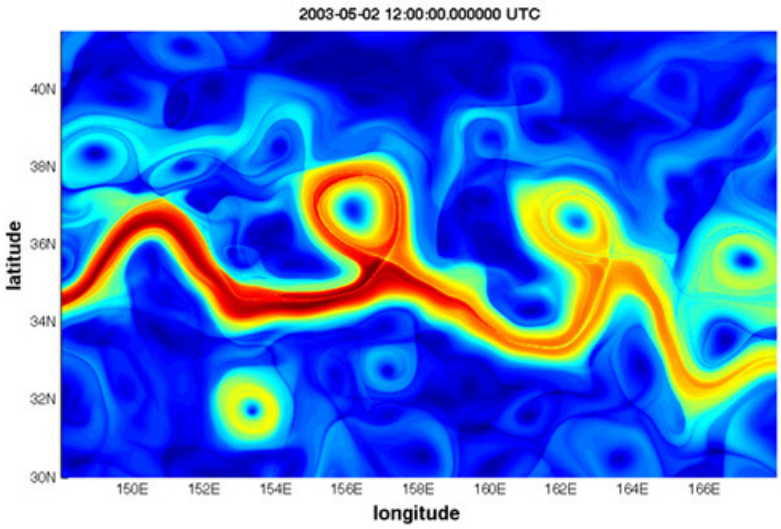

Figure 1: M-function coloring for a dataset from the Kuroshio current [2]. Red represents fast-moving regions, blue slow. Stable and unstable manifolds appear as thin yellow lines.

is the so-called transition matrix (the Jacobian of current position with respect to initial position). An equivalent definition of λ is that, for a locally linear flow field, it represents the maximal exponential growth rate of two infinitesimally close trajectories passing through point $\mathbf{X}_0$ at time t_0:

$$|\delta\mathbf{X}(\mathbf{X}_0, t)| \leq e^{\lambda t}|\delta\mathbf{X}_0|$$

These two tools, the M-function and FTLE, provide a launching point for a Lagrangian analysis of the flow. For example, one can produce visualizations like that in figure 1, in which the eddies, jets, and stable and unstable manifolds are clearly visible.

3 Algorithms

There are two main computational tasks to be performed in this project: computation of the particle trajectories, and analysis of those trajectories to obtain structural information. Where applicable, both low-order and high-order methods will be implemented, for learning purposes as well as to investigate the time feasibility of the higher-order methods. These orders will be verified in the standard way—by estimating errors over a variety of grid sizes, then estimating the slope of a log-log plot of error versus grid spacing.

3.1 Trajectory computation

Let $\mathbf{u}(\mathbf{x}, t)$ be a continuous velocity field in two or three dimensions, i.e.

$$\mathbf{u}(\mathbf{x}, t) = (u, v) = (u(x, y, t), v(x, y, t))$$

in two dimensions, or

$$\mathbf{u}(\mathbf{x}, t) = (u, v, w) = (u(x, y, z, t), v(x, y, z, t), w(x, y, z, t))$$

in three dimensions. A particle trajectory $\mathbf{X}(\mathbf{X}_0, t)$ must then satisfy

$$\dot{\mathbf{X}}(\mathbf{X}_0, t) = \mathbf{u}(\mathbf{X}(\mathbf{X}_0, t), t)$$

where $\dot{\mathbf{X}}$ denotes $\frac{d\mathbf{X}}{dt}$. In three dimensions, for example, this boils down to the system of differential equations

$$\begin{aligned}
\dot{X}(\mathbf{X}_0, t) &= u(X(\mathbf{X}_0, t), Y(\mathbf{X}_0, t), Z(\mathbf{X}_0, t), t) \\
\dot{Y}(\mathbf{X}_0, t) &= v(X(\mathbf{X}_0, t), Y(\mathbf{X}_0, t), Z(\mathbf{X}_0, t), t) \\
\dot{Z}(\mathbf{X}_0, t) &= w(X(\mathbf{X}_0, t), Y(\mathbf{X}_0, t), Z(\mathbf{X}_0, t), t)
\end{aligned}$$

or, in abbreviated form,

$$\begin{aligned}
\dot{X} &= u(X, Y, Z, t) \\
\dot{Y} &= v(X, Y, Z, t) \\
\dot{Z} &= w(X, Y, Z, t).
\end{aligned}$$

The trajectory can thus be determined from a given $\mathbf{u}$ by

$$\mathbf{X}(\mathbf{X}_0, t^*) = \int_{t_0}^{t^*} \mathbf{u}(\mathbf{X}(\mathbf{X}_0, t), t)\, dt.$$

Applying this formula on actual velocity data presents two computational tasks. First, if the velocity components are given only on a spatio-temporal grid, say for example $u(x_i, y_j, z_k, t_l) = u_{i,j,k,l}$, these components must be interpolated between the grid points as the particles travel there. Second, once velocity is interpolated, a numerical integration scheme must be chosen.

3.1.1 Velocity Interpolation

Velocity values are assumed to be given on a so-called Arakawa C-grid (in accordance with the ROMS standard to be discussed later). This means that values for the velocity components u, v, and w are assumed to be staggered within each grid box, so that each is given at the center of a different pair of opposite faces (figure 2). Each of these scalar components will be interpolated individually to the location of each particle. Because the third dimension (depth) differs radically from the other two, the fields will be interpolated first horizontally, and then vertically (in the 3D case).

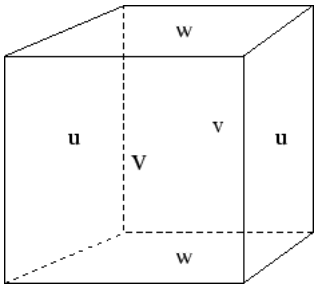

Figure 2: Arakawa c-grid box [6]

The simplest way to interpolate horizontally is to use piecewise bilinear functions. That is, at a fixed height z_k and time t_l, we define a four-parameter bilinear function

$$f(x, y) = c_1 + c_2 x + c_3 y + c_4 xy$$

on $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$ by fitting it to the four given values

$$\begin{aligned} f(x_i, y_j) &= u_{i,j} \\ f(x_{i+1}, y_j) &= u_{i+1,j} \\ f(x_i, y_{j+1}) &= u_{i,j+1} \\ f(x_{i+1}, y_{j+1}) &= u_{i+1,j+1} \end{aligned}$$

and approximate the velocity at (x, y) by

$$u(x, y) \approx f(x, y).$$

This simple interpolation scheme is first-order and not smooth, so as a next step, we plan to implement bicubic interpolation, which is third-order. In this case, we approximate u by a bicubic function

$$f(x, y) = \sum_{i,j=0}^{3} c_{ij} x^i y^j$$

on the rectangle of interest. To solve for the sixteen unknowns c_{ij}, we must fit not only to the given values of u at the rectangle corners, but also to estimated values of the partial derivatives $\partial_x u$, $\partial_y u$, and $\partial_{xy} u$ at those four corners. These partial derivatives can be estimated by centered finite differences using adjacent grid points, i.e.

$$\begin{aligned} \partial_x u_{i,j} &\approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} \\ \partial_y u_{i,j} &\approx \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} \\ \partial_{xy} u_{i,j} &\approx \frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{4\Delta x \Delta y}, \end{aligned}$$

all of which are second order. At boundaries of the domain, these formulas must be abandoned in favor of one-sided difference approximations.

After horizontal interpolation, we interpolate vertically to each point between two interpolated planes. A simple, non-smooth first-order scheme is piecewise linear interpolation from the nearest neighbors directly above and below. For a more accurate (third-order) scheme, we can interpolate a cubic polynomial through the four nearest neighbors, two directly above and two directly below the point in question. For example, the Lagrangian form of the cubic polynomial interpolated through points (z_0, u_0), (z_1, u_1), (z_2, u_2), (z_3, u_3) is

$$\begin{aligned} f(z) =& \frac{(z-z_1)(z-z_2)(z-z_3)}{(z_0-z_1)(z_0-z_2)(z_0-z_3)}u_0 + \frac{(z-z_0)(z-z_2)(z-z_3)}{(z_1-z_0)(z_1-z_2)(z_1-z_3)}u_1 \\ &... + \frac{(z-z_0)(z-z_1)(z-z_3)}{(z_2-z_0)(z_2-z_1)(z_2-z_3)}u_2 + \frac{(z-z_0)(z-z_1)(z-z_2)}{(z_3-z_0)(z_3-z_1)(z_3-z_2)}u_3 \end{aligned}$$

Again, one-sided interpolations must be used at the vertical boundaries.

Finally, interpolation must be done in time as well. This is a one-dimensional problem, so we will attack it with the same tools as vertical interpolation. Namely, piecewise linear interpolants followed by piecewise cubic.

3.1.2 Time Integration

To determine a trajectory given velocity data, we must then integrate this velocity in time to get the position at each time. A simple first-order implicit method is backward Euler, namely

$$\mathbf{X}_{n+1} = \mathbf{X}_n + \mathbf{u}(\mathbf{X}_{n+1}, t_{n+1}),$$

which is known to be A-stable and L-stable. For a higher-order scheme, we will use the Milne-Hamming scheme, which is a fourth-order implicit multistep (Adams-Moulton) predictor-corrector method. Its equations are:

$$\hat{\mathbf{X}}_{n+1} = \mathbf{X}_{n-3} + \frac{4\Delta t}{3}\left(2\mathbf{u}(\mathbf{X}_n, t_n) - \mathbf{u}(\mathbf{X}_{n-1}, t_{n-1}) + \mathbf{u}(\mathbf{X}_{n-2}, t_{n-2})\right)$$

for the predictor and

$$\mathbf{X}_{n+1} = \frac{9}{8}\mathbf{X}_n - \frac{1}{8}\mathbf{X}_{n-2} + \frac{3\Delta t}{8}\left(\mathbf{u}(\hat{\mathbf{X}}_{n+1}, t_{n+1}) + 2\mathbf{u}(\mathbf{X}_n, t_n) + \mathbf{u}(\mathbf{X}_{n-1}, t_{n-1})\right)$$

for the corrector. We choose this method because it is currently implemented in the ROMS model (see section 6).

3.2 Analysis Tools

3.2.1 M-function Calculation

Since the M-function is just an arc length, it can be calculated in parallel with, and using the same integration schemes as, the trajectory computation.

3.2.2 Lyapunov Exponent Calculation

The two main aspects of the maximum Lyapunov exponent calculation are approximating the transition matrix L and calculating the eigenvalues of $L^T L$. For the two-dimensional case, we have

$$L(\mathbf{X}(\mathbf{X}_0, t), t) = \begin{bmatrix} \frac{\partial X(\mathbf{X}_0, t)}{\partial X_0} & \frac{\partial X(\mathbf{X}_0, t)}{\partial Y_0} \\ \\ \frac{\partial Y(\mathbf{X}_0, t)}{\partial X_0} & \frac{\partial Y(\mathbf{X}_0, t)}{\partial Y_0} \end{bmatrix},$$

and the idea is to approximate these partial derivatives using finite differences. To that end, we initially lay four particles directly to the left and right of the initial position of interest (with sufficiently small separation ΔX_0) and up and down (separation ΔY_0), calculate their trajectories up to the current time, and approximate L by

$$\begin{bmatrix} \frac{X(X_0+\frac{\Delta X_0}{2},Y_0,t)-X(X_0-\frac{\Delta X_0}{2},Y_0,t)}{\Delta X_0} & \frac{X(X_0,Y_0+\frac{\Delta Y_0}{2},t)-X(X_0,Y_0-\frac{\Delta Y_0}{2},t)}{\Delta Y_0} \\ \\ \frac{Y(X_0+\frac{\Delta X_0}{2},Y_0,t)-Y(X_0-\frac{\Delta X_0}{2},Y_0,t)}{\Delta X_0} & \frac{Y(X_0,Y_0+\frac{\Delta Y_0}{2},t)-Y(X_0,Y_0-\frac{\Delta Y_0}{2},t)}{\Delta Y_0} \end{bmatrix}$$

(A higher-order method could presumably be used here as well, but this was not discussed.) The three-dimensional case is entirely analogous.

Calculating the eigenvalues of $L^T L$ is relatively simple, since it is either a 2×2 or 3×3 matrix, leading to a characteristic polynomial that is either quadratic or cubic. Solving the characteristic equation thus amounts to finding the roots of a quadratic or cubic polynomial. Both of these problems have tractable closed-form solutions (e.g. the quadratic formula in the 2×2 case).

4 Implementation

All programming will be done in MATLAB (currently version 2015b). The initial runs will be on my personal laptop, a MacBook Pro with a 2.6 GHz Intel Core i5 processor and 8 GB of memory. However, calculating trajectories, M-functions, and FTLEs lends itself to massive parallelization (eventually about 800,000 particles in the same flow field), so we plan to eventually move to the Deep Thought 2 cluster on the University of Maryland campus to take advantage of its capacity for parallelization.

5 Validation

To validate our implementation, we will test it on three well-understood dynamical systems: the Duffing oscillator, the Lorenz three-variable system, and Hill's spherical vortex. In each case, we are given a system of differential equations specifying phase-space velocity, which can be discretized to an Arakawa c-grid. Based on these discrete grid values, we will calculate particle trajectories and

subsequently, create M-function and FTLE maps. These trajectories and maps should then match the dynamics of the systems in question.

The Duffing oscillator comes about as a generalization of a harmonic oscillator to the case when the restoring force is nonlinear. The undamped case we will implement here is a common standard for testing in Lagrangian dynamics [3]. Its dynamics look like

$$\begin{aligned}\dot{x} &= y \\ \dot{y} &= x - x^3 + \epsilon \sin t\end{aligned}$$

where ϵ is a parameter that can be set to zero for the unforced case. This case has an analytical solution against which our numerical trajectories can be judged. We will also test a variant of this, the rotating Duffing oscillator[3]:

$$\begin{aligned}\dot{x} &= x\sin(2\beta t) + y(\beta + \cos(2\beta t)) + [-(x\cos(\beta t) - y\sin(\beta t))^3 + \epsilon\sin(\omega t)]\sin(\beta t) \\ \dot{y} &= x(-\beta + \cos(2\beta t)) - y\sin(2\beta t) + [-(x\cos(\beta t) - y\sin(\beta t))^3 + \epsilon\sin(\omega t)]\cos(\beta t).\end{aligned}$$

The Lorenz three-variable system gives rise to the famous Lorenz attractor, a canonical example of chaotic dynamics. Derived from a simple model of the atmosphere, its equations are

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

We will be looking for trajectories that find and follow the Lorenz attractor.

Hill's spherical vortex is a three-dimensional axisymmetric flow field described in spherical coordinates by a streamfunction

$$\psi = -\frac{3}{4}Ur^2\left(1 - \frac{r^2}{a^2}\right)\sin^2\theta$$

where θ is the polar angle (measured from the positive axis of symmetry). From this streamfunction, radial and azimuthal velocities can be generated via

$$\begin{aligned}u_r &= \frac{1}{r^2\sin\theta}\frac{\partial\psi}{\partial\theta} \\ u_\theta &= -\frac{1}{r\sin\theta}\frac{\partial\psi}{\partial r}.\end{aligned}$$

which can in turn be transformed into Cartesian velocities. Again, analytical solutions will be available for comparison, in the form of level sets of the streamfunction. One feature to look for will be the confinement of the flow to planes through the symmetry axis, which we expect to be only approximate due to numerical error.

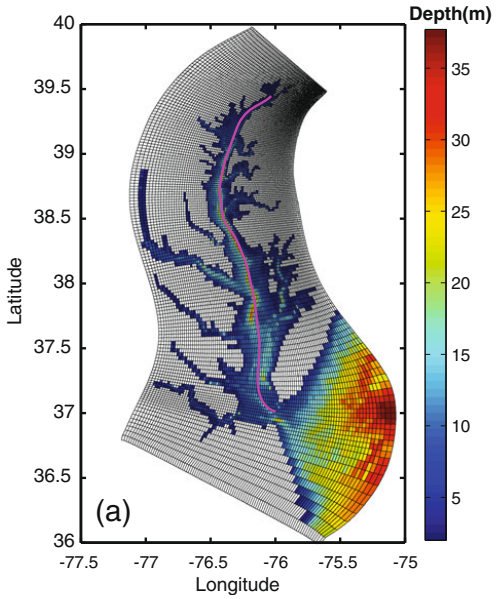

Figure 3: Curvilinear coordinates for Chesapeake ROMS model [4]

6 Testing

We will test our Lagrangian analysis tools on a velocity field generated by a model of the Chesapeake Bay. The model was implemented using ROMS (Regional Ocean Modeling System), a primitive-equations-based ocean modeling platform that can be adopted to various geographic regions [5]. The Chesapeake Bay ROMS model (ChesROMS) uses a curvilinear coordinate system molded to the shape of the bay (figure 3). For simplicity, velocity interpolation will be done in this index space before being transformed back into real space, although the trajectory computation will be done in real space. The irregular boundaries can be handled via a no-slip or free-slip condition.

7 Timeline

- First Semester
  - First half: October - Mid-November
    - Project proposal presentation and paper
    - 2D and 3D interpolation
  - Second half: Mid-November - December
    - 2D trajectory implementation and validation
    - M function implementation and validation

        * Mid-year report and presentation

- Second Semester
  - First half: January - February
    * 3D trajectory implementation and validation
    * FTLE implementation
  - Second half: March - April
    * Visualizations and further analysis
    * Final presentation and paper

8 Deliverables

- Code
  - Routines that lay down particle lattice and calculate trajectories from velocity data
  - Routines that calculate M-function and FTLE based on trajectories
- Results
  - Series of visualizations (images, movies, graphs) based on these functions, for Chesapeake Bay data and test problems
- Reports
  - Project proposal and presentation
  - Mid-year progress report and presentation
  - Final paper and presentation
- Databases
  - Chesapeake Bay ROMS dataset

References

[1] Shadden, S. C., Lekien, F. & Marsden, J. (2005). Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows. *Physica D, 212*, pp. 271-304.

[2] Mendoza, C. & Mancho, A. M. (2010). Hidden geometry of ocean flows. *Physical Review Letters, 105* (038501), pp. 1-4.

[3] Ide, K. & Small, D. & Wiggins, S. (2002). Distinguished hyperbolic trajectories in time-dependent fluid flows: analytical and computational approach for velocity fields defined as data sets. *Nonlinear Processes in Geophysics, 9*, pp. 237-263.

[4] Xu, J. et al. (2012). Climate forcing and salinity variability in Chesapeake Bay, USA. *Estuaries and Coasts, 35*, pp. 237-261.

[5] Regional Ocean Modeling System. (n.d.). Retrieved October 19, 2015, from https://www.myroms.org.

[6] [Online image of Arakawa C-grid box]. Retrieved October 5, 2015 from http://mitgcm.org/sealion/online_documents/node45.html.