AMSC 663 Mid-Year Report: 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 and build a set of Lagrangian analysis tools for an oceanic flow whose velocity is proscribed on a spatio-temporal grid. The main tools 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 modeled 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 jet streams, 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.

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 with initial position $\mathbf{X}_0$, 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 with initial position $\mathbf{X}_0^*$ over the time interval spanning forward and backward time τ from the current time t^*. Sometimes we will be interested in looking only forward or backward in time, and will define M accordingly by

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

or

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

The choice should be clear from the context.

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, it is likely to lie within one coherent structure. Conversely, if it finds itself getting massively stretched along some axis (as time runs either forward or backward), it is likely to lie on the boundary between two coherent structures. Thus we need a way to

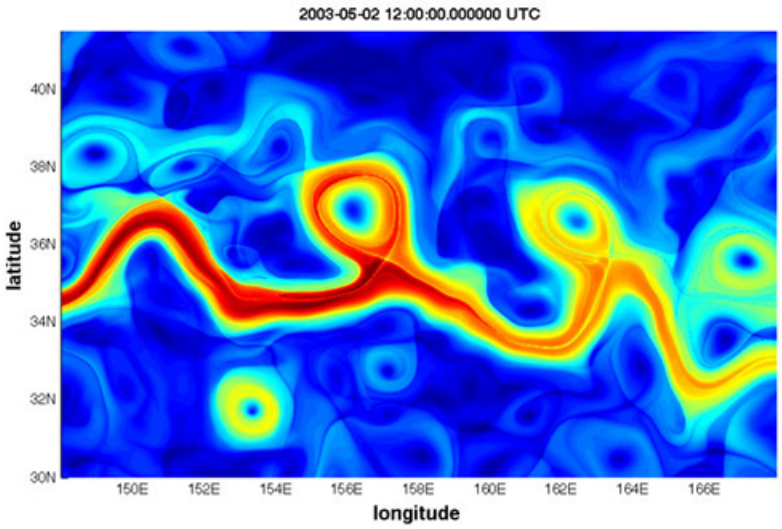

Figure 1: M-function coloring for a dataset from the Kuroshio current [2]. Red represents fast-moving regions, blue slow. Stable and unstable manifolds appear as thin yellow lines.

quantify the degree to which two nearby trajectories diverge in time. To that end, we introduce the maximal finite-time Lyapunov exponent (FTLE):

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

where ρ denotes the spectral radius and

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

is the so-called transition matrix at time t (the Jacobian of current position with respect to initial position). An equivalent definition of λ is that, for the locally linearized flow field, it represents the exponential growth rate of an infinitesimal fluid parcel along its direction of maximal stretching.

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, jet streams, 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, in order to investigate the speed vs. accuracy tradeoff often inherent in deciding between those alternatives.

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

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

There are two parts to applying this formula to Eulerian velocity data. First, if the velocity components are given only on a spatio-temporal grid, e.g. $u(x_i, y_j, z_k, t_l) = u_{i,j,k,l}$, 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 the given data for u, v, and w are staggered within each grid box, so that, u is

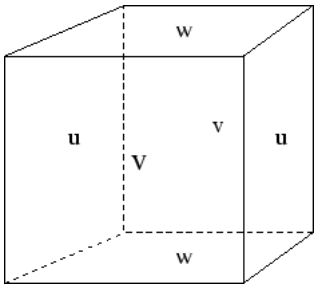

Figure 2: Arakawa c-grid box [6]

given at the centers of the west and east faces, v north and south, and w top and bottom (figure 2). Each of these scalar components will be interpolated independently to the location of each particle. For simplicity, the 3D interpolation will be broken into a 2D horizontal interpolation followed by 1D vertical interpolation. Temporal interpolation also is treated as a 1D problem following spatial interpolation.

The simplest way to interpolate horizontally is to use bilinear splines. A bilinear spline is a function of the form

$$f(x,y) = \sum_{i,j=0}^{1} c_{ij} x^i y^j = c_{00} + c_{10}x + c_{01}y + c_{11}xy$$

defined on a single grid box, where the parameters $c_{00}, c_{10}, c_{01}, c_{11}$ are chosen so that the values of f at the corners match those of the given function. In other words, given data u, the interpolant f on $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$ must satisfy

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

This simple scheme has $O(h^2)$ error for $h = \Delta x = \Delta y$, and is only C^0, with discontinuous derivatives at the box edges.

A more accurate method is to use bicubic splines. In this case, we approximate u by a bicubic function

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

within each box. To solve for the sixteen unknowns c_{ij}, we must fit not only to the given values of u at the four corners, but also to estimated values of the partial derivatives $\partial_x u$, $\partial_y u$, and $\partial_{xy} u$ at those corners. This method is known to be $O(h^4)$ under certain conditions.

For the 1D vertical interpolation problem, we compare two methods: linear splines and cubic polynomials interpolated through the nearest four vertical neighbors. That is, given $z \in [z_k, z_{k+1}]$ and 2D-interpolated values $u_k = u(x, y, z_k)$, $u_{k+1} = u(x, y, z_{k+1})$, etc., we approximate $u(x, y, z)$ using a linear spline by

$$f(z) = w \cdot u_k + (1 - w) \cdot u_{k+1}$$

where

$$w = \frac{z_{k+1} - z}{z_{k+1} - z_k}$$

which is $O(h^2)$, or, for better accuracy, using a cubic polynomial through the nearest four vertical neighbors, by

$$\begin{aligned} f(z) = &\frac{(z - z_k)(z - z_{k+1})(z - z_{k+2})}{(z_{k-1} - z_k)(z_{k-1} - z_{k+1})(z_{k-1} - z_{k+2})} u_{k-1} \\ &+ \frac{(z - z_{k-1})(z - z_{k+1})(z - z_{k+2})}{(z_k - z_{k-1})(z_k - z_{k+1})(z_k - z_{k+2})} u_k \\ &+ \frac{(z - z_{k-1})(z - z_k)(z - z_{k+2})}{(z_{k+1} - z_{k-1})(z_{k+1} - z_k)(z_{k+1} - z_{k+2})} u_{k+1} \\ &+ \frac{(z - z_{k-1})(z - z_k)(z - z_{k+1})}{(z_{k+2} - z_{k-1})(z_{k+2} - z_k)(z_{k+2} - z_{k+1})} u_{k+2}. \end{aligned}$$

which is $O(h^4)$ (although it is not C^1, as it does not match derivatives, unlike cubic splines). At the vertical boundaries, the nearest four neighbors will not be symmetric about the query point, so that, for example, the interpolating polynomial will be the same for the upper two layers.

Finally, the 1D temporal interpolation problem will be treated using the same two methods as the 1D vertical problem.

3.1.2 Time Integration

For velocity integration, we use only explicit methods for simplicity. A well-known high-order multistage method is the famous Runge-Kutta fourth-order (RK4) scheme:

$$\begin{aligned} \mathbf{k}_1 &= \Delta t \cdot \mathbf{u}\left(\mathbf{X}_n, t_n\right) \\ \mathbf{k}_2 &= \Delta t \cdot \mathbf{u}\left(\mathbf{X}_n + \frac{\mathbf{k}_1}{2}, t_n + \frac{\Delta t}{2}\right) \\ \mathbf{k}_3 &= \Delta t \cdot \mathbf{u}\left(\mathbf{X}_n + \frac{\mathbf{k}_2}{2}, t_n + \frac{\Delta t}{2}\right) \\ \mathbf{k}_4 &= \Delta t \cdot \mathbf{u}\left(\mathbf{X}_n + \mathbf{k}_3, t_n + \Delta t\right) \\ \mathbf{X}_{n+1} &= \mathbf{X}_n + \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right). \end{aligned}$$

We also plan to implement the Milne-Hamming scheme, which is a stable fourth-order multistep 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

The M-function 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 (i) approximating the transition matrix L and (ii) 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 programs are written in MATLAB (version 2015b). When possible, code will be run on a MacBook Pro with a 2.6 GHz Intel Core i5 processor and 8 GB of memory. However, this is expected to become prohibitive for an eventuality of about 10^6 particles in the testing problem, so we plan to do those runs on the Deep Thought 2 cluster on the University of Maryland campus. The algorithms involved lend themselves to massive parallelization, since each trajectory, M-function, etc. can be computed independently of the others. Particle locations are stored in column vectors, and any function of the particles (e.g. their grid box indices) is stored in a vector of the same size. A prevailing design concept was to avoid looping through particles by vectorizing all operations.

Bilinear interpolation of $u(x, y)$ on $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$ was implemented explicitly via the formula

$$\begin{aligned} f(x,y) = w_y \left(w_x u_{i,j} + (1-w_x)u_{i+1,j}\right) \\ + (1-w_y)\left(w_x u_{i,j+1} + (1-w_x)u_{i+1,j+1}\right) \end{aligned}$$

with weights

$$\begin{aligned} w_x &= \frac{x_{i+1} - x}{x_{i+1} - x_i} \\ w_y &= \frac{y_{j+1} - y}{y_{j+1} - y_j}. \end{aligned}$$

In implementing these formulas, x and y become column vectors with the particle locations, and the only tricky part is finding the box indices i and j corresponding to each particle. This was simply accomplished using the `histcounts` function in MATLAB.

Bicubic interpolation on $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$ was performed by scaling the problem to $[0, 1]^2$, doing the interpolation there, and scaling back. The coefficients in

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

can be determined from the values of u and its derivatives by

$$\begin{bmatrix} c_{00} \\ c_{10} \\ c_{20} \\ c_{30} \\ c_{01} \\ c_{11} \\ c_{21} \\ c_{31} \\ c_{02} \\ c_{12} \\ c_{22} \\ c_{32} \\ c_{03} \\ c_{13} \\ c_{23} \\ c_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\ 9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\ -6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\ 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ -6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\ 4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} u(0,0) \\ u(1,0) \\ u(0,1) \\ u(1,1) \\ u_x(0,0) \\ u_x(1,0) \\ u_x(0,1) \\ u_x(1,1) \\ u_y(0,0) \\ u_y(1,0) \\ u_y(0,1) \\ u_y(1,1) \\ u_{xy}(0,0) \\ u_{xy}(1,0) \\ u_{xy}(0,1) \\ u_{xy}(1,1) \end{bmatrix},$$

although this system of equations was hard-coded to take into account the column-vector nature of $u(0,0)$, $u(1,0)$, etc. The derivatives in question were approximated using the second-order finite difference approximations. For interior grid boxes, we used the centered differences

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

while for grid boxes bordering an edge of the domain, these formulas were replaced by one-sided differences. For example, when $i = 1$ we used the second-order approximations

$$\begin{aligned}\partial_x u_{1,j} &\approx \frac{1}{2\Delta x}\left(-3u_{1,j} + 4u_{2,j} - u_{3,j}\right) \\ \partial_{xy} u_{1,j} &\approx \frac{3u_{1,j-1} - 3u_{1,j+1} - 4u_{2,j-1} + 4u_{2,j+1} + u_{3,j-1} - u_{3,j+1}}{4\Delta x \Delta y},\end{aligned}$$

and when $i = j = 1$, we used

$$\partial_{xy} u_{1,1} \approx \frac{9u_{1,1} - 12u_{1,2} + 3u_{1,3} - 12u_{2,1} + 16u_{2,2} - 4u_{2,3} + 3u_{3,1} - 4u_{3,2} + u_{3,3}}{4\Delta x \Delta y}.$$

Again, some care had to be taken to ensure that the operations remained vectorized. For example, because of the different formulas for boundary boxes and interior boxes, we had to be able to grab one set of particle indices at a time—those in the interior, or on the left boundary, or the top boundary, etc.

5 Validation

To validate our interpolation schemes, we apply them to the simple analytic function $u(x, y) = e^y \sin(\pi x)$. That is, we discretize this function on a grid, apply the schemes to this grid data, and verify that the error goes to 0 as the grid spacing goes to 0. In fact, for grid spacing $h = \Delta x = \Delta y$, we expect the maximum error to be $O(h^2)$ for bilinear and $O(h^4)$ for bicubic. The results are shown in figure 3. Using the two finest grids, the estimated order of accuracy p in $\|u - f\|_2 = O(h^p)$ is $p \approx 3.0029$ for bilinear and $p \approx 4.3569$ for bicubic, both somewhat higher than expected.

The computational times of these figures are also compared in figure 4. As expected, bicubic interpolation takes significantly longer (roughly 10-20 times), and the time grows roughly linearly with the number of grid points.

To validate our trajectory computation, we test it on some well-understood nonlinear dynamical systems: the Duffing oscillator (2D) and Hill's spherical vortex (3D), and, time permitting, on the rotating Duffing oscillator (2D) and

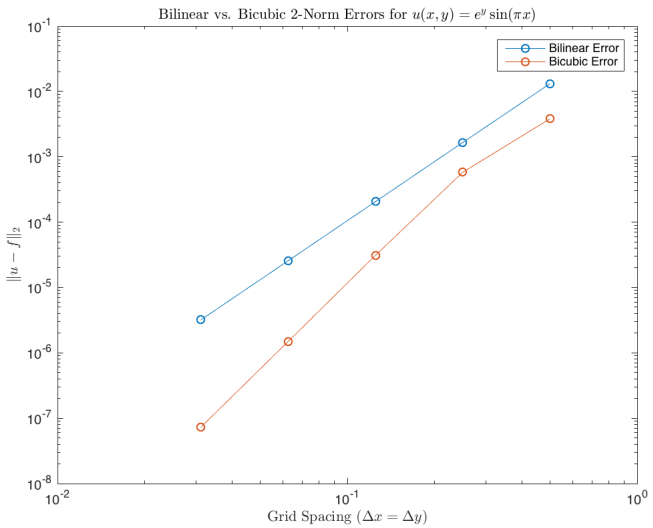

Figure 3: Convergence of bilinear and bicubic interpolation

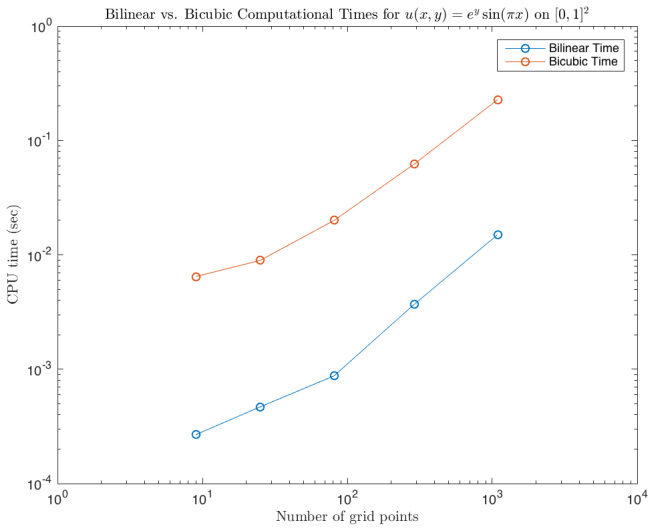

Figure 4: Time of bilinear and bicubic interpolation

Lorenz three-variable system (3D). In each case, we discretize phase-space velocity to an Arakawa c-grid, calculate particle trajectories from this data, create M-function and FTLE maps, and compare these trajectories and maps to the known dynamics of the systems in question, quantitatively and visually.

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 equations are

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

where ϵ is a forcing parameter. The unforced case ($\epsilon = 0$) is an autonomous system with an analytical solution (in terms of elliptic functions) against which our numerical trajectories can be judged. Another feature to look for is the closure of the trajectories, arising from the fact that this represents a potential flow. Finally, the stable and unstable manifolds form S-shapes around the origin, so we look for the M-function to reveal these shapes at the right time scale.

Figure 5 shows exact trajectories coming from the formula

$$y = \pm\sqrt{\left(x^2 - \frac{x^4}{2}\right) - \left(x_0^2 - \frac{x_0^4}{2}\right)}$$

for the trajectory beginning at $(x_0, 0)$, with the stable/unstable figure-eight manifold ($x_0 = \pm\sqrt{2}$) highlighted. For comparison, numerically computed trajectories using the same initial conditions are shown below, using RK4, bilinear interpolation, $\Delta x = \Delta y = 0.2$, and $\Delta t = 0.02$. Clearly, the overall picture is correct, although there is some visible error for trajectories close to the manifold. One possible direction to go from here would be to check that error goes to zero as $\Delta x \to 0$, both visually and numerically. For the numerical part, it will be useful to (eventually) implement the exact trajectories as analytical functions of t, which is possible using Jacobi elliptic functions, namely [9]:

$$\begin{aligned}x(t) &= c_1 \mathrm{cn}\left(t\sqrt{c_1^2 - 1} + c_2, m\right) \\ y(t) &= -c_1\sqrt{c_1^2 - 1} \cdot \mathrm{sn}\left(t\sqrt{c_1^2 - 1} + c_2, m\right) \mathrm{dn}\left(t\sqrt{c_1^2 - 1} + c_2, m\right)\end{aligned}$$

where

$$m = \sqrt{\frac{c_1^2}{2(c_1^2 - 1)}}$$

and c_1, c_2 come from x_0, y_0 via

$$\begin{aligned}c_1 \mathrm{cn}(c_2, m) &= x_0 \\ -\sqrt{c_1^2 - 1} \cdot \mathrm{sn}(c_2, m)\mathrm{dn}(c_2, m) &= y_0.\end{aligned}$$

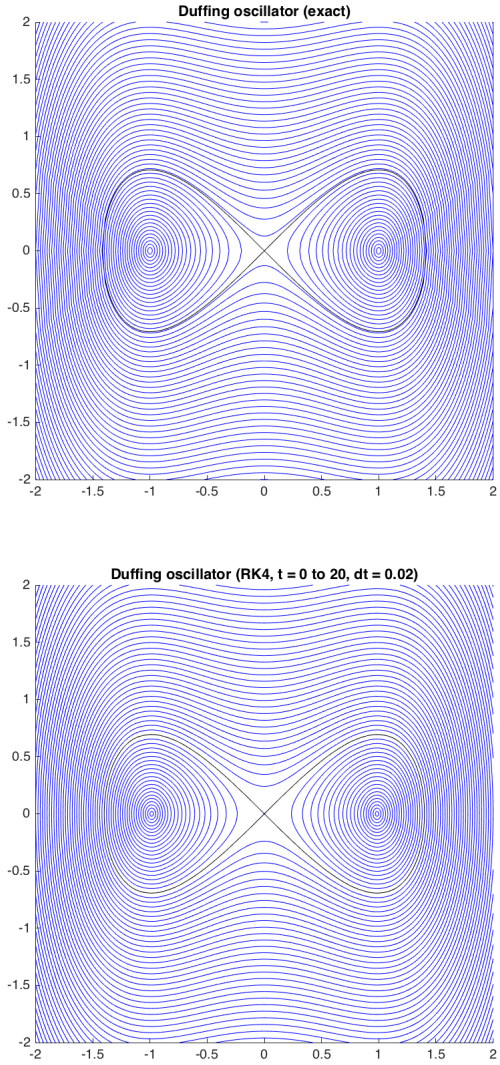

Figure 5: Exact (top) and approximate (bottom) trajectories for the Duffing oscillator, using the same initial data. The stable/unstable manifold has been colored black in both cases (in the second, as an approximate trajectory).

Once this is implemented, we can calculate the root-mean-square error of a trajectory up to a given time and see that this goes to zero as $\Delta x \to 0$.

Figure 6 shows a colored graph of the M-function, using a data-grid spacing of $\Delta x = 0.2$ and a particle-grid spacing of one-tenth of that. In the top figure, the M-function is calculated over the interval $t = 0$ to 6 but the graph is colored by the particle's position at $t = 0$. In the bottom figure, the integration is run backwards in time over the same time interval. The choice of $t = 6$ was motivated by this being the right timescale to bring out the stable manifold (top, blue) and unstable manifold (bottom, blue) as separate entities. It's worth noting that all particles move counterclockwise around the elliptic fixed points $(-1, 0)$ and $(1, 0)$ so that, as expected, they approach the hyperbolic fixed point at $(0, 0)$ along the stable manifold and exit along the unstable one.

Time permitting, we will also validate on a variant of the Duffing oscillator, the rotating Duffing oscillator[3]:

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

Hill's spherical vortex (figure 7) 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 z-axis). From this streamfunction, radial and azimuthal velocities can be generated via

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

which, after some manipulation, gives Cartesian velocities

$$u = -\frac{3Uxz}{2a^2}$$
$$v = -\frac{3Uyz}{2a^2}$$
$$w = \frac{3U(2x^2 + 2y^2 + z^2 - a^2)}{2a^2}$$

The fixed points of this flow are $(0, 0, -a)$ and $(0, 0, a)$, and the stable and unstable manifolds are the sphere of radius a and center the origin, and pieces of the line $x = y = 0$ lying inside and outside the sphere. Again, analytical solutions will be available for comparison, in the form of level sets of the streamfunction. We will look for (potentially) three-dimensional trajectories to

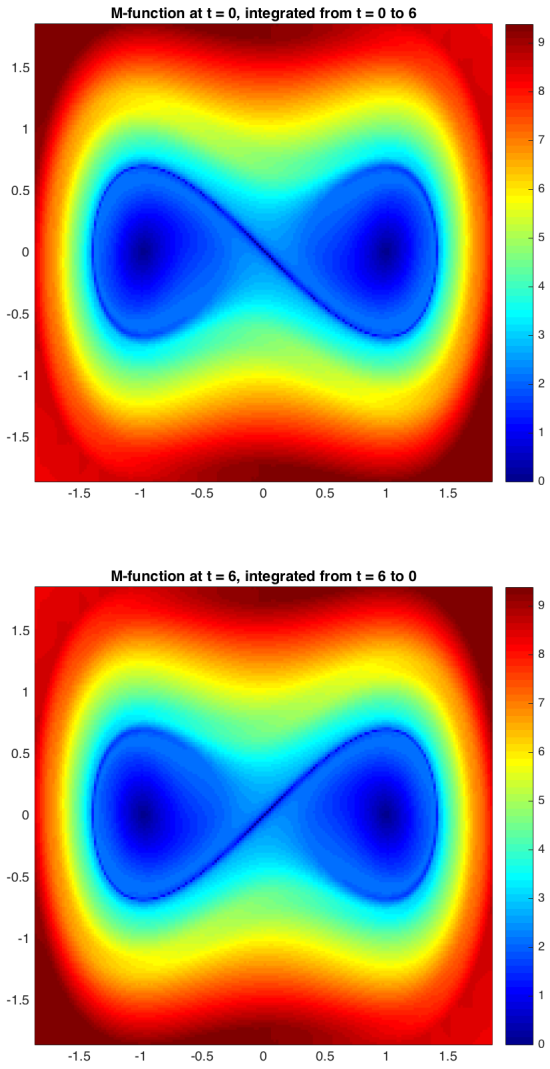

Figure 6: M-function for Duffing oscillator, integrated between $t = 0$ and $t = 6$ (top, integrated forward and pictured at $t = 0$; bottom, integrated backward and pictured at $t = 6$).

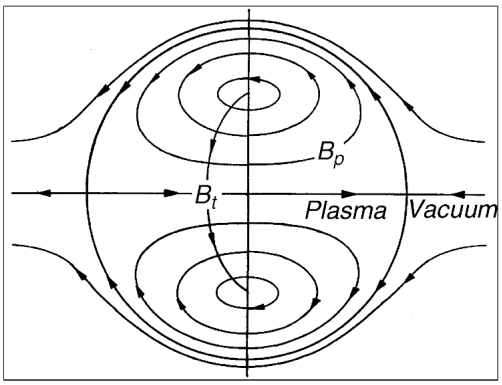

Figure 7: Hill's spherical vortex [10]

be confined to a plane through the origin (as is required of axisymmetric flows), as well as for closure of those trajectories lying within the sphere. Eventually we will plot cross-sections of the M-function and look for them to clearly bring out the known manifolds.

Time permitting, we will also validate on the Lorenz three-variable system, 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 to find and follow the Lorenz attractor. We may also plot cross-sections of the M-function to see if they can bring out the attractor.

6 Testing

We will test our Lagrangian analysis tools on a velocity field generated by a model of the Chesapeake Bay. The model is an implementation of 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 8). 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. Eventually our methods will have to be adapted to allow for land-water boundaries within the domain, as is clear from the figure. To ensure that particles do not exit the water domain, a no-slip or free-slip boundary condition must be implemented.

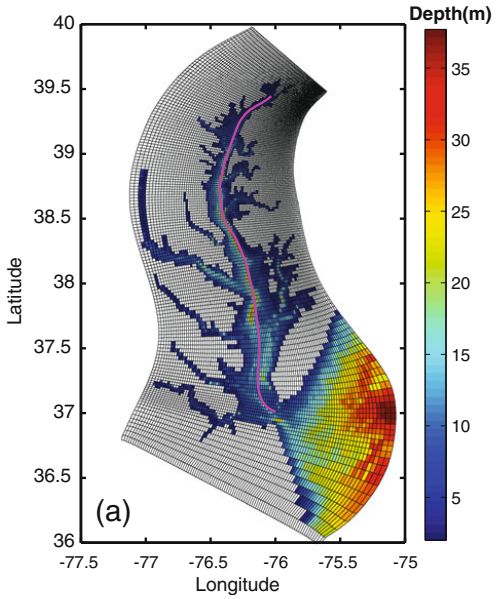

Figure 8: Curvilinear coordinates for Chesapeake ROMS model [4]

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
    - Tailor all existing code to work with ROMS data
  - Second half: March - April
    - Apply tools to Chesapeake dataset
    - Do deeper analysis based on FTLE and M-function
    - 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.

[7] Bilinear interpolation. (2015, November 21). In Wikipedia, The Free Encyclopedia. Retrieved 06:11, December 13, 2015, from https://en.wikipedia.org/w/index.php?title=Bilinear_interpolation&oldid=691695764.

[8] Bicubic interpolation. (2015, November 13). In Wikipedia, The Free Encyclopedia. Retrieved 06:42, December 13, 2015, from https://en.wikipedia.org/w/index.php?title=Bicubic_interpolation&oldid=690508277

[9] Salas, A. H. & Castillo H., J. E. (2014) Exact solution to Duffing equation and the pendulum equation. *Applied Mathematical Sciences, 8* (176) pp. 8781 - 8789.

[10] [Online image of Hill's spherical vortex]. Retrieved October 5, 2015 from http://www.physics.ucla.edu/plasma-exp/conferences/TopicalConferences/IPELS97/Spheromak.html.