U.S.N.A.---Trident Scholar project report; no. XXX (2005)

TRAJECTORY AND INVARIANT MANIFOLD COMPUTATION FOR FLOWS IN THE CHESAPEAKE BAY

by

Midshipman 1/c Nathan F. Brasher, Class of 2005
United States Naval Academy
Annapolis, MD

_____
(signature)

Certification of Advisers Approval

Professor Reza Malek-Madani
Mathematics Department

_____
(signature)

_____
(date)

Assoc. Professor Gary O. Fowler
Mathematics Department

_____
(signature)

_____
(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research and Scholarship

_____
(signature)

_____
(date)

USNA-1531-2

Abstract

The field of mathematics known as dynamical systems theory has seen great progress in recent years. A number of techniques have been developed for computation of dynamical systems structures based on a data set of a given flow, specifically Distinguished Hyperbolic Ttrajectories (DHTs) and their invariant manifolds.

In this project, algorithms in MATLAB® have been successfully implemented and applied to a number of test problems, as well as to the Chesapeake Bay flow data generated by the QUODDY shallow-water finite-element model. A number of interesting discoveries have been made including instabilities of convergence of the DHT algorithm and evidence of lobe dynamics in the Chesapeake.

Additionally, MATLAB® code has been developed to compute Synoptic Lagrangian Maps (SLMs). When applied to an oceanographic flow, SLMs produce plots of the time that it takes particles in various regions to encounter the coast or escape to the open ocean. Such maps are of interest to the oceanography community. A new algorithm for SLM computation has been developed resulting in orders of magnitude increase in efficiency. Previously SLM computation for a week of flow data was a problem limited to massively parallel supercomputers. With the new algorithm, similar data is computed in a few days on a single processor machine.

The development of platform-independent MATLAB® implementation of the algorithms for computation of DHTs, invariant manifolds and SLMs should prove valuable tools for studying the dynamics of complex oceanographic flows.

Keywords: Chesapeake Bay, dynamical systems theory, hyperbolic trajectory, invariant manifold, synoptic Lagrangian map

Acknowledgements

Most importantly I must thank my advisers, Prof. Malek-Madani and Assoc. Prof. Fowler. They set me on course with a fantastic project and have kept me honest ever since.

Thanks also to Prof. Nakos for his QUODDY implementation.

Thanks to NOAA and Dr. Gross for his work modeling the Chesapeake Bay.

Thanks to Mr. Spinks and Mr. Disque from the CADIG labs for their support and instruction as I struggled to administer my own Linux domain along with the project.

Thanks to Prof. Shade and the entire Trident Scholar Committee for their hard work and dedication.

Thanks to Prof. Pierce, Prof. Baker, Asst. Prof. Popovici and Asst. Prof. Liakos for many interesting conversations and ideas and thanks to the entire Chesapeake Bay Seminar for listening to my talks.

INTRODUCTION **4**

BACKGROUND INFORMATION **7**

TRAJECTORY COMPUTATION **10**

- **PART I: METHODOLOGY** 10
- **PART II:RESULTS** 16

SYNOPTIC LAGRANGIAN MAPS **22**

- **PART I: METHODOLOGY** 22
- **PART II: RESULTS** 31

DHT COMPUTATION **40**

- **PART I: METHODOLOGY** 40
- **PART II: RESULTS** 45

INVARIANT MANIFOLD COMPUTATION **52**

- **PART I: METHODOLOGY** 52
- **PART II: RESULTS** 56

CONCLUSIONS **64**

REFERENCES **66**

APPENDIX: MATLAB CODE **68**

Introduction

10,000 years ago, at the dawn of the Holocene epoch, the first waters from melting glaciers reached the mouth of what is today the Chesapeake Bay. For the next several thousand years, as the glaciers retreated, the Susquehanna River carved out a valley. Slowly, as the sea levels rose, river beds and streams were submerged. By three thousand years ago what we know as the Chesapeake Bay had assumed its present form.

Today, a myriad of physical processes continue to shape and reshape the Bay. Erosion and sedimentation constantly adjust the coastline. Freshwater from the rivers mixes with saltwater from the Atlantic Ocean, creating a salinity gradient which allows literally thousands of species to interact. Nutrients, dissolved gases, phytoplankton and other drifters circulate with the tides and currents supporting the diverse ecology present. Each aspect and organism in this ecology is intricately linked with the others. Any change to the environment of the Bay impacts all of them.

Of all the species that inhabit the Chesapeake Bay and its surrounding environment, it is humans that have the greatest impact. Industrial waste, toxic chemicals and other pollutants directly threaten water quality. Excess nitrogen and other nutrients may create algae blooms, blocking sunlight and negatively affecting underwater vegetation that many species use for shelter and as spawning grounds. To best understand how to preserve this resource, it is important to understand how our actions impact and propagate through the Chesapeake Bay system.

This project comes as the result of a year-long study of the Chesapeake as a dynamical system. Some of the basic questions investigated include: If material follows the currents, where will it go and over what time frame? How long will that material remain in the Bay given the

coordinates of its origin? How can we better characterize the mechanics of the flow using the tools of applied mathematics? To answer these questions, three areas of results were produced: trajectory computation, computation of Synoptic Lagrangian Maps (SLMs) and the computation of Distinguished Hyperbolic Trajectories (DHT) and their respective invariant manifolds.

Trajectory computation answers the first question of where and over what time frame material travels. Based on fluid flow data generated by the QUODDY computer model, MATLAB code was written that used a 4th order Runge-Kutta method to track the path of a particle adrift with the currents. Other aspects of this work required that the project be a strictly two-dimensional problem. Because of this requirement, the computation analyzed only the surface currents. These trajectories, then, track the motion of material which remains on the surface and does not penetrate to deeper layers.

To answer questions of how long material remains in the Bay, the ideas of Lipphardt and Kirwan [10] on synoptic Lagrangian maps were applied to the Chesapeake. The computation of SLMs produces plots of total residence time vs. starting time and location. SLMs are primarily useful as a visualization tool. By summarizing data from hundreds of thousands of trajectories, it is possible to glean more useful information from these plots than it is from observing current fields or drifter trajectories. I was able to contribute to this field by developing an algorithm for the computation of SLMs up to 60 times faster than was previously possible.

Finally, in order to facilitate a deeper understanding of the mechanics of the flow, the work of Wiggins [1] on computation of hyperbolic trajectories and their associated invariant manifolds was applied to the Chesapeake Bay. Computing these structures for the Chesapeake system makes it possible to apply years of research in dynamical systems theory to this work. In the process of this investigation, it was discovered that flow interaction with complex boundaries

can develop these structures. Previously it was thought that large scale gyres and hyperbolic separation points were necessary for such manifolds to exist. Neither of these are present in the Chesapeake Bay. This result shows that the class of problems for which invariant manifold analysis is applicable is larger that previously thought. In the future this may lead to new methods of analyzing complex flow, particularly in littoral environments similar to the Chesapeake Bay.

The next sections of this report address the relevant background information as well as the methodology and results produced for each of the aforementioned aspects of the project. Additionally, new avenues of research are suggested that have arisen as a direct result of this work.

Background Information

In 1992, at the request of the Office of Naval Research, several mathematicians began meeting with oceanographers at Woods Hole Oceanographic Institute. The objective was to investigate the similarities between an oceanographer's approach to studying particle transport in the ocean and the mathematical field known as dynamical systems. In the coming years this collaboration would produce algorithms for applying a mathematical dynamical systems understanding to the ocean. The work of this group on DHTs [4] and invariant manifolds [2] formed the backbone of this project.

The first application of this group's research was a study of transport in the Monterrey Bay, located on the Pacific coast, 100 miles south of San Francisco. Using surface velocity data captured by CODAR (Coastal Ocean Dynamics Application Radar), and extrapolated using the technique of normal mode analysis [11], Wiggins, Lipphardt and Kirwan analyzed the flow for invariant manifolds and associated structures in the flow. The analysis technique known as lobe dynamics (refer to [7]) showed promise detecting filaments, jets and other methods of material transport. Later, Kirwan and Lipphardt would develop the idea of synoptic Lagrangian Maps as summaries of particle transport behavior. The full history of this collaboration is described in [15].

The aim of this project was to take the work of Wiggins, Kirwan, Lipphardt and others, and apply these algorithms to the Chesapeake Bay. The Chesapeake presented a number of unique challenges, mostly because of its complex geometry. The domains that had been analyzed in the past by typically had simple, concave boundaries as in the case of the Monterey Bay.

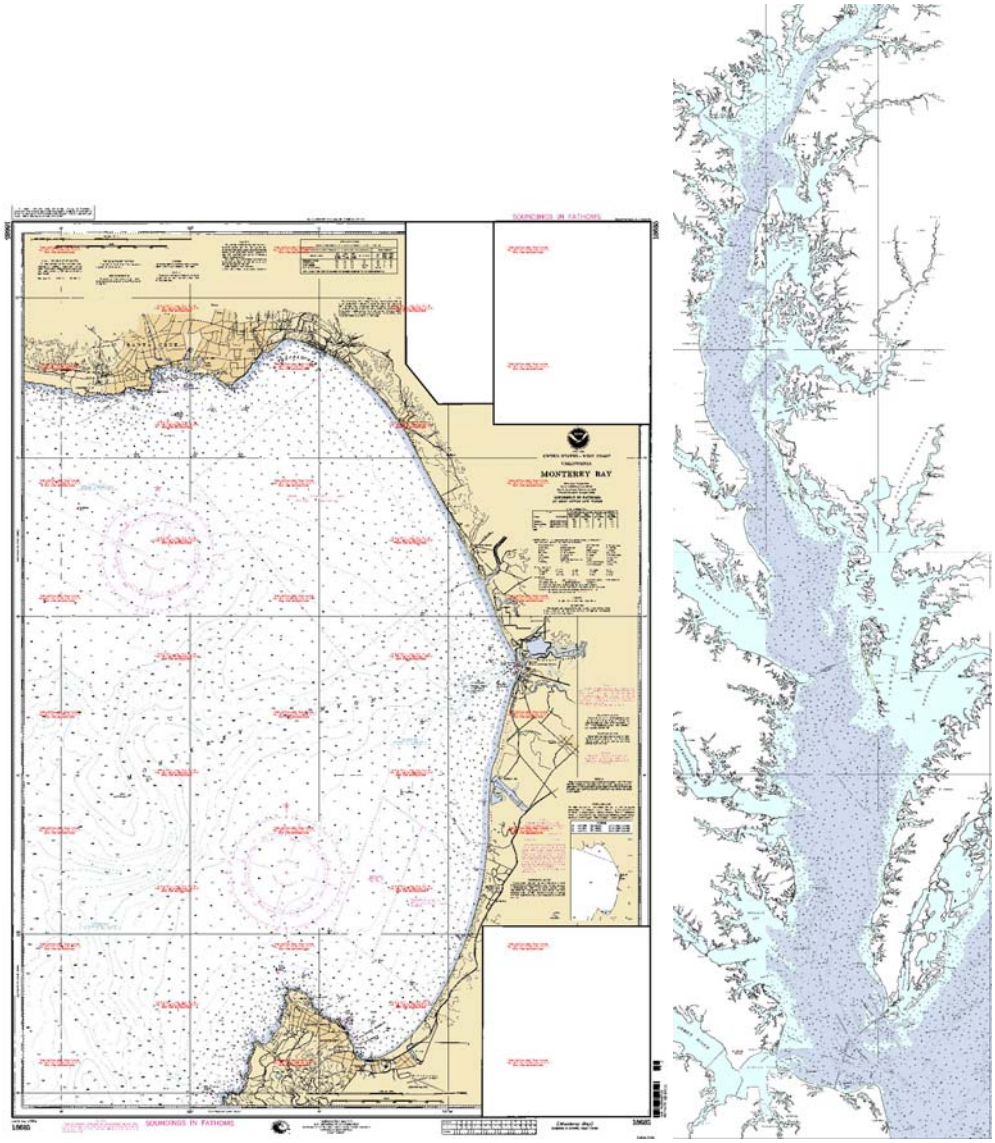


Figure 1: Boundary geometry comparison, Monterey Bay chart on the left, Chesapeake on the right, images courtesy of NOAA.

The Chesapeake Bay has a long, complex, non-convex coastline that makes issues of particle tracking more difficult. Additionally, the real-time surface velocity data used on the Monterey Bay is not available for the Chesapeake.

In lieu of experimental data, the shallow-water finite-element QUODDY model was used to generate velocity data. QUODDY is a robust ocean model developed by Lynch et al. [9] and

tested on many different coastal domains. The model has consistently shown good agreement with experimental data and is widely used in the ocean modeling community. In collaboration with NOAA, a 9700 node, 17000 element mesh was generated allowing QUODDY to track velocity, salinity, temperature and pressure in 3D. Since this work was strictly 2D, the the top layer of the data set, corresponding to surface currents, was analyzed.

QUODDY solves the shallow water wave equation on its mesh of linearly interpolated triangular elements.

$$\frac{\partial^2 H}{\partial t^2} + \tau_o \frac{\partial H}{\partial t} - \nabla_{xy} \cdot \left(\nabla_{xy} \cdot \left(H\overline{\mathbf{v}}\overline{\mathbf{v}} \right) + gH\nabla_{xy}\varsigma + \mathbf{f} \times H\overline{\mathbf{v}} - \tau_o H\overline{\mathbf{v}} - H\mathbf{\Psi} + H\mathbf{\Gamma} - H\overline{\mathbf{R}} \right) = 0. \quad (1)$$

In this equation an over-bar indicates vertical averaging and the symbol ∇_{xy} is the Laplacian differentiated with respect to x and y only, the vertical dimension is suppressed. The symbol ζ denotes free surface height, a function that varies with x, y and t while H denotes the total water column height, that is, the depth plus the surface height ζ. The term $\mathbf{f} \times H\overline{\mathbf{v}}$ comes from the Coriolis force. $H\mathbf{\Psi}$ and $H\mathbf{\Gamma}$ represent the surface wind stress and bottom frictional stress respectively. The wind stress $\mathbf{\Psi}$ is derived from wind measurements taken at stations along the coast of the Bay. Source terms generated by experimental flow measurements are included for all of the major tributaries, including the Susquehanna, Patuxent, Potomac, Rappahannock, James and York rivers. Finally, tidal forcing at the Atlantic interface is included in the model; this forcing is what drives much of the flow throughout the domain.

TRAJECTORY COMPUTATION

PART I: Methodology

The first step to analyzing surface flow in the Chesapeake Bay is to compute trajectories, that is, the position as a function of time of a drifter buoyant enough to remain on the surface. This amounts to solving the differential equation $\dot{\mathbf{x}} = \mathbf{u}(\mathbf{x}, t)$, where $\mathbf{x}$ indicates position and $\mathbf{u}$ indicates velocity, in this case the velocity field output by QUODDY.

To solve this differential equation numerically, 4th order Runge-Kutta, a standard solution technique was used. Early on in the project, when computing trajectories of systems for which the analytical solution was already known, this method was chosen in order to compare the results with MATLAB's ode45. The ode45 function is one of the most well-known and accurate differential equation solvers available. It uses a mix of 4th and 5th order Runge-Kutta combined with an adaptive time step method. This capability made it a good tool with which to evaluate the accuracy of interpolation and trajectory results of the non-adaptive 4th order Runge-Kutta. Unfortunately, in the case of the Chesapeake, the velocity data is available only at discrete time steps, so the adaptive time step method is not useful. A fixed time step, 4th order method was therefore used in the place of ode45.

The real challenge, however, to trajectory computation was that the velocity field is specified only on discrete points. The trajectories that need to be calculated are not generally, if ever, located exactly on these points. Thus, a spatial interpolation method was needed. The simplest method of interpolation would be to use the linear-interpolation of the QUODDY finite-element mesh itself. Each point in the triangular element is linearly interpolated based on the three surrounding nodes. In this case the resulting solution is C^0, that is, continuous everywhere

but not generally differentiable. Unfortunately, both the distinguished hyperbolic trajectory and invariant manifold algorithms due to Wiggins et al. [4] require the calculation of spatial partial derivatives of the velocity field. A linear method would not suffice as the partial derivatives would be discontinuous at each node and nodal line on the QUODDY mesh. Therefore, in order to obtain a smoother interpolation, the method of Radial Basis Functions was chosen. This method works as follows. Given a set of N data points $\{x_i,y_i\} \in \mathbf{R}^2$ and a set of values z_i at each point, a smooth surface denoted by $z = f(x,y)$ is fit to the points as a sum of radially-symmetric basis functions centered at each point.

$$f(x,y) = \sum_{i=1}^{N} c_i \phi(x - x_i, y - y_i). \qquad (2)$$

Functions that are typically used for ϕ include the Hardy Multivariate function $\phi(x,y) = \sqrt{x^2 + y^2 + k}$, where k is some constant, and the well-known Gaussian $\phi(x,y) = e^{-k(x^2+y^2)}$. It is easy to see that because these basis functions are each C^∞, the resulting interpolation surface will also be C^∞. All that remains is to solve for the coefficients c_i. This is done by evaluating (2) at each of the data points resulting in

$$f(x_j, y_j) = z_j = \sum_{i=1}^{N} c_i \phi(x_j - x_i, y_j - y_i). \qquad (3)$$

Equation (3) can then be cast into matrix form

$$\mathbf{Gc} = \mathbf{z}, \qquad (4)$$
$$\mathbf{G}_{ij} = \phi(x_j - x_i, y_j - y_i).$$

When applied to the QUODDY output using either of the C^∞ functions mentioned above, the resulting matrix **G** is dense and has 9700x9700 elements. Even with fast MATLAB

algorithms, solving equation (4) would take a tremendous amount of time and computing resources. Given that this interpolation would need to be repeated for each time step in the QUODDY output, this was unacceptable. However, by easing the requirement that the basis functions be C^{∞}, and cutting off the functions beyond a given radius (the so-called radius of support) the matrix **G** could be made sparse, a much more tractable problem. The compactly-supported radial basis functions of Wendland [12] do exactly that.

$$\begin{aligned} W_2^0 &= (1-r)_+^2 \in C^0, \\ W_2^2 &= (1-r)_+^4 (4r+1) \in C^2, \\ W_2^4 &= (1-r)_+^6 (35r^2 + 18r + 3) \in C^4. \end{aligned} \tag{5}$$

The superscript on the W indicates the degree of continuity and the subscript indicates the dimension. Incidentally, the Wendland functions for 3D have the same form as those for 2D. It can be shown that equation (4) is non-singular when using these functions given that the data points $\{x_i, y_i\}$ are distinct. For a proof of this, refer to [12]. Below is an illustration of the first three Wendland functions described above. The surface plotted is $z = W_2^n(x, y)$ where is n=0,2,4.

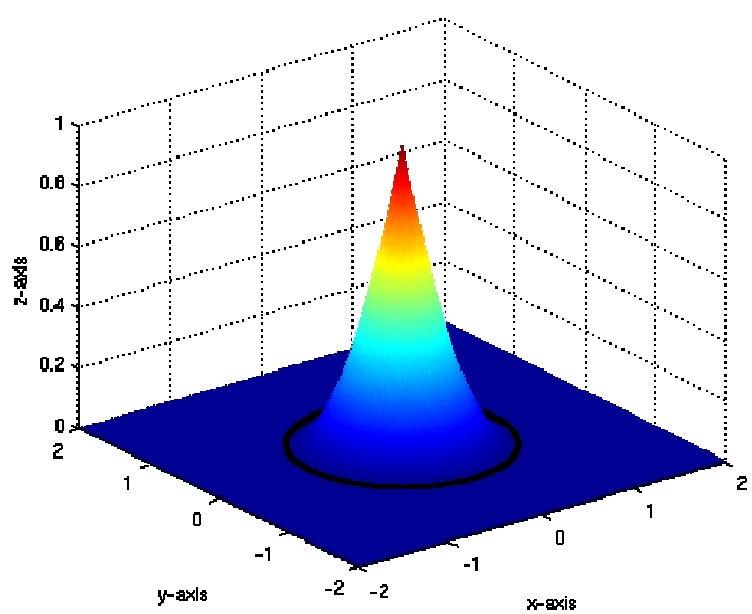


Figure 2: C^0 Wendland function.

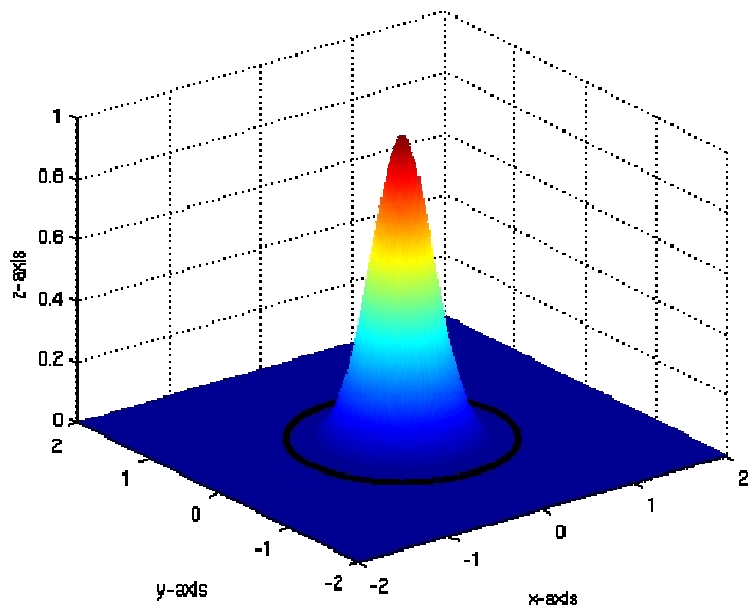


Figure 3: C^2 Wendland function.

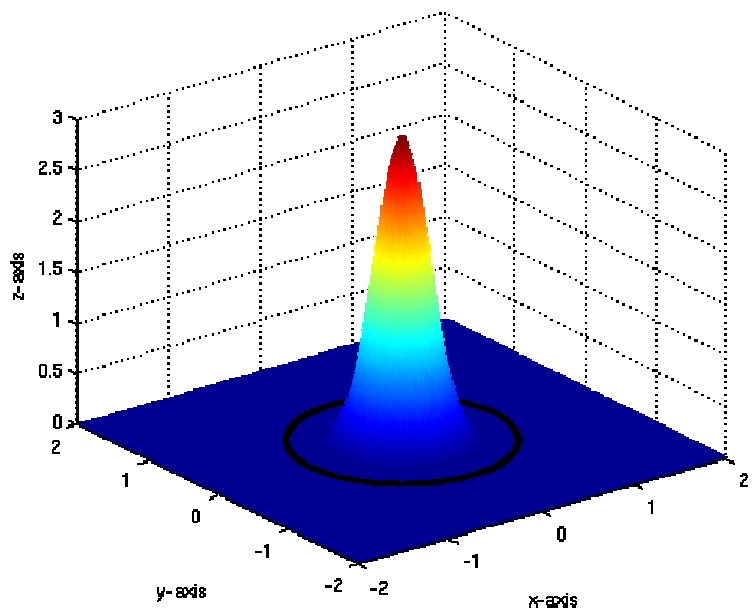


Figure 4: C^4 Wendland function.

The dark ring in each figure shows the radius of support. The functions vanish beyond this radius. W_2^2 was used throughout this project as this was the simplest function to evaluate that

still allows computation of first partial derivatives. Partial derivates of the interpolation can be computed in the following manner.

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{N} c_i \frac{\partial \phi(x - x_i, y - y_i)}{\partial x}, \qquad (6)$$
$$\frac{\partial f}{\partial y} = \sum_{i=1}^{N} c_i \frac{\partial \phi(x - x_i, y - y_i)}{\partial y}.$$

In the case of W_2^2 these partial derivates become:

$$\frac{\partial \phi(x,y)}{\partial x} = \frac{\partial \phi}{\partial r}\frac{\partial r}{\partial x} = \left(20r^4 - 60r^3 + 60r^2 - 20\right)_+ \left(\frac{x}{\alpha\sqrt{x^2 + y^2}}\right) \qquad (7)$$
$$= \left(20r^4 - 60r^3 + 60r^2 - 20\right)_+ \left(\frac{x}{\alpha^2 r}\right) = 20(1-r)_+^3 \left(\frac{x}{\alpha^2}\right).$$

The calculation proceeds similarly for the y direction:

$$\frac{\partial \phi}{\partial y} = 20(1-r)_+^3 \left(\frac{y}{\alpha^2}\right). \qquad (8)$$

For this project α was chosen equal to 5000m. This value gave the best balance between accuracy and computational time. It is worth noting that doubling the radius of support results in approximately 16 times more computational time. This is because doubling the radius effectively increases the area (and number of data points) of concern around each data point by a factor of 4. Additionally, **G** is constructed as a matrix with *N*x*N* elements and because of this there are 16 times the previous number of matrix entries, hence the increase in computation time.

An additional challenge that arose in the interpolation of the model data was the use of different coordinate systems. The coordinates of each node were given in latitude and longitude, while the current velocity was given in meters per second. To resolve this, the node coordinates were projected onto the plane using a Universal Transverse Mercator projection. UTM has the

advantage that its coordinates, called Easting and Northing respectively, are given in meters. With the flow velocity specified in m/s, this eliminates the need for any further unit conversions. The figures in this paper all use the UTM coordinate system.

The coordinate transform and interpolation coefficients c_i were computed beforehand by a pre-processing MATLAB algorithm. Once the pre-processing algorithm is complete, 90-day trajectories could be computed in less than two minutes on the Linux workstations that were used. This trajectory computation method served as the basis for invariant manifold computation as well as the computation of synoptic Lagrangian maps.

PART II: Results

Trajectories were computed as a first step towards visualizing and understanding flow patterns. The plot below represents trajectory paths tracked for 12 hrs; the black dots are the particle initial positions. The gray mesh in the background represents the finite elements used in the QUODDY model; coordinates are given in Easting (x-axis) and Northing (y-axis).

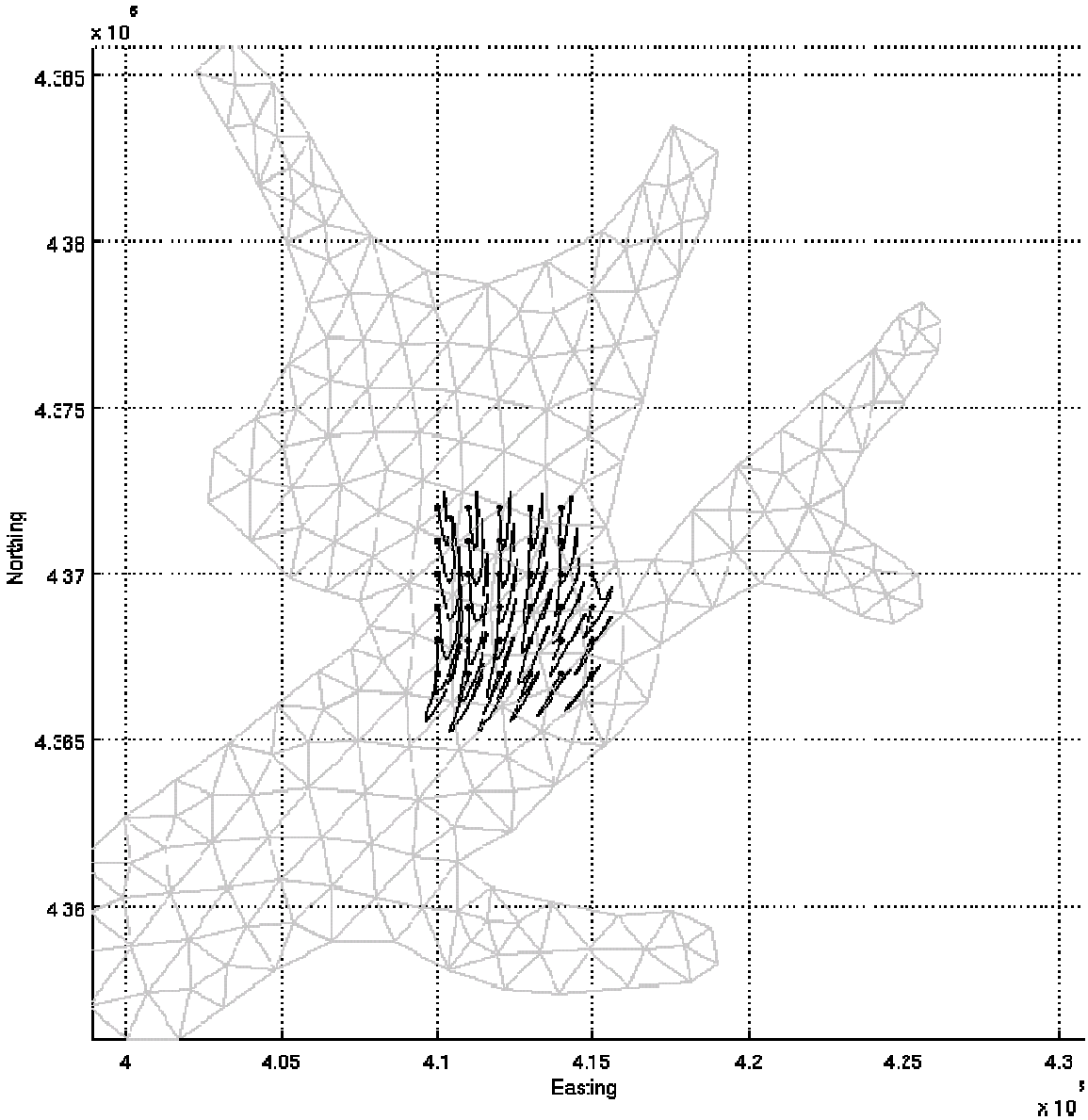


Figure 5: 12-hr particle trajectories at the Chesapeake headwaters.

The graph lines are 5 km apart in all cases. It was immediately evident that the 12-hr cycle of the tides was the predominant factor in material transport. It was necessary, therefore, to ensure that the time step in the Runge-Kutta scheme was small enough to resolve the tidal period. Therefore, the trajectory computation algorithm uses a 1 hour time step. For the trajectories in Figure 5 at the mouth of the Susquehanna, the tidal forcing is quite evident in the sinusoidal nature of the trajectories. The tidal influence this far up the Bay is, however, relatively weak compared to locations further south. Additionally, it is important to note that the trajectories generally move south of their initial positions. Over time, surface material in all parts of the Chesapeake follow a general motion from the northern end, to the Atlantic interface near the Norfolk / Newport News area. This is due in part to the large inflow from rivers like the Susquehanna in the northern end. The timescale of this north to south migration is on the order of several months. However, as is shown in the following section on synoptic Lagrangian maps, it is extremely rare for a particle to migrate from the extreme northern end to the Atlantic Ocean. Typically it will encounter the coast before it completes the journey.

The area depicted in Figure 6 is the narrowest stretch of the Chesapeake. Here the Bay is only 7 km across. This is also the stretch near Annapolis traversed by the Chesapeake Bay Bridge. This region proved to be of great importance in later studies on the computation of DHTs and their invariant manifolds.

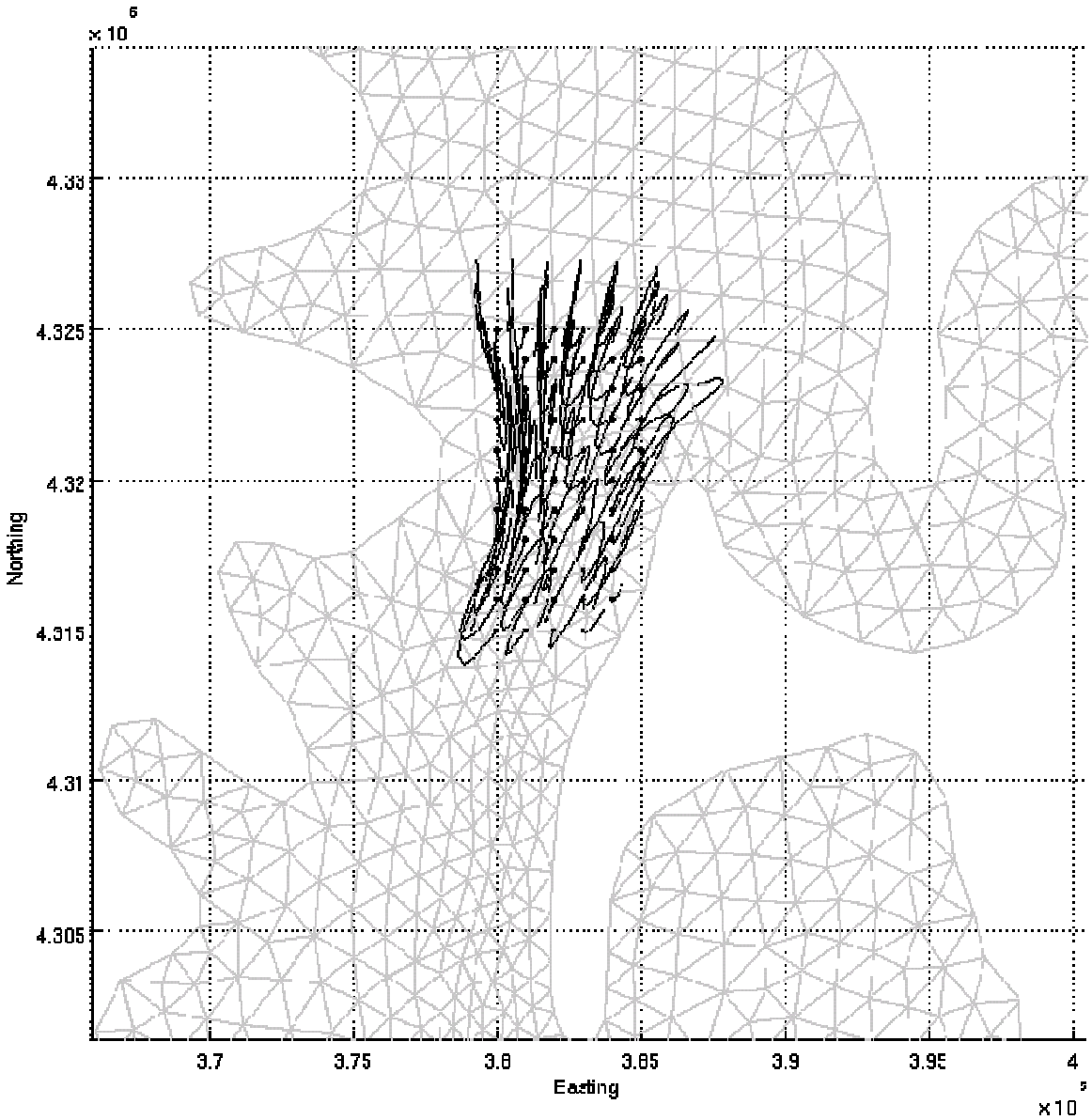


Figure 6: 12-hr particle trajectories near Kent Island.

Again in Figure 6 as in Figure 5 the tidal forcing is evident. Current velocity in this region is unusually high. This is due probably to a Bernoulli Effect caused by the narrowing of the channel as well as inflow from the Chester River to the east.

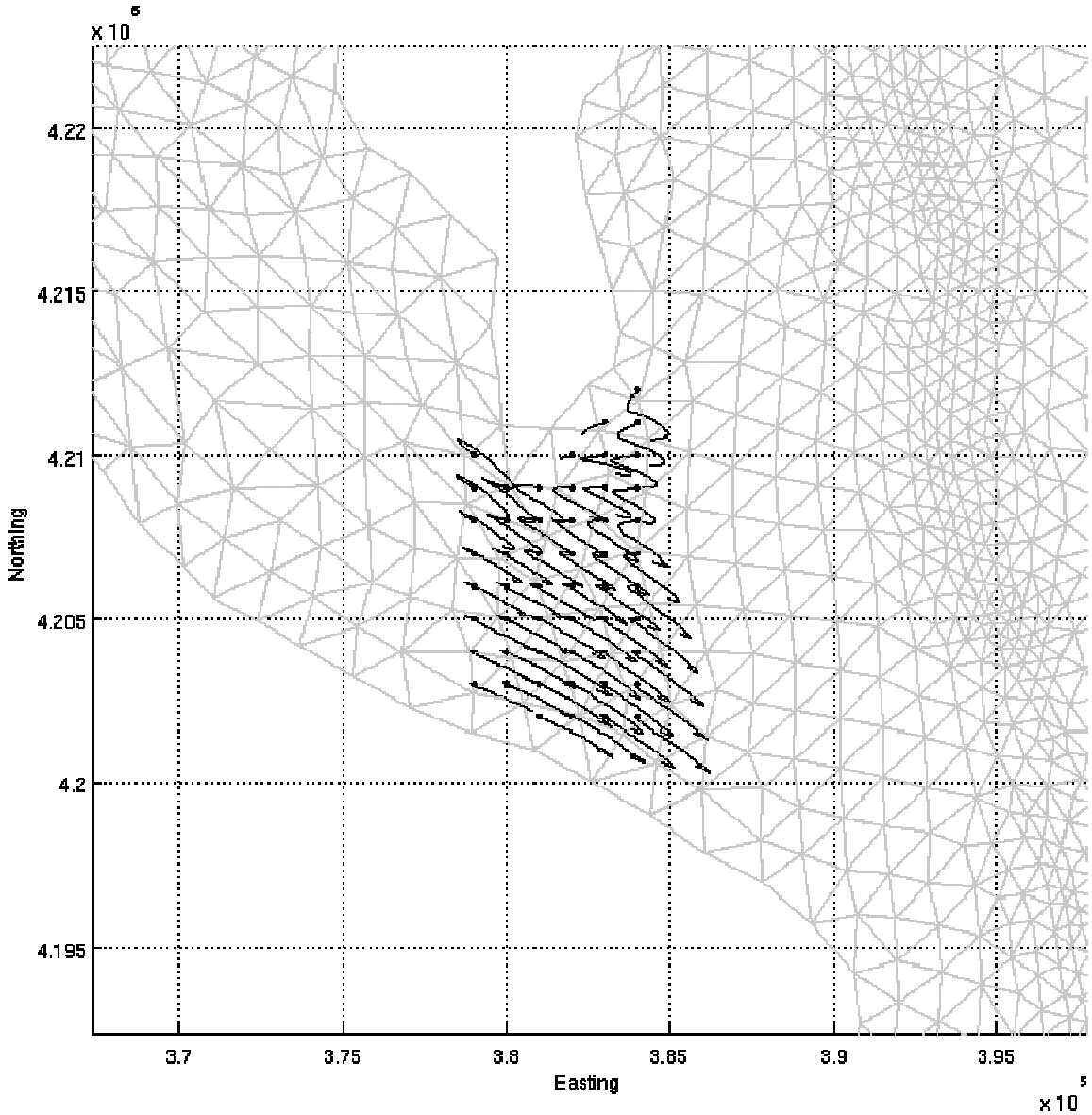


Figure 7: 12-hr particle trajectories at the mouth of the Potomac.

Figure 7 shows the behavior of material trajectories at the mouth of the Potomac River. As one would expect, the volume of flow from the Potomac tends to push the trajectories to the east. There are a few trajectories near the northern border of the Potomac mouth, near the 4210000m Northing line, that flow westward. This is evidence of a small scale vortex. This type of eddy is precisely what would be expected from the inflow of a large volume of water.

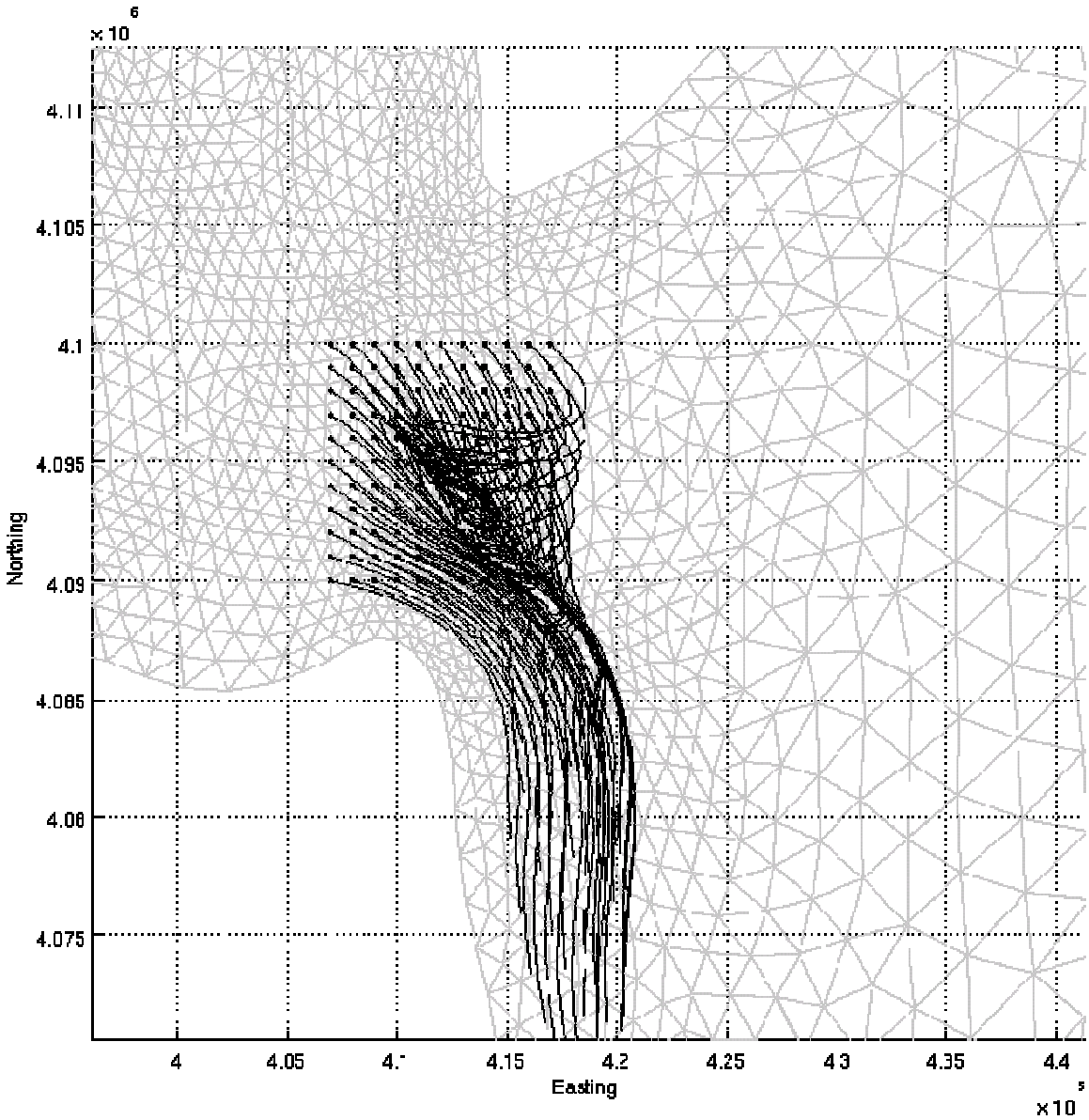


Figure 8: 12-hr particle trajectories at the mouth of the Chesapeake.

At the mouth of the Chesapeake Bay, tidal forcing and ocean currents dominate the flow. The huge tidal factor is evident in the westward motion of the trajectories along the eastern side of the grid. Note also that all the trajectories are swept to the south along the coast. This is due to the Gulf Stream. The Gulf Stream is a powerful current that carries warm water from the Gulf of Mexico along the east coast of the United States and across the Atlantic to Europe. This current

detached from the east coast in the vicinity of Cape Hatteras, NC, just south of the mouth of the Chesapeake. A large scale eddy forms north of this branch point causing the strong southern current visible in Figure 8. For a more in depth study of the dynamics of the Gulf Stream refer to [16] and [17]. From these preliminary trajectory investigations it is apparent that the QUODDY model is, at least qualitatively, accurate. Most of the effects that one would expect to see, forcing of the tides, river inflow etc. are present. Assessing quantitatively how accurate the model is, and/or assimilating real data to improve its accuracy is a possible future direction for research.

This is just a survey of the interesting regions of the Chesapeake Bay. It is not practical to plot and discuss the nuances of flow in every region of the Chesapeake in this manner. Better methods of flow visualization and summarization of flow behavior are needed. To this end, the methods of Wiggins, Lipphardt, Kirwan, et al. are applied in the following sections in order to better understand and identify important features in the flow.

SYNOPTIC LAGRANGIAN MAPS

PART I: Methodology

Synoptic Lagrangian maps provide a summary of particle behavior for a given flow. The idea was pioneered by Lipphardt et al. [10] in their work on analysis of flow in the Monterey Bay. For this work on the Chesapeake type I SLMs were computed, that is, plots of the forward residence time. Forward residence time is defined as the time it takes a particle, starting at a given location, to either escape to the open ocean or to come within a certain distance of the coast. In their study of the Monterrey Bay, Lipphardt et al. defined this distance to be 500m. The zone 500m from the coast was labeled the "coastal buffer zone". In this region waves and other shallow water effects make transport dynamics somewhat different than they are further from the coast. Hence we cut off trajectories when they come into this "buffer zone" and consider them to have encountered the coast. Because of the large size of the Chesapeake Bay compared to the Monterrey Bay, and because of its relatively shallow coastal regions, this buffer zone was defined to be within 1 km of the coastline.

The basic idea to computing a type I SLM is to first define a grid covering the entire Bay, and to integrate the trajectory of a particle starting at that grid point. The particle's total time is tracked along its trajectory until it encounters the coastal zone or escapes to the ocean. Once the time has been computed for each grid point, the resulting plot gives information about particle transport and mixing. Regions with high turnover show as patches of low residence time, while regions which are relatively stagnant show higher residence time. By summarizing massive amounts of trajectory data in this manner it is possible to better understand dynamics of the flow

than it is with either velocity field or particle trajectory plots. Figure 9 shows the grid used for SLM computation. Grid points are spaced at 1km intervals.

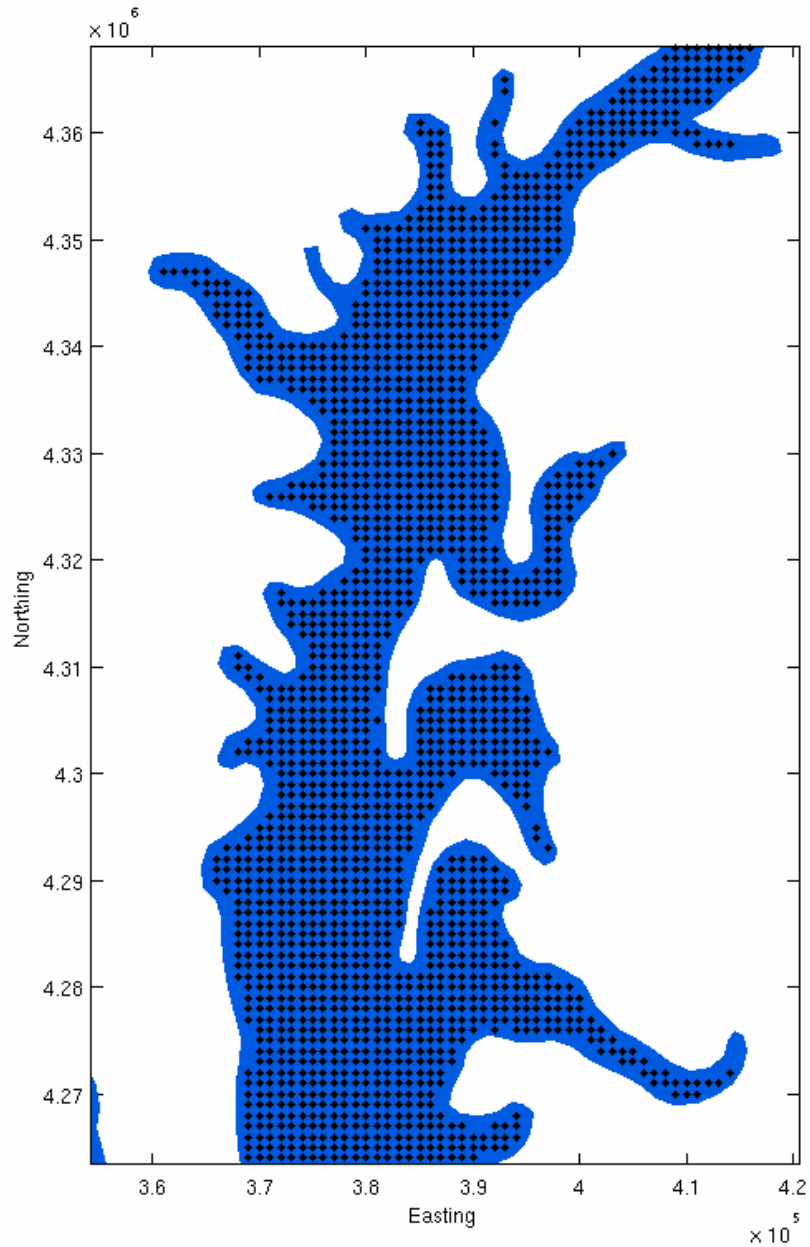


Figure 9: Grid covering the Chesapeake Bay, the coastal buffer zone is not covered.

Also valuable are animations of the synoptic Lagrangian maps at successive time steps. Observing the movement of patches of certain residence time can be even more enlightening that the static pictures. Unfortunately with the grid method described in the previous paragraph, the

computation of several successive SLMs requires tremendous computing power. Even on a several-hundred processor supercomputing cluster, the computation of a week's worth of SLM data computed every hour is a task that requires several days. This computation is not a task possible for single-processor machines. I have developed a new, efficient algorithm for computing several successive SLMs which makes use of previously computed data to speed up the computation.

The critical idea behind my algorithm is that once the path and total residence time for a particle have been computed, the residence time at each position on the particles path for the entire time that it remains in the Bay is also known. For example, if the residence time of a particle is computed to be 20 days, and that particle moves 1km in the next hour, the residence time at its new position 1km away is 20 days minus one hour. The particle will still encounter the coast at the same time and follow the same path.

Instead of re-computing trajectories for each grid point at each time step, compute trajectories for the entire grid only once and store the residence time *as well as* the entire path for each particle starting at each grid point. This "particle-tracking" method is more efficient because it makes use of previously computed information. The initial grid that was chosen to initiate the particle-tracking method was not laid out in a square grid fashion as in Figure 9, but rather on a hexagonal grid with 1km spacing as in Figure 10. The hex layout results in more initial grid points, 12000 as opposed to approximately 10000, but because the points have more neighbors to begin with, fewer and smaller gaps open as the mesh is advanced in time. This results in fewer new points inserted at each time step and lower total computational time.

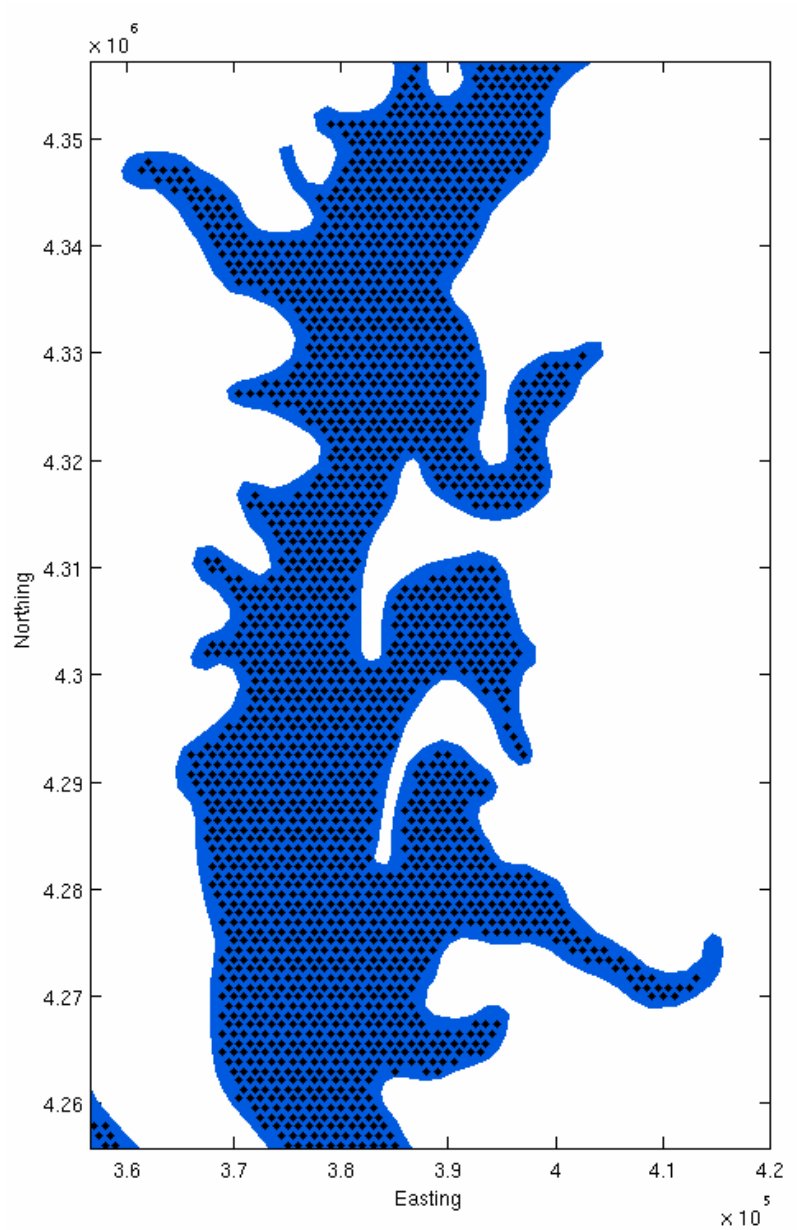


Figure 10: Hexagonal grid used to initiate the particle-tracking method.

To begin the particle-tracking method, compute a trajectory for each point on the hex grid as with the original algorithm and store the data. The first SLM is the only time-intensive computation in that it is the only time step for which trajectories must be computed for each data point.

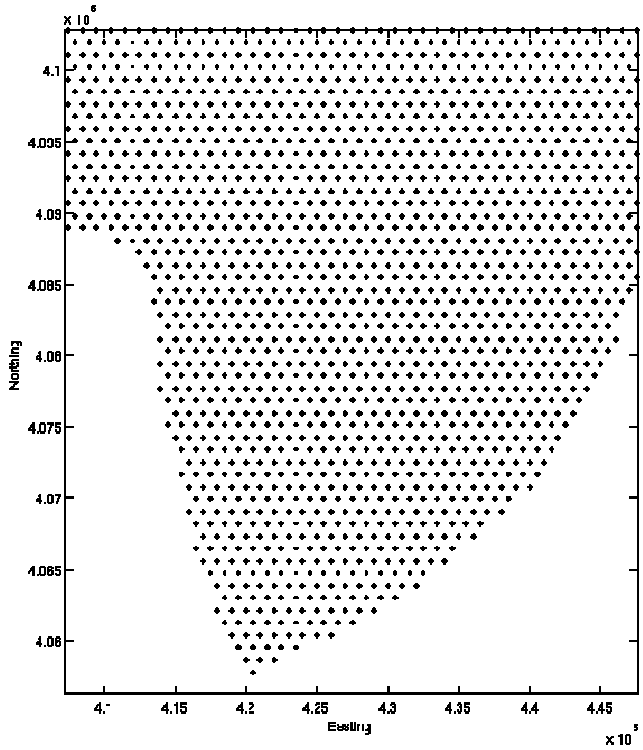


Figure 11: A section of the initial grid.

For the next time step, the grid becomes deformed as each point is advanced forward in time.

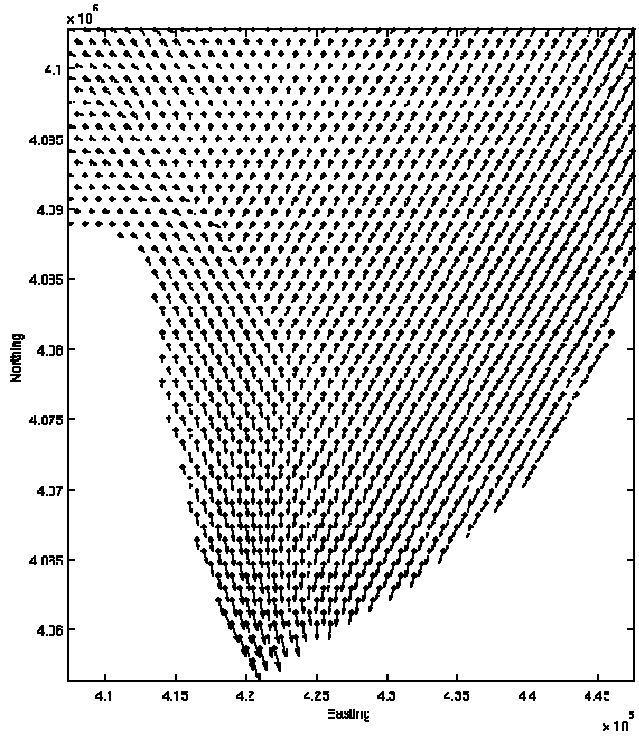


Figure 12: Velocity vectors showing advection of initial grid.

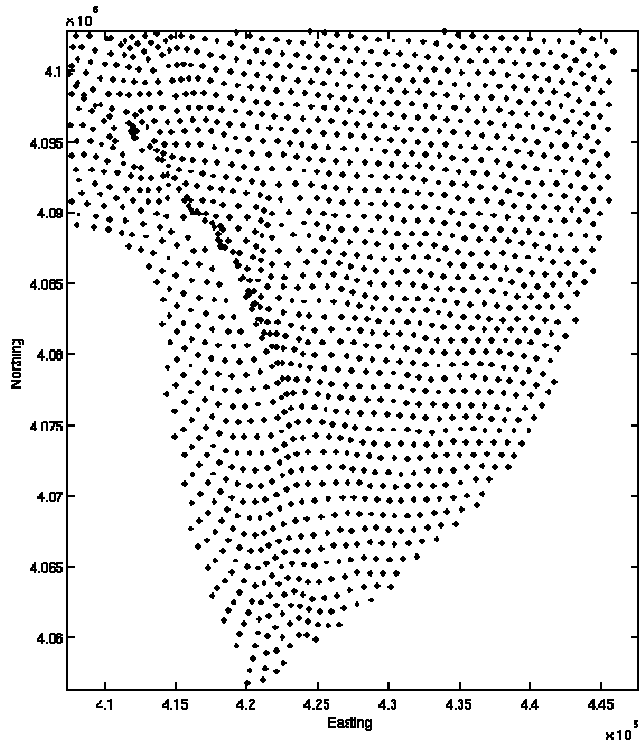


Figure 13: Grid after advection, note the gaps that occur.

In the gaps that emerge, insert new grid points such that the spacing is kept roughly constant.

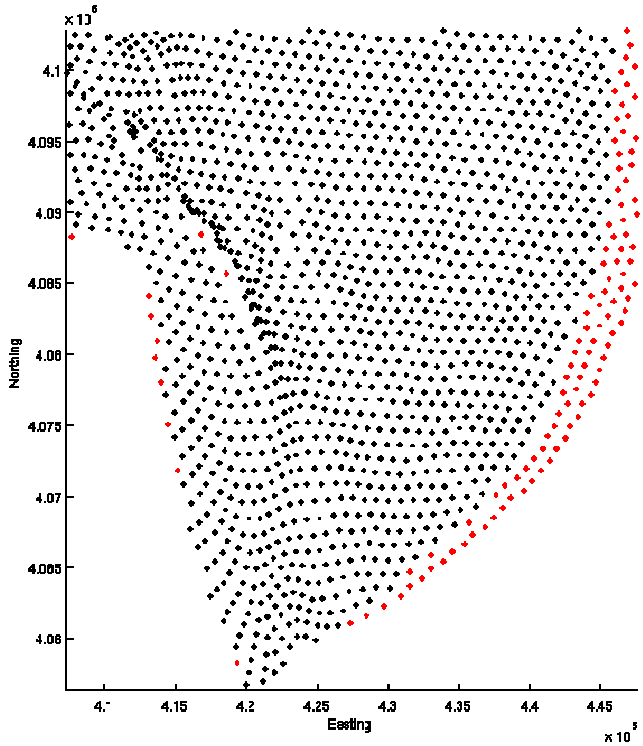


Figure 14: An illustration of point insertion, newly inserted points in red, original points in black.

Compute the particle paths and residence time for the new grid points *only*. The residence time for the grid points advanced from a previous time are already known.

Once it was discovered that data from previous calculations could be effectively used to compute SLMs at later times, development of the algorithm was relatively simple. The only difficult part came in developing an effective point insertion technique. A technique was needed that would (a) fill in the necessary gaps in coverage that arise and (b) be simple enough that the time taken for point insertion would not be significantly greater than the time taken to compute the resultant trajectories. If (a) is not satisfied the SLMs would have large gaps in residence time data, resulting in inaccurate plots. If (b) is not satisfied it would compromise the speed-up and advantage over the original method.

My point insertion technique works as follows.

1. Define a test grid on a finer scale than the hex grid used to initiate the algorithm.
2. Assign a potential function, see equation (9), to each point in the grid that has been deformed, as in Figure 13.
3. Search the test grid for a location with a low value of the potential function, indicating that it is not covered well by the deformed grid.
4. If the function value is below a certain threshold, insert a new point at this location. Continue searching the test grid until every point is above the threshold indicating that the entire domain is well covered.

The test grid was used for the point insertion was of the same hex layout as the original grid, except with 200m spacing. This results in a grid of approximately 300,000 points that need to be searched each time during the point insertion phase. The choice of 200m spacing was

driven more by the interests of time than any other factor. Increasing the test grid resolution beyond this does not appear to contribute to the accuracy of the point insertion algorithm.

The potential function was a simple hat-shaped function set to 0 beyond the distance of the initial spacing. It is similar in concept to the radial basis functions of the interpolation section.

$$P(x, y) = (1 - r)_+ , \quad (9)$$
$$r = \frac{\sqrt{x^2 + y^2}}{1km}.$$

The total potential function that is tested at each test grid point is the sum of a potential function *P* centered at each point on the deformed mesh. If a test grid point had a potential value of 0, it would indicate that it is at least 1km from the nearest point of the deformed grid.

The potential function approach was chosen to solve the "gap" problem inherent to a finite grid. When testing on a discrete grid, it is possible to have gaps in the deformed mesh that are missed when the test grid does not have a test point in the missed region.

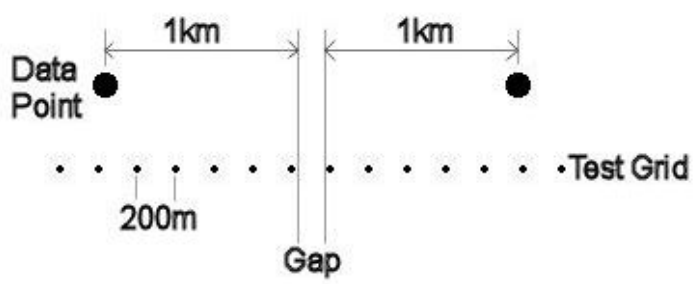


Figure 15: Illustration of the "gap" problem.

By using the potential function method and testing for a threshold we select a point near the gap even though the test grid skips over.

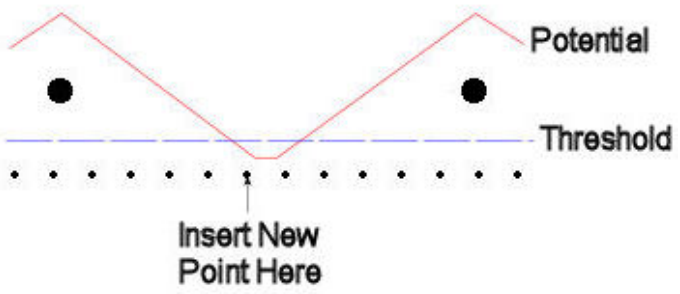


Figure 16: Illustration of the potential function method.

After insertion the total potential function is above the threshold along the entire test grid indicating good coverage.

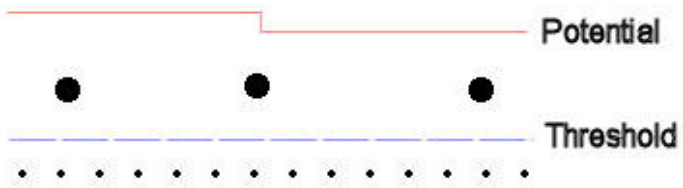


Figure 17: Coverage after point insertion.

Using this algorithm it becomes necessary to only compute trajectory and residence time for roughly 300 points at each time step. This is as compared to the approximately 12,000 points on the initial grid. The time savings is even more than the apparent 40x speed up produced by computing such a small fraction of the total number of points. Many of the inserted points are replacing those that have just disappeared from entering the coastal buffer zone. These points also have short residence time and are quickly computed. Thus, the final speed-up is 50-60 times.

PART II: Results

Tracking particles in the Lagrangian manner of the particle-tracking method creates a large efficiency increase. Computing SLMs every hour for a week's worth of data using the original method would take three and a half months of continuous computation on a single machine (2535 hours). Using the particle-tracking method the same is accomplished in approximately 50 hours of continuous computation. Additionally, a "crowding" effect was noted. In some regions of the Bay, trajectories crowded together opening up gaps in the SLM coverage. New points were inserted in the gaps, increasing the total number of data points. Figure 18 shows the total number of points in the computed SLM at each time increment in red. The total number included the number of points carried over from the previous increment plus the newly inserted points for which residence time had to be calculated. Due to this crowding effect, the total number of points gradually increased from its initial value of 12,414 until it reached a roughly steady state at around 18,000 trajectories. It would appear that 18,000 is a critical data point density. At this point, roughly the same number of trajectories are encountering the coast or escaping to the ocean as are inserted in the resultant gaps.

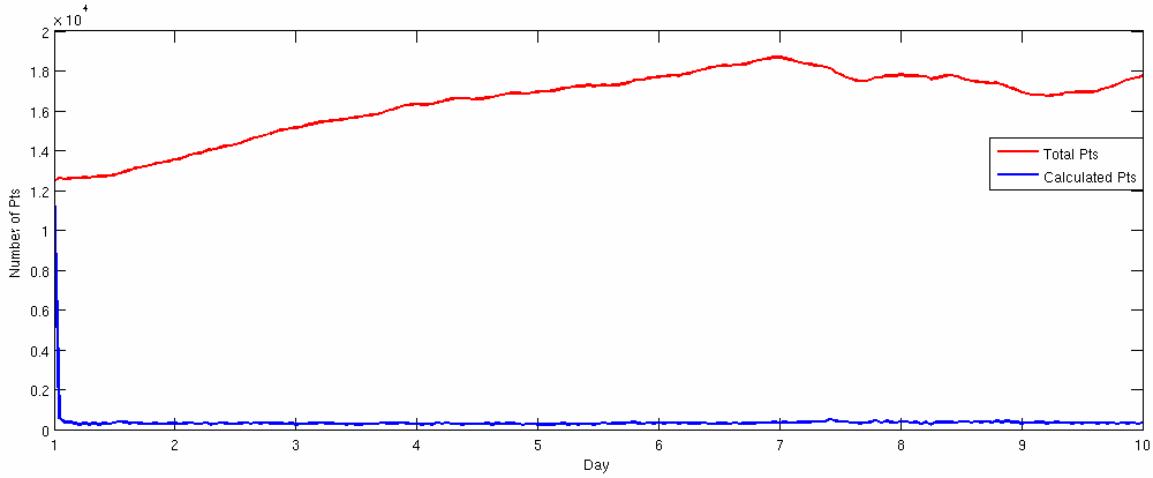


Figure 18: SLM total points (red) and calculated points (blue).

Also evident in Figure 18 is just how small a fraction the number of points that need to be calculated (blue line) is of the overall total. Figure 19 shows the number of points requiring calculation in greater detail.

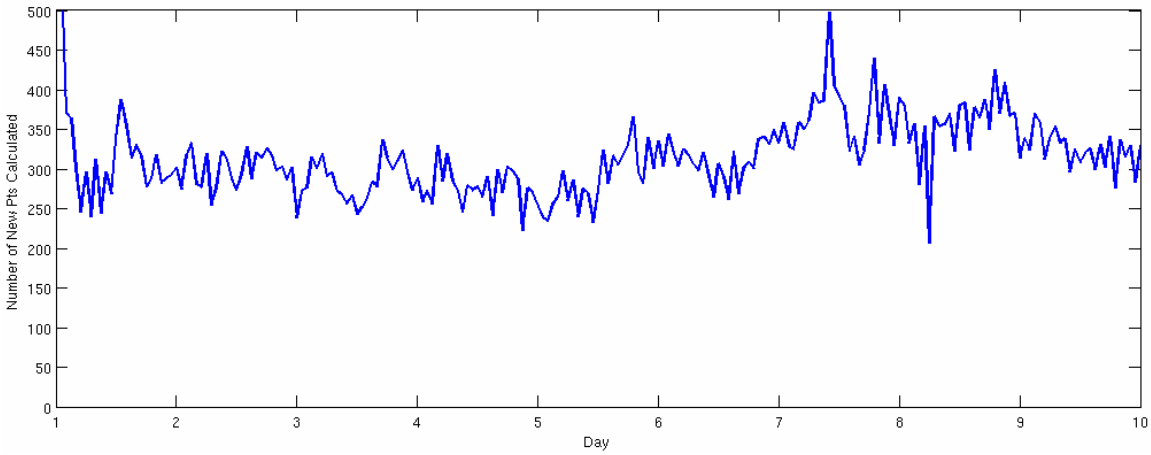
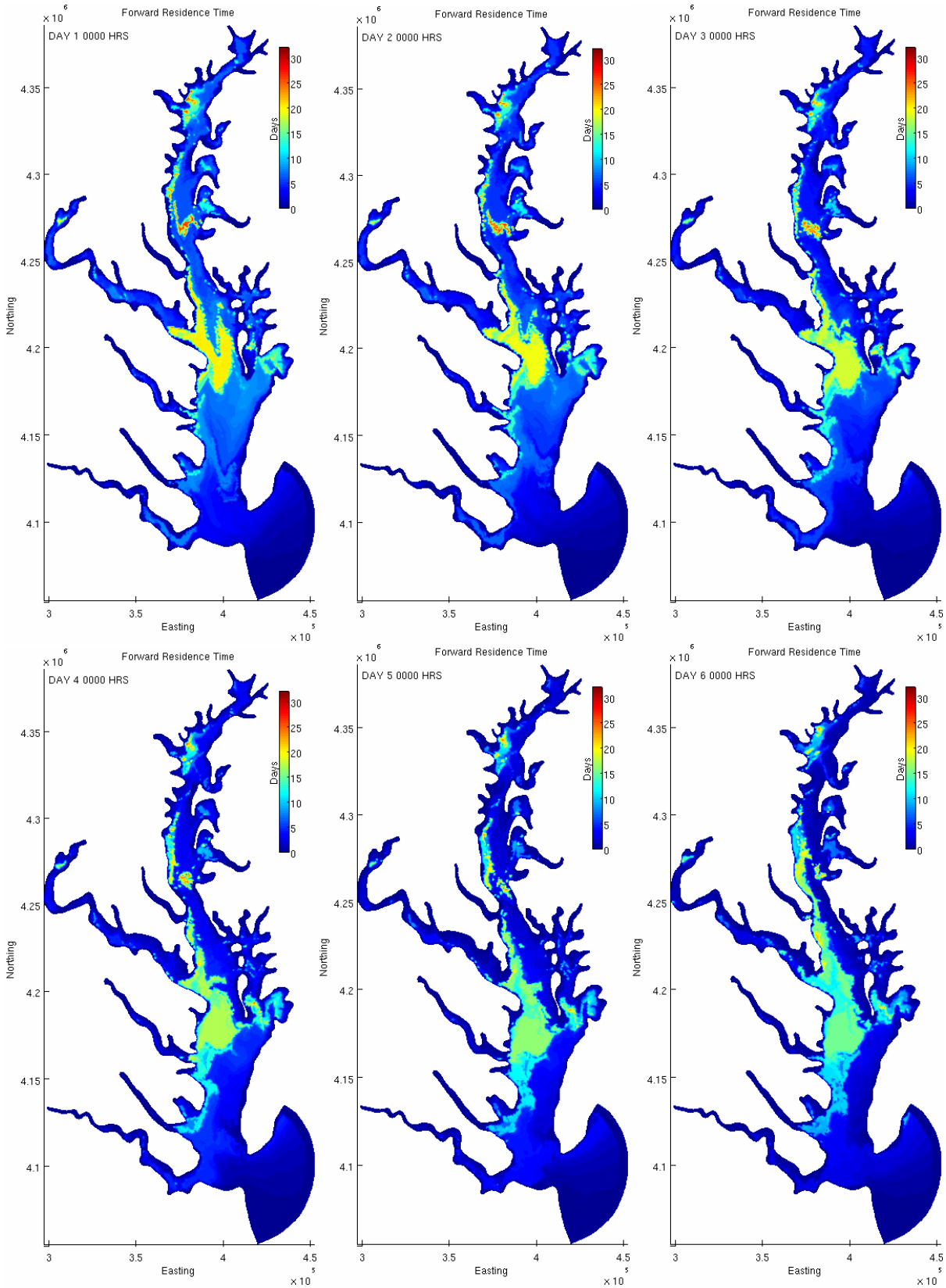


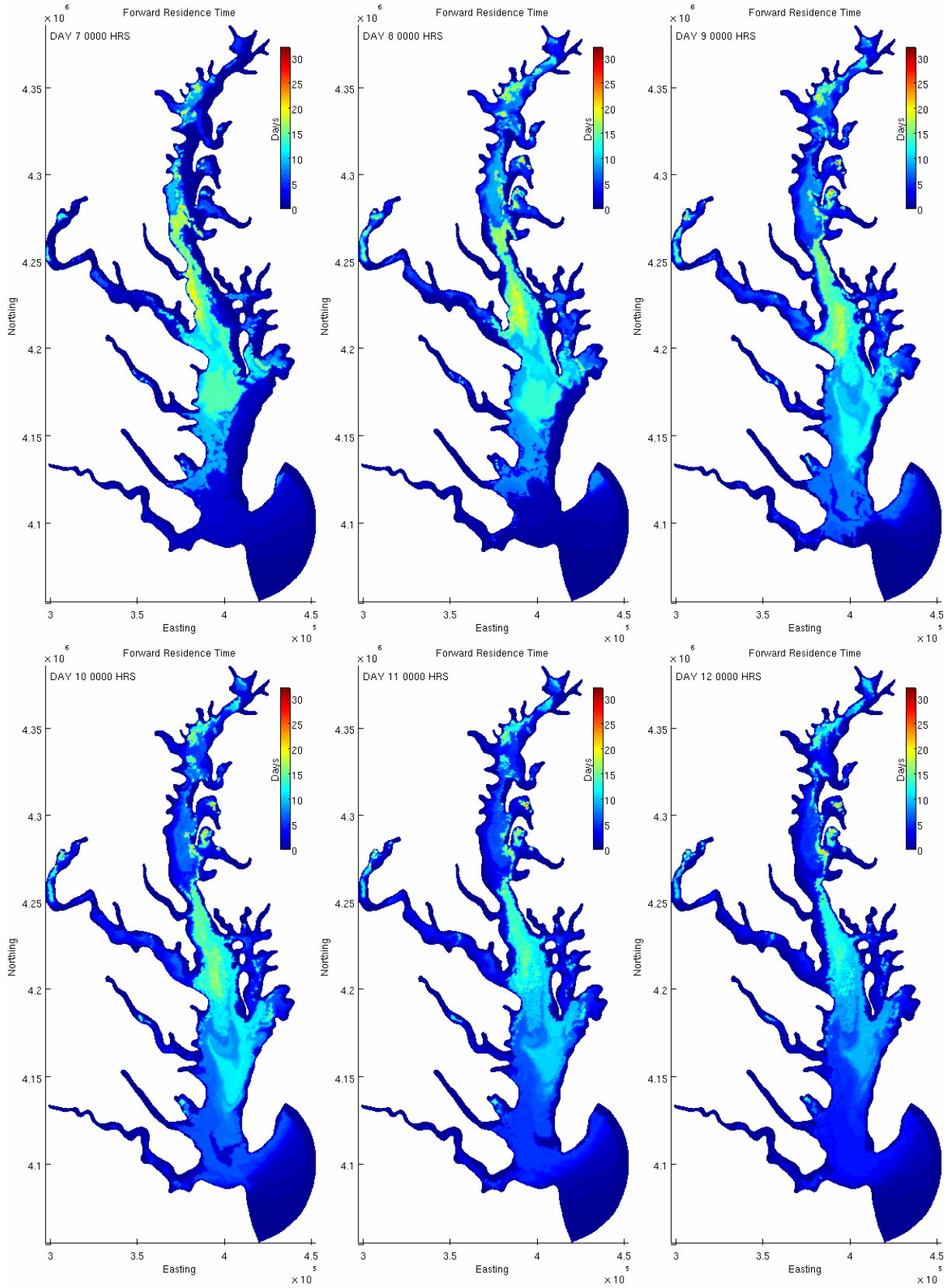
Figure 19: Number of calculated points.

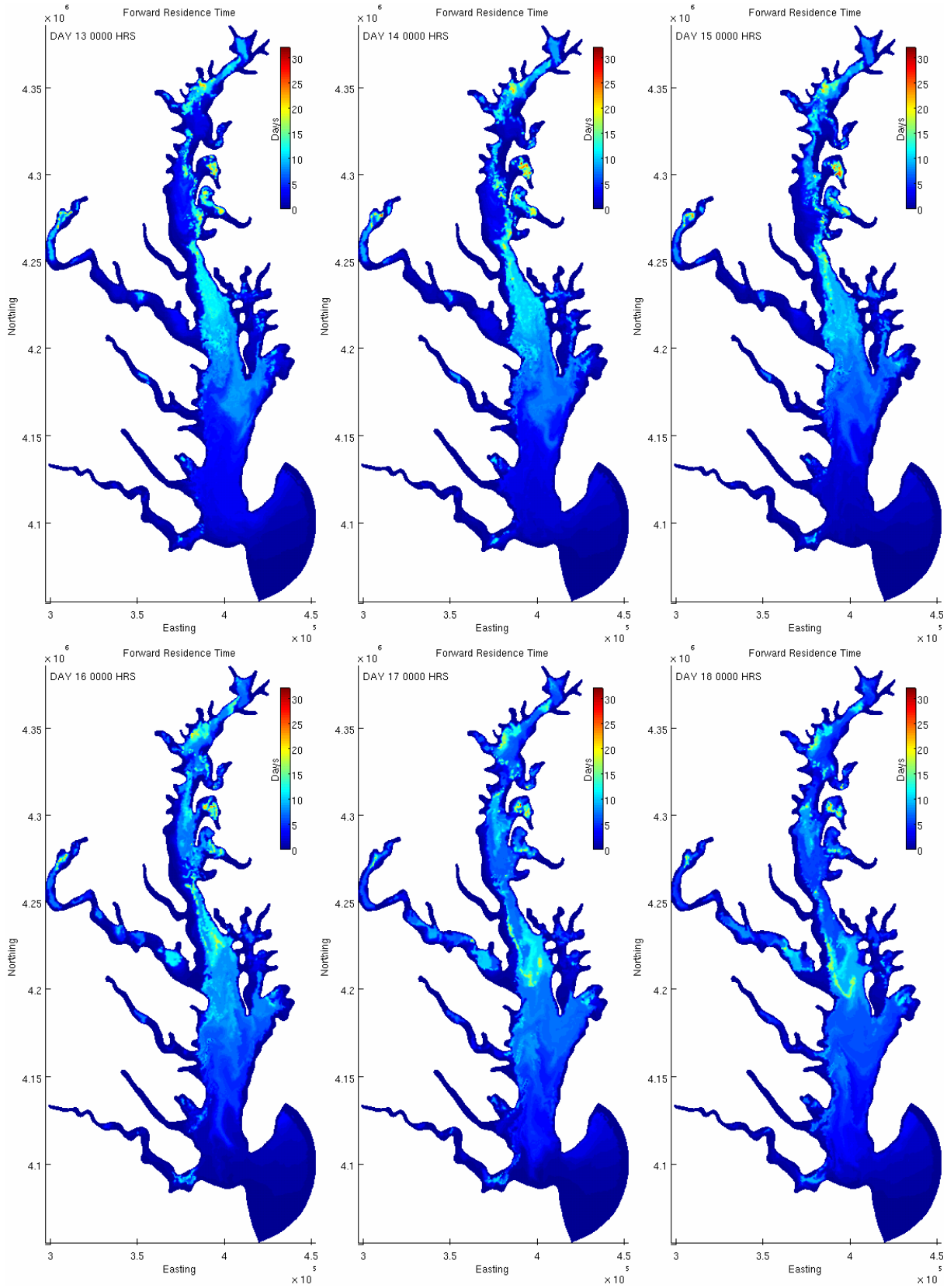
The number of new points at any given time varied somewhat, but stayed relatively close to its mean value of 293 (σ=54.6). These values are calculated excluding the computation requirement of 12,414 for the initial time step.

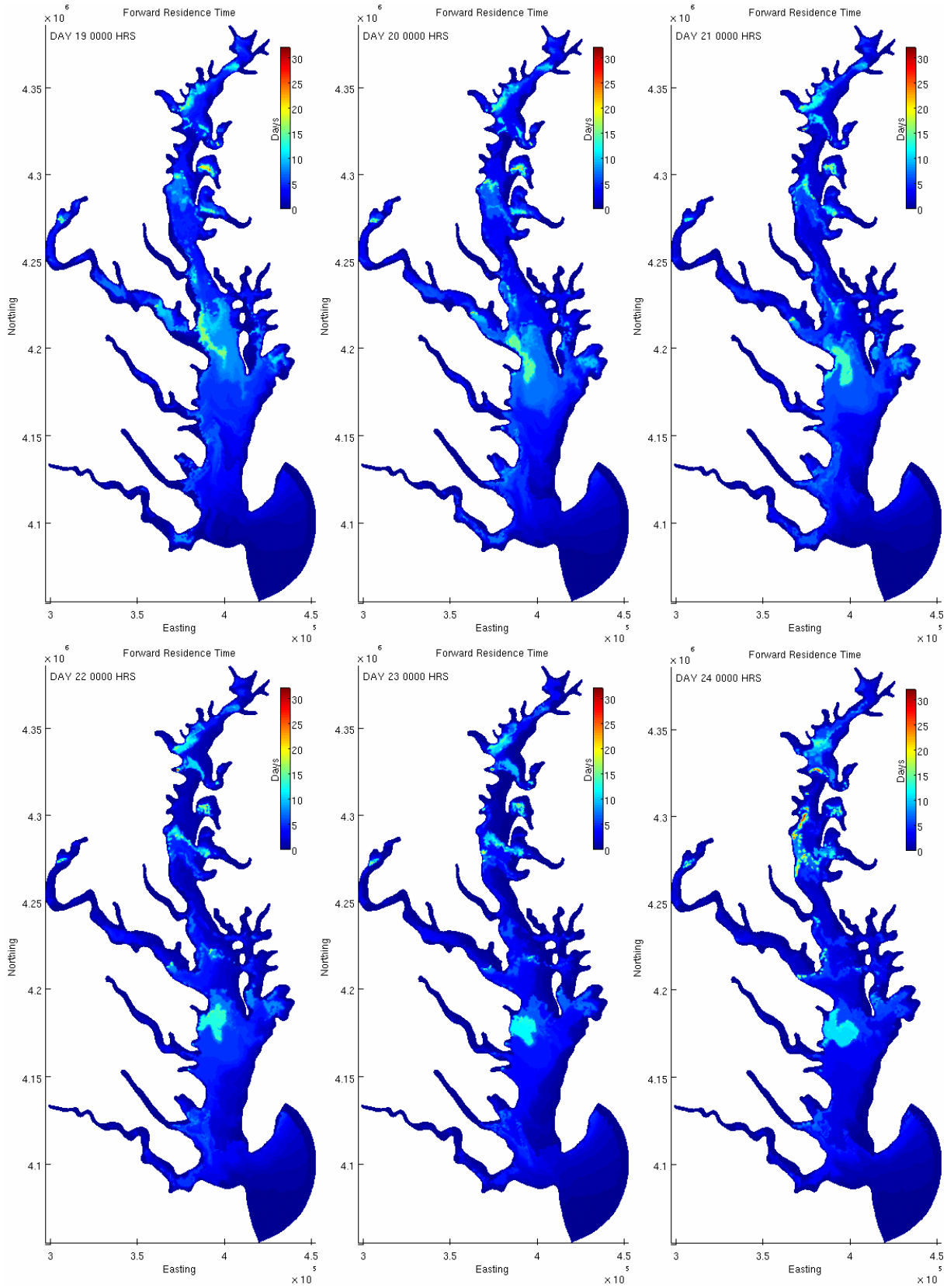
It should be noted, however, that the efficiency of this algorithm relies of the computation of many SLMs at small time intervals. In this project, SLMs were computed for each 1 hr time increment resulting in the low number of inserted points seen above. If this interval is lengthened, the resultant gaps, and number of inserted points grow larger, decreasing efficiency. Indeed, because of the initial requirement to calculate every point, the particle-tracking is no faster than the original method for isolated time synoptic Lagrangian maps.

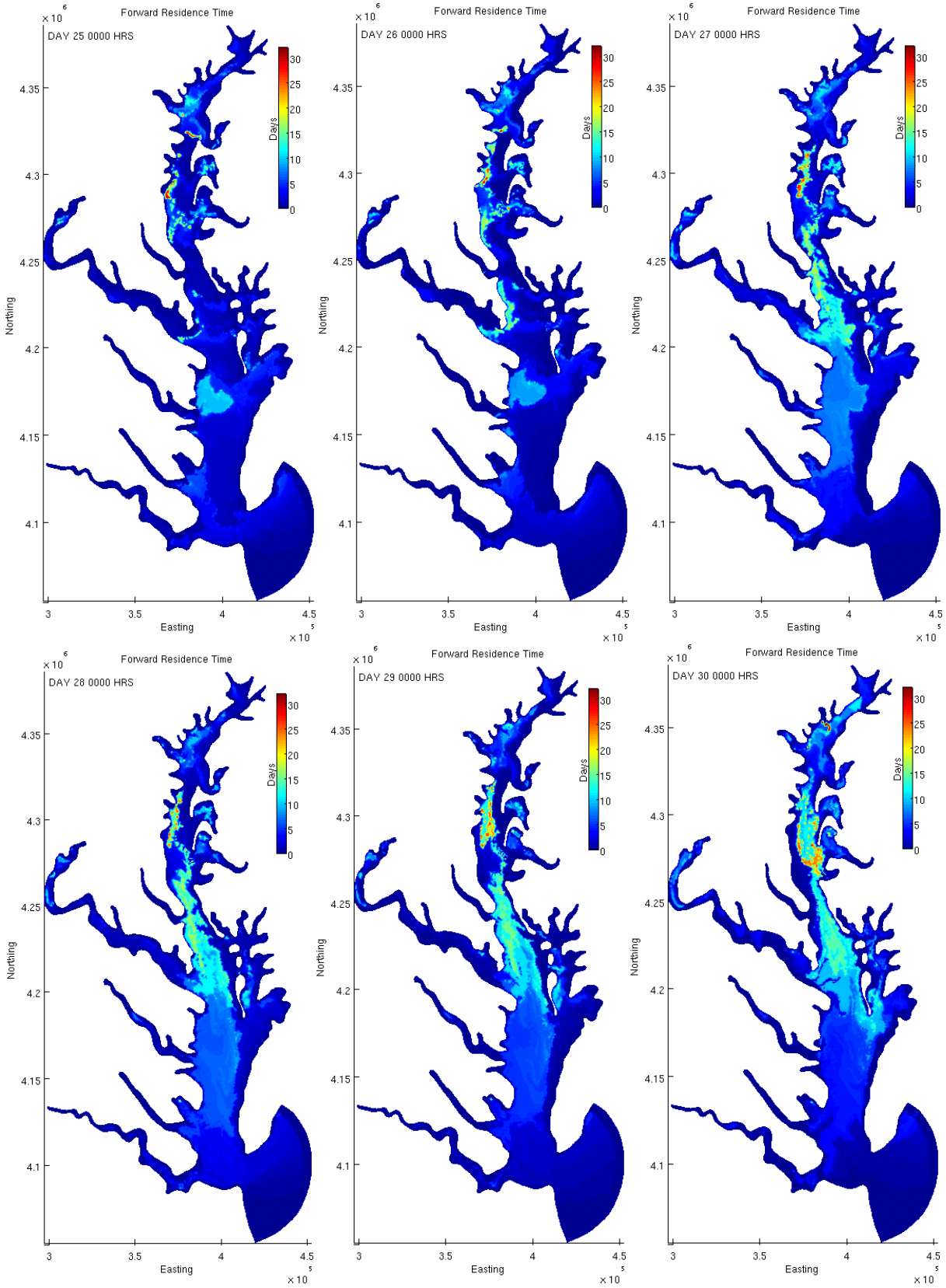
The ultimate test of the particle-tracking method was to compute SLMs for one month at one-hour intervals. The plots below are the results at midnight of each successive day.


Forward Residence Time
DAY 1 0000 HRS
DAY 2 0000 HRS
DAY 3 0000 HRS
DAY 4 0000 HRS
DAY 5 0000 HRS
DAY 6 0000 HRS
Days
Northing
Easting
× 10^6
× 10^5


Forward Residence Time
DAY 7 0000 HRS
DAY 8 0000 HRS
DAY 9 0000 HRS
DAY 10 0000 HRS
DAY 11 0000 HRS
DAY 12 0000 HRS
Days
Northing
Easting
× 10^6
× 10^5


Forward Residence Time
DAY 13 0000 HRS
DAY 14 0000 HRS
DAY 15 0000 HRS
DAY 16 0000 HRS
DAY 17 0000 HRS
DAY 18 0000 HRS
Days
Northing
Easting
×10^6
×10^5


Forward Residence Time
DAY 19 0000 HRS
DAY 20 0000 HRS
DAY 21 0000 HRS
DAY 22 0000 HRS
DAY 23 0000 HRS
DAY 24 0000 HRS
Days
Northing
Easting
×10^6
×10^5


Forward Residence Time
DAY 25 0000 HRS
DAY 26 0000 HRS
DAY 27 0000 HRS
DAY 28 0000 HRS
DAY 29 0000 HRS
DAY 30 0000 HRS
Days
Northing
Easting
x 10^6
x 10^5

These plots show a wealth of interesting information. In contrast to the trajectory plots of Figure 5, 6, 7 and 8, trajectory behavior and time scale is immediately visible for the entire domain. The ultimate fate of the particle trajectories is also calculated by the algorithm. The large yellow patch visible on Days 1 and 2 eventually meanders down the Bay and out to the ocean while surrounding water encounters the coast. This explains the sharp transition in residence time from 20 days for the yellow patch and only a week for the surrounding material. Also of interest is the period from Day 12 through Day 24 when the entire Bay shows relatively low residence time. Upon inspection of the data set, it was observed that the average and maximum current magnitude was elevated during this period to an average of approximately 1 m/s and a max of 2.4 m/s. This is as compared to roughly a .7 m/s average current and 1.5 m/s maximum for the other periods. It is not clear what cause these elevated current values, possibly intensified winds or storm activity, but it is clear that these currents forced material close to the shore faster than would have otherwise happened.

The data set that was used to compute these SLMs was a 91-day output generated by an in-house QUODDY run. The necessary winds, inflow and tidal data were supplied by NOAA from a data base starting on February 2, 1999 and ending on May 8, 1999. The selection of the year 1999 was more or less inconsequential. Data from more recent years could have just as easily been used; 1999 was what was immediately available at the time. The choice of the early spring, however, was important. It is well known that during the spring stratification in the Chesapeake is the strongest. Later on in the fall a higher degree of vertical mixing is observed. Since this project was restricted to two dimensions, data was chosen from the time of year when flow in the third, vertical, dimension was the weakest. Thus, these plots can be viewed as

summaries of buoyant particle motion as well as neutrally-buoyant particles near the surface. Because of the stratification of the flow, particles near the surface currents should remain near the surface and follow similar flow patterns.

DHT COMPUTATION

PART I: Methodology

The main focus of this project was the computation of distinguished hyperbolic trajectories for the Chesapeake Bay. The following section will describe an implementation of the algorithm described by Wiggins. For a more thorough treatment, refer to [1].

We seek to solve for a trajectory of the general non-autonomous system $\dot{\mathbf{x}} = \mathbf{u}(\mathbf{x}, t)$, such that the solution is hyperbolic and stays within a small subdomain for the length of time associated with the given data set. The requirement that the trajectory be hyperbolic is satisfied if the velocity field local to the trajectory exhibits exponential dichotomy. Consider the linearized system $\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x}$, where $\mathbf{A}(t)$ is a linear first approximation if $\mathbf{u}$.

$$\mathbf{A}(t) = \begin{bmatrix} \frac{\partial u^1}{\partial x^1} & \frac{\partial u^1}{\partial x^2} \\ \frac{\partial u^2}{\partial x^1} & \frac{\partial u^2}{\partial x^2} \end{bmatrix}. \tag{10}$$

The fundamental solution matrix to the linearized system, denoted by $\mathbf{X}(t)$, is defined such that $\mathbf{X}(t)\mathbf{x}(t_0) = \mathbf{x}(t+t_0)$. The linearized system exhibits exponential dichotomy if there exists constants K_1, K_2, λ_1, λ_2 and a projection operator $\mathbf{P}$ such that:

$$\begin{aligned} \left\| \mathbf{X}(t)\mathbf{P}\mathbf{X}^{-1}(\tau) \right\| &\le K_1 e^{-\lambda_1 (t-\tau)}, \qquad && t \ge \tau, \\ \left\| \mathbf{X}(t)(\mathbf{I}-\mathbf{P})\mathbf{X}^{-1}(\tau) \right\| &\le K_2 e^{\lambda_2 (t-\tau)}, \qquad && t \le \tau. \end{aligned} \tag{11}$$

The requirement that the DHT is confined to relatively small subdomain is more general. This is of value when trying to compute the invariant manifolds of the DHT. Invariant manifolds require long data sets in order to generate on accurate computation, to analyze a certain region of

a dynamical system it is necessary that the DHT remains in that region for a time long enough (usually a few weeks) in order to get a well-developed picture of the manifold structure.

The algorithm developed by Wiggins works iteratively, improving the accuracy of the DHT calculation with each cycle. Denote the calculation at the end of the nth cycle as $\mathbf{x}^{(n)}$. If the initial equation that the DHT solves is $\dot{\mathbf{x}} = \mathbf{u}(\mathbf{x},t)$ define a new variable $\mathbf{y} = \mathbf{x} - \mathbf{x}^{(n)}$. Also, linearize the velocity field $\mathbf{u}$ about the guess $\mathbf{x}^{(n)}$ defining $\mathbf{A}(t)$ as in equation (10)

$$\dot{\mathbf{y}} = \dot{\mathbf{x}} - \dot{\mathbf{x}}^{(n)} = \mathbf{u}(\mathbf{y} + \mathbf{x}^{(n)}, t) - \dot{\mathbf{x}}^{(n)} = \mathbf{A}(t)\mathbf{y} + \mathbf{u}(\mathbf{y} + \mathbf{x}^{(n)}, t) - \mathbf{A}(t)\mathbf{y} - \dot{\mathbf{x}}^{(n)}. \quad (12)$$

$$\mathbf{f}(\mathbf{y},t) = \mathbf{u}(\mathbf{y} + \mathbf{x}^{(n)}, t) - \mathbf{A}(t)\mathbf{y} - \dot{\mathbf{x}}^{(n)}, \quad (13)$$

$$\dot{\mathbf{y}} = \mathbf{A}(t)\mathbf{y} + \mathbf{f}(\mathbf{y},t). \quad (14)$$

Now, as the vector fields have effectively changed, it may appear that we have to re-interpolate for the function $\mathbf{f}$. However, with a little clever pre-computation we can avoid this and do operations purely on the components. Define the following sets of components

$$\begin{aligned} 1 &= \sum_n \hat{k}_n \phi\left(x^1 - x_n^1, x^2 - x_n^2\right), \quad (15) \\ x^1 &= \sum_n \hat{x}_n \phi\left(x^1 - x_n^1, x^2 - x_n^2\right), \\ x^2 &= \sum_n \hat{y}_n \phi\left(x^1 - x_n^1, x^2 - x_n^2\right). \end{aligned}$$

Now denote the coefficients representing the x^1-component of the velocity field $\mathbf{u}$ as $\hat{u}_n^1$. Using the above formula for $\mathbf{f}$ we can find its corresponding coefficients as follows:

$$\hat{f}_n^1 = \hat{u}_n^1 - \mathbf{A}_{11}\hat{x}_n - \mathbf{A}_{12}\hat{y}_n + \left(\mathbf{A}_{11}\mathbf{x}^{1(n)} + \mathbf{A}_{12}\mathbf{x}^{2(n)} + \dot{\mathbf{x}}^{1(n)}\right)\hat{k}_n, \qquad (16)$$
$$\hat{f}_n^2 = \hat{u}_n^2 - \mathbf{A}_{21}\hat{x}_n - \mathbf{A}_{22}\hat{y}_n + \left(\mathbf{A}_{21}\mathbf{x}^{1(n)} + \mathbf{A}_{22}\mathbf{x}^{2(n)} + \dot{\mathbf{x}}^{2(n)}\right)\hat{k}_n.$$

Thus we are able to compute the interpolation of **f** indirectly. The advantage of staying in the "coefficient space" is that equation (16) is a formula; it avoids the need for the expensive interpolation calculation of equation (3). This is my own contribution independent of the work of Wiggins.

Now that we have the equation in linearized form, the next step is diagonalize **A***(t)*. Construct a time-dependant transformation $\mathbf{T}^{(n)}(t)$ such that under the coordinate transformation $w = \mathbf{T}^{(n)}(t)y$ the matrix $\mathbf{A}(t)$ is diagonalized. Denote the diagonalized matrix as **D**. Finding $\mathbf{T}^{(n)}(t)$ involves the singular value decomposition of the fundamental solution matrix **X.** To first compute the fundamental solution matrix a matrix version of simple 4th – order Runge-Kutta was used. Given that $\mathbf{X}(0)=\mathbf{I}$, we find $\mathbf{X}(t_{i+1})$ based on $\mathbf{X}(t_i)$ by the following formula:

$$\begin{aligned}
\mathbf{K}_1 &= \mathbf{A}(t_i)\mathbf{X}(t_i), \qquad (17)\\
\mathbf{K}_2 &= \frac{1}{2}\left[\mathbf{A}(t_i) + \mathbf{A}(t_{i+1})\right]\left[\mathbf{X}(t_i) + \frac{h}{2}\mathbf{K}_1\right],\\
\mathbf{K}_3 &= \frac{1}{2}\left[\mathbf{A}(t_i) + \mathbf{A}(t_{i+1})\right]\left[\mathbf{X}(t_i) + \frac{h}{2}\mathbf{K}_2\right],\\
\mathbf{K}_4 &= \mathbf{A}(t_{i+1})\left[\mathbf{X}(t_i) + h\mathbf{K}_3\right],\\
\mathbf{X}(t_{i+1}) &= \mathbf{X}(t_i) + \frac{h}{6}\left[\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4\right].
\end{aligned}$$

Once we have **X** in hand we compute its singular value decomposition at each instant in time, $\mathbf{X}=\mathbf{BSR}^T$. Both **R** and **B** are orthogonal matrices, e.g. $\mathbf{B}^T = \mathbf{B}^{-1}$, and **S** is a diagonal matrix. Note that the singular value decomposition is not unique because **R** and **B** are not generally the same matrix, as is the case in simple diagonalization. Because of this fact we can guarantee that the matrix **S** is positive definite (MALTABs built-in function svd does this). We can write **S** as

$e^{\mathbf{\Sigma}}$, using the usual matrix exponential. Thus $\mathbf{X}(t) = \mathbf{B}(t)\, e^{\mathbf{\Sigma}(t)}\mathbf{R}(t)^{\mathrm{T}}$**.** According to [4], the transformation **T** that diagonalizes $\mathbf{A}(t)$ can be constructed from **B R** and **Σ** as follows: $\mathbf{T}(t) = e^{t\mathbf{D}}\mathbf{R}^{\mathrm{T}}(t_L)\mathbf{R}(t)\, e^{\mathbf{\Sigma}(t)}\mathbf{B}(t)^{\mathrm{T}}$. $\mathbf{D} = \mathbf{\Sigma}(t_L)/\, t_L$ Let us assume, without loss of generality, that $\mathbf{D}_{11} > 0$ and $\mathbf{D}_{22} < 0$. The next iteration of the distinguished hyperbolic trajectory can be computed using the following formula:

$$
\begin{aligned}
w^{1,(n+1)} &= -e^{D_{11}t}\int_{t}^{\infty} e^{-D_{11}s}\, g^{1}\left(\mathbf{x}^{(n)}(s), s\right)ds, \\
w^{2,(n+1)} &= e^{D_{22}t}\int_{\infty}^{t} e^{-D_{22}s}\, g^{2}\left(\mathbf{x}^{(n)}(s), s\right)ds, \\
w &= \mathbf{T}(t)y, \\
\mathbf{g}(x,t) &= \mathbf{T}(t)\mathbf{f}(x,t).
\end{aligned}
\tag{18}
$$

The integral of equation (18) is computed numerically using the trapezoid rule. This gives us the next iteration of the DHT calculation in w-coordinates. To move back to the original x-coordinates multiply by $\mathbf{T}^{-1}(t)$ and add $\mathbf{x}^{(n)}$ to obtain $\mathbf{x}^{(n+1)}$. The algorithm continues iteratively until $\| \mathbf{x}^{(n+1)} - \mathbf{x}^{(n)} \| < tolerance$.

Wiggins notes in [4], that because of the exponential nature of the equation $\dot{\mathbf{X}} = \mathbf{A}(t)\mathbf{X}$, it is possible that solving for **X** in this manner risks overrunning machine arithmetic. The matrix Runge-Kutta method of (17) will therefore work only for a limited time. It is possible to solve for the matrix **X** in terms of the components of its singular value decomposition, as Wiggins suggests in appendix B of [4]. This leads to equation (19)

$$\dot{\mathbf{X}} = \mathbf{A}(t)\mathbf{X}, \tag{19}$$
$$\dot{\mathbf{B}}e^{\boldsymbol{\Sigma}}\mathbf{R} + \mathbf{B}\dot{\boldsymbol{\Sigma}}e^{\boldsymbol{\Sigma}}\mathbf{R} + \mathbf{B}e^{\boldsymbol{\Sigma}}\dot{\mathbf{R}} = \mathbf{A}\mathbf{B}e^{\boldsymbol{\Sigma}}\mathbf{R}$$
$$\mathbf{B}^{\mathrm{T}}\left(\dot{\mathbf{B}}e^{\boldsymbol{\Sigma}}\mathbf{R} + \mathbf{B}\dot{\boldsymbol{\Sigma}}e^{\boldsymbol{\Sigma}}\mathbf{R} + \mathbf{B}e^{\boldsymbol{\Sigma}}\dot{\mathbf{R}}\right)\mathbf{R}^{\mathrm{T}}e^{-\boldsymbol{\Sigma}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}\mathbf{B}e^{\boldsymbol{\Sigma}}\mathbf{R}\mathbf{R}^{\mathrm{T}}e^{-\boldsymbol{\Sigma}}$$
$$\mathbf{B}^{\mathrm{T}}\dot{\mathbf{B}} + \dot{\boldsymbol{\Sigma}} + e^{\boldsymbol{\Sigma}}\dot{\mathbf{R}}\mathbf{R}^{\mathrm{T}}e^{-\boldsymbol{\Sigma}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}\mathbf{B}.$$

Now parameterize **B, R and Σ** in the following manner

$$\mathbf{B} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}, \tag{20}$$
$$\mathbf{R} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix},$$
$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}.$$

This leads to a set of four differential equations in four unknowns. Let $\mathbf{H}=\mathbf{B}^{\mathrm{T}}\mathbf{A}\mathbf{B}$.

$$\dot{\theta} = \frac{1}{2}(H_{12} - H_{21}) + \frac{1}{2}(H_{12} + H_{21})\coth(\sigma_2 - \sigma_1), \tag{21}$$
$$\dot{\phi} = \frac{1}{2}(H_{12} + H_{21})\operatorname{csch}(\sigma_2 - \sigma_1),$$
$$\dot{\sigma}_1 = H_{11},$$
$$\dot{\sigma}_2 = H_{22}.$$

The components of these equations generally increase linearly in time, as opposed to exponentially with equation (18). This allows for the solution of **X** and **T** over longer-length data sets. This improvement proved to be invaluable when seeking to analyze data sets weeks, even months in duration.

PART II: Results

The application of Wiggins' DHT algorithm to the Chesapeake Bay was remarkably difficult. The chief difficulty arises because the instantaneous stagnation points (ISPs) that Wiggins uses as the first guess $\mathbf{x}^{(0)}$ in his algorithm simply do not exist in the Bay. The ISP curve, denoted $\boldsymbol{x}_{ISP}(t)$, is not a solution of the system $\dot{\mathbf{x}} = \mathbf{u}(\mathbf{x},t)$, but rather a curve of points of zero velocity, that is $\mathbf{u}(\boldsymbol{x}_{ISP}(t),t) = \mathbf{0}$. Continuous curve of this type exist in the toy problems presented in [3], such as the rotating Duffing oscillator. Examining this problem, success was achieved computing the ISPs and the DHT after successive iterations of the scheme described in Part I. However, when similar techniques were tried in order to analyze the Chesapeake, they were unsuccessful. The problem is that the constant periodic forcing of the tides keeps everything moving. The regions of zero velocity that do appear are short-lived. These regions have a timescale of only a few hours and are therefore useless for computation of hyperbolic trajectories.

In the absence of an ISP curve, trajectories were computed in the manner described in section I and used for the initial $\mathbf{x}^{(0)}$ guess. Trajectories in many different locations throughout the Chesapeake were tested. The only region in which *any* initial guess ever resulted in convergence of equation (17) was the narrow neck depicted in Figure 20. This is also the location of the Chesapeake Bay bridge connecting mainland Maryland to Kent Island and the Eastern Shore.

Most of the trajectories that were tried resulted in divergence of equation (18). The distance between successive iterations $\| \mathbf{x}^{(n+1)} - \mathbf{x}^{(n)} \|_{\infty}$, defined as the maximum distance between the curves at any given time, grew without bound until the resulting curve was far outside the

boundaries of the Bay. After searching for the hyperbolic stretching which is characteristic of the existence of an unstable manifold, and hence a hyperbolic trajectory, it became evident that the Kent Island region was again the only region that exhibited this stretching. It was here that initial guess trajectories were computed in the search for a DHT. Ultimately one of the "guess" trajectories resulted in convergence. This trajectory is depicted in Figure 20. The black square just above the 4330000 Northing line is the origin of the trajectory, it ends near the bottom of the figure. This trajectory is 21 days long.

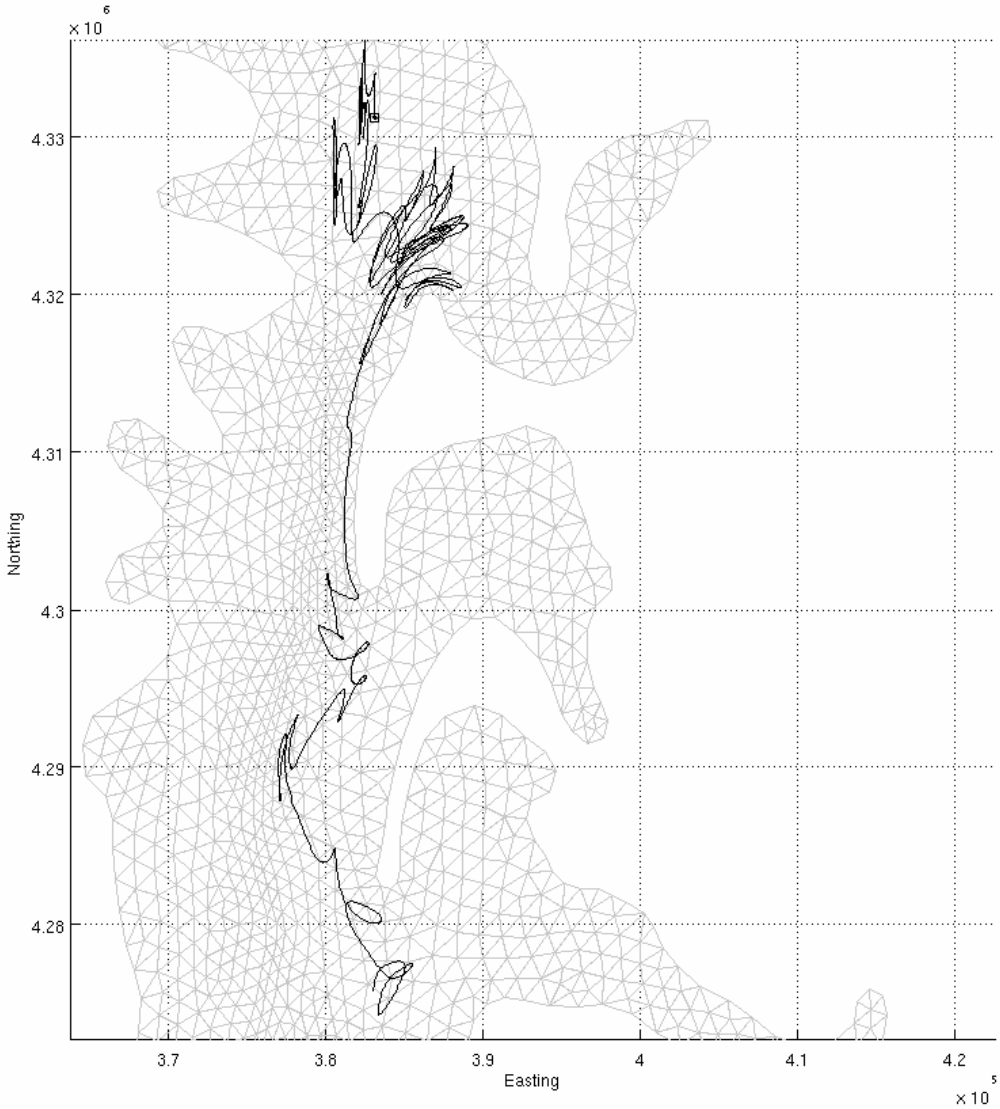


Figure 20: Hyperbolic trajectory of the Chesapeake Bay.

Iterative refinement of the guess trajectory resulted in little change. Where as with divergent guesses differences $\| \mathbf{x}^{(n+1)} - \mathbf{x}^{(n)} \|_\infty$ measured in tens of kilometers or more were observed, the maximum distance between the guess trajectory, and the result of the DHT algorithm was 290 meters. After 10 iterations the distance between successive $\mathbf{x}^{(n)}$ was on the order of tens of centimeters. This is particularly small when compared to the roughly 50 kilometer span of the resulting DHT.

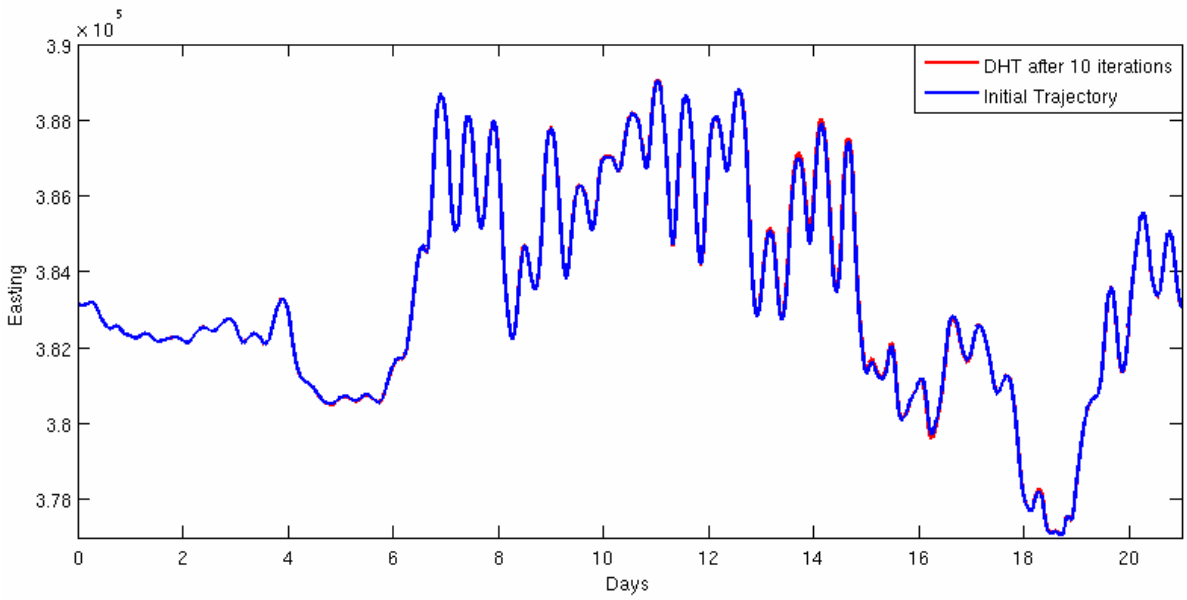


Figure 21: Initial guess trajectory and solution after 10 iterations.

Figure 21 shows the Easting coordinate vs. time of the initial trajectory in blue, and the same for the final result in red. The red line is difficult to see because of the degree of overlap. Between Day 14 and 16 the small difference between initial guess and final solution is somewhat visible. A fixed point of equation (18) to had effectively been guessed to within 300 meters.

Also of note is the instability inherent in the DHT solution for the Chesapeake Bay. While for simpler systems convergence of equation (18) is robust, that is nearby first guesses result in convergence to the same final result, the same is not true of the Chesapeake. When the

initial guess is shifted to the west by 50 meters, the DHT algorithm does not converge. The initial distance $\| \mathbf{x}^{(1)} - \mathbf{x}^{(0)} \|$ was 1.2 km and the distances grow with each successive iteration. Figure 22 shows a similar plot with the Easting coordinate of the westward shifted trajectory, and the Easting coordinate of the curve after 3 iterations. Note the large divergence near Day 12 as well as the sharp direction reversals near Day 15 and 17. These kinks are characteristic of solutions that quickly diverge to nonsense. At this point, after only three solutions, the result was already outside the bounds of the Chesapeake Bay.

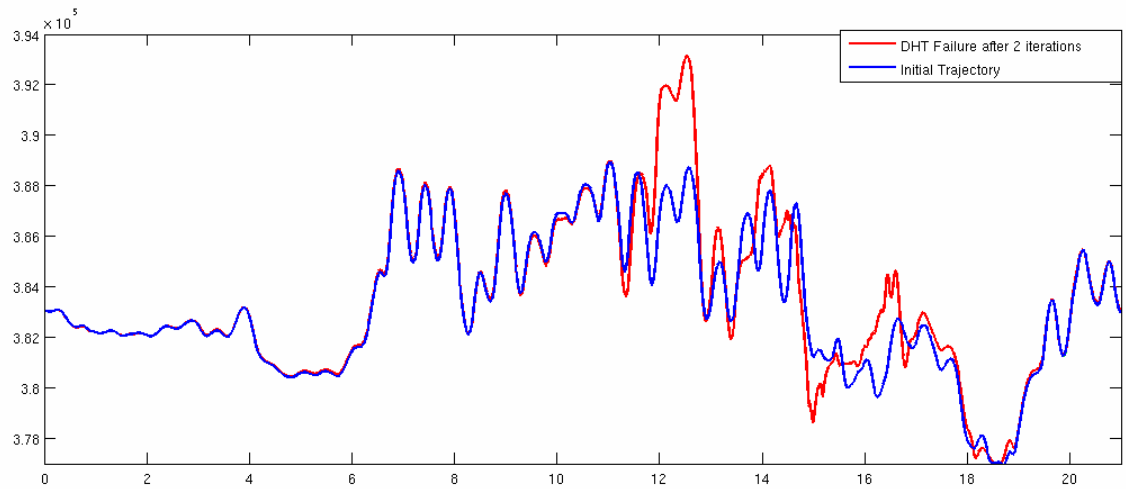


Figure 22: Perturbed initial trajectory and divergence after 3 iterations.

In light of these difficulties one might infer that there were problems with Wiggins' algorithm or, at the very least, errors in the implementation. However, the same algorithm, when applied to systems where the DHT solution was well known, such as the rotating Duffing oscillator, worked flawlessly. The resultant solutions were within the analytical solution to the order of 10^{-6}. Additionally, the success of computation of the stable and unstable manifolds of the hyperbolic trajectory from Figure 20 confirms the correctness of the algorithm. The resulting solution is indeed a DHT. Unfortunately, because of the instabilities shown by Figure 22, the DHT algorithm becomes something of a mere confirmation. The algorithm indicates when a

DHT has been accurately guessed, but is not an effective method for finding that trajectory. This suggests that additional work needs to be done on defining criteria of convergence for equation (18) as well as good criteria for an initial guess in the absence of a well-defined curve of instantaneous stagnation points.

There is also some question as to why the region about Kent Island is the only region for which hyperbolic trajectory computation succeeds. One idea is that of the connection to the normal modes of Lipphardt et al. [11]. Normal modes define flow patterns for a region analogous to the frequencies of a vibrating string. A string, like that of a violin, has distinct frequencies at which it vibrates. Any tone produced by this string can be expressed as a linear combination of these frequencies. Similarly, “frequencies”, or normal modes, for a littoral region can be computed defining flow patterns. Any flow pattern can be defined as a linear combination of the normal modes. These normal modes are typically large scale circulations. A narrow point such as the Kent Island channel would tend to separate these circulations, creating a clockwise rotation to the north and an opposite counter-clockwise circulation to the south. Such counter-rotating vortices create the hyperbolic separation necessary for hyperbolic trajectories and their respective stable and unstable manifolds to exist. These counter-rotating vortices are exactly the kind of hyperbolic behavior that Wiggins investigates in his work with the wind-driven double gyre model [1]. Viewed in this way, it makes sense that the narrowest region in the Chesapeake would also produce the greatest hyperbolic separation and the only region where DHT computation was successful.

Figure 23 illustrates the idea of hyperbolic separation generated by the normal modes. This normal mode was computed by MIDN 1/C Grant Gillary as part of [14]. Note that the two

gyres split at the northern point of Kent Island creating a hyperbolic point denoted by the “H”. The hyperbolic trajectory of Figure 20 remains in precisely this region for over a week.

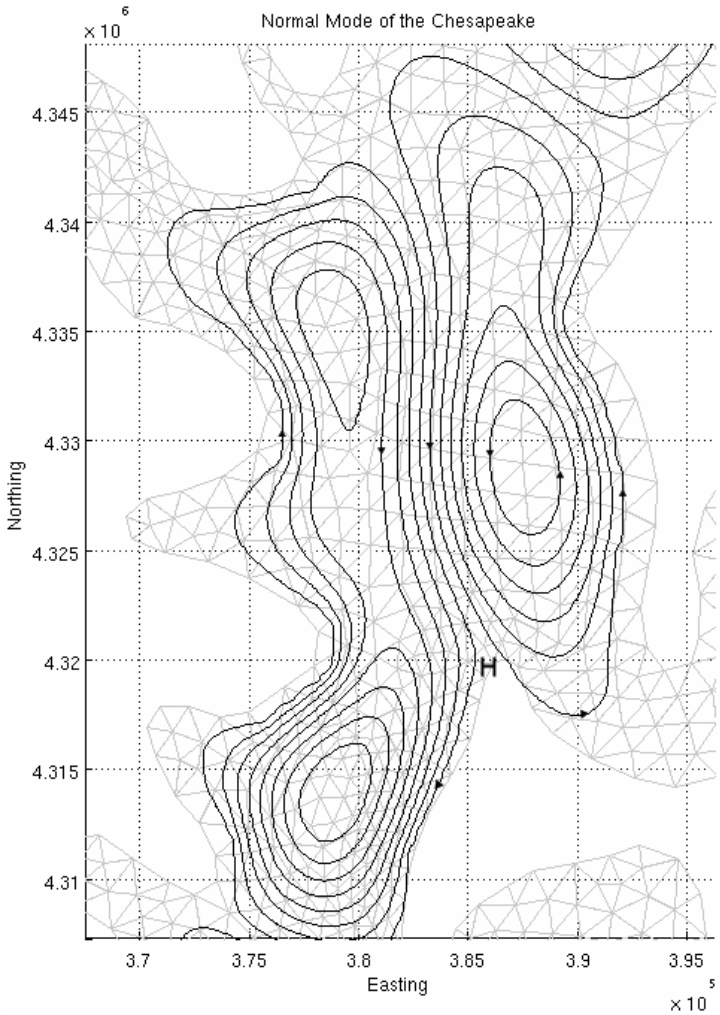


Figure 23: Hyperbolic point created by normal modes.

Much research needs to done still in the area of hyperbolic trajectory computation for dynamical systems as complex as the Chesapeake Bay. At the time of this writing only a single solution of equation (18) has been found. This is due to the instability difficulties that were described. Rigorous mathematical study needs to be conducted to describe why the Kent Island region is the only hyperbolic region in the Bay. If the hypothesized connection to Normal Modes

is accurate, it would make sense that a similar narrow neck in other regions would similarly be regions of high hyperbolic separation. This is a direction for future research

INVARIANT MANIFOLD COMPUTATION

PART I: Methodology

Once the DHT has been computed in the manner of the previous section, it is possible to compute the stable and unstable manifolds. The stable manifold of a hyperbolic trajectory $\mathbf{x}_H(t)$ is stable in the sense that every point along the manifold, denoted by $W_{loc}^{s}(\mathbf{x}_H(t),t)$, will collapse back to some arbitrarily small neighborhood of $\mathbf{x}_H(t)$ in forward time. Similarly, every point along the length of the unstable manifold $W_{loc}^{u}(\mathbf{x}_H(t),t)$ will collapse back to the hyperbolic trajectory in backward time. The manifolds are invariant in the sense that every point starting along either manifold will remain on it for all time. This gives them the special property of being a barrier to transport. When plotted the invariant manifolds mark off regions, or lobes, and dictate the mass transport dynamics between these lobes. Below is a plot of the stable and unstable manifolds of the system defined by equation (22).

$$\dot{x} = x,\ \dot{y} = x^2 - y. \qquad (22)$$

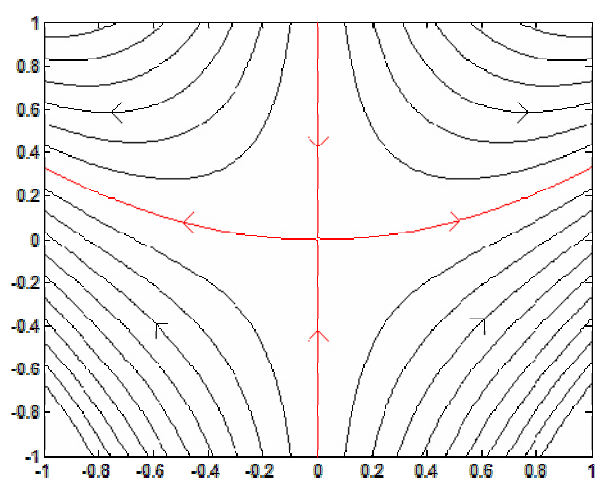


Figure 24: Stable and unstable manifolds of the system defined by equation (22).

To numerically compute the invariant manifolds, one first computes the eigenvalues and eigenvectors of the matrix **A** as defined in equation (10). As **A** represents the linear behavior of the vector field around the hyperbolic trajectory, the matrix **A** will have one positive and one negative eigenvalue. The eigenvector corresponding to the positive eigenvalue represents the direction of maximum stretching, and defines the initial direction of the unstable manifold. Likewise, the eigenvector corresponding to the negative eigenvalue represents the direction of maximum compression and the initial direction of the stable manifold.

For the unstable manifold, once the direction of the initial segment has been calculated, 5 discrete points, or manifold nodes, are placed along this vector. In the case of the Chesapeake Bay these initial nodes are places at intervals of 500m. These nodes are advected with the velocity field forward in time. As they stretch apart, new nodes are inserted to capture all the folds and structure of the unstable manifold.

Computation of the stable manifold is analogous. However, the procedure is done in reverse. The nodes are placed along the initial segment which is calculated at the end of the data set. The nodes are then evolved backwards in time. The procedure for inserting new points is the same as with the unstable manifold.

All that is needed, then, is an accurate method of deciding when and where to insert the new nodes to maintain integrity of the numerical procedure. The method cited in [2] as being the most effective when applied to oceanographic flows is an interpolation and insertion scheme originally developed by Dritschel [5] for use in computing vorticity filaments.

The basic idea behind this insertion scheme is that regions in which the manifold is tightly folded, i.e. regions of high curvature, need a higher density of nodes to properly capture

the dynamics in this region. Regions of lower curvature can tolerate a lower density, reducing the total number of nodes and the total computation time.

Given that the unstable manifold is approximated by a series of N nodes, calculate the local curvature for each node based on the position of its two neighbors. Local curvature is defined as the reciprocal of the radius of the circumcircle described by these three points.

$$W_{loc}^{u}\left(x_H(t),t\right) \sim \{x_1, x_2, x_3 \mathrm{K}\ x_i \mathrm{K}\ x_N\}, \tag{23}$$
$$\kappa_i = \frac{\left\|\left(x_i - x_{i-1}\right)\times\left(x_{i+1} - x_i\right)\right\|}{\left\|\left\|x_i - x_{i-1}\right\|^2\left(x_{i+1} - x_i\right) + \left\|x_{i+1} - x_i\right\|^2\left(x_i - x_{i-1}\right)\right\|}.$$

A weighted averaged of curvature is defined along the manifold.

$$\overset{(}{\kappa}_i = \frac{w_{i-1}\kappa_{i-1} + w_i\kappa_i}{w_{i-1} + w_i}, \qquad w_i = \frac{\left\|x_{i+1} - x_i\right\|}{\left\|x_{i+1} - x_i\right\|^2 + \delta^2}, \tag{24}$$
$$\bar{\kappa}_i = \frac{1}{2}\left(\overset{(}{\kappa}_i + \overset{(}{\kappa}_{i+1}\right).$$

Now a point density function is defined as a function of the curvature $\bar{\kappa}_i$ and the distance between successive nodes,

$$\sigma_i = \left(\mu\sqrt{\bar{\kappa}_i} + \bar{\kappa}_i\right)\left\|x_{i+1} - x_i\right\|. \tag{25}$$

In this equation the constant μ is an arbitrary constant that is changed to increase or decrease the resolution of the manifold. Once the density σ is defined, we seek to distribute nodes such that σ is relatively constant and less than 1 along the entire length of the manifold. To do this an interpolation method is needed.

For this project cubic splines were used as the method of interpolation. MATLAB has a built-in function for computing cubic splines which made this option attractive in terms of ease of coding and computational speed. In [2] Wiggins suggests the interpolation method of

Dritschel and Ambaum [6] that matches an approximation of curvature at each node, where cubic splines match first and second derivatives. The aim of the Dritschel-Ambaum interpolation was to produce a smooth value of the curvature along the manifold to work with the density function of equation (24). However, remembering that the curvature of a continuous curve is defined in terms only of its first and second derivatives, cubic splines give a continuous value for curvature along the manifold if not as smooth as the method due to Dritschel and Ambaum. Having decided that the computational advantages of the MATLAB spline package outweighed the smoothness advantage of Dritschel-Ambaum interpolation, the former was chosen.

Once the manifold has been advected forward in time, gaps in the approximation and new folds develop that need to be addressed. First the density σ is calculated for the entire length of the manifold and a cubic spline interpolation is calculated. Then new nodes are distributed along the manifold in the following manner. Define N_{new} as the number of new nodes to be distributed in order to ensure that σ is less than 1

$$N_{new} = \left\lfloor \sum_{i=1}^{N_{old}} \sigma_i \right\rfloor + 2 \,. \tag{26}$$

Now place new node x_i between old nodes x_j and x_{j+1} such that

$$\sum_{k=1}^{j-1} \sigma_k + p\sigma_j = \left(\frac{N_{old}}{N_{new}} \right)(i-1). \tag{27}$$

Here p is a parameter for the interpolation. If we parameterize the old curve as $x(s)$ where $x(j) = x_j$, the new node x_i is placed at $x(j+p)$. Once the new nodes have been placed according to the interpolation and insertion scheme described above, the new nodes are advected forward in time and the process is repeated.

PART II: Results

Below is a plot of the stable manifold at Day 3 computed using the method of Section I. Because the stable manifold starts at the end of the data set and is computed backwards, it is well developed early in the data set.

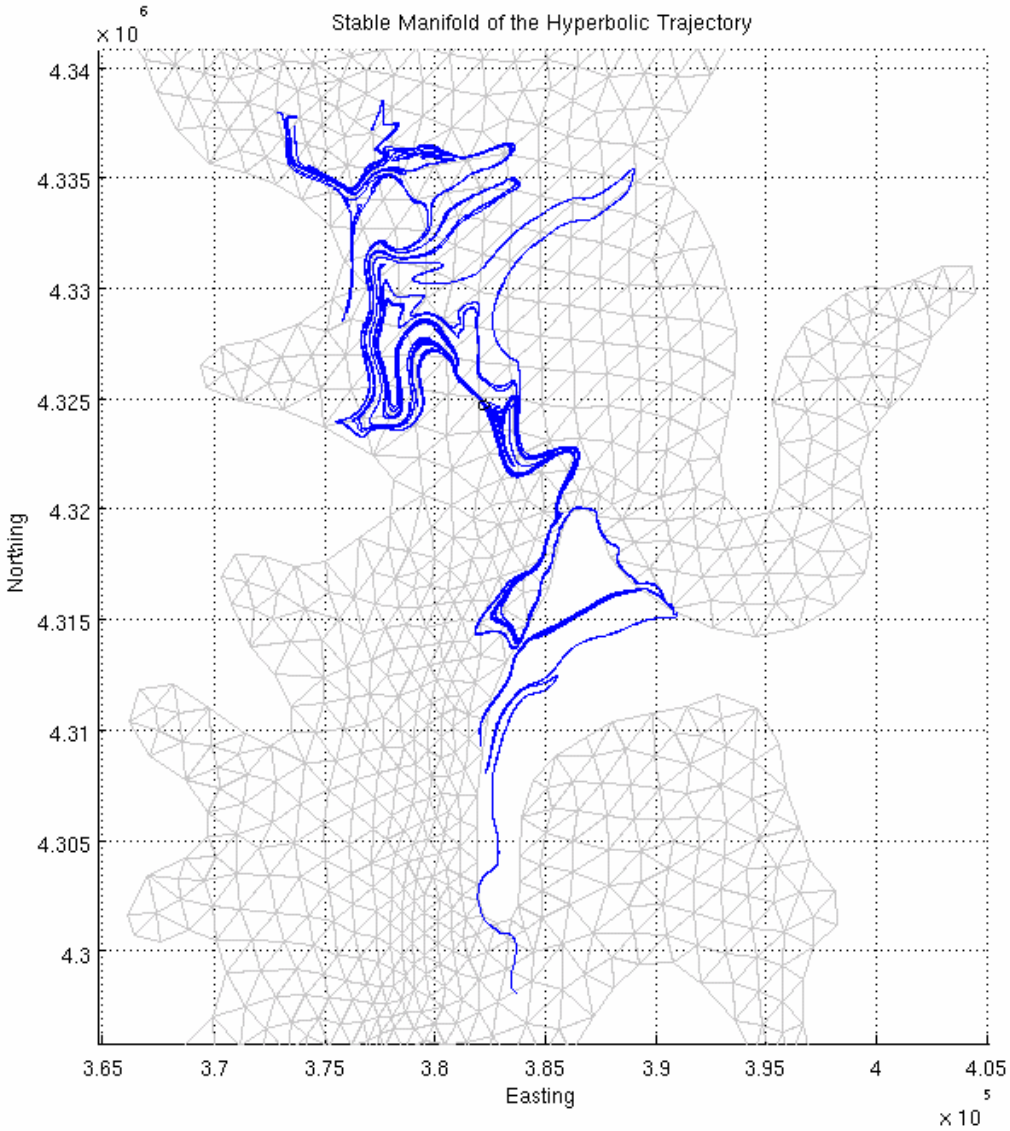


Figure 25: Stable manifold on Day 3.

Note the interaction of the stable manifold with the coast. It has not yet been determined why this happens or what the significance of this interaction is. Research needs to be done, starting with simpler dynamical systems, to determine the significance of this coastal interaction.

The following figures show the unstable manifold of the same hyperbolic trajectory and enlargements of the regions surrounding the trajectory on Day 21.

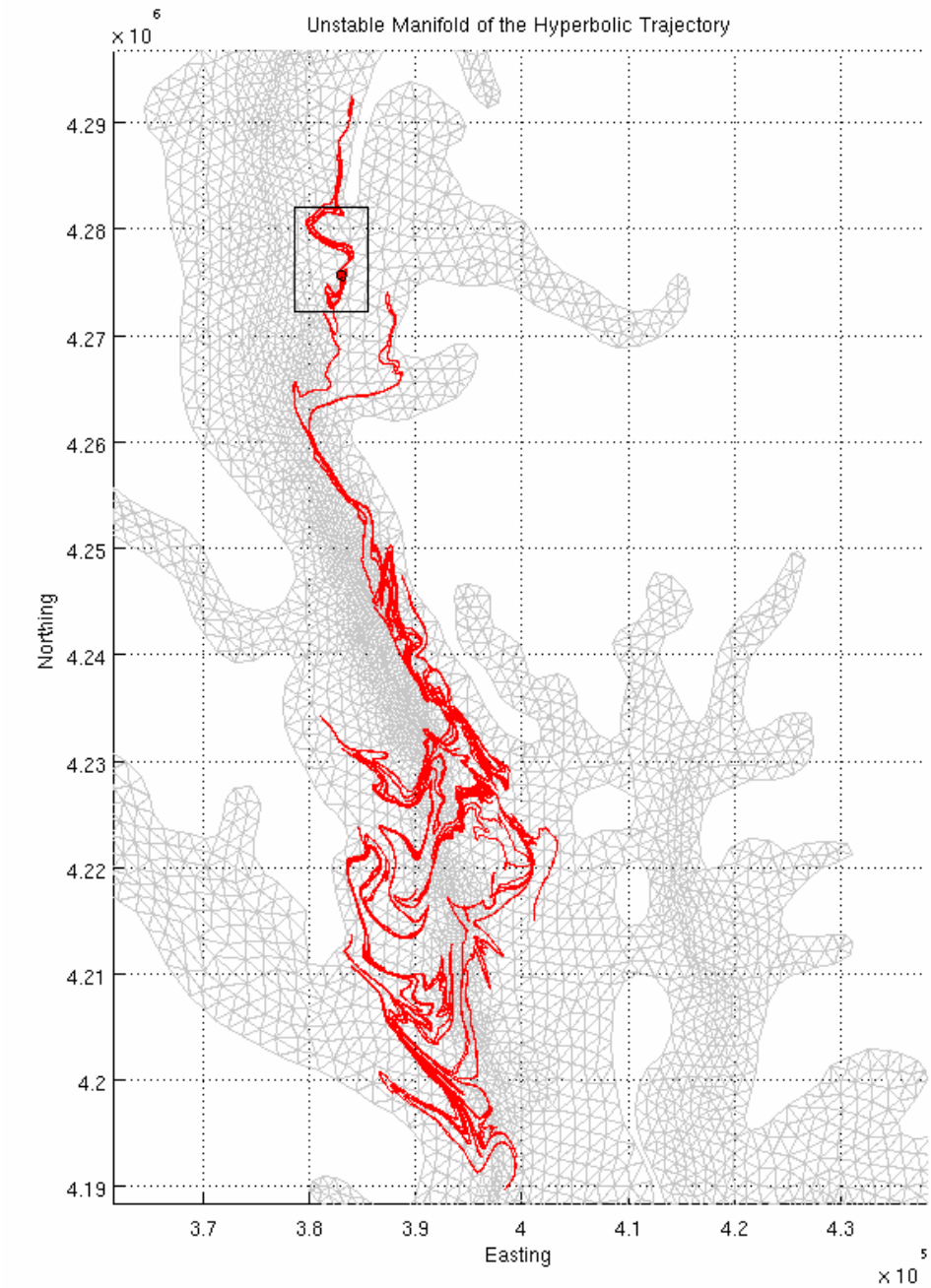


Figure 26: Unstable manifold on Day 21.

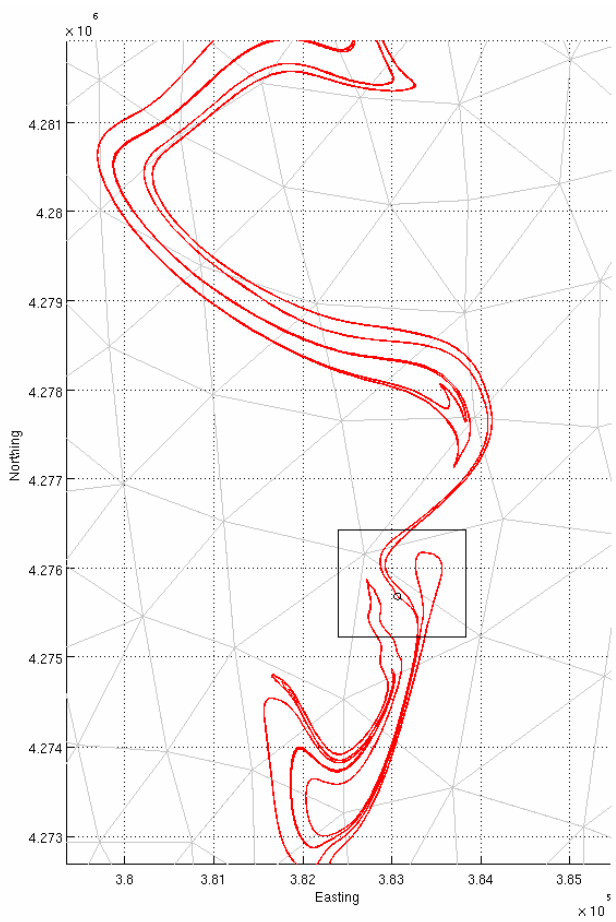


Figure 27: Enlargement of region around the hyperbolic trajectory.

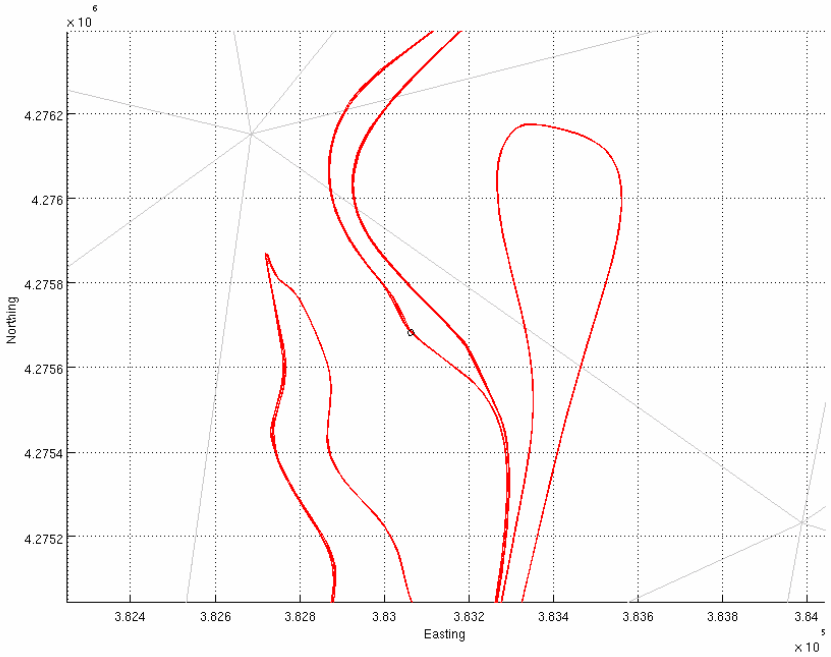
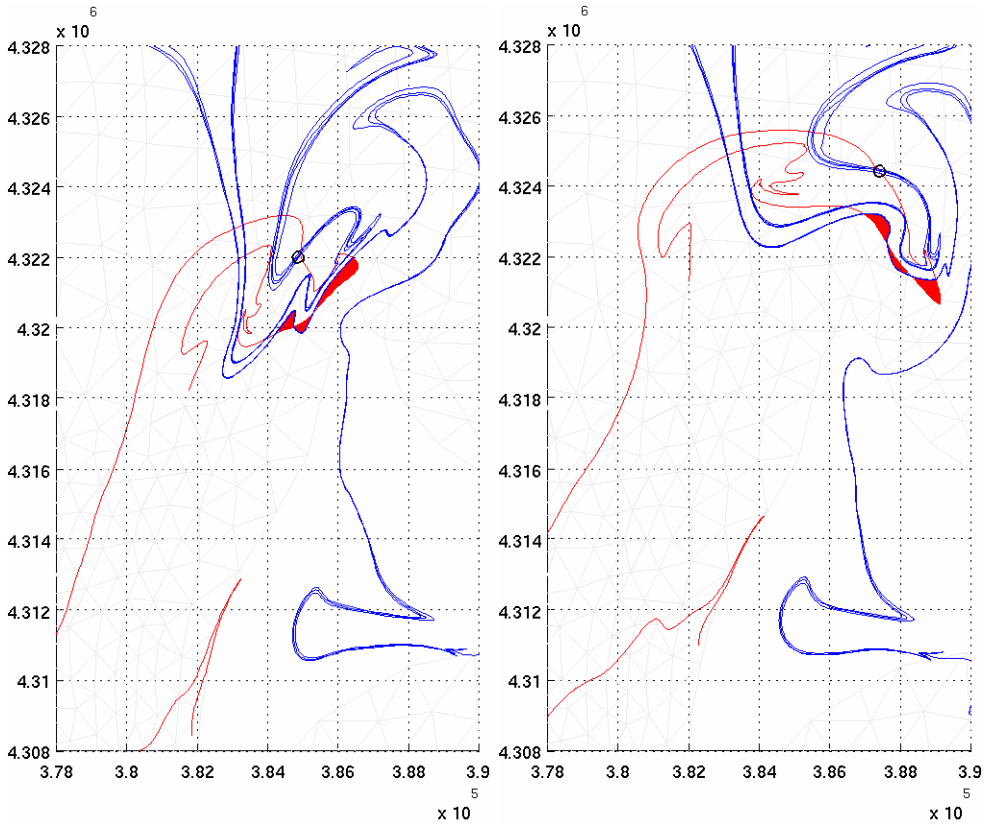
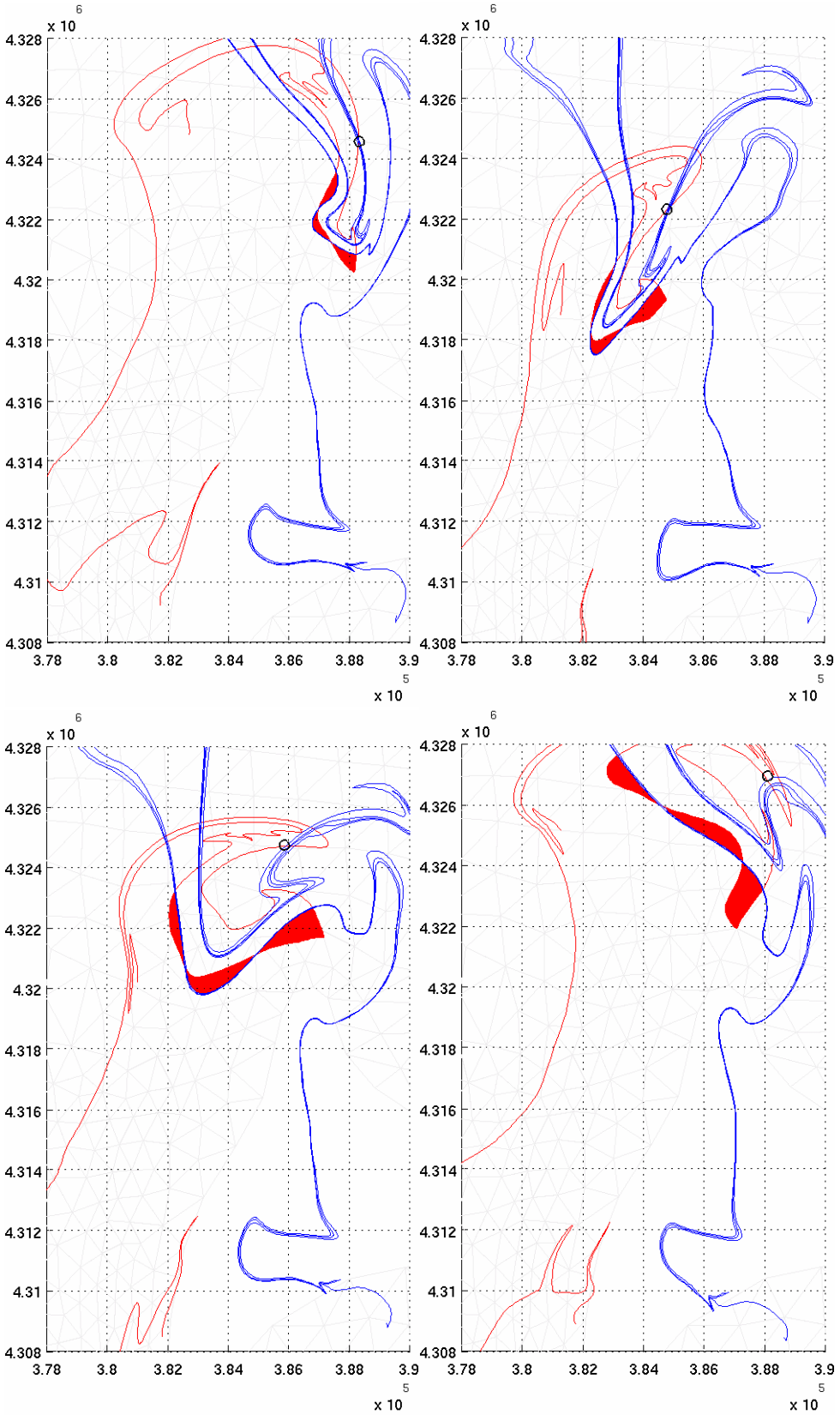
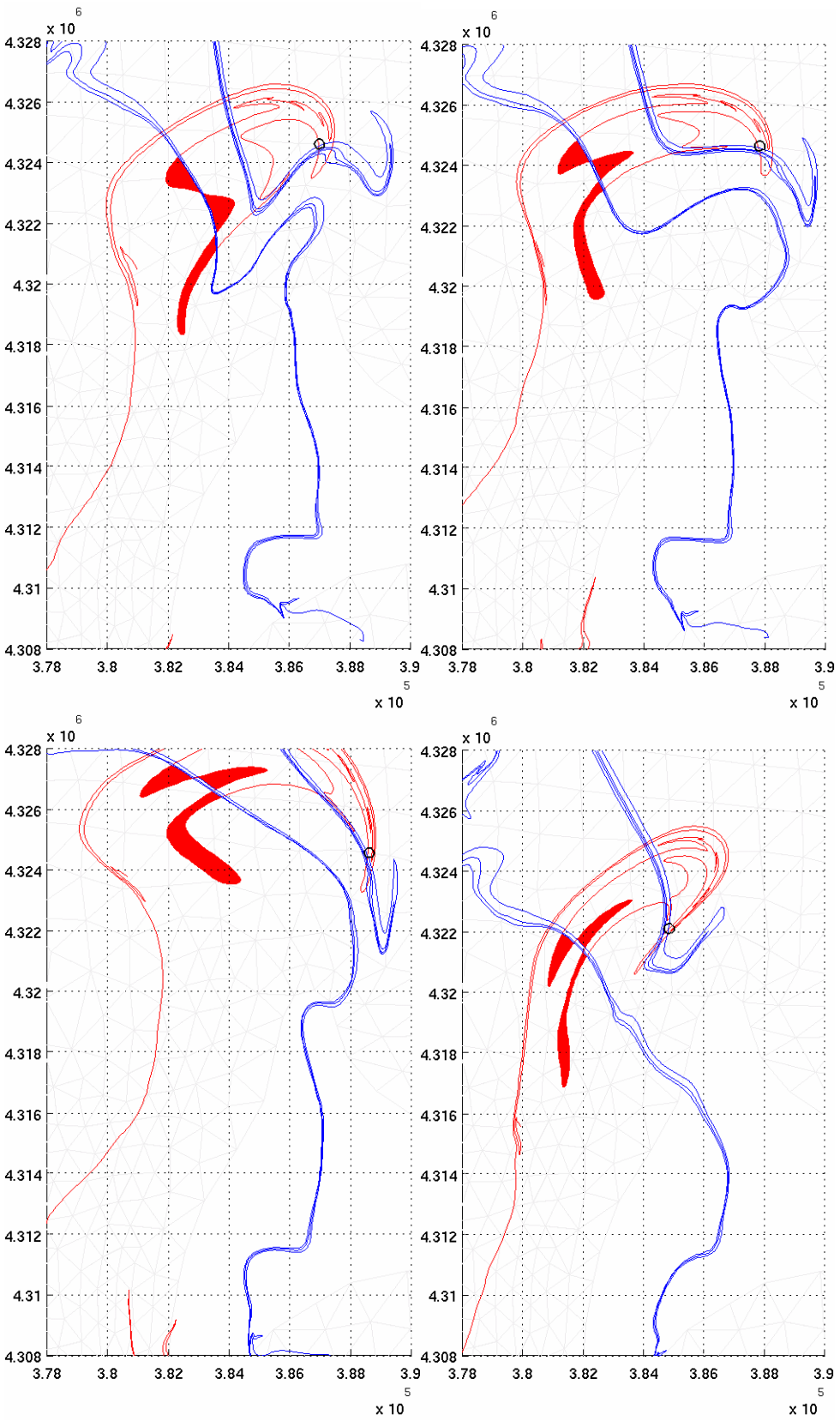


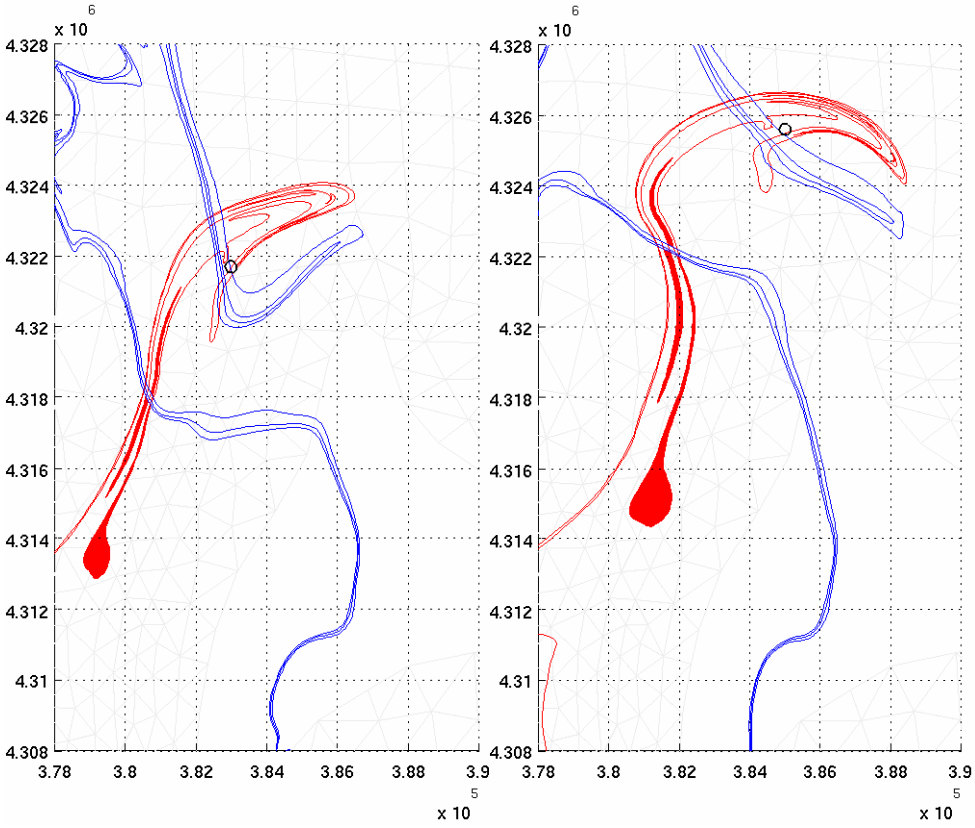
Figure 28: Successive enlargement.

As described in the previous section these curves mark off lobes. When the stable and unstable manifolds intersect, the resultant lobes typically stretch into long filaments. This is an example of what Wiggins describes as "irreversible" mass transport [7]. As the stable manifold contracts back to the neighborhood of the hyperbolic trajectory, one edge of the lobe is shortened. Simultaneously the unstable manifold is lengthening, stretching the other edge. The figures below show a time sequence of this dynamic. Successive figures are shown at 4 hour intervals beginning on Day 11. Note that the area within each lobe does not remain constant as it should in an incompressible fluid. This is because the surface layer is not incompressible in the sense that $\mathbf{u}_x+\mathbf{v}_y\neq\mathbf{0}$. The three-dimensional divergence $\mathbf{u}_x+\mathbf{v}_y+\mathbf{w}_z$ is equal to 0. Waves on the surface create a non-zero $\mathbf{w}_z$ term which allows for apparent non-zero divergence when viewing only the surface currents.




x 10^6
4.328
4.326
4.324
4.322
4.32
4.318
4.316
4.314
4.312
4.31
4.308
3.78
3.8
3.82
3.84
3.86
3.88
3.9
x 10^5


x 10^6
4.328
4.326
4.324
4.322
4.32
4.318
4.316
4.314
4.312
4.31
4.308
3.78
3.8
3.82
3.84
3.86
3.88
3.9
x 10^5



Here the time evolution and stretching of the lobes is clearly evident. The easternmost lobe shows a fluid jet, rather than simply stretching, the material in this region is flung away from the hyperbolic trajectory. This jet took 48 hrs to form. Material advection in this time scale is not easily visible when examining frozen-time velocity fields. Only after computing the invariant manifolds is this behavior evident.

Certainly, upon examining other regions of the manifolds similar examples of lobe dynamics would be observed. The computation of manifolds for a 21-day period only scratches the surface of the analysis of which this method is capable. The rich dynamics that have been made visible by examining these manifolds makes it clear that there are structures and dynamics hidden within a data set as complex as this. Future research is necessary to examine the manifolds properly and the structures that they form. The difficulties of the hyperbolic trajectory

algorithm currently impede considerably the progress of this research. However, since it is known that the Kent Island neck produces the greatest hyperbolicity, trying trajectories at later times in the data set are likely to also converge to solutions of equation (18).

Additionally, the numbers of nodes in both the stable and unstable manifold grow exponentially as the computation progresses. Because of this fact, it is impractical to compute manifolds with adequate resolution for time lengths longer than three weeks. A parallel computing implementation of Wiggins' algorithm could improve this. The investigation of invariant manifold behavior has merit and demands further research.

Conclusions

My initial goal was to simply implement the algorithms of Wiggins in MATLAB, freeing them of the platform issues and difficulties associated with conventional computer code. Along the way I successfully reproduced hyperbolic trajectory and invariant manifold results for the rotating Duffing system and a host of smaller systems. Ultimately, I was also able to produce results for the QUODDY simulation of the Chesapeake Bay. In the early spring I had my doubts as to whether the algorithms would be applied successfully to the Chesapeake. Wiggins, in an e-mail to my adviser, expressed similar doubts and abandoned this approach to analyzing the Bay while mentioning SLMs as a possible alternative approach. As I took time out to learn about the computation of synoptic Lagrangian maps I noticed large inefficiencies in the current methods of computation. Eliminating these led to my improved algorithm. This new algorithm also allowed me to compute SLMs on single-processor workstations, freeing them from the need for the computational power of a large supercomputer. This, combined with a MATLAB implementation, makes SLM computation accessible to many more researchers than was previously possible. Eventually the invariant manifold analysis was similarly successful and it produced the results that are presented here. I certainly met my original goals and far exceeded them. In the process I was able to learn much and devise many useful results.

The work I have done opens the door for many future research avenues. Most important is to create a data assimilation scheme, similar to that used by Lipphardt et al. for the Monterey Bay, and apply it to the Chesapeake. This would allow the algorithms I developed to be applied to more realistic data. I have little doubt that the QUODDY model is accurate, and that the results would be similar. However, the resulting validation of my findings would be quite valuable. Also there are a few open questions as a result of my research that bear investigation,

most notably the difficulties inherent in DHT computation for the Chesapeake Bay and the interaction of the stable manifold with the Chesapeake coastline.

I am ultimately very grateful to my advisers and numerous others for the advice and guidance that they provided. I am similarly grateful to Prof. Shade for the program that she has helped to develop which gave me the opportunity to grow and learn as much as I have. My understanding of transport in the Chesapeake Bay, and dynamical systems in general, grew in leaps and bounds as this project progressed. Yet, at the end, there are as many unanswered questions and exciting new research directions as there were when I started.

REFERENCES

[1] Mancho A., Small D., and Wiggins S. : *Computation of hyperbolic trajectories and their stable and unstable manifolds for oceanographic flows represented as data sets*, Nonlinear Processes in Geophysics (2004) 11: 17–33

[2] Mancho, A., Small, D., Wiggins, S., and Ide, K.: *Computation of stable and unstable manifolds of hyperbolic trajectories in two-dimensional, aperiodically time-dependent vector fields*, Physica D, 182, 188–222, 2003.

[3] Wiggins, S. et al.: *Existence and Computation of Hyperbolic Trajectories of Aperiodically Time Dependent Vector Fields and Their Approximations*, International Jounal of Bifurcation and Chaos ,Vol. 13, 6, 1449-1457, 2003.

[4] Wiggins, S. et al.: *Distinguished Hyperbolic Trajectories in Time Dependent Fluid Flows: Analytical and Computational Approach for Velocity Fields Defined as Data Sets*, Nonlinear Processes in Geophysics, 9, 237-263, 2002.

[5] Dritschel, D.: *Contour dynamics and contour surgery: Numerical algorithms for extended, high-resolution modelling of vortex dynamics in two-dimensional, inviscid, incompressible flows*, Comp. Phys. Rep., 10, 77–146, 1989.

[6] Dritschel, D. and Ambaum M.: *A contour-advective semi-Lagrangian numerical algorithm for simulating fine-scale conservative dynamic fields*, Quart. J. Roy. Meteor. Soc., 123, 1097-1123, 1997.

[7] Coulliette, C., Wiggins, S. *Intergyre Transport in a Wind-Driven, Quasigeostrophic Double Gyre: An Application of Lobe Dynamics*. Nonlinear Processes in Geophysics, **7**, 59-85, 2002.

[8] Lynch. D. and Werner F. *Three-Dimensional Hydrodynamics on Finite Elements. Part I: Linearized Harmonic Model*. Inter. Jour. Numer. Meth Fluids, 7, 871-909, 1987.

[9] Lynch. D. and Werner F. *Three-Dimensional Hydrodynamics on Finite Elements. Part II: Non-Linear Time-Stepping Model*. Inter. Jour. Numer. Meth Fluids, 12, 507-533, 1991.

[10] Lipphart B. and Kirwan D., *Synoptic Lagrangian Maps: Surface Particle Transport in the Coastal Ocean* <http://newark.cms.udel.edu/~brucel/slmaps>

[11] Kirwan D. et al., *Blending HF radar and model velocities in Monterey Bay through Normal Mode Analysis*, Journal of Geophysical Research, 105, 3425-3450, 2000.

[12] Wendland H., *On the smoothness of positive definite and radial functions*, Advances in Computational and Applied Mathematics, 101, 177-188, 1998.

[13] Wendland H., *Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree*, Advances in Computational Mathematics 4, 389-396, 1995.

[14] Gillary G., *Normal Mode Analysis of the Chesapeake Bay,* Trident Scholar Report, 2005.

[15] Jones C., *A Dynamics Group Looks at the Ocean*, SIAM News, 33, 2, 2002.

[16] Auer, S.J., *Five-year climatological survey of the Gulf Stream System and its associated rings,* Journal of Geophysical Research, 92, 11709-11726, 1987.

[17] Frankignoul, C., de Coetlogon G., Joyce T. M., and Dong S. F., *Gulf Stream variability and ocean-atmosphere interactions,* Journal of Physical Oceanography, 31, 3516-3529, 2001.

Appendix: MATLAB Code

Given below is the computer code to calculated the ϕ functions and their partial derivatives:

```
function z=phi(x,y,a)
%Wedlands C2 compactly supported radial basis function
R=sqrt(x.^2+y.^2)/a;
z=(R<1).*(4*R+1).*(1-R).^4;

function z=phix(x,y,a)
%Wedlands C2 compactly supported radial basis function
%Partial x-derivative
R=sqrt(x.^2+y.^2)/a;
z=(R<1).*(20*R.^3-60*R.^2+60*R-20).*(x./(a^2));

function z=phiy(x,y,a)
%Wedlands C2 compactly supported radial basis function
%Partial x-derivative
R=sqrt(x.^2+y.^2)/a;
z=(R<1).*(20*R.^3-60*R.^2+60*R-20).*(y./(a^2));
```

Next is a script for computing simple trajectories.

```
%load 1999
%ti=1;

traj=zeros(2,n);
traj(:,1)=[x0;y0];



for i=1:n-1
    k1=dot(squeeze(cf(:,i,1)),phi(traj(1,i)-x,traj(2,i)-y,a));
    j1=dot(squeeze(cf(:,i,2)),phi(traj(1,i)-x,traj(2,i)-y,a));
    k2=.5*(dot(squeeze(cf(:,i,1)),phi(traj(1,i)+.5*dt*k1-x,traj(2,i)+.5*dt*j1-
y,a))+dot(squeeze(cf(:,i+1,1)),phi(traj(1,i)+.5*dt*k1-x,traj(2,i)+.5*dt*j1-y,a)));

    j2=.5*(dot(squeeze(cf(:,i,2)),phi(traj(1,i)+.5*dt*k1-x,traj(2,i)+.5*dt*j1-
y,a))+dot(squeeze(cf(:,i+1,2)),phi(traj(1,i)+.5*dt*k1-x,traj(2,i)+.5*dt*j1-y,a)));

    k3=.5*(dot(squeeze(cf(:,i,1)),phi(traj(1,i)+.5*dt*k2-x,traj(2,i)+.5*dt*j2-
y,a))+dot(squeeze(cf(:,i+1,1)),phi(traj(1,i)+.5*dt*k2-x,traj(2,i)+.5*dt*j2-y,a)));

    j3=.5*(dot(squeeze(cf(:,i,2)),phi(traj(1,i)+.5*dt*k2-x,traj(2,i)+.5*dt*j2-
y,a))+dot(squeeze(cf(:,i+1,2)),phi(traj(1,i)+.5*dt*k2-x,traj(2,i)+.5*dt*j2-y,a)));

    k4=dot(squeeze(cf(:,i+1,1)),phi(traj(1,i)+dt*k3-x,traj(2,i)+dt*j3-y,a));

    j4=dot(squeeze(cf(:,i+1,2)),phi(traj(1,i)+dt*k3-x,traj(2,i)+dt*j3-y,a));

    traj(1,i+1) = traj(1,i) + dt*(k1+2*k2+2*k3+k4)/6;
    traj(2,i+1) = traj(2,i) + dt*(j1+2*j2+2*j3+j4)/6;
end
clear j1 j2 j3 j4 k1 k2 k3 k4
```

Below is a script for computing trajectories in reverse time. Useful for finding where material came from.

```
%load 1999

traj=zeros(2,t);
traj(:,t)=[x0;y0];

%Timestep is 1hr, or 3600s increments
dt=-3600;


for i=t:-1:2
    k1=dot(squeeze(cf(:,i,1)),phi(traj(1,i)-x,traj(2,i)-y,a));
    j1=dot(squeeze(cf(:,i,2)),phi(traj(1,i)-x,traj(2,i)-y,a));
    k2=.5*(dot(squeeze(cf(:,i,1)),phi(traj(1,i)+.5*dt*k1-x,traj(2,i)+.5*dt*j1-
y,a))+dot(squeeze(cf(:,i-1,1)),phi(traj(1,i)+.5*dt*k1-x,traj(2,i)+.5*dt*j1-y,a)));

    j2=.5*(dot(squeeze(cf(:,i,2)),phi(traj(1,i)+.5*dt*k1-x,traj(2,i)+.5*dt*j1-
y,a))+dot(squeeze(cf(:,i-1,2)),phi(traj(1,i)+.5*dt*k1-x,traj(2,i)+.5*dt*j1-y,a)));

    k3=.5*(dot(squeeze(cf(:,i,1)),phi(traj(1,i)+.5*dt*k2-x,traj(2,i)+.5*dt*j2-
y,a))+dot(squeeze(cf(:,i-1,1)),phi(traj(1,i)+.5*dt*k2-x,traj(2,i)+.5*dt*j2-y,a)));

    j3=.5*(dot(squeeze(cf(:,i,2)),phi(traj(1,i)+.5*dt*k2-x,traj(2,i)+.5*dt*j2-
y,a))+dot(squeeze(cf(:,i-1,2)),phi(traj(1,i)+.5*dt*k2-x,traj(2,i)+.5*dt*j2-y,a)));

    k4=dot(squeeze(cf(:,i-1,1)),phi(traj(1,i)+dt*k3-x,traj(2,i)+dt*j3-y,a));

    j4=dot(squeeze(cf(:,i-1,2)),phi(traj(1,i)+dt*k3-x,traj(2,i)+dt*j3-y,a));
    traj(1,i-1) = traj(1,i) + dt*(k1+2*k2+2*k3+k4)/6;
    traj(2,i-1) = traj(2,i) + dt*(j1+2*j2+2*j3+j4)/6;
end
%plot(traj(1,:),traj(2,:),'k')
```

Below is the m-file brasher.m containing the code for my SLM algorithm, this is the work that I am most proud of.

```
%Improved Algorithm for SLM Computation
%Nathan Brasher, US NAVAL ACADEMY




a=5000;
n=2185;
dt=3600;
for k=170:721
    %Update the traces matrix and SLM for this time-slice
    ['Day ',num2str(ceil(k/24)),' ',num2str(mod((k-1),24)),'00 HRS']
    traces=traces(:,2:(2187-k));
    npath=size(traces); npath=npath(1)/2;

    list=[];
    for l=1:npath
        if traces((2*l-1),1) ~=0
            slm(l,1:2,k) = traces((2*l-1):2*l,1);
            slm(l,3,k) = slm(l,3,k-1)-(1/24);
            slm(l,4,k) = slm(l,4,k-1);
            list=[list,l];
        end
    end
    numl=length(list);
    slm(1:numl,:,k)=slm(list,:,k);
    slm((numl+1):20000,:,k)=zeros(20000-numl,4,1);
```

```
    list=[2*list-1;2*list];
    list=reshape(list,1,2*numl);
    traces=traces(list,:);




    %MeshBuilder routine
    %Determine Insertion of New Points
    xmesh=slm(1:numl,1,k);
    ymesh=slm(1:numl,2,k);


    numnew=0;
    for m=1:310296
        R=sqrt((xx(m)-xmesh).^2+(yy(m)-ymesh).^2)/1e3;
        if sum((R<1).*(1-R)) < .1
            xmesh=[xmesh;xx(m)];
            ymesh=[ymesh;yy(m)];
            numnew=numnew+1;
        end
    end


    [num2str(numl),' pts held over']
    [num2str(numnew),' new points']
    [num2str(numl+numnew),' total pts']
    start=numl+1;
    stop=numl+numnew;
    traces=[traces;zeros(2*numnew,2186-k)];




for j=start:stop
rtime=91;
fate=0;

path=zeros(2,n);
path(:,k)=[xmesh(j),ymesh(j)];


for i=k:n-1
    k1=dot(squeeze(cf(:,i,1)),phi(path(1,i)-x,path(2,i)-y,a));
    j1=dot(squeeze(cf(:,i,2)),phi(path(1,i)-x,path(2,i)-y,a));
    k2=.5*(dot(squeeze(cf(:,i,1)),phi(path(1,i)+.5*dt*k1-x,path(2,i)+.5*dt*j1-
y,a))+dot(squeeze(cf(:,i+1,1)),phi(path(1,i)+.5*dt*k1-x,path(2,i)+.5*dt*j1-y,a)));
    j2=.5*(dot(squeeze(cf(:,i,2)),phi(path(1,i)+.5*dt*k1-x,path(2,i)+.5*dt*j1-
y,a))+dot(squeeze(cf(:,i+1,2)),phi(path(1,i)+.5*dt*k1-x,path(2,i)+.5*dt*j1-y,a)));
    k3=.5*(dot(squeeze(cf(:,i,1)),phi(path(1,i)+.5*dt*k2-x,path(2,i)+.5*dt*j2-
y,a))+dot(squeeze(cf(:,i+1,1)),phi(path(1,i)+.5*dt*k2-x,path(2,i)+.5*dt*j2-y,a)));
    j3=.5*(dot(squeeze(cf(:,i,2)),phi(path(1,i)+.5*dt*k2-x,path(2,i)+.5*dt*j2-
y,a))+dot(squeeze(cf(:,i+1,2)),phi(path(1,i)+.5*dt*k2-x,path(2,i)+.5*dt*j2-y,a)));
    k4=dot(squeeze(cf(:,i+1,1)),phi(path(1,i)+dt*k3-x,path(2,i)+dt*j3-y,a));
    j4=dot(squeeze(cf(:,i+1,2)),phi(path(1,i)+dt*k3-x,path(2,i)+dt*j3-y,a));
    path(1,i+1) = path(1,i) + dt*(k1+2*k2+2*k3+k4)/6;
    path(2,i+1) = path(2,i) + dt*(j1+2*j2+2*j3+j4)/6;

    if sum(bnd(path(1,i+1)-coast(:,1),path(2,i+1)-coast(:,2),1e3)) > 0
        fate = 1;
        rtime=(i+1-k)/24;
        break
    elseif sum(bnd(path(1,i+1)-ocean(:,1),path(2,i+1)-ocean(:,2),5e3)) > 0
        fate = 2;
        rtime= (i+1-k)/24;
        break
    end
```

```
end
if mod(j+1-start,100) == 0, [num2str(j+1-start),' of ',num2str(stop+1-start),' computed'],end
slm(j,1,k)=xmesh(j);
slm(j,2,k)=ymesh(j);
slm(j,3,k)=rtime;
slm(j,4,k)=fate;
traces((2*j-1):2*j,:)=path(:,k:n);
path=zeros(2,n);
end
end
clear i j k path k1 k2 k3 k4 j1 j2 j3 j4 list numl numnew start stop npath rtime fate
save 'SLM.mat' slm traces
```

Next comes slmplot.m, the script I used to generate plots of the SLM data.

```
%load plotterdata
%tL=size(slm); tL=tL(3);
hold on
for k=1:721
    cla
    npts=nnz(slm(:,1,k));
    k
    caxis([0 32])
    xmesh=[slm(1:npts,1,k);coast(:,1);ocean(:,1);2.9e5;2.89e5;4.55e5;4.55e5];
    ymesh=[slm(1:npts,2,k);coast(:,2);ocean(:,2);3.92e6;4.42e6;3.9e6;4.41e6];
    z=[slm(1:npts,3,k);zeros(1836,1)];
    zi=griddata(xmesh,ymesh,z,xi,yi);

    pcolor(xi-150,yi-150,zi)
    alpha(alfa);

    shading flat

    axis image
    axis([2.97e5 4.53e5 4.054e6 4.386e6])
    trisurf(ele,x,y,zeros(9700,1)); shading flat
    xlabel('Easting')
    ylabel('Northing')

    title('Synoptic Lagrangian Map')
    if mod(k-1,24) < 10
        text(3e5, 4.38e6,['DAY ',num2str(ceil(k/24)),' 0',num2str(mod(k-1,24)),'00 HRS'])
    else
        text(3e5, 4.38e6,['DAY ',num2str(ceil(k/24)),' ',num2str(mod(k-1,24)),'00 HRS'])
    end
    m(k,:)=getframe();

end
clear k npts tL
```

Next is an algorithm for ISP computation, this was never successful outside of test problems.

```
function isp=find_isp(g)

load test

g0=g;
isp=zeros(2,n);
F=zeros(2,1);
Fold = zeros(2,1);
```

```
J=zeros(2);




for t=1:n
    F(1)=dot(squeeze(cf(:,t,1)),phi(g(1)-x,g(2)-y,a));
    F(2)=dot(squeeze(cf(:,t,2)),phi(g(1)-x,g(2)-y,a));
    J(1,1)=dot(squeeze(cf(:,t,1)),phix(g(1)-x,g(2)-y,a));
    J(1,2)=dot(squeeze(cf(:,t,1)),phiy(g(1)-x,g(2)-y,a));
    J(2,1)=dot(squeeze(cf(:,t,2)),phix(g(1)-x,g(2)-y,a));
    J(2,2)=dot(squeeze(cf(:,t,2)),phiy(g(1)-x,g(2)-y,a));


    while dot(F,F) > 1e-6
        dx = -(J\F);
        g=g+dx;
        Fold = F;
        F(1)=dot(squeeze(cf(:,t,1)),phi(g(1)-x,g(2)-y,a));
        F(2)=dot(squeeze(cf(:,t,2)),phi(g(1)-x,g(2)-y,a));
        J(1,1)=dot(squeeze(cf(:,t,1)),phix(g(1)-x,g(2)-y,a));
        J(1,2)=dot(squeeze(cf(:,t,1)),phiy(g(1)-x,g(2)-y,a));
        J(2,1)=dot(squeeze(cf(:,t,2)),phix(g(1)-x,g(2)-y,a));
        J(2,2)=dot(squeeze(cf(:,t,2)),phiy(g(1)-x,g(2)-y,a));
        if g(1) < 3.6e5 | g(1) > 4.2e5 | g(2) < 4.24e6 | g(2) > 4.38
            break
        end
    end
    isp(:,t) = g;
end
```

Below is the code for my DHT implementation:

```
function dht = find_dht(isp)
%MATLAB Implementation of the Wiggins DHT Algorithm
%Nathan Brasher last updated 25 JAN 2005
cla, hold on
load 1999smooth
tL=(n-1)*dt;
iter = 1;
xn = 100*ones(2,n);
x0 = zeros(2,n);
F=zeros(N,n,2);


A=zeros(2,2,n);

while iter <=2
    iter

    if iter == 1
        x0 = isp;
    else
        x0 = xn;
    end

    x0t=tdiv(x0,dt);
    %Compute A(t)
    for t=1:n
        A(1,1,t)=dot(squeeze(cf(:,t,1)),phix(x0(1,t)-x,x0(2,t)-y,a));
        A(1,2,t)=dot(squeeze(cf(:,t,1)),phiy(x0(1,t)-x,x0(2,t)-y,a));
        A(2,1,t)=dot(squeeze(cf(:,t,2)),phix(x0(1,t)-x,x0(2,t)-y,a));
        A(2,2,t)=dot(squeeze(cf(:,t,2)),phiy(x0(1,t)-x,x0(2,t)-y,a));
    end

    %Compute f(x,t)
```

```
    for t=1:n
        F(:,t,1)=cf(:,t,1)-A(1,1,t).*cfx-A(1,2,t).*cfy+(A(1,1,t)*x0(1,t)+A(1,2,t)*x0(2,t)-
x0t(1,t)).*cfk;
        F(:,t,2)=cf(:,t,2)-A(2,1,t).*cfx-A(2,2,t).*cfy+(A(2,1,t)*x0(1,t)+A(2,2,t)*x0(2,t)-
x0t(2,t)).*cfk;
    end

    %Compute Transformation T to diagonalize A, and diagonal matrix D with eigenvalues d1 and d2
    [T,d1,d2]=find_FSM(A,dt,tL);

    %Check that the trajectory is hyperbolic
    %if d1*d2 > 0
    %    disp('Trajectory not hyperbolic')
    %    break
    %end


    %compute g(w,t)
    for t=1:n
        f(:,t) = T(:,:,t) *[dot(squeeze(F(:,t,1)),phi(x0(1,t)-x,x0(2,t)-
y,a));dot(squeeze(F(:,t,2)),phi(x0(1,t)-x,x0(2,t)-y,a))];
    end

    %Calculate integral to get DHT in w-coordinates, apply T(t) to change to y-coordinates
    if d1 > 0
        for t=1:n
            xn(1,t) = -exp(d1*(t-1)*dt)*integrate(exp(-d1*(0:dt:tL)).*f(1,:),t,n,dt);
            xn(2,t) =  exp(d2*(t-1)*dt)*integrate(exp(-d2*(0:dt:tL)).*f(2,:),1,t,dt);
            xn(:,t) = inv(T(:,:,t))*xn(:,t);
        end
    else
        for t=1:n
            xn(1,t) =  exp(d1*(t-1)*dt)*integrate(exp(-d1*(0:dt:tL)).*f(1,:),1,t,dt);
            xn(2,t) = -exp(d2*(t-1)*dt)*integrate(exp(-d2*(0:dt:tL)).*f(2,:),t,n,dt);
            xn(:,t) = inv(T(:,:,t))*xn(:,t);
        end
    end
    max(max(abs(xn)))
    %Finally change back to original x-coordinates
    xn = xn+x0;
    plot(xn(1,:),xn(2,:));
    %Update iteration no.
    iter = iter + 1;
end

dht = xn;


%Simple First-Order Discrete Differentiation
function out=tdiv(in,dt)
n=length(in);
d=diff(in,1,2);
out=(1/dt)*[in(:,2)-in(:,1),(d(:,1:n-2)+d(:,2:n-1))/2,in(:,n)-in(:,n-1)];

%Solve d/dt X = AX, compute svd, get transformation T all in one fell swoop
function [T,d1,d2]=find_FSM(A,dt,tL)
n=max(size(A));

k=0;
s=eye(2);
X=zeros(2,2,n);
T=zeros(2,2,n);
X(:,:,1) = eye(2);
B(:,:,1) = eye(2);
R(:,:,1) = eye(2);
E(:,:,1) = zeros(2);

%Use 'naive' ,method initially
while log(s(1,1))-log(s(2,2)) < .1
```

```
    k=k+1;
    k1 = A(:,:,k)*X(:,:,k);
    k2 = .5*(A(:,:,k)+A(:,:,k+1))*(X(:,:,k)+(dt/2)*k1);
    k3 = .5*(A(:,:,k)+A(:,:,k+1))*(X(:,:,k)+(dt/2)*k2);
    k4 = A(:,:,k+1)*(X(:,:,k)+dt*k3);
    X(:,:,k+1) = X(:,:,k) + (dt/6)*(k1+2*k2+2*k3+k4);
    [b,s,r]=svd(X(:,:,k+1));
    B(:,:,k+1) = b;
    R(:,:,k+1) = r;
    E(:,:,k+1) = [log(s(1,1)),0;0,log(s(2,2))];
end

[theta,Bt] = findparam(B(:,:,k+1));
[phi,Rt]   = findparam(R(:,:,k+1));
s1 = E(1,1,k+1);
s2 = E(2,2,k+1);

%Use alternative dynamical system to avoid exponential blow-up
for i=k+1:n-1
    Hi = H(A,theta,Bt,i);
    j1 = Hi(1,1);
    k1 = Hi(2,2);
    l1 = .5*(Hi(1,2)-Hi(2,1))+.5*(Hi(1,2)+Hi(2,1))*coth(s2-s1);
    m1 = .5*(Hi(1,2)+Hi(2,1))*csch(s2-s1);
    Hi = H(A,theta+(dt/2)*l1,Bt,i+.5);
    j2 = Hi(1,1);
    k2 = Hi(2,2);
    l2 = .5*(Hi(1,2)-Hi(2,1))+.5*(Hi(1,2)+Hi(2,1))*coth(s2+(dt/2)*k1-s1-(dt/2)*j1);
    m2 = .5*(Hi(1,2)+Hi(2,1))*csch(s2+(dt/2)*k1-s1-(dt/2)*j1);
    Hi = H(A,theta+(dt/2)*l2,Bt,i+.5);
    j3 = Hi(1,1);
    k3 = Hi(2,2);
    l3 = .5*(Hi(1,2)-Hi(2,1))+.5*(Hi(1,2)+Hi(2,1))*coth(s2+(dt/2)*k2-s1-(dt/2)*j2);
    m3 = .5*(Hi(1,2)+Hi(2,1))*csch(s2+(dt/2)*k1-s1-(dt/2)*j2);
    Hi = H(A,theta+dt*l3,Bt,i+1);
    j4 = Hi(1,1);
    k4 = Hi(2,2);
    l4 = .5*(Hi(1,2)-Hi(2,1))+.5*(Hi(1,2)+Hi(2,1))*coth(s2+dt*k3-s1-dt*j3);
    m4 = .5*(Hi(1,2)+Hi(2,1))*csch(s2+dt*k3-s1-dt*j3);
    s1 = s1 + (dt/6)*(j1+2*j2+2*j3+j4);
    s2 = s2 + (dt/6)*(k1+2*k2+2*k3+k4);
    theta = theta + (dt/6)*(l1+2*l2+2*l3+l4);
    phi = phi + (dt/6)*(m1+2*m2+2*m3+m4);
    B(:,:,i+1) = ortho(theta,Bt);
    R(:,:,i+1) = ortho(phi,Rt);
    E(:,:,i+1) = [s1,0;0,s2];
end

d1=s1/tL;
d2=s2/tL;

%Compute final Transformation
for i=1:n
    T(:,:,i) = [exp((i-1)*dt*d1-E(1,1,i)),0;0,exp((i-1)*dt*d2-
E(2,2,i))]*R(:,:,n)'*R(:,:,i)*B(:,:,i)';
end




%Integration using the trapezoid rule
function I = integrate(v,a,b,dt)
I=0;
if b > a
    I=I+.5*dt*(v(a)+v(b));
end
if b>a+1
    I=I+dt*sum(v(a+1:b-1));
end
```

```
function K = H(A,theta,type,t)
if mod(t,1) == 0
    K=ortho(theta,type)'*A(:,:,t)*ortho(theta,type);
else
    K=ortho(theta,type)'*.5*[A(:,:,t-.5)+A(:,:,t+.5)]*ortho(theta,type);
end


function [t,type] = findparam(b)
if b(1,1)*b(2,2) < 0
    type = 2;
else
    type=1;
end
    t=acos(b(1,1));
if b(1,2) < 0
    t=-t;
end

function A = ortho(ang,type)
if type == 1
    A = [cos(ang),sin(ang);-sin(ang),cos(ang)];
else
    A = [cos(ang),sin(ang);sin(ang),-cos(ang)];
end
```

Next is an algorithm for computation of the unstable manifold of the DHT.

```
function usm=find_usm(dht,t)
%Computation of the Unstable Manifold of a
%general dynamical system (aperiodic non-autonomous)
%defined as a data set in 'test.mat'
%Uses Dritschel-Ambaum criteria for point insertion, Dritschel node
%redistribution and cubic spline interpolation
%Nathan Brasher   last updated 29 NOV 2004

%cla
%hold on
load 1999

dt=3600;

ti=1;
delta=1e3;

usm=zeros(2,5,505);


if t < ti
    disp(['Manifold Cannot be Output before t=',num2str(dt*(ti-1))])
    return
end



A=zeros(2);

A(1,1)=dot(squeeze(cf(:,ti,1)),phix(dht(1,ti)-x,dht(2,ti)-y,a));
A(1,2)=dot(squeeze(cf(:,ti,1)),phiy(dht(1,ti)-x,dht(2,ti)-y,a));
A(2,1)=dot(squeeze(cf(:,ti,2)),phix(dht(1,ti)-x,dht(2,ti)-y,a));
A(2,2)=dot(squeeze(cf(:,ti,2)),phiy(dht(1,ti)-x,dht(2,ti)-y,a));

[v,e] = eig(A);
e=diag(e)
if e(1) > 0
    eu = v(:,1);
```

```
else
    eu = v(:,2);
end
manifold = eu*[-1000 -500 0 500 1000];
manifold(1,:) = manifold(1,:) + dht(1,ti);
manifold(2,:) = manifold(2,:) + dht(2,ti);

n=5;
manifold

for i=ti:t-1

    n=length(manifold);
    manifoldn = advance(manifold,n,cf,x,y,dt,i,a);

    d=diff(manifoldn,1,2);

    %Determination of manifold curvature and point density
    K=2*(d(1,1:n-2).*d(2,2:n-1)-d(2,1:n-2).*d(1,2:n-1))./nm([d(1,2:n-1).*nm(d(:,1:n-
2)).^2+d(1,1:n-2).*nm(d(:,2:n-1)).^2;d(2,2:n-1).*nm(d(:,1:n-2)).^2+d(2,1:n-2).*nm(d(:,2:n-
1)).^2]);
    K=abs([K(1),K,K(n-2)]);
    w=nm(d)./(nm(d).^2+4*delta.^2);
    w=[w,0];
    k1 = (w(1:n-1).*K(1:n-1)+w(2:n).*K(2:n))./(w(1:n-1)+w(2:n));
    k2 = .5*(k1(1:n-2)+k1(2:n-1));
    sig = .025*(20*sqrt(k2)+k2).*nm(d(:,2:n-1));
    sig = [sig(1),sig];


    %Dritschel-Ambaum Node Redistribtuion2
    q=sum(sig);
    N=round(q)+2;
    if N > 4
        pp=zeros(N-1,1);
        manifold =zeros(2,N);
        j=1;
        for k=2:N
            p=(1/sig(j))*((q/N).*(k-1)-sum(sig(1:j-1)));
            while p > 1
                j=j+1;
                p=(1/sig(j))*((q/N).*(k-1)-sum(sig(1:j-1)));
            end
            if p > 0
                pp(k-1)=j+p;
            end
        end
        manifold =
[manifoldn(:,1),[spline(1:n,manifoldn(1,1:n),pp),spline(1:n,manifoldn(2,1:n),pp)]',manifoldn(:,n)
];
    else
        manifold = manifoldn;
    end

    if i
        N=length(manifold);
        usm(:,1:N,i+1) = manifold;

    end
    disp([num2str(i),' ',num2str(N),' nodes'])

%    cla
%    plot(manifold(1,:),manifold(2,:),'b-o')
%    plot(dht(1,i+1),dht(2,i+1),'ko')
%    plot(x,y,'.','MarkerSize',1)
%    legend(['Time-',num2str(i+1),'  No of Nodes-',num2str(max(N+1,n))])
%    axis([4e5 4.3e5 4.18e6 4.25e6])
%    m(:,i-ti+1)=getframe;
end
```

```
%save m.mat m
%movie2avi(m,'unstable.avi','FPS',8)

function manifold = advance(manifold,n,cf,x,y,dt,i,alpha)
for j=1:n
    k1=dot(squeeze(cf(:,i,1)),phi(manifold(1,j)-x,manifold(2,j)-y,alpha));
    j1=dot(squeeze(cf(:,i,2)),phi(manifold(1,j)-x,manifold(2,j)-y,alpha));
    k2=.5*(dot(squeeze(cf(:,i,1)),phi(manifold(1,j)+.5*dt*k1-x,manifold(2,j)+.5*dt*j1-
y,alpha))+dot(squeeze(cf(:,i+1,1)),phi(manifold(1,j)+.5*dt*k1-x,manifold(2,j)+.5*dt*j1-
y,alpha)));
    j2=.5*(dot(squeeze(cf(:,i,2)),phi(manifold(1,j)+.5*dt*k1-x,manifold(2,j)+.5*dt*j1-
y,alpha))+dot(squeeze(cf(:,i+1,2)),phi(manifold(1,j)+.5*dt*k1-x,manifold(2,j)+.5*dt*j1-
y,alpha)));
    k3=.5*(dot(squeeze(cf(:,i,1)),phi(manifold(1,j)+.5*dt*k2-x,manifold(2,j)+.5*dt*j2-
y,alpha))+dot(squeeze(cf(:,i+1,1)),phi(manifold(1,j)+.5*dt*k2-x,manifold(2,j)+.5*dt*j2-
y,alpha)));
    j3=.5*(dot(squeeze(cf(:,i,2)),phi(manifold(1,j)+.5*dt*k2-x,manifold(2,j)+.5*dt*j2-
y,alpha))+dot(squeeze(cf(:,i+1,2)),phi(manifold(1,j)+.5*dt*k2-x,manifold(2,j)+.5*dt*j2-
y,alpha)));
    k4=dot(squeeze(cf(:,i+1,1)),phi(manifold(1,j)+dt*k3-x,manifold(2,j)+dt*j3-y,alpha));
    j4=dot(squeeze(cf(:,i+1,2)),phi(manifold(1,j)+dt*k3-x,manifold(2,j)+dt*j3-y,alpha));
    manifold(:,j) = manifold(:,j) + (dt/6)*[k1+2*k2+2*k3+k4;j1+2*j2+2*j3+j4];
end

function k =nm(a)
k=sqrt(a(1,:).^2+a(2,:).^2);
```

Finally, the code necessary for computation of the stable manifold is below.

```
function stm=find_stm(dht,t)
%Computation of the Stable Manifold of a
%general dynamical system (aperiodic non-autonomous)
%defined as a data set in 'test.mat'
%Uses Dritschel-Ambaum criteria for point insertion, Dritschel node
%redistribution and cubic spline interpolation
%Nathan Brasher   last updated 29 NOV 2004

%cla
%hold on
load 1999

stm=zeros(2,5,505);


ti=1;
delta=1e3;
tL=505;

if t < ti
    disp(['Manifold Cannot be Output before t=',num2str(dt*(ti-1))])
    return
end



A=zeros(2);

A(1,1)=dot(squeeze(cf(:,tL,1)),phix(dht(1,tL)-x,dht(2,tL)-y,a));
A(1,2)=dot(squeeze(cf(:,tL,1)),phiy(dht(1,tL)-x,dht(2,tL)-y,a));
A(2,1)=dot(squeeze(cf(:,tL,2)),phix(dht(1,tL)-x,dht(2,tL)-y,a));
A(2,2)=dot(squeeze(cf(:,tL,2)),phiy(dht(1,tL)-x,dht(2,tL)-y,a));

[v,e] = eig(A);
e=diag(e)
if e(1) > 0
    eu = v(:,2);
```

```
else
    eu = v(:,1);
end
manifold = eu*[-1000 -500 0 500 1000];
manifold(1,:) = manifold(1,:) + dht(1,tL);
manifold(2,:) = manifold(2,:) + dht(2,tL);

n=5;


for i=tL:-1:t+1

    n=length(manifold);
    manifoldn = regress(manifold,n,cf,x,y,dt,i,a);

    d=diff(manifoldn,1,2);

    %Determination of manifold curvature and point density
    K=2*(d(1,1:n-2).*d(2,2:n-1)-d(2,1:n-2).*d(1,2:n-1))./nm([d(1,2:n-1).*nm(d(:,1:n-
2)).^2+d(1,1:n-2).*nm(d(:,2:n-1)).^2;d(2,2:n-1).*nm(d(:,1:n-2)).^2+d(2,1:n-2).*nm(d(:,2:n-
1)).^2]);
    K=abs([K(1),K,K(n-2)]);
    w=nm(d)./(nm(d).^2+4*delta.^2);
    w=[w,0];
    k1 = (w(1:n-1).*K(1:n-1)+w(2:n).*K(2:n))./(w(1:n-1)+w(2:n));
    k2 = .5*(k1(1:n-2)+k1(2:n-1));
    sig = .025*(20*sqrt(k2)+k2).*nm(d(:,2:n-1));
    sig = [sig(1),sig];


    %Dritschel-Ambaum Node Redistribtuion
    q=sum(sig);
    N=round(q)+2;
    if N > 4
        pp=zeros(N-1,1);
        manifold =zeros(2,N);
        j=1;
        for k=2:N
            p=(1/sig(j))*((q/N).*(k-1)-sum(sig(1:j-1)));
            while p > 1
                j=j+1;
                p=(1/sig(j))*((q/N).*(k-1)-sum(sig(1:j-1)));
            end
            if p > 0
                pp(k-1)=j+p;
            end
        end
        manifold =
[manifoldn(:,1),[spline(1:n,manifoldn(1,1:n),pp),spline(1:n,manifoldn(2,1:n),pp)]',manifoldn(:,n)
];
    else
        manifold = manifoldn;
    end

    if i
        N=length(manifold);
        usm(:,1:N,i+1) = manifold;

    end
    disp([num2str(i),' ',num2str(N),' nodes'])


%    if mod(i,6) == 0
%    cla
%    plot(manifold(1,:),manifold(2,:),'b-o')
%    plot(dht(1,i+1),dht(2,i+1),'ko')
%    plot(x,y,'r.','MarkerSize',1)
%    legend(['Time-',num2str(i+1),'  No of Nodes-',num2str(max(N+1,n))])
%    axis([3.6e5 4.1e5 4.26e6 4.36e6])
```

```
%    m(:,i-ti+1)=getframe;
%    end
end
%save m.mat m
%movie2avi(m,'dritschel.avi','FPS',8)

function manifold = regress(manifold,n,cf,x,y,dt,i,alpha)
dt=-dt;
for j=1:n
    k1=dot(squeeze(cf(:,i,1)),phi(manifold(1,j)-x,manifold(2,j)-y,alpha));
    j1=dot(squeeze(cf(:,i,2)),phi(manifold(1,j)-x,manifold(2,j)-y,alpha));
    k2=.5*(dot(squeeze(cf(:,i,1)),phi(manifold(1,j)+.5*dt*k1-x,manifold(2,j)+.5*dt*j1-
y,alpha))+dot(squeeze(cf(:,i-1,1)),phi(manifold(1,j)+.5*dt*k1-x,manifold(2,j)+.5*dt*j1-
y,alpha)));
    j2=.5*(dot(squeeze(cf(:,i,2)),phi(manifold(1,j)+.5*dt*k1-x,manifold(2,j)+.5*dt*j1-
y,alpha))+dot(squeeze(cf(:,i-1,2)),phi(manifold(1,j)+.5*dt*k1-x,manifold(2,j)+.5*dt*j1-
y,alpha)));
    k3=.5*(dot(squeeze(cf(:,i,1)),phi(manifold(1,j)+.5*dt*k2-x,manifold(2,j)+.5*dt*j2-
y,alpha))+dot(squeeze(cf(:,i-1,1)),phi(manifold(1,j)+.5*dt*k2-x,manifold(2,j)+.5*dt*j2-
y,alpha)));
    j3=.5*(dot(squeeze(cf(:,i,2)),phi(manifold(1,j)+.5*dt*k2-x,manifold(2,j)+.5*dt*j2-
y,alpha))+dot(squeeze(cf(:,i-1,2)),phi(manifold(1,j)+.5*dt*k2-x,manifold(2,j)+.5*dt*j2-
y,alpha)));
    k4=dot(squeeze(cf(:,i-1,1)),phi(manifold(1,j)+dt*k3-x,manifold(2,j)+dt*j3-y,alpha));
    j4=dot(squeeze(cf(:,i-1,2)),phi(manifold(1,j)+dt*k3-x,manifold(2,j)+dt*j3-y,alpha));
    manifold(:,j) = manifold(:,j) + (dt/6)*[k1+2*k2+2*k3+k4;j1+2*j2+2*j3+j4];
end

function k =nm(a)
k=sqrt(a(1,:).^2+a(2,:).^2);
```