UNIK 4660 - Oppgave 2

Dang Ha The Hien

April 26, 2016

1 Introduction

Since I have implemented some extra things which are not included in the assignment, I think it is a good idea to have a summary table of my code first. Here you will find a short explanation for each function. For the sake of simplicity, some functions' parameters are fixed as constant inside the implementation. HSV colormap is used in most cases, which must be converted to RGB when plotting with imshow(). All of the techniques implemented in this project are resolution-independent.

General Functions:	
demo.m	Demo all functionalities of this project. Please run each demo separately or it can crash your machine
readData.m	Load the two fields stored in HDF5 into Matlab environment.
plotVectorField.m	Plot vector field with small brushstrokes
interpolateVector.m	Bilinear interpolate vector field, return 0 for out of bound.
normalizeField.m	Normalize vector field to unit length, too small vectors become 0
Numerical Integrator:	
forwardEuler.m	Forward Euler algorithm. Options: step_size, max_length
rungeKutta.m	4th order Runge-Kutta. Same above options
adaptiveRungeKutta.m	Adaptive Runge-Kutta. More options: return evenly spaced or not, and minimum arc length of each step.
Plot streamlines:	
plotStreamLinesGrid.m	Seed points taken from a grid
plotStreamLinesPoisson.m	Seed points generated with Poisson Disc
plotStreamLinesProp.m	Seed points distributed by a degree of interest scalar field
plotStreamLinesTopobased.m	Seed points chosen using topology info
Line Integral Convolution	
lineInterConv.m	LIC, can use any numerical integrator. Many algorithm parameters are fixed and must be configured inside the function
Illuminated Stream Lines:	
computeIlluminateTextureMap.m	Compute the illuminated texture map
plotIlluminatedLines.m	Plot illuminated lines with fixed view and light source
animateStreamLine.m	Produce animated illuminated lines
Vortex and Criticals	
calProperty.m	Can calculate Enstrophy, Q_criteria, Vector magnitude, and Topological Degree
classifyCritical.m	Classify critical points using Trace-Determinants plane
Ultility External Functions:	
generate_poisson_2d.m	(downloaded) Generate poisson disc in 2D
bitmapplot.m	(downloaded) Draw lines onto bitmap
bitmaptext.m	(downloaded) Draw texts onto bitmap

2 Exercise 1:

Three numerical integrators are implemented for this project: *forward Euler*, *4th order Runge-Kutta*, and *adaptive Runge-Kutta*[1]. I first test the accuracy and performance of those algorithms by drawing single

[1] fixed TOL = 0.001, rho = 1

stream line on the isabel dataset. The best algorithm (which is the adaptive Runge-Kutta) is then used by default in all further experiments if nothing stated. Most of the function has a function handler as a parameter that you can select what integrator to be used.

2.1 Plotting Single Stream Line

Figure 1 shows the experiment result when drawing a single stream line start at the same seed point $(500, 500)$. The output resolution is $(700, 700)$. Maximum length is fixed at 2000, which allows the stream line to grow fully. Note that I do not check if the streamline repeats itself.

The three algorithms produce almost identical results when step size is small ($h = 0.5$). However, when we increase the step size to $h = 10$, the Euler forward algorithm works poorly when the stream line's curvature is high, which is expected. The RK4 is working correctly in both cases, since the affective step size is not equal to h. However, RK4 is very slow compared to Euler (at least 4 times)

The adaptive RK4 is step size independent, and produces accurate field line with very few sample points and very high performance. In our experiment, the number of sample points in adaptiveRK4 is usually 10-50 times smaller than in Euler method. This excellent property makes adaptive RK4 ideal for many visualization task, such as illuminated stream lines where the complexity of the algorithm depends on the number of sample points. Note that in the LIC algorithm, we need evenly-spaced field lines. Therefore we need to interpolate the field line produced by RK4. I use linear instead of Hermite interpolation as suggested in the fast LIC paper for the sake of simplicity. This hurts the performance and accuracy of the adaptive RK4 algorithm pretty much, as explained later in the next section.

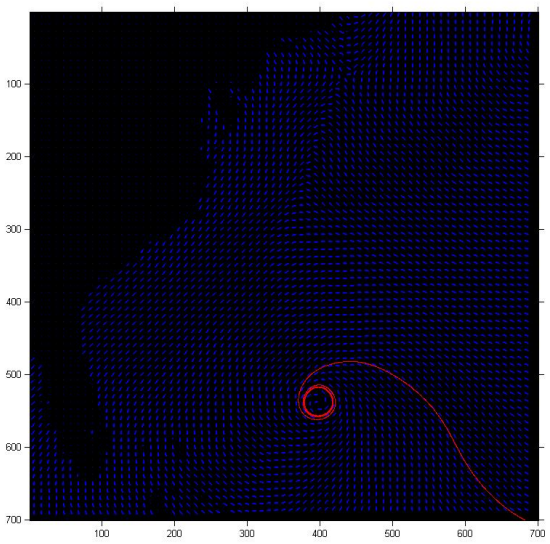
2.2 Plotting Stream Lines Using Different Seed Points Strategy

I implemented 4 different seed points strategy, including *rectangular grid*, *Poisson disc*, *degree of interest*, and *topology-based*.

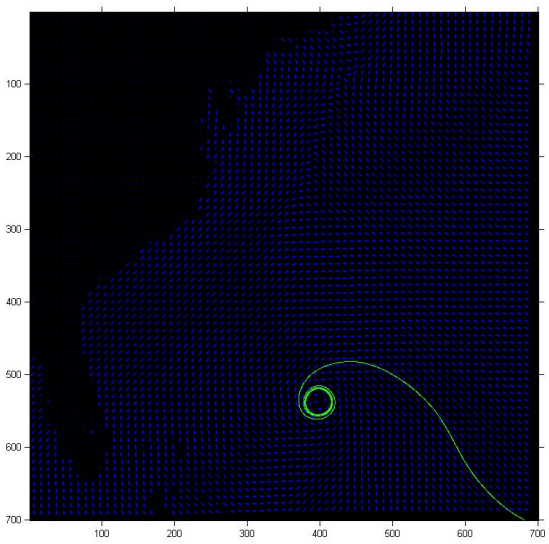
- *Rectangular grid*: seed points are taken from a rectangular grid.
- *Poisson disc*: seed points are generated using Poisson disc distribution, which creates random but even coverage seed points. Note that I do not implement the Poisson disc sampling algorithm myself.
- *Degree of interest*: one can choose any scalar field to be the degree of interest. I used velocity magnitude and enstrophy in this experiment. Monte Carlo sampling process is used to sample seed points from that degree of interest.
- *Topology-based*: since the topological degree algorithm does not guaranteer to only catch critical points (especially in the isabel dataset), I select those critical points manually. This method generates seed points based on the types of critical points.

Since there would be too many figures to plot, I only use adaptive RK4 in this experiment. One can easily change the numerical integrator as demonstrated in the *demo.m* file. Figure 2 shows stream lines of the isabel dataset using different seed point strategy, while 3 shows the same result for the metsim1 dataset. Figure 4 shows isabel's stream lines with different length using Poisson disc method.

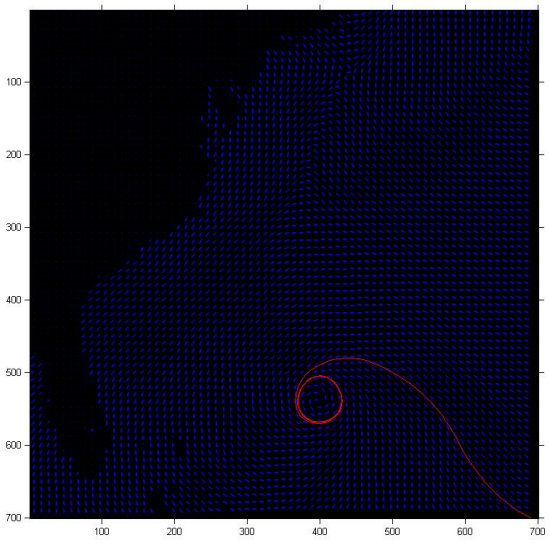
Conclusion: the stream line visualization technique depends heavily on the seed points selection. The stream line density varies very much even with the same seed point density, depending on the topology of the vector field. Therefore, the topology-based technique produces stream lines with nicer density and focuses more on the critical points. I believe that the evenly spaced stream line algorithm can produce an even better visualization, since we can control the stream lines density directly. However, I could not get it done for this project, because there is an implementation trick to quickly check if a new sample point is far enough from all selected sample points, which is hard to integrate into my program flow.



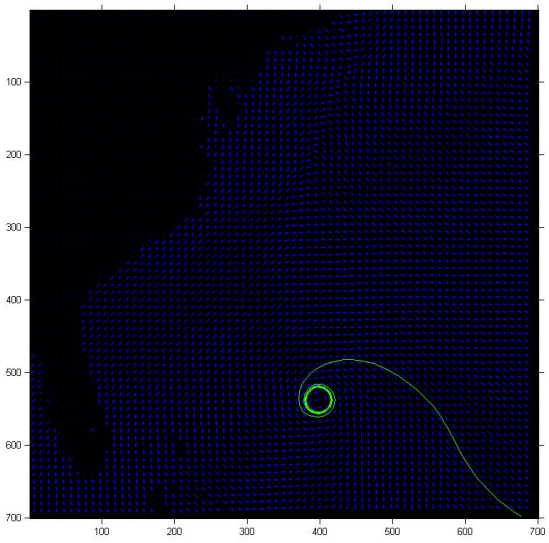
(a) Euler Forward h = 0.5



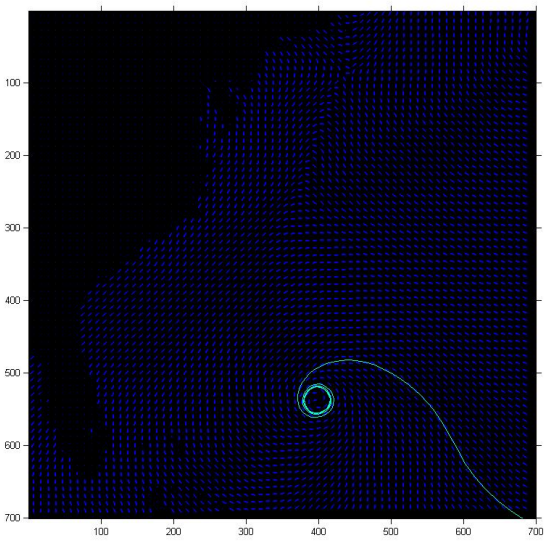
(b) Runge-Kutta h = 0.5



(c) Euler Forward h = 10

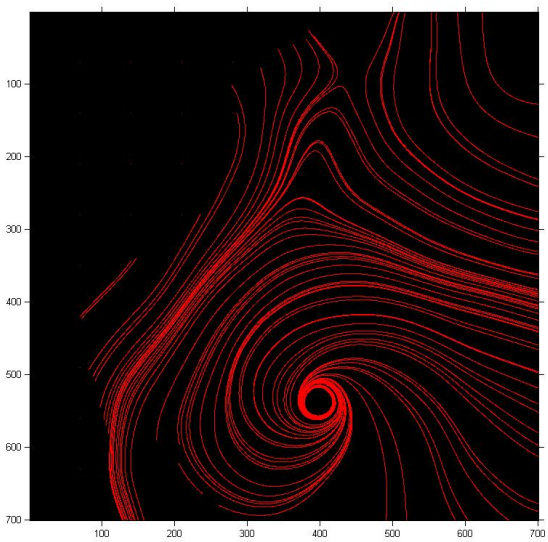


(d) Runge-Kutta h = 10

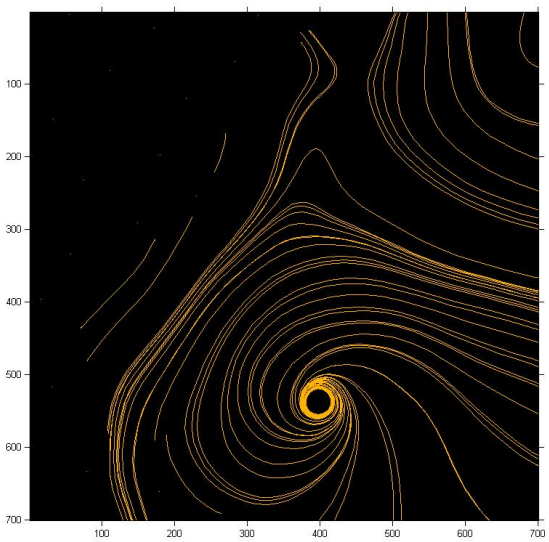


(e) Adaptive Runge-Kutta

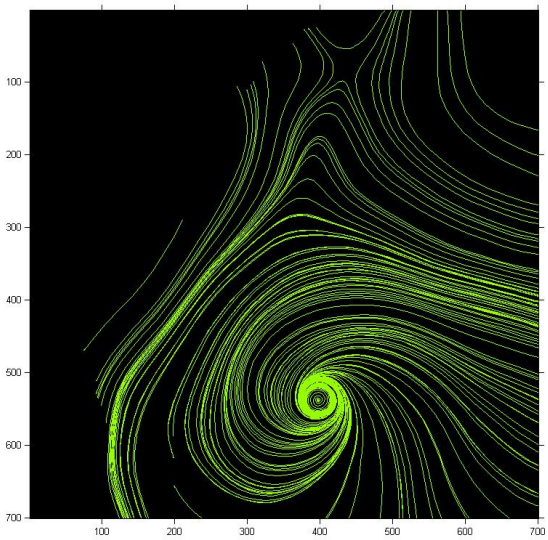
Figure 1: Experiment with three different numerical integrator methods, using different step size.



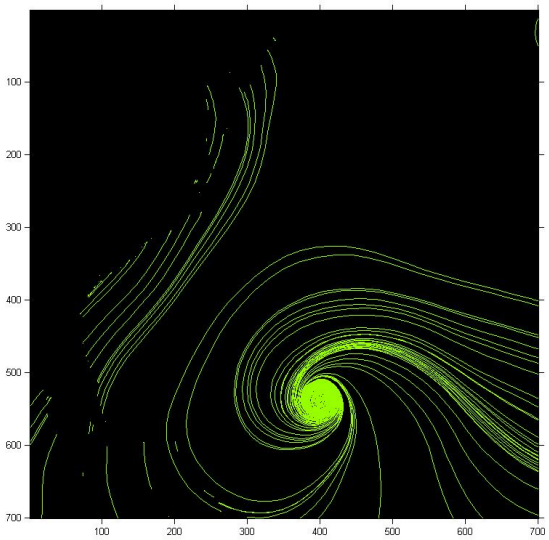
(a) Rectangular grid



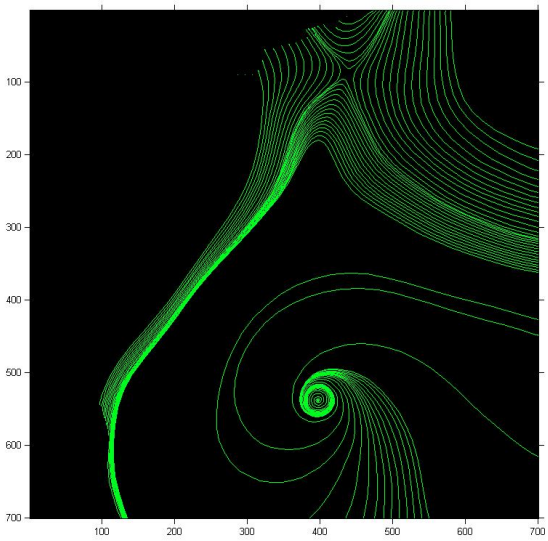
(b) Poisson disc



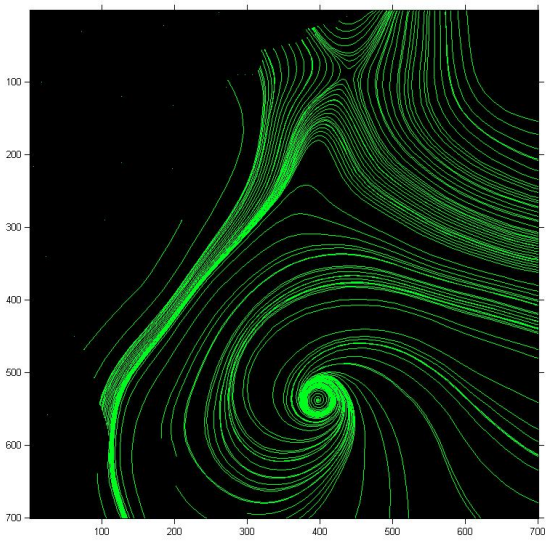
(c) Velocity magnitude as degree of interest



(d) Enstrophy as degree of interest

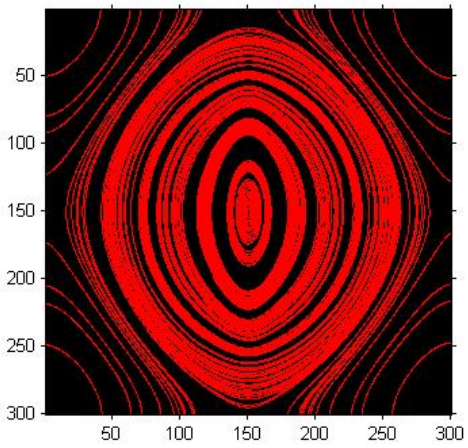


(e) Topology-based

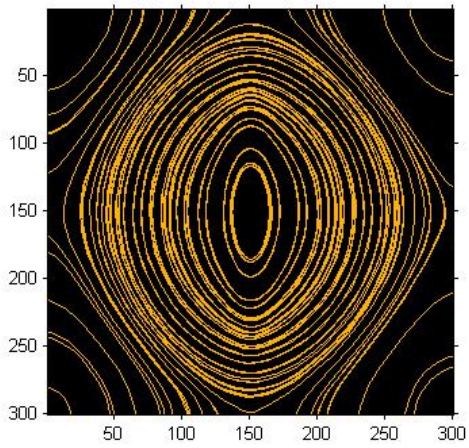


(f) Topology-based together with Poisson disc

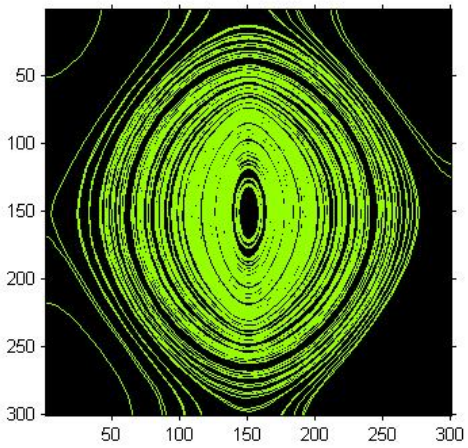
Figure 2: Stream lines of isabel dataset, drawn with different seeds point strategy



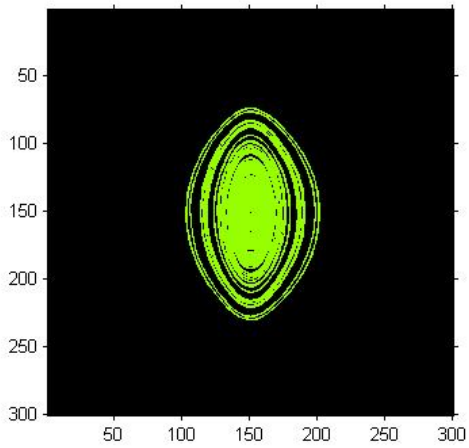
(a) Rectangular grid



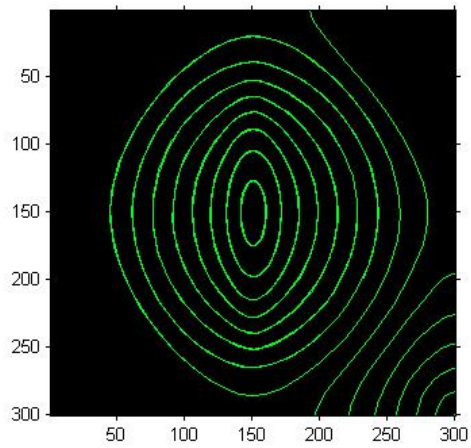
(b) Poisson disc



(c) Velocity magnitude as degree of interest

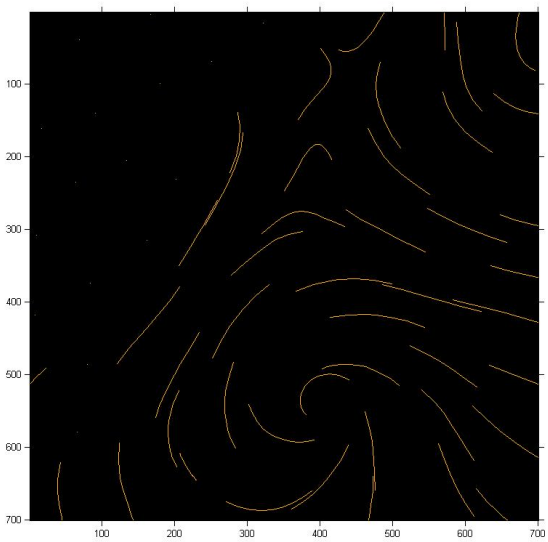


(d) Enstrophy as degree of interest

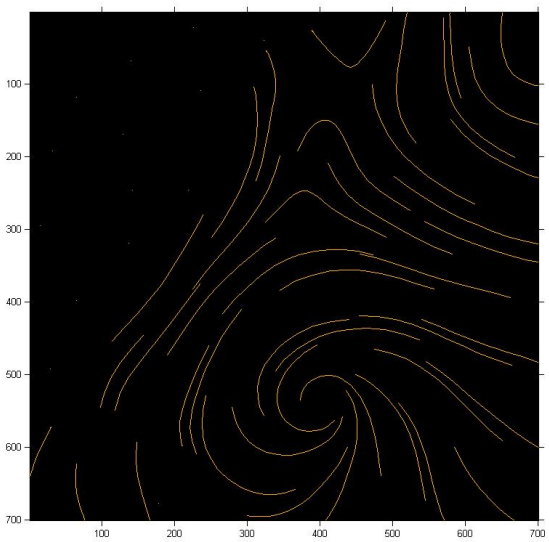


(e) Topology-based

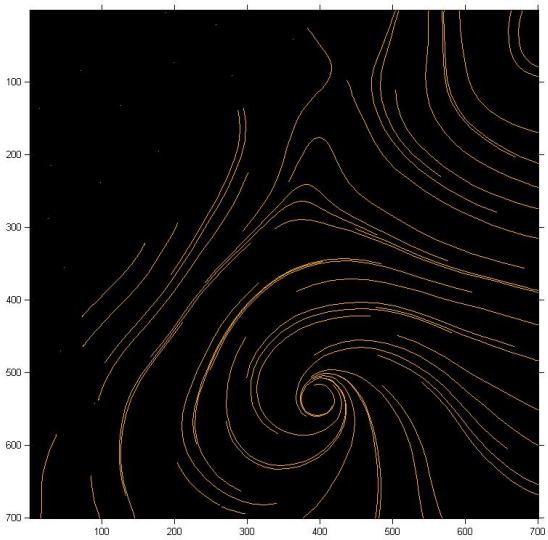
Figure 3: Stream lines of metsim1 dataset, drawn with different seeds point strategy. Note that the images are the transposed versions of the one presented in the assignment. I do not know the exact reason but it could be because the imshow() function of matlab is different



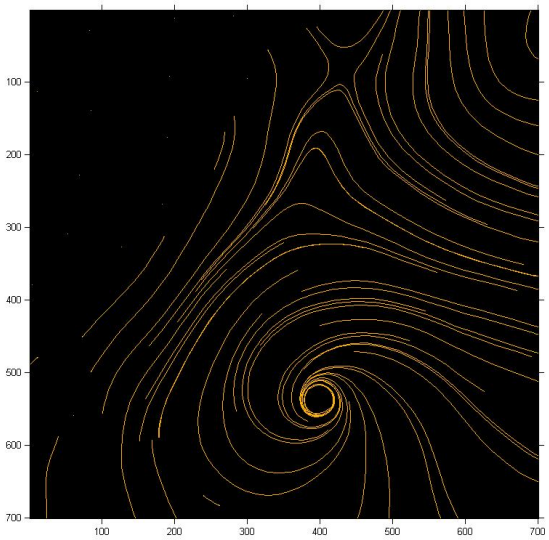
(a) Poisson disc with max length = 50



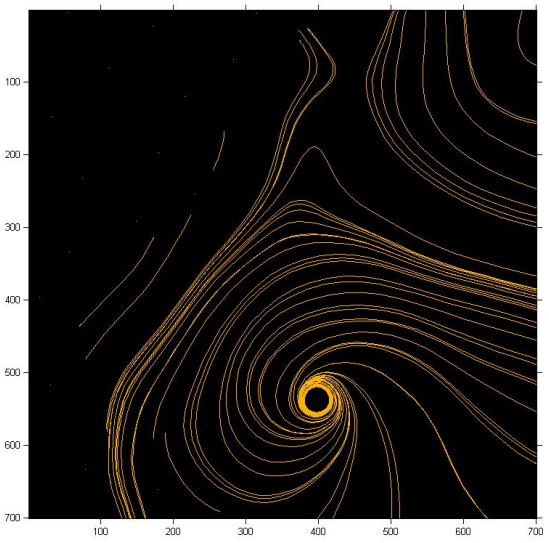
(b) Poisson disc with max length = 100



(c) Poisson disc with max length = 200



(d) Poisson disc with max length = 300



(e) Poisson disc with max length = 2000

Figure 4: Stream lines of isabel dataset using Poisson disc seed point distribution with different max length

LIC Performance	Isabel 700x700	Metsim1 300x300
Euler - L default	111.7s	14s
Euler - L = 15	62.4s	Not done
Adaptive RK4 - default	158.0s	29.2s
Adaptive RK4 - L = 15	86.5s	Not done
RK4 - L default	Too slow!	

Table 1: Table comparing LIC performances

3 Exercise 2 - Line Integration Convolution:

3.1 Introduction

LIC is a texture-based, popular, and powerful algorithm that can be used to visualize vector field. It takes a (usually) random white noise input texture, and perform one-dimensional convolution (i.e. blurring or defusing) at every pixel, along the stream line starting from that pixel in both positive and negative directions. As a result, the image after blurring has the intensity values that strongly correlated along each stream line, and no correlation between neighboring stream lines

The formal convolution process is done in the following way. We are about to compute the intensity for a pixel at x_0. There is a stream line σ going through that point at $x_0 = \sigma(s_0)$. Given a symmetric[2] kernel function $k()$ which is already normalized to unity, the intensity of the pixel x_0 is computed as:

$$I(x_0) = \int_{s_0-L}^{s_0+L} k(s - s_0)T(\sigma(s))ds. \tag{1}$$

with T is the input random texture and $2L$ is the filter length. The original LIC algorithm evaluates the above formula for every pixels, which is very time consuming. In my implementation, I use fast LIC technique to speed up the process.

3.2 My fast LIC implementation

I followed most of the guidelines suggested in the fast LIC paper [Detlev.S, Hans.C.H et. al.]. I first subdivide the image into smaller blocks (25x25 in this case), and run the fast LIC for the first pixel of each block, then the second, and so on. This is done to minimize the total number of stream lines to be evaluated. For each pixel, if that pixel is not hit yet (I set minNumHits = 1), I initiate a stream line starting at the center of that pixel to both direction with the length equal to $L + M$ for each side. With $2L$ is the filter length and set to 1/10th of the image width by default. Varying this parameter yield different result, as shown in the next subsection. M is the maximum steps that we can shift the kernel along the stream line (in both direction) to incrementally compute the intensity of pixels along that stream line. M is fixed at 100 at the beginning, and set to 0 when more than 90% pixels have been hit. I use a constant filter kernel, which allows us to shift the kernel easily. The step size is fixed at $h_t = 0.5$.

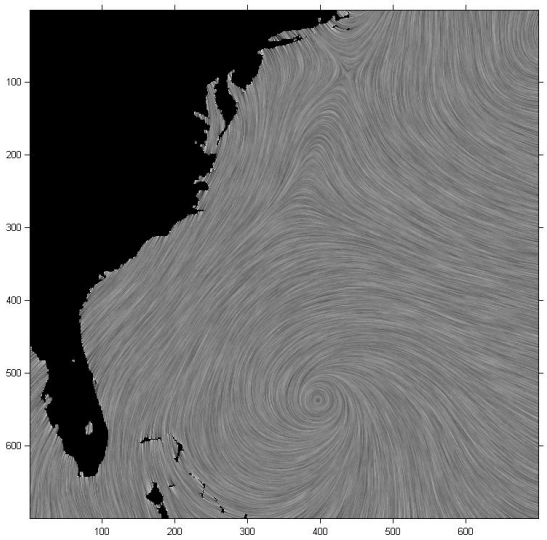
I use adaptive RK4 integrator, which is fast and has error control. However, I use linear interpolation instead of cubic Hermite as suggested in the paper. In fact, I believe this is the only difference in my implementation. This linear interpolation process make adaptive RK4 become slower than forward Euler for this task. The RK4 is simply too slow for this task.

3.3 LIC Results

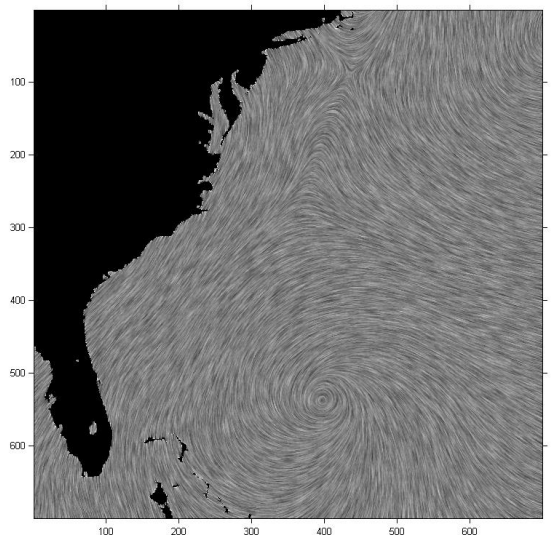
Table 3.3 shows the performance of LIC using Euler forward and adaptive RK4. I did not run the RK4, since it is just too slow. The adaptive RK4 is a bit slower than the Euler - L, without giving any obvious improvement on the result. This is because linear interpolation is used to create evenly spaced sample points, which costs extra computation and introduces extra errors.

Obviously, decreasing filter length L will speed up the algorithm. However, the lower L, the more noisy the image, since the filter length is not big enough to maintain strong correlation along each stream line. This affect is obvious in figure 5

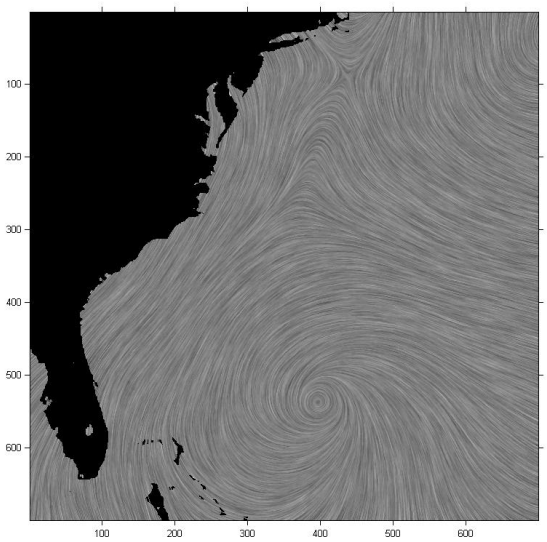
[2]if we use anisotropic convolution kernel instead of a symmetric one, we get the Oriented LIC algorithm



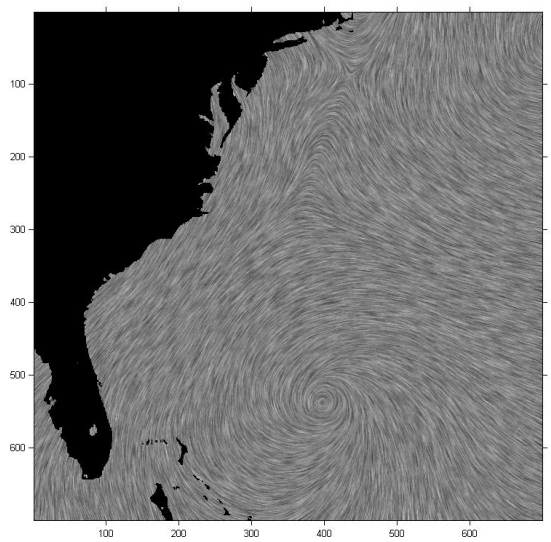
(a) LIC Euler Forward on isabel - L default



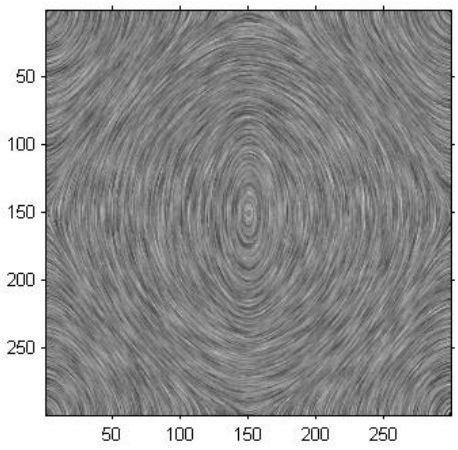
(b) LIC Euler Forward on isabel - L=15



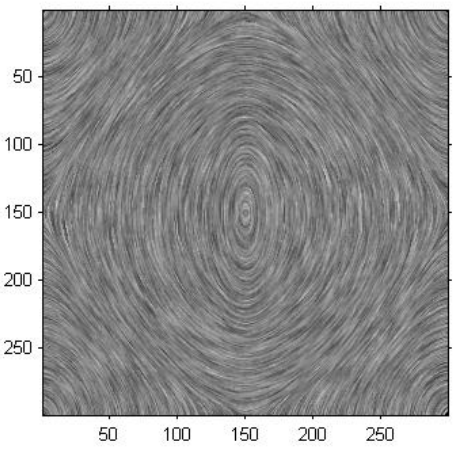
(c) LIC Adaptive RK4 on isabel - L default



(d) LIC Adaptive RK4 on isabel - L=15



(e) LIC Euler Forward on metsim1 - L default



(f) LIC Adaptive RK4 on metsim1 - L default

Figure 5: LIC results using different options. The same input textures are used for each dataset

3.4 Comparing geometric field lines and LIC texture

In my opinion, LIC is the more general technique, which can produce high quality visualization for any vector field using default options, regardless of its topological structure.

On the other hand, the geometric field lines techniques depends heavily on the way the seed points are chosen, which in turn depends on the topological structure of the field lines. In my opinion, the geometric approach is harder to use correctly. However, when using it correctly, it could highlight important features and structures of the field. The geometric approach is also more flexible, which can be changed to focus on different aspects of the field, and can be adapted to different use cases.

4 Extras

Besides the geometric field lines and LIC techniques, I have also implemented several interesting algorithms. In this section, I will give short explanations and results of some experiments.

4.1 Vorticity

The Q criterion and enstrophy are used to identify the vortex. Since the range of enstrophy is very big, plotting the square root of enstrophy produces better visualization. The Q criterion does not have this issue. Figure 6 shows a nice visualization of Q and enstrophy for the isabel dataset. We see that the vortex sheets are visible when using enstrophy, but absent when using Q criterion.

4.2 Topological Degree

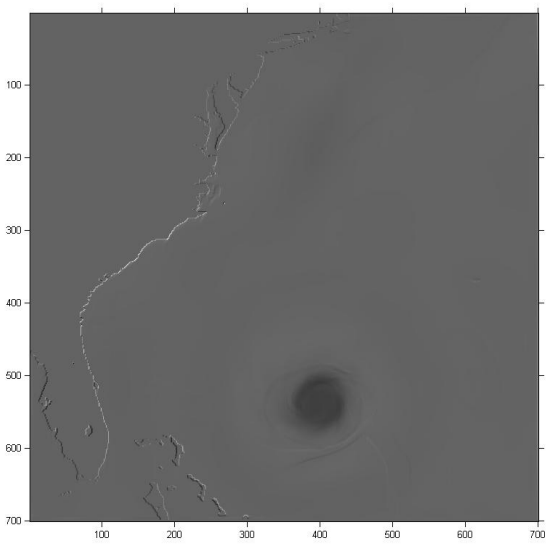
I also calculate the topological degree to identify critical points in the vector field. Unfortunately, the algorithm produce many false detections for the isabel dataset, probably because of the zero velocity area. I also use the trace determinant and delta of the gradient tensor to classify the critical points.

4.3 Illuminated Stream Lines

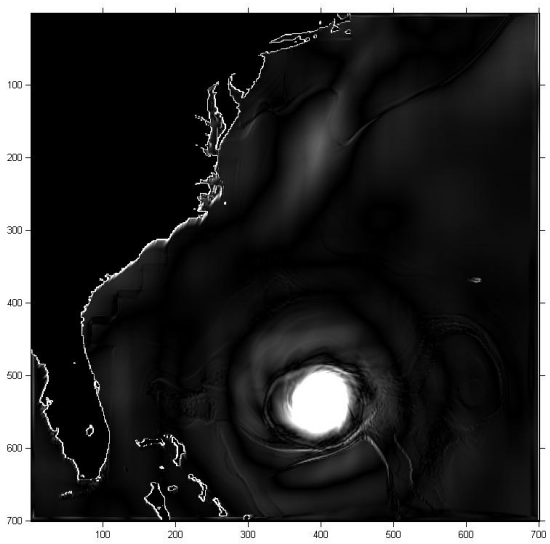
The illuminated stream lines algorithm is also implemented. I assume the stream lines are placed at the plane $z = 0$ and the view point is at the middle and above the field. Moving the light source will lead to different results, as demonstrated in figure 8.

4.4 Animated Stream Lines

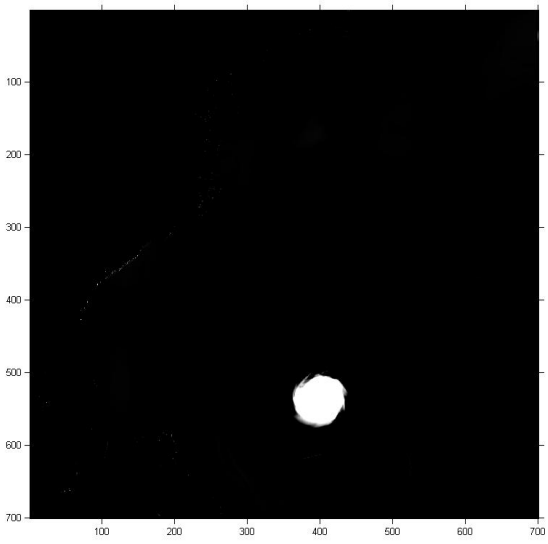
I also implement the animated stream lines, following the suggestion in the illuminated stream line paper. Please check the file 'Animated_ISL.mp4' file in the Reports/Figures folder.



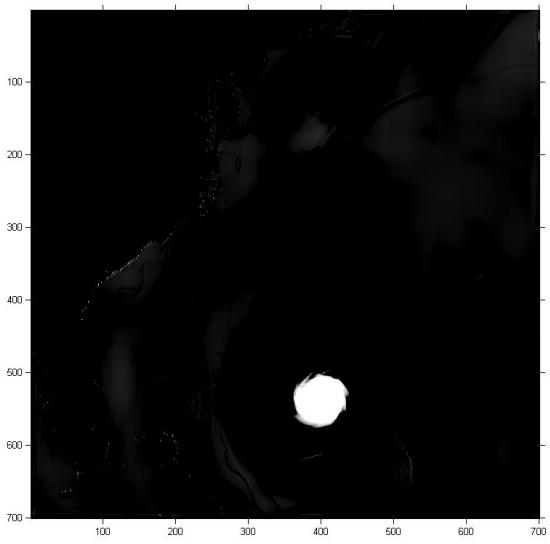
(a) Enstrophy of isabel dataset



(b) Sqrt of enstrophy

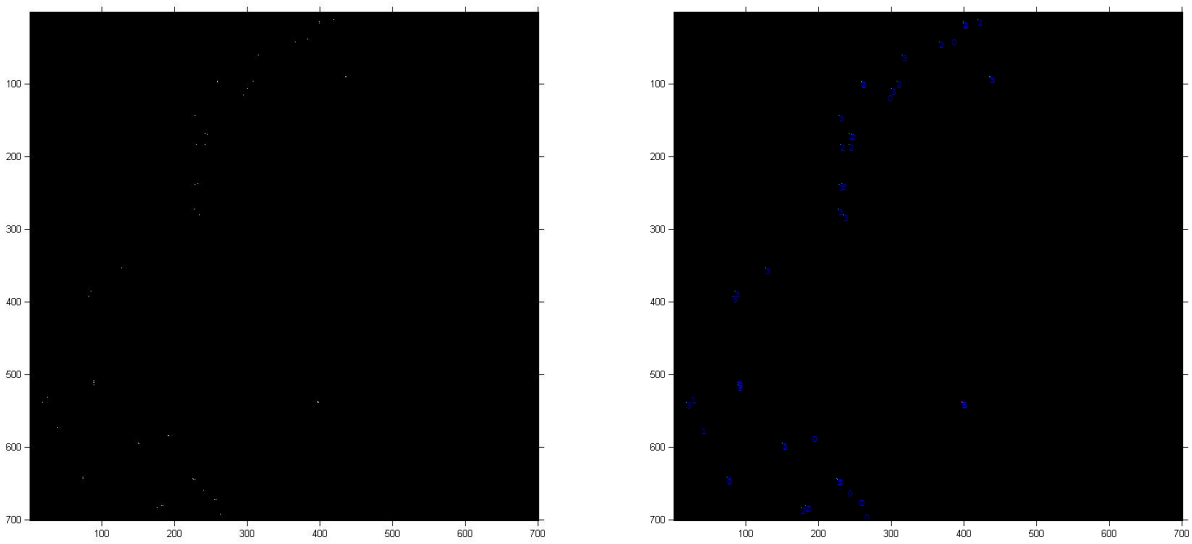


(c) Q criterion of isabel dataset



(d) Sqrt of Q criterion

Figure 6: Vortex identification using enstrophy and Q criterion

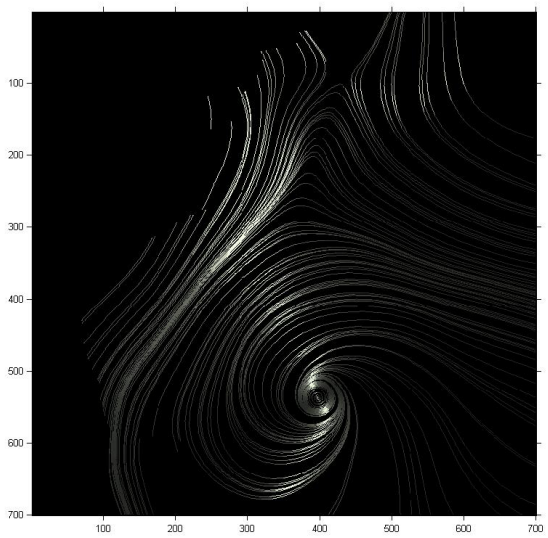


(a) Points with > 0 topological degree are marked

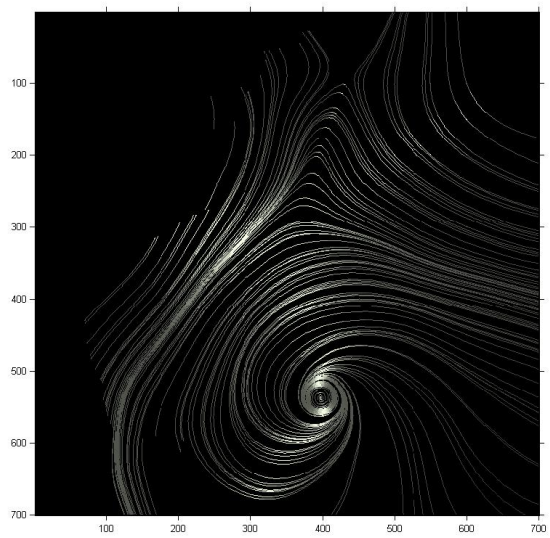
(b) Critical point classification

Figure 7: Topological degree and critical point classification

(a) The t1-t2 texture map



(b) Light source is at $(0, 0)$



(c) Light source is at the view point (head light)

Figure 8: Illuminated Stream Lines