Mandatory assignment TEK9600

Åshild Telle

April 2021

1 Field lines

1.1 Definitions and Background

Field lines are curves that display the direction of a given vector field, always following the tangential direction. They display how an imaginary particle would move in that particular field.

Field lines can be found by simple numerical integration – the field line direction for a point in a given position can be determined from it's velocity direction at that point.

A curve

$$c(s) = [x(s), y(s)]$$

which is truly tangential to the velocity field starting at a given point (x_0, y_0) can be represented by

$$c(s) = [x_0, y_0] + \int_{s_0}^{s} \frac{v(x(t), y(t))}{||v(x(t), y(t))||} dt$$

where $s \in [s_0, S]$ for some s_0, S, determining the length of the curve.

Numerically we can approximate this curve by a number of line segments, represented by a number of points $([x_1, y_1], ..., [x_N, y_N])$. We can find these using a Forward Euler approach [1], given by

$$[x_n, y_n] = [x_{n-1}, y_{n-1}] + hv(x_{n-1}, y_{n-1})$$

where h is some small value, giving an error of order $O(h^2)$. Alternatively, using a Runge-Kutta 4 approach [1], we can define

$$\begin{aligned} k_1 &= v(x_{n-1}, y_{n-1}) \\ k_2 &= v(x_{n-1} + h/2, y_{n-1} + h/2) \\ k_3 &= v(x_{n-1} + h/2, y_{n-1} + h/2) \\ k_4 &= v(x_{n-1} + h, y_{n-1} + h) \end{aligned}$$

then the next point can be represented by

$$[x_n, y_n] = [x_{n-1}, y_{n-1}] + h\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

with a $O(h^5)$ accuracy.

One can integrated backward as well as forward to get a curve that *passes* through a given point, rather than starts at it.

1.2 Methods

I've implemented three suggested methods for seeding strategies – uniform, random and density based. The first two were straight forward to implement, giving a regular or random distribution across the domain. The last one required some more work and is also considerably slower the first two. For all of them I assumed that we wanted to work with normalized data – displaying the direction of the flow only, not the magnitude.

The implementations below are done for a data set with assumed dimensions $(X, Y, 2)$. For simplicity the coordinates of each point are assumed to be integers, such that if we have $X \times Y$ data points, we assume that the x axis goes from 0 to X, and the y axis from 0 to Y. Furthermore, the vector field is represented in a semi-continuous way using interpoltion with splines, such that points accessed outside of integer points are given reasonable values.

To avoid collision/overlap between the different curves, I subdivided the domain into a Cartesian mesh with a fairly fine resoltion. For every field line calculated, a mapping from the coordinates of each point to this mesh was calculated and cells in the mesh overlapping with the line was marked as "visited". For all subsequential field lines, the calculation of these were stopped if it reached a cell that had already been visited. Note that the boolean mesh was only updated after calculating a streamline, meaning that the same field line could continue even if it came really close to or even crossed itself.

All code is implemented in Python.

1.2.1 Uniform distribution

Given a desired number of seeding points N, and a data set with dimensions X and Y, one can define

$$N_x = \left\lfloor \sqrt{(}N \cdot X/Y) \right\rfloor$$

and

$$N_y = \lfloor N/N_x \rfloor$$

and then N_x and N_y points in each direction. If N is a square number and $X = Y$, then $N_x = N_y$ and $N_x \cdot N_y = N$; if not, this should ensure proper proportion such that our seeding points are placed in a pattenr with a regular distance between each, in both directions.

1.2.2 Random distribution

Given a desired number of seeding points N, and a d, we simply define N random numbers between 0 and X and N random numbers between 0 and Y, giving us a set of N coordinates used as seeding points.

1.2.3 Density based distribution

My implementation here is based on similar ideas as presented in [2]: I'm subdividing the domain using a regular grid – sort of in a similar manner as the uniform approach, but with many more potential seeding points. Here I also discarded all points in the part of the domain where the velocity was zero.

I then calculate the distance to all field lines previously calculated (starting with 0), and choose the coordinate point which is furthest away from all field lines calculated up to this point. Here we only considered points from the initially uniformly generated list, which makes the list of points to iterate through feasible to work with. Using the coordinates of this point as the new seeding point, we calculate a new field line and updated the distance to the other feasible points iteratively.

1.2.4 Numerical experiments

For each of the two datasets, *Isabel* and *Metsim*, I compared different seeding strategies, different field line lenghts as well as different number of seeding points. I chose the length of the field line to be inversely proportional to the number of seeding points used. For N seeding points, the corresponding length was chosen as

$$L = 10 \cdot \frac{X}{N}$$

allowing for shorter lines, attempting to keep the density somewhat consistent. For all of these experiments I used a step length $h = 0.1$ and the Runge-Kutta 4 scheme.

For the Isabel data set, which seemed to have the most interesting curvature features, I also compared the result of a single seeding point placed in the lower right corner for the Forward Euler as well as the Runge-Kutta 4 implementation, for different choices of step length h. Here I used a set length of 1000.

1.3 Results

The distributions of seeding points for the Isabel data are displayed in Figure 1, and the corresponding field lines in Figure 2. Similarly the seeding points and lines for the Metsim data are displayed in Figure 3 and 4.

For the Isabel data, we see that all seeding strategies and all number of field lines capture where the spiral is. The saddle point, however, seems somewhat harder to capture. Considering the cases $N = 10$ and $N = 50$, the random strategy – for this simulation – seems to perform worse than the two others, which are somewhat similar. That is, of course, random and might not be similar for the next time we run the algorithm. For higher values of N all of them seem to capture the field fairly good. The field lines for the random strategy are somewhat more cluttered than the others. The density based algorithm seem to follow the curves better than the regular one; it flows a bit better, giving a higher sense of continuity. For the seeding points, as displayed in Figure 1, the uniform and random are distributed as expected and do not feature any special patterns. However for the density plot one can detect weak patterns going perpendicular to the flow.

For the Metsim data, the behaviour in the middle as well as the corners seems hardest to capture. The density based algorithm performs best here, for $N = 50$ and $N = 100$; however, for some reason, as we move to $N = 500$ and $N = 1000$ the space in the middle does not get covered (I'm not sure why). Apart from that, all of them perform reasonably well for all values of N except form $N = 10$. Here we get a fairly uneven distribution of the field lines. For the seeding point distributions, again the uniform and random ones are predetermined without any surprises, while we can see some perpendicular patterns for the density based one.

The comparison of the Forward Euler and Runge-Kutta 4 implemenations is displayed in Figure 5. For different h values, we see that if h is chosen to be larger, there is a tendency for the field line to be "caught" in a circle instead of moving towards the middle of the spiral, which is likely to be the analytic "true" behaviour in this case. For each of the h values we're starting with moving towards the middle of the circle, then at some point – closer to the middle the lower h we have – it reaches a point where the field line from then on moves in a circle. The Runge-Kutta 4 method in overall performs better than the Forward Euler one, and it's only the smallest h value combined with the Runge-Kutta 4 scheme that is able to fully capture the spiral movement all the way to the middle (on the scales plotted).

1.4 Discussion

Working with visual interpretation, it's not always easy to compare two representations; there is a certain degree of subjectivity in the jugdement. However, one can fairly easily assess whether the

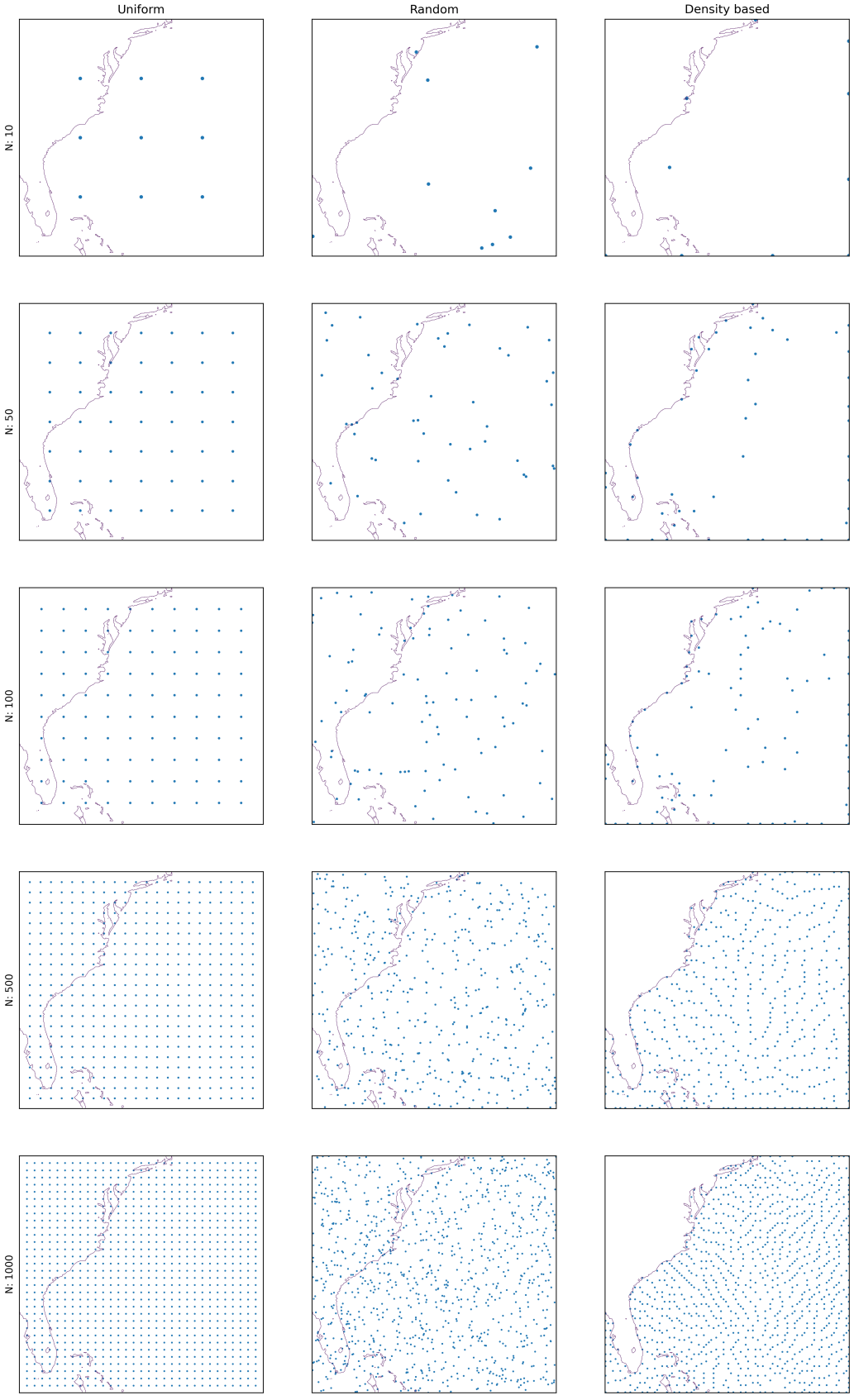

Figure 1: Distribution of seeding points used to generate field lines for each seeding strategy, for different lengths, for the Isabel data set

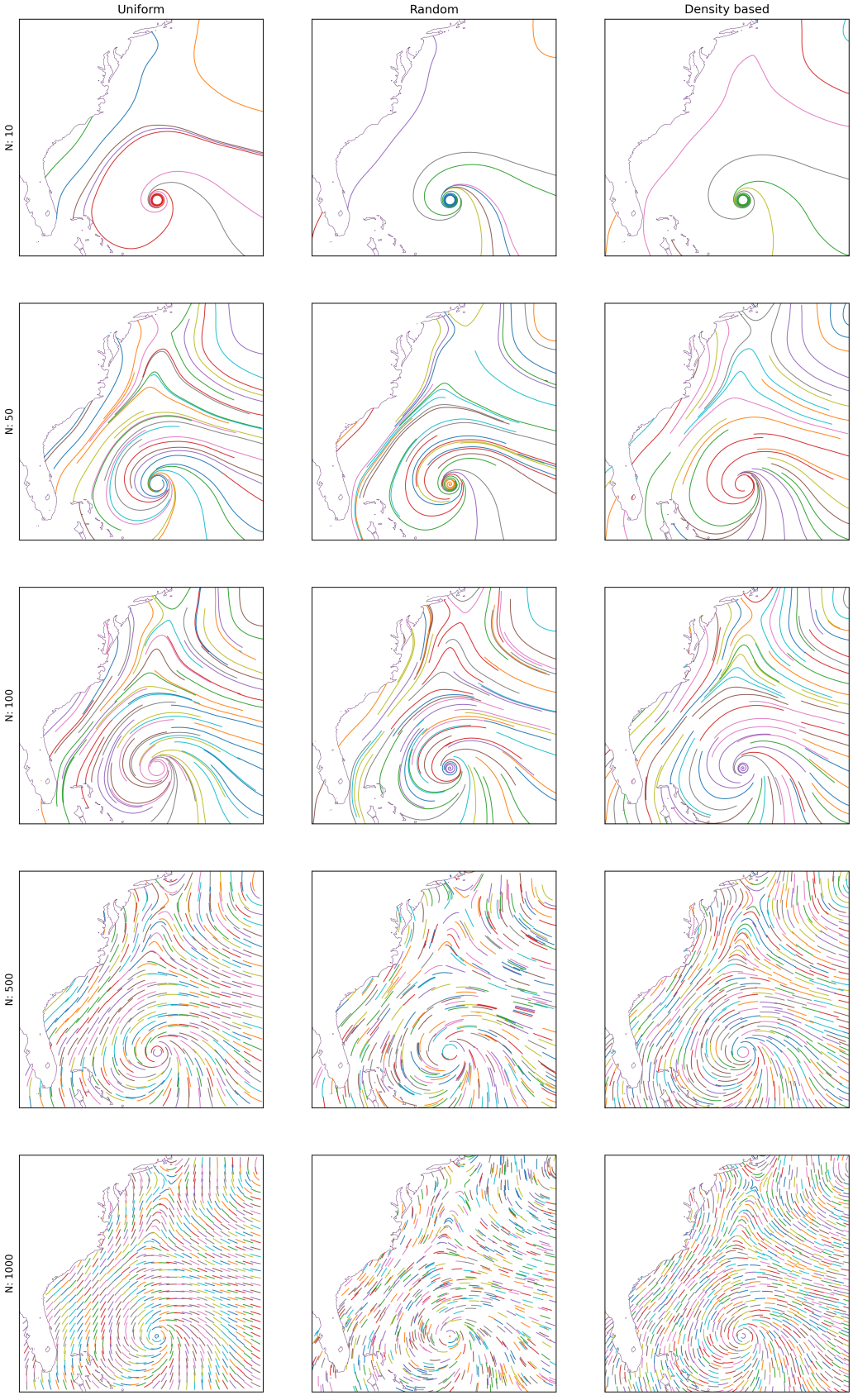

Figure 2: Field lines calculated using different seeding strategies, lengths and number of field lines, for the Isabel data set.

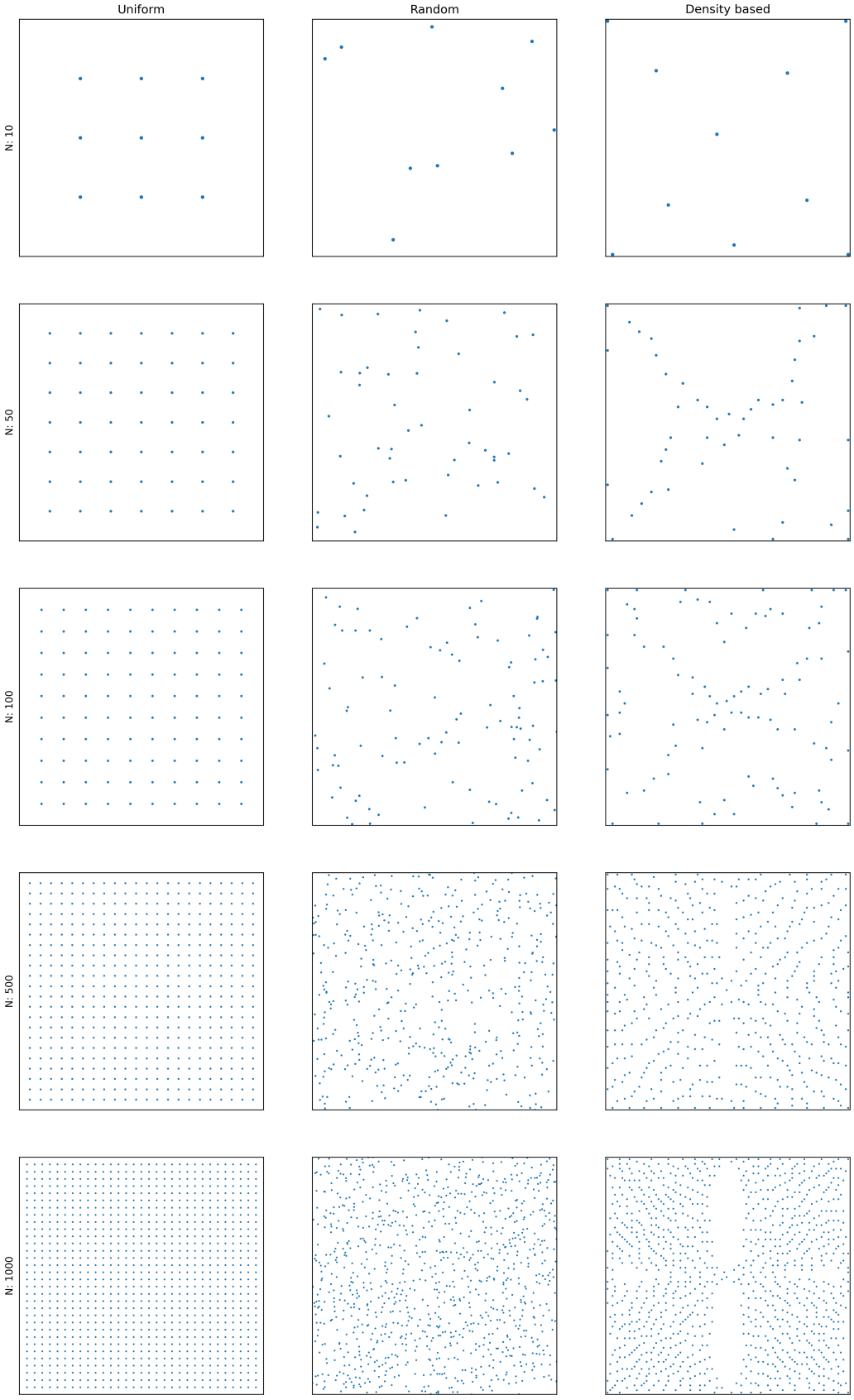

Figure 3: Distribution of seeding points used to generate field lines for each seeding strategy, for different lengths, for the Metsim data set.

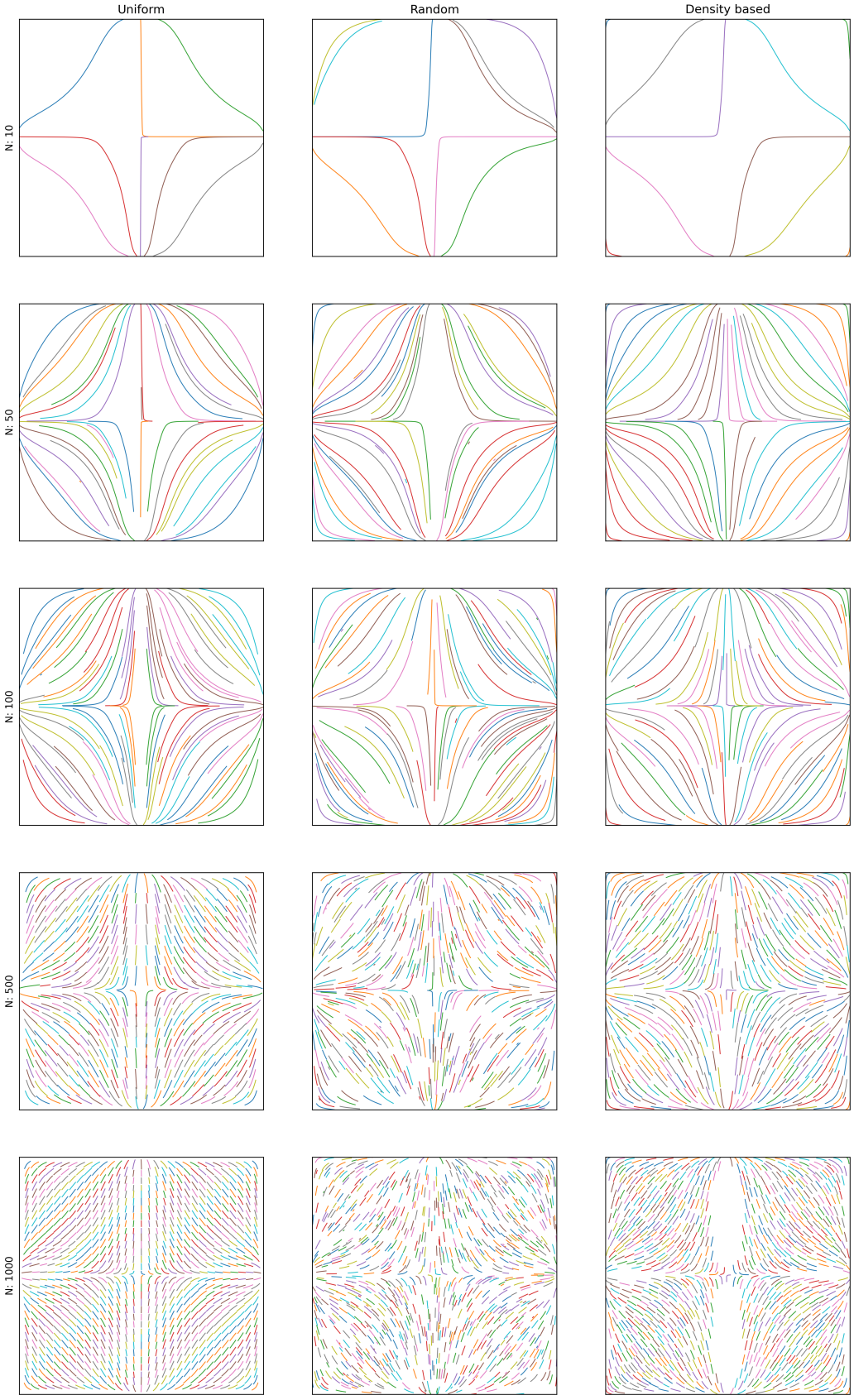

Figure 4: Distribution of seeding points used to generate field lines for each seeding strategy, for different lengths, for the Metsim data set.

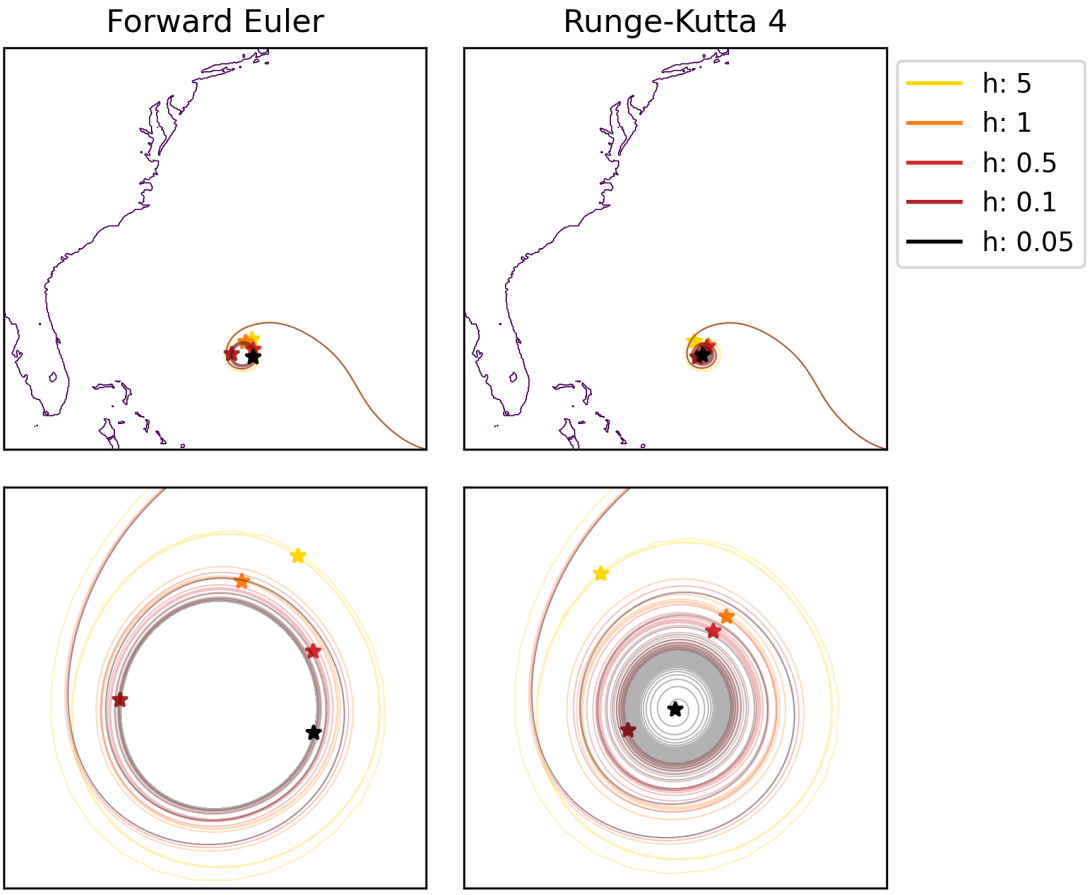

Figure 5: Comparison of Forward Euler and Runge-Kutta 4 implementations, for different choices of h. The two upper panels display the whole picture, the two lower ones a zoomed in verison focusing on the spiral. The stars highlight where the field lines stop.

behaviour around critical points have been covered. Furthermore, in a more subjective fashion, simplicity should trumph density; if one can display the most interesting features with fewer lines moving towards a denser representation doesn't really add anything. This might, however, depend on a numbre of factors, including what your field lookl ike, what you try to investigate, etc.

From the results presented above, all N values above 10 are able to outline the main behaviour of the whole vector field, including indicationg the location of critical points, for both data sets. N values 50 and 100 are fairly similar in apparence, both featuring long strokes. The density seem fairly similar. Moving towards N values of 500 and 1000 we get a much denser representation, displaying more details while the global understanding of how the field behaves remain fairly unaltered. For the isabel data set the density based algorithm by far seem to outperform the others with respect to displaying critical points as well as giving the most even representation in terms of where the field lines are.

With respect to the two different shcemes – Forward Euler and Runge Kutta 4 – I found, not surprisingly, smaller h values to give better precision, and the Runge-Kutta 4 scheme to perform better than the Forward Euler one. This was especially clear focusing on the part of the domain where we have the spiral. Whether or not it is important to capture the behaviour in all parts probably depends on the nature of the data set and the points of interest, which again depends on the research question being investigated. All of the chosen h values are able to display that there probably is a spiral in that region in the picture (but one can't be completely sure); but only the finest one, combined with the Runge-Kutta 4 scheme, is able to fully capture the spiral movement to the very centre.

Event though the behaviour of the middle of the spiral requires pretty high resolution, the field lines capture the behaviour close to correctly in most of the domain; the tradeoff is only large in the parts of the domain with high curvature. A way of balancing correctness and feasibility could be to use a variable h, lowering the steps with larger angular motion.

2 Line Integral Convolution

2.1 Definitions and Background

Line integral convolution (LIC) is a visualization technique for vector fields, defined by a convolution of an image and a vector field. One starts with an image, which often is just randomly generated, and then use the field to compute a new image where each pixel represent an average along the direction of the field.

Let T be an initial texture image of dimensions X_T, Y_T, and v a vector field. If $\mathbf{r_0}$ is a point in the image, one can compute a field line σ passing through $\mathbf{r_0}$ tangential to v. Then a value $I(\mathbf{r_0})$ can be computed by integrating of length L along the field line, in both directions:

$$I(r_0) = \int_{-L/2}^{L/2} k(s - r_0) T(c(s)) ds$$

where L is the length of the curve and k the filter kernel.

There are several possible approaches to discretizise this. A fairly simple straight-forward would be to, for each pixel $r_0 = (x, y)$, where $x \in [0, T_x - 1]$ and $y \in [0, T_y - 1]$, compute (for L an even number)

$$\begin{aligned} I(r_0) &= \sum_{n=-L}^{L} h \cdot T(x_n, y_n) \\ &= \frac{1}{2L+1} \sum_{n=-L}^{L} \cdot T(x_n, y_n) \end{aligned}$$

Length / (X_T, Y_T)	25	50	100
(250, 250)	10	21	41
(500, 500)	42	88	164
(1000, 1000)	170	368	667

Table 1: Time used to generate the LIC image for the Isabel data, in seconds

Length / (X_T, Y_T)	25	50	100
(250, 250)	5	11	21
(500, 500)	21	44	87
(1000, 1000)	88	179	358

Table 2: Time used to generate the LIC image for the Metsim data, in seconds

which simply averages all pixel values along the streamline, using the pixel values associated with each coordinate.

2.2 Methods

In my implementation I used the average as presented above. I used a vectorized version of the methods calculated for Exercise 1, which – using Python – is a lot faster. For cases close to the boundary, i.e. where the streamlines continued outside of the domain I simply did not include these in the average. For all calculations I used a filter to detect zero values – i.e. for the Isabel data, the land – which do not get averaged out otherwise.

2.2.1 Numerical experiments

For both data sets, the Isabel and the Metsim data, I tried out different length and different resolutions. The Isabel data set has original dimensions 500×500, while the Metsim data set has original dimensions 127×127. For each data set I tried with resoltions 250×250, 500×500 and 1000×1000. I also tried with field line lengths 25×25, 50×50 and 100×100. For all of these I used a step length $h = 1$, and the Runge-Kutta 4 scheme.

For the Isabel data set I also investigated the two different schemes, Forward Euler and Runge-Kutta 4. I tried with a few different time step h values. For this experiment, I used a resolution of 500×500 and field line length of 25.

2.3 Results

The LIC representation is displayed in Figure 6 and 7. With longer field lines we see that there is a more continuous pattern, where one can follow the field lines continuously from the edge. However longer field lines, as well as higher resolutions, seem to give a more blurred image which makes it somewhat harder to make out the features of the vector field.

The time used to calculate the data used to plot the different representations are listed in Table 1 and 2. The time seem to increase in a quadratic manner with respect to both resolution and length, but all within fairly reasonable time range. I didn't find time to be a limiting factor, but memory – decreasing step size h further easily made the program crash.

For the Forward Euler / Runge-Kutta 4 / different step lenght experiments, we see that for $h = 5$ there is a weak tendency in the larger picture (left) to display the features of the vector field; however as we zoom in (right) the resolution is hopeless. With $h = 1$ and $h = 0.5$ we have fairly similar behaviour. The Runge-Kutta 4 captures the middle of the spiral somewhat better.

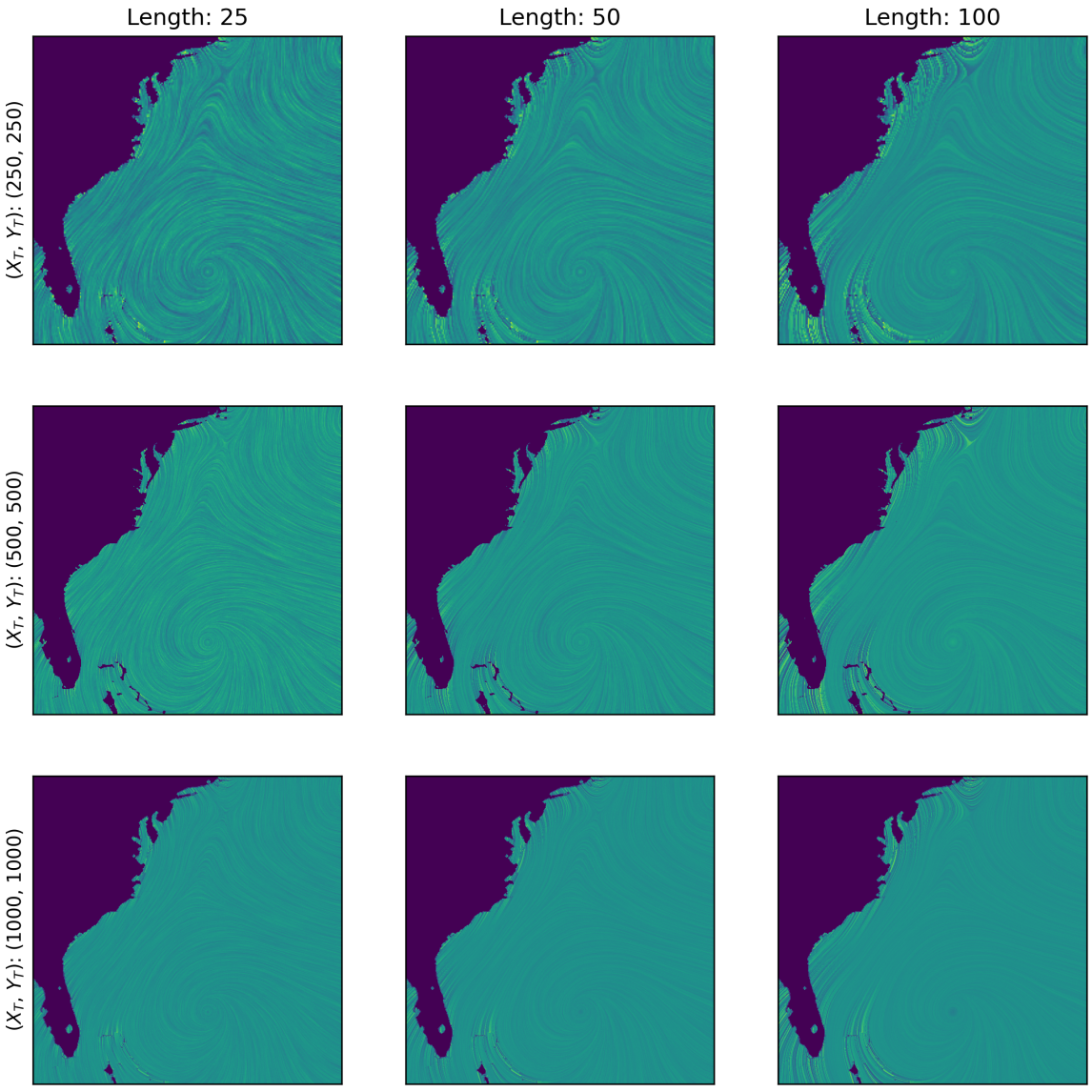

Figure 6: Visualization of the Isabel data using LIC, for different lenghts of the field lines and different resolutions of the initial texture image.

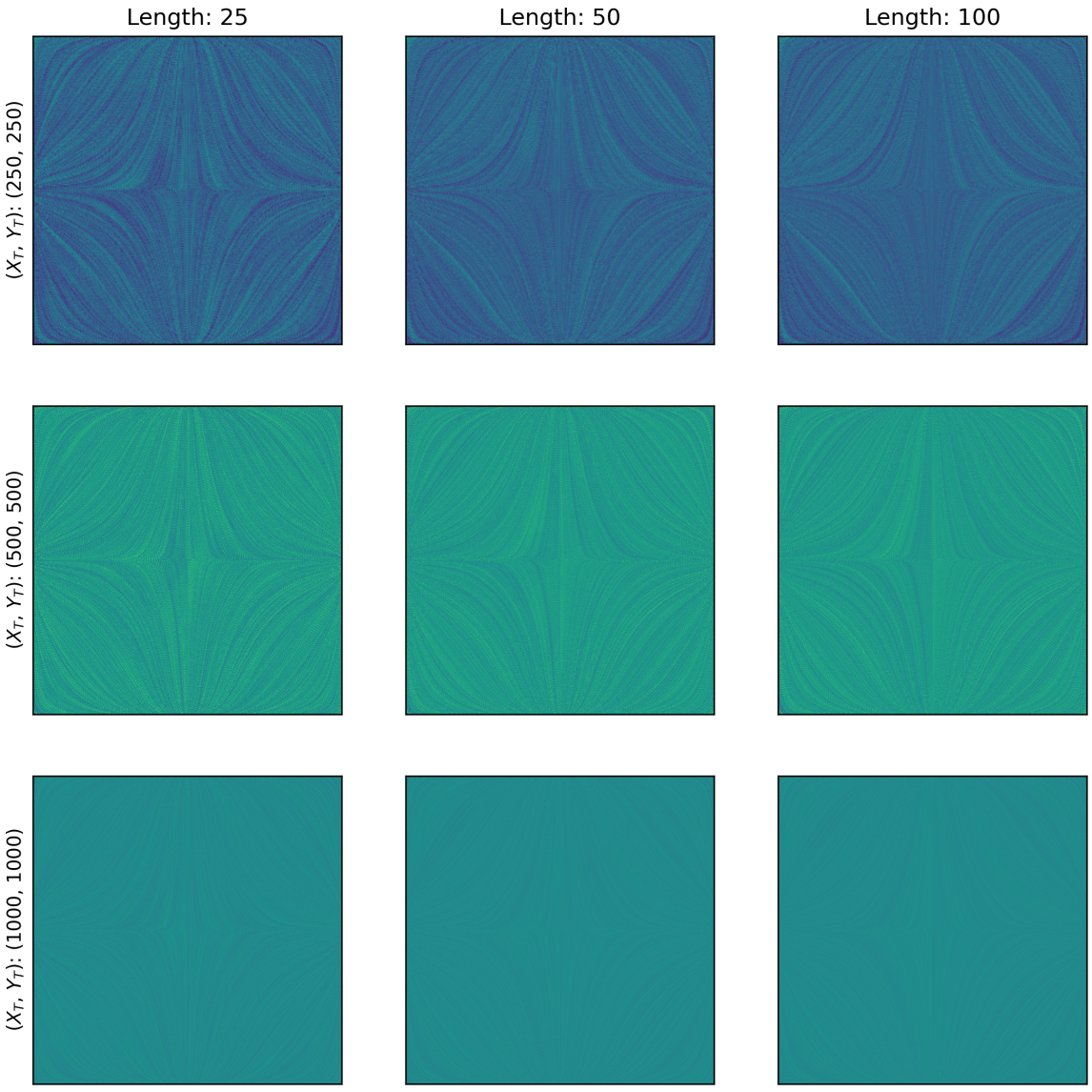

Figure 7: Visualization of the Metsim data using LIC, for different lenghts of the field lines and different resolutions of the initial texture image.

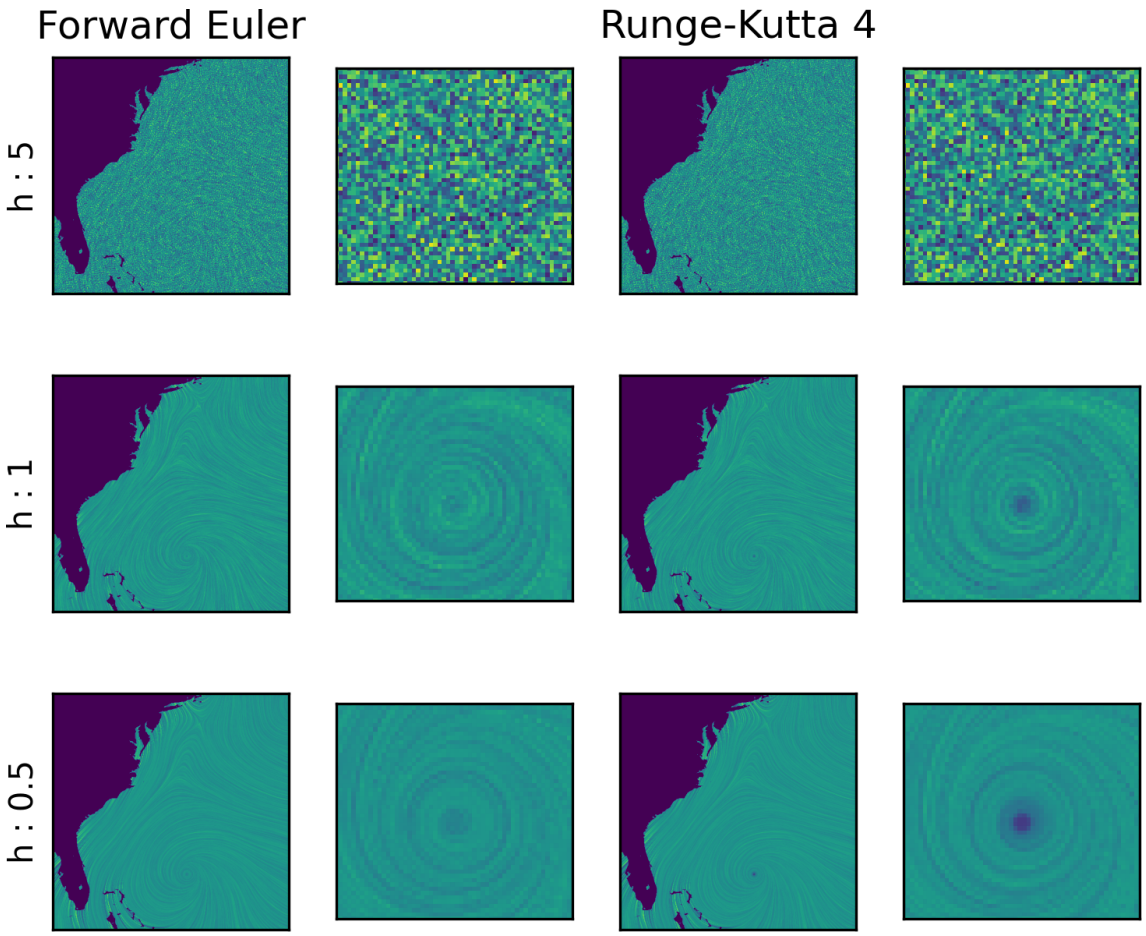

Figure 8: Visualization of the Isabel data using different schemes and step lengths. For each scheme and step length I plot the full image (left) and a zoomed in region (right); corresponding to the same region used in exercise 1.

2.4 Discussion

The LIC technique gives a global picture of the vector field, and is visually appealing and easy to understand. It did, however, prove somewhat difficult to find a balance between making the patterns clear using long lines, and not blurring out the picture too much. Surprisingly it didn't help to increase the resolution – at all. This might however be a visual deception, as the pictures are displayed as rather small in this report.

Compared to the field lines, the LIC images display the vector field everywhere and all features are capture. The field lines display local behaviour, and important features might get lost. However, the streamlines are crystal clear in it's representation without danger of being blurred out. For the different step sizes, it's also not really clear from the pictures whether there is a spiral or if parts of the domain are moving in circles.

The LIC technique in general can in general be used to create very impressive presentation of a vector field. My implementation could probably be further improved to get pictures with clearer pattern – I've seen better examples than the ones I've produced.

References

[1] Marek Fiser. Real time visualization of 3d vector field with CUDA. `http://www.marekfiser.com/Projects/Real-time-visualization-of-3D-vector-field-with-CUDA/4-Vector-field-integrators-for-stream-line-visualization`.

[2] Abdelkrim Mebarki. Adaptive distance grid based algorithm for farthest point seeding streamline placement. *Open Computer Science*, 6, 01 2016.