CMIS18 - HW2

Isabela Blucher

May 4, 2018

The Advection Problem with Semi Lagrangian Time Integration

One of the problems posed this week is related to advection, which is the transport of a substance by bulk motion. We are given the equation $\frac{\partial \phi}{\partial t} = -(u \cdot \nabla)\phi$, which describes the advection process over time. In detail, ϕ defines a scalar field in a regular 2-dimensional grid and $u = (y, -x)^T$ is a given velocity field which defines how the field moves over time.

We use Semi Lagrangian Time Integration to solve the advection equation and to analyze how the scalar field changes over the passing of time. The idea behind the solution is, consider the grid nodes as particles (grid nodes are fixed and particles move in the scalar field), trace the particles back in time to find their value in the past and then copy the value to the present.

Backtracking update equations for the grid coordinates are derived from the definition of a vector field $\frac{\partial \mathbf{x}}{\partial t} = u$, where we apply a first order backward difference to obtain $\mathbf{x}^{t-\Delta t} = \mathbf{x}^t - \Delta t u$, where $\mathbf{x} = (x, y)$ and u is the velocity field. Knowing the past particle coordinates it is possible find the past scalar field value, but since we do not know if $\mathbf{x}^{t-\Delta t}$ is a grid node, we interpolate from the enclosing grid nodes at that location to find the value of $\phi^{t-\Delta t}$.

Having derived that scheme, we can implement it and see the evolution of the scalar field during a period of time. The parameters that can be varied here are grid size (N), distance between nodes (Δx and Δy), maximum time interval (T) and time step (Δt). We initialize a square meshgrid, so that we have x and y coordinates, and we use the Matlab `peaks` function to initialize a scalar field that fits our grid. When running the code, we observe that the scalar field ϕ rotates. Another change to the ϕ is its apparent dissipation, as we can see from the figures below, which are frames from one full revolution of the plot.

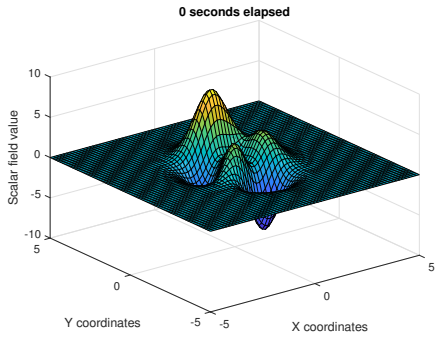

Figure 1: Scalar field at t = 0s

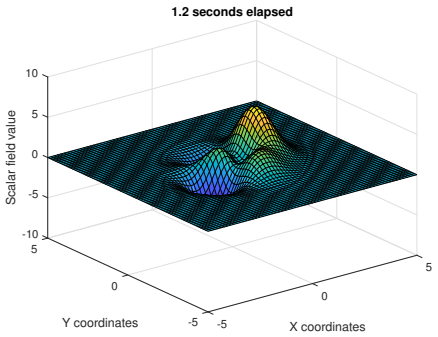

Figure 2: Scalar field at t = 1.2s

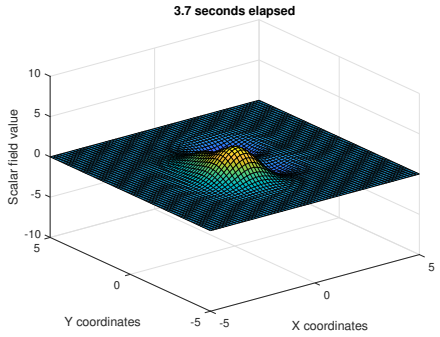

Figure 3: Scalar field at t = 3.7s

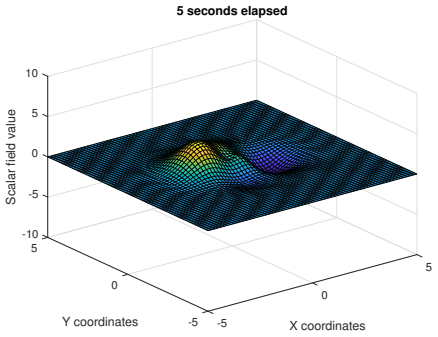

Figure 4: Scalar field at t = 5s

For the simulation shown in Figures 1, 2, 3 and 4, the total time for one revolution is T = 5 seconds. We run 100 iterations to reach that maximum time, which means that our time step is defined by $\Delta t = T/100 = 0.05$. For the grid size we define the variables N = 64 (numer of nodes), and L = 10 (length of grid). The grid goes from $-\frac{L}{2}$ to $\frac{L}{2}$, with node distances defined as $\Delta x = \Delta y = \frac{L}{N}$.

From visual inspection of the figures above, one can see that the shape of the scalar field seems to dissipate with the passing of time. This can be explained due the way that the scheme was formulated with implicit time stepping, which means we are never adding anything to our scalar field. It can also be explained by the interpolation used to find $\phi^{t-\Delta t}$, which smooths out the scalar field and adds to the dissipation effect.

After implementation we define an error measure $E^t = \sum_{ij} \Delta x \Delta y \phi_{ij}^t$ where t is the current time step, for one full revolution we measure $\varepsilon^T = |E^0 - E^T|$, and want to see how parameter changes affect ε^T, with T as a fixed value. We focus on changing the values of $\Delta x, \Delta y$ to see how distance between nodes affect it, and also Δt to see the consequences of changes in the time step.

We fix $\Delta x, \Delta y = 10/64$ and $T = 5$ change Δt, and by testing the graph we get the following plot.

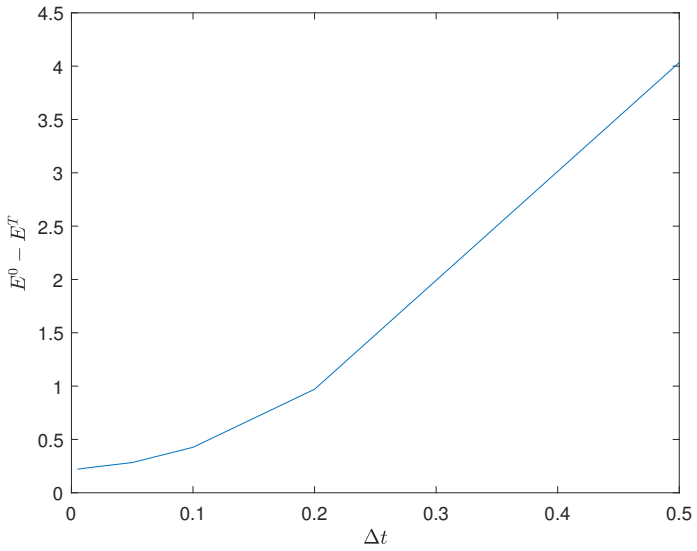

Figure 5: Error measure relating to time step variation

The bigger the time step, the bigger the error measure. For very small values of Δt it would be expected that our error measure value blows up due to precision problems.

Now, by fixing the time step and changing the grid step values we get Figure 6.

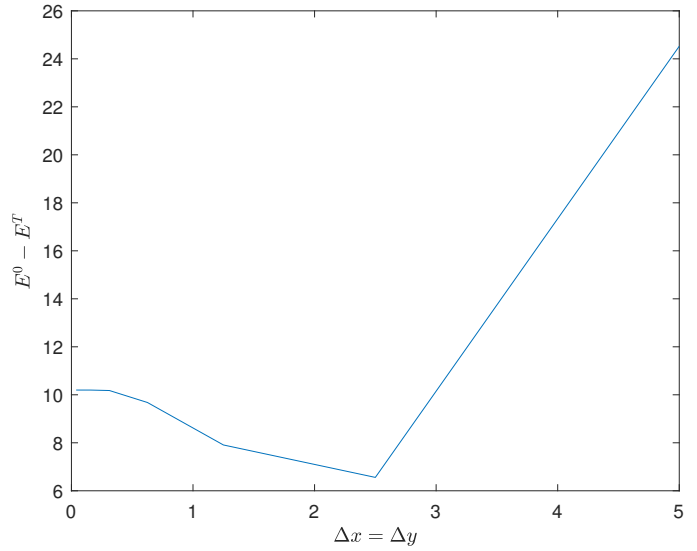

Figure 6: Error measure relating to grid step variation

With Figures 5 and 6, we could define a maximum acceptable error, such as $e = (E^0 - E^T) * 0.05$, and see where the two plots intersect, this would give us optimal parameter values for the advection problem.

The problem here is that we are not considering how efficient our algorithm is for that selection of parameters, so we could repeat the plotting process, but instead of using an error measure, the actual runtime of the program is computed.

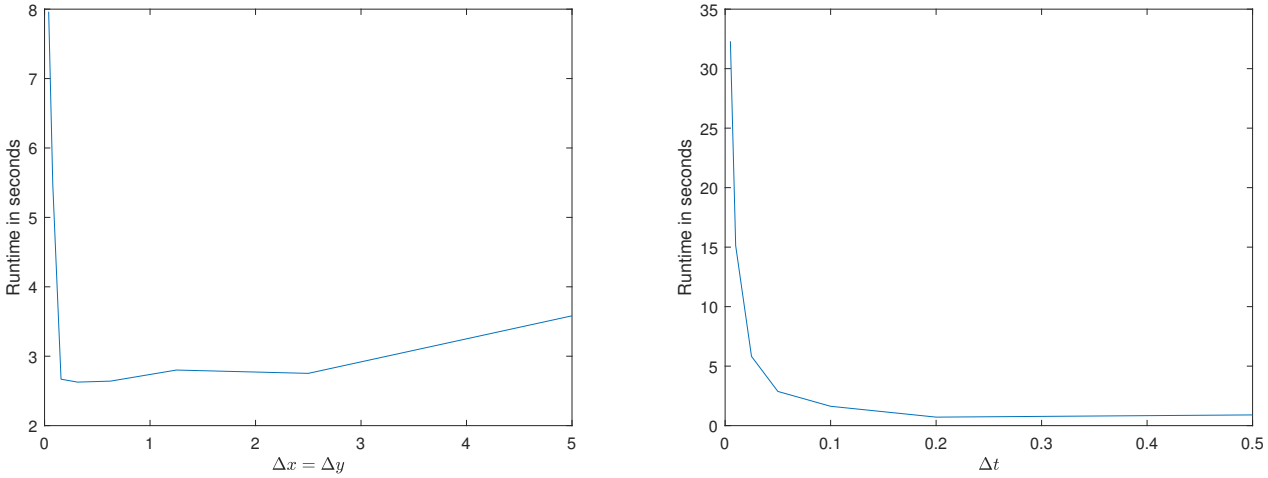

Figure 7: Plots of runtime in seconds relating to grid step and time step variations

For the plots in Figure 7, we apply the same process of fixing a maximum acceptable time we're willing to wait for the computations and then find the intersection between the two pictures. Now we have optimal parameter values for our error measure and for an acceptable runtime, which means we can find optimal parameters for an accurate and efficient solution to the advection problem.

The Mean Curvature Flow Problem

The second problem we're dealing with is the mean curvature flow problem, also referred to as curve shortening flow when dealing with curves. The mean curvature flow is given by the PDE $\frac{\partial \phi(\mathbf{x},t)}{\partial t} = \nabla_{\mathbf{x}} \cdot \frac{\nabla_{\mathbf{x}} \phi(\mathbf{x},t)}{\|\nabla_{\mathbf{x}} \phi(\mathbf{x},t)\|}$, where $\phi(\mathbf{x}, t)$ is initialized as a signed distance field and $\nabla_{\mathbf{x}} = (\frac{\partial}{\partial \mathbf{x}})^T$.

We discretize our problem for a 2-dimensional grid so we can derive a scheme to update ϕ as the time passes. The final update scheme for our problem $\phi_{ij}^{t+1} = \phi_{ij}^t + \Delta t k_{ij}$.

To choose what time step we're using we need to use the mechanics known as the Courant-Friedrichs-Lewy condition so that we can prevent that the speed of ϕ becomes larger than what the grid can represent. The condition essentially stated that $\Delta t \leq \frac{h}{2\kappa_{max}}$, where $\kappa_{max} = \frac{1}{max(\Delta x, \Delta y)}$ and $h = min(\Delta x, \Delta y)$.

A simulator was implemented with our scheme so we can observe the evolution of a given signed distance field over the passing of time. We are given an input image which is transformed into a signed distance field and then run our algorithm on it for chosen parameter values of $\Delta x, \Delta y$. Δt is always computed as $\frac{h}{2\kappa_{max}}$ so that we always satisfy the CLF condition.

Also, the solution for the ghost nodes was similar to the one used last week. We add top, bottom, left and right borders of ghost nodes that we can use in the computations of our update of k_{ij}. All of the ghost nodes have the same ϕ value as the boundary node adjacent to it.

We run the algorithm for a given interval of T, with time steps of Δt. By choosing $T = 100$ and $\Delta x = \Delta y = 1$, we get the input and output images in Figure 8.

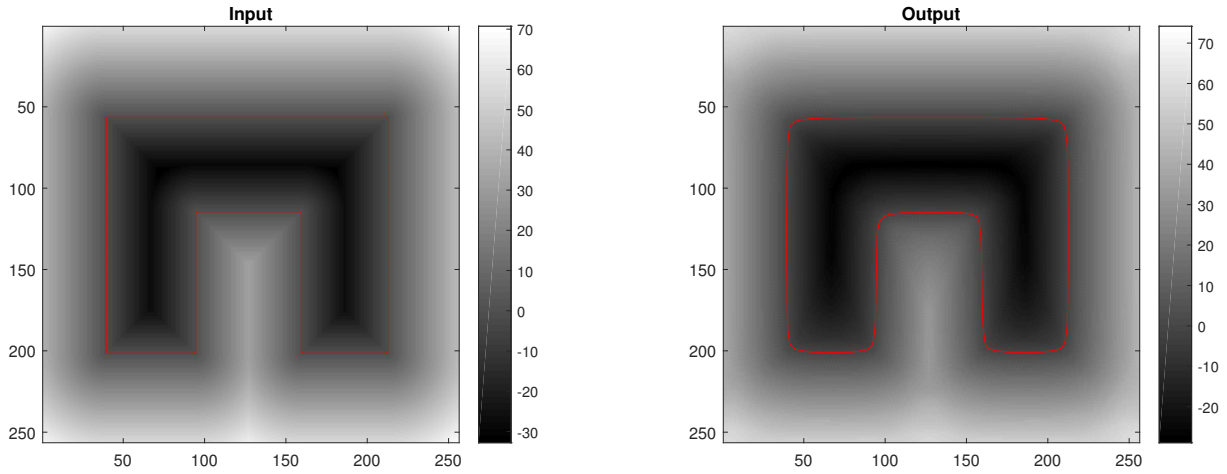

Figure 8: Inut and output state of signed distance field under mean curvature flow after with T = 100

By inspecting the images, one can conclude that the first thing that changes in the signed distance field are the corners, which become increasingly round. This happens because corners have bigger gradient values, and looking at the given PDE we can see that the bigger the gradient, the faster the shape change.

One hypothesis that we could formulate is that, for a big enough T we could probably see our signed distance field become circular enough that it would start shrinking inward and out of existence. This happens because round spheres evolve under mean curvature flow by shrinking inward. It's also possible to play with the $\Delta x, \Delta y$ parameter values and see for what values the shrinking seems to be faster than others.

Running the algorithm for $T = 1000$ and $\Delta x = \Delta y = 1.5$, we get Figure 9 as the output.

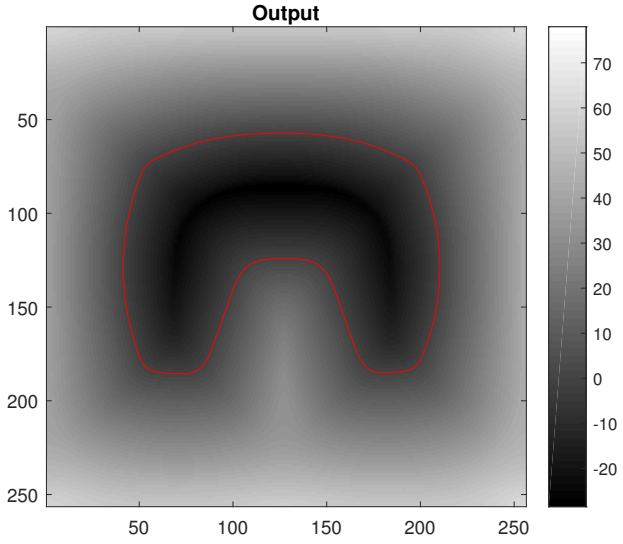

Figure 9: Output image and signed distance field for T = 1000

Running the algorithm for $T = 3000$ and $\Delta x = \Delta y = 1$, we get Figure 9 as the output.

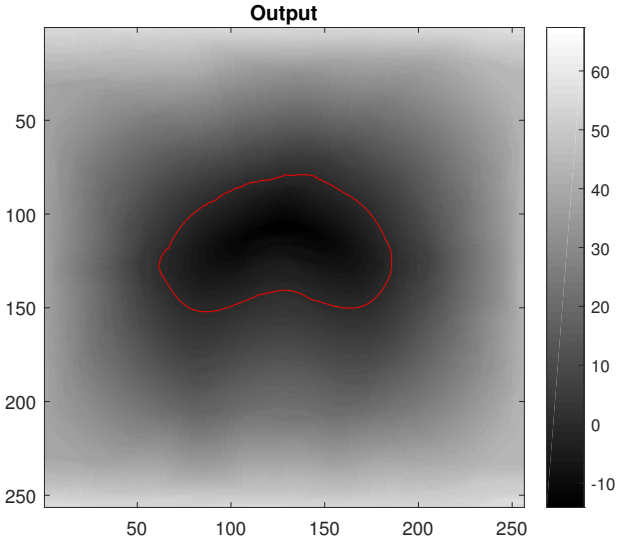

Figure 10: Output image and signed distance field for T = 3000

Figure 10 shows the signed distance field at a much smaller and rounder shape than the original input, which means that our hypothesis of shrinking inwards can be confirmed.

Another experiment that can be done is change $\Delta x, \Delta y$ and measure the runtime in seconds for a fixed simulated time T. This way, we can see how making the problem bigger or smaller (store more or less variables and do more or less computations), affects the efficiency of the mean curvature flow algorithm.

For this experiment we fix $T = 100$ and run for $\Delta x = \Delta y \in \{0.5, 0.75, 1, 1.25, 1.5, 2\}$ and measure the runtime for each of the different parameter values to see how it affects the efficiency of the algorithm.

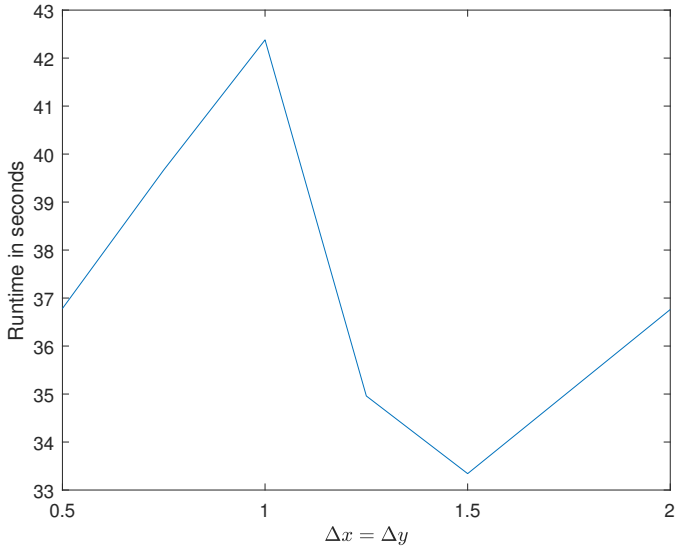

Figure 11: Runtime relating to changes in $\Delta x = \Delta y$

Figure 11 shows the runtime efficiency for different values of the grid step. It seems that the runtime does not vary that much, but the optimal parameter value for running the algorithm is $\Delta x = \Delta y = 1.5$.

In conclusion many experiments can be done for both problems and algorithm implementations, but one of them is playing with parameter values and analyzing how it changes a defined error measure or efficiency. This can help us determine the best values for optimal robustness and efficiency.