Computational Methods in Simulation hand-in week 1

KXS806

1 THEORETICAL PART

1.1 Semi Lagrangian Time Integration

The Semi-Lagrangian scheme is a numerical method that is widely used in numerical weather prediction models for the integration of the equations governing atmospheric motion. Here's the basic idea of doing this procedure:

- Convert nodes into particles
- Trace particles back in time
- Find the values in the past
- Copy the values to the present

Here we will use advection terms to illustrate how the Semi Lagrangian Time Integration works. First, we have the PDE as follows:

$$\frac{\partial \phi}{\partial t} = -(\mathbf{u} \cdot \nabla)\phi$$

where ϕ is a scalar field and u is the associated velocity field.
First, we convert nodes into particles and then trace particles back in time as shown in fig 1.

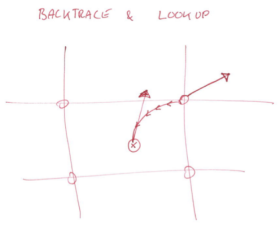


Fig. 1. trace back

Here we need to use interpolation to look up the value as the particle we are looking at is not located at any already given nodes. In our implementation, we use linear interpolation to get the value.

After tracing back in time we get the value of the particle, then we simply copy forward in time and copy the value to the current node position.

To be more specific, we need to do some math here. After we transform into particles, from a particle viewpoint we can know that any quantity $\phi(t)$ will change in time as $\frac{D\phi(t)}{Dt} = k$.

From a grid viewpoint, ϕ is moving through space and we should think of it as a spatially varying function $\phi(x, t)$. So

$$\begin{aligned}\frac{D\phi(\mathbf{x}, t)}{Dt} &= \sum_i \frac{\partial \phi}{\partial \mathbf{x}_i}\frac{\partial \mathbf{x}_i}{\partial t} + \frac{\partial \phi}{\partial t} \\ &= (\mathbf{u} \cdot \nabla)\phi + \frac{\partial \phi}{\partial t}\end{aligned}$$

So we can say the particle view is the same as the grid view

$$\frac{D\phi(t)}{Dt} = (\mathbf{u} \cdot \nabla)\phi + \frac{\partial \phi}{\partial t}$$

In order to solve $\frac{D\phi(t)}{Dt} = k$ we can choose an implicit first order time integration

$$\phi^t = \phi^{t-\Delta t} + \Delta t k$$

As we already know k = 0, we only need to find $\phi^{t-\Delta t}$ so we can find ϕ^t. We need to trace back in time here to find the imaginary particle that will hit the location we are currently updating.
From the definition of a vector field, we know that

$$\frac{\partial \mathbf{x}}{\partial t} = \mathbf{u}$$

We apply a first-order backward finite difference approximation

$$\frac{\mathbf{x}^t - \mathbf{x}^{t-\Delta t}}{\Delta t} = \mathbf{u}$$

Using this equation we can fin d the particle position at $t - \Delta t$

$$\mathbf{x}^{t-\Delta t} = \mathbf{x}^t - \Delta t \mathbf{u}$$

So later in code, we will use this to find the node position we want to know ϕ^t for. Then we can update ϕ by

$$\phi^t \leftarrow \phi^{t-\Delta t}$$

1.2 Mean Curvature Flow

Mean curvature flow is used to smooth the surfaces, it can also be used to do matching from one shape to another shape. It is given by the PDE as follows.

$$\frac{\partial \phi(\mathbf{x}, t)}{\partial t} = \nabla_{\mathbf{x}} \cdot \frac{\nabla_{\mathbf{x}}\phi(\mathbf{x}, t)}{\|\nabla_{\mathbf{x}}\phi(\mathbf{x}, t)\|}$$

where $\phi(\mathbf{x}, t) : \mathbb{R}^n \times \mathbb{R}_+ \mapsto \mathbb{R}$ is a level set function and $\nabla_{\mathbf{x}} = \left(\frac{\partial}{\partial \mathbf{x}}\right)^T$.
As shorthand

$$\frac{\partial \phi}{\partial t} = \underbrace{\nabla \cdot \frac{\nabla \phi}{\|\nabla \phi\|}}_{\equiv \kappa}$$

we can define the right side as k, so $\kappa = \nabla \cdot \frac{\nabla \phi}{\|\nabla \phi\|}$. We rewrite this using $\nabla \cdot (a\mathbf{v}) = \nabla a \cdot \mathbf{v} + a\nabla \cdot \mathbf{v}$ where $a \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^n$

$$\kappa = \nabla\phi \cdot \nabla\left(\frac{1}{\|\nabla\phi\|}\right) + \frac{1}{\|\nabla\phi\|}\nabla \cdot \nabla\phi$$

Using $\nabla \cdot \nabla\phi = \nabla^2\phi$ and $\|\nabla\phi\| = \sqrt{\nabla\phi^T\nabla\phi}$

$$\kappa = \nabla\phi \cdot \nabla\left(\left(\nabla\phi^T\nabla\phi\right)^{-\frac{1}{2}}\right) + \frac{\nabla^2\phi}{\|\nabla\phi\|}$$

So

$$\kappa = \nabla\phi \cdot \nabla\left(\left(\nabla\phi^T\nabla\phi\right)^{-\frac{1}{2}}\right) + \frac{\nabla^2\phi}{\|\nabla\phi\|}$$

Using the chain rule

$$\nabla\left(\left(\nabla\phi^T\nabla\phi\right)^{-\frac{1}{2}}\right) = -\frac{1}{2}\left(\nabla\phi^T\nabla\phi\right)^{-\frac{3}{2}} 2(\nabla(\nabla\phi))\nabla\phi$$

Introducing the Hessian symbol $\mathbf{H} = (\nabla(\nabla\phi))$

$$\kappa = \frac{\mathrm{tr}(\mathbf{H})}{\|\nabla\phi\|} - \frac{\nabla\phi^T\mathbf{H}\nabla\phi}{\|\nabla\phi\|^3}$$

where $\nabla^2\phi = \mathrm{tr}(\mathbf{H})$

1.2.1 ***Spatial Discretization in 2D.*** Then we need to give a definition of spatial discretization in 2D: given a 2D grid and let ϕ_{ij} to be the value of ϕ at node (i,j). With central difference approximations, we can get

$$\nabla\phi_{i,j} \approx \begin{bmatrix} D_x\phi_{ij} \\ D_y\phi_{ij} \end{bmatrix} = \begin{bmatrix} \frac{\phi_{i+1,j}-\phi_{i-1,j}}{2\Delta x} \\ \frac{\phi_{i,j+1}-\phi_{i,j-1}}{2\Delta y} \end{bmatrix}$$

So we can get

$$\left\|\nabla\phi_{ij}\right\| \approx \sqrt{\left(D_x\phi_{ij}\right)^2 + \left(D_y\phi_{ij}\right)^2}$$

and we define the right-hand side as g_{ij} Then we can use 2D discretization and get

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2\phi}{\partial x^2} & \frac{\partial^2\phi}{\partial x y} \\ \frac{\partial^2\phi}{\partial x y} & \frac{\partial^2\phi}{\partial y^2} \end{bmatrix} \approx \begin{bmatrix} D_{xx}\phi & D_{xy}\phi \\ D_{xy}\phi & D_{yy}\phi \end{bmatrix}$$

so we can get those approximations:

$$D_{xx}\phi_{ij} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2}$$

$$D_{yy}\phi_{ij} = \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2}$$

$$D_{xy}\phi_{ij} = \frac{\phi_{i+1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j+1} + \phi_{i-1,j-1}}{4\Delta x\Delta y}$$

So after putting these things together, we can get the approximation of k as

$$k_{ij} = \frac{\left(D_x\phi_{ij}\right)^2 D_{yy}\phi_{ij} + \left(D_y\phi_{ij}\right)^2 D_{xx}\phi_{ij} - 2D_{xy}\phi_{ij}D_x\phi_{ij}D_y\phi_{ij}}{g_{ij}^3}$$

1.2.2 ***numerical problems.*** However, by looking at the formula we can see there can be some numerical problems such as

- k_{ij}could become unrealistic large or small if $g_{ij} \to 0$
- If $g_{ij} \to 0$ then we may have a division by zero
- If ϕ is a signed distance field then $g_{ij} \approx 1$ almost everywhere

We can solve those problems with three kinds of Numerical Remedies

- We can add some value and we redefine our discretization as

$$k_{ij} = \frac{(\cdots)}{g_{ij}^3 + \varepsilon}$$

  where the user specified constant $\varepsilon \ll 1$ is a threshold value of the same order as the numerical precision
- we can modify g_{ij} if ϕ is a signed distance field as

$$g_{ij} \leftarrow \begin{cases} g_{ij} & \text{if } g_{ij} > \frac{1}{2} \\ 1 & \text{otherwise} \end{cases}$$

- we can clamp the discrete approximation as there's a bound on the max value of mean curvature.

$$k_{ij} \leftarrow \max\left(-\kappa_{\max}, \min\left(k_{ij}, \kappa_{\max}\right)\right)$$

1.2.3 ***Temporal Discretization.*** Using 1st order forward difference we have

$$\frac{\partial\phi_{ij}}{\partial t} \approx \frac{\phi_{ij}^{t+1} - \phi_{ij}^t}{\Delta t}$$

The final update scheme becomes

$$\phi_{ij}^{t+1} = \phi_{ij}^t + \Delta t k_{ij}$$

where $\Delta t \ll 1$

1.2.4 ***CFL condition.*** When we are using the final scheme with a fast speed there might be some problems, for example, artifacts appearing just before a blow-up of ϕ. This means that time-step ϕ_{ij} would have changed more than the grid size can express.

So we need to prevent the update scheme to use a speed that is larger than the grid can represent. One possible way is called CFL condition as follows:

CFL conditions must hold at all grid nodes that

$$\frac{\mathbf{u}_x\Delta t}{\Delta x} + \frac{\mathbf{u}_y\Delta t}{\Delta y} \le C_{\max}$$

where u is the speed and C_max is the Courant number which is 1 in our case. So for our problem, it is

$$\frac{\kappa_{\max}\Delta t}{\Delta x} + \frac{\kappa_{\max}\Delta t}{\Delta y} \le 1$$

Bu defining h = $min(\Delta x, \Delta y)$ we can find the conservative bound

$$2\frac{\kappa_{\max}\Delta t}{h} \le \frac{\kappa_{\max}\Delta t}{\Delta x} + \frac{\kappa_{\max}\Delta t}{\Delta y} \le 1$$

and get a condition as:

$$\Delta t \le \frac{h}{2\kappa_{\max}}$$

Later in the practical part, we get Δt <= 0.0018934911242603552, and we used 0.001 as the time stamp.

2 PRACTICAL PART

2.1 Advection Problem

In this part, we managed to simulate an advection-type problem with Semi-Lagrangian Implicit time integration. We used an analytical velocity field $u(x, y) = (y, x)^T$ and used total positive volume as a measure of conservation.

There will be some errors during simulation because we are using node to stand for particles and some points between four points will have a value depending on the average value of four surrounding points which is our interpolate strategy. So each iteration there will be some smoothing on the peak of values and cause some error.

Using grid size as [40,80,100,120,140] and time stamp value as [0.03,0.02,0.01,0.005] we did the simulation and computed the error and got the error image as fig 2

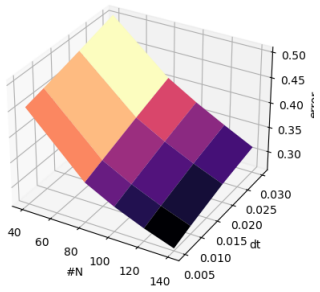


Fig. 2. error plot

From the plot, we can see the curve in the x-axis direction has the shape of a quadratic curve. It's because the real grid size is not N, the interpolation is a linear method and it will take the average distance into account. In a 2D field, the distance between two points has a square root so it will have that shape.

For the experiment, we managed to visually show the whole procedure of the rotation and we got the video as follow links:

- rotation with 40 nodes and 0.03 time-step
- rotation with 140 nodes and 0.03 time-step
- rotation with 140 nodes and 0.005 time-step

2.2 Mean Curvatue Flow

In this part, we first defined a signed distance field and then implemented a function to compute mean curvature with von Neumann boundary condition and also modified g_{ij} to be 1 if it's less than 1/2 to avoid k becoming infinite.

Then we created a grid instance with the shape of a square as fig 3

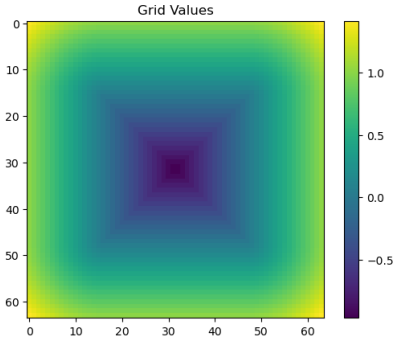
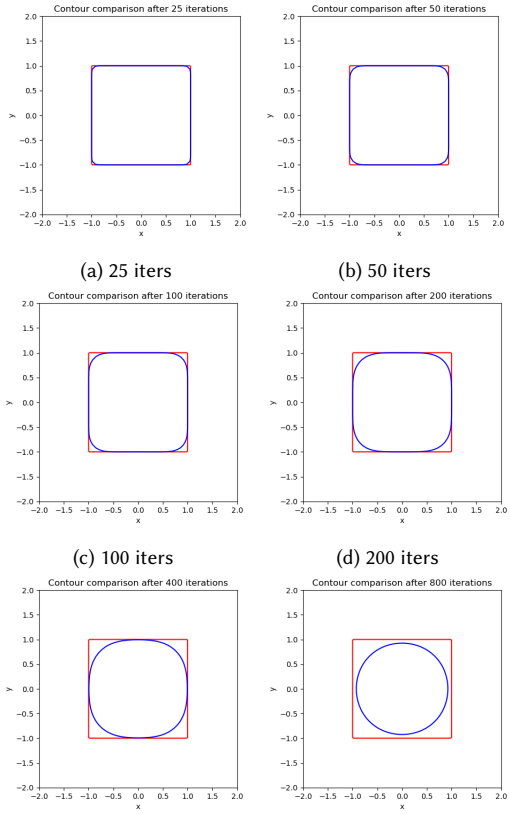


Fig. 3. square grid

Then we do the compute with dt = 0.0001 with iter number from [25,50,100,200,400,800] and get the following result:



(a) 25 iters (b) 50 iters

(c) 100 iters (d) 200 iters

(e) 400 iters (f) 800 iters

And this is the figure of the mean curvature field from the last iteration:

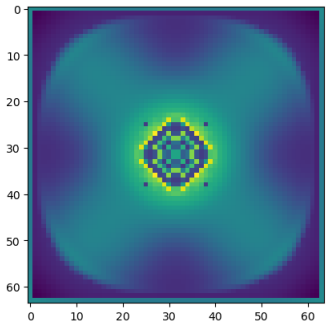
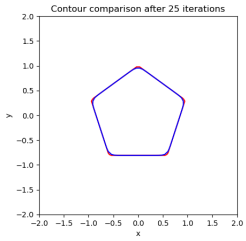


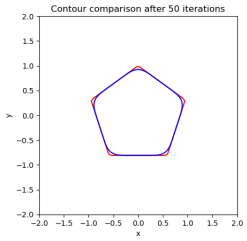
Fig. 5. mean curvature field

From the figure we can notice there are some black areas in the four corners, it's because the ghost node in those places are unused and being contaminated by the data from other ghost nodes around them.

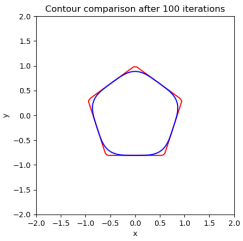
For the experiment part we managed to change the shape of the initial grid and get the result as follows:



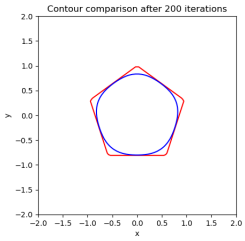
(a) 25 iters



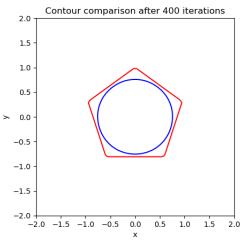
(b) 50 iters



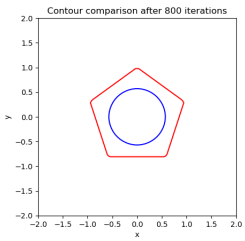
(c) 100 iters



(d) 200 iters



(e) 400 iters



(f) 800 iters

From the above figures, we can see the function works well on different kinds of polygons.

3 CONCLUSION

We learned how to apply semi-lagrangian time integration in the advection problem and implemented it. Also, we learned mean curvature flow and applied it to different kinds of polygons.