Semi-Lagrangian Crank-Nicolson Method

Maximilian Williams

June 2021

What is this?

A litle latex document for making sure that I understand what I am talking about for the semi-lagrangian Crank-nicolson scheme.

The problem

We wish to solve the advection diffusion equation using numerical methods. We will keep our analysis to 1-Dimensional for simplisity, but it should be very easy to generalize to 2 and 3 dimensions.

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = \kappa\frac{\partial^2 u}{\partial t^2} + s(u, x, t) \tag{1}$$

If we assume that a is constant, then we can employ the Crank-Nicolson scheme to solve this. This is a semi-explicit scheme. The general idea is to approximate the spatial derivatives, apply the Crank-Nicolson scheme and solve for the u's at timestep $n+1$ given you know them at a timestep n. We do this by writting this equation in matrix form and using linear algebra to solve for the quantities at timestep $n+1$. However, this may become unstable and inaccurate if a is non constant. To rememdy this, we employ the semi-lagrange crank nicolson method.

Crank Nicolson

If we have an equation of the form:

$$\frac{\partial u}{\partial t} = F(x, t, u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial t^2}) \tag{2}$$

If we discritize the domain in time and space steps we can write the quantity u_i^n to mean u at timestep n and spacestep i then we can write the above equation as:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}(F_i^{n+1} + F_i^n), \tag{3}$$

where Δt is the length of the timestep and the arguments of F has been suppressed. Note that F_i^n is F evaluated at time step n and space step i. The evaluation of F_i^n can be done using finite difference schemes.

Semi-Lagrange Crank-Nicolson Method

We wish to solve a general problem of the form:

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = f, \tag{4}$$

where f is arbitrary smooth function of t, x, u and its derivatives.
We can write this using the material deriviave:

$$\frac{Du}{Dt} = f \tag{5}$$

and integrate in the lagrangian frame:

$$u^{n+1} = u_*^n + \int_{n\Delta t}^{(n+1)\Delta t} f d\tau \tag{6}$$

We can discreize this integral simply using the midpoint rule to give:

$$u^{n+1} = u_*^n + \frac{\Delta t}{2}(f_*^n + f^{n+1}). \tag{7}$$

Here we note what the $*$ subscript means. Lets suppose we have a lattice point at a location x within our lattice. x_* is then the location of the fluid at x a time Δt ago. In other words, in a time Δt the fluid travelled from x_* to x. Note that x_* is not necessarily the location of a lattice point. For this reason, we have to use extrapolation to find u and f at x_*.

Finding x_* to first order

Lets suppose that we are at a time t and considering a particle at position x_0. The evolution of the position of this particle is described by:

$$\frac{\partial x}{\partial t} = v, \tag{8}$$

and is simply due to it being advected along. We note the position of this particle at a time $t - \Delta t$ as x_*. To first order in time, we can write:

$$x_* = x_0 - v(x_*, t - \Delta t)\Delta t + \mathcal{O}(\Delta t^2). \tag{9}$$

You will note that x_* is mentioned on both sides here, and so we cannot solve for it exactly. Rather, we use an iterative method using:

$$x_*^{(k+1)} = x_0 - v(x_*^{(k)}, t - \Delta t)\Delta t \tag{10}$$

where $x_*^{(k)}$ is the value of x_* obtained at step k of iteration. We might stop the algorithm once $x_*^{(k+1)}$ and $x_*^{(k)}$ differ by only some error ϵ. Note that the value of the velocity v at the location x_* is not in general a lattice point and we only know the value of v at lattice points. To get the value of v at x_* we have to use interpolation.

Interpolation

There are several method for interpolation that can be used. The less accurate the interpolation, the higher the numerical diffusivity induced in the solution. It was found in other work that bicubic interpolation is the lowest order interpolation than can be used for simulating flows relavant to geologic scales. So here we choose to use the bicubic interpolation. Apparently this can be done in numpy, and since Im running low on time I think I'll just use numpys implimentation. However, I may need to code this by hand in the polar case.

Two level schemes and second order accurate determination of x_*

We are going to impliment a two level time scheme. We do this because it allows us to evaluate x_* to second order time accuracy, which is needed to stop significant numerical diffusion in our numerical solution. We will assume that we have already developed a scheme to update v at a time t given that we know u at a time $t - \Delta t$, so we will not worry about how v is updated, we will simply say that we always know v at the current time and all previous relavant times. We begin our simulation with knowing v and u at an initial time t. We then use 7 to get u at time $t + \Delta t$. In doing this, we need to compute x_* and we do this using our first order method above. Next we use our knowlegde of u at time $t + \Delta t$ to find v at $t + \Delta t$ which we assume is implimented elsewhere. We now know u and v at times t and $t + \Delta t$. We next work to obtain $t + 2\Delta t$. To do this, we use 7 with double the timestep, that is:

$$u^{n+2} = u_*^n + \frac{2\Delta t}{2}(f_*^n + f^{(n+2)}). \tag{11}$$

We note that this requires the computation of u_* and f_* which require the computation of x_*. Here is where the two step scheme comes in handy. We know the advection velocity v at times t and $t + \Delta t$. We apply a similar argument to 9 to get:

$$x_* = x_0 - v(x_*, t + \Delta t)(2\Delta t), \tag{12}$$

where x_0 is the particles position at t. This is second order accurate in Δt. We also use a similar algorithm to the first order accurate case:

$$x_*^{(k+1)} = x_0 - v(x_*^{(k)}, t + \Delta t)(2\Delta t). \tag{13}$$

So, we are now able to update u from time t to time $t + 2\Delta t$ with second order accuracy. Note that we required the time between these two, $t + \Delta t$ to do this. This is why we need this two level scheme. We can now apply similar reasoning to update the other half step, getting u at $t + 3\Delta t$ using $t + \Delta t$. We repeat this flip flopping between the two time levels and progress until we want to terminate the program.

A more detailed look at my case

In my case I want to solve the 2D analog of

$$\frac{\partial T}{\partial t} + v\frac{\partial T}{\partial x} = \kappa \nabla^2 T + s, \tag{14}$$

where s is some source term that depends only on space x.
We will write $\vec{T}$ to indicate the vector repressenting the temperature in the domain. We will use $\hat{}$ to denote matricies. Writting this in the semi-lagrangian crank-nicholson scheme we get:

$$\vec{T}^{(n+1)} = \vec{T}_*^n + \frac{\Delta t}{2}(\kappa(\hat{\mathcal{L}}\vec{T})^{(n+1)} + \vec{s}^{(n+1)} + \kappa(\hat{\mathcal{L}}\vec{T})_*^{(n)} + \vec{s}_*^n), \tag{15}$$

where $\mathcal{L}$ is the matrix repressentation of the Laplace operator and superscripts denote the timestep that the field is taken at. The subscript $*$ is treated similar to in the general case above. We note that we can write this as:

$$(\hat{I} - \kappa\frac{\Delta t}{2}\hat{\mathcal{L}})\vec{T}^{(n+1)} = \vec{T}^{n}_{*} + \frac{\Delta t}{2}(\vec{s}^{(n+1)} + \kappa(\hat{\mathcal{L}}\vec{T})^{(n)}_{*} + \vec{s}^{n}_{*}) \tag{16}$$

If we assume that our source term is known for all times (even into the future) as is the case here, then the entire right hand side of the above equation can be computed. So long as we can invert the matrix $(\hat{I} - \kappa\frac{\Delta t}{2}\hat{\mathcal{L}})$ we then have a method for updating T. We can use our two time level scheme above to do all this. I think I have all the tools to do this now.

Dealing with Tensors

A little note to myself:
I have to be REALLY REALLY careful about assuming tensor symmetries.
In two dimensions, T becomes a rank 2 tensor, with each entry being the value of temperature at a lattice point. Throughout this discussion in higher dimensions we will use Eistien notation. So T becomes T^i_j. The discritized laplace operator acting on T^i_j returns another rank 2 tensor of similar form. We note that the spatial derivatives which $\mathcal{L}$ repressents occure in the two directions repressented by the i and j indexes in T^i_j. When applying $\mathcal{L}$ on T^i_j we must therefore contract over i and j. Therefore, $\mathcal{L}$ is a rank 4 tensor, $\mathcal{L}^{\alpha j}_{\beta i}$. So, all the matricies in 1 dimension become rank 4 tensors. We can now write out a two dimensional version of equation **??** as:

$$(\delta^\alpha_i\delta^j_\beta - \kappa\frac{\Delta t}{2}\mathcal{L}^{\alpha j}_{\beta i})(T^i_j)^{(n+1)} = (T^i_j)^{(n+1)}_{*} + \frac{\Delta t}{2}((s^i_j)^{(n+1)} + (s^i_j)^{(n)} + \kappa(\mathcal{L}^{\alpha j}_{\beta i}T^i_j)^{(n)}_{*}) \tag{17}$$

Now we must clarify how we can "invert" the rank 4 tensor $M^{\alpha j}_{\beta i} = \delta^\alpha_i\delta^j_\beta - \kappa\frac{\Delta t}{2}\mathcal{L}^{\alpha j}_{\beta i}$. What do we even mean by invert here? By the inverse of M we mean the rank 4 tensor N such that:

$$N^{\beta i'}_{\alpha j'}M^{\alpha j}_{\beta i} = \delta^{i'}_i\delta^{j'}_j \tag{18}$$

I'm just guessing this inverse exists. It should. There is no way in hell I want to write the program to do this, so I will use numpy to do this (they do have a method for this). We have two more important questions. What are the actual values of $\mathcal{L}^{\alpha i}_{\beta j}$ (this one is critical) and how can we impliment $*$ in a matrix format. The latter is not all that important, but it would be nice to do.

Discritized Laplace Operator in 2D cartesians

We can view $\mathcal{L}^{\alpha j}_{\beta i}$ with indexes α and β fixed as a rank 2 tensor that we are constracting (over both indexes) with T^i_j.
Lets take the simple case where we approximate the second order spatial derivative $\frac{\partial^2 T}{\partial x^2}T$ by $\frac{T^{i+1}_j - 2T^i_j + T^{i-1}_j}{\Delta x^2}$ and similary for $\frac{\partial^2 T}{\partial x^2}T$.

$$T^i_j = \begin{bmatrix} T^0_0 & T^1_0 & T^2_0 \\ T^0_1 & T^1_1 & T^2_1 \\ T^0_2 & T^1_2 & T^2_2 \end{bmatrix} \tag{19}$$