Linear Advection and Diffusion

AMS 209: Foundations of Scientific Computing
Fall 2017
Martin Rodriguez

Abstract

The goal of this project to use model a one-dimensional advection and diffusion equation numerically. For diffusion we used a centered spatial discretization and forward time. In the case of advection, we continue to use forward time and use both an backward space also called upwind and a centered difference. The code is tested on a grid size of $N = 32$ and $N = 128$ for the diffusion, advection, and the advection-diffusion case. The upwind method provides the best solution for the advection partial differential equation (PDE) and the center finite difference method solutions explodes eventually. However, when viscosity is added then the center method resolves the PDE.

1 Methods

There were two parts to the project. We needed to implement the PDE solver in Fortran and a Python scheduler. The Fortran implementation solves

$$u_t + au_x = \kappa u_{xx}, \tag{1}$$

with two subcases where $a = 0$ and $\kappa = 0$. In the following sections we will describe the discretization methods and the modules implementation.

1.1 Discretization

When solving the diffusion equation then we discretization using

$$u_i^{n+1} = u_i^n + \kappa \frac{\Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n), \tag{2}$$

and we need to satisfy the CFL condition

$$\kappa \Delta t \leq \frac{\Delta x^2}{2}.$$

The diffusion equation uses the initial condition

$$u_0 = \begin{cases} 0, & x \in [0, 1) \\ 100, & x = 1 \end{cases}$$

The diffusion solver is implemented `pde_solver_module.F90` in the function `diffuse_update()`.

When solving the advection PDE then we use the upwind discretization

$$u_i^{n+1} = u_i^n - \frac{a\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) \tag{3}$$

and the centered discretization shown below

$$u_i^{n+1} = u_i^n - \frac{a\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n). \tag{4}$$

The advection method needs to satisfy the CFL condition

$$|a|\Delta \leq \Delta x. \tag{5}$$

This portion is implemented in the `pde_solver_module.F90` in the function `advect_update()`.

1.2 Fortran Modules

In this code we implemented a total of four modules and the driver. The modules are the following:

- `advection_diffusion.F90`: This file is driver of the entire code. All of the subroutines are called from here and results are are saved to a text file.
- `setup_module.F90`: This module calls the file `pde.init` and sets up all of the runtime parameters.
- `initialize_module.F90`: This modules includes the subroutines `grid_init()`, `diffuse_init(uold)`, `advect_init(x,uold)`, and the `simulation_init(x,uold)`. This module serves to initialize the grid and the initial conditions for the diffusion equation or the advection or the advection-diffusion equation.
- `pde_solver_module.F90`: This module includes all the necessary pieces for solving the PDE. It includes `cfl()`, `bc(unew)`, `diffuse_update(uold,unew)`, `advect_update(uold,unew)`, `check_error(uold,unew)`, and `compute_timestep(t,frameNumber,writeOutput)`. The `compute_timestep` module computes the time step but checks whether the time step is small enough to write the data at every tenth of a second. The input `frameNumber` counts the number of frames that have been saved to a text file.

2 Results and Discussion

(a) The Figures 1 through 4 show the solution to diffusion equation as it reaches steady state with $N = 32$. The $t_{max} = 0.7$. The figures 7 through 8 show the diffusion equation with $N = 128$.

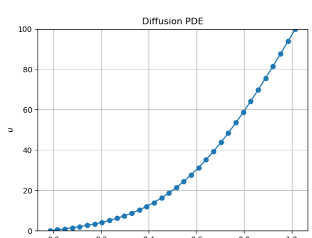

Figure 1: This is diffusion at $0.2t_{max}$.

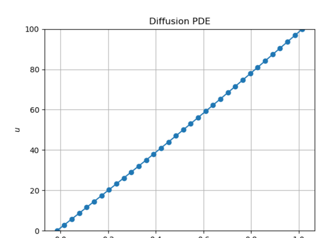

Figure 2: This is diffusion at $0.5t_{max}$.

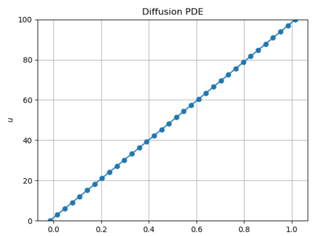

Figure 3: This is diffusion at $0.8t_{max}$.

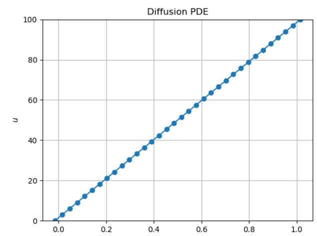

Figure 4: This is diffusion at t_{max}.

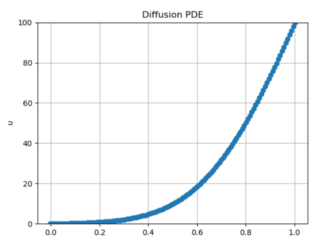

Figure 5: This is diffusion at $0.2t_{max}$.

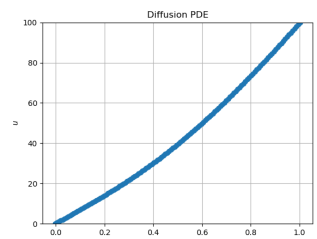

Figure 6: This is diffusion at $0.5t_{max}$.

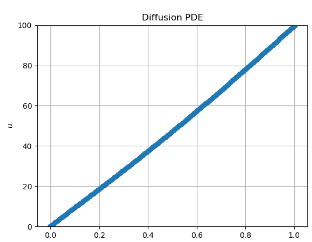

Figure 7: This is diffusion at $0.8t_{max}$.

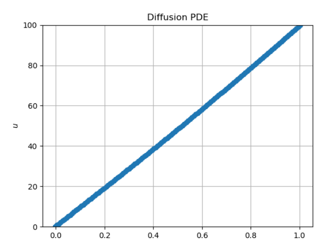

Figure 8: This is diffusion at t_{max}.

(b) The max time for the case $N = 32$ is $t_{max} = 0.70$ while for $N = 128$ the $t_{max} = 0.41$. I believe that the finer grid increases the accuracy and thus reaching a steady state much faster.

(c) If the Δt_{diff} does not satisfy the CFL condition then numerical method will unstable and not solve the equation.

(d) There is discrepancy in the max time found for both grids. The max time for the coarser grid is 0.70 while for the finer grid it was 0.41.

(e) In order to see one period of the advection PDE then we need to run the time to $t_{max} = 1$.

(f) In the following we show the solution of the Advective PDE in with $N = 32$ and $N = 128$ with both the discretization types. Figures 9 and

10 show the solution using the upwind method while Figures 11 and 12.

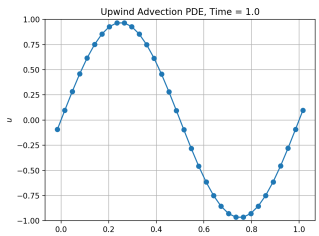

Figure 9: This is advection at $t_{max} = 1$.

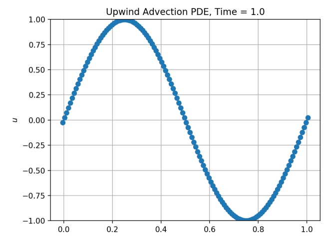

Figure 10: This is advection at $t_{max} = 1$.

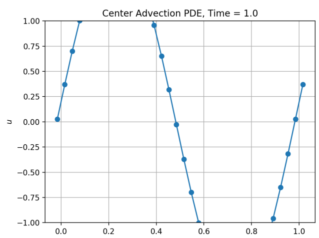

Figure 11: This is diffusion at $t_{max} = 1$.

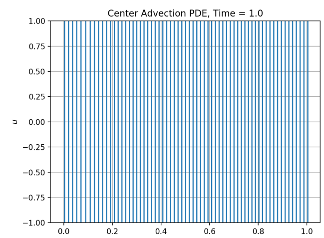

Figure 12: This is advection at $t_{max} = 1$.

(g) As we saw from the previous figures the best method is the upwind discretization. So now we let $Ca = 0.9$ and $Ca = 1.2$ as shown in Figures 13 and 14. We can see that for $Ca = 0.9$ the system loses energy. In contrast, when $Ca = 1.2$ then the system gains energy.

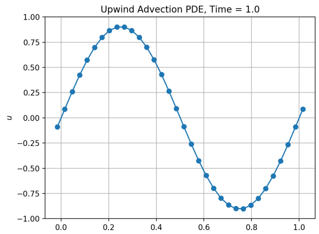

Figure 13: This has a CFL constant of $Ca = 0.9$.

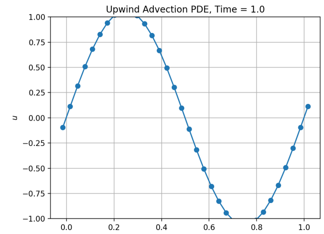

Figure 14: This has a CFL constant of $Ca = 1.2$.

(h) Now we get to solve both the full advection-diffusion PDE. In this case we choose the center discretization method and let $\kappa = 0.0156$. As we can see in Figure 15 the introduction of diffusion helps stabilize the centered discretization scheme.

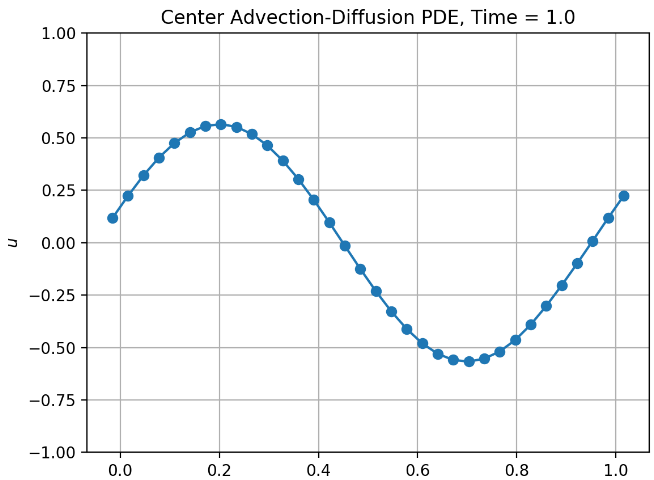

Figure 15: The solution to the advection-diffusion PDE using the centered discretization method.

3 Conclusion

As we can see from the previous results, the Fortran implementations solve the PDEs succesfully. In addition, the Fortran code collects the `pde.init` file and reads the runtime parameters, then collects results and writes text files. The Python script then plots the results. Although I obtain correct solutions, I believe there is some parts of the code that can be optimized or better organized. The biggest issue was automating when to output the text files for the correct time.