Numerical PDE: Linear Advection and Diffusion Equation

Liliya Milenska

Abstract

We implement a Fortran routine to solve the 1D advection-diffusion equation using finite-difference methods (two advection schemes and one diffusion scheme). A Python routine is also implemented and used to initialize the necessary runtime parameters and to organise multiple runs of the Fortran program and finally to plot the results. Three specific cases are studied here: pure diffusion, pure advection and full advection-diffusion.

1 Introduction

We consider the linear advection-diffusion PDE, as defined in Equation (1) below. Here $a$ is a constant advection velocity, and $\kappa \geq 0$ is a constant diffusion coefficient. Throughout this report we shall use $a = 1$ and $\kappa = 1.156 cm^2/s$ (the material diffusivity of copper), unless stated otherwise.

$$u_t + au_x = \kappa u_{xx}$$

Before solving the complete advection-diffusion equation, we consider solving the pure diffusion or the pure advection case separately, by setting either $a$ or $\kappa$ to zero. In order to fully define the problem, we impose an initial condition at $t = 0$ as $u(x, 0) = u_0(x)$, as well as boundary conditions as $u(x_a, t) = g_a(t)$ and $u(x_b, t) = g_b(t)$ for $x_a \leq x \leq x_b$ and for any time $t > 0$. Numerically, we impose the boundary conditions using a ”Ghost-Cell” method, namely by defining them on two exterior points, $x_0 = x_a - \Delta x/2$ and $x_{N+1} = x_b + \Delta x/2$. In these expressions $\Delta x = \frac{x_b-x_a}{N}$, where N is the desired spatial resolution. One last thing to take into account is the so-called CFL condition, which defines the size of the time-step $\Delta t$ and is necessary for the stability and convergence of our methods. This condition can be implemented as: $C_\kappa = \kappa \frac{\Delta x}{\Delta t}$ for the pure diffusion case and $C_a = |a| \frac{\Delta x}{\Delta t}$ for pure advection. In the full advection-diffusion case it can be expressed as $\Delta t = C_a \min(\frac{\Delta x}{|a|}, \frac{\Delta x^2}{2\kappa})$.

2 Fortran implementation

We shall not go over the Fortran implementation in too much detail, but we must mention the main elements. The program structure has three main components: general subroutines defining global variables, subroutines related to advection and subroutines related to diffusion.

The global definition routines are for example read_init_module (to read input parameters), write_data (to write output to a *.dat file), setup_module (to initialize the read parameters) and grid_init (to initialize the spatial grid based on these parameters).

The routines relevant to the diffusion process are: diffuse_init, which defines the initial conditions for diffusion, and diffuse_update, which calculates (using a Forward difference approximation scheme) and saves the solution at each successive time step.

For the pure advection case we have a similar set of subroutines: advect_init and advect_update respectively. Here the only difference here is that there are two implemented methods: the Upwind scheme (defined in upwind.f90) or the Centered scheme (defined in centered.f90). Either method can be chosen by the User through the runtime parameter initialization.

When considering the full advection-diffusion case, the main program calls both the diffusion method and the (chosen) advection method to calculate the solution at each time-step.
3 Python implementation

The Python routine writes the User-defined parameters to a *.init file, runs the Fortran program and plots results. One execution of the Python routine allows for multiple parameter initializations and runs. The structure of the routine is as follows.

A function to compile: The first function takes as input the path to the directory containing the Fortran files, changes the working directory to that path, checks whether the Fortran code has already been compiled (whether there is a *.exe file) and either executes a make clean followed by make, or just make. This compiles the Fortran code.

A function to write: The next two functions work together to write the necessary runtime parameters to a *.init file in the working directory. The program runtimeParameters_init takes the User-input, checks whether a *.init file is not already present (in which case it renames it accordingly), then creates a advect_diff.init file and calls the runtimeParameters_write function (writing each variable on a separate line, preceded by its name).

A function to run: The third function simply runs the created executable.

Two functions to plot: Two functions plot the results. The output structure is motivated by the specific questions we try to answer in this study. First, plot_data generates four plots for each run, displaying the solution at the time steps: 0.2t_{max}, 0.5t_{max}, 0.8t_{max} and t_{max}. The steady state solution is also plotted in the case of diffusion (f(x) = 100x), as well as the initial profile in the case of advection (or advection-diffusion). The second function, plot_data_adv can be called at the end of the program. Its aim is to solve the problem studied in section 4.3 and is not used elsewhere. It displays the solutions of the advection-diffusion equation at t = t_{max} using two spatial resolutions (N = 32, 128), either using the upwind or the centered scheme.

The main function: The main function of the Python routine begins by requesting a long list of User input. It then calls the relevant functions to compile, write the parameters, run the Fortran executable and finally - plot the results. At the end of each run the User may choose to run the routine again.

4 Results

In this section we present the results from specific runs, analyzing different aspects of the solution to the 1D Advection-Diffusion equation.

4.1 Pure Diffusion

We begin by considering the pure diffusion problem by taking a = 0 in equation (1), with κ > 0. The discrete form of the explicit finite difference scheme for the pure diffusion case we use is:

\[ u_{i}^{n+1} = u_{i}^{n} + \kappa \frac{\Delta t}{\Delta x^2} (u_{i+1}^{n} - 2u_{i}^{n} + u_{i-1}^{n}) \] (2)

The initial conditions for the diffusion problem are specified in Equation 3 below.

\[ u_{0}(x) = \begin{cases} 
0^\circ F, & \text{for } 0 \leq x < 1 \\
100^\circ F, & \text{for } x = 1 
\end{cases} \] (3)

The boundary conditions are such that the temperature is held constant at the two end-points of the spatial domain, for any time-step n. These conditions are defined numerically as u_{0}^{0}(x) = 0^\circ F and u_{N+1}^{n} = 100^\circ F.
4.1.1 Maximum time-step

When solving the Advection-Diffusion problem numerically, we must specify a maximum time-step for the solution. In the case of pure diffusion, this $t_{\text{max}}$ is defined as the step at which the material’s temperature reaches a steady state solution (for the defined diffusivity). The can be determined based on the $L_1$ error $E^n$: when it becomes smaller than a certain threshold value $\epsilon$ (input by the User), the maximum time-step has been reached. This condition (Equation (4) below) specifies the moment after which successive time-steps do not cause significant variations of the solution, and it is calculated and verified at each time-step.

$$||E^n||_1 = \Delta x \sum_{n=1}^{N} |u_i^n - u_i^{n-1}| < \epsilon$$  \hspace{1cm} (4)

With the already specified parameters, taking $\epsilon = 10^{-4}$ and considering two spatial resolutions $N = 32, 128$, we have the following maximum time-steps:

$$t_{\text{max}} = \begin{cases} 1665 & \text{for } N = 32 \\ 16176 & \text{for } N = 128 \end{cases}$$  \hspace{1cm} (5)

Furthermore, with the time-step size defined to satisfy the CFL condition, these time-steps correspond to a maximum time for convergence of:

$$T = \begin{cases} 0.703s & \text{for } N = 32 \\ 0.427s & \text{for } N = 128 \end{cases}$$  \hspace{1cm} (6)

Figures 1 and 2 below show the solutions for these two spatial resolutions. The red dots represent the numerical solution and the black line represents the exact steady-state solution.

![Figure 1: Solution to the pure diffusion equation at $t = 0.2t_{\text{max}}, 0.5t_{\text{max}}, 0.8t_{\text{max}}$ and $t_{\text{max}}$ for $N = 32$.](image)

4.1.2 Different grid resolution

There is, naturally, a difference between the two chosen grid resolutions. Indeed, since the definition of $N$ implies spatial step-size $\Delta x$, from the CFL condition this also implies different time-step sizes. This in turn changes the way the solution is calculated at each successive time step, as shown in Equation (2).
We can see that a larger spatial resolution implies a considerably larger amount of time-steps until the steady-state solution is reached. However, since the corresponding time-step size is considerably smaller, the resulting total time until convergence is smaller than the case with a smaller spatial resolution.

### 4.1.3 The CFL condition

We would like to test the solution in the case where the CFL condition is not satisfied. To do this we implement a value for $C_a > 1$ (for example $C_a = 1.2$). As we might expect, the program keeps running without converging. Indeed, recalling the purpose of the CFL condition, it is a necessary condition for convergence of this type of numerical method. Consequently, we cannot obtain any value for the maximum time-step $t_{\text{max}}$.

### 4.2 Pure Advection

Next we consider the pure advection problem by setting $\kappa = 0$ in Equation (1). We can consider two different types of initial conditions: a smooth, continuous, sinusoidal profile defined in Equation (7) and a discontinuous shock defined in Equation (8) below (with $x_0 = 0.3$, the location of the initial discontinuity). In this section we shall focus only on the first case.

$$u_0(x) = \sin(2\pi x)$$

$$u_0(x) = \begin{cases} 
1 & \text{for } x < x_0 \\
-1 & \text{for } x > x_0 
\end{cases}$$

For the smooth sinusoidal profile, the boundary conditions we impose are periodic: $u_0^n = u_N^n$ and $u_{N+1}^n = u_1^n$. For the discontinuous shock wave we define the Dirichlet condition, which is expressed numerically as: $u_0^n = u_1^n$ and $u_{N+1}^n = u_N^n$.

As previously stated, we consider the upwind and the centered schemes. The discrete equations defining those two schemes are given in Equations (9) and (10) respectively.

$$u_{i+1}^n = u_i^n - \alpha \frac{\Delta t}{\Delta x} (u_i^n - u_{i-1}^n)$$
\[ u_i^{n+1} = u_i^n - \frac{\Delta t}{2\Delta x} (u_{i+1}^n - u_{i-1}^n) \]  
(10)

4.2.1 Maximum time-step

To define the maximum time-step, we calculate the time needed by the sine wave to travel one complete period over the defined domain (here \( x_a \) and \( x_b \) specify the initial and final points of the spatial grid): 
\[ t_{\text{max}} = \frac{(x_b - x_a)}{a\Delta t} \]. In our case, with the specified parameters, the maximum time-step is then easily determined as being \( t_{\text{max}} = 1/\Delta t \).

4.2.2 Comparing Upwind and Centered

Here we run the program using our two grid resolution sizes \( N = 32, 128 \), and applying both schemes. The resulting solutions appear in Figure 3 below. We can easily see that the upwind scheme converges well, while the centered scheme leads to instabilities. The red dots represent the numerical solution at \( t = t_{\text{max}} \) and the black line shows the initial temperature profile at \( t = 0 \).

4.2.3 The CFL condition

In the previous section we identified the upwind scheme as being more appropriate for solving the advection problem, leading to a steady solution. Here we shall consider that scheme again, this time imposing a time-step size, which doesn’t satisfy the CFL condition. The result can be seen for \( N = 128 \) in Figure 4 below. We can see that already at \( t = 0.2t_{\text{max}} \) serious instabilities appear in the solution, and these are only amplified as we take further time-steps. This implies that the solution does not return to the initial profile as it did when the CFL condition was satisfied. Here again the CFL condition is a necessary condition for solution convergence. The \( C_a = 0.9 \) case is plotted in Figure 5 to show the CFL-satisfying advection solution (which presents no instabilities and returns to its initial profile).

4.3 1D Advection-Diffusion

Finally, we consider the full advection-diffusion problem. To solve this numerically, we combine our separate advection and diffusion solutions sequentially. This “operator-splitting” technique first solves
Figure 4: Solutions at $t = 0.2t_{\text{max}}, 0.5t_{\text{max}}, 0.8t_{\text{max}}$ and $t_{\text{max}}$ for $N = 128$ and $C_a = 1.2$

the advection part of the problem, using one of the two schemes ($D_x$), resulting in: $u_i^t = u_i^n - a\Delta t D_x u_i^n$. Then we apply the scheme solving the diffusion part, ($D_x^2$), resulting in: $u_i^{n+1} = u_i^n + \kappa \Delta t D_x^2 u_i^n$.

We determine a diffusion coefficient $\kappa > 0$ such that the time-step size is the same for both the advection and the diffusion (with $C_a = 1$) and taking into account the CFL conditions: $\kappa = \frac{a\Delta x}{2}$. Taking the centered scheme (which lead to instabilities before) for the advection part, we plot the results in Figure 6. We can immediately see that the previous instabilities have disappeared, and that the solution evolves steadily with each consecutive time-step. The non-zero diffusion coefficient does indeed help to suppress the numerical instabilities we observed before.

5 Conclusion

We successfully implemented a Fortran and a Python routine, which allowed us to solve different cases of the 1D advection-diffusion equation and reach some interesting observations. First, when analysing pure diffusion processes, we saw that larger spatial resolution values lead to faster convergence. Then, in pure advection, we saw that our two finite-difference schemes were not equally efficient, the centered scheme leading quickly to instabilities in the solution. In both cases, not satisfying the CFL condition lead to unstable and diverging solutions. And finally, considering the full advection-diffusion case, we saw that a non-zero diffusion coefficient was capable of suppressing those instabilities. This implementation could now be used to investigate further configurations, with different initial and boundary conditions, as well as various parameters. In particular, for some examples of the discontinuous initial advection profile, please refer to the link below.\[1\]

\[1\]Liliya Milenska’s personal web page - AMS209 Final Project
Figure 5: Solutions at $t = 0.2t_{\text{max}}, 0.5t_{\text{max}}, 0.8t_{\text{max}}$ and $t_{\text{max}}$ for $N = 128$ and $C_a = 0.9$

Figure 6: Advection-diffusion solutions at $t = 0.2t_{\text{max}}, 0.5t_{\text{max}}, 0.8t_{\text{max}}$ and $t_{\text{max}}$ for $N = 32$ with centered scheme advection.