Theo Wecker

Math Methods Project

November 22, 2019

1. Introduction and Goals

I started my original project, and completed my initial summary, to quickly realize that my project on eigenvalues of random matrices I was planning to do was far too difficult. Luckily, I had started early enough to change topics. My goal of this project is to try to fully cement my understanding of solving partial differential equations (PDEs) and their properties while simultaneously increasing my competence in *Python* and *LaTex*.

I am not familiar, at all, with numerical differential equation solving methods and am fairly new to scientific *Python*, so this is a great learning experience for me. I will be coding some difficult (for me) numerical methods and explaining, in words, points that I cannot fully understand how to code. I will be describing a common method for solving PDEs and will include an example I coded in *Python*.

PDEs are extremely important in physics and frequently appear in a variety of fields including: diffusion, fluid mechanics, Poisson's equation, and wave mechanics with Schrödinger's equation. PDEs are connected with our course, as we have spent the last few weeks learning ways to solve them. The equation I will be looking at is the simplied diffusion equation. This equation could be solved with Green's functions but we can more easily approximate solutions with complex initial conditions and functions with the numerical method described below.

2. Brief Overview of Partial Differential Equations

Partial differential equations are equations which include unknown functions of different (or multiple) variables and their partial derivatives. When PDEs are solved numerically, we need to specify time steps, grid spacing, and initial values similar to numerical solutions of ordinary differential equations. When we are discussing ordinary differential equations and PDEs we generally start off with classification of problems. Most PDEs can be separated into two categories: initial value problems and boundary value problems. PDEs can be numerically solved with several different schemes. I will be starting from one of the most fundamental PDE solution schemes on a simple PDE: the forward time centered space scheme on the heat equation.

3. Modeling the Simplified Diffusion Equation

The heat equation is one of the easier equations to solve and is defined as,

$$\frac{\partial}{\partial t}T(x,t) = \kappa \frac{\partial^2}{\partial x^2}T(x,t) \tag{1}$$

where $T(x,t)$ is the temperature at position x and at time t. The most common way to solve an equation of this type numerically is to use a method called the forward time centered space (FTCS) scheme. This method is very similar to the Euler method which was studied in class (I referred to the notes provided to complete this derivation.) To solve the heat equation we start by discretizing the time and position derivatives. The time derivative is commonly discretized using a forward derivative [2],

$$\frac{\partial T(x,t)}{\partial t} = \frac{T(x_i, t_n + \tau) - T(x_i, t_n)}{\tau} \tag{2}$$

The position derivative is discretized using a centered derivative,

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{T(x_i + h, t_n) - 2T(x_i, t_n) + T(x_i - h, t_n)}{h^2} \tag{3}$$

We can simplify these two expressions and make them usable by a computation algorithm with the definitions [1],

$$t_n = n\tau \quad \text{and} \quad x_i = ih - \frac{L}{2} \tag{4}$$

where i and n are position and time steps. This shorthand allows us to write,

$$T_i^n = T(x_i, t_n) \tag{5}$$

These definitions allow us to write equations 2 and 3, respectively, as,

$$\frac{\partial T(x,t)}{\partial t} = \frac{T_i^{n+1} - T_i^n}{\tau} \tag{6}$$

and

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2} \tag{7}$$

If we plug these equations back into our initial PDE we get,

$$\frac{T_i^{n+1} - T_i^n}{\tau} = \kappa \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2} \tag{8}$$

which we can easily rearrange into an equation which can be computed in a for-loop,

$$T_i^{n+1} = T_i^n + \frac{\kappa\tau}{h^2}(T_{i+1}^n - 2T_i^n + T_{i-1}^n) \tag{9}$$

This discretization method makes the left hand side of the equation only depend on future times, $n+1$, and the right hand side depend on the current time step, n. This reduces the coupled equations into a system of non-linear ordinary differential equations. If entered into a computer, we will then get temperature as an output with time and position steps as inputs.

With this equation, one can now specify their boundary conditions and initial values and compute how diffusion of heat will propagate through a system. For this example, I will be specifying Dirichlet boundary conditions (specifically, homogeneous boundary conditions where the temperature is zero at the boundaries and the temperature's derivatives.) I will also be specifying that there is a delta function spike in the center of the problem's grid space.

To code this problem a pseudo-algorithm can be defined basically as: define initial parameters, use initial conditions and boundary conditions, set up a for-loop with the FTCS equation, and plot the results as a 3D graph and a contour plot.

There are a few limitations of this method though. The accuracy is increased by using smaller time steps and a smaller mesh size but this increases the computational time needed significantly. My simple algorithm could be improved by making the mesh size more highly detailed for where there is more action taking place (i.e. the center where the diffusion of heat is happening) as we learned from Wolfgang Bangerth's talk. I do not have the coding ability to make this but it could be solved like that by using an existing mesh parameterizer in *Python*. Another issue is that this solution is not stable for every value of τ. We can see this by rewriting our FTCS solution as,

$$T_i^{n+1} = T_i^n + \frac{\tau}{2t_\lambda}(T_{i+1}^n - 2T_i^n + T_{i-1}^n) \tag{10}$$

where

$$t_\lambda = \frac{h^2}{2\kappa} \tag{11}$$

This is a useful quantity to define because t_λ is the time it takes for the width of the Gaussian function to increase from 0 to h [2]. From testing I found that when t_λ is less than τ, this solution blows up and is unstable. I believe that this is due to the different amount of error in the temporal and space approximations. The temporal difference method is accurate to the first order and the space difference method is accurate to the second order [1]. I believe a way to fix this instability would be to make a new method that would allow for both time and position differences to have the same order of error. A way to improve this program would be to add a stability check using Von Neumann stability analysis which uses an algebraic criterion to determine stability of solutions [3]. However, the straightforward FTCS method is still valid and is very accurate for small time steps. I added a code check that automatically looks for stable or unstable solutions by comparing τ to t_λ.

4. Computational Analysis of Heat Diffusion with Python

I wrote a Python code to solve the heat equation with the FTCS scheme. I made a plot of temperature as a function of position and time. I used time steps of 10 microseconds and a grid of 120 points with a system length of $L = 1$. I also defined the diffusion coefficient to be $\kappa = 2$ (the mass diffusivity or diffusion coefficient depends on a system's physical characteristics.) The initial conditions are that temperature is zero everywhere (homogeneous) except at the center where there is a delta function.

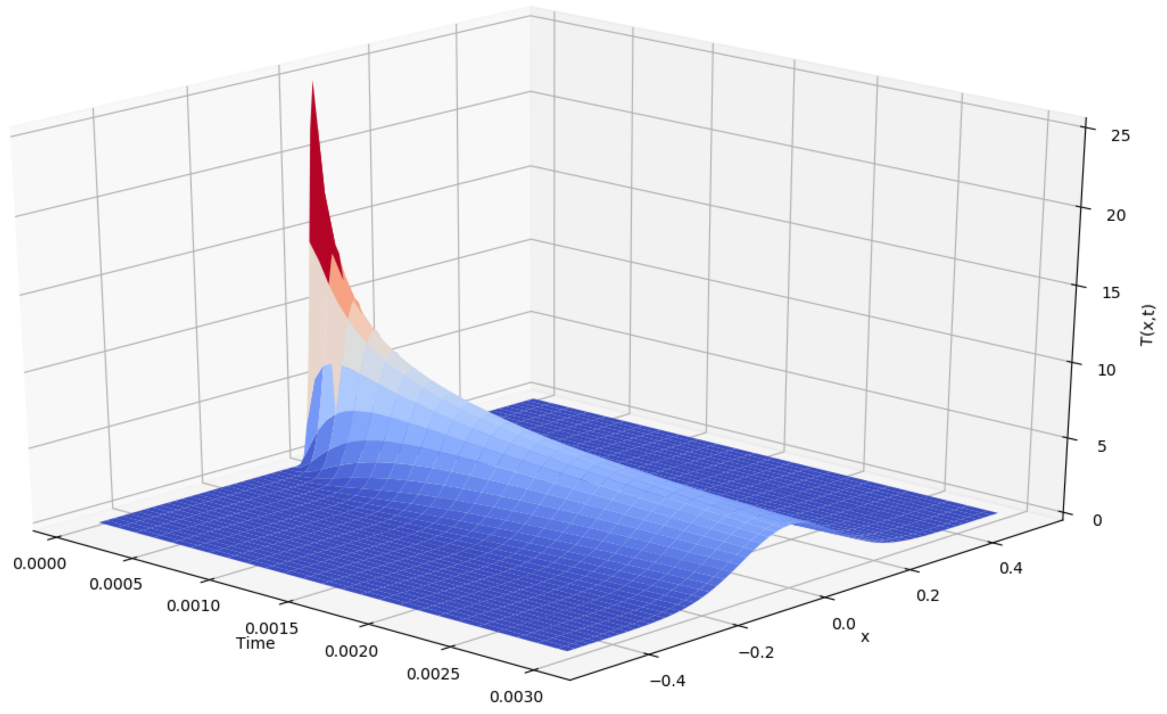

Figure 1: The diffusion equation of a delta function spike in a system.

The warmer colors (or height) in the graph represent hotter temperatures. As we can see from the plot, the delta function spike spreads out over time, as it should. This visualization of diffusion could be used in a variety of contexts: we could model the diffusion of heat in a solid, we could examine wave mechanics of matter, or we could look at several transport processes in gasses. This visualization helps tie in the material we learned about solving numerical ordinary differential equations with our recent study of solution of PDEs. This method could be applied to problems with complex initial or boundary conditions that cannot be solved analytically like the ones we have solved in class. It can also be used as a teaching tool to show a visualization of the solution of PDEs.

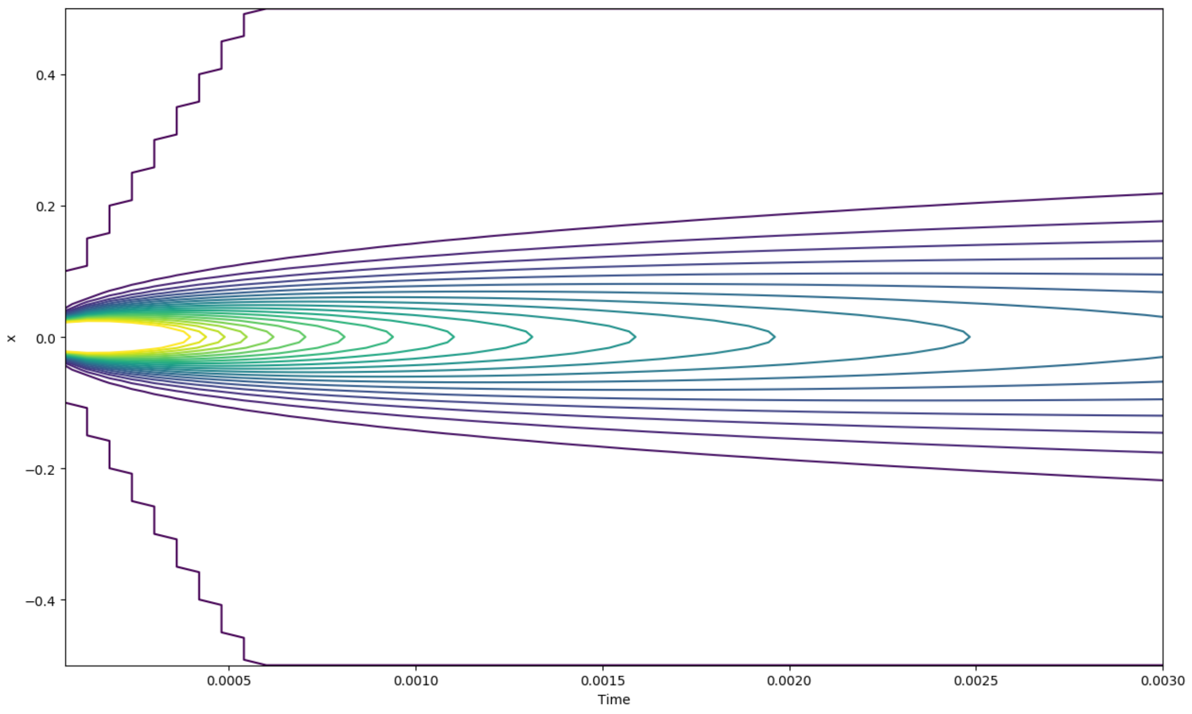

Figure 2: A temperature contour plot showing the propagation of heat from left to right in the system.

I also made a contour plot of the same data. This allows us to more easily see the gradient of the diffusion over time. This might be useful for seeing how fast heat dissipates in an device to guarantee that heat is dissipated before it reaches a critical heat sensitive component in that device. It's easy to see that the data at the edges of the simulation do not make physical sense. This graph illustrates another limitation in the code. The contour plot shows that the temperature at the boundaries is not computed with the code and that data should be neglected.

As I described earlier in this paper, we can also look at an unstable solution made with a new diffusion coefficient of $\kappa' = 20$ instead of the previous value of $\kappa = 2$.

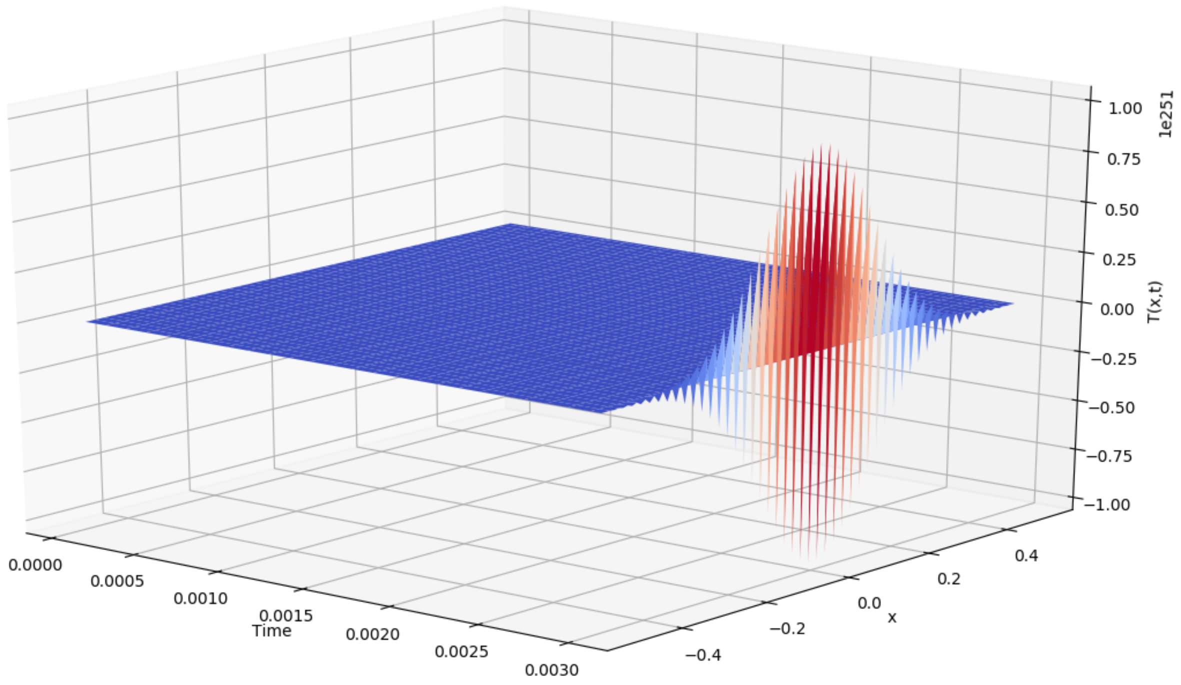

Figure 3: A plot of temperature versus position and time showing the instability that can occur if a $t_\lambda < \tau$ coefficient is used.

We can easily see that this graph makes no physical sense. The temperature axis has units on the scale of 10^{251} and there is no diffusion along the time axis. There is a strange pattern on the $t = 0$ axis which would require further study to understand. It seems that when the time scale is too large or the grid is too small that the temperature does not diffuse through medium in this simulation.

5. Conclusion

I did not think it was possible to separate a second order PDE into a system of linear ordinary differential equations that could be solved with a method similar to Euler's method. I was extremely surprised that a PDE could be written in a familiar looking manner and solved with forward and centered differences. I thought that I would be able to cover more than one method for solving partial differential equations but it turns out that even one of the simplest requires pages of explanations. An advantage of the FTCS scheme turns out to be its disadvantage: it is simple. This simplicity allows this solver to run, even with thousands of time steps, in a matter of seconds. Yet, the designer of the code has to be very careful with their constants and conditions. The biggest way my code could be improved is to allow for a larger variety of initial and boundary conditions and to make it easier to input different source functions, as I only tested a delta function.

There are so many applications of solving PDEs in physics that it is hard to understate the importance of being able to quickly model and solve the many unique differential equations with a variety different initial and boundary conditions. In this project, I discovered the complexity of solving PDEs numerically and the attention to detail required to find solutions that make physical sense.

References

[1] Steven Chapra, and Raymond Canale. *Numerical Methods for Engineers*. McGraw-Hill, p. 840-848, 2002.

[2] Santosh Gupta. *Numerical Methods for Engineers*. New Academic Science, p. 283-286, 2014.

[3] Eugene Isaacson, and Herbert Bishop Keller. *Analysis of Numerical Methods*. Courier Corporation, p. 523, 2012.