Partial Differential Equations

—

Parabolic: Diffusion Equation

Keegan W$^{\text{m.}}$ Karbach

April 13, 2020

Introduction

Partial differential equations are multivariate extensions of the ordinary differential equations explored in previous projects. That is, they are equations that relate rates of change with respect to two or more variables. These equations form the foundation upon which many physical systems can be modeled. Most of the physical phenomena studied in physics can be expressed using systems of these equations. Fluid mechanics is described using the Navier-Stokes equations, electricity and magnetism using Maxwell's equations, and general relativity using Einstein's field equations. Clearly, the application of numerical methods to solve partial differential is of paramount value.

There is, however, more difficulty in the implementation of these applications than there is in the implementation of the ordinary differential equation methods that we have explored previously. Unlike ordinary differential equations which center around the fundamental existence and uniqueness theorems, partial differential equations have no such unified theory. Instead, we will require different approaches for each classification of equation. Partial differential equations are classified into three groups based on the equation

$$a\frac{\partial^2 A}{\partial x^2} = b\frac{\partial^2 A}{\partial x \partial y} + c\frac{\partial^2 A}{\partial y^2} + d\frac{\partial A}{\partial x} + e\frac{\partial A}{\partial y} + fA + g = 0$$

Where $A(x, y)$ is a function of two independent variables and the discriminant $\Delta = b^2 - 4ac$. The partial differential equation is classified as

$$\text{Parabolic for } \Delta = 0$$
$$\text{Hyperbolic for } \Delta > 0$$
$$\text{Elliptic for } \Delta < 0$$

In this project, we will examine parabolic equations, namely the heat equation.

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

These type of equations generally are set up as initial value problems and thus are defined over a finite physical domain and a time domain ranging from zero to infinity. They are numerically solved using 'marching' methods – finite differences are used to solve the temperature distribution

at each time step using the previous time step. We will primarily use a forward time, centered space scheme (shown below), refining it to the DuFort-Frankel scheme in the final problem.

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

The above equation uses a forward derivative for the time derivative and a centered derivative for the space derivative. It is implemented by solving for T_i^{n+1}. The physical system examined in this project represents a rod, modeled as one-dimensional, with a length L and a heat-diffusion coefficient κ. The rod is heated at a point, x_0, giving us an initial condition of a delta function $T(x, 0) = \delta(x - x_0)$. Three boundary conditions are examined

(i) Dirichlet: prescribing temperature values at the endpoints of the rod

$$T_1^n = T_N^n = 0$$

(ii) Neumann: prescribing temperature flux values at the endpoints of the rod

$$\left.\frac{\partial T}{\partial x}\right|_{x=-L/2} = \left.\frac{\partial T}{\partial x}\right|_{x=L/2} = 0$$

(iii) Periodic: setting both the flux and the value of both endpoints equal to each other

$$T_1^n = T_N^n$$
$$\left.\frac{\partial T}{\partial x}\right|_{x=-L/2} = \left.\frac{\partial T}{\partial x}\right|_{x=L/2}$$

1 Problem 6.1

Show that Equation (6.11) is a solution to the diffusion equation.

For reference, Equation 6.11 from Garcia is

$$T_G(x, t) = \frac{1}{\sigma(t)\sqrt{2\pi}} \exp\left[\frac{-(x - x_0)^2}{2\sigma^2(t)}\right]$$

and the diffusion equation is

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

We want to show that the Gaussian function is a solution to the diffusion equation. First, we must find the first partial derivative of T_G with respect to time:

$$\begin{aligned}\frac{\partial}{\partial t} T(x, t) &= \frac{\partial}{\partial t}\left(\frac{1}{\sigma(t)\sqrt{2\pi}} \exp\left[\frac{-(x - x_0)^2}{2\sigma^2(t)}\right]\right) \\ &= \frac{1}{\sqrt{2\pi}} \frac{d\sigma}{dt} \exp\left[-\frac{(x - x_0)^2}{2\sigma^2(t)}\right]\left(\frac{(x - x_0)^2}{\sigma^4(t)} - \frac{1}{\sigma^2(t)}\right)\end{aligned}$$

Now we need the second partial derivative with respect to time:

$$\frac{\partial^2}{\partial x^2}T(x,t) = \frac{\partial^2}{\partial x^2}\left(\frac{1}{\sigma(t)\sqrt{2\pi}}\exp\left[\frac{-(x-x_0)^2}{2\sigma^2(t)}\right]\right)$$
$$= \frac{1}{\sqrt{2\pi}}\exp\left[-\frac{(x-x_0)^2}{2\sigma^2(t)}\right]\left(\frac{(x-x_0)^2}{\sigma^5(t)} - \frac{1}{\sigma^3(t)}\right)$$

Using $\sigma(t) = \sqrt{2\kappa t}$ and $\sigma'(t) = \kappa/\sqrt{2\kappa t} = \kappa/\sigma(t)$, we can now return to the diffusion equation:

Proof.

$$\frac{\partial T_G}{\partial t} \stackrel{?}{=} \kappa\frac{\partial^2 T_G}{\partial x^2}$$
$$\frac{1}{\sqrt{2\pi}}\frac{d\sigma}{dt}\exp\left[-\frac{(x-x_0)^2}{2\sigma^2(t)}\right]\left(\frac{(x-x_0)^2}{\sigma^4(t)} - \frac{1}{\sigma^2(t)}\right) \stackrel{?}{=} \kappa\left(\frac{1}{\sqrt{2\pi}}\exp\left[-\frac{(x-x_0)^2}{2\sigma^2(t)}\right]\left(\frac{(x-x_0)^2}{\sigma^5(t)} - \frac{1}{\sigma^3(t)}\right)\right)$$
$$\frac{1}{\sqrt{2\pi}}(\kappa)\exp\left[-\frac{(x-x_0)^2}{2\sigma^2(t)}\right]\left(\frac{(x-x_0)^2}{\sigma^5(t)} - \frac{1}{\sigma^3(t)}\right) \stackrel{\checkmark}{=} \kappa\left(\frac{1}{\sqrt{2\pi}}\exp\left[-\frac{(x-x_0)^2}{2\sigma^2(t)}\right]\left(\frac{(x-x_0)^2}{\sigma^5(t)} - \frac{1}{\sigma^3(t)}\right)\right)$$
$$\text{RHS} = \text{LHS}$$
$$\therefore T_G = \frac{1}{\sqrt{2\pi}\sigma(t)}\exp\left[\frac{-(x-x_0)^2}{2\sigma^2(t)}\right] \text{ is a solution to the diffusion equation.}$$

□

Note: Please see Appendix B for the intermediate steps not shown above.

2 Problem 6.3

Run the `dftcs` *program with a variety of values for N and τ*

The `dftcs` program, as initially designed, utilizes Dirichlet boundary conditions. The operational array (`tt[]`) is initialized to zeros and then, during the simulation, the array is updated for all interior points using the scheme:

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

Which is translated into Python as

```
tt[1:(N-1)] = (tt[1:(N-1)] +
               coeff*(tt[2:N] + tt[0:(N-2)] - 2*tt[1:(N-1)]))
```

As we are not overwriting the first and last point of the array, they remain zero throughout the iterations. Therefore we have Dirichlet boundary conditions where

$$T_1^n = T_N^n = 0$$

For the first part of the problem, N is held at 61 for a step size of $h = L/(N-1) = .01\bar{6}$ units. Computational runs were taken for $\tau = \{1, 5, 10, 50, 100\} \times 10^{-5}$ and the results are shown below.

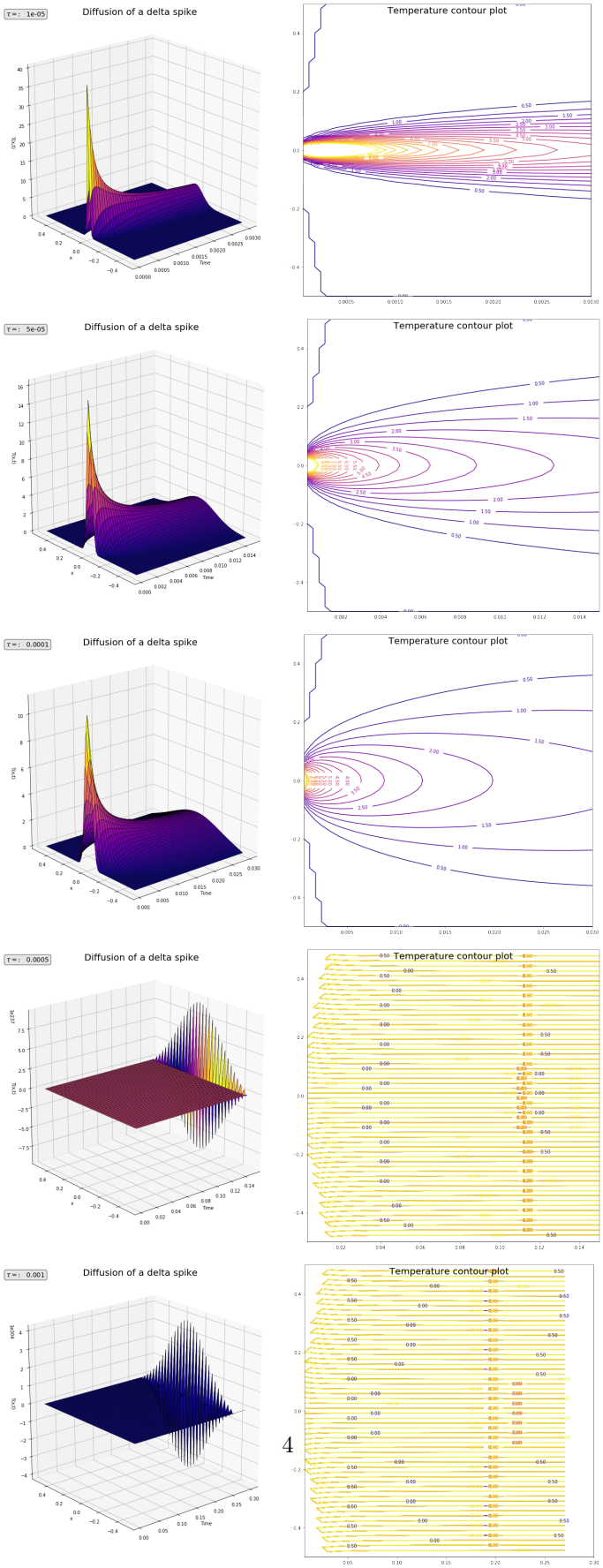
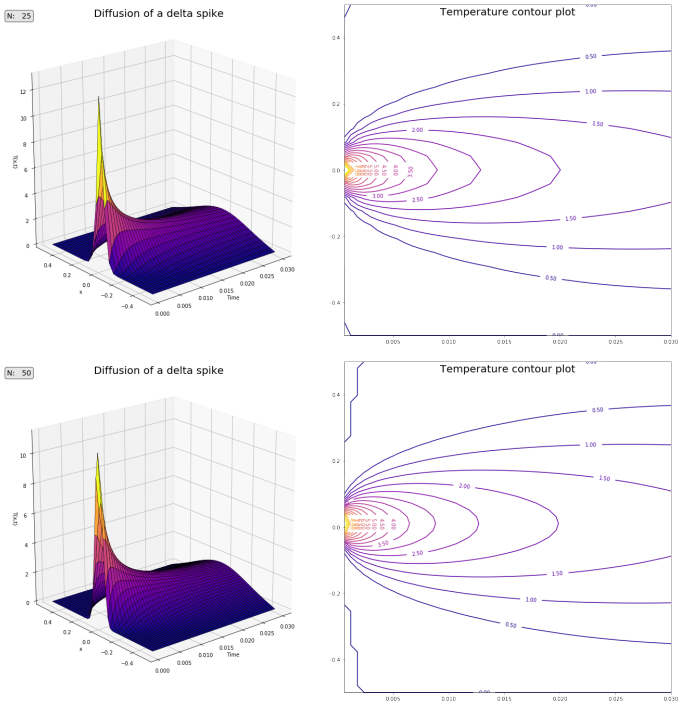


Figure 1: Heat diffusion across a variety of time steps.

Note that when the time step is larger than 1×10^{-4} the solution is unstable. This is due to the given stability criterion of $\tau < h^2/2\kappa$ for a stable solution. Defining L and κ both equal to one and holding $N = 61 \rightarrow h^2/2 = 1.3\bar{8} \times 10^{-4}$ which gives us an upper bound for τ to obtain a stable solution. In the fourth and fifth image, much like the Euler method, the errors of the numerical scheme are magnified at each iteration and instead of converging, the solution diverges. This effect can be seen on the vertical axis limits as the increase hundreds of orders of magnitude from the convergent solutions. The shape of the function *appears* to spread out as the time step increases, but this is due to the fact that we have a longer time domain with larger time steps using the same number of iterations – we are seeing the energy disperse further across the physical domain (notice the order of magnitude difference in the range of time values between the first and third image). Somewhat counter-intuitively, the computation time increases as the time step decreases. This effect is due to the function that handles the contour plot; due to the error magnification, the contour plot takes much longer to determine where to place the contours. When omitting this function, there is very little difference in computation time over all the runs as there is no change in the number of iterations taken, and thus the number of operations performed is the same.

For the second part of this problem, τ is held at 1.4×10^{-4} while computational runs were taken for $N = \{25, 50, 65, 70, 75\}$. I did change these values from the prescribed values to better explore the numerical range around which the solution goes from stable to unstable.



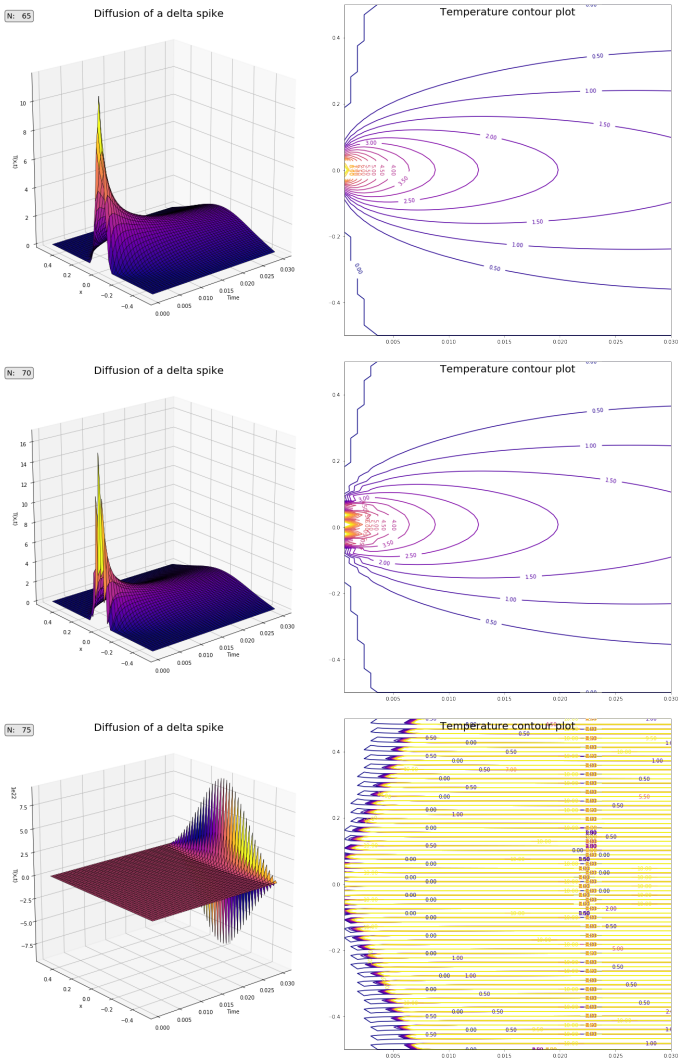


Figure 2: Heat diffusion across a variety of spatial steps.

The stability criterion here is $\tau < t_\sigma$ where $t_\sigma = h^2/2\kappa \rightarrow h = .014$ and thus we require $N < 72$ for a stable solution. Notice that the fourth image ($N = 70$) is beginning to show numerical artifacts at the lower end of the temporal range, despite its convergence and the fifth image ($N = 75$) is divergent, as expected. This divergent solution, in contrast to the corresponding solutions from the first part of this problem, only magnifies the error by 21 orders of magnitude due to its numerical proximity to the stability bifurcation point. As the time step is not changing, the temporal range of the simulation does not change over the five runs – the width of the contour plots is identical for convergent solutions. However, it can be observed that the 'smoothness' of the contours does increase as N increases due to the increasing fineness of the spatial discretization. Computation time increases again over the computational runs, but as above, this increase is almost non-existent when omitting the plotting functions.

3 Problem 6.6

Modify the `dftcs` *program to implement Neumann boundary conditions:*

$$\left.\frac{\partial T}{\partial x}\right|_{x=-L/2} = \left.\frac{\partial T}{\partial x}\right|_{x=L/2} = 0$$

(a) *Method of Images solution for* $T(x,0) = \delta(x)$

The Method of Images solution is given by

$$T_G(x,t) = \sum_{n=-\infty}^{\infty} \frac{1}{\sigma(t)\sqrt{2\pi}} e^{-\frac{(x-nL)^2}{2\sigma^2(t)}}$$

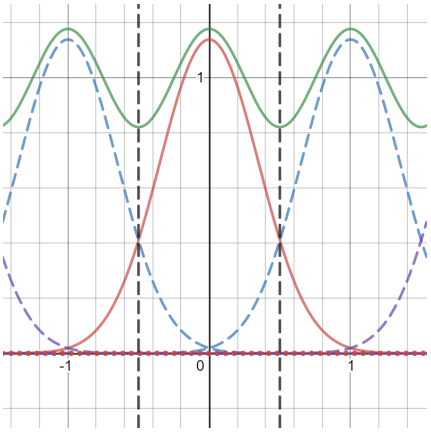


Figure 3: This figure illustrates the method of images solution for Neumann boundary conditions with a centered initial condition. The solid red line represents the physical distribution of temperatures while the dashed lines represent the images. The solid green line is the sum solution from above, illustrating zero flux at the boundaries.

(b) *Method of Images solution for* $T(x,0) = \delta(x - L/4)$

$$T_G(x,t) = \sum_{n=-\infty}^{\infty} \frac{1}{\sigma(t)\sqrt{2\pi}} e^{-\frac{(x-((-1)^n(L/4)+nL))^2}{2\sigma^2(t)}}$$

(c) *Computational comparison to parts (a) and (b)*

The implementation of Neumann boundary conditions requires the modification of the `dftcs` routine by the addition of two lines of code.

```
tt[1:(N-1)] = (tt[1:(N-1)] +
  coeff*(tt[2:N] + tt[0:(N-2)] - 2*tt[1:(N-1)]))

# Neumann Boundary Conditions
tt[0] = tt[1]
tt[-1] = tt[-2]
```

Note here that we use the same initial approach as we do with Dirichlet boundary conditions: we initialize the entire array to zero and (in the first line of the algorithm) we

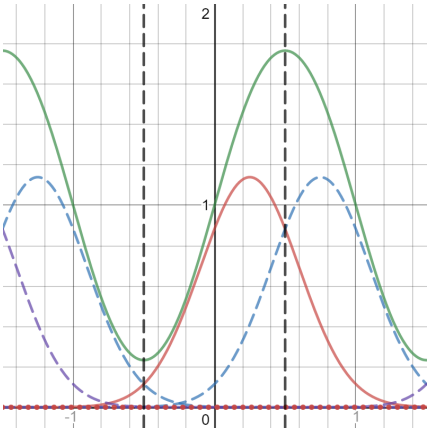


Figure 4: This figure illustrates the method of images solution for Neumann boundary conditions with an offset initial condition. The solid red line represents the physical distribution of temperatures while the dashed lines represent the images. The solid green line is the sum solution from above, illustrating zero flux at the boundaries.

update all the points except for the boundaries. While this maintains the boundaries at zero for the Dirichlet conditions, we need to update them for Neumann conditions so we set the first and last point of the array to the second and second to last points of the array, in effect forcing zero slope at the boundaries.

The figure below contains the results of two computational runs, both using $\tau = 1 \times 10^{-4}$ and $N = 62$. The first image is of the centered initial condition, while the second is of the offset initial condition.

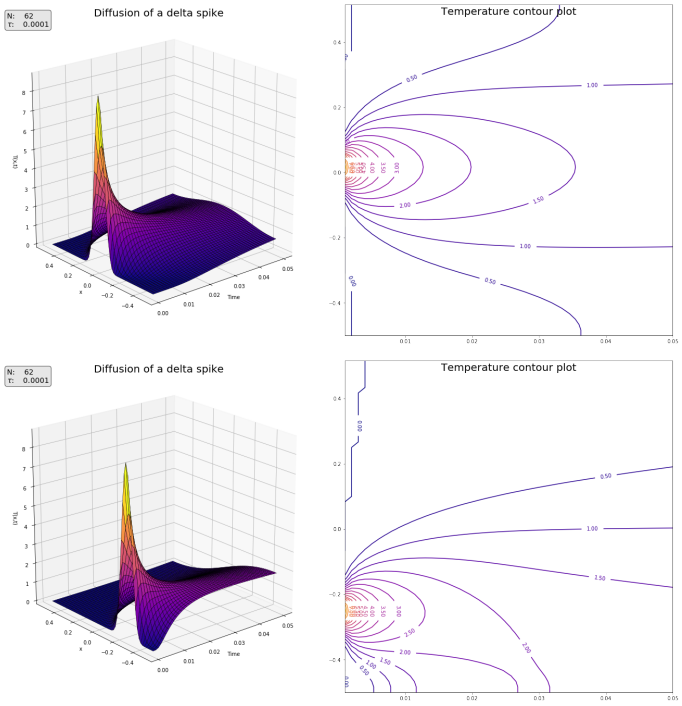


Figure 5: Neumann boundary conditions

(d) *Explanation of spatial discretization*

When calculating the flux, we need to find the slope between two points in our spatial discretization, thus we need an additional two points - one for each boundary point - which leads us to $h = L/(N-2)$. Additionally, for each boundary, we need one point inside and one point outside of the physical domain to calculate the flux across the boundary meaning that our first spatial point will lie $h/2$ outside of the first boundary. Setting that boundary to the value $-L/2$ gives us the equation for spatial discretization $x_i = (i - 3/2)h - L/2$.

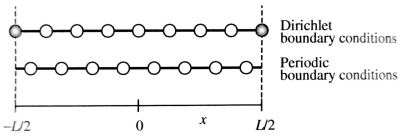


Figure 6: Here we can see graphically what has been explained above. Note that there are two points not shown that lie outside the physical domain.[1]

4 Problem 6.7

Modify the `dftcs` *program to implement Neumann boundary conditions:*

$$\left.\frac{\partial T}{\partial x}\right|_{x=-L/2} = \left.\frac{\partial T}{\partial x}\right|_{x=L/2} = 0$$

(a) *Method of Images solution for* $T(x,0) = \delta(x)$

The Method of Images solution is given by

$$T_G(x,t) = \sum_{n=-\infty}^{\infty} \frac{1}{\sigma(t)\sqrt{2\pi}} e^{-\frac{(x-nL)^2}{2\sigma^2(t)}}$$

(b) *Method of Images solution for* $T(x,0) = \delta(x - L/4)$

$$T_G(x,t) = \sum_{n=-\infty}^{\infty} \frac{1}{\sigma(t)\sqrt{2\pi}} e^{-\frac{(x-((L/4)+nL))^2}{2\sigma^2(t)}}$$

(c) *Computational comparison to parts (a) and (b)*

To implement periodic boundary conditions, the `dftcs` routine is modified from the original program by the addition of two lines of code.

```
tt[1:(N-1)] = (tt[1:(N-1)] +
  coeff*(tt[2:N] + tt[0:(N-2)] - 2*tt[1:(N-1)]))

# Periodic Boundary Conditions
tt[0] = tt[-2]
tt[-1] = tt[1]
```

[1] Garcia pg 218

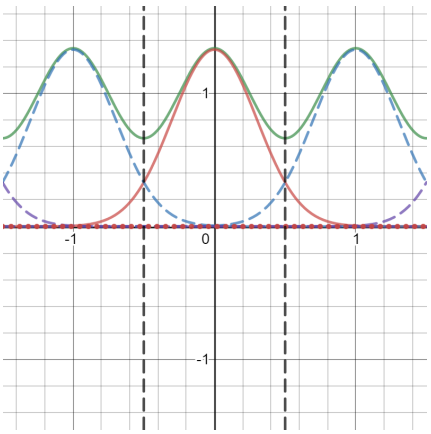


Figure 7: This figure illustrates the method of images solution for periodic boundary conditions with a centered initial condition. The solid red line represents the physical distribution of temperatures while the dashed lines represent the images. The solid green line is the sum solution from above, illustrating equal value and flux at the boundaries. Note that this solution is equivalent to the solution for a centered initial condition with Neumann boundary conditions.

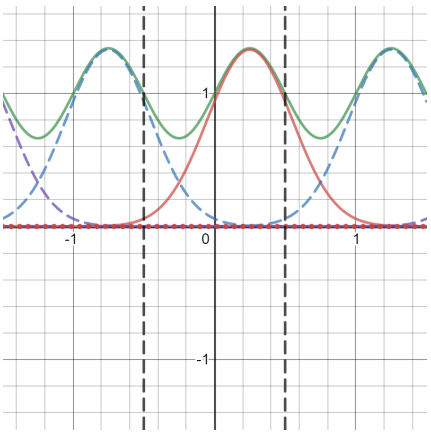


Figure 8: This figure illustrates the method of images solution for periodic boundary conditions with an offset initial condition. The solid red line represents the physical distribution of temperatures while the dashed lines represent the images. The solid green line is the sum solution from above, illustrating equal value and flux at the boundaries.

Note here that we use the same initial approach as we do with Dirichlet boundary conditions: we initialize the entire array to zero and (in the first line of the algorithm) we update all the points except for the boundaries. While this maintains the boundaries at zero for the Dirichlet conditions, we need to update them for periodic conditions, setting both the values and fluxes of each boundary point equal to each other. To do this, we 'knit' the first two points and the last two points, effectively forcing both the value and slope at the endpoints to be equivalent.

The figure below contains the results of two computational runs, both using $\tau = 1 \times 10^{-4}$ and $N = 62$. The first image is of the centered initial condition, while the second is of the offset initial condition.

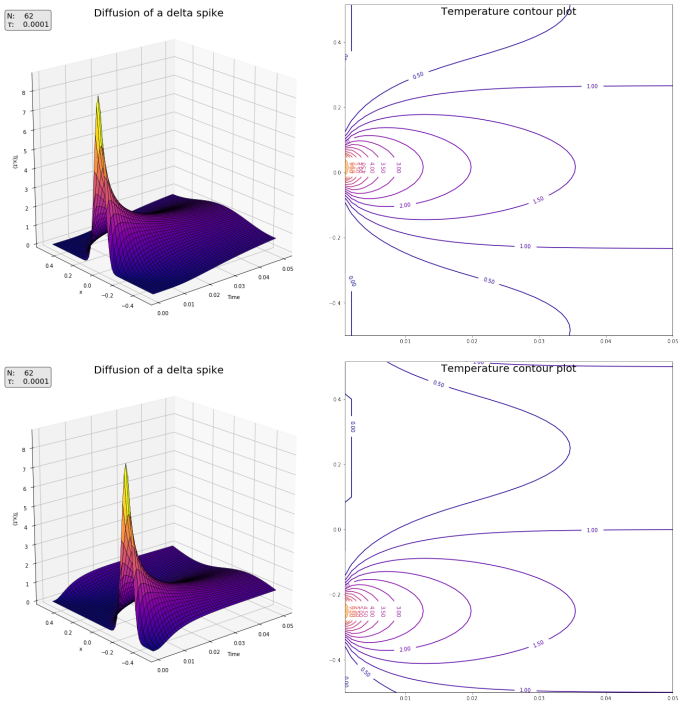


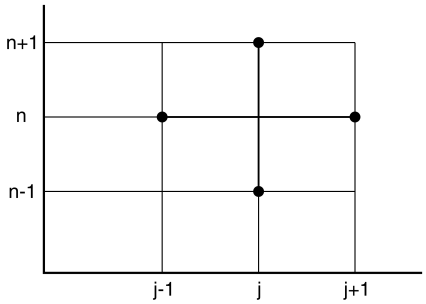
Figure 9: Periodic boundary conditions

(d) *Explanation of spatial discretization*

Much like with the Neumann boundary conditions, we need to calculate the flux at the boundary points. Therefore we have to set up an identical spatial discretization as we did before.

5 Problem 6.9

Implement the DuFort-Frankel scheme



The DuFort-Frankel scheme is a three-level scheme in time. Applying finite differences gives us

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

Which can be solved for T_i^{n+1} lending

$$T_i^{n+1} = \frac{2\alpha}{2\alpha+1}(T_{i+1}^n + T_{i-1}^n) + \frac{2\alpha - 1}{2\alpha + 1} T_i^{n-1}$$

where $\alpha = \kappa\tau/h^2$.

This can be translated into the Python code

```
tt[1:(N-1),istep] = ((1-2*coeff)/(1+2*coeff) * tt[1:(N-1),istep-2] +
                        (2*coeff)/(1+2*coeff) * (tt[2:N,istep-1] +
                                                 tt[0:(N-2),istep-1]))
```

Note that, due to the fact that this scheme requires data from both $t = n$ and $t = n-1$, we must have both the initial condition ($T(x,0)$) and a first iteration ($T(x,\tau)$) to utilize this algorithm. In this implementation, we have used the FTCS routine to calculate the first step from the given initial condition prior to using the DuFort-Frankel scheme for the remaining iterations.

Using this method and a variety of values for N and τ, the following results were obtained.

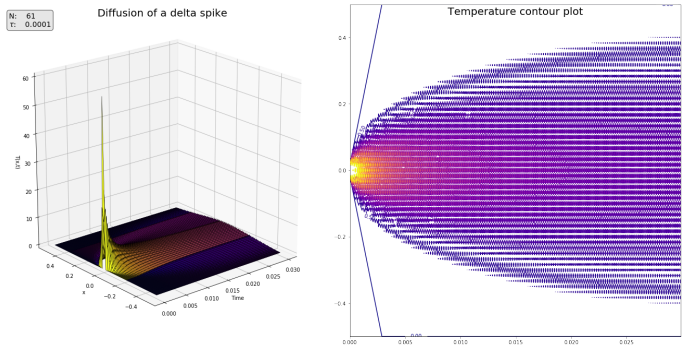


Figure 10: DuFort-Frankel method for $(N, \tau) = (61, 1\text{e}-4)$

Notice that the solution does converge, the method is stable, but there are systemic numerical artifacts present that limit the predictive power of the model.

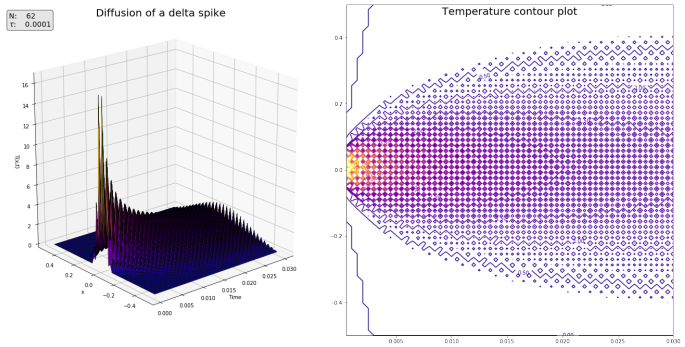


Figure 11: DuFort-Frankel method for $(N, \tau) = (62, 1\text{e}-4)$

Here we have increased the quantity N by one, notice that the numerical artifacts are even more pronounced. The method is still stable, as the solution converges despite the artifacts.

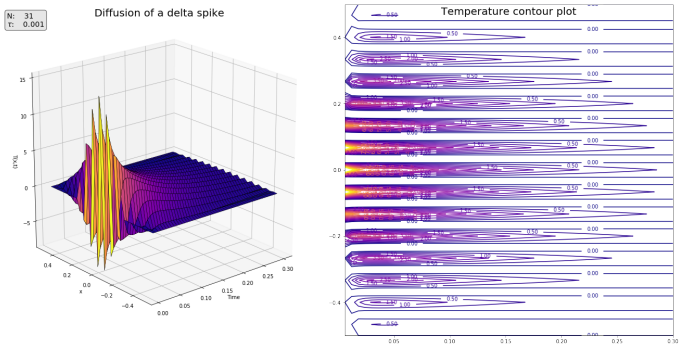


Figure 12: DuFort-Frankel method for $(N, \tau) = (31, 1\text{e} - 3)$

Here we can see the most pronounced evidence of numerical artifacts, they appear to become worse as N and τ decrease.

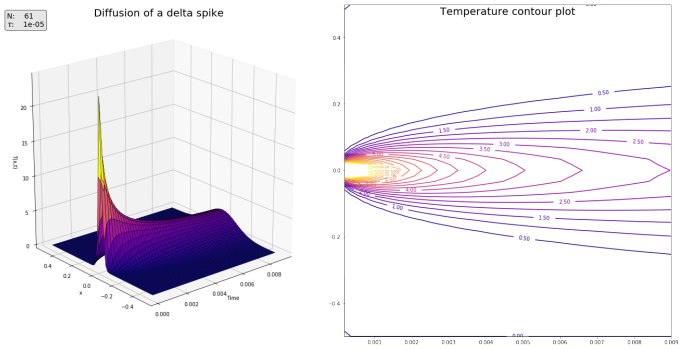


Figure 13: DuFort-Frankel method for $(N, \tau) = (61, 1\text{e} - 5)$

Finally, with a small value for τ, we see results that are analogous to the FTCS scheme that was implemented earlier, sans artifacts.

Conclusion:

In this project we developed the FTCS scheme to numerically solve the heat equation. This 'marching' method involves using one previous time step to calculate the current time step, evolving the given temperature distribution over time. We modeled a one-dimensional rod with diffusion constant κ and length L using a simulated delta function as an initial condition, $T_0 = \delta(x - x_0) \rightarrow T_0 = 1/h$ at x_0. The solution set for the heat equation takes the form of a normalized Gaussian with a standard deviation $(\sigma(t) = \sqrt{2\kappa\tau})$ that increases with time, leading to a temperature distribution that spreads over the physical domain as time increases.

We then explored the three prototypical boundary conditions: Dirichlet, where temperature values are prescribed at the boundaries, Neumann, where temperature fluxes are prescribed at the boundaries, and periodic, where the values and fluxes of the two boundaries are set equal to each other. This analysis was quite enjoyable as we first used the method of images to find analytical solutions to the heat equation, and subsequently modified our FTCS routine to implement these

boundary conditions. Much like we learned in the Monte Carlo integration project, it is often the case that difficult analytical problems are much more easily solved using numerical techniques.

We took the briefest of looks at stability analysis; I, personally, spent quite a while looking over von Neumann stability analysis – there will be much deeper analysis of stability in future projects. Here, we saw that our FTCS routine was unstable for $\tau > h^2/2\kappa$. At this bifurcation point, the errors inherent in our numerical method become magnified and the solution, as opposed to converging to an equilibrium as $t \to \infty$, diverges.

Finally, we refined our FTCS scheme to the DuFort-Frankel scheme. This scheme, centered in both time and space, is a three-level scheme: it uses the two previous time steps to compute the current time step. In order to start this scheme, we needed to use the FTCS routine on the first time step, calculating off of the initial condition. The implementation of these routines were relatively straightforward, even with the more complicated DuFort-Frankel method. In fact, the main coding struggle that was encountered in this project came down to how Python handles 3D plots, although that turned out to be relatively straightforward as well as my plotting package is very well documented online. Given more time, I would have done more error analysis of the methods run, although all of my experience in von Neumann error analysis has been done on matrix methods which seemed slightly different than what I read in chapter 9 of Garcia.

Appendices

A: Code

B: Problem 6.1 pencil work

References

[1] Alejando L Garcia. *Numerical methods for physics*. 2015. ISBN 9781514136683.

[2] Leif Rune Hellevik. Numerical methods for engineers, Jan 2020. URL `http://folk.ntnu.no/leifh/teaching/tkt4140/._main067.html`.

[3] John Douglas Moore. Introduction to partial differential equations, May 2003. URL `http://web.math.ucsb.edu/~moore/pde.pdf`.

[4] Tim Sauer. *Numerical analysis*. Pearson, 2nd ed edition, 2012. ISBN 9780321783677.