Finite Difference Schemes and Exact Solution for Multi-layer Heat Diffusion through Several Media

Stefan Cline

27 April 2022

1 Introduction

The following paper aims to accomplish two goals. For the first, it focuses on the techniques developed by R.I. Hickson, S.I. Barry, G.N. Mercer, and H.S. Sidhu for both exact solutions to the heat equation for perfectly thermally coupled layers of different mediums, and their corresponding finite difference schemes (FDS) used. Second, it provides a thoroughly worked out example that aims to highlight both the effectiveness and flexibility of the technique (measured by order of accuracy and $L2$ norm computations).

The problem statement that is generically considered by the authors is that of a multilayered diffusion equation. This expands on the commonly solved single layer parabolic heat diffusion equations that are commonly found in introductory texts on partial differential equations (PDEs). Here, the addition of an unspecified number of boundary layers (simulating a change in physical material) adds a rich complexity both physically and numerically. The paper explores three separate types of problems: matching diffusivities (i.e. perfect thermal contact), matching conductivities, and jump condition matching. The first two problems are quite similar, as such, only the first type is explored in this paper. The worked through example considers three layers to highlight the complexity of the exact technique, but to also keep the example simple enough to be followed by the reader. After working through the exact solution, a breakdown of the FDS is provided, and the code used to solve both the exact and the FDS will be provided in the appendix.

The motivation for this paper is rooted in my interest and prior experience in the semiconductor industry. Microchips and microprocessors are sensitive to heat distributions both while in use, and during the Extreme Ultra-Violet (EUV) and Deep Ultra-Violet (DUV) fabrication processes, thereby making the study of heat diffusion through a multilayered substance a practical industry concern[3].

Lastly, a note is made that this paper will utilize the term, "main article", which refers to "Finite difference schemes for multilayer diffusion" by the above mentioned authors. This is done instead of citing it formally throughout as this would quickly become excessive[1].

2 Determining Exact Solution for Three Boundary Layers

The authors in their paper showed the FDS and exact results of their final solutions and error calculations, but didn't provide those for us[1]. Hence, we need to develop and solve our own system to be able to verify if their technique does in fact work. As such, this section is a self-derived problem that is solved using the authors' techniques[1][2]. Thus, the following is the self-derived three layer boundary problem where we assume perfect thermal contact (Figure 1). The following are defined as: $D_i \equiv$ diffusion coefficients, $l_i \equiv$ the layer length, and $U_i \equiv$ the solution for each layer. We also start with an initial condition of a piece wise function:

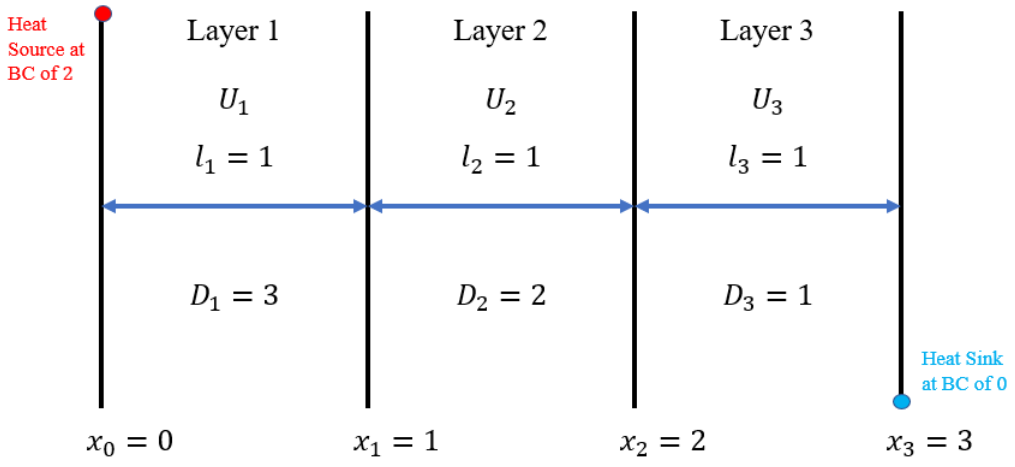

Figure 1: Three Layer Setup

$$f(x) = \begin{cases} 2, & 0 \leq x \leq 1 \\ 2, & 1 < x \leq 2 \\ -2x + 6, & 2 < x \leq 3 \end{cases} \tag{1}$$

Now, in order to solve our multiple layer system, we'll utilize the procedures outlined by the authors[2]. As such, we need to first determine our boundary conditions. From the assumption of perfect thermal contact, we have that the value for any H_i will be

understood as $H_i \to \infty$. As such:

$$U_i(x_i, t) = U_{i+1}(x_i, t) \tag{2}$$

$$D_i \frac{\partial U_i(x_i, t)}{\partial x_i} = D_{i+1} \frac{\partial U_{i+1}(x_i, t)}{\partial x_{i+1}} \tag{3}$$

Also, we utilize the generalized notation in the form of **Robin boundary conditions** as the nomenclature is important later. It is also important to note that the strategy for solving this type of problem exactly is to let $u_i(x,t) = w_i(x,t) + v_i(x,t)$ where $w \equiv$ the *steady state solution* and $v \equiv$ the *transient solution.* So,

$$a_1 w_1 + b_1 \frac{\partial w_1}{\partial x} = \theta_1, \qquad \text{at } x = x_0 \tag{4}$$

$$a_2 w_n + b_2 \frac{\partial w_n}{\partial x} = \theta_2, \qquad \text{at } x = x_n \tag{5}$$

This provides us with a list of some initial conditions given our problem setup, we note $n = 3$ as we only have three layers, and:

$$a_1 = a_2 = 1, \qquad b_1 = b_2 = 0, \qquad \theta_1 = 2, \qquad \theta_2 = 0 \tag{6}$$

$$D_1 = 3, \qquad D_2 = 2, \qquad D_3 = 1 \tag{7}$$

We now wish to solve generically for w_i:

$$D_i \frac{\partial^2 w_i}{\partial x^2} = 0 \qquad \Rightarrow \qquad w_i(x) = q_i(x - x_0) + h_i \tag{8}$$

Utilizing the BCs (4) and (5) we note:

$$h_1 = \frac{\theta_1 - b_1 q_1}{a_1}, \qquad (a_2 l_3 + b_2) q_3 + a_2 h_3 = \theta_2 \tag{9}$$

$$\Rightarrow h_1 = 2, \qquad q_3 = -h_3 \tag{10}$$

The relationships below give us our various q_i and h_i

$$q_{i+1} = \frac{D_i}{D_{i+1}} q_i = \frac{D_1}{D_{i+1}} q_1 \tag{11}$$

$$h_{i+1} = h_i + q_i \left(\frac{D_i}{H_i} + l_i \right) \tag{12}$$

With further algebraic manipulation we solve for q_1 as (we make a note that the authors of the referenced paper made an error and the equivalent plus one isn't present in their algebraic manipulations on the denominator):

$$q_1 = \frac{(a_1\theta_2 - a_2\theta_1) D_3}{1 + a_1 b_2 D_1 - a_2 b_1 D_3 + a_1 a_2 D_1 D_3 \left(\frac{L}{D_{av}} \right)} = \frac{(-2)1}{1 + (1)(3)\left(\frac{3}{2}\right)} = -\frac{4}{11} \tag{13}$$

As such, we see that:

$$q_1 = -\frac{4}{11}, \qquad q_2 = -\frac{6}{11}, \qquad q_3 = -\frac{12}{11} \tag{14}$$

$$h_1 = 2, \qquad h_2 = \frac{18}{11}, \qquad h_3 = \frac{12}{11} \tag{15}$$

This means our respective equations for our steady state solution are:

$$w_1 = -\frac{4}{11} x + 2 \tag{16}$$

$$w_2 = -\frac{6}{11}(x - 1) + \frac{18}{11} \tag{17}$$

$$w_3 = -\frac{12}{11}(x - 2) + \frac{12}{11} \tag{18}$$

As a spoiler, we check our work above using numerical integration (which aided in discovering how the error was found in the companion paper), showing the steady state solution outlined for our various $x_{i-1} < x < x_i$ domain intervals[2]. We note a few interesting points. First, we don't get one straight final steady state solution line as might be expected; this is a departure from our intuition of how heat transfer in a one dimensional system through a single medium works. The different boundary layers only transport heat across themselves so quickly, and we note that **a lower D_i corresponds with more rapid heat transfer**. Secondly, we also see from the equations above that this process is recursive, and could be applied more generically and algorithmically to a larger number of layers[2].

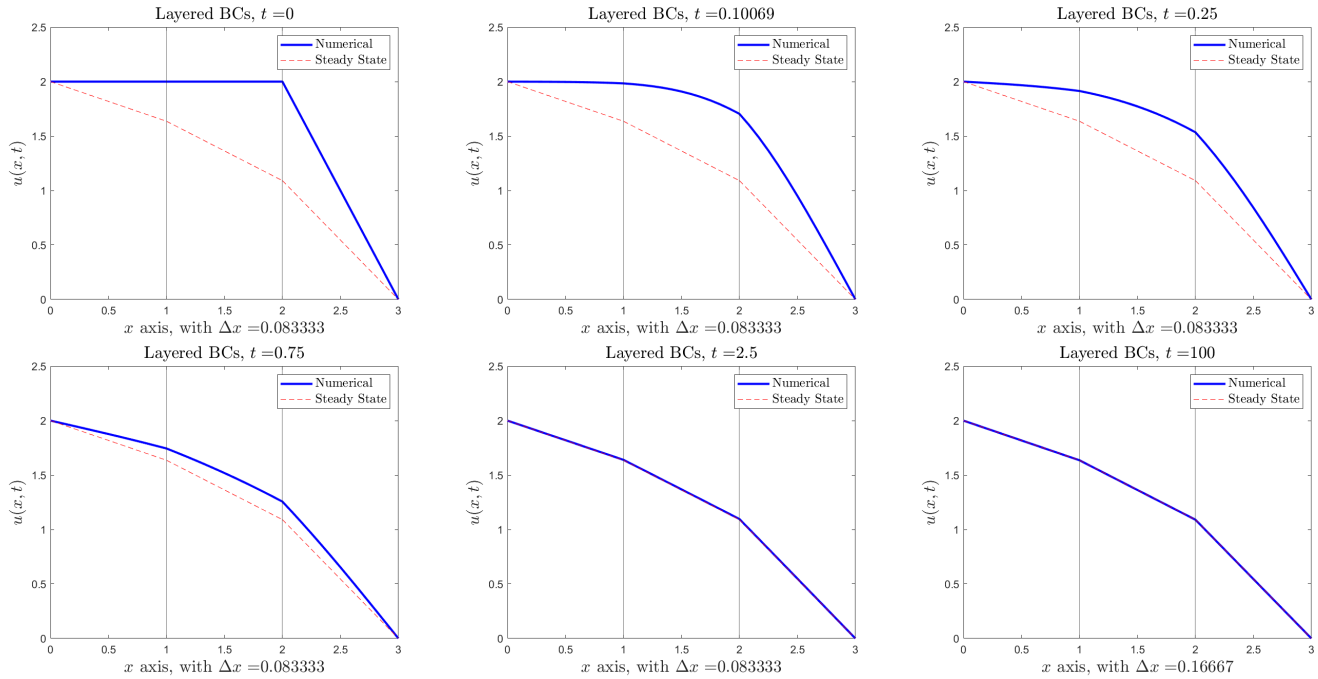

Figure 2: Numerical Integration to the Steady State Solution via FDS

The images in Figure 2 were captured via simple MATLAB code, and special care should be taken to view the time-steps of the photos. We display several images from $0-1$ seconds, and the final two are from $t = 2.5, 100$, where we finally see a full adherence to the steady state solution that we determined analytically; this is displayed both initially and for a much longer period of time. We also observe that here $\Delta x = 1/6$ instead of the typical $\Delta x = 1/10$ (for reasons that will be discussed later).

Now, after confirming that we have indeed calculated the steady state solution correctly, we now need to find the transient solution as our final solution will be $u_i(x,t) = w_i(x) + v_i(x,t)$. For the boundary conditions and other values, as well as the relationship that $v_i(x,0) = f_i(x) - w_i(x) = g_i(x)$, gives:

$$a_1 v_1 + b_1 \frac{\partial v_1}{\partial x} = 0, \qquad \text{at } x = 0 \qquad \Rightarrow v_1(x) = 0 \tag{19}$$

$$a_2 v_1 + b_2 \frac{\partial v_3}{\partial x} = 0, \qquad \text{at } x = 0 \qquad \Rightarrow v_n(x) = 0 \tag{20}$$

$$v_1(x,0) = 2 - w_1 \qquad \Rightarrow \qquad g_1(x) = \frac{4}{11}x \tag{21}$$

$$v_2(x,0) = 2 - w_2 \qquad \Rightarrow \qquad g_2(x) = -\frac{2}{11} + \frac{6}{11}x \tag{22}$$

$$v_3(x,0) = -2x + 6 - w_3 \qquad \Rightarrow \qquad g_3(x) = -\frac{10}{11}x + \frac{30}{11} \tag{23}$$

Using separation of variables our eigenfunction solutions for any $v_i(x,t) = X_i(x)T(t)$ (also using the shorthand notation of $\sqrt{D_i} = d_i$):

$$X_i(x) = K_{1,i} \sin\left(\frac{\lambda_m}{d_i}(x - x_{i-1})\right) + K_{2,i} \cos\left(\frac{\lambda_m}{d_i}(x - x_{i-1})\right) \tag{24}$$

$$T(t) = e^{-\lambda_m^2 t} \tag{25}$$

We see nicely that due to our boundary conditions, with $b_1 = b_2 = 0$ we'll have:

$$K_{2,1} = \frac{-b_1 \lambda_m}{a_1 d_1} = 0 \tag{26}$$

Hence, our first equation, $X_1(x)$ is (after making the choice of $K_{1,1} = 1$):

$$X_1(x) = \sin\left(\frac{\lambda_m}{d_i}(x - x_{i-1})\right) = \sin\left(\frac{\lambda_m x}{d_1}\right) \tag{27}$$

Continuing to determine the values of our other coefficients, we utilize the assumption that we have the same flux through the layer boundaries given by both:

$$K_{1,i+1} = \frac{d_i}{d_{i+1}}\left[K_{1,i} \cos\left(\frac{\lambda_m l_i}{d_i}\right) - K_{2,i} \sin\left(\frac{\lambda_m l_i}{d_i}\right)\right] \tag{28}$$

$$K_{2,i+1} = K_{1,i}\left[\sin\frac{\lambda_m l_i}{d_i} + \frac{\lambda_m d_i}{H_i}\cos\frac{\lambda_m l_i}{d_i}\right] + K_{2,i}\left[\frac{-\lambda_m d_i}{H_i}\sin\frac{\lambda_m l_i}{d_i} + \cos\frac{\lambda_m l_i}{d_i}\right] \tag{29}$$

But we note again in the above that we're letting $H_i \to \infty$, therefore we reduce the above to and include our final n coefficients equation:

$$K_{2,i+1} = K_{1,i} \sin \frac{\lambda_m}{d_i} + K_{2,i} \cos \frac{\lambda_m}{d_i} \tag{30}$$

$$0 = K_{1,n} \left[a_2 \sin \frac{\lambda_m l_n}{d_n} + \frac{\lambda_m b_2}{d_n} \cos \frac{\lambda_m l_n}{d_n} \right] + K_{2,n} \left[-\frac{\lambda_m b_2}{d_n} \sin \frac{\lambda_m l_n}{d_n} + a_2 \cos \frac{\lambda_m l_n}{d_n} \right] \tag{31}$$

$$0 = K_{1,3} \sin \frac{\lambda_m}{d_3} + K_{2,3} \cos \frac{\lambda_m}{d_3} \tag{32}$$

Critically then, we're in a good place to use (29) and (31) and eventually (32). So, specifically for $K_{1,2}$ we have:

$$K_{1,2} = \frac{d_1}{d_2} \left[K_{1,1} \cos \left(\frac{\lambda_m}{d_1} \right) + \underbrace{K_{2,1}}_{0} \sin \frac{\lambda_m}{d_1} \right] = \frac{d_1}{d_2} \cos \left(\frac{\lambda_m}{d_1} \right) \tag{33}$$

Utilizing the other boundary flux condition:

$$K_{2,2} = K_{1,1} \left[\sin \frac{\lambda_m}{d_1} \right] + \underbrace{K_{2,1}}_{0} \left[\cos \frac{\lambda_m}{d_1} \right] \tag{34}$$

$$K_{2,2} = \sin \frac{\lambda_m}{d_1} \tag{35}$$

Again for $K_{1,3}$

$$K_{1,3} = \frac{d_2}{d_3} \left[K_{1,2} \cos \left(\frac{\lambda_m}{d_2} \right) - K_{2,2} \sin \left(\frac{\lambda_m}{d_2} \right) \right] \tag{36}$$

$$K_{1,3} = \frac{d_2}{d_3} \left[\left(\frac{d_1}{d_2} \left[\cos \left(\frac{\lambda_m}{d_1} \right) \right] \right) \cos \left(\frac{\lambda_m}{d_2} \right) - \sin \frac{\lambda_m}{d_1} \sin \left(\frac{\lambda_m}{d_2} \right) \right] \tag{37}$$

$$K_{1,3} = \frac{d_2}{d_3} \left[\frac{d_1}{d_2} \cos \left(\frac{\lambda_m}{d_1} \right) \cos \left(\frac{\lambda_m}{d_2} \right) - \sin \frac{\lambda_m}{d_1} \sin \left(\frac{\lambda_m}{d_2} \right) \right] \tag{38}$$

As such, we have an expression purely in terms of trig functions and λ_m for $K_{1,3}$. Also,

$$K_{2,3} = K_{1,2} \left[\sin \frac{\lambda_m}{d_2} \right] + K_{2,2} \left[\cos \frac{\lambda_m}{d_2} \right] \tag{39}$$

$$K_{2,3} = \frac{d_1}{d_2} \cos \left(\frac{\lambda_m}{d_1} \right) \sin \frac{\lambda_m}{d_2} + \sin \frac{\lambda_m}{d_1} \cos \frac{\lambda_m}{d_2} \tag{40}$$

We now place all of these values into each other until we're left with a transcendental expression we can solve for in terms of λ_m, and keeping in mind that $d_i = \sqrt{D_1}$:

$$0 = K_{1,3} \sin \frac{\lambda_m}{d_3} + K_{2,3} \cos \frac{\lambda_m}{d_3} \tag{41}$$

$$0 = K_{1,3} \sin \lambda_m + K_{2,3} \cos \lambda_m \tag{42}$$

$$0 = \left(d_2 \left[\frac{d_1}{d_2} \cos \left(\frac{\lambda_m}{d_1} \right) \cos \left(\frac{\lambda_m}{d_2} \right) - \sin \frac{\lambda_m}{d_1} \sin \left(\frac{\lambda_m}{d_2} \right) \right] \right) \sin \lambda_m \tag{43}$$

$$+ \left(\frac{d_1}{d_2} \cos \left(\frac{\lambda_m}{d_1} \right) \sin \frac{\lambda_m}{d_2} + \sin \frac{\lambda_m}{d_1} \cos \frac{\lambda_m}{d_2} \right) \cos \lambda_m \tag{44}$$

Using MAPLE's roots function we solve for and keep the first seven λ_m values:

$$\lambda_m = \begin{cases} m = 1: & 1.397471655 \\ m = 2: & 2.800269586 \\ m = 3: & 4.009928200 \\ m = 4: & 5.582827933 \\ m = 5: & 6.847666687 \\ m = 6: & 8.295778221 \\ m = 7: & 9.558768567 \end{cases} \tag{45}$$

As such, we now have what we need for our various values of $K_{i,j}$, which gives a general coefficient of C_m and our transient solution is:

$$v_i(x,t) = \sum_{m=1}^{\infty} C_m e^{-\lambda_m^2 t} X_i(x) \tag{46}$$

With initial condition and projection of our basis to get the various C_m as:

$$v_i(x,0) = g_i(x) = \sum_{m=1}^{\infty} C_m X_i(x) \tag{47}$$

$$C_m = \frac{\sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} g_i(x) X_i(x)\ dx}{\sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} X_i^2(x)\ dx} \tag{48}$$

Explicitly writing our eigenfunctions:

$$X_1(x) = \sin\left(\frac{\lambda_m}{d_1} x\right) \tag{49}$$

$$X_2(x) = K_{1,2} \sin\left(\frac{\lambda_m}{d_2}(x-1)\right) + K_{2,2} \cos\left(\frac{\lambda_m}{d_2}(x-1)\right) \tag{50}$$

$$X_3(x) = K_{1,3} \sin\left(\lambda_m(x-2)\right) + K_{2,3} \cos\left(\lambda_m(x-2)\right) \tag{51}$$

We use the above to calculate the exact solution (below) and will display the results with the completed FDS.

$$u_i(x,t) = w_i(x) + \sum_{m=0}^{\infty} C_m X_i(x) e^{-\lambda_m^2 t} \tag{52}$$

$$u(x,t) = u_1(x_1,t) + u_2(x_2,t) + u_3(x_3,t), \qquad \text{where} \qquad 0 \le x_1 \le 1,\ 1 \le x_1 \le 2,\ 2 \le x_1 \le 3 \tag{53}$$

This exact solution will be useful, but we make a final note of just how much effort was needed to produce it!

3 Finite Difference Scheme (FDS) for Matching Diffusivities with Perfect Thermal Contact

For the heat equation, we're numerically solving the PDE $u_t = bu_{xx}$. As such, the forward-time central-space scheme (6.3.1) from Strikwerda provides us with an acceptable FDS for the majority of the points in our scheme (here we share the notation of the main article as $\Delta x, \Delta t, D_i$ instead of h, k, b respectively).

$$\frac{v_m^{n+1} - v_m^n}{\Delta t} = D_i \frac{v_{m+1}^n - 2v_m^n + v_{m-1}^n}{\Delta x^2} \Rightarrow v_m^{n+1} = D_i \mu (v_{m+1}^n - 2v_m^n + v_{m-1}^n) + v_m^n, \text{ with } \mu = \frac{\Delta t}{\Delta x^2} \tag{54}$$

Special care now needs to be taken as $D_i\mu \le 1/2$ is required in order for the scheme to be dissipative of order 2. We also note that our choice of $D_i\mu$ will make the scheme accurate of order (1,2) or not; hence, we need to restrict it.

Now for the novel component of the main article. In order to properly account for the barrier points we need to have some sort of appropriate averaging. With a little bit of prior FDS understanding, their equations (11) and (13) can be understood as:

$$v_m^{n+1} = \mu[D_{i+1} v_{m+1}^n - (D_{i+1} + D_i) v_m^n + D_i v_{m-1}^n] + v_m^n \tag{55}$$

This is effectively an averaging between both separate layers and their respective diffusivity coefficients as to the instantaneous value of $v_{m=x_i}^n$. The below figure demonstrates the algorithm used in the code of Section 4:

In the above exact computations, it was clear that the formulas, given certain starting conditions, could become extremely cumbersome, whereas after the initial set up of the code for the FDS, further alterations to it become trivial. In Figure 4 we display the initial and final time instances for a ten layer scheme, evenly spaced at one length apart, with $D_i = 3, 2, 1, 3, 2, 1, 3, 2, 1, 3$. Computing this by hand or creating a recursive algorithm to handle the various cases could be extremely challenging but are quite simple with the FDS. At the very least, this method could be used to brute force an easier path to uncovering steady state solutions for more complicated problems.

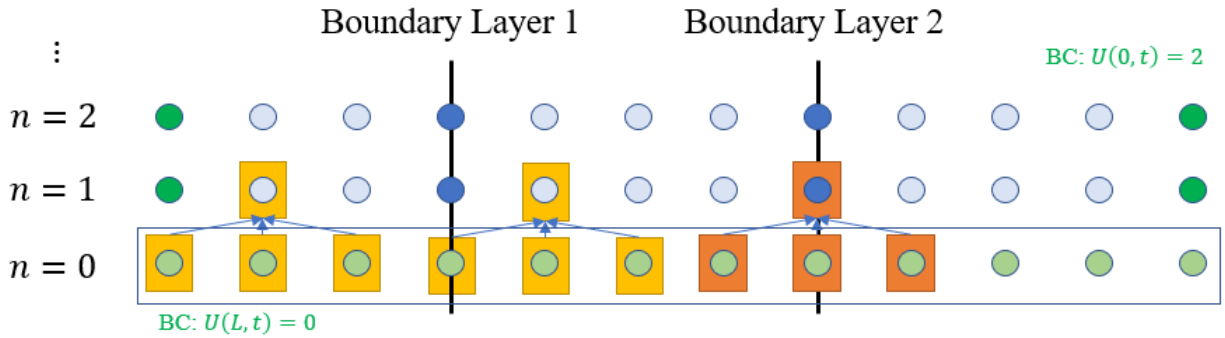

Figure 3: The first row in the thin box is the initial condition given in this example: the piecewise function. The far left and right boundary conditions are also given as a source and sink respectively. The yellow boxes and light blue points are computed first in a single pass via (56). The orange boxes with dark blue points are computed via (55).

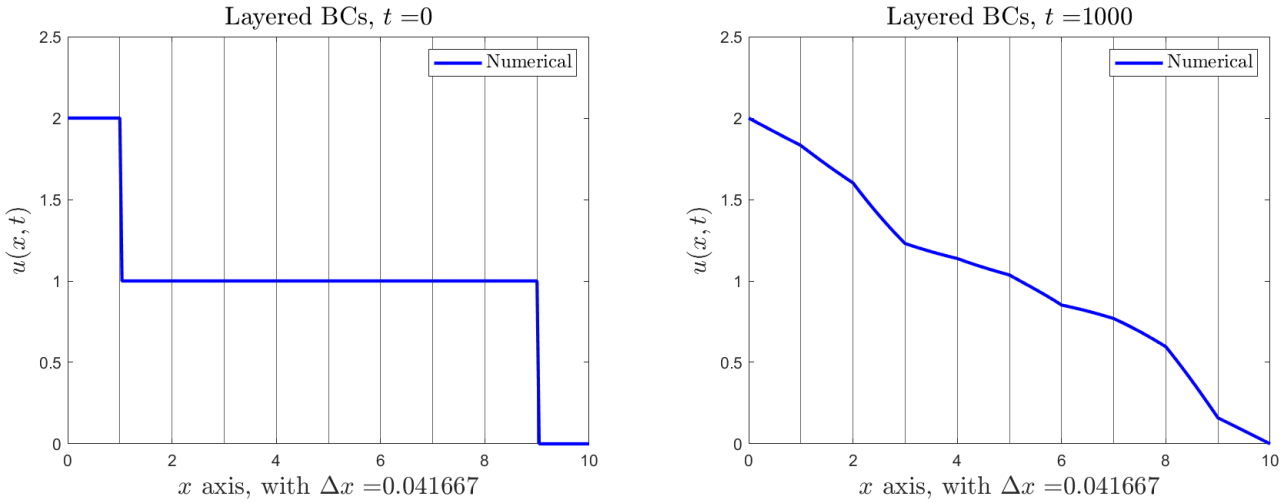

Figure 4: Ten layer system. On the left, the initial condition. On the right, the steady state solution. The time required to compute and plot the image on the right was negligible.

Revisiting the stipulation on $D_i\mu$, we see that this puts a stronger restriction on how fine we can make our mesh. In the example used for this paper, the largest D_i, or $\max(D_i)$, used was 3, meaning that $\Delta t = \Delta x^2/6$ in order to ensure no D_i values would cause a lack of diffusivity or stability in the scheme. This has an obvious consequence; for large time scales and more intensively fine grids, this scheme becomes especially expensive. Utilizing MATLAB's "Run and Time" function to compile ten snapshot photos and finish running at $\Delta x = 1/6$, was 14.83 seconds. To do the same at $\Delta x = 1/24$ was 420.69 seconds. Therefore, refining the mesh size from $\Delta x \to \Delta x/4$ caused a roughly 28 times increase in run time on the adjusted scheme seen in Figure 4. This drastically limits the scheme's usefulness for various real world applications as increasing the mesh density caused the computation of the simulation to take almost half of the time of the physical system being simulated! Another division by two to Δx would cause the scheme to take longer than the actual physical system's progression; thereby making it only analytically useful if time isn't a pressing concern, but not practical for real time applications.

4 Exact Solution, FDS, and Error Comparison for Perfect Thermal Contact

Figure 5 displays a series of plots that show even at extremely low resolution, namely $\Delta x = 1/6$, that we have a strong adherence to the exact solution by our FDS.

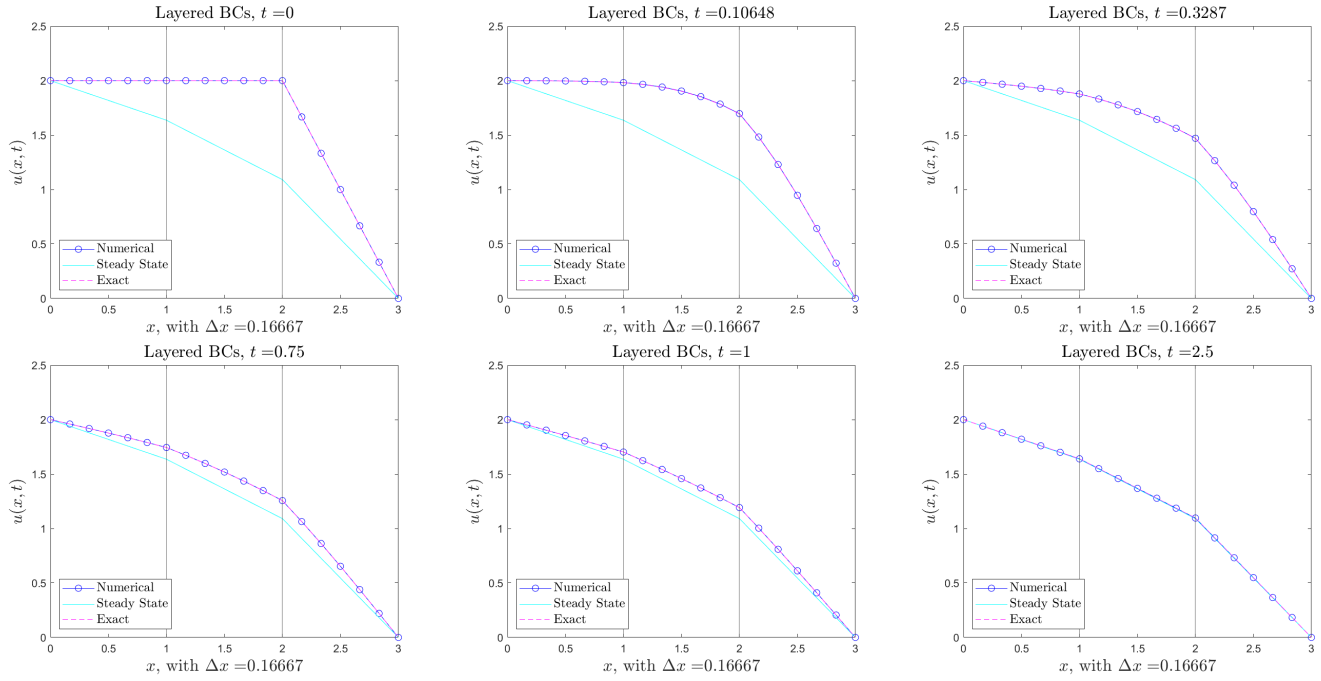

Figure 5: Exact solution compared to the FDS

Now for the errors. The $L2$ norms were computed at the end of each full FDS simulation for a fixed Δx. The order of accuracy computations were then done by comparing the errors of the previous and current Δx. The results are shown in the MATLAB output table below. We can see that using either norm gave us the same order of accuracy:

```
Final_Error_Results =

  6×5 string array

    "Delta x"     "L2 Error"      "Order r"     "Inf Error"      "Order r"
    "0.16667"     "3.4962e-05"    "0"           "2.8809e-05"     "0"
    "0.083333"    "8.7362e-06"    "2.0007"      "7.199e-06"      "2.0007"
    "0.041667"    "2.1838e-06"    "2.0002"      "1.7995e-06"     "2.0002"
    "0.020833"    "5.4592e-07"    "2.0001"      "4.4992e-07"     "2.0001"
    "0.010417"    "1.3648e-07"    "2"           "1.1248e-07"     "2"
```

Figure 6: Error computations using both the $L2$ and Supremum norms

The formulas used for calculating the order of accuracy and $L2$ norm were:

$$r = \frac{\left| \ln\left(\frac{\|\cdot\|_{\Delta x/2}}{\|\cdot\|_{\Delta x}} \right) \right|}{\ln(2)} \qquad \text{with } L2 \text{ as} \qquad \text{Error}(t_n) = \left(\Delta x \sum_{i=1}^{m} |u(t_n, x_i) - v_i^n|^2 \right)^{(1/2)} \tag{56}$$

Note that the $\|\cdot\|$ notation denotes the $L2$ norm, and the $L2$ norm is calculated via the equation on the right.

5 Conclusion and Discussion

A few criticisms of the main article[1] and it's companion reference paper[2] for the exact solutions are minimal. The error in the exact solution reference paper undermines the remaining solutions' veracity and cannot be implemented directly as is. Fortunately, the authors' method for the FDS was correct, and allowed for the analytical error to be quickly spotted. Furthermore, the system in the main article being solved wasn't provided beyond boundary conditions, thereby rendering the graphical aids unhelpful, as it isn't apparent what initial conditions were being used. Lastly, no detailed outline is given of the exact solution to the problem statement (which could have nullified the previous comment). Therefore, their error calculations, graphs, and conclusions though likely accurate given the repeatability of their processes, weren't verified in their own work. Simply put, they didn't clearly outline what it was that they were solving.

Criticisms aside, the main article and companion paper provide an excellent introduction to the idea of both multiple layer heat equation PDEs, and their FDS implementation methods. The authors were able to provide a functioning and impressive recursive set of equations to determine the eigenfunctions' coefficients, $K_{i,j}$, for the spatial solutions, $X_i(x)$ (and subsequently for the various C_m), thereby making the process accessible and solvable for those with an introductory understanding of parabolic PDEs.

Pivoting to the FDS itself, a clear strength of the numerical approximation is its ease of implementation. To arrive at a fully functioning second order accurate scheme required: little technical knowledge, adherence to a single stipulation on $\max(D_i)\mu$, and the careful establishment of the boundary values for a self-replacing $[1 \times n]$ array in a loop. Comparing this to the overhead required to learn how to compute exact solutions for boundary layer PDEs is considerable. The exact solution's generic Robin boundary conditions, the implementation of the recursive equations for the $K_{i,j}$, and the establishing of unique functions for both the steady state and transient solutions, was far from trivial. Also generalizing the code would require multiple layers of looped function definitions, an understanding of PDEs generally, would be non-trivial to implement, and prone to errors thereby requiring more robust case testing. Even with the exact solution's moderate level of complexity, if the problem statement shifted to needing to find the solution with forcing functions, dynamic boundary conditions, or non-constant densities (i.e. $D_i \to D_i(x)$), the exact solution would need to be completely reworked. In contrast, we can be *a priori* confident that the FDS would handle these changes easily. At it's core, the FDS isn't deadlocked to Robin Boundary conditions with static material properties whereas the exact solution is.

A final obvious advantage of the scheme is it's ability to rapidly predict the steady state solution given extremely low mesh densities. For quick computations and re-utilization of code (perhaps by someone not well versed in PDEs or the solutions outlined in the reference paper[2]), the steady state for a complex layered system could be found rather easily. Echoing the last paragraph in a slightly different way; the FDS' strength is clearly it's simplicity.

The numerical approximation scheme however has some clear downfalls. The obvious first candidate was discussed in Section 3; decreasing the step-size drastically increases overall computation time. If we wish to utilize this scheme to aid in solving heat distributions across a computer chip as an example, we make an immediate note that feature sizes on a chip are measured in nanometers. A typical transistor on an EUV produced microprocessor can be less than 100nm. Making a conservative estimation that we'll only be concerned with step sizes of 10,000nm, we would still require $\Delta x = 0.001$ to run a simulation on a 1cm chip in 1D; this would simply be untenable without serious computing power or craftier techniques.

We also state an untested concern in regards to the lengths between boundaries. In the exact solution, this distance is just an abstracted parameter and only effects the solution qualitatively. If a relatively course mesh is used in the FDS however, errors could occur (especially at early stages of the simulation), if the distance between the two numbers isn't a ratio of rationals. Some inherent round-off error would occur. This effect would likely be negligible for small enough step sizes, but larger step sizes could see some error creep.

Lastly, and most interestingly, we see that the physical properties of the material directly impact the effectiveness of the FDS! This is strikingly counter-intuitive or at least unexpected. We noted in Section 2 that a lower D_i corresponds to higher heat transfer. As such, an FDS for materials in perfect thermal contact that share heat rather rapidly, would be able to be more finely simulated than materials that share heat slowly *for otherwise the same simulation setup.* This is again caused by our stipulation: $\max(D_i)\mu \leq 1/2$. In our example problem throughout the paper, the 3 dictated the size of our Δt. Generically:

$$\max(D_i)\mu \leq 1/2 \qquad \Rightarrow \qquad \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2\max(D_i)} \qquad \Rightarrow \qquad \Delta t \leq \frac{\Delta x^2}{2\max(D_i)} \tag{57}$$

As such, a fixed Δx implies that a larger and larger D_i for a layer forces Δt to be smaller and smaller; again implying that a decrease in diffusivity of a layer's material results directly in increased computational time for the scheme. The type of material being examined could require the usage of a completely different FDS.

We end by acknowledging the scheme isn't a catch all, but clearly has potential for a wide range of applications. A final statement is made that agrees with the authors; the method is extremely flexible for a wide array of problems, and is also easy to implement.

6 References

[1] Hickson, R. I., Barry, S. I., Mercer, G. N., & Sidhu, H. S. (2011). Finite difference schemes for multilayer diffusion. Mathematical and Computer Modelling, 54(1-2), 210-220.

[2] Hickson, Roslyn & Barry, S.I. & Mercer, G.N.. (2009). Critical times in multilayer diffusion. Part 1: Exact solutions. International Journal of Heat and Mass Transfer - INT J HEAT MASS TRANSFER. 52. 5776-5783. 10.1016/j.ijheatmasstransfer.2009.08.013.

[3] "All about Light and Lasers in Lithography." ASML, https://www.asml.com/en/technology/lithography-principles/light-and-lasers.

[4] Strikwerda, John C. Finite Difference Schemes and Partial Differential Equations. Society for Industrial and Applied Mathematics, 2004.

7 Appendix A: Code Used

First is the main code used for computing the exact and approximate solutions while comparing them. Second is the code used to produce a ten layer system. We make a note that this code is highly inflexible. It only allows for the solving of this "exact" problem set up. The only things that can be easily changed in the first block of code are: the initial conditions, and the values of the three D_i (which are labelled as b below).

```
%% Final Project Code

close all
clear all
clc

%array of delta x values for cycling
delx = [1/6, 1/12, 1/24]% , 1/48, 1/96];
delxcount = length(delx);
b = [3, 2, 1]; % b values from u_t = b*u_xx, in the paper this is D_i, b used initially as
    Strikwerda uses b
Bm = length(b);

% Heavy Content Load for Exact Solution
% view paper for details

%Lambda's computed in MAPLE, could be optimized by computing them in a
%single script in MATLAB, not done here due to time constraints

lambda = [1.397471655, 2.800269586, 4.009928200, 5.582827933, 6.847666687, 8.295778221,
    9.558768567, 11.00922329, 12.44277444, 13.69468804];
layers = 3;
LL = length(lambda);
CmNum = 0;
CmDen = 0;
% Diffusion coefficients
d1 = sqrt(3); d2 = sqrt(2); d3 = 1;
%Cm placeholder
Cm = zeros(1,LL);

%Note that the below are NOT optimized, these equations could be put into a
%loop to generalize the code and make it applicable for all robin BC
%problems

% Defining the X_im(x) functions
funX1 = @(x,Lam) (sin(Lam./d1.*(x)));
funX2 = @(x,Lam) (((d1./d2.*cos(Lam./d1).*sin(Lam./d2.*(x-1))) + (sin(Lam./d1).*cos(Lam./
    d2.*(x-1)))));
funX3 = @(x,Lam) (((d2./d3.*(((d1./d2.*cos(Lam./d1).*cos(Lam./d2))-(sin(Lam./d1)).*(sin(
    Lam./d2))))).*sin(Lam.*(x-2))) ...
        + (d1./d2.*cos(Lam./d1).*sin(Lam./d2)+sin(Lam./d1).*cos(Lam./d2)).*cos(Lam.*(x-2))
            );
%Squared X_im
funX1sqr = @(x,Lam) (sin(Lam./d1.*(x))).^2;
funX2sqr = @(x,Lam) (((d1./d2.*cos(Lam./d1).*sin(Lam./d2.*(x-1))) + (sin(Lam./d1).*cos(Lam
    ./d2.*(x-1))))).^2;
funX3sqr = @(x,Lam) (((d2./d3.*(((d1./d2.*cos(Lam./d1).*cos(Lam./d2))-(sin(Lam./d1)).*(sin
    (Lam./d2))))).*sin(Lam.*(x-2))) ...
        + (d1./d2.*cos(Lam./d1).*sin(Lam./d2)+sin(Lam./d1).*cos(Lam./d2)).*cos(Lam.*(x-2))
            ).^2;
%Functions with g_i(x)
funX1g1 = @(x,Lam) (sin(Lam./d1.*(x))).*(4./11.*x);
```

```
funX2g2 = @(x,Lam) (((d1./d2.*cos(Lam./d1).*sin(Lam./d2.*(x-1))) + (sin(Lam./d1).*cos(Lam./d2.*(x-1))))).*(-2./11 + 6./11.*x);
funX3g3 = @(x,Lam) (((d2./d3.*(((d1./d2.*cos(Lam./d1).*cos(Lam./d2))-(sin(Lam./d1)).*(sin(Lam./d2))))).*sin(Lam.*(x-2))) ...
        + (d1./d2.*cos(Lam./d1).*sin(Lam./d2)+sin(Lam./d1).*cos(Lam./d2)).*cos(Lam.*(x-2))).*(-10./11.*x+30./11);

% Precomputing Coefficients C_m
for m = 1:LL
   CmNum = integral(@(x) funX1g1(x,lambda(m)),0,1);
   CmNum = integral(@(x) funX2g2(x,lambda(m)),1,2) + CmNum;
   CmNum = integral(@(x) funX3g3(x,lambda(m)),2,3) + CmNum;

   CmDen = integral(@(x) funX1sqr(x,lambda(m)),0,1);
   CmDen = integral(@(x) funX2sqr(x,lambda(m)),1,2) + CmDen;
   CmDen = integral(@(x) funX3sqr(x,lambda(m)),2,3) + CmDen;

   Cm(m) = CmNum / CmDen;
end


%Computing FDS alongside of Exact Solution

for q = 1: length(delx)

    x = 0:delx(q):3;
    %delt not optimized, hard set for this problem to make sure D_i mu
    %<=1/2 for largest D_i which in this case is 3
    delt = delx(q)^2/6;
    M = length(x);
    t = 0:delt:2.5;

    u_old = zeros(1,M);
    u_new = zeros(1,M);

    %Steady State Solutions, outside loop
    x1 = 0:delx(q):1;
    x2 = 1:delx(q):2;
    x3 = 2:delx(q):3;
    y1 = -4/11*x1+2;
    y2 = -6/11*(x2-1)+18/11;
    y3 = -12/11*(x3-2)+12/11;
    %Used for exact solution
    MM = length(x1);
    v1 = zeros(1,MM); v2 = zeros(1,MM); v3 = zeros(1,MM);



    for T = t
        %below is for the IC
        if T == t(1) % establishing staggered start IC
            for X = 1:M
                if x(X) <= 1
                    u_old(X) = 2;
                elseif x(X) > 1 & x(X) <= 2
                    u_old(X) = 2;
                else
                    u_old(X) = -2*x(X)+6;
                end
            end
```

```
102             u_new = u_old;
103             u1 = 2 + x1.*0;
104             u2 = 2 + x2.*0;
105             u3 = 6 - 2*x3;
106 
107         else
108             % This portion is for the second time step and after
109             % Below Portion for the FDS
110             for X = 2:M-1
111                 u_new(1) = 2; u_new(end) = 0; % Hardcoding of static BCs
112                 if x(X) == 1
113                     % boundary layer one
114                 elseif x(X) == 2
115                     % boundary layer two
116                 else
117                     % typical intermediate points using 6.3.1 of Strikwerda
118                     if x(X) < 1
119                         B = b(1);
120                     elseif x(X) >1 & x(X) < 2
121                         B = b(2);
122                     else
123                         B = b(3);
124                     end
125                     u_new(X) = delt*B*(u_old(X+1)-2*u_old(X)+u_old(X-1))/delx(q)^2 + u_old(
                            X);
126                 end
127             end
128             for X = 1:M
129                 if x(X) == 1
130                     LP1 = X;
131                 elseif x(X) == 2
132                     LP2 = X;
133                 else
134 
135                 end
136             end
137             % doing the averaging of the Boundary Layer points, x0 and x1
138             u_new(LP1) = (b(2)*u_old(LP1+1)-(b(2)+b(1))*u_old(LP1)+b(1)*u_old(LP1-1))/(
                    delx(q)^2)*delt+u_old(LP1);
139             u_new(LP2) = (b(3)*u_old(LP2+1)-(b(3)+b(2))*u_old(LP2)+b(2)*u_old(LP2-1))/(
                    delx(q)^2)*delt+u_old(LP2);
140 
141             % Exact Solution Computations
142             for m = 1:LL
143                 % first for v1
144                 X1 = funX1(x1,lambda(m)).*Cm(m).*exp(-lambda(m).^2.*T);
145                 v1 = X1 + v1;
146                 % for v2
147                 X2 = funX2(x2,lambda(m)).*Cm(m).*exp(-lambda(m).^2.*T);
148                 v2 = X2 + v2;
149                 % for v3
150                 X3 = funX3(x3,lambda(m)).*Cm(m).*exp(-lambda(m).^2.*T);
151                 v3 = X3 + v3;
152             end
153             u1 = -4/11.*x1+2 + v1;
154             u2 = -6/11.*(x2-1)+18/11 + v2;
155             u3 = -12/11.*(x3-2)+12/11 + v3;
156             X1 = 0; X2 = 0; X3 = 0; v1 = zeros(1,MM); v2 = zeros(1,MM); v3 = zeros(1,MM);
157 
158         % Counter used to track progress during longer computational runs
```

```
159 %         end
160 %         if mod(T,100) == 0
161 %             disp(T)
162 %         end
163
164         % Plotting the Exact, Numerical and Steady State solutions every
165         % loop to be in a movie format
166 %         if T == t(1) | T == t(12) | T == t(24) | T == t(36) | T == t(48) | T == t(60) |
    T == t(72)...
167 %                     | T == t(84) | T == t(96) | T == t(109) | T == t(163) | T == t(217)
     | T == t(325) ...
168 %                     | T == t(end)
169             plot(x,u_new,'b-o','LineWidth',2)
170             hold on
171             plot(x1,y1,'-c','LineWidth',2)
172             plot(x2,y2,'-c','LineWidth',2)
173             plot(x3,y3,'-c','LineWidth',2)
174             plot(x1,u1,'--r','LineWidth',2)
175             plot(x2,u2,'--r','LineWidth',2)
176             plot(x3,u3,'--r','LineWidth',2)
177             ylim([0,2.5])
178             title("Layered BCs, $t=$"+T,'Interpreter','latex','FontSize',16)
179             xlabel("$x$, with $\Delta x=$"+delx(q),'Interpreter','latex','FontSize',16)
180             ylabel('$u(x,t)$','Interpreter','latex','FontSize',16)
181             xline(1)
182             xline(2)
183             xline(3)
184             legend('Numerical','Steady State','','','Exact','Interpreter','latex','
                FontSize',14,'location','SouthWest')
185             hold off
186 %
187             pause(.01)
188
189             %Taking images at various timesteps
190 %             if T == t(1)
191 %                 saveas(gcf,'Exact_Approx_1.png')
192 %             elseif T == t(12)
193 %                 saveas(gcf,'Exact_Approx_2.png')
194 %             elseif T == t(24)
195 %                 saveas(gcf,'Exact_Approx_3.png')
196 %             elseif T == t(36)
197 %                 saveas(gcf,'Exact_Approx_4.png')
198 %             elseif T == t(48)
199 %                 saveas(gcf,'Exact_Approx_5.png')
200 %             elseif T == t(60)
201 %                 saveas(gcf,'Exact_Approx_6.png')
202 %             elseif T == t(72)
203 %                 saveas(gcf,'Exact_Approx_7.png')
204 %             elseif T == t(84)
205 %                 saveas(gcf,'Exact_Approx_8.png')
206 %             elseif T == t(96)
207 %                 saveas(gcf,'Exact_Approx_9.png')
208 %             elseif T == t(109)
209 %                 saveas(gcf,'Exact_Approx_10.png')
210 %             elseif T == t(163)
211 %                 saveas(gcf,'Exact_Approx_11.png')
212 %             elseif T == t(217)
213 %                 saveas(gcf,'Exact_Approx_12.png')
214 %             elseif T == t(325)
215 %                 saveas(gcf,'Exact_Approx_13.png')
```

```
%               elseif T == t(end)
%                   saveas(gcf,'Exact_Approx_14.png')
%               else
%
%               end
%               pause(.01)
%           end

            u_old = u_new;

    end

    %Computing the Errors
    %first, putting the exact piecewise solution into one array
    ue = u1;
    ue(end) = [];
    ue = [ue,u2];
    ue(end) = [];
    ue = [ue,u3];

    % Sum for L2 Norm
    summeA = 0;
    for X = 1:M
        summeA = summeA + abs(u_new(X)-ue(X))^2;
    end
    %L2 Norm
    L2EA(q) = sqrt(delx(q)*summeA);
    %Sup Norm
    SupEA(q) = max(abs(u_new-ue));

    %Order of accuracy calculation for both L2 and Sup Norm
    if q > 1
        R2LA(q) = abs(log(L2EA(q)/L2EA(q-1)))/log(2);
        RSupA(q) = abs(log(L2EA(q)/L2EA(q-1)))/log(2);
    end

    %End of actions, loop restarts at next time step

end

Final_Error_Results = [["Delta x",delx]',["L2 Error",L2EA]',["Order r",R2LA]',["Inf Error",SupEA]',["Order r",RSupA]']
```

```
%% Final Project Code (non-optimized adaptation for several layers)

close all
clear all
clc

delx = [1/6, 1/12, 1/24]; % Delta x in the spatial scheme
b = [3,2,1,3,2,1,3,2,1,3]; % b values from u_t = b*u_xx, D_i in main article
Bm = length(b);

for q = 1:1 % length(delx)

    x = 0:delx(q):10;
    delt = delx(q)^2/6;
    M = length(x);
    t = 0:delt:1000;

    u_old = zeros(1,M);
    u_new = zeros(1,M);

    for T = t
        if T == t(1) % establishing staggered start IC
           for X = 1:M
              if x(X) <= 1
                  u_old(X) = 2;
              elseif x(X) > 1 & x(X) <= 9
                  u_old(X) = 1;
              else
                  u_old(X) = 0;
              end
           end
           u_new = u_old;

        else

            for X = 2:M-1
               u_new(1) = 2; u_new(end) = 0; % Hardcoding of static BCs
               if x(X) == 1 | x(X) == 2 | x(X) == 3 | x(X) == 4 | x(X) == 5 | x(X) == 6 |
                  x(X) == 7 | x(X) == 8 | x(X) == 9
                   % boundary layer skips
               else
                   % typical intermediate points using 6.3.1 of Strikwerda
                   if x(X) < 1
                       B = b(1);
                   elseif x(X) >1 & x(X) < 2
                       B = b(2);
                   elseif x(X) >2 & x(X) < 3
                       B = b(3);
                   elseif x(X) >3 & x(X) < 4
                       B = b(4);
                   elseif x(X) >4 & x(X) < 5
                       B = b(5);
                   elseif x(X) >5 & x(X) < 6
                       B = b(6);
                   elseif x(X) >6 & x(X) < 7
                       B = b(7);
                   elseif x(X) >7 & x(X) < 8
                       B = b(8);
                   elseif x(X) >8 & x(X) < 9
                       B = b(9);
```

```
60                      else
61                          B = b(10);
62                      end
63                      u_new(X) = delt*B*(u_old(X+1)-2*u_old(X)+u_old(X-1)) + u_old(X);
64                  end
65              end
66              for X = 1:M
67                  if x(X) == 1 | x(X) == 2 | x(X) == 3 | x(X) == 4 | x(X) == 5 | x(X) == 6 |
                        x(X) == 7 | x(X) == 8 | x(X) == 9
68                      LP(x(X)) = X;
69                  else
70
71                  end
72              end
73              for BB = 1:Bm-1
74                  u_new(LP(BB)) = (b(BB+1)*u_old(LP(BB)+1)-(b(BB+1)+b(BB))*u_old(LP(BB))+b(
                        BB)*u_old(LP(BB)-1))/(delx(q)^2)*delt+u_old(LP(BB));
75              end
76          end
77          if mod(T,100) == 0
78              disp(T)
79          end
80
81          % Plotting the Solution
82          if T == t(1) | T == t(23) | T == t(45) | T == t(66) | T == t(88) | T == t(109) | T
                == t(217)...
83                      | T == t(649) | T == t(1081) | T == t(2161) | T == t(10801) | T == t(
                            end) %| T == T
84              plot(x,u_new,'LineStyle','-','Color','b','LineWidth',2)
85              hold on
86              ylim([0,2.5])
87              title("Layered BCs, $t=$"+T,'Interpreter','latex','FontSize',16)
88              xlabel("$x$ axis, with $\Delta x=$"+delx(q),'Interpreter','latex','FontSize'
                    ,16)
89              ylabel('$u(x,t)$','Interpreter','latex','FontSize',16)
90              for XX = 1:Bm
91                  xline(XX)
92              end
93              legend('Numerical','Interpreter','latex','FontSize',12)
94              hold off
95
96  %             if T == t(1)
97  %                 saveas(gcf,'Ten_Layer_1_1.png')
98  %             elseif T == t(23)
99  %                 saveas(gcf,'Ten_Layer_1_2.png')
100 %             elseif T == t(45)
101 %                 saveas(gcf,'Ten_Layer_1_3.png')
102 %             elseif T == t(66)
103 %                 saveas(gcf,'Ten_Layer_1_4.png')
104 %             elseif T == t(88)
105 %                 saveas(gcf,'Ten_Layer_1_5.png')
106 %             elseif T == t(109)
107 %                 saveas(gcf,'Ten_Layer_1_6.png')
108 %             elseif T == t(217)
109 %                 saveas(gcf,'Ten_Layer_1_7.png')
110 %             elseif T == t(649)
111 %                 saveas(gcf,'Ten_Layer_1_8.png')
112 %             elseif T == t(1081)
113 %                 saveas(gcf,'Ten_Layer_1_9.png')
114 %             elseif T == t(2161)
```

```
115 %                     saveas(gcf,'Ten_Layer_1_11.png')
116 %                 elseif T == t(end)
117 %                     saveas(gcf,'Ten_Layer_1_10.png')
118 %                 else
119 %
120 %                 end
121                 pause(.1)
122             end
123                 u_old = u_new;
124         end
125 end
```