Davis Math Lab Project Report, Spring 2026
Numerical Methods of the Heat Equation

Luiza Coimbra Sanabria
Abinand Gopal
John Walker

June 15, 2026

1 Introduction

Here, we utilize numerical methods to solve the heat equation given Dirichlet boundary conditions, a single spatial dimension, and fixed endpoints.

$$\begin{cases} u_t - u_{xx} = 0, \ x \in [a,b] \\ u(x,0) = f(x), \ x \in [a,b] \\ u(a,t) = g_a(t), \ u(b,t) = g_b(t) \\ a < b, \ t \geq 0 \end{cases} \tag{1.1}$$

The heat equation is widely used to model a variety of events, such as heat propagation, fluid flow, etc. In this paper, we closely follow the approach of Song & Wong, 2023, to develop an effective method to accurately solve for $u(x,t)$ based off the integral equation approach. We utilize collocation and develop a method of adaptive Gaussian quadrature to evaluate integral terms. Later, we expand on this using the semi-group property of the heat equation and Chebyshev interpolation to develop a faster method that is equally precise.

2 Preliminaries

Here, we provide context to the key components of our method.

2.1 Chebyshev Polynomials

Chebyshev polynomials can be defined over the interval $[-1, 1]$ in terms of the following recurrence relation:

$$\begin{cases} T_0(x) = 1 \\ T_1(x) = x \\ T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \end{cases} \tag{2.1}$$

When evaluating a Chebyshev polynomial at a specific value within the interval $[-1, 1]$, we use the definition $T_n(x) = \cos(n \arccos(x))$.

We can utilize Chebyshev polynomials as a basis for interpolation. Given a function $f(y)$ over the interval $[a, b]$, we may represent it as follows:

$$f(y) \approx \sum_{i=0}^{n} c_i T_i \left(\frac{2y}{b-a} - \frac{a+b}{b-a} \right) \tag{2.2}$$

To solve for the coefficients, c_i, we can construct a basic linear system of equations by evaluating the Chebyshev polynomials included in our sum and our function at the roots of the n-th Chebyshev polynomial. These roots are defined over the interval $[-1, 1]$ by

$$x_j = \cos\left(\frac{2j+1}{2n} \pi \right) \tag{2.3}$$

such that $j = 0, \ldots, n-1$. When evaluating our desired function at these nodes, we must apply a transformation of

$$a + \frac{b-a}{2}(x_i + 1) \to y_i \tag{2.4}$$

Chebyshev interpolation is a more appropriate practice for estimating oscillatory functions than a monomial basis, particularly at large degrees of n. Using Chebyshev polynomials, the interpolation error becomes bounded above by

$$\max_{a \le x \le b} \left| \frac{(b-a)^n}{2^{2n-1} n!} f^{(n)}(x) \right|. \tag{2.5}$$

2.2 Gaussian Quadrature

For Gaussian quadrature, we estimate an integral using the following form

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{i=0}^{n} w_i f\left(\frac{b-a}{2} x_i + \frac{b+a}{2} \right) \tag{2.6}$$

where the x_i's are the n roots of the n-th degree Legendre polynomial. Legendre polynomials can be defined as follows:

$$L_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n. \tag{2.7}$$

These roots are not equidistant compared to other basic methods, like Newton-Cotes. Using Gaussian quadrature, we incur an error term of

$$E_n(f) = \max_{a<\xi<b} |\frac{(b-a)^{2n+1}(n!)^4}{(2n+1)((2n)!)^3} f^{2n}(\xi)|. \tag{2.8}$$

This quadrature method is exact for polynomials up to degree $2n-1$.

3 Theoretical Background

To begin, we introduce the initial heat potential,

$$J_{[a,b]}[f](x,t) = \int_a^b K(x-y,t)f(y)dy \tag{3.1}$$

where $K(x,t)$ is the heat kernel given by

$$K(x,t) = \frac{1}{\sqrt{4\pi t}} \exp\left(-\frac{x^2}{4t}\right). \tag{3.2}$$

Then, we have the double-layer heat potential that we split into two terms: $D_a[\phi_a](x,t)$ and $D_b[\phi_b](x,t)$.

$$D_a[\phi_a](x,t) = \int_0^t H(a-x,t-\tau)\phi_a(\tau)d\tau \tag{3.3}$$

$$D_b[\phi_b](x,t) = \int_0^t H(x-b,t-\tau)\phi_b(\tau)d\tau \tag{3.4}$$

such that

$$H(y,s) = \frac{y}{4\sqrt{\pi s^3}} \exp\left(-\frac{y^2}{4s}\right) \tag{3.5}$$

with $\phi_a(t)$ and $\phi_b(t)$ as unknowns. To begin, we assume the solution to the heat equation (7.6) can be written as

$$u(x,t) = J_{[a,b]}[f](x,t) + D_a[\phi_a](x,t) + D_b[\phi_b](x,t),\ x \in [a,b] \tag{3.6}$$

Then, we take the limit as $x \to a^+$ and as $x \to b^-$ of our assumed solution shown in Eq. (3.6) to obtain the jump relation. The resulting limits get us the following:

$$\begin{cases} J_{[a,b]}[f](a,t) - \frac{1}{2}\phi_a(t) + \int_0^t H(a-b,t-\tau)\phi_b(\tau)d\tau = g_a(t) \\ J_{[a,b]}[f](b,t) - \frac{1}{2}\phi_b(t) + \int_0^t H(a-b,t-\tau)\phi_a(\tau)d\tau = g_b(t) \end{cases} \tag{3.7}$$

Rearranging further, we get

$$\begin{cases} \phi_a(t) - 2\int_0^t H(a-b,t-\tau)\phi_b(\tau)d\tau = -2g_a(t) + 2J_{[a,b]}[f](a,t) \\ \phi_b(t) - 2\int_0^t H(a-b,t-\tau)\phi_a(\tau)d\tau = -2g_b(t) + 2J_{[a,b]}[f](b,t) \end{cases} \tag{3.8}$$

At this point, we have rid ourselves of the spatial dependence, and now we only rely on time.

We now aim to solve for $\phi_a(t)$ and $\phi_b(t)$ so that we can evaluate the double-layer heat potential in our final solution of the heat equation (3.6).

4 Collocation Methods

We expand $\phi_a(t)$ and $\phi_b(t)$ over the interval $[0,t]$ using Chebyshev polynomials, $T_i(t)$, up to degree $p-1$. Let $\tilde{\tau}$ denote transformed values of τ to the interval $[-1,1]$. So, we rewrite Eq. (3.8) as

$$\begin{cases} \sum_{i=0}^{p-1} c_i T(\tilde{t_i}) + \sum_{i=0}^{p-1} d_i \Theta_i(t) \approx -2g_a(t) + 2J_{[a,b]}[f](a,t) \\ \sum_{i=0}^{p-1} d_i T(\tilde{t_i}) + \sum_{i=0}^{p-1} c_i \Theta_i(t) \approx -2g_b(t) + 2J_{[a,b]}[f](b,t) \end{cases} \tag{4.1}$$

such that

$$\Theta_i(t) = -2\int_0^t H(a-b, t-\tau) T_i(\tilde{\tau}) d\tau. \tag{4.2}$$

To solve for these coefficients, we construct a matrix equation. We find the p roots of the Chebyshev polynomial of degree $p-1$, denoted by $\tilde{t}_i$ for $i = 1, \ldots, p$. Then, we transform these roots to fit within our desired time interval, $[0,t]$, to get t_i. Now, we can evaluate the terms of Eq. (4.1) at each of these p nodes.

Let

$$\mathbf{T} = \begin{bmatrix} T_0(\tilde{t}_1) & T_1(\tilde{t}_1) & \ldots & T_{p-1}(\tilde{t}_1) \\ T_0(\tilde{t}_2) & T_1(\tilde{t}_2) & \ldots & T_{p-1}(\tilde{t}_2) \\ \vdots & \vdots & \ddots & \vdots \\ T_0(\tilde{t}_p) & T_1(\tilde{t}_p) & \ldots & T_{p-1}(\tilde{t}_p) \end{bmatrix}, \tag{4.3}$$

and let

$$\mathbf{\Theta} = \begin{bmatrix} \Theta_0(t_1) & \Theta_1(t_1) & \ldots & \Theta_{p-1}(t_1) \\ \Theta_0(t_2) & \Theta_1(t_2) & \ldots & \Theta_{p-1}(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \Theta_0(t_p) & \Theta_1(t_p) & \ldots & \Theta_{p-1}(t_p) \end{bmatrix} \tag{4.4}$$

Then, we can construct the final matrix

$$\mathbf{\Phi} = \begin{bmatrix} \mathbf{T} & \mathbf{\Theta} \\ \mathbf{\Theta} & \mathbf{T} \end{bmatrix}. \tag{4.5}$$

Let

$$h_a(t) = -2g_a(t) + 2J_{[a,b]}[f](a,t) \tag{4.6}$$

$$h_b(t) = -2g_b(t) + 2J_{[a,b]}[f](b,t) \tag{4.7}$$

as seen in Eq (4.1). Our next step is to evaluate $h_a(t)$ and $h_b(t)$ at each root, t_i. We place the

evaluations into the entries of a column vector, $\mathbf{h}$, such that

$$\mathbf{h} = \begin{bmatrix} h_a(t_1) \\ \vdots \\ h_a(t_p) \\ h_b(t_1) \\ \vdots \\ h_b(t_p) \end{bmatrix}. \tag{4.8}$$

We now have the final matrix equation,

$$\mathbf{\Phi x} = \mathbf{h}, \tag{4.9}$$

where $\mathbf{x}$ is a column vector with the entries being the coefficients c_i followed by d_i. Once we are able to solve for $\mathbf{x}$, we have our interpolants of $\phi_a(t)$ and $\phi_b(t)$ at these roots that we can plug into Eq. (3.6) to solve for $u(x,t)$.

5 Evaluation of Integrals

As such, our problem becomes how do we evaluate the integrals $J_{[a,b]}[f](x,t)$ and Θ_i(t) precisely and efficiently.

Observe the initial heat potential term (3.1). For small values of t, $K(x,t)$ creates sharp peaks in the curve we wish to evaluate. With our Θ_i integrals (4.2), we notice the H term (3.5) replicates this same behavior for small values of y. These sharp peaks hold important information needed in our evaluation of these integrals; however, Newton-Cotes quadrature methods are susceptible to missing these spikes and require more nodes for greater accuracy. Thus, we choose to implement an adaptive Gaussian quadrature method to evaluate both integral terms.

The adaptive portion of our method expands on composite methods of quadrature. Given the interval $[a,b]$, we first apply Gaussian quadrature over the entire interval to obtain an estimation, I_{tot}, for the evaluation of a given curve. Then, we split the interval into two equal sub-intervals. We repeat the method of Gaussian quadrature (2.6) on both sub-intervals to obtain two new estimations, I_1 and I_2. Given a certain tolerance, tol, we are satisfied with the initial evaluation over the entire interval given

$$|I_{\text{tot}} - (I_1 + I_2)| \leq \text{tol} \tag{5.1}$$

We avoid taking the relative error in the event that $I_{tot} = 0$. If the difference is greater than the tolerance, then we divide our interval further. We repeat the same process described above by applying it to our sub-intervals from the previous iteration. We split the sub-interval in half, evaluate the halves and then compare to the evaluation over the entire the sub-interval.

This method of quadrature allows us to accurately evaluate Eq. (3.1) and Eq. (4.2) to build our matrix equation (4.5) to solve for the coefficients of the Chebyshev interpolations of $\phi_a(t)$ and $\phi_b(t)$. We pass our block matrix through a LU-decomposer with partial pivoting to retrieve $\mathbf{P}$, $\mathbf{L}$, and $\mathbf{U}$. We may then pass these new matrices and vector $\mathbf{h}$ through a solver that completes forward and

backward substitution to solve for our coefficient vector **c**. These can be substituted into the double-layer heat potential. Then, we can evaluate the double-layer heat potential term after substitution using our adaptive Gaussian quadrature method as well as the initial heat potential term (3.1) as seen in the assumed solution (3.6).

6 Semi-group Property

To expand on the first method described above, we want to utilize the semi-group property of the heat equation. This property tells us that evolving an initial state for a time T_1 and then evolving the resulting state for a time T_2 is equivalent to evolving the initial state for a total time of T such that $T = T_1 + T_2$. Thus, we split our time interval into sub-intervals when choosing to evaluate at a certain t_f. This hopefully allows us to reduce runtime while maintaining accuracy.

To implement, we first observe our time interval $[0, t_f]$ and select the number of sub-intervals, s. We obtain a value

$$\Delta t = \frac{t_f}{s} \tag{6.1}$$

. We apply the first method described in the previous section, using the same boundary conditions as in Eq. (7.6), up to Δt to obtain a solution of $u(x, \Delta t)$. Note, this solution is only dependent on x. To reduce the number of integral evaluations and increase the speed of calculation, we create a Chebyshev interpolation with respect to x of this $u(x, \Delta t)$ using the process described in the Preliminaries section. The resulting interpolant, $\mu_{\Delta t}(x)$, is now the state that follows from evolving our initial system by a time Δt. Using the semi-group property, we may now use $\mu_{\Delta t}(x)$ as our new initial state. The resulting state creates the following novel boundary conditions:

$$\begin{cases} u_t^{(1)} - u_{xx}^{(1)} = 0, \ x \in [a, b] \\ u^{(1)}(x, 0) = \mu_{\Delta t}^{(0)}(x), \ x \in [a, b] \\ u^{(1)}(a, t) = g_a(t + \Delta t), \ u^{(1)}(b, t) = g_b(t + \Delta t) \\ a < b, \ t \geq 0 \end{cases} \tag{6.2}$$

Note, we must also adjust our spatial boundary conditions to account for the time evolution by shifting by a Δt. We repeat this process until we reach our final time, t_f. So, at each k-th iteration, we develop new boundary conditions

$$\begin{cases} u_t^{(k-1)} - u_{xx}^{(k-1)} = 0, \ x \in [a, b] \\ u^{(k-1)}(x, 0) = \mu_{\Delta t}^{(k-2)}(x), \ x \in [a, b] \\ u^{(k-1)}(a, t) = g_a(t + (k-1)\Delta t), \ u^{(k)}(b, t) = g_b(t + (k-1)\Delta t) \\ a < b, \ t \geq 0 \end{cases} \tag{6.3}$$

until we reach a total of s iteration

7 Numerical Experiments

To examine the efficacy of our methods, we construct three oscillatory solutions to the heat equation. We observe the precision of the numerically-obtained solution and the time it takes the program to run, comparing between the two methods. We evaluate each solution at $t = 0.6$.

Our first example is simply

$$u(x,t) = e^{-t}\sin(x) \tag{7.1}$$

over the interval $[0, 2\pi]$, which takes the boundary conditions

$$\begin{cases} u(x,0) = \sin(x) \\ u(0,t) = 0 \\ u(2\pi,t) = 0 \end{cases} \tag{7.2}$$

We apply the first method, without evaluating over sub-intervals of the time interval.

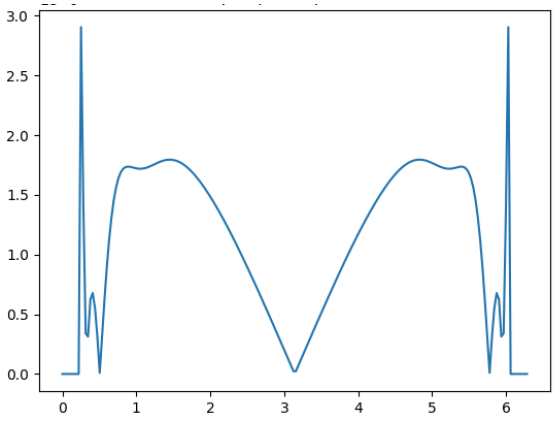

Figure 1: $e^{-t}\sin(x), p = 8$

Note: Scaling of y-axis is of order 10^{-5}.

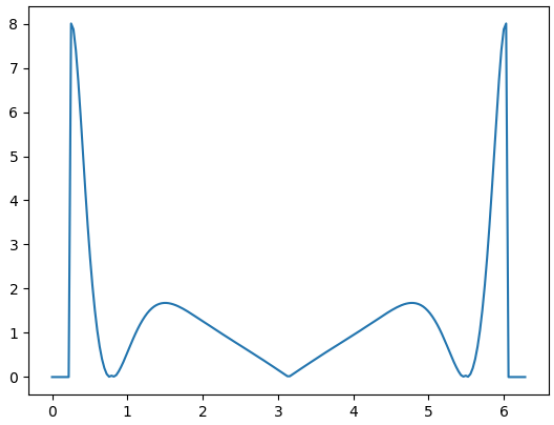

Figure 2: $e^{-t}\sin(x), p = 16$

Note: Scaling of y-axis is of order 10^{-6}.

We see that our initial solution is accurate to an order of 10^{-5} for $p = 8$. The time it took the program to carry out these evaluations along the interval $[0, 2\pi]$ is 3.47 seconds. For the second solution, we have an accuracy on the order of 10^{-6} with a time of 4.93. So, we note that the solver is producing solutions of good accuracy and that it is behaving expectantly (higher value of p implies greater accuracy).

Next, we evaluate using the semi-group property. Let s denote the number of sub-intervals we split the time interval into.

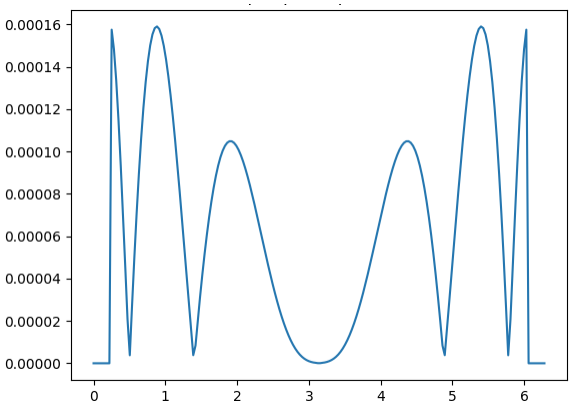

Figure 3: $e^{-t}\sin(x), p = 8,\ s = 4$

Note: Scaling of y-axis is of order 10^{-4}.

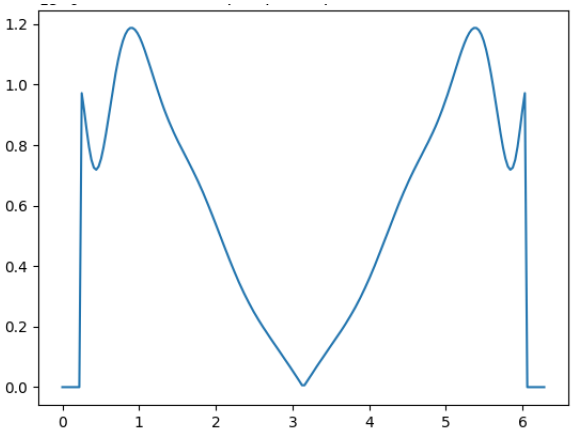

Figure 4: $e^{-t}\sin(x), p = 16,\ s = 4$

Note: Scaling of y-axis is of order 10^{-6}.

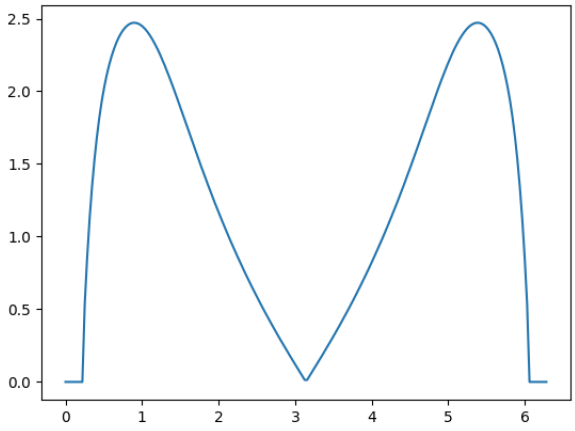

Figure 5: $e^{-t}\sin(x), p = 12,\ s = 4$

Note: Scaling of y-axis is of order 10^{-6}.

The times to complete the three figures in order was: 1.47 seconds, 8.49 seconds, 3.47 seconds. Once again, we see our accuracy and evaluation time increase for higher values of p. To compare across the methods, we see that the evaluation time decreases for $p = 8$ when we divide the time interval. However, the accuracy decreases. If we compare the cases of $p = 12$ and $s = 4$, we see greater accuracy and a comparable evaluation time to the solutions obtained in the first method.

Our second example is highly oscillatory with

$$u(x,t) = e^{-50\pi t}\sin(2500\pi^2 x) \tag{7.3}$$

over the interval $[0, 2\pi]$. This solution has similar boundary conditions as the previous example with

$$\begin{cases} u(x,0) = \sin(2500\pi^2 x) \\ u(0,t) = 0 \\ u(2\pi,t) = 0 \end{cases} \tag{7.4}$$

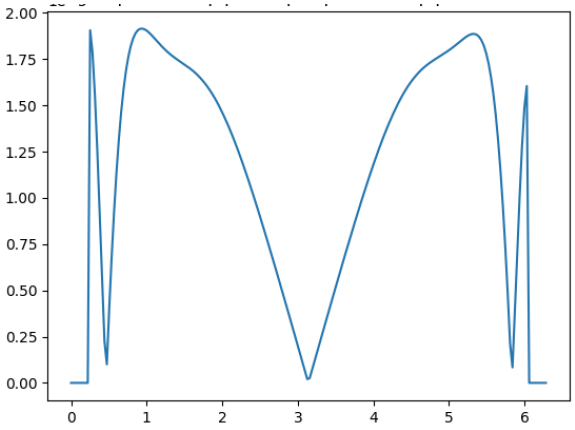

Figure 6: $e^{-50\pi t}\sin(2500\pi^2 x), p = 8$

Note: Scaling of y-axis is of order 10^{-5}.

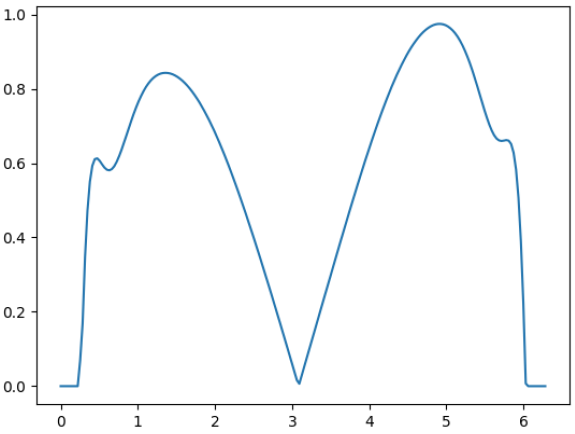

Figure 7: $e^{-50\pi t}\sin(2500\pi^2 x), p = 16$

Note: Scaling of y-axis is of order 10^{-5}.

When using the first method to create these figures, the evaluations took 4.62 seconds and 6.35 seconds respectively. We can see that the evaluation time increases as we pick a larger value of p. However, we see very little improvement of accuracy.

To use the second method, we get the following figures:

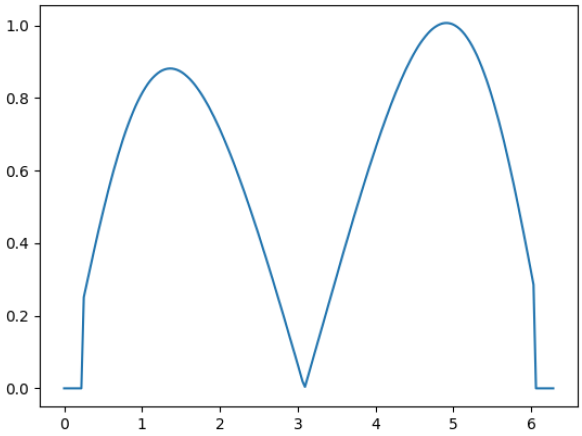

Figure 8: $e^{-50\pi t}\sin(2500\pi^2 x), p = 8, s = 4$

Note: Scaling of y-axis is of order 10^{-5}.

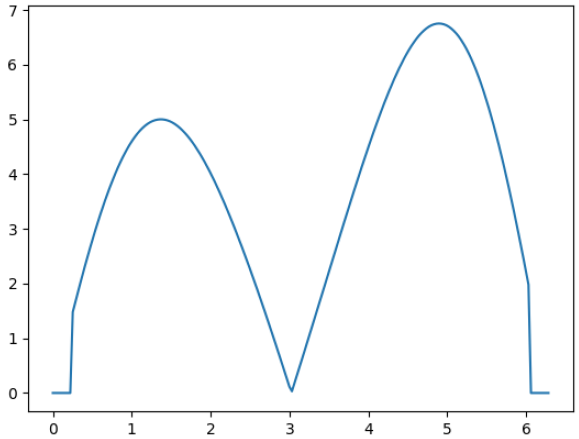

Figure 9: $e^{-50\pi t}\sin(2500\pi^2 x), p = 12, s = 4$

Note: Scaling of y-axis is of order 10^{-6}.

We can see that our first scenario of $p = 8$ and $s = 4$ gives us the same order of precision, but with half the evaluation time at 2.4 seconds. In fact, in a separate test with $p = 8$ and $s = 8$, we can increase our accuracy to the order of 10^{-6} with an evaluation time of only 2.83 seconds as shown below in Figure 10.

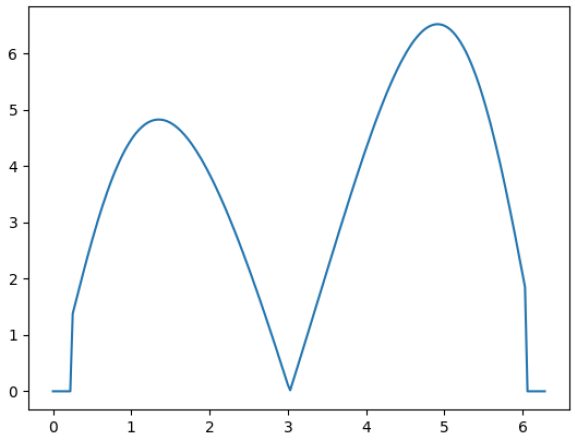

Figure 10: $e^{-50\pi t}\sin(2500\pi^2 x), p = 8, s = 8$

Note: Scaling of y-axis is of order 10^{-6}.

Our final example is

$$u(x,t) = e^{-50\pi t}\sin(2500\pi^2 x) + e^{-20\pi t}\cos(400\pi^2 x) \tag{7.5}$$

over the interval $[0, 2\pi]$. This solution has the boundary conditions

$$\begin{cases} u(x,0) = \sin(2500\pi^2 x) + \cos(400\pi^2 x) \\ u(0,t) = e^{-20\pi t} \\ u(2\pi,t) = e^{-20\pi t} \end{cases} \tag{7.6}$$

We again apply the first method, using $p = 8$ and $p = 16$, to numerically evaluate.

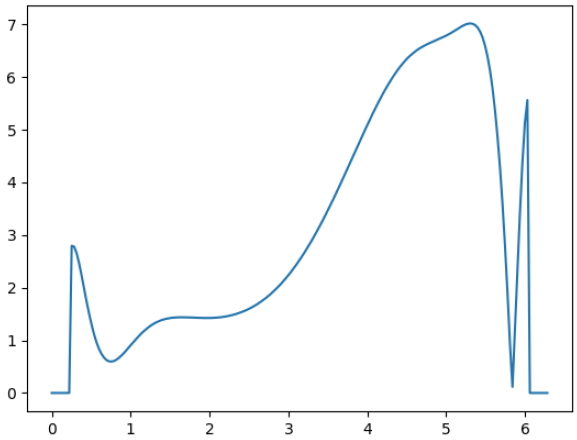

Figure 11: $e^{-50\pi t}\sin(2500\pi^2 x) + e^{-20\pi t}\cos(400\pi^2 x), p = 8$

Note: Scaling of y-axis is of order 10^{-5}.

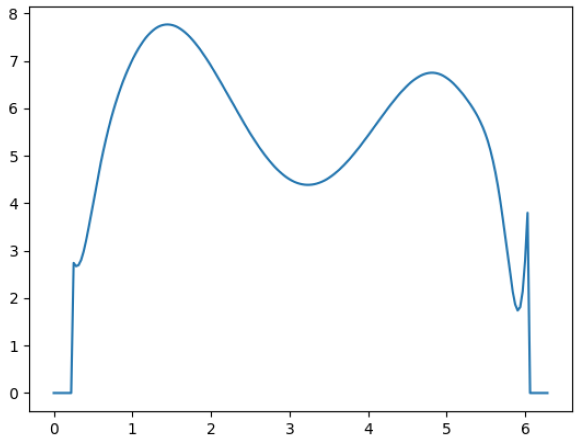

Figure 12: $e^{-50\pi t}\sin(2500\pi^2 x) + e^{-20\pi t}\cos(400\pi^2 x), p = 16$

Note: Scaling of y-axis is of order 10^{-5}.

We see that the accuracy of the two different values of p is of the same order; however, they have a difference in runtime. For $p = 8$, there is a runtime of 5.22 seconds. The runtime is 6.94 seconds for $p = 16$. When we compare this to our second method, we get the same order of accuracy for $p = 8$ and $s = 4$ but a shorter evaluation time of 2.74 seconds.

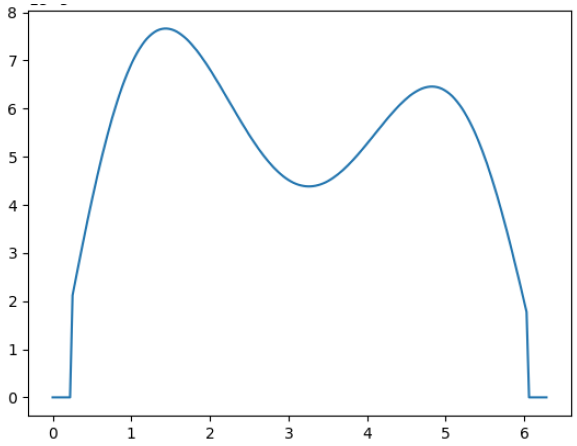

Figure 13: $e^{-50\pi t}\sin(2500\pi^2 x) + e^{-20\pi t}\cos(400\pi^2 x), p = 8, s = 4$

Note: Scaling of y-axis is of order 10^{-5}.

8 Conclusion

As it appears, our resulting process allows us to numerically solve for oscillatory solutions of the heat equation. Our method is accurate up to an order of 10^{-6} before taking too much runtime. We also note how we can reduce runtime and increase the precision of our solutions by using the semi-group property and Chebyshev interpolations to evaluate over sub-intervals of the time interval.

Going forward, we would like to develop faster methods while still maintaining accuracy. Additionally, using the methods outlined in this report, they can be further developed to apply to time-dependent spatial boundary conditions, $[a(t), b(t)]$. This would add additional integral terms to our system equations outlined in Eq (3.8).

9 References

Song, C., Wang, J. (2023). A fast adaptive method for the heat equation with moving or free boundaries in one dimension. arXiv. https://arxiv.org/abs/2310.15048