Numerical Approaches to the Heat Equation

Andrew Shea

April 14, 2025

Introduction

The heat equation is a fundamental partial differential equation (PDE) that describes how heat diffuses through a medium over time.

It appears in a wide range of fields:

- Physics (e.g., heat conduction, diffusion processes)
- Engineering (e.g., thermal analysis)
- Finance (e.g., Black-Scholes equation)

What We'll Cover

- Introduction to the problem and methods
- An analytical solution to our problem
- Derivation of Numerical Method
- Physics Informed Neural Networks (PINNs)
- Implementation of each method
- Results Comparisons
- Conclusions and insight from data

The 1D Heat Equation

The classical 1D heat equation (On a rod of length L):

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2}, \quad x \in [0, L],\ t \geq 09$$

Where:

- $u(x, t)$ is the temperature at position x and time t
- k is the thermal diffusivity constant

Initial Condition (IC):

$$u(x, 0) = f(x) \quad \text{(initial temperature distribution)}$$

Boundary Conditions (BCs):

$$u(0, t) = T_1, \quad u(L, t) = T_2 \quad \text{(Dirichlet BCs – fixed temperature at ends)}$$

Obtaining the 1D Analytical Solution

We will now solve the 1D heat equation under the follwing conditions:

$$\frac{\partial u(x,t)}{\partial t} = k\frac{\partial^2 u(x,t)}{\partial x^2}$$

Initial Condition (IC):

- $u(x,0) = \sin(\pi x)$
- The sine function was chosen as the initial condition because it will give a clear initial temperature distribution across our domain, satisfying our boundary conditions

Boundary Conditions (BCs):

- $u(0,t) = 0$ and $u(1,t) = 0$
- These are homogeneous Dirichlet boundary conditions, meaning the temperature at the ends of our domain are fixed at 0

Obtaining the 1D analytical solution

To solve the heat equation, we will use the method of **separation of variables**.
We assume the solution has the form:

$$u(x,t) = \phi(x)G(t)$$

Where $\phi(x)$ is a function of x and $G(t)$ is a function of t.
We can subsitute this into our heat equation to get the following:

$$\phi(x)\frac{dG(t)}{dt} = kG(t)\frac{d^2\phi(x)}{dx^2}$$

Obtaining the 1D analytical solution

Dividing both sides by $\phi(x)G(t)$ to separate the variables:

$$\frac{1}{kG(t)}\frac{dG(t)}{dt} = \frac{1}{\phi(x)}\frac{d^2\phi(x)}{dx^2}$$

Both sides are equal to a constant, which we denote as $-\lambda$.
This results in two ordinary differential equations (ODEs):

$$\frac{d^2\phi(x)}{dx^2} + \lambda\phi(x) = 0$$

$$\frac{dG(t)}{dt} = -k\lambda G(t)$$

Obtaining the 1D analytical solution

We begin with the ODE for $\phi(x)$:

$$\frac{d^2\phi(x)}{dx^2} + \lambda\phi(x) = 0$$

This standard 2nd Order ODE has the general solution:

$$\phi(x) = A\sin(\sqrt{\lambda}x) + B\cos(\sqrt{\lambda}x)$$

Applying the boundary conditions: $\phi(0) = 0$ and $\phi(1) = 0$ gives $\sin(\sqrt{\lambda}) = 0$

Thus, $\sqrt{\lambda} = n\pi$ where n is a positive integer.

So, the eigenvalues are:

$$\lambda_n = (n\pi)^2$$

And the corresponding eigenfunctions are:

$$\phi_n(x) = A_n\sin(n\pi x)$$

Obtaining the 1D analytical solution

We now solve the ODE for $G(t)$:

$$\frac{dG(t)}{dt} = -k\lambda G(t)$$

This is a simple first-order linear differential equation. The solution is of the form:

$$G(t) = Ce^{-k\lambda t}$$

Now, we substitute the eigenvalue $\lambda_n = (n\pi)^2$ into this equation:

$$G_n(t) = C_n e^{-k(n\pi)^2 t}$$

So, the time-dependent part of the solution for each n is:

$$G_n(t) = C_n e^{-kn^2\pi^2 t}$$

Obtaining the 1D analytical Solution

Now that we've solved both ODEs, the full solution is:

$$u(x,t) = \sum_{n=1}^{\infty} A_n \sin(n\pi x) e^{-k(n\pi)^2 t}$$

The constants A_n are determined using the initial condition:

$$u(x,0) = \sum_{n=1}^{\infty} A_n \sin(n\pi x) = \sin(\pi x)$$

This is a Fourier sine series, giving the following A values:

$$A_1 = 1, \quad A_n = 0 \text{ for } n \neq 1$$

Final Analytical Solution

Since only $A_1 = 1$ and all other $A_n = 0$, the infinite sum reduces to a single term.
The final solution is:

$$u(x, t) = \sin(\pi x)\, e^{-k\pi^2 t}$$

This function satisfies:

- The heat equation
- The boundary conditions: $u(0, t) = u(1, t) = 0$
- The initial condition: $u(x, 0) = \sin(\pi x)$

Numerical Approach: Crank-Nicolson Method

We now turn to a numerical method for solving the 1D heat equation:

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2}$$

We will use the **Crank-Nicolson method**, which is:

- Second-order accurate in both time and space.
- Unconditionally stable for linear problems.

It works by averaging the Forward Euler and Backward Euler methods, making it a balanced, more accurate approach

Crank-Nicolson Scheme Derivation

We start by discretizing the domain into a spacial grid and a time grid:

$$\textit{Space Grid}: \; x_i = i\Delta x, \; \textit{for } i = 0, 1, 2, 3...$$

$$\textit{Time Grid}: \; t^n = n\Delta t, \; \textit{for } n = 0, 1, 2, 3...$$

$$\textit{This gives}: \; u_i^n \approx u(x_i, t^n)$$

Crank-Nicolson Scheme Derivation

Next, we discretize the time derivative

$$\frac{\partial u}{\partial t}(x_i, t^{n+\frac{1}{2}}) \approx \frac{u_i^{n+1} - u_i^n}{\Delta t}$$

And then we average the central difference for space at time step n and $n+1$

$$\frac{\partial^2 u}{\partial x^2}(x_i, t^{n+\frac{1}{2}}) \approx \frac{1}{2} \left. \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} + \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2} \right)$$

Crank-Nicolson Scheme Derivation

Now we can plug each of our approximations into the heat equation:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = k \cdot \frac{1}{2} \left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} + \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2} \right)$$

Next we multiply by Δt and factor $(\Delta x)^2$

$$u_i^{n+1} - u_i^n = \frac{k\,\Delta t}{2(\Delta x)^2} \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n + u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1} \right)$$

Crank-Nicolson Scheme Derivation

To simplify things we will let $s = \frac{k\Delta t}{(\Delta x)^2}$ to give us this:

$$u_i^{n+1} - u_i^n = \frac{s}{2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n + u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

And lastly, we can rearrange and simplify our equation with all of our $n+1$ terms on the left.

$$(1+s)u_i^{n+1} - 2s\, u_{i+1}^{n+1} - 2s\, u_{i-1}^{n+1} = (1-s)u_i^n + 2s\, u_{i+1}^n + 2s\, u_{i-1}^n$$

Transition from Single Equation to System of Equations

In the Crank-Nicolson scheme, we have a single equation for each grid point in the spatial domain. We can apply this equation at every grid point in the domain, resulting in a set of equations, one for each spatial point. Thus, the system of equations becomes:

$$\begin{aligned}
u(x_1, t+\Delta t) &= \text{function of known values at time t} \\
u(x_2, t+\Delta t) &= \text{function of known values at time t} \\
&\vdots \\
u(x_N, t+\Delta t) &= \text{function of known values at time t}
\end{aligned}$$

This forms a linear system for all unknowns at $t + \Delta t$.

Solving each system of equations

Each time step involves solving a linear system to find the values of $u(x, t)$ at every spatial point x for that specific time.

- The size of each system corresponds to the number of spatial points N_x in the discretized domain.
- At each time step t_n, the linear system has N_x unknowns, representing the values of $u(x, t_n)$ for each x.
- The system is solved for each t_n, yielding a vector $\mathbf{u}^n$ of length N_x.

Once the system is solved at each time step, the solutions are pieced together:

$$\mathbf{u}(x, t_0), \mathbf{u}(x, t_1), \ldots, \mathbf{u}(x, t_{N_t})$$

In this way, the numerical solution is built layer by layer, starting from the initial condition and iterating forward in time.

Introduction to PINNs

Physics-Informed Neural Networks (PINNs) are a type of machine learning approach to solving equations.

- PINNs utilize neural networks and governing (PDEs) by embedding the physics directly into the network's training process.
- PINNs learn continuous solutions that satisfy the PDE, boundary, and initial conditions.

Introduction to PINNs

Physics-Informed Neural Networks (PINNs) are a type of machine learning approach to solving equations.

- PINNs utilize neural networks and governing (PDEs) by embedding the physics directly into the network's training process.
- PINNs learn continuous solutions that satisfy the PDE, boundary, and initial conditions.

Key Components of PINNs:

- **Neural Network**: A feed-forward neural network that takes space and time coordinates as inputs, then outputs the solution
- **Loss Function**: A combination of:
  - **Physics-based loss**: Penalizes the network for not satisfying the PDE.
  - **Boundary/Initial Condition loss**: Ensures the network adheres to the boundary or initial conditions.

Introduction to PINNs

Physics-Informed Neural Networks (PINNs) are a type of machine learning approach to solving equations.

- PINNs utilize neural networks and governing (PDEs) by embedding the physics directly into the network's training process.
- PINNs learn continuous solutions that satisfy the PDE, boundary, and initial conditions.

Key Components of PINNs:

- **Neural Network**: A feed-forward neural network that takes space and time coordinates as inputs, then outputs the solution
- **Loss Function**: A combination of:
  - **Physics-based loss**: Penalizes the network for not satisfying the PDE.
  - **Boundary/Initial Condition loss**: Ensures the network adheres to the boundary or initial conditions.

Why PINNs?

- They can solve complex PDEs without discretizing the domain.
- They handle higher dimensions better
- They can work inversely

Neural Network Architecture in PINNs

The PINN uses a neural network with 3 parts to learn the solution

- **Input Layer**: The network receives spatial (x) and temporal (t) coordinates as input. These represent the points in the domain of the PDE.

Neural Network Architecture in PINNs

The PINN uses a neural network with 3 parts to learn the solution

- **Input Layer**: The network receives spatial (x) and temporal (t) coordinates as input. These represent the points in the domain of the PDE.
- **Hidden Layers**: The network has one or more hidden layers made of nodes (neurons) that process the information. This is where the learning and adjustments of the NN are done.

Neural Network Architecture in PINNs

The PINN uses a neural network with 3 parts to learn the solution

- **Input Layer**: The network receives spatial (x) and temporal (t) coordinates as input. These represent the points in the domain of the PDE.
- **Hidden Layers**: The network has one or more hidden layers made of nodes (neurons) that process the information. This is where the learning and adjustments of the NN are done.
- **Output Layer**: The final layer of the network outputs the solution to the PDE at the given coordinates x and t.

The goal of the network is to approximate the solution of the PDE without explicitly solving it through traditional methods.

Loss Function in PINNs

In a PINN, a **loss function** is used to train and penalize the neural network

- **What is Loss?** The loss is a measure of error. It tells us how far the network's predicted solution is from the true solution.

Loss Function in PINNs

In a PINN, a **loss function** is used to train and penalize the neural network

- **What is Loss?** The loss is a measure of error. It tells us how far the network's predicted solution is from the true solution.
- **PDE Loss**: This part of the loss function ensures that the network's output satisfies the governing equation (PDE).

Loss Function in PINNs

In a PINN, a **loss function** is used to train and penalize the neural network

- **What is Loss?** The loss is a measure of error. It tells us how far the network's predicted solution is from the true solution.
- **PDE Loss**: This part of the loss function ensures that the network's output satisfies the governing equation (PDE).
- **Boundary Condition Loss**: This ensures that the network respects the boundary conditions (values at the edges of the domain).

Loss Function in PINNs

In a PINN, a **loss function** is used to train and penalize the neural network

- **What is Loss?** The loss is a measure of error. It tells us how far the network's predicted solution is from the true solution.
- **PDE Loss**: This part of the loss function ensures that the network's output satisfies the governing equation (PDE).
- **Boundary Condition Loss**: This ensures that the network respects the boundary conditions (values at the edges of the domain).
- **Initial Condition Loss**: The network is also penalized if it doesn't match the initial condition (the solution at the start of the process).

Training the PINN (Data Generation & Loss Function)

The PINN is trained by optimizing the loss function.

- **Step 1: Data Generation**
  - The input data consists of random samples for spatial x and temporal t values.
  - These points are fed into the network to generate predictions for the solution $u(x, t)$.
- **Step 2: Loss Function Calculation**
  - The loss function combines multiple components, each weighted differently:
    - **Physics-based loss**: Ensures the network satisfies the PDE by penalizing deviations from the PDE.
    - **Boundary Condition loss**: Enforces the correct values at the boundaries of the spatial domain.
    - **Initial Condition loss**: Ensures the solution is correct at $t = 0$.

Training the PINN (Backpropagation & Repetition)

- **Step 3: Backpropagation and Optimization**
  - Using the computed loss, backpropagation is performed to penalize the neural network
  - The optimizer adjusts the weights and biases of the network to minimize the total loss.
- **Step 4: Repeat**
  - This process is repeated over multiple training iterations, or "epochs" to improve the model's accuracy.

Crank-Nicolson Implementation Details

Simulation Parameters

- Domain: $x \in [0, 1]$, $t \in [0, 1]$
- Diffusivity coefficient: $k = 0.1$
- Initial condition: $u(x, 0) = \sin(\pi x)$
- Boundary conditions: $u(0, t) = u(1, t) = 0$

Discretization Details

- Spatial steps: $N_x = 400 \quad (\Delta x = 0.0025)$
- Time steps: $N_t = 20000 \quad (\Delta t = 0.00005)$
- Total Points: 8,000,000
- Central difference in space, trapezoidal rule in time

Computational Methods

- Solved for every interior point iteratively, then graphed
- Utilized the structure of matrices produced from CN scheme to speed up computation

PINN Methodology and Setup

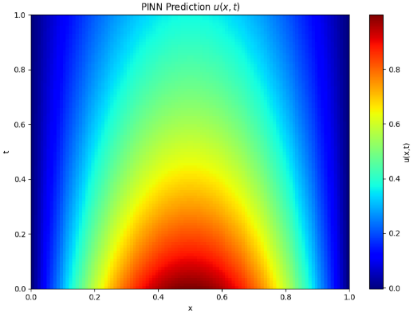
Simulation Parameters

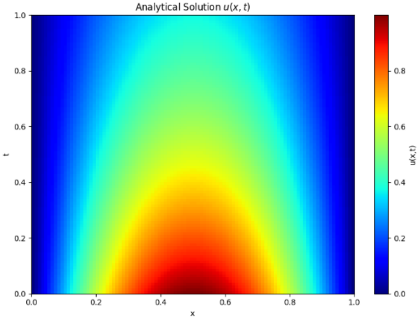
- Domain: $x \in [0, 1]$, $t \in [0, 1]$
- Diffusivity coefficient: $k = 0.1$
- Initial condition: $u(x, 0) = \sin(\pi x)$
- Boundary conditions: $u(0, t) = u(1, t) = 0$

Training Parameters

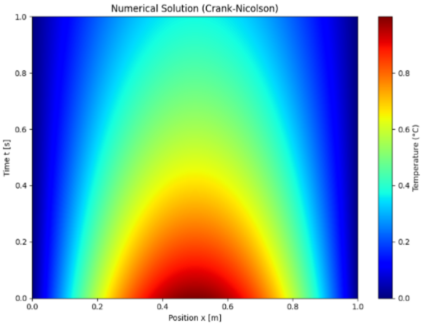
- **Number of Training Points**: 2000 randomly sampled points within the spatial and temporal domains
- **Epochs**: 15000 epochs for training
- **Weights**: PDE = 10, IC = 1, BC = 1
- **Learning Rate**: .001

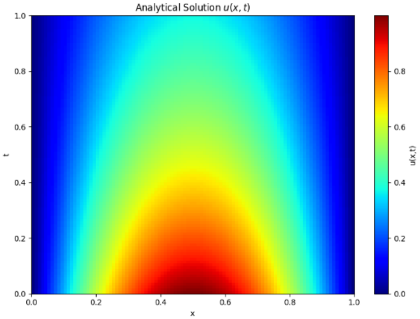
PINN vs Analytical Solution



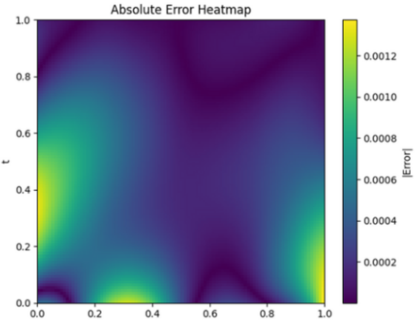


Crank-Nicolson vs Analytical Solution

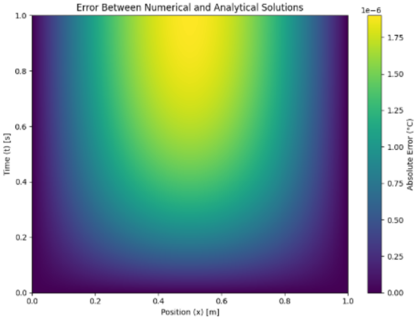




PINN Error vs Crank Nicolson Error




```
_____PINN Metrics_____
Maximum Absolute Error (Max Error): 0.001374
Mean Squared Error (MSE): 0.000000
Mean Absolute Error (MAE): 0.000333
```




```
_____Crank Nicolson Metrics_____
Max Error: 0.000002 °C
Mean Squared Error (MSE): 0.000000 °C²
Mean Absolute Error (MAE): 0.000001 °C
```

Conclusions

- Both the PINN and Numerical Method did an exceptional job at approximating the solution
- Overall, the numerical methods error metrics were better than the PINNs
- The numerical method was implemented more efficiently, and both were ran on the CPU
- The error maps show clear patterns for the numerical method, and unpredictable patterns for the PINN
- The PINN graphed better than the numerical method because of discretization