# Spectral Methods for Partial Differential Equations

Elijan Jakob Mastnak
Student ID: 28181157

December 2020

## Contents

**Assignment**

1. Use the Fourier spectral method to solve the heat diffusion equation in one dimension for $x \in [0, L]$ using the initial temperature distribution

$$T(x, 0) \propto \exp\left[-\frac{\left(x - \frac{L}{2}\right)^2}{\sigma^2}\right]$$

   and each of the following two sets of boundary conditions:

   (a) periodic boundary conditions $T(0, t) = T(L, t)$

   (b) homogeneous Dirichlet boundary conditions $T(0, t) = T(L, t) = 0$

2. Solve the same problem—with homogeneous boundary—using B-splines and the collocation method. Compare the results with the Fourier spectral method.

3. *Optional:* Use both methods to solve the problem with a different initial temperature distribution, for example the sinusoidal distribution

$$T(x, 0) = \sin\left(\frac{\pi x}{L}\right)$$

---

# 1 Theory

*To jump right to the solution, see Section 2.*

This report involves solving the one-dimensional heat diffusion equation

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2}, \quad x \in [0, a]$$

where $D$ is the heat diffusion coefficient.

## 1.1 Fourier Series Solution

To solve the heat equation with the Fourier method, we write the temperature $T(x, t)$ at a given position and time $(x, t)$ using the Fourier series ansatz

$$T(x, t) = \sum_{k=0}^{N-1} c_k(t) e^{-2\pi i f_k x}$$

where $f_k = \frac{k}{L}$. Inserting this expression into the heat equation produces

$$\sum_k \dot{c}_k(t) e^{-2\pi i f_k x} = D \sum_k (-4\pi^2 f_k^2) c_k(t) e^{-2\pi i f_k x}$$

from which we get an expression for the coefficients $c_k(t)$:

$$\dot{c}_k(t) = -4\pi^2 D f_k^2 c_k(t)$$

The problem thus reduces to finding the coefficients $c_k(t)$. Because of our convenient choice of a Fourier series ansatz, the initial coefficients $C_k \equiv c_k(t=0)$ are a Fourier transform of the initial condition $T(x,0)$. We can find $c_k(t)$ with the analytic solution

$$c_k(t) = C_k e^{-4\pi^2 f_k^2 D t} \tag{1}$$

where $C_k$ are the initial coefficients at $t = 0$. More generally, we could find the $c_k(t)$ with an an integration scheme, e.g. Euler's method

$$c_k(t+h) = c_k(t) - 4\pi^2 D f_k^2 c_k(t) \cdot h \tag{2}$$

When solving for the coefficients $c_k(t)$ numerically, the step size must satisfy

$$\left| \frac{c_k(t+h)}{c_k(t)} \right| = \left| 1 - 4\pi^2 D f_k^2 \cdot h \right| < 1$$

### 1.2 Collocation Method using Cubic B-Splines

To solve the heat equation with the collocation method, we use the ansatz

$$T(x,t) = \sum_{k=-1}^{N+1} c_k(t) B_k(x)$$

where $B_k(x)$ is a cubic B-spline centered at $x = x_k$ (see Appendix A). We split the $x$ interval $[0, L]$ into $N$ collocation points $\{x_j\}$ separated by the uniform step size $h$, so that

$$x_j = \frac{j}{N} L = jh, \qquad j = 0, 1, \ldots, N$$

Substituting the spline ansatz into the heat equation and evaluating the result at $x = x_j$ produces

$$\sum_{k=-1}^{N+1} \dot{c}_k(t) B_k(x_j) = D \sum_{k=-1}^{N+1} c_k(t) \frac{\partial^2 B_k(x_j)}{\partial x^2}, \qquad j = 0, 1, \ldots N$$

which, using a finite difference approximation of the second position derivative, leads to the system of equations

$$\dot{c}_{j-1}(t) + 4\dot{c}_j(t) + \dot{c}_{j+1}(t) = \frac{6D}{h^2}\big[c_{j-1}(t) - 2c_j(t) + c_{j+1}(t)\big], \qquad j = 0, 1, \ldots, N$$

The homogeneous boundary condition $T(0,t) = 0$ gives $c_{-1} = -4c_0 - c_1$. Using the $B$-splines with the natural spline condition, which requires

$$\sum_{k=-1}^{n+1} c_k(t) B_k''(x)\big|_{x=0,L} = 0,$$

we get the equality $c_0 = c_n = 0$ and thus $c_{-1} = -c_1$ at the left endpoint. Analogously, the natural spline condition applied to the right endpoint produces $c_{n+1} = -c_{n-1}$. Solving the spline ansatz for $T(x,t)$ reduces to solving the matrix equation

$$\mathbf{A}\dot{\boldsymbol{c}} = \mathbf{B}\boldsymbol{c}$$

where

$$\mathbf{A} = \begin{bmatrix} 4 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & & \ddots & & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}, \qquad \mathbf{B} = \frac{6D}{h^2} \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & & \ddots & & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix} \tag{3}$$

and $\boldsymbol{c} = \boldsymbol{c}(t) = \big[c_1(t), \dots c_{n-1}(t)\big]^T$. The initial condition $T(x_j, 0) = f(x_j)$ reads

$$\mathbf{A}\boldsymbol{c}^{(0)} = \boldsymbol{f}$$

where $\boldsymbol{f} = \big[f(x_1), \dots, f(x_{n-1})\big]^T$. We find the coefficients $\boldsymbol{c}$ at times $t_i = ih_t$ with

$$\boldsymbol{c}^{(i+1)} = \boldsymbol{c}^{(i)} + h_t\mathbf{A}^{-1}\mathbf{B}\boldsymbol{c}^{(i)} = (\mathbf{I} + h_t\mathbf{A}^{-1}\mathbf{B})\boldsymbol{c}^{(n)} \tag{4}$$

This is essentially an explicit Euler scheme, and is potentially unstable. We are better off with the implicit, Crank-Nicholson-style scheme

$$\left(\mathbf{A} - \frac{h_t}{2}\mathbf{B}\right)\boldsymbol{c}^{(i+1)} = \left(\mathbf{A} + \frac{h_t}{2}\mathbf{B}\right)\boldsymbol{c}^{(i)} \tag{5}$$

We then find $T(x_j, t_i)$ with the known $\boldsymbol{c}^{(i)}$ using

$$\boldsymbol{T} = \mathbf{A}\boldsymbol{c}(t_i) \tag{6}$$

where $\boldsymbol{T} = \big[T(x_1, t_i), \dots, T(x_{n-1}, t_i)\big]^T$

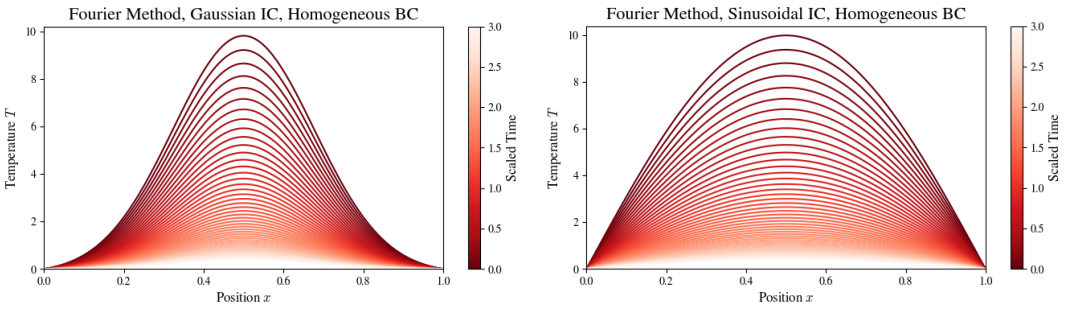



Figure 1: Heat diffusion in a one-dimensional rod with homogeneous boundary conditions and a Gaussian and sinusoidal initial temperature distribution. Found with the Fourier method.

# 2 Fourier Solution

## 2.1 Initialization Steps

For the Fourier method, I first define the simulation parameters $L$, $D$, $T_0$ and $\sigma$, which are the rod length; diffusion constant; and amplitude and standard deviation of

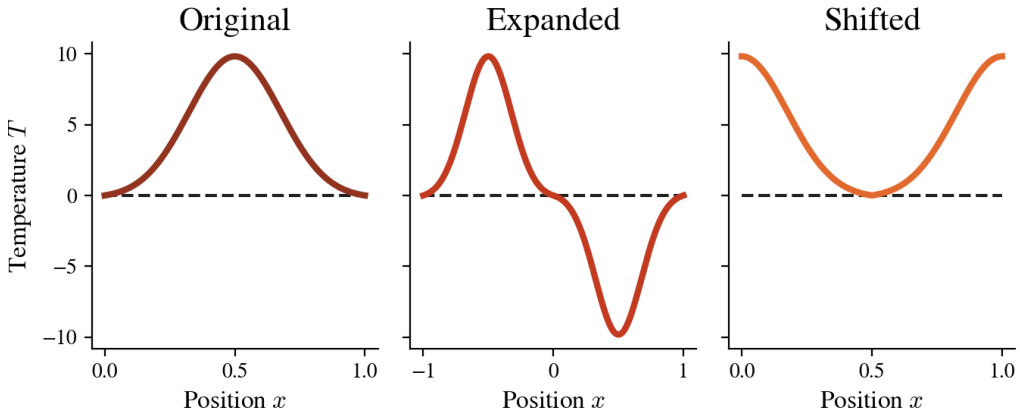



Figure 2: Modifications to the initial temperature distribution (left), needed to satisfy the homogeneous (middle) and periodic (right) boundary conditions used with the Fourier method.

the initial Gaussian temperature distribution, respectively. Next, I define a discrete grid of position and time values on which to find the solution and generate the initial temperature distribution—note the vertical translation of the initial Gaussian distribution to $T = 0$ to satisfy the homogeneous boundary conditions.
The following Python code shows the initialization procedure with some representative values of $L$, $D$, $T_0$ and $\sigma$.

```
""" Initialization steps for the Fourier method """
import numpy as np
# peak temperature, rod length, diffusion constant, gaussian std. dev.
T0, L, D, sigma = 10, 1.0, 0.1, 0.25
N_x = 2**10  # power of two for use with fft
x = np.linspace(0, L, N_x)
T = gauss(x, T0, L/2, sigma)   # local utility function returning gauss curve
T -= gauss(0, T0, L/2, sigma)  # for agreement of IC and homogeneous BC
```

## 2.2 Homogeneous Boundary Conditions

I implemented the Fourier method with homogeneous boundary conditions as follows:

- Create an asymmetric (odd) expansion of initial distribution about the origin, shown in the middle graph of Figure 2. This step, which creates a "periodic" initial distribution with both endpoints at zero, means the later use of the discrete Fourier transform will preserve the homogeneous boundary conditions.
- Find the initial Fourier coefficients with a Fourier transform of the initial temperature distribution, as described in Subsection 1.1.
- Propagate the initial coefficients through time. After experimenting with both the analytic solution in Equation 1 and the Euler scheme in Equation 2, I opted for the simpler analytic solution for the remainder of the report. Note that the coefficients $c_k$ are in general complex numbers, which requires the use of complex math.

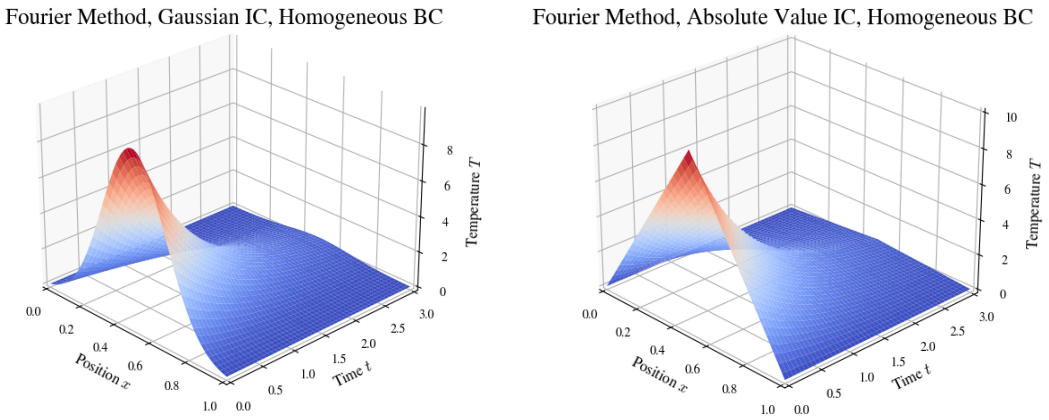



Figure 3: Heat diffusion in a 1D rod with homogeneous conditions and a Gaussian and absolute-value initial temperature distribution. Found with the Fourier method.

- Find the temperature distribution $T(x,t)$ with an inverse Fourier transform of the time-dependent coefficients. Since the original distribution was artificially doubled in length in the asymmetric expansion, the desired solution is only the first half of the inverse transform, as shown in the following code block.

```
from numpy.fft import fft, ifft, fftshift, ifftshift
c0 = fft(T)  # coefficients are DFT of initial temperature distribution
T_solutions = np.zeros((N_x, N_t))        # matrix of T(x) at each time t
for i, t in enumerate(times):
    c = get_ck_homogeneous(t, c0, L, D)  # find coefficients c_k(t)
    T = np.real(ifft(c))[:N_x]           # keep only first N_x points
    T_grid[:, i] = T                     # save solution T(x, t)
```

The coefficient-finding function `get_ck_homogeneous`, a straightforward implementation of Equation 1, is below. Note the use of "frequencies" $f_k$ within the Nyquist range $\left[-\frac{1}{2}\cdot\frac{N_x}{2L}, \frac{1}{2}\cdot\frac{N_x}{2L}\right]$—the extra $\frac{1}{2}$ factor accounts for doubling the initial temperature distribution in the asymmetric expansion.

```
def get_ck_homogeneous(t, c0, L, D):
    """ Returns Fourier coefficients c_k(t) at time t for homogeneous BC """
    fc = 0.5*len(c0)/(2*L)  # analog of a Nyquist frequency for x
    f_k = np.linspace(0, len(c0)-1, len(c0))/(2*L) - fc
    c_k = c0 * np.exp(-4*(np.pi**2)*(f_k**2)*D*t)  # uses analytic solution
    return c_k
```

Figures 1 and 3 show two and three-dimensional representations, respectively, of the changing temperature distribution in a one-dimensional rod for various initial temperature distributions and homogeneous boundary conditions.

## 2.3 Periodic Boundary Conditions

The Fourier method solution with periodic boundary condition requires two changes from the homogeneous solution, first when modifying the initial solution and second when calculating the coefficients $c_k(t)$. These are:

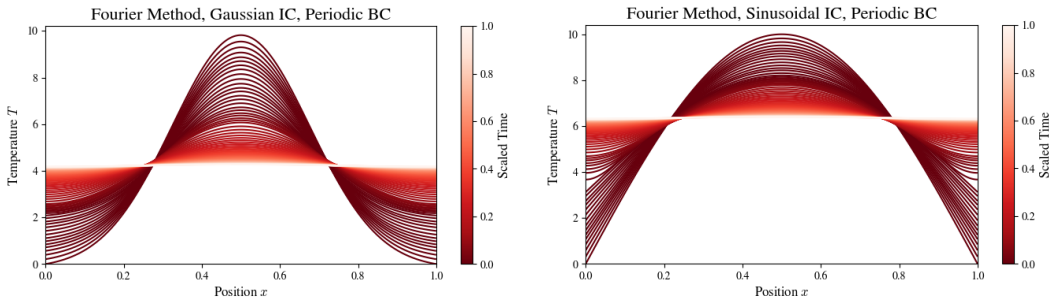



Figure 4: Heat diffusion in a 1D rod with periodic conditions and a Gaussian and sinusoidal initial temperature distribution. Found with the Fourier method. Note that the initial curves are closer spaced in time than the later ones to more uniformly sample the initially rapid exponential temperature decay.

1. Shift the initial temperature distribution with e.g. Numpy's `fftshift` or a similar function, as shown in the third graph of Figure 2. This step, just like in the *Discrete Fourier Transform* report, makes the initial condition "periodic", which is necessary for successful subsequent application of the DFT.

   The earlier line `T = np.concatenate((T, -T[::-1]))` would be replaced by

   ```
   T = np.fftshift(T)  # make distribution periodic for use with DFT
   ```

2. Use only positive frequencies $f_k$ when propagating the coefficients $c_k$ through time. I discovered the need for positive frequencies through experiment rather than an *a priori* theoretical prediction—I found using both negative and positive frequencies results in homogeneous boundary conditions.

The Fourier coefficients $c_k(t)$ for periodic boundary condition are found as follows:

```
def get_ck_periodic(t, c0, L, D):
    """ Returns Fourier coefficients c_k(t) at time t for periodic BC """
    f_k = np.linspace(0, len(c0)-1, len(c0))/(L)  # only positive frequencies
    c_k = c0 * np.exp(-4*(np.pi**2)*(f_k**2)*D*t)
    return c_k
```

Figures 4 and 5 show two and three-dimensional representations, respectively, of the changing temperature distribution in a one-dimensional rod for various initial temperature distributions periodic boundary conditions. Note that the area under the temperature distribution curve is constant throughout the simulations, corresponding to conservation of energy. This is best seen in the 3D representations in Figure 5, where the initial temperature falls between $T = 0$ and $T = 10$ and decays over time to the average value $T = 5$.

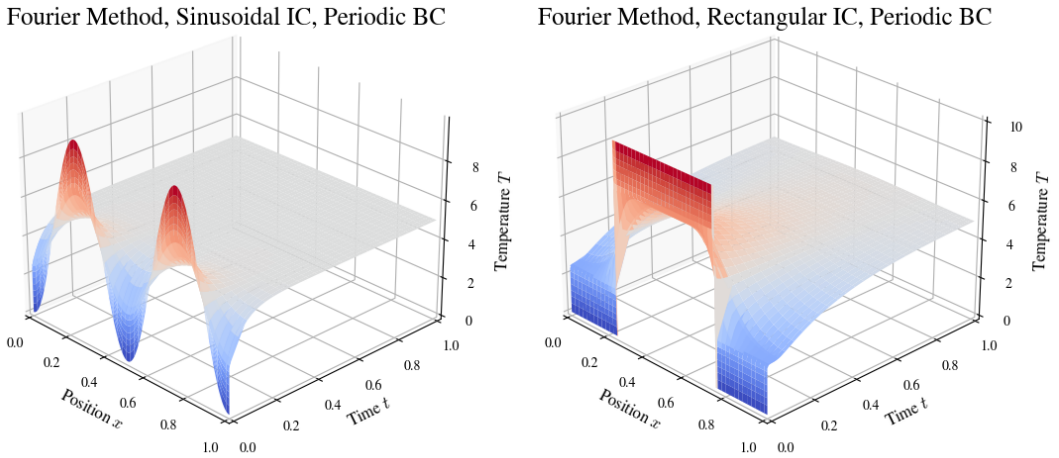



Figure 5: Heat diffusion in a 1D rod with periodic boundary conditions and a sinusoidal and rectangular initial distribution. Found with the Fourier method. Note the decay to the average value $T = 5$, which preserves the area under the temperature distribution curve in agreement with energy conservation.

# 3 Collocation Method with Cubic B-Splines

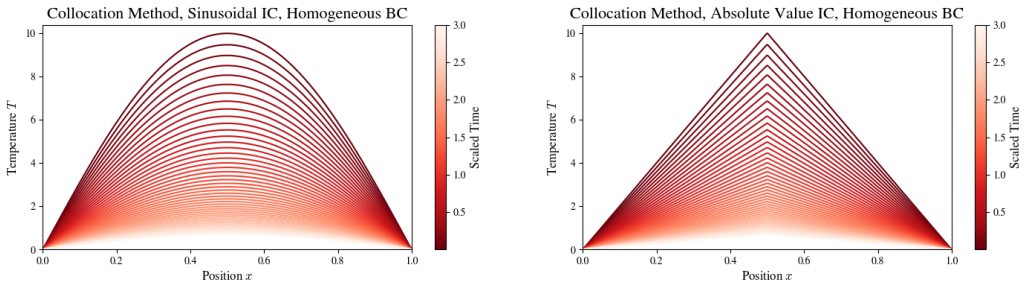
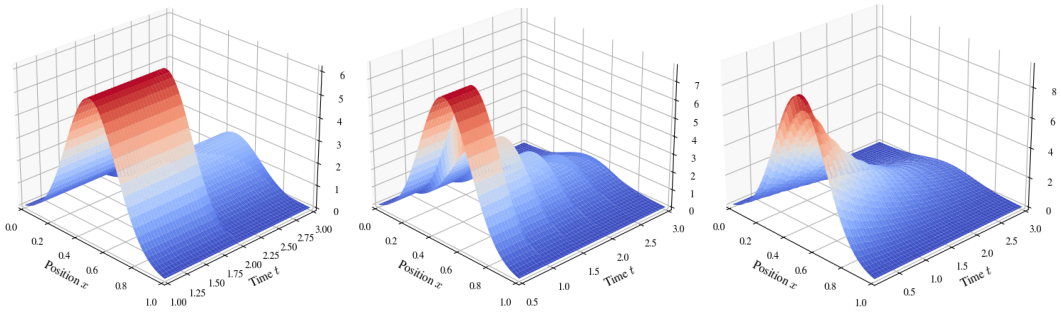



Figure 6: Heat diffusion in a 1D rod with periodic boundary conditions and a sinusoidal and absolute-value initial distribution. Found with the collocation method.

I tested the collocation method using both the explicit and implicit schemes in Equations 4 and 5, respectively. I found the methods performed similarly, both giving reasonable results for time steps smaller than $10^{-3}$, and progressively "choppier" results for larger time steps, as shown in Figure 7. Since both methods, at least on a macroscopic scale, produce visually identical results, I have not differentiated between the two in the titles of this section's figures.

## 3.1 Initialization Steps

I define the simulation parameters and position and time grids analogously to the Fourier method, as shown for the sake of completeness in the code block below.




(a) $h_t = 0.1$ (b) $h_t = 0.05$ (c) $h_t = 0.01$

Figure 7: Progressively "choppier" collocation method solution for large time steps $h_t$ with implicit coefficient calculation. Explicit calculation behaves analogously.

```
""" Initialization steps for the collocation methods """
T0, L, D, sigma = 10, 1.0, 0.1, 0.25
N_x = 500  # number of points in the x partition
x = np.linspace(0, L, N_x)       # N_x points spanning rod length
dx = x[1] - x[0]                 # position step size
tmin, tmax, dt = 0.0, 1.0, 1e-4  # min time, max tie, time step
N_t = (tmax - tmin) / dt         # points in time grid
N_solutions = 50  # number of times to sample solution T(x, t)
t_indices = np.linspace(1, N_t, N_solutions)  # indices of solution samples
t = t_indices * dt                            # convert indices to times
```

Initializing the matrices and vectors discussed in Subsection 1.2 requires some attention to detail regarding matrix dimensions. For example, for a position grid of $N_x$ points, the matrices $\mathbf{A}$ and $\mathbf{B}$ have dimensions $(N_x - 2) \times (N_x - 2)$, not $N_x \times N_x$. I initialized the matrices $\mathbf{A}$ and $\mathbf{B}$ with a direct implementation of Equation 3.

```
A = np.diag(4.0*np.ones(N_x-2), k=0) + np.diag(np.ones(N_x-3), k=1) +
↪  np.diag(np.ones(N_x-3), k=-1)  # Nx-2 by Nx-2
A_inv = np.linalg.inv(A)  # inverse of A
B = 6.0 * D/(dx**2) * np.diag(-2.0*np.ones(N_x-2), k=0) +
↪  np.diag(np.ones(N_x-3), k=1) + np.diag(np.ones(N_x-3), k=-1)
```

### 3.2 Explicit Calculation of Coefficients

For the explicit method, I first calculate the "step" matrix for numerical integration with a direct implementation of Equation 4. Numpy's vector algebra function `np.dot` then allows for an elegant one-line solution for the initial coefficients.

```
step = np.eye(N_x-2, N_x-2) + dt*np.dot(A_inv, B)  # I + dt*A^{-1}*B
c0 = np.dot(A_inv, T[1:N_x-1])  # initial coefficients; note length N_x - 2
```

To find the rod's temperature distribution at later times, I first calculate the time-dependent coefficients $\boldsymbol{c}(t)$ according to Equation 4. Numpy's convenient `np.power` function performs the necessary matrix multiplications in a single line of code.

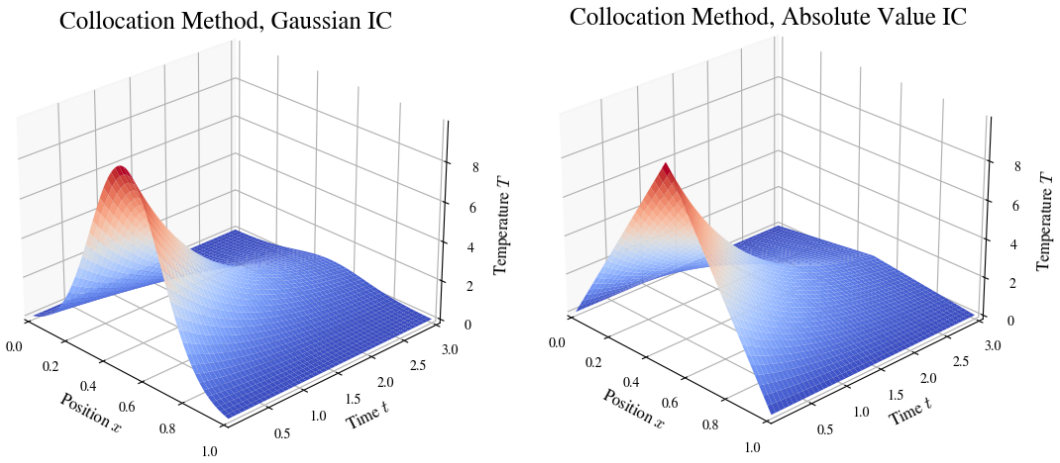



Figure 8: Heat diffusion in a 1D rod with homogeneous conditions and a Gaussian and absolute-value initial temperature distribution. Found with the collocation method. Note the similarity to the Fourier method results in Figure 3.

With $\boldsymbol{c}(t)$ known, I find the corresponding temperature distribution with $\boldsymbol{T} = \mathbf{A}\boldsymbol{c}(t)$ (Eqn. 6). Note that $\boldsymbol{c}$ is of length $N_x - 2$, so we must manually add the homogeneous boundary points with `np.concatenate`. The following code illustrates the procedure:

```
T_solutions = np.zeros((N_x, N_solutions))     # holds T(x) at each time t
for i, n in enumerate(time_indeces):
    c = np.dot(np.power(step, int(n)), c0)     # find coefficients c(t)
    T = np.concatenate(([0], A.dot(c), [0]))   # T = Ac, plus endpoints
    T_solutions[:, i] = T
```

### 3.3 Implicit Calculation of Coefficients

The implicit method is a straightforward implementation of Equation 5—the "step" matrix for numerical integration is now a product of two matrices, and the coefficients are found according to

$$\boldsymbol{c}^{(i+1)} = \left[\mathbf{A} - \frac{h_t}{2}\mathbf{B}\right]^{-1}\left[\mathbf{A} + \frac{h_t}{2}\mathbf{B}\right]\boldsymbol{c}^{(i)}$$

The corresponding Python code reads

```
dt, A, A_inv, B  # same time step and matrices as in explicit method
step_plus = A + 0.5*dt*B    # first ``step matrix''
step_minus = A - 0.5*dt*B   # second ``step matrix''
step = np.dot(np.linalg.inv(step_minus), step_plus)
# ... remaining code analogous to explicit method
```

As mentioned in the introduction to this section, I did not notice any significant differences between the explicit and implicit methods for coefficient calculation. I assume this similarity is due to the relatively "well-behaved" nature of the exponentially-solved differential equation defining the coefficients $\boldsymbol{c}(t)$. For a more chaotic/oscil-

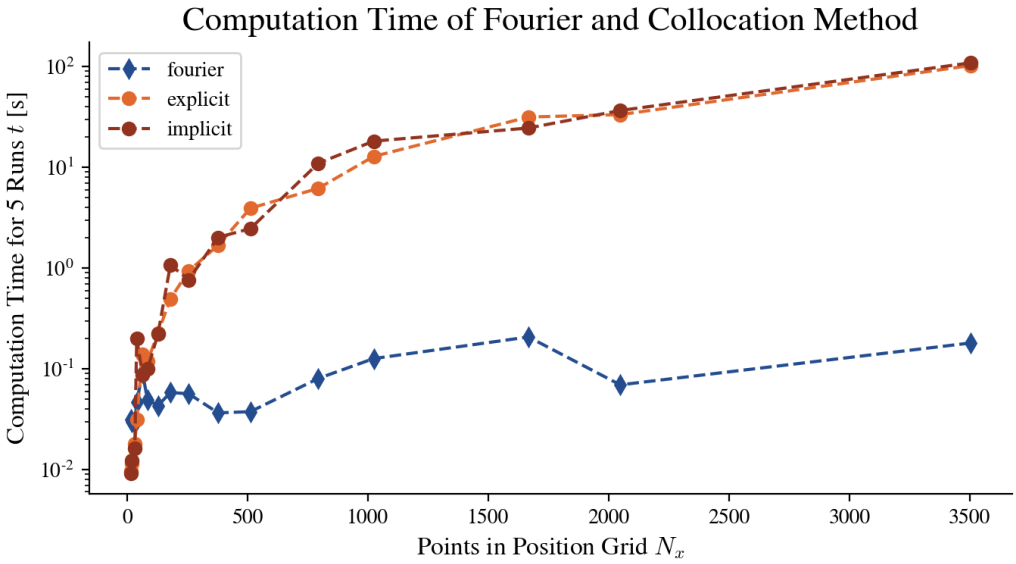



Figure 9: Time required by each method to solve the 1D heat equation—note the logarithmic scale. The $\mathcal{O}(N \log N)$ FFT-based Fourier method performs best.

lating/unstable differential equation, the stability of the implicit method would be more important, although this is just a guess.

### 3.4 Timing Comparison of Fourier and Collocation Methods

Finally, to compare the Fourier and collocation methods' performance, I tested the computation time required by each method to solve the 1D rod's heat diffusion problem as a function of the number of points $N_x$ used to partition the rod's length. Some notes:

- As simulation parameters, I used a Gaussian initial distribution, homogeneous boundary conditions, and calculated the temperature distribution at fifty equally spaced times from $t = 0$ to the point of decay to a near-zero distribution at $t = 3.0$, using a time step of $10^{-4}$ for the collocation method.
- Figure 9 shows the results. As expected, the Fourier method, based on the $\mathcal{O}(N \log N)$ FFT, performs considerably better than the collocation methods, which use $\mathcal{O}(N^2)$ matrix multiplication.
- I intentionally tested $N_x$ that both were and were not powers of two, hypothesizing that the Fourier method's FFT would perform better at the powers of two. Although I would need more data for a conclusive result, the guess seems reasonable—note for example the better performance at $N_x = 2048$ compared to the non-power-of-two $N_x \approx 1650$, even though the latter uses fewer points.

# A Definition of the Cubic B-Splines

$$B_k(x) = \begin{cases} 0 & x \leq x_{k-1} \\ \frac{1}{h^3}(x - x_{k-2})^3 & x_{k-2} < x \leq x_{k-1} \\ \frac{1}{h^3}(x - x_{k-2})^3 - \frac{4}{h^3}(x - x_{k-1})^3 & x_{k-1} < x \leq x_k \\ \frac{1}{h^3}(x_{k+2} - x)^3 - \frac{4}{h^3}(x_{k+1} - x)^3 & x_k < x \leq x_{k+1} \\ \frac{1}{h^3}(x_{k+2} - x)^3 & x_{k+1} < x \leq x_{k+2} \\ 0 & x > x_{k+2} \end{cases}$$

# References


[1] Matthew J. Hancock. "The 1-D Heat Equation". Lecture 1 In *Linear Partial Differential Equations—MIT Course No. 18.303.* Cambridge MA, 2006. https://ocw.mit.edu/courses/mathematics/18-303-linear-partial-differential-equations-fall-2006/lecture-notes/heateqni.pdf