The assignment

Part 1: Solving a wave problem with sparse matrices

In this part of the assignment, we want to compute the solution to the following (time-harmonic) wave problem:

$$\begin{aligned} \frac{\mathrm{d}^2 u}{\mathrm{d}x^2} + k^2 u = 0 \qquad & \text{in } (0,1), \\ u = 0 \qquad & \text{if } x = 0, \\ u = 1 \qquad & \text{if } x = 1, \end{aligned}$$

with wavenumber $k = 29\pi/2$.

In this part, we will approximately solving this problem using the method of finite differences.
We do this by taking an evenly spaced values $x_0 = 0, x_1, x_2, \ldots, x_N = 1$ and approximating the value of u for each value: we will call these approximations u_i. To compute these approximations, we use the approximation:

$$\frac{\mathrm{d}^2 u_i}{\mathrm{d}x^2} \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$$

where $h = 1/N$.

With a bit of algebra, we see that the wave problem can be written as

$$(2 - h^2 k^2) u_i - u_{i-1} - u_{i+1} = 0$$

if x_i is not 0 or 1, and

$$\begin{aligned} u_i = 0 \qquad & \text{if } x_i = 0, \\ u_i = 1 \qquad & \text{if } x_i = 1. \end{aligned}$$

This information can be used to re-write the problem as the matrix-vector problem $\mathbf{A}\mathbf{u} = \mathbf{f}$, where $\mathbf{A}$ is a known matrix, $\mathbf{f}$ is a known vector, and $\mathbf{u}$ is an unknown vector that we want to compute. The entries of $\mathbf{f}$ and $\mathbf{u}$ are given by

$$\begin{aligned} [\mathbf{u}]_i &= u_i, \\ [\mathbf{f}]_i &= \begin{cases} 1 & \text{if } i = N, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

The rows of $\mathbf{A}$ are given by

$$[\mathrm{A}]_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases}$$

if $i = 0$ or $i = N$; and

$$\boxed{[\mathrm{A}]_{i,j} = \begin{cases} 2 - h^2 k^2 & \text{if } j = i, \\ -1 & \text{if } j = i + 1, \\ -1 & \text{if } j = i - 1. \\ 0 & \text{otherwise,} \end{cases}}$$

otherwise.

Write a Python function that takes N as an input and returns the matrix A and vector f. You should use an appropriate sparse storage format for the matrix A.

The function `scipy.sparse.linalg.spsolve` can be used to solve a sparse matrix-vector problem. Use this to **compute the approximate solution for**

your problem for N=10, N=100, and N=1000. Use `matplotlib` (or any other plotting library) to **plot the solutions for these three values of N**.

Briefly (1-2 sentences) comment on your plots: How different are they from each other? Which do you expect to be closest to the actual solution of the wave problem?

This wave problem was carefully chosen so that its exact solution is known: this solution is $u_{\text{exact}}(x) = \sin(kx)$. (You can check this by differentiating this twice and substituting, but you do not need to do this part of this assignment.)

A possible approximate measure of the error in your solution can be found by computing

$$\max_i |u_i - u_{\text{exact}}(x_i)| .$$

Compute this error for a range of values for N of your choice, for the method you wrote above. On axes that both use log scales, **plot N against the error in your solution**. You should pick a range of values for N so that this plot will give you useful information about the methods.

For the same values of N, **measure the time taken to compute your approximation for your function**. On axes that both use log scales, **plot N against the time taken to compute a solution**.

We now want to compute an approximate solution where the error measure is 10^{-8} or less. By looking at your plots, **pick a value of N that you would expect to give error of 10^{-8} or less**. **Briefly (1-2 sentences) explain how you picked your value of N and predict how long the computation will take**.

Compute the approximate solution with your value of N. Measure the time taken and the error, and **briefly (1-2 sentences) comment on how these compare to your predictions**. Your error may turn out to be higher than 10^{-8} for your value of N: if so, you can still get full marks for commenting on why your prediction was not correct. Depending on your implementation and your prediction, a valid conclusion in the section could be "My value of N is too large for it to be feasible to complete this computation in a reasonable amount of time / without running out of memory".

Part 2: Solving the heat equation with GPU acceleration

In this part of the assignment, we want to solve the heat equation

$$\begin{aligned} \frac{\mathrm{d}u}{\mathrm{d}t} &= \frac{1}{1000}\frac{\mathrm{d}^2u}{\mathrm{d}x^2} && \text{for } x \in (0,1), \\ u(x,0) &= 0, && \text{if } x \neq 0 \text{ and } x \neq 1 \\ u(0,t) &= 10, \\ u(1,t) &= 10. \end{aligned}$$

This represents a rod that starts at 0 and is heated to 10 at both ends.

Again, we will approximately solve this by taking an evenly spaced values $x_0 = 0, x_1, x_2, \ldots, x_N = 1$. Additionally, we will take a set of evenly spaced times $t_0 = 0, t_1 = h, t_2 = 2h, t_3 = 3h, \ldots$, where $h = 1/N$. We will write $u_i^{(j)}$ for the approximate value of u at point x_i and time t_j (i.e. $u_i^{(j)} \approx u(x_i, t_j)$).

Approximating both derivatives (similar to what we did in part 1), and doing some algebra, we can rewrite the heat equation as

$$\begin{aligned} u_i^{(j+1)} &= u_i^{(j)} + \frac{u_{i-1}^{(j)} - 2u_i^{(j)} + u_{i+1}^{(j)}}{1000h}, \\ u_i^{(0)} &= 0, \\ u_0^{(j)} &= 10, \\ u_N^{(j)} &= 10. \end{aligned}$$

This leads us to an iterative method for solving this problem: first, at $t = 0$, we set

$$u_i^{(0)} = \begin{cases} 10 & \text{if } i = 0 \text{ or } i = N, \\ 0 & \text{otherwise;} \end{cases}$$

then for all later values of time, we set

$$u_i^{(j+1)} = \begin{cases} 10 & \text{if } i = 0 \text{ or } i = N, \\ u_i^{(j)} + \dfrac{u_{i-1}^{(j)} - 2u_i^{(j)} + u_{i+1}^{(j)}}{1000h} & \text{otherwise.} \end{cases}$$

Implement this iterative scheme in Python. You should implement this as a function that takes N as an input.

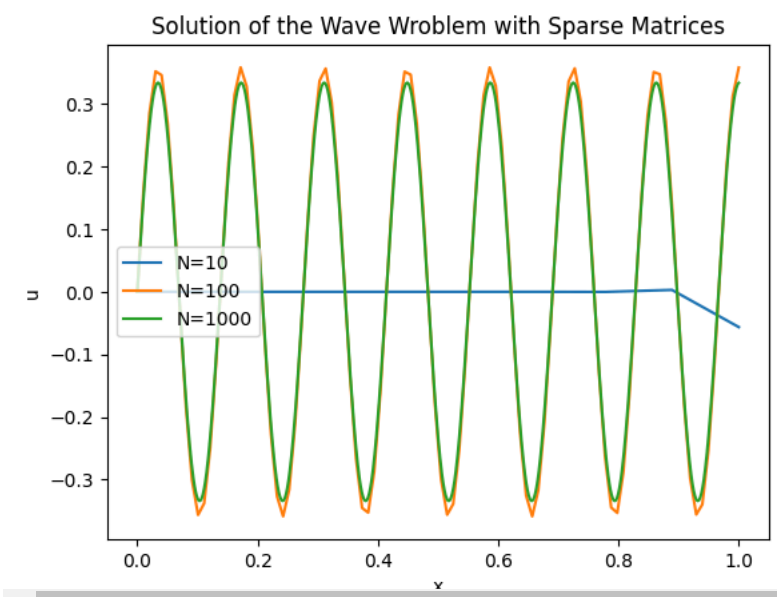
Using a sensible value of N, **plot the temperature of the rod at $t = 1$, $t = 2$ and $t = 10$. Briefly (1-2 sentences) comment on how you picked a value for N.**

Use `numba.cuda` to parallelise your implementation on a GPU. You should think carefully about when data needs to be copied and be careful not to copy data to/from the GPU when not needed.

Use your code to estimate the time at which the temperature of the midpoint of the rod first exceeds a temperature of 9.8. Briefly (2-3 sentences) describe how you estimated this time. You may choose to use a plot or diagram to aid your description, but it is not essential to include a plot.

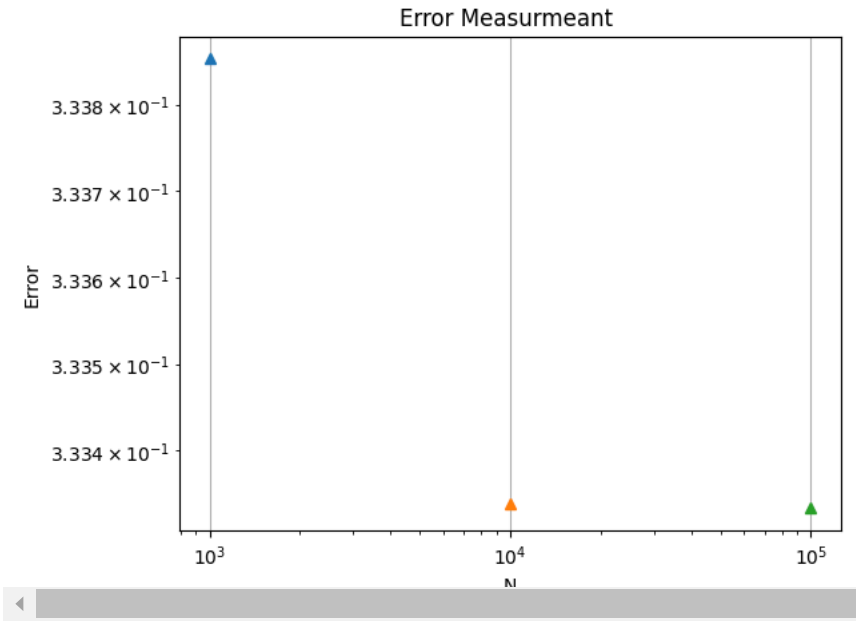
Part **1**

```
import numpy as np
from scipy.sparse import *
import matplotlib.pyplot as plt

def discretise_wave(N):

    rows = []
    columns = []
    data = []

    f = np.zeros(N, dtype=np.float64)
    # If i=N, f=1
    # f[N] = 1
#According to the formula of the wave problem
#wavenumber k=29π/2
#where h=1/N
#(2−h2k2)ui−ui−1−ui+1=0
    h = 1 / N
    k = 29 * np.pi / 2
    diagonal = 2 - (h**2 * k**2)

#Loop through boundry conditions
    for i in range(N):
                # print(i)
                rows.append(i)
                columns.append(i)
                data.append(diagonal)
                if i==N-1:
                 f[i] = 1
    for i in range(1,N-1):
                rows.append(i)
                columns.append(i - 1)
                data.append(-1)

                rows.append(i)
                columns.append(i + 1)
                data.append(-1)

    rows.append(0)
    columns.append(0)
    data.append(1)

    rows.append(N-1)
    columns.append(N-1)
    data.append(1)

    row_ind = np.array(rows)
    col_ind = np.array(columns)
    data = np.array(data)

    return coo_matrix((data, (row_ind, col_ind)), shape=(N , N )).tocsr(), f

def plot_waves(N):
    for n in N:
        A, f = discretise_wave(n)
        u = linalg.spsolve(A, f)
        x = np.linspace(0, 1, n)
        plt.plot(x, u, label=f'N={n}')

    plt.xlabel('x')
    plt.ylabel('u')
    plt.title('Solution of the Wave Wroblem with Sparse Matrices')
    plt.legend()
    plt.show()

#Compute for N=10, N=100, and N=1000
N = [10, 100, 1000]
plot_waves(N)
```



- Answer: Whenever the input size grows, the waves bacome more clearer; that's why when N=10 the wave didn't appear properly. But for N=100 and 1000 the waves were more precise and seem to be possible solutions for the problem. However, I expect N=1000 to be the solution for the problem as it depicts the details of the wave more accuretly.

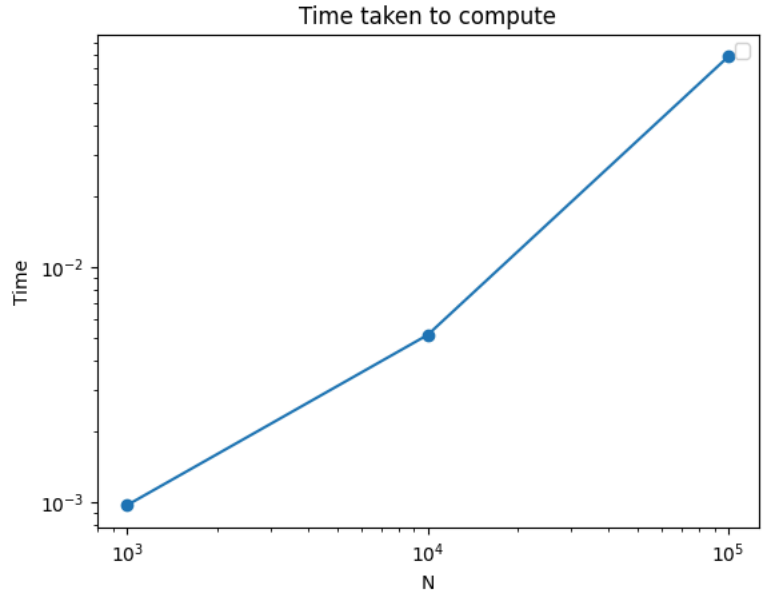
```
# error = max(|u_numerical - u_exact|)
#uexact(x)=sin(kx)

def Plotting_Error(N):
    for n in N:
        A, f = discretise_wave(n)
        u = linalg.spsolve(A, f)
        k=29 * np.pi / 2
        u_exact=np.sin(k*n)
        error = np.max(np.abs(u - u_exact))
        plt.loglog(n, error, '^-')

    plt.xlabel('N')
    plt.ylabel('Error')
    plt.title('Error Measurmeant ')
    plt.grid(True)
    plt.show()

N = [1000,10000,100000]
Plotting_Error(N)
```




```
# For the same values of N
# measure the time taken to compute your approximation for your function. On axes that both use log scales, plot N
# against the time taken to compute a solution.
```

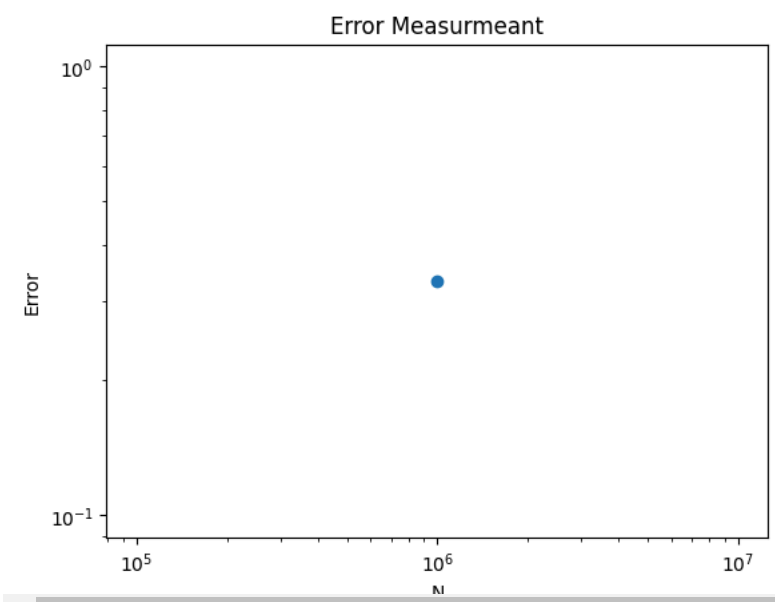
```
times = []
def Plotting_Error(N):
    for n in N:
        A, f = discretise_wave(n)
        t = %timeit -o -q -n1 -r1 linalg.spsolve(A, f)
        times.append(t.average)


N = [1000,10000,100000]
Plotting_Error(N)
plt.figure()
plt.loglog(N, times, 'o-')
plt.legend()
plt.title("Time taken to compute")
plt.xlabel("N")
plt.ylabel("Time")
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are
```



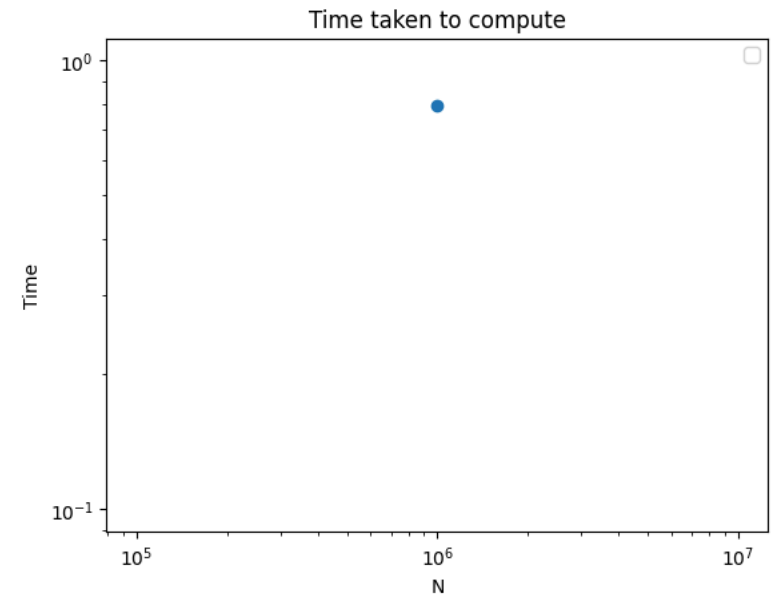
- Answer: I think that N=1,000,000 might give an error of 10^-8 or less. The reason why I chose this number is becuase according to the error plot, whenever the N increases, the error decreases so I decided to choose larger value than the last calculated N perviously.

```
A, f = discretise_wave(1000000)
u = linalg.spsolve(A, f)
k=29 * np.pi / 2
u_exact=np.sin(k*1000000)
error = np.max(np.abs(u - u_exact))
plt.loglog(1000000, error, 'o-')

# Plot settings
plt.xlabel('N')
plt.ylabel('Error')
plt.title('Error Measurmeant')
plt.show()
```




```
A, f = discretise_wave(1000000)
t = %timeit -o -q -n1 -r1 linalg.spsolve(A, f)
plt.figure()
plt.loglog(1000000, t.average, 'o-')
plt.legend()
plt.title("Time taken to compute")
plt.xlabel("N")
plt.ylabel("Time")
plt.show()
```

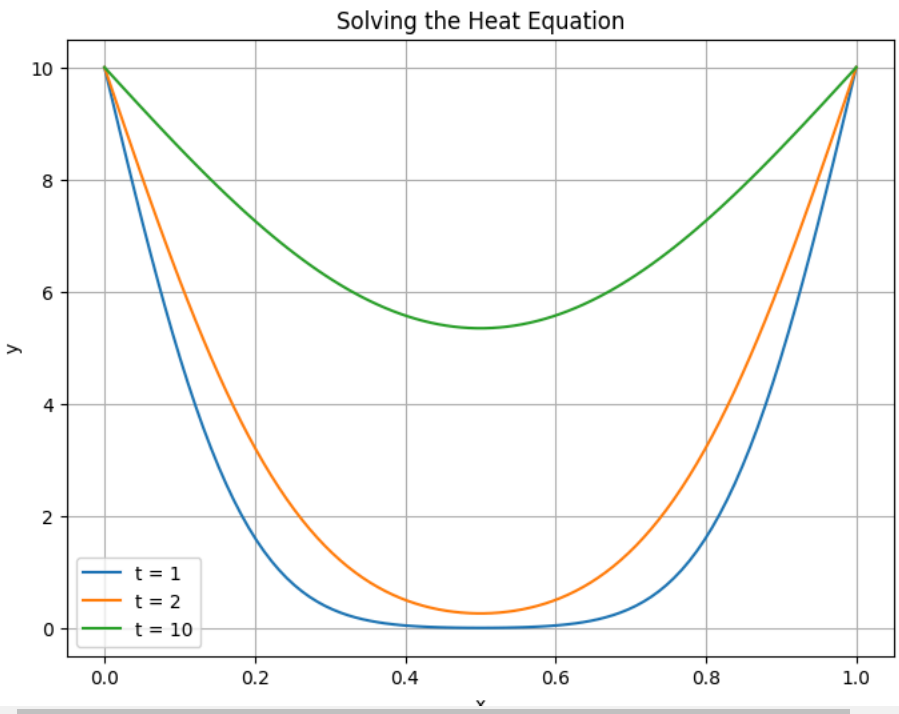
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are
```



- Answer: I still believe that when we increase the N value the error will decrease, however, it seems like one million is still small to produce an error value of 10^-8. Therefore, I cann't increase the N value to get the intended result as it will be very large to compute without running out of memory. (I actully tried after I got my prediction result, but the computation interrupted as I ran out of memory and the kernel shut down).

```
import numpy as np
import matplotlib.pyplot as plt

def heat_problem(N):

#Define as stated in the information given for the heat equation
  h = 1 / N
  MaxTime = 10 #bc we're going to plot t=1,2,10
  dt = 0.001 #use a reasonable value to have accurate result
  Timesteps = int(MaxTime / dt)
  u = np.zeros((N, Timesteps + 1)) #Initilize an array with zeros
```

```
#Initial conditions of heat equation
  u[0, 0] = 10
  u[N-1, 0] = 10

  for j in range(Timesteps):
    for i in range(1, N - 1): #This loop will iterate to implement the conditions of the heat equation
      u[i, j + 1] = u[i, j] + (u[i - 1, j] - 2 * u[i, j] + u[i + 1, j]) / (1000 * h)
      u[0, j + 1] = 10
      u[N-1, j + 1] = 10

  return u

#Give a value of N then call the function
N = 100
dt=0.001
u = heat_problem(N)

#Fill the times to plot them
temperature  = [1, 2, 10]
plt.figure(figsize=(8, 6))
for t in temperature :
  timeSt = int(t / dt)
  plt.plot(np.linspace(0, 1, N), u[:, timeSt],label=f"t = {t}")

plt.xlabel("x")
plt.ylabel("y")
plt.title("Solving the Heat Equation")
plt.legend()
plt.grid(True)
plt.show()
```



- Answer: I tried to choose various values of N for example, I started with 10 but the plot wasn't smooth at all. However, I noticed that whenever we increase the size of N athe result will become more accurate and smooth. Furthermore, I noticed a relation between the size of N value and dt, it's important to choose good values to obtain the best possible solution. That's why I chose N=100 taking into account the value of dt.

```
from numba import cuda
cuda.detect()
```

```
Found 1 CUDA devices
id 0             b'Tesla T4'                              [SUPPORTED]
                      Compute Capability: 7.5
                           PCI Device ID: 4
                              PCI Bus ID: 0
                                    UUID: GPU-5205a7c1-bd5d-5da6-d586-03b449a85365
                                Watchdog: Disabled
             FP32/FP64 Performance Ratio: 32
Summary:
        1/1 devices are supported
```

```
True
```

```
from numba import cuda
@cuda.jit
def heat_problem_cuda(N,u, Timesteps ,h,dt):

    i, j = cuda.grid(2)
    if i >= 0 and i < N and j < Timesteps:
        if i == 0 or i == N:
            u[i, j + 1] = 10
        else:
            u[i, j + 1] = u[i, j] + (u[i - 1, j] - 2 * u[i, j] + u[i + 1, j]) / (1000 * h)


import matplotlib.pyplot as plt
import numpy as np
import math


N = 100
h = 1 / N
MaxTime = 10
dt = 0.01
Timesteps = int(MaxTime / dt)
u = np.zeros((N, Timesteps + 1))

#conditions of the heat equation
u[0, 0] = 10
u[N-1, 0] = 10

#transfer it to the GPU
im_gpu = cuda.to_device(u)

threadsperblock = (16, 16)
blockspergrid_x = math.ceil(im_gpu.shape[1] / threadsperblock[1])
blockspergrid_y = math.ceil(im_gpu.shape[0]  / threadsperblock[0])
blockspergrid = (blockspergrid_x, blockspergrid_y)
heat_problem_cuda[blockspergrid, threadsperblock](N, im_gpu, Timesteps, h, dt)

#transfer it back to the host
u = im_gpu.copy_to_host()

# temperature  = [1, 2, 10]
# plt.figure(figsize=(6, 6))
# for t in temperature :
#     timeSt = int(t / dt)
#     plt.plot(np.linspace(0, 1, N), u[:, timeSt], label=f"t = {t}")

# plt.xlabel("x")
# plt.ylabel("Temperature")
# plt.title("Solving the Heat Equation")
# plt.legend()
# plt.grid(True)
# plt.show()


%%timeit
heat_problem(N)
```

```
2.79 s ± 860 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%%timeit
heat_problem_cuda[blockspergrid, threadsperblock](N, im_gpu, Timesteps, h, dt)
# cuda.synchronize()
```

```
44.2 µs ± 856 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

- Answer: I predict that the time at which the temperature will exceeds 9.8 will be 0.5. After calculating the midpoint which is: N/2=50, then by visually monitoring the plot for the heat equation and recording the first time the rod exceeded 9.8, i came into the conclusion that the time is 0.5. Additionally, if we create a loop to iterate over the points and check our array that has the index of the midpoint, we will be able to obtain the time.

```
Start coding or generate with AI.
```