```
import numpy as np
import math
from scipy.sparse import coo_matrix, linalg
from matplotlib import pyplot as plt
from timeit import timeit
from numba import cuda
```

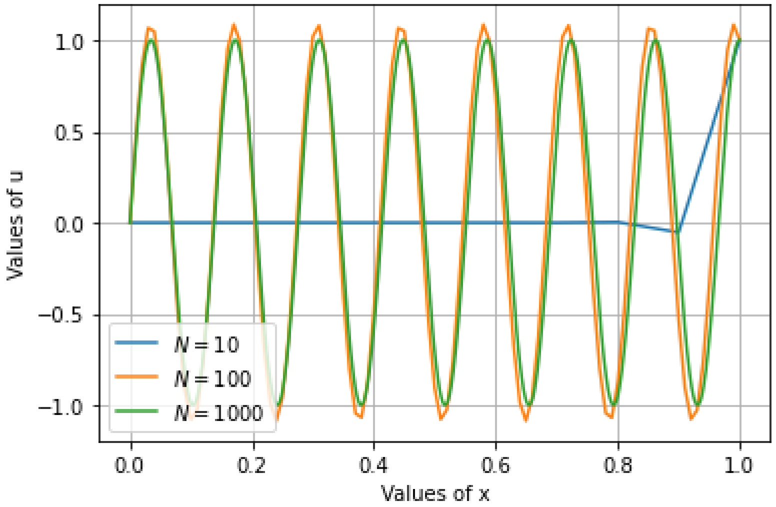
```
def matrix_sparse(N):
    k = 29*np.pi/2
    rows = [0,N]
    cols = [0,N]
    data = [1,1]
    f = np.zeros((N+1))
    h = 1/N
    f[N] = 1
    for i in range(1,N):
        rows += [i, i, i]
        cols += [i, i+1, i-1]
        data += [2-(h*k)**2, -1, -1]
    rows = np.array(rows)
    cols = np.array(cols)
    data = np.array(data)
    A = coo_matrix((data, (rows, cols)), (N+1, N+1))
    return A, f
```

```
dia = [10, 100, 1000]
for i in dia:
    xi = np.linspace(0,1,i+1)
    Ai, fi = matrix_sparse(i)
    soli = linalg.spsolve(Ai, fi)
    plt.plot(xi, soli)

plt.legend(["$N=10$","$N=100$","$N=1000$"],loc=3)
plt.grid()
plt.xlabel("Values of x")
```

```
plt.ylabel("Values of u")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/scipy/sparse/linalg/dsolve/linsolve.py:145: SparseEff
  SparseEfficiencyWarning)
/usr/local/lib/python3.7/dist-packages/scipy/sparse/linalg/dsolve/linsolve.py:145: SparseEff
  SparseEfficiencyWarning)
/usr/local/lib/python3.7/dist-packages/scipy/sparse/linalg/dsolve/linsolve.py:145: SparseEff
  SparseEfficiencyWarning)
```



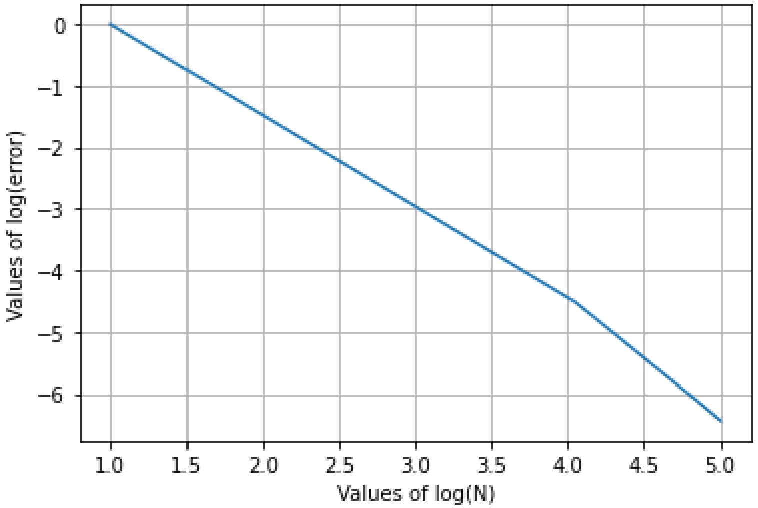
Answer : When N is small where N=10, the plot is linear, but when N increases to 100 and 1000, the plots will be similar and smoother as N increasing. Therefore, the plot of N=1000 will be closest to the actual solution.

```
def error(N):
    k = 29*np.pi/2
    x = np.linspace(0,1,N+1)
    A, f = matrix_sparse(N)
    ui = linalg.spsolve(A, f)
    u_ex = np.sin(k*x)
    return max(abs(ui-u_ex))

N_1 = np.linspace(10,100000,10,dtype=int)
err = []
```

```
for i in N_1:
    err += [np.log10(error(i))]
N_2 = np.log10(N_1)
plt.plot(N_2,err)
plt.grid()
plt.xlabel("Values of log(N)")
plt.ylabel("Values of log(error)")
```

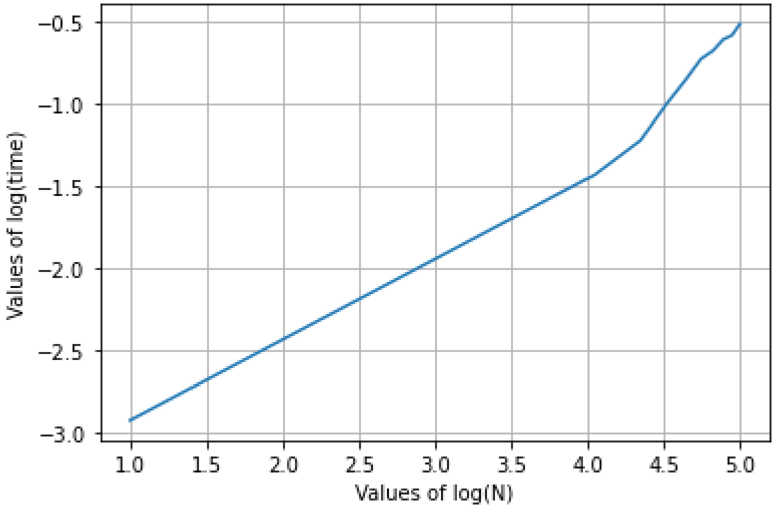
```
/usr/local/lib/python3.7/dist-packages/scipy/sparse/linalg/dsolve/linsolve.py:145: SparseEff
  SparseEfficiencyWarning)
Text(0, 0.5, 'Values of log(error)')
```




```
time_1 = []
for i in N_1:
    time_1 += [np.log10(timeit(lambda: error(i),number=1))]

plt.plot(N_2,time_1)
plt.grid()
plt.xlabel("Values of log(N)")
plt.ylabel("Values of log(time)")
```

```
/usr/local/lib/python3.7/dist-packages/scipy/sparse/linalg/dsolve/linsolve.py:145: SparseEff
  SparseEfficiencyWarning)
Text(0, 0.5, 'Values of log(time)')
```



Answer : By looking at the plots of log10(N) to log10(error) and log10(time), we find it is nearly linear. Thus, we can use the estimated slope where k=-1.6 to predict the value of N where log10(N)=5.94, so N=870000. For the second plot, the estimate slope k=0.9, thus log10(time)=0.35, where time=2.24s.

```
time_2 = timeit(lambda: error(870000),number=1)
error_2 = error(870000)
print('time is ',time_2, 'error is ',error_2)
```

```
time is  2.473340908000864 error is  1.445554753086442e-07
```

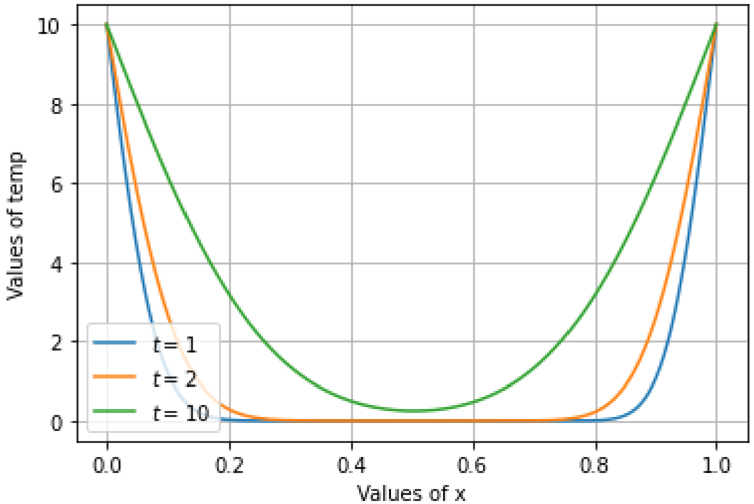
Answer : The predicted time is similar to the exact time, but the error is higher than expected. The reason is that when N exceeds 1e+6, h will be 1e-6, but we have h square in our function, which means the error will exceed the accuracy of the floating number in python, then the error could not be more precise.

```
def heat_matrix(N,T):
    h = 1/N
```

```
    times = T*N
    u = np.zeros(N+1)
    u[0] = 10
    u[N] = 10
    for j in range(1,times+1):
        Y = u
        for i in range(1,N):
            u[i] = Y[i]+(Y[i-1]-2*Y[i]+Y[i+1])/(1000*h)
    return u
```

```
N_2 = np.linspace(0,1,501)
T_1 = [1,2,10]
for i in T_1:
    u_1 = heat_matrix(500,i)
    plt.plot(N_2, u_1)

plt.legend(["$t=1$","$t=2$","$t=10$"],loc=3)
plt.grid()
plt.xlabel("Values of x")
plt.ylabel("Values of temp")
plt.show()
```



Answer: The thermal conductivity of the rob becomes higher as N increasing. But N should be less than 1000 because when N>1000, the temperature inside the rod will be higher than 10 which is unnormal. Therefore, in order to check how the temperature changes with time, we choose N=500.

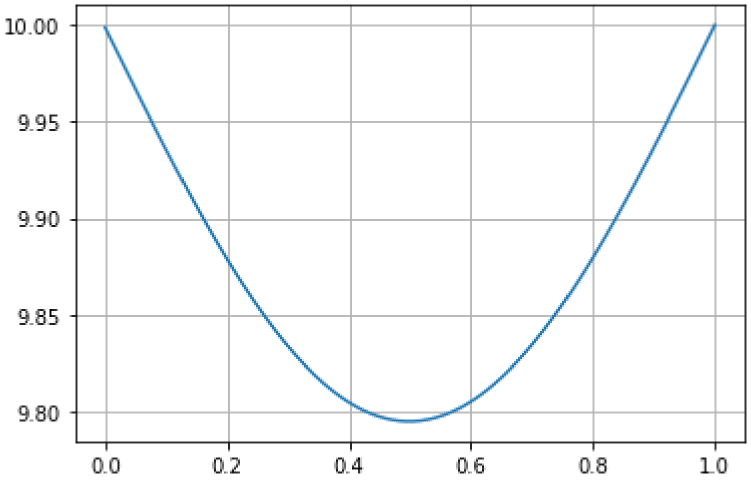
```
@cuda.jit
def heat_matrix_gpu(N,u,Y):
    idx = cuda.threadIdx.x + cuda.blockDim.x * cuda.blockIdx.x
    if idx < N:
        u[idx] = Y[idx]+(Y[idx-1]-2*Y[idx]+Y[idx+1])/(1000/N)
```

```
def heat_matrix_cuda(N,T):
    times = T*N
    u = np.zeros(N+1)
    u[0] = 10
    u[-1] = 10
    Y = np.zeros(N+1)
    u_device = cuda.to_device(u)
    Y_device = cuda.to_device(Y)
    threads_per_block = 128
    blocks_per_grid = 64
    for j in range(times):
        Y_device = u_device
        heat_matrix_gpu[blocks_per_grid,threads_per_block](N,u_device,Y_device)
    return u_device.copy_to_host()
```

```
n = 500
N_3 = np.linspace(0,1,n+1)
u_3 = heat_matrix_cuda(n,420)
plt.plot(N_3,u_3)
plt.grid()
T_2 = np.linspace(415,425,10,dtype=int)
n = 500
for i in T_2:
    U = heat_matrix_cuda(n,i)
```

```
    if U[math.ceil(n/2)] > 9.8:
        print('First time when temperature exceeds 9.8 is ',i)
        break
```

```
/usr/local/lib/python3.7/dist-packages/numba/cuda/dispatcher.py:488: NumbaPerformanceWarning: Grid size 64 will likely result in GPU unde
  warn(NumbaPerformanceWarning(msg))
First time when temperature exceeds 9.8 is  423
```



Answer: Firstly, we find when T=420, the temperature of the rob is nearly 9.8, then we test 10 data between 415 and 425. Finally we can find it exceeds 9.8 when T=423.