

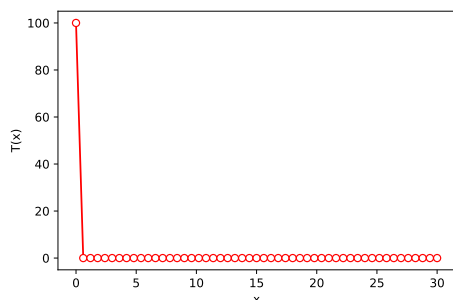
Computer Lab Assignment 7

Stationary State of the Heat Equation in 1D and 2D

No code will be provided for this computer lab. Instead you are asked to write it all from scratch. Please save all notebooks for future use. To produce the animations in last part, the FFMPEG package needs to be installed on your computer and windows need to pop out of your notebook. So you cannot use the datahub or GoogleColab for this and most future assignments.

Stationary State of the 1D heat equation

Discretize the interval $x=[0,L]$ in N sections using $N+1$ equally spaced points in order to solve the 1D heat equation with the boundary conditions $T(x=0)=100$ and $T(x=L)=0$ and the initial condition $T(x>0)=0$. Please note that Python only understands the notation $T[i]$ but not $T(x=0)$, $T(x=L)$, and $T(x>0)$.



(1) $N=50$ and $L=30$ are good values to start but your code should work for any N and L . First you need to decide which index i corresponds to point $x=0$ and which index corresponds to $x=L$. Then write two lines of code to set up a NumPy array T to represent the specified initial conditions.

(2) Write the main, outer loop over different *iterations* to obtain the stationary state of the 1D heat equation. (Please refer to lecture 11 on PDEs slide 25 and following if something is unclear.) Inside the main loop introduce an inner loop to update every temperature *in the interior* of the interval $[0,L]$. The update formula in the Jacobi method is

$$T_i^{\text{new}} = \frac{1}{2} [T_{i-1} + T_{i+1}]. \quad (*)$$

(3) Before entering the inner loop introduce a second temperature array T^{new} . Choose the appropriate command from the following three options: $T^{\text{new}}=T$ and $T^{\text{new}}=\text{np.copy}(T)$ and $T^{\text{new}}[:,]=T[:,]$. Then update the interior points of T^{new} according to equation (*) and copy the results in T^{new} back to T so that they are the starting temperatures for the next iteration.

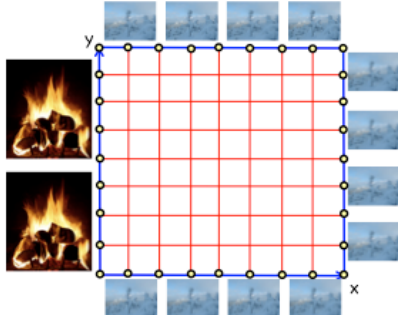
(4) It may be useful to plot the temperature distribution in every iteration. Pop the graphics window of your notebook and then try:

```
plt.clf()
plt.plot(x,T, 'ro-',mfc='w')
plt.show()
plt.draw()
plt.pause(0.05) # choose the time argument between 0.01 and 0.5
```

Stationary State of the Heat Equation in 2D

In this section, we want to find the stationary temperature distribution in a 2D square with the following boundary conditions:

**Stationary state for
different boundary conditions**

$$\frac{\partial T}{\partial t} = 0 \quad \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$


Imagine you are renting a room of size $L \times L$ that has three cold outside walls and one wall that is heated by your neighbor. The outside temperature is kept fixed at 0°C and inside wall at 100°C . Divide the square into $N \times N$ segments. $N=51$ is a good resolution to start. Solving this problem numerically requires a 2D array T_{ij} or in Python language an array $T[i, j]$.

Convert the analytic form of the 2D heat equation given in the lecture slide above into the discretized form using T_{ij} . Derive the 2D version of equation (*) given above.

$$T_{i,j}^{\text{new}} = \dots \left[\dots \dots \dots \right] \quad (**)$$

- (1) Write a code to set the boundary value of the 2D array $T(i, j)$.
- (2) Similar to part 1 introduce a main, outer loop over *iterations*.
- (3) Inside the main loop, introduce a nested pair of inner loops over i and j that cover all interior points of the square above. At each point, you should set the value of $T_{\text{new}}(i, j)$ according to equation (**).
- (4) As in part 1, copy the data back and forth between T and T^{new} .
- (5) To plot the temperature distribution in 3D, at the beginning of your code please add


```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
X = np.linspace(0, L, n) # where n is number of segments
Y = np.linspace(0, L, n)
X, Y = np.meshgrid(X, Y)
fig = plt.figure(dpi=200)
```

Inside the outer loop please add

```
fig.clear()
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, antialiased=False)
plt.draw()
plt.pause(0.05) # choose the time argument between 0.01 and 0.5
```

(6) *Optional:* Instead of using a temporary array T^{new} , change the update formula (**) so that you overwrite each $T[i, j]$ element immediately:

$$T_{i,j} = \dots \left[\dots \dots \dots \right]$$

Answer two questions:

- a) Does the code now converge faster or slower to the stationary temperature distribution?
- b) Does it make a difference in which order you update the elements? You could imagine interchanging the inside i and j loops, or running either one backwards.

1D and 2D Animations Captured in mp4 Movie Files

On bCourses, you find two files ‘wave1d.ipynb’ and ‘wave2d.ipynb’ that illustrate how easily .mp4 animation files can be generated with Python. Run each code, test whether it runs without errors on your computer, and click on the resulting .mp4 files to check that they can be played back on your computer.

Look inside both wave notebook files and identify all commands required to write the .mp4 files. First make a copy of the notebooks with your 1D and 2D heat equation solvers because the following change may break them. Then cut and paste all commands that are necessary to write .mp4 files. Run your notebooks again and see if proper .mp4 files were generated. Good luck and congratulation when you have succeeded in making your first animation.