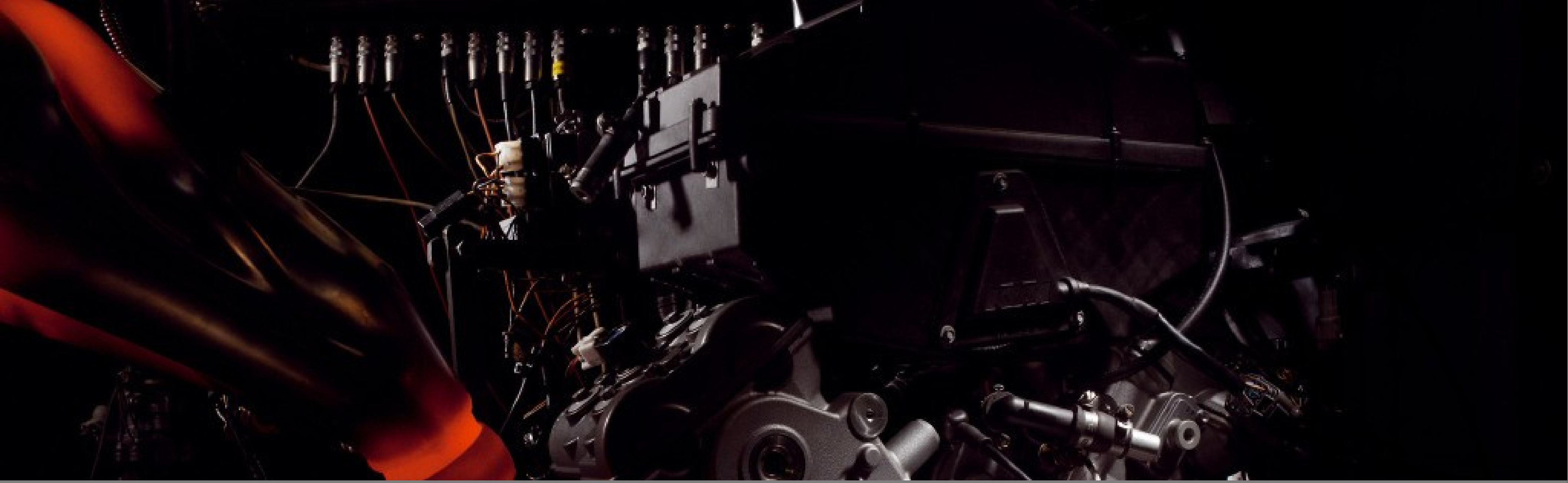


Solving the Heat Equation in Python!

By Melvyn Ian Drag

Learning Objectives for Tonight:

After this tutorial you will :

- Know what the heat equation is.
- Know a way solve the Heat Equation in Python
- Have a better familiarity with several impressive Python libraries.
- Be able to generate animations of your solutions.

Why Python?

“ Whereas a mathematical idea is a timeless thing, few things are more ephemeral than computer hardware and software. ”

-Tristan Needham

Useful Libraries

“The SciPy Stack”:

The SciPy “core library”
NumPy
Matplotlib
IPython

Further reference:

Thompson

How to Get All This Stuff?

Download either:

Anaconda

-Or-

Enthought Canopy

The Hea[illegible]ation

The Discretization of the Heat Equation

$$u_t = u_{xx}$$

$$\frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} = \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2}$$

$$u(x,t+\Delta t) = u(x,t) + \frac{\Delta t}{\Delta x^2}(u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t))$$

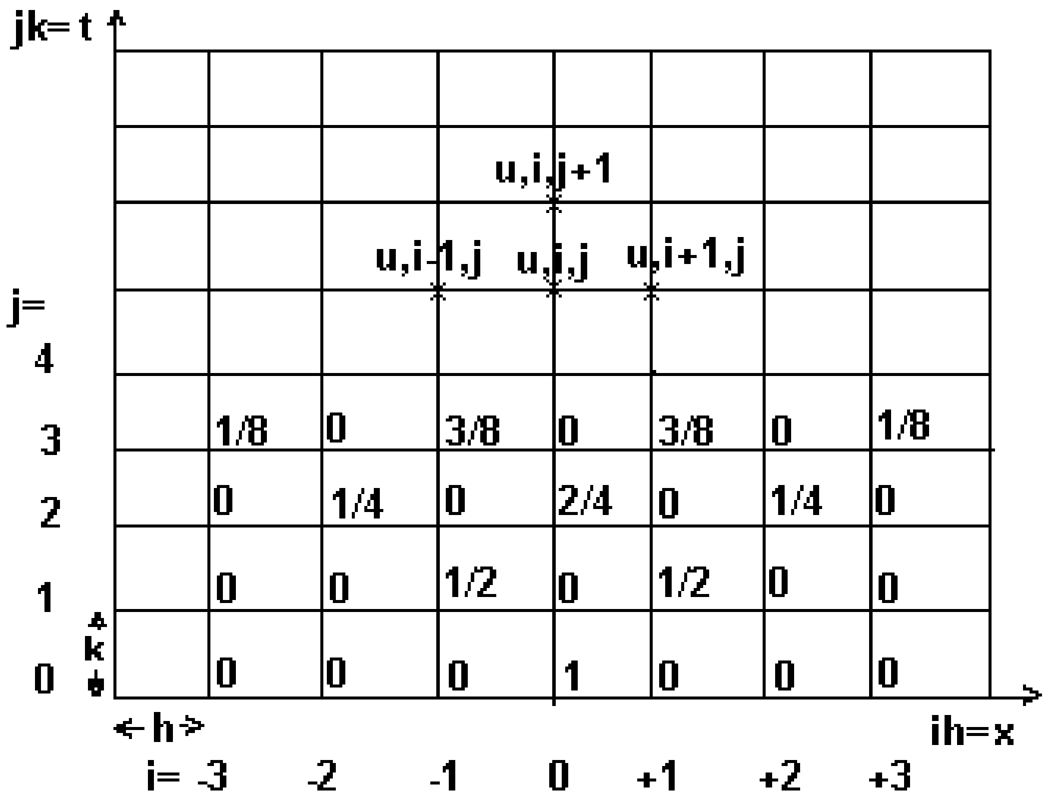
$$u_{i,j+1} = u_{i,j} + r(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})$$

Remember this:

$$u_{i,j+1} = u_{i,j} + r(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})$$

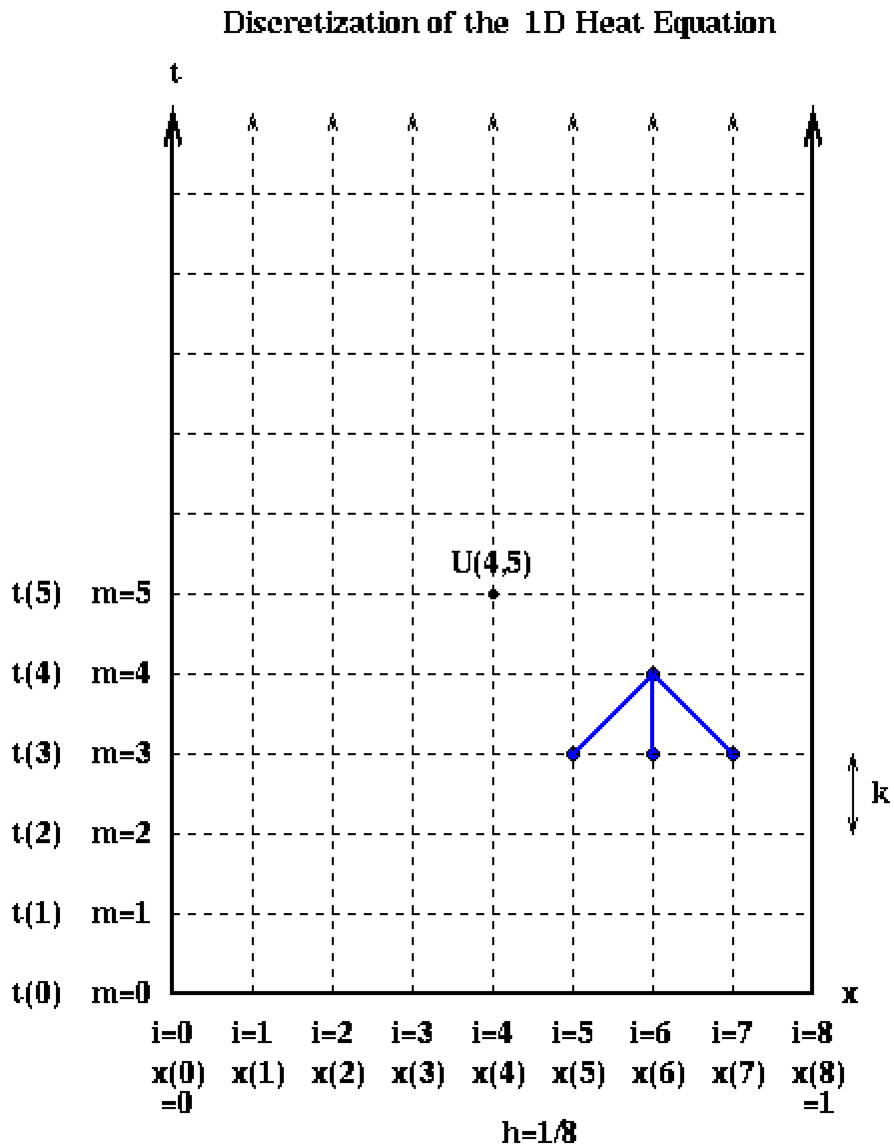
The Mesh

Let's study this for a minute:



We can ignore the fractions in the image.
I took this image from voting.ukscientists.com/diffus,
and they needed the fractions for something else.

A Look at a Mesh From a UC Berkeley Lecture



Repetion is the mother of all learning:

$$\boxed{u_{i,j+1} = u_{i,j} + r(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})}$$

Boundary Conditions

Or we could mix these
conditions . . .

Making a Matrix Equation

Now Making a Matrix Equation

$$u_{i,j+1} = u_{i,j} + r(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})$$

$$\begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ u_{3,j} \\ u_{4,j} \\ u_{5,j} \end{bmatrix} = \begin{bmatrix} -2u_{1,j} + u_{2,j} \\ u_{1,j} - 2u_{2,j} + u_{3,j} \\ u_{2,j} - 2u_{3,j} + u_{4,j} \\ u_{3,j} - 2u_{4,j} + u_{5,j} \\ u_{4,j} - 2u_{5,j} \end{bmatrix}$$

If we add just a little more seasoning to the above equation, everything will work out just right.

Finally, the

Let's Now Look at Some Code Together.

Note to self:
Start with constBC code

Then Show Animation Gif

Then
“Object oriented” matplotlib
Show FuncAnimation
- interval
- repeat
- global parameters for update
- adding the current time to the Animation
- Mention the anatomy of matplotlib tutorial online
- mention the matplotlib website and the gallery
- Mention jakevdp blog

Lambda functions
Then:
How to make diagonal arrays and matrices.
How to import from other files in your directory. Various import options.

Should I use Arrays or Matrices ?

Short answer

Use arrays.
They are the standard vector/matrix/tensor type of numpy. Many numpy function return arrays, not matrices.

There is a clear distinction between element-wise operations and linear algebra operations.

You can have standard vectors or row/column vectors if you like.

The only disadvantage of using the array type is that you will have to use dot instead of * to multiply (reduce) two tensors (scalar product, matrix vector multiplication etc.).

THEY DON'T MENTION THAT THERE ARE NO SPARSE ARRAYS! This is also a legitimate reason to use matrices.

Note to self:
Demo how to setup a sparse matrix.
Then demo the times compared to
The 2D array

Lets Get Our Hands Dirty:

Your Task:

0. Import library(ies) for our sparse matrices.

1. Change the 2D array "T" in our code to a Sparse Matrix.

2. Change the boundary conditions from constant to variable.

Here's how:

Change our domain to (-pi/2, pi/2). Use the exact solution to define the value of u at the endpoints at each time step.

3. To make sure you understand how the animation function works, create a separate animation of the real solution at each time step. It should look exactly like the numerical solution!

The Heat Equation in 2D

The 2D Discretization

$$u_t(x,y,t) = u_{xx}(x,y,t) + u_{yy}(x,y,t)$$

$$\frac{u_{i,j,k+1} - u_{i,j,k}}{\Delta t} =$$

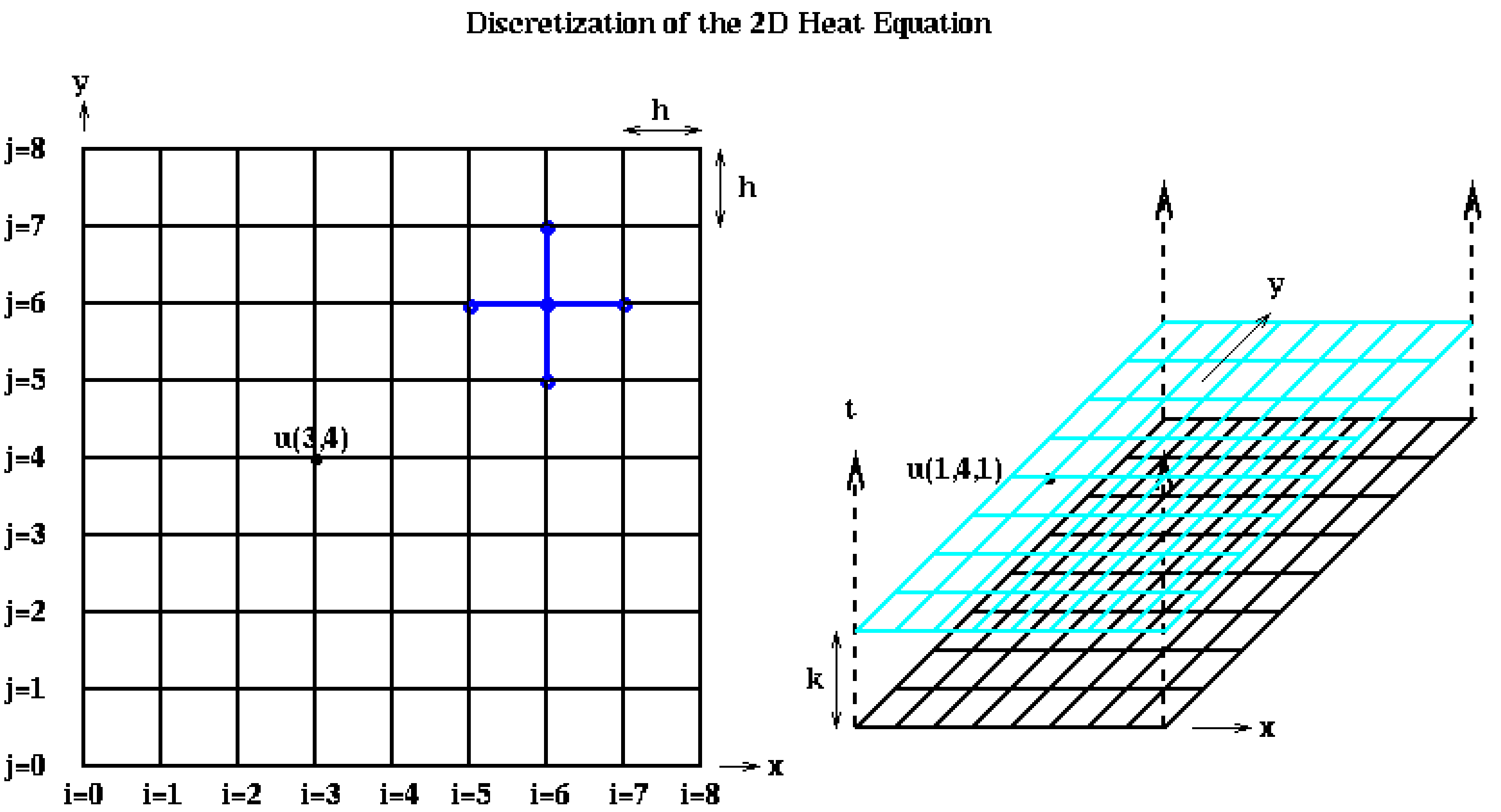
$$\frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{\Delta x^2} + \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{\Delta y^2}$$

$$u_{i,j,k+1} = u_{i,j,k} + r(u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k})$$

$$+r(u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k})$$

Where we have let $\Delta x = \Delta y$

Another Mesh from the UC Berkeley Site



Our New Matrix, Graciously Provided By UC Berkeley

Matrix for Discrete Poisson Problem

$$
\left[\begin{array}{cccc|cccc|cccc|cccc}
4 & -1 & & & -1 & & & & & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & & & & & \\
 & -1 & 4 & -1 & & & -1 & & & & & & & & & \\
 & & -1 & 4 & & & & -1 & & & & & & & & \\
\hline
-1 & & & & 4 & -1 & & & -1 & & & & & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & & & & & \\
 & & -1 & & & -1 & 4 & -1 & & & -1 & & & & & \\
 & & & -1 & & & -1 & 4 & & & & -1 & & & & \\
\hline
 & & & & -1 & & & & 4 & -1 & & & -1 & & & \\
 & & & & & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & & & & & -1 & & & -1 & 4 & -1 & & & -1 & \\
 & & & & & & & -1 & & & -1 & 4 & & & & -1 \\
\hline
 & & & & & & & & -1 & & & & 4 & -1 & & \\
 & & & & & & & & & -1 & & & -1 & 4 & -1 & \\
 & & & & & & & & & & -1 & & & -1 & 4 & -1 \\
 & & & & & & & & & & & -1 & & & -1 & 4
\end{array}\right]
$$

Let's Now Look at Some Code Together.

Your Task: Implement a Code With Function Boundary Conditions.

Just as in the last project, you have to take a code which works for constant boundary conditions and modify it in such a way that it takes boundary conditions which vary over time.