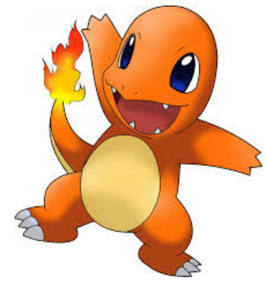
Activity 6: The Heat Equation



Primary Goal: Learn how to code isotropic and anisotropic diffusion.
Secondary Goal: Understand how to solve a PDE numerically.

1.) Isotropic Diffusion

Download the Matlab file **heat.m** for the Isotropic Heat Diffusion from our course website. To download a m-file, it is best to right-click on the link and select "Save As...".

The pseudocode for the Forward Euler solution to the Heat Equation is shown in Figure 1. Open the m-file and match up the Matlab code to the pseudocode. Note there is a slight difference in the loop structure since we kept track of time t rather than the iteration number n.

Forward Euler Method for solving $u_t = K\,\Delta u$ with Neumann boundary conditions

Input: Initial temperature profile $f(x,y)$
Output: Solution $u(x,y)$ to the Heat Equation
Parameters: Time step Δt, Stopping Time T

Initialize $u^0 = f$
for $n \leftarrow 1$ to $T\,/\,\Delta t$ *(Line 16)*
Calculate the finite difference approximations u_{xx}^n and u_{yy}^n . *(Lines 17-18)*
Update $u^{n+1} = u^n + \Delta t\; K \left[u_{xx}^n + u_{yy}^n\right]$ *(Line 19)*

Figure 1. Pseudocode for numerical solution to the isotropic heat equation. Line numbers in the file heat.m are provided for comparison.

a.) This code is set up to process a grayscale image. For example, try running the code on the cameraman image.

```
A = imread('cameraman.tif');
B = heat (A);
```

b.) To extend isotropic diffusion to color images, we could run the code on each of the 3 RGB channels of A and store the result in the corresponding channel of B.

```
clear B;
A = imread('peppers.png');
for i=1:3; B(:,:,i) = heat(A(:,:,i)); end;
imshow(B);
```

2.) Anisotropic Diffusion

Now we want to look at the harder case when the conductivity $K(x, y)$ is not a constant. The anisotropic Heat Equation is

$$u_t = \nabla \cdot (K \nabla u) = \frac{\partial}{\partial x}(K u_x) + \frac{\partial}{\partial y}(K u_y).$$

The Forward Euler solution will be very similar to #1, except we need to compute the conductivity matrix K and slightly different derivatives.

a.) Let's modify the **heat.m** file from #1. First we want to be able to control the edge-stopping parameter a through the command line. Add the value a as an input in the function line.

```
function [u] = heat (f, a)
```

When we call the function, we will have to specify the value of a.

b.) We will have to calculate the conductivity function K every iteration based on the current image u. Let's use the Perona-Malik exponential edge-stopping function

$$K = e^{-(\,\|\nabla u\|/a\,)^2}$$

where

$$\|\nabla u\| = \sqrt{u_x^2 + u_y^2}.$$

As the first step inside the for loop, calculate the forward differences to approximate the first derivatives u_x and u_y:

$$u_x \approx D_x^+ u = u(x+1, y) - u(x, y)$$
$$u_y \approx D_y^+ u = u(x, y+1) - u(x, y)$$

Next combine these derivatives to compute the magnitude of the gradient as we did in Activity 5. Finally use the Matlab exponential function `exp` to compute the matrix K.

c.) Next we need to replace the second derivative approximations. Rather than calculating u_{xx} directly, we will have to calculate $\frac{\partial}{\partial x}(K u_x)$ in steps by performing forward then backward differences:

$$\frac{\partial}{\partial x}(K u_x) \approx D_x^-(K\, D_x^+ u).$$

Using the same u_x that you used to compute K, perform pointwise multiplication to compute the product $K u_x$ and store it in a matrix P. Then compute a backward difference on P:

$$D_x^- P = P(x, y) - P(x-1, y).$$

Do a similar procedure to approximate $\frac{\partial}{\partial y}(K u_y)$.

d.) Finally we need to change the update step:

Update $u^{n+1} = u^n + \Delta t \; \left[\frac{\partial}{\partial x}(K u_x^n) + \frac{\partial}{\partial y}(K u_y^n)\right]$

e.) Test your code on the grayscale cameraman image with the value $a = 20$.

```
A = imread('cameraman.tif');
B = heat(A,20);
```

Try the values $a = 5$ and $a = 100$. What happens to the image?