A Journey Through Image Inpainting With Partial Differential Equations - Solutions

Created by Mark Goldwater and Jonah Spicher

Content:
- Inpainting with Heat Equation
- Convolutional Kernels
- Discrete Calculus
- Calculus of Variations
- Inpainting inspired by professional restorator approach

References:
- **Scale-Space and Edge Detection Using Anisotropic Diffusion - Perona and Malik**
- **Image Inpainting with the Heat Equation - Kalish**
- **Math 257: Finite Difference Methods**
- **Image Inpainting - Sapio and Ballester**
- **Digital image processing: p054 - Anisotropic Diffusion (Sapiro)**
- **Digital image processing: p053- Calculus of Variations (Sapiro)**
- **create_image_and_mask.m from Parisotto and Schoenlieb at Cambridge**

Inpainting Using the Heat Equation

1. To start, imagine a 1 dimensional image of a solid black bar, as shown below. The center portion of the bar has been obscured from $x = 0$ to $x = L$.

In order to most accurately get values at the edge of our boundary, we should consider the "heat flux" in the image surrounding our boundary. To find this, we find the difference in the value of the two pixels outside of our boundary. For example, if our boundary is at L, then we would find $u(L+1,m) - u(L+2,m)$.

Assuming that $u_x(0,m) = u_x(-1,m)$ is a more accurate approximation than assuming that $u(0,m) = u(-1,m)$, so Neumann boundary conditions allow our inpainting algorithm to fill in gradients more accurately.

(a) Use a centered difference equation

$$f_x(x,t) = \frac{f(x+\Delta x, m) - f(x-\Delta x, m)}{2\Delta m}$$

to rewrite the Neumann boundary conditions at $x = 0$ and $x = L$ for some heat flux $u_x = C$.

Solution: Replacing the function $f(x,t)$ with the function $u(x,t)$, which represents the region we are inpainting, and evaluating it at $x = 0$ and $x = L$ will give the solution to this problem.

$$u_x(0,t) = C \approx \frac{u(0+\Delta x, m) - u(0-\Delta x, m)}{2\Delta m} = \frac{u(\Delta x, m) - u(-\Delta x, m)}{2\Delta m}$$

$$u_x(L,t) = C \approx \frac{u(L+\Delta x, m) - u(L-\Delta x, m)}{2\Delta m}$$

(b) Simplify these expressions to get an expression for $u(-\Delta x, m)$ and $u(L+\Delta x, m)$.

Solution: Solving these expressions for the $u(-\Delta x, m)$ and $u(L+\Delta x, m)$ terms gives us the following two equations.

$$u(-\Delta x, m) = u(\Delta x, m) - 2C\Delta m$$

$$u(L+\Delta x, m) = u(L-\Delta x, m) + 2C\Delta m$$

(c) $u(-\Delta x, m)$ and $u(L+\Delta x, m)$ are the values on u one behind the left-hand boundary of 0 and and one beyond the right-hand boundary of L.

Using the discretized heat equation in the information box above, come up with **two equations** to calculate both u_0^{m+1} and u_L^{m+1}

Solution: The key here is to apply the discretized version of the heat equation which we have previously derived: $u_j^{m+1} = u_j^m + s(u_{j+1}^m - 2u_j^m + u_{j+1}^m)$. If we setup two equations with u_0^{m+1} and u_L^{m+1} on the left-hand side respectively as shown here,

$$u_0^{m+1} = u_0^m + s(u_1^m - 2u_0^m + u_{-1}^m)$$

$$u_L^{m+1} = u_L^m + s(u_{L+1}^m - 2u_L^m + u_{L-1}^m)$$

we can take our equations for $u(-\Delta x, m)$ and $u(L+\Delta x, m)$ from the previous part and substitute them in for u_{-1}^m and u_{L+1}^m respectively to yield the following two equations.

$$u_0^{m+1} = u_0^m + s(2u_1^m - 2u_0^m - 2C\Delta m)$$

$$u_L^{m+1} = u_L^m + s(2C\Delta m - 2u_L^m + 2u_{L-1}^m)$$

2. Now, lets begin to see how we can apply these discretized equations to a MATLAB implementation of the obfuscated bar problem.

(a) Write an expression for the value of an arbitrary pixel j at time $m+1$ using the discretized version of the heat equation for when $0 < j < L$.

Solution: This is simply the discretized heat equation that we have previously derived.

$$u_j^{m+1} = u_j^m + s(u_{j+1}^m - 2u_j^m + u_{j+1}^m)$$

(b) Using the starter code **here**, insert your expressions for I_j^{m+1} for the endpoints of the inpainting region, from **Problem 1 Part c**, and for the inside of the inpainting region from the previous part of this problem. Set $s = \frac{1}{2}$. (*Note:* the actual bar in the code is a two dimensional image, but we are treating each column as one pixel, which makes it one dimensional and not just a singular line of pixels that you would have to squint at to see).

Solution: Download the solution **here**

(c) Can you think of situations where the heat equation would be less effective? What are potential limitations of the heat equation in image inpainting?

Solution: This approach works well only when the areas surrounding each hole are homogeneous. When the hole spans a sharp edge, the edge pixels are diffused, causing the edge to be lost in the restored section as seen in the subsequent figure

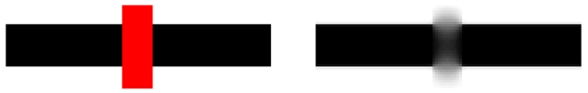

Figure 1: Left: An image of an edge with a hole. Right: the same image reconstructed using the heat equation.

(*Both this solution text and figure are from* ***this*** *Mathworks article.*)

Image Inpainting Via Information Propagation Along Isophotes

1. (a) The kernel is [1 -2 1]. If the two is lined up over x[t], then the discrete convolution $a * x[t]$ will be $x[t-1] - 2x[t] + x[t+1]$, which matches our equation.

   (b) This kernel is a 2D 3x3 matrix:

   $$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

   We can write a discrete approximation of the Laplacian by using two approximations for the second derivative:

   $$I_{xx} + I_{yy} \approx I[x-1,y] - 2I[x,y] + I[x+1,y] + I[x,y-1] - 2I[x,y] + I[x,y+1]$$

   Then, if we imagine placing the center of our matrix over $I[x,y]$, we can fill in the coefficients.

   (c) This is similar to problem a, but with a different approximation and 2D matrices.

   $$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \approx \begin{bmatrix} I[x+1,y] - I[x,y] \\ I[x,y+1] - I[x,y] \end{bmatrix}$$

   This gives the two kernels:

   $$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

   $$\begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2. (a) orig is the image we want to recover. You can use it to compare the results of your algorithm to the correct image. u is the image with a region missing, which requires inpainting. Finally, the mask covers the missing region, and is used to tell the code which part of the image to update.

   (b) i. When you are done, lines 18, 19, and 20 should look like this:

   ```
   hL = [0 1 0;1 -4 1; 0 1 0];
   hX = [0 0 0; 0 -1 1; 0 0 0];
   hY = [0 1 0; 0 -1 0; 0 0 0];
   ```

ii. Lines 44 - 48:

```
lap = imfilter(diff_im(:,:,c),hL,'conv');
gradX = imfilter(diff_im(:,:,c),hX,'conv');
gradY = imfilter(diff_im(:,:,c),hY,'conv');
dlapX = imfilter(lap,hDX,'conv');
dlapY = imfilter(lap,hDY,'conv');
```

(c) Simply replace I_y with gradY, I_x with gradX, and ϵ with 0.1. This gives the following for lines 56 and 57.

```
N_hatX = -gradY ./ sqrt(gradX.^2 + gradY.^2 + 0.1);
N_hatY = gradX ./ sqrt(gradX.^2 + gradY.^2 + 0.1);
```

(d) Since β is the dot product of two vectors:

```
beta = N_hatX.*dlapX + N_hatY.*dlapY;
```

(e) The function call you want specifically is:

```
guillermo_inpaint(u, 800, mask).
```

The result you get should look as follows:

Figure 2: Original image on the left and inpainted image on the right.

Anisotropic Diffusion

1. Let's consider a classic example in the realm of *Calculus of Variations*. Consider a situation where we have a particle at position x_0 at time t_0 and at position x_1 at time t_1 as shown below.

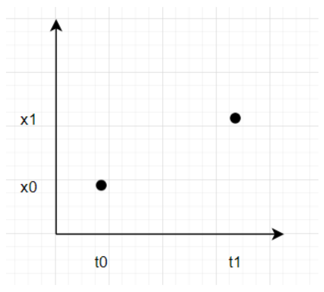

Figure 3: Position vs. Time graph for a particle for which the position is known at t_0 and t_1.

By applying the Euler-Lagrange equation, derive a function that represents the shortest path between these two points that the particle takes (*Hint:* recall that the length of a curve u can be calculated using the following integral: $\int_{x_0}^{x_1} \sqrt{1+u_x^2 dx})$?

Solution: The integral we are trying to optimize when where $F(u, u_x) = \sqrt{1+u_x^2}$ is $\int_{x_0}^{x_1} \sqrt{1+u_x^2} dx$ which is the length of the curve u_x from x_0 to x_1!

Applying the Euler-Lagrange equation to F(u,u$_x) here gives us the following$

$$\frac{u_{xx}}{(1+u_x^2)^{\frac{3}{2}}} = 0$$

In order for this to be true:

$$u_{xx} = 0 \Rightarrow u_x = a \Rightarrow u(x) = ax + b$$

Where a and b are constants. Thus we have proven what we already know: the shortest path between two points is a line drawn between them.

2. Write out the discrete differential equation used to perform gradient descent on the quadratic function $f(x) = x^2$. Given that $x_0 = 3$, calculate x_1, x_2, and x_3.

Solution: If $f(x) = x^2$, then $f_x(x) = 2x$. Knowing this, the difference equation to perform gradient descent is as follows

$$x_{t+1} = x_t - dt(2x_t)$$

Given the starting point $x0 = 3$, we can calculate the first three time steps from here as follows (with $dt = 0.1$

$$x_1 = 2.4 = 3 - 0.1(2(3))$$

$$x_2 = 1.92 = 2.4 - 0.1(2(2.4))$$

$$x_3 = 1.536 = 1.92 - 0.1(2(1.92))$$

If we were to keep iterating we would eventually find the local minimum of x$_t = 0$.

3. Read the Motivation section of the anisotropic diffusion Wikipedia page to see how the initial anisotropic diffusion equation given at the beginning of this section was derived. Nothing to do here, just read!

Solution: N/A

4. Look at Equation 7 in the Original paper for anisotropic diffusion, otherwise known as Perona-Malik diffusion, which is the discretized version of this type of diffusion.

   (a) Derive this discretized version from the original equation.

   Solution: The original version of the equation provided is $I_t = div(c(x,y,t)\nabla I) = c(x,y,t)\nabla I + \nabla c \cdot \nabla I$. Keep in mind that what we are trying to calculate here is $I_{i,j}^{t+1}$, the image at every pixel at the next time step. in order to do this we take the strategy that the paper took and take the current image at time t, $(I_{i,j}^{t})$, and add a four-neighbor discretization of the Laplacian operator on I, each scaled by its appropriate constant as determined by the scalar field $c(x,y,t)$. It is clear looking at the original equation that because the Laplacian is the divergence of the gradient, what we have is a scaled Laplacian. This gives us the final result.

$$I_{i,j}^{t+1} = I_{i,j}^{t} + \lambda[c_N \cdot \nabla_N I + c_S \cdot \nabla_S I + c_E \cdot \nabla_E I + c_W \cdot \nabla_W I]$$

   Note that $\nabla_{N,S,E,W}$ are not gradient operators, but nearest neighbor differences such that

$$\begin{aligned}\nabla_N I_{i,j} &\equiv I_{i-1,j} - I_{i,j}\\ \nabla_S I_{i,j} &\equiv I_{i+1,j} - I_{i,j}\\ \nabla_E I_{i,j} &\equiv I_{i,j+1} - I_{i,j}\\ \nabla_W I_{i,j} &\equiv I_{i,j-1} - I_{i,j}\end{aligned}$$

   and $c_{N,S,E,W}$ is the chosen smoothing function evaluated with the input of its respective directional differences. If it's still not clear why this is the correct solution, try comparing it to how you came up with the Laplacian convolutional Kernel. The equation for a discretized Laplacian in two dimensions should look pretty similar to this after some substitutions with the nearest neighbor difference operators.

   (b) Take a look at the implementation in **anisodiff2D.m** and make sense of it using your knowledge of convolutional kernel tricks from earlier in the problem set and part (a) of this problem.

   Solution: N/A