# Partial Differential Equations - Class Notes

Steven Blythe

March 23, 2022

## Project Information

If you want to mess with an image, you have to think about how computers view images. The image is stored. When we see it, we see an image, but what does the computer see?

Computers stores images as a point of pixels akin to a grid. Pixels contain RGB value information and location. Pictures are generally large.

The first thing is: An image is a grid of numbers (Array, matrix). This is what the computer sees.

Color: Saturation, hue, not the focus.

Focus: Grayscale, same functions, can be extended into color. Add extra dimensions.

Idea: Have a grid of values with pixels and a value at each point (Tells us the gray scale, point of scale between black and white).

### Image Processing

Let's say we have a black and white digital image.

Here, we have:

- $u(x, y)$ : $u$ (int) is the darkness (grayscale) of the image at point $(x, y)$.
- $u = 0$ black, $u = 255$ = white

Steps to our process:

1. Remove Noise. In the case of our project, we will remove Salt and Pepper noise.
2. Remove Blur.

We know Heat, Laplace, Wave, and Transport.

We want to use Heat.

Here, we want to remove the 'spikes' in our image. The idea is if we apply heat to our point, the peak of the spike reduces in magnitude and spreads out.

When we smoothen our picture, we also blur our ideal picture.

1. First step: Take an image and blur it.

   The heat equation blurs things. This will cause the salt and pepper noise fade.

   Here, let us consider an initial image with noise. Our initial image has $t = 0$. Then, we apply

   $$u_t = u_{xx} + u_{yy} \tag{1}$$
   $$u(x, y, 0) = f(x, y) \tag{2}$$

   Here, $f(x, y)$ is our image, the initial condition.

   However, when we apply our heat function, we have a pro and a con:

   - Pro: Salt and pepper noise are pretty much gone.
   - Con: Whole image is blurred.

   If we just have one point of noise, Let's say we have an $m \times n$ white grid and black dot in the center, then let us consider the following:

   $$f(x, y) = \delta\left(x - \frac{L}{2}\right)\delta\left(y - \frac{M}{2}\right) \tag{3}$$

   Here, we have the following conditions:

   (a) $u(x, y, 0) = f(x, y)$

   and

   (a) $u(0, y, t) = f(0, y)$

   (b) $u(L, y, t) = f(L, y)$

   (c) $u(x, 0, t) = f(x, 0)$

   (d) $u(x, M, t) = f(x, M)$

## March 11, 2022

How do you get rid of Salt and Pepper noise without blurring the image at the same time?

The heat equation removes the noise, but blurs as well.

Selective blur: Blur in some direction. It all depends on the boundary.

If we are parallel to the boundary, we could care less.

We want to blur in the direction perpendicular to the gradient.

We are going to try to modify the heat equation so that it does not blur the edge.

Here, we define:

- $\eta$ : Direction of gradient
- $\xi$ : Direction normal to gradient

To blur only in the direction perpendicular to $\nabla u$, use

$$u_t = u_{xx} + u_{yy} - u_{yy} \tag{4}$$
$$u_t = u_{xx} \tag{5}$$

Here, $\Delta u$ is $u_{xx} + u_{yy}$ and the component in the direction of $\nabla u$ is $u_{yy}$.

The equation that blurs only in the direction normal to the gradient is

$$u_t = \Delta u - u_{\eta\eta} \tag{6}$$
$$= u_{\xi\xi} + u_{\eta\eta} - u_{\eta\eta} \tag{7}$$
$$= u_{\xi\xi} \tag{8}$$

Note: $\Delta u = u_{xx} + u_{yy} = u_{\xi\xi} + u_{\eta\eta}$ since $\xi \perp \eta$ and they are unit vectors.

How do we express $u_{\eta\eta}$ in terms of $u_x, u_y, u_{xx}, u_{yy}, u_{xy}$?

Let $\vec{n}$ be a unit vector.

$$u_n = \nabla u \cdot \vec{n} \tag{9}$$

This is called the directional derivative (Calc III)

$$u_{nn} = (u_n)_n \tag{10}$$
$$= \nabla u_n \cdot \vec{n} \tag{11}$$
$$= \nabla(\nabla u \cdot \vec{n}) \cdot \vec{n} \tag{12}$$
$$= \nabla\nabla u \vec{n} \cdot \vec{n} \tag{13}$$

Here, $\nabla\nabla u$ is the tensor. The tensor is also called the Hessian Matrix.

$\vec{u} \cdot \vec{v} = \vec{v}^T \vec{u}$

$\vec{n} = n_1 \hat{i} + n_2 \hat{j}$

$$= \begin{bmatrix} n_1 & n_2 \end{bmatrix} \begin{bmatrix} u_{xx} & u_{xy} \\ u_{xy} & u_{yy} \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \tag{14}$$
$$= \begin{bmatrix} n_1 & n_2 \end{bmatrix} \begin{bmatrix} u_{xx}n_1 + u_{xy}n_2 \\ u_{xy}n_1 + u_{yy}n_2 \end{bmatrix} \tag{15}$$
$$= n_1(u_{xx}n_1 + u_{xy}n_2) + n_2(u_{xy}n_1 + u_{yy}n_2) \tag{16}$$
$$= n_1^2 u_{xx} + 2n_1 n_2 u_{xy} + n_2^2 u_{yy} \tag{17}$$

We know the following:

$$\vec{\nabla} = \frac{\nabla}{||\nabla u||} \tag{18}$$

$$= \frac{\langle u_x, u_y \rangle}{\sqrt{u_x^2 + u_y^2}} \tag{19}$$

$$= \left\langle \frac{u_x}{\sqrt{u_x^2 + u_y^2}}, \frac{u_y}{\sqrt{u_x^2 + u_y^2}} \right\rangle \tag{20}$$

$$\vec{\xi} = \left\langle -\frac{u_y}{\sqrt{u_x^2 + u_y^2}}, \frac{u_x}{\sqrt{u_x^2 + u_y^2}} \right\rangle \tag{21}$$

Here, let us call the first vector $n_1$ and the second $n_2$. Now:

$$u_{\xi\xi} = \frac{u_y^2}{u_x^2 + u_y^2} u_{xx} - \frac{2u_x u_y}{u_x^2 + u_y^2} u_{xy} + \frac{u_x^2}{u_x^2 + u_y^2} u_{yy} \tag{22}$$

$$= \frac{u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}}{u_x^2 + u_y^2} \tag{23}$$

Here, let us write:

$$u_t = u_{\xi\xi} \tag{24}$$

$$\Downarrow \tag{25}$$

$$u_t = \frac{u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}}{u_x^2 + u_y^2} \tag{26}$$

This was given credit to James Sethian (Berkeley) in 1988. This is called the Level Set Equation (Mean Curvature Equation)

Idea: We have a picture with three different circles with different sizes. The circle with the smallest radius has the largest curvature. The Mean Curvature Equation attacks the circles with the smallest curvatures first, so our salt and pepper noise, which can be seen as a minute circle, is attacked first.

When the level set equation is applied to an image, the boundaries of each level set move with speed proportional to their curvature.

One thing to note about two circles: Smaller circles become smaller at a faster rate than bigger circles since the radius is smaller and the curvature is bigger.

**Grayson's Theorem** (For 2-D level sets)

For any closed simple curve, as it evolves under the influence of $u_t = u_{\xi\xi}$, the curve the curve becomes more and more circle-like and disappears as a single point.

## March 21, 2022

Effect on an image with Salt and Pepper Noise

When applying our our algorith, the circle around the smile is thicker since inside boundary of circle has large curvature and goes in faster.

The dots are like circles with really large curvatures so they go away quickly.

At a small T: Eliminates noise. At a large T: The image shrinks and mutates.

Where we have a curved line running through the boundaries of the picture, when do you stop? When the image looks the "Best."

Can we modify level set to work better?

Idea: use $u_t = u_{\xi\xi} - \alpha(u - f(x, y))$, where $\alpha > 0$.

$u$ is the current image, $f$ is the initial image. The first term pushes away, modifies, and distorts, whereas the second term wants to bring us back to the original image.

Let's consider $u_t = -\alpha(u - f(x))$, where our straight line is $f(x)$ is the modified image and $u$ is the original. Where the gaps are bigger in our image, the change is greater.

If you run this method, the $u_{\xi\xi}$ will tend to blur in the orthogonal direction (to gradient) and $-\alpha(u-f)$ will tend toward the initial picture.

There two terms compete with each other and eventually the .e.net stabilizes as $t \to \infty$ (steady-state)

The idea is that you get a method that removes the noise but doesn't deviate too much from the original image. $\alpha$ is arbitrary, so you find the $\alpha$ that works best.

**Deblurring an Image**

We know that the heat equation will blur an image. $u_t = u_{xx} + u_{yy}$.

To deblur, perhaps we could run the backward heat equation $u_t = -(u_{xx} + u_{yy})$, but we know this is unstable and ill-posed.

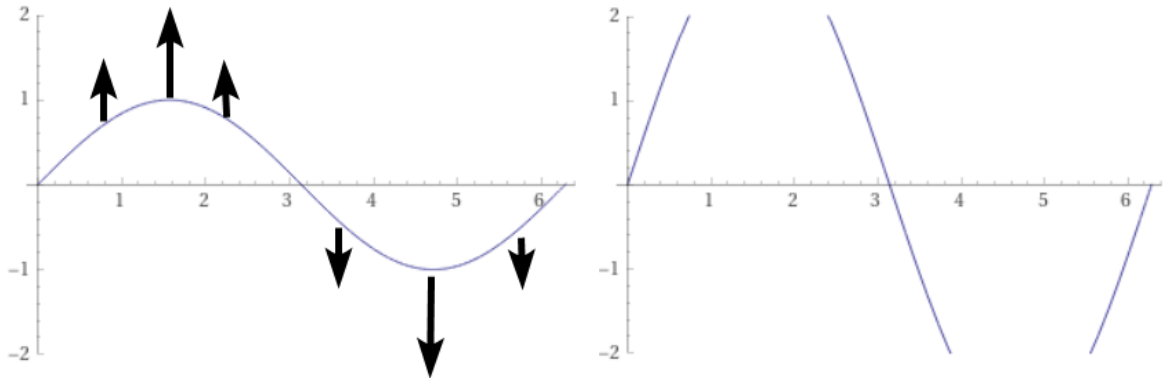
Perhaps we could tweak the backward heat equation?

Idea $u_t = -\alpha u_{xx}, \alpha > 0$ is unstable.

However, $u_t = -|u_x|u_{xx}$ is not ill-posed

Why is our new equation not ill-posed? Let us consider the following function:

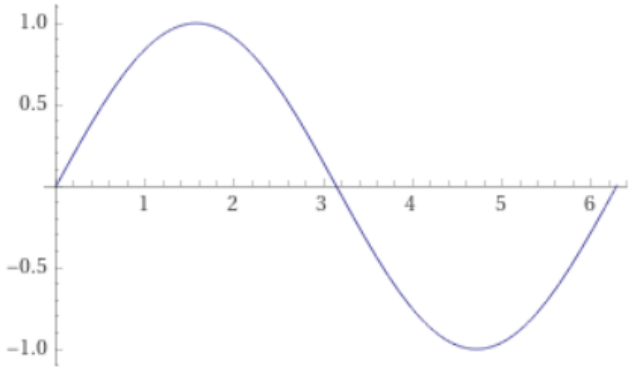
$$\begin{aligned} u_t &= -u_{xx} \\ u(x,0) &= \sin x \\ u_{xx} &= -\sin x \\ u_t &= \sin x \end{aligned}$$




$u(x,t) = e^t \sin x$, the amplitude tends to infinity.

If we apply the shock filter, then:

$$\begin{aligned} u_t &= -|u_x|u_{xx} \\ u(x,0) &= \sin x \\ u_t &= |cosx| \sin x \end{aligned}$$




Here, where $u_x = 0$, there is no change. Opposed to our first image, our graph will no longer increase its amplitude drastically. The image is stable at the peaks now and the change is greatest at the midpoints. As time goes on, the curve would beome boxy.

As time goes on, the new image will start taking the save of square sine waves, but will eventually exhibit a graph akin to discontinuous lines at the peaks with a point at the mid point, similar to the infinite fourier.

Note:

1. The solution does not blow up
2. Discontinuities are encouraged

## March 23, 2022

Definition: Discontinuities that appear in solutions as time progresses are called shocks.

Objective of our algorithm $u_t = -|u_x|u_{xx}|$: Places where the rate of change is 0 exhibits no change, and places near 0 rate of change experiences minimal change. Inflection points also exhibit no change.

The algorithm pushes towards a piecewise constant solution where the values of each constant part corresponds to the values of $f$ where $f' = 0$.

The endpoints of each constant interval are where $f'' = 0$.

Note: The shock filter equation will make noise worse.

Note: 2-D Version

$$u_t = -|\nabla u|\Delta u \tag{27}$$
$$= -\sqrt{u_x^2 + u_y^2}(u_{xx} + u_{yy}) \tag{28}$$

Recall $\nabla u = \langle u_x, u_y \rangle$

**Implementation**

It is important to let a computer know how to find a derivative. With an image, we have a box of numbers (Array/matrix) How do we take these numbers and create derivatives? $u$ is our image Our computer simply has pixel values (grayscale) If we are finding $u_x$, what does it mean to the computer?

Derivatives - Numerically

On a computer, we can only specify the value of a function at certain values.

We will have a function at many points.

Looking at $o$, what does the derivative represent?

The derivative represents a slope.

We can do rise over run, but we must have two points.

We can use the point to the right (Forward Difference)

We can also use the point to the left (Backward Difference)

Using both is called Centered Difference, which is preferred.

Let's say we want to approximate $u'(x_0)$.

Let use consider:

$$u_1 = u(x_1) \tag{29}$$
$$u_2 = u(x_2) \tag{30}$$
$$\vdots \tag{31}$$

We could do the following:

- Forward Difference

$$u'(x_2) \approx \frac{u_3 - u_2}{\Delta x} \tag{32}$$

- Backward Difference

$$u'(x_2) \approx \frac{u_2 - u_1}{\Delta x} \tag{33}$$

- Centered Difference

$$u'(x_2) \approx \frac{u_3 - u_1}{2\Delta x} \tag{34}$$

Here, let us consider $j$ as the $x-$dimension, $k$ as the $y-$dimension, and $n$ as the $t-$dimension.:

$$u_{j,k}^n = u(x_j, y_k, t^n) \tag{35}$$
$$= u(j\Delta x, k\Delta y, n\Delta t) \tag{36}$$

Here, we assume $x_0 = y_0 = t^0 = 0$.

Here, if we want to find the $x-$partial with forward difference:

$$u_x(x_j, t_k, t^n) = {u_x}^n{}_{j,k} \tag{37}$$

$$\approx \frac{u^n_{j+1,k} - u^n_{j,k}}{\Delta x} \tag{38}$$

If we want to consider change in $y$ with forward difference,

$${u_y}^n{}_{j,k} \approx \frac{u^n_{j,k+1} - u^n_{j,k}}{\Delta y} \tag{39}$$

Now, let us consider $x$ again, but in terms of backward difference:

$${u_x}^n{}_{j,k} \approx \frac{u^n_{j,k} - u^n_{j-1,k}}{\Delta x} \tag{40}$$

**Second Derivatives Numerically**

Here, let us use forward difference for our next equation:

$$u_{xx} = (u_x)_x{}^n{}_{j,k} = \frac{(u_x)^n_{j+1,k} - (u_x)^n_{j,k}}{\Delta x} \tag{41}$$

Here, we found our terms in the previous equation,

$$\approx \frac{\frac{u^n_{j+1,k} - u^n_{j-1,k}}{\Delta x} - \frac{u^n_{j,k} - u^n_{j-1,k}}{\Delta x}}{\Delta x} \tag{42}$$

$$\approx \frac{u^n_{j+1,k} - 2u^n_{j,k} + u^n_{j-1,k}}{(\Delta x)^2} \tag{43}$$

Let us consider centered for both,

$$(u_{xy})^n_{j,k} = (u_x)_y{}^n{}_{j,k} \tag{44}$$

$$\approx \frac{(u_x)^n_{j,k+1} - (u_x)^n_{j,k-1}}{2\Delta y} \tag{45}$$

$$= \frac{\frac{u^n_{j+1,k+1} - u^n_{j-1,k+1}}{2\Delta x} - \frac{u^n_{j+1,k-1} - u^n_{j-1,k-1}}{2\Delta x}}{2\Delta y} \tag{46}$$

$$= \frac{u^n_{j+1,k+1} - u^n_{j-1,k+1} - u^n_{j+1,k-1} + u^n_{j-1,k-1}}{4\Delta x \Delta y} \tag{47}$$

**Equation Simulation**

*See comments, bpeqs*

Only update the interior points.

*See comments, bpeqspts*

Here, we have $u(x, y, 0)$ as the original image.

We want to step up the value of time.

**Heat Equation**

We are able to make $\Delta x$ and $\Delta y$. We want to make both a length of 1.

$$u_t = u_{xx} + u_{yy} \tag{48}$$

Here, let us consider forward difference in time:

$$\frac{u^{n+1}_{j,k} - u^n_{j,k}}{\Delta t} = \frac{u^n_{j+1,k} - 2u^n_{j,k} + u^n_{j-1,k}}{(\Delta x)^2} + \frac{u^n_{j,k+1} - 2u^n_{j,k} + u^n_{j,k-1}}{(\Delta y)^2} \tag{49}$$

Here, let $\Delta x = \Delta y = h$.

$$\frac{u^{n+1}_{j,k} - u^n_{j,k}}{\Delta t} = \frac{u^n_{j+1,k} + u^n_{j-1,k} - 4u^n_{j,k} + u^n_{j,k+1} + u^n_{j,k-1}}{h^2} \tag{50}$$

Here, we have all the old image and we want to find the new image.