A Comparison of Different Image Denoising Variants using Partial Differential Equations

sahanajoshi1996

April 2022

1 Introduction

The need to denoise images is all too common and has wide range of applications, ranging from surveillance, medical imagery, terrain mapping, locomotion planning, etc. It is also a common data cleaning step before the image is ready for other vision tasks such as classification and localization. This project will explore image denoising using the tools of partial differential equations. It will also explore the effect of choosing different functions which smooth the image at different rates. There is a detailed analysis for each function with examples and plots.

2 Problem Setup

Let the image that we are trying to denoise be $I(x, y)$ with x and y denoting the location of a pixel. The energy functional $E(I(x, y))$ when framed to compute the noise in the image, acts as a measure of noise, which can reduced by gradient descent. The equation below shows the formulation of the energy functional framed to reduce noise:

$$E(I(x, y)) = \int_{\Omega} c(||\nabla I||)dxdy \tag{1}$$

where Ω is the domain of the image, and c is any non-decreasing function. To make the notation more explicit, $c(||\nabla I||)$ is $L(I, I_x, I_y, x, y)$. **The intuition is that noisy pixels are oddities compared to their background and hence induce a gradient, by reducing the overall gradient of the image, or in a general case, a non-decreasing function of the gradient, the noise in the image also reduces.**

To reduce E, the gradient of E with respect to I should be the direction of update I(but with a negative sign for descent), given by:

$$I_t = -\nabla_I E \tag{2}$$

where t is the gradient descent fictitious time variable.

The Euler Lagrange equation below shows the relation between the partial derivatives of L with respect to I, I_x and I_y as with the gradient of E.

$$\nabla_I E = L_I - \frac{\partial L_{I_x}}{\partial x} - \frac{\partial L_{I_y}}{\partial y} \tag{3}$$

Plugging in L in eqn. 3 and simplifying, we get :

$$I_t = \nabla.[\frac{\dot{c}(||\nabla I||)}{||\nabla I||}\nabla I] \tag{4}$$

Eqn 4 characterizes Perona-Malik diffusion Depending on different functions chosen for c, we get different denoising effects. In this project, we will explore three options for c**: (1)** $c(l) = \frac{l^2}{2}$**, (2)** $c(l) = l$ **and (3)** $c(l) = \log(l)$.

3 Implementation

3.1 $c(l) = \frac{l^2}{2}$

3.1.1 Continuous PDE

Using c in 4:

$$\dot{c}(||\nabla I||) = \frac{1}{||\nabla I||} \tag{5}$$

which makes the PDE

$$I_t = I_{xx} + I_{yy} \tag{6}$$

3.1.2 Discretize and Compute CFL condition

This is a 2D equation and hence would require 2D Fourier transform to compute the CFL condition. Leveraging the symmetry of the problem, we can compute the CFL condition for 1D heat equation $I_t = bI_{xx}$ (where b is the diffusion coefficient) and replace Δx with $\frac{\Delta x}{\sqrt{2}}$. The CFL condition for the 1D heat equation (as derived in class with forward difference for time variable and central difference for space variable is :

$$\Delta t \leq \frac{(\Delta x)^2}{2b} \tag{7}$$

Replacing Δx with $\frac{\Delta x}{\sqrt{2}}$ and b as 1:

$$\Delta t \leq \frac{(\Delta x)^2}{4} = \frac{(\Delta y)^2}{4} \tag{8}$$

3.1.3 Finite gradient update equation

$$\frac{I(t+1)-I(t)}{\Delta t} = \frac{I(t,x+\Delta x,y)+I(t,x-\Delta x,y)-2I(t,x,y)}{\Delta x^2} + \frac{I(t,x,y+\Delta y)+I(t,x,y-\Delta y)-2I(t,x,y)}{\Delta x^2} \tag{9}$$

So the update step is:

$$I(t+1) = \Delta t \frac{I(t,x+\Delta x,y)+I(t,x-\Delta x,y)+I(t,x,y+\Delta y)+I(t,x,y-\Delta y)-4I(t,x,y)}{\Delta x^2} + I(t) \tag{10}$$

3.1.4 Code

Here is a snapshot of the code, full code can be found in Jupyter Notebook

3.2 $c(l) = l$

3.2.1 Continuous PDE

Using c in 4:

$$\dot{c}(||\nabla I||) = 1 \tag{11}$$

which makes the PDE:

$$I_t = \frac{I_x^2 I_{yy} - 2I_x I_y I_{xy} + I_y^2 I_{xx}}{(I_x^2 + I_y^2)^{\frac{3}{2}}} \tag{12}$$

But this might (will) lead to I_t going to ∞ when the norm of the gradient goes to 0 because the numerator has a second order on I_x and I_y and the denominator has a third order. To avoid this, we modify the energy functional to the integral of $c(\sqrt{I_x^2 + I_y^2 + \epsilon^2})$. Reworking the Euler Lagrange equation, the PDE is now:

$$I_t = \frac{I_x^2 I_{yy} - 2I_x I_y I_{xy} + I_y^2 I_{xx} + \epsilon^2(I_{xx} + I_{yy})}{(I_x^2 + I_y^2 + \epsilon^2)^{\frac{3}{2}}} \tag{13}$$

```
for iter in range(iterations):
    prev_image = np.copy(noisy_image)

    for i in range(del_x, dim_x - del_x):
        for j in range(del_y, dim_y - del_y):
            noisy_image[i][j] = prev_image[i][j] + (del_t / (del_x ** 2)) * \
            (prev_image[i + del_x][j] + prev_image[i - del_x][j] + \
            prev_image[i][j + del_y] + prev_image[i][j - del_y] + \
            - 4 * prev_image[i][j])
```

Figure 1: Code for linear equation

3.2.2 Discretize and Compute CFL condition

The equation above resembles the geometric heat equation with diffusion only in the tangential direction ξ with diffusion constant as $\frac{1}{||\nabla I||}$. The CFL condition for this is

$$\begin{aligned} \Delta t &\leq \frac{1}{2b}\Delta x^2 \\ &\leq \frac{||\nabla I||}{2}\Delta x^2 \end{aligned} \tag{14}$$

To find the largest possible timestep we can take, we need to find the smallest value $||\nabla I||$ can take, which is ϵ. Using this, the CFL condition is:

$$\Delta t \leq \frac{\epsilon}{2}\Delta x^2 \tag{15}$$

To pick an ϵ value which is helpful, we can use the histogram of the gradients across the image are choose ϵ in the range of the gradients.

3.2.3 Finite gradient update equation

Using forward difference for the time variable and central differences for the first and second derivative terms of the space variables, we have(ignoring variables which remain constant to keep equations concise):

$$\begin{aligned} I_t &= \frac{I(t+1) - I(t)}{\Delta t} \\ I_x &= \frac{I(x+\Delta x) - I(x-\Delta x)}{2\Delta x} \\ I_y &= \frac{I(y+\Delta y) - I(y-\Delta y)}{2\Delta y} \\ I_{xx} &= \frac{I(x+\Delta x) - 2I(x) + I(x-\Delta x)}{\Delta x^2} \\ I_{yy} &= \frac{I(y+\Delta y) - 2I(y) + I(y-\Delta y)}{\Delta y^2} \end{aligned}$$

$$I_{xy} = \frac{I(x+\Delta x, y+\Delta y) + I(x-\Delta x, y-\Delta y) - I(x-\Delta x, y+\Delta y) - I(x+\Delta x, y-\Delta y)}{4\Delta x \Delta y} \tag{16}$$

which can be used in Equation 13 to compute the gradient update.

3.2.4 Code

Here is a snapshot of the code, full code can be found in Jupyter Notebook

3.3 $c(l) = \log l$

3.3.1 Continuous PDE

Using c in 4:

$$\dot{c}(||\nabla I||) = \frac{1}{||\nabla I||} \tag{17}$$

$$\begin{aligned} I_t &= \nabla.[\frac{1}{||\nabla I||^2}\nabla I] \\ &= \frac{\partial}{\partial x}[\frac{I_x}{I_x^2 + I_y^2}] + \frac{\partial}{\partial y}[\frac{I_y}{I_x^2 + I_y^2}] \\ &= \frac{I_{xx}[I_x^2 + I_y^2] - [2I_xI_{xx} + 2I_yI_{xy}]I_x + I_{yy}[I_x^2 + I_y^2] - [2I_yI_{yy} + 2I_xI_{xy}]I_y}{(I_x^2 + I_y^2)^2} \\ &= \frac{I_{xx} + I_{yy}}{I_x^2 + I_y^2} - \frac{2[I_x^2I_{xx} + 2I_xI_yI_{xy} + I_y^2I_{yy}]}{(I_x^2 + I_y^2)^2} \end{aligned} \tag{18}$$

3.3.2 Discretize and Compute CFL condition

There are two problems here : (1)Since it is a non-linear PDE, we cannot directly apply Fourier transform to arrive at the CFL condition. (2) There is also the squared-gradient term in the denominator which can go to zero when I_x and I_y are both 0 making it impossible to take a time-step when this situation is

```
for iter in range(iterations):
    prev_image = np.copy(noisy_image)

    for i in range(del_x, dim_x - del_x):
        for j in range(del_y, dim_y - del_y):
            ix = calc_ix(prev_image, i, j)
            iy = calc_iy(prev_image, i, j)
            ixx = calc_ixx(prev_image, i, j)
            iyy = calc_iyy(prev_image, i, j)
            ixy = calc_ixy(prev_image, i, j)

            _sum = (ix ** 2) * iyy + (iy ** 2) * ixx

            _sum += -2 * ix * iy * ixy + (eps ** 2) * (ixx + iyy)
            _sum /= (ix ** 2 + iy ** 2 + eps ** 2) ** (3 / 2)

            noisy_image[i][j] = prev_image[i][j] + del_t * _sum
```

Figure 2: Code for geometric equation

encountered. Focusing on the second problem, we can modify energy equation by adding a constant ϵ which helps us in circumventing this situation as shown below:

$$\dot{c}(||\nabla I||) = \frac{1}{||\nabla I||} = \frac{1}{\sqrt{I_x^2 + I_y^2 + \epsilon^2}} \tag{19}$$

Using this change we can get :

$$\begin{aligned} I_t &= \frac{\partial}{\partial x}[\frac{I_x}{I_x^2 + I_y^2 + \epsilon^2}] + \frac{\partial}{\partial y}[\frac{I_y}{I_x^2 + I_y^2 + \epsilon^2}] \\ &= \frac{I_{xx}[I_x^2 + I_y^2 + \epsilon^2] + I_{yy}[I_x^2 + I_y^2 + \epsilon^2] - I_x[2I_xI_{xx} + 2I_yI_{xy}] - I_y[2I_yI_{yy} + 2I_xI_{xy}]}{(I_x^2 + I_y^2 + \epsilon^2)^2} \\ &= \frac{[I_{xx} + I_{yy}]\epsilon^2 + I_{xx}[I_y^2 - I_x^2] + I_{yy}[I_x^2 - I_y^2] - 4I_xI_yI_{xy}}{(I_x^2 + I_y^2 + \epsilon^2)^2} \\ &= \frac{[I_{xx} + I_{yy}]\epsilon^2 + [I_{xx} - I_{yy}][I_y^2 - I_x^2] - 4I_xI_yI_{xy}}{(I_x^2 + I_y^2 + \epsilon^2)^2} \end{aligned} \tag{20}$$

Luckily, this PDE is quasi-linear, i.e, the highest order derivatives are of first degree, so we can linearize this by assuming first-order derviatives as constants and then compute the CFL condition :)

Let $I_x = a$ and $I_y = b$

$$\begin{aligned} I_t &= \frac{[I_{xx} + I_{yy}]\epsilon^2 + [I_{xx} - I_{yy}][b^2 - a^2] - 4abI_{xy}}{(a^2 + b^2 + \epsilon^2)^2} \\ &= \frac{I_{xx}[\epsilon^2 + b^2 - a^2] + I_{yy}[\epsilon^2 + a^2 - b^2] - 4abI_{xy}}{(a^2 + b^2 + \epsilon^2)^2} \end{aligned} \tag{21}$$

This is a 1-D diffusion is a particular direction determined by ϵ, b and a. The diffusion coefficient is determined by the coefficients of I_{xx} and I_{yy}

$$\begin{aligned} coeffI_{xx} &= \frac{\epsilon^2 + a^2 - b^2}{(\epsilon^2 + a^2 + b^2)^2} = t^2 \\ coeffI_{yy} &= \frac{\epsilon^2 + b^2 - a^2}{(\epsilon^2 + a^2 + b^2)^2} = u^2 \end{aligned} \tag{22}$$

Diffusion coefficient m is given by:

$$\begin{aligned} m &= \sqrt{t^2 + u^2} \\ &= \sqrt{\frac{\epsilon^2 + b^2 - a^2 + \epsilon^2 + a^2 - b^2}{(\epsilon^2 + a^2 + b^2)^2}} \\ &= \frac{\sqrt{2}\epsilon}{\epsilon^2 + a^2 + b^2} \end{aligned} \tag{23}$$

Reusing Eqn. 14 with $b = \frac{\sqrt{2}\epsilon}{\epsilon^2+a^2+b^2}$, we get:

$$\Delta t \leq \frac{\epsilon^2 + a^2 + b^2}{2\sqrt{2}\epsilon}\Delta x^2 \tag{24}$$

The tightest time-step happens when $a^2 + b^2$ is 0, or gradient is 0 which leads to the CFL condition:

$$\Delta t \leq \frac{\epsilon}{2\sqrt{2}}\Delta x^2 \tag{25}$$

3.3.3 Finite gradient update equation

Using forward difference for the time variable and central differences for the first and second derivative terms of the space variables, we have(ignoring variables which remain constant to keep equations concise):

$$\begin{aligned}
I_t &= \frac{I(t+1) - I(t)}{\Delta t} \\
I_x &= \frac{I(x+\Delta x) - I(x-\Delta x)}{2\Delta x} \\
I_y &= \frac{I(y+\Delta y) - I(y-\Delta y)}{2\Delta y} \\
I_{xx} &= \frac{I(x+\Delta x) - 2I(x) + I(x-\Delta x)}{\Delta x^2} \\
I_{yy} &= \frac{I(y+\Delta y) - 2I(y) + I(y-\Delta y)}{\Delta y^2}
\end{aligned}$$

$$I_{xy} = \frac{I(x+\Delta x, y+\Delta y) + I(x-\Delta x, y-\Delta y) - I(x-\Delta x, y+\Delta y) - I(x+\Delta x, y-\Delta y)}{4\Delta x \Delta y} \tag{26}$$

which can be used in Equation 20 to compute the gradient update.

3.3.4 Code

Here is a snapshot of the code, full code can be found in Jupyter Notebook

4 Results and Analysis

4.1 Energy functional Plot

When performing gradient descent, the energy functional should reduce with each iteration. This can be used as a smoke test to ensure that the functional is reducing, the same is shown in the images below.

```python
for iter in range(iterations):
    prev_image = np.copy(noisy_image)

    for i in range(del_x, dim_x - del_x):
        for j in range(del_y, dim_y - del_y):
            ix = calc_ix(prev_image, i, j)
            iy = calc_iy(prev_image, i, j)
            ixx = calc_ixx(prev_image, i, j)
            iyy = calc_iyy(prev_image, i, j)
            ixy = calc_ixy(prev_image, i, j)

            #todo
            _sum = (ixx + iyy) * (eps ** 2) + (ixx - iyy) * (iy ** 2 - ix ** 2)

            _sum -= -4 * ix * iy * ixy
            _sum /= (ix ** 2 + iy ** 2 + eps ** 2) ** (2)

            noisy_image[i][j] = prev_image[i][j] + del_t * _sum
```

Figure 3: Code for log-equation

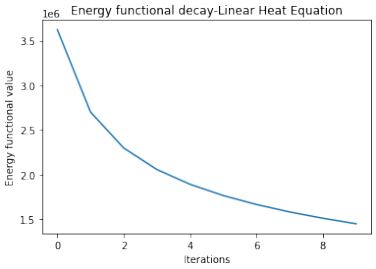

Figure 4: $c(l) = \frac{l^2}{2}$

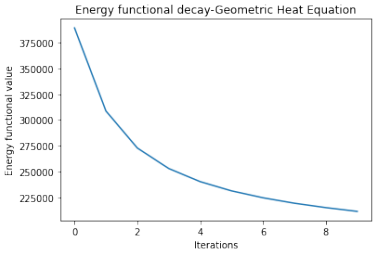

Figure 5: $c(l) = l$

4.2 Qualitative results

We take an original, clean image and add Gaussian noise to it as shown in the figure below.

Following are the denoising processes at different iterations.

The geometric heat equation does a much better job at denoising compared to the linear heat equation. It retains the sharpness of edges because the diffu-

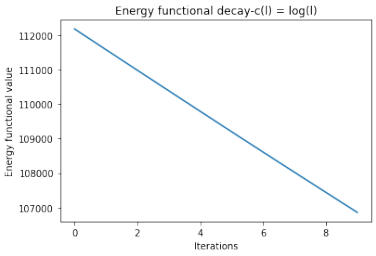

Figure 6: $c(l) = log(l)$

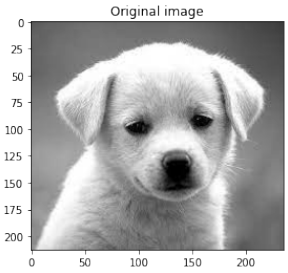

Figure 7: Original Image

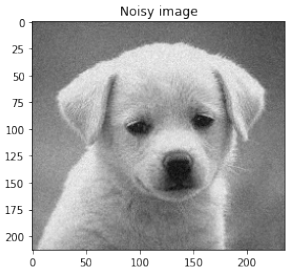

Figure 8: Noisy Image

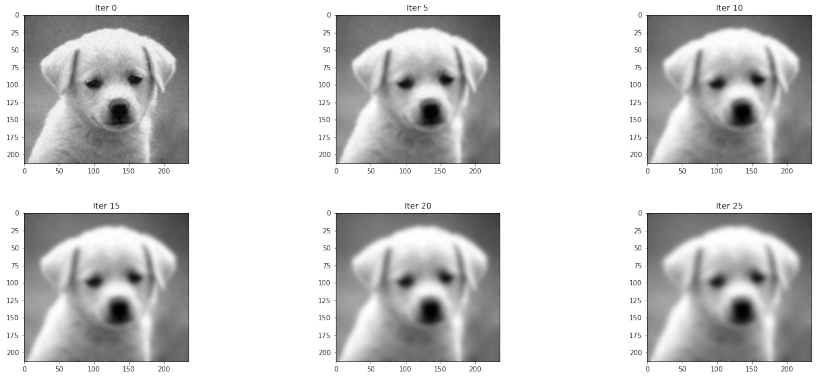

Figure 9: Denoising using Linear Heat Equation

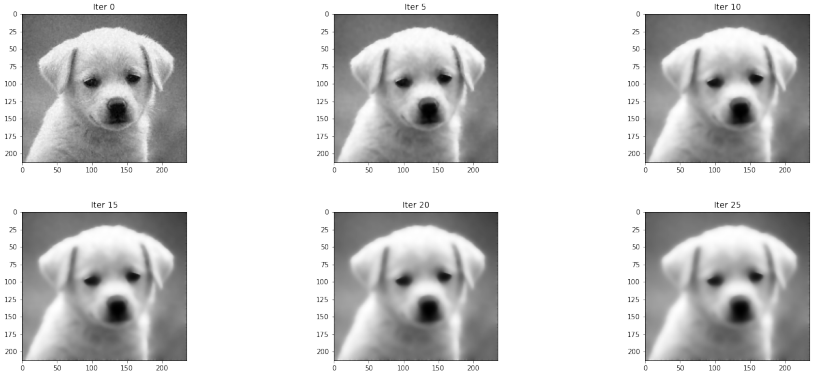

Figure 10: Denoising using Geometric Heat Equation

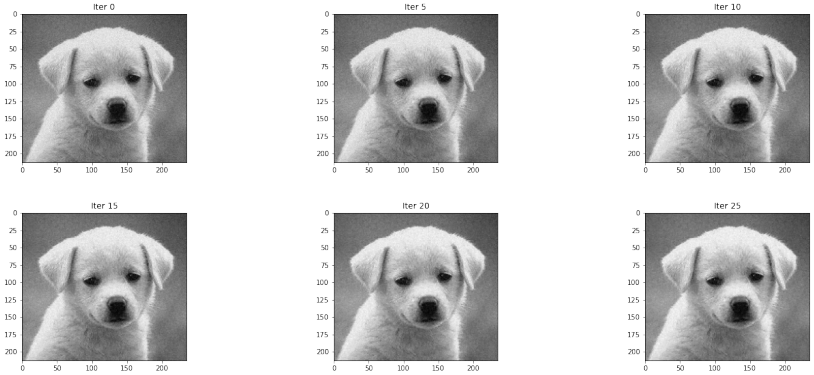

Figure 11: Denoising using Log-Equation

sion happens only along the tangential direction of the edges. The linear heat equation does a better job than log-equation, my intuition is that the log function maps a large domain to a smaller range compared to linear and quadratic, making gradients which are fairly far apart result in similar energies.

ϵ plays a key role in the denoising process: if it is too large, the effect resembles the linear heat equation, if it is too small, it would make the gradient update step unstable.

4.3 Quantitative results

To see which method performs better, we take the mean-squared error(MSE) between original image and the denoised image at different iterations and be-

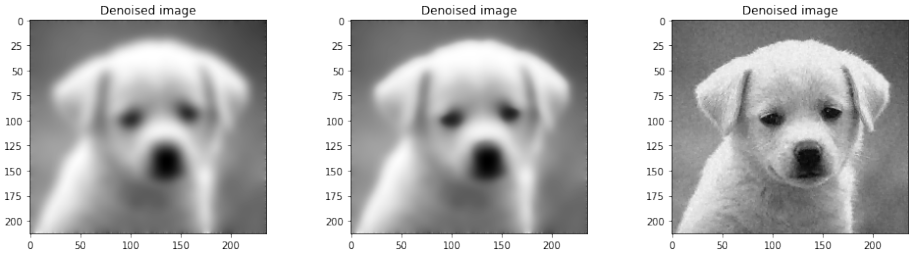

Figure 12: Linear, Geometric and Log Equation(L to R) at iteration 50

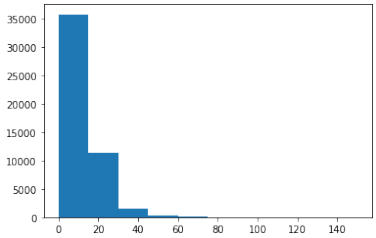

Figure 13: Histogram of norm of gradient to select good ϵ

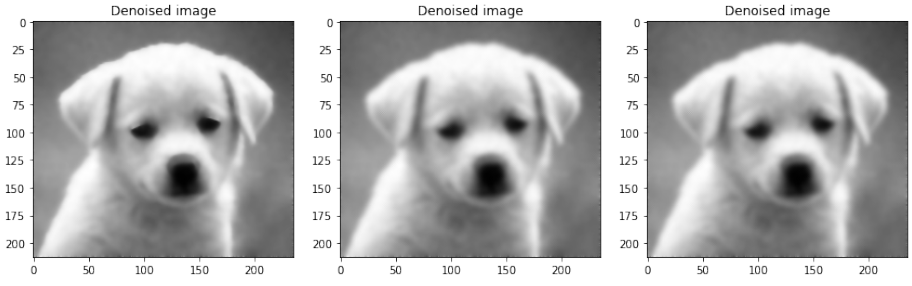

Figure 14: Effect on ϵ: from left $\epsilon = 20, 300, 1000$ for Geometric Heat Equation

tween the two methods. As seen from the figure below, we can see at each iteration, the MSE of linear heat equation is higher compared to geometric heat equation, showing that it performs better. Linear heat equation performs better than log-equation.

5 Conclusion

- Geometric heat equation performs better than Linear heat equation qualitatively and quantitatively
- Geometric heat equation penalizes smooth edges more and sharp edges lesser, accentuating them with each iteration, this can be seen in the way the eyes of the puppy become sharper and the outline of the puppy becomes more merged with the background with each iteration
- Linear heat equation diffuses along tangential and normal direction, giving a blurring effect.

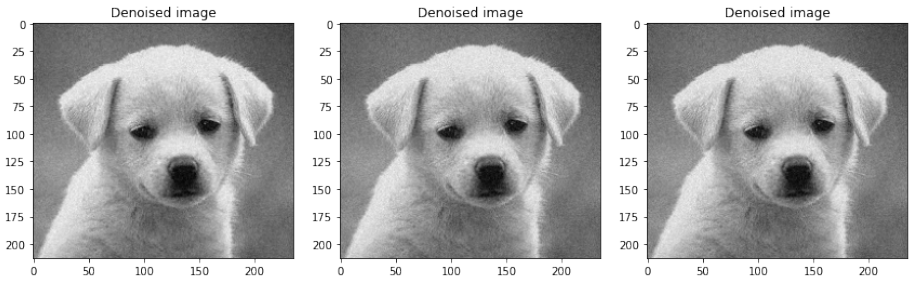

Figure 15: Effect on ϵ: from left $\epsilon = 20$, 300, 1000 for Log Equation

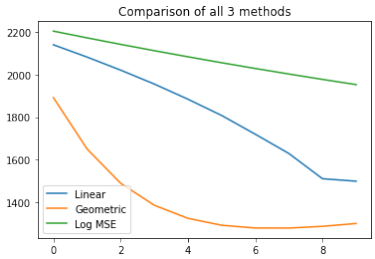

Figure 16: MSE across iterations