Assignment 2: Total Variation Optimization

Indian Institute of Technology Delhi
Computational Optical Imaging
PYL 759

Tamal Majumder

Entry No: 2023PHS7226

Assignment done under the supervision of
Prof. Kedar Khare

25 th August, 2024

1 Problem Statement

1. Take a grayscale image $g(x, y)$ as input.
2. **Outputs:**
   (a) **Numerical value of TV(g):** Compute the total variation (TV).
   (b) **Matrix** $h = \nabla g \cdot TV(g)$**:** Compute the gradient of the total variation with respect to the image.
3. **Iterative Optimization:**
   - Perform 25 recursive iterations of the following operation in a loop:

   $$g^{(n+1)} = g^{(n)} - 5 \times 10^{-3} \frac{\|g^{(n)}\|_2}{\|h^{(n)}\|_2} \cdot h^{(n)}$$

   - The step size is kept small, and no line-search is employed.
4. **Display and Comment:**
   - Display the resultant image after 25 iterations alongside the original one.
   - Comment on the observations.

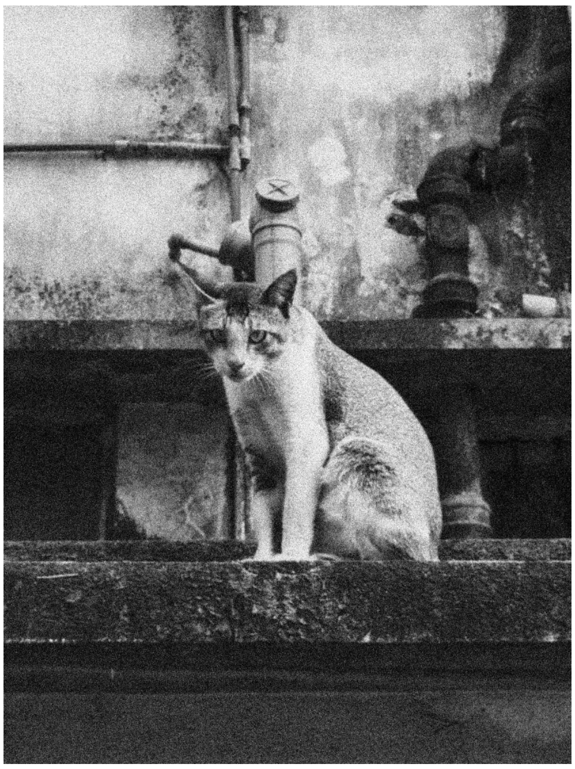


Figure 1: Input Image

2 Mathematical algorithm

The below mention Mathematical formulas were coded up and implemented step by step. There were basically three steps: The total Variation calculation step, the Gradient of TV, and The Weight update step.

1. **Total Variation (TV) Calculation:**

$$TV(g) = \sum_{x,y} \sqrt{\left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2}$$

2. **Gradient of TV (TV(g)):**

   - First, compute the gradients $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$.
   - Normalize the gradients:

$$h_x = \frac{\frac{\partial g}{\partial x}}{\sqrt{\left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2}}, \quad h_y = \frac{\frac{\partial g}{\partial y}}{\sqrt{\left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2}}$$

   - Calculate the gradient of TV:

$$h = \nabla \cdot (h_x, h_y)$$

3. **Weight Update Rule:**

$$g^{(n+1)} = g^{(n)} - \alpha \frac{\|g^{(n)}\|_2}{\|h^{(n)}\|_2} \cdot h^{(n)}$$

* Here the step size is $\alpha = 5 \times 10^{-3}$. (Given)
* Number of epochs = 25

Using this algorithm iteratively for 25 epochs and using learning rate/step size $\alpha = 0.005$ we were able to obtain the desired result.

3 Observed Results :

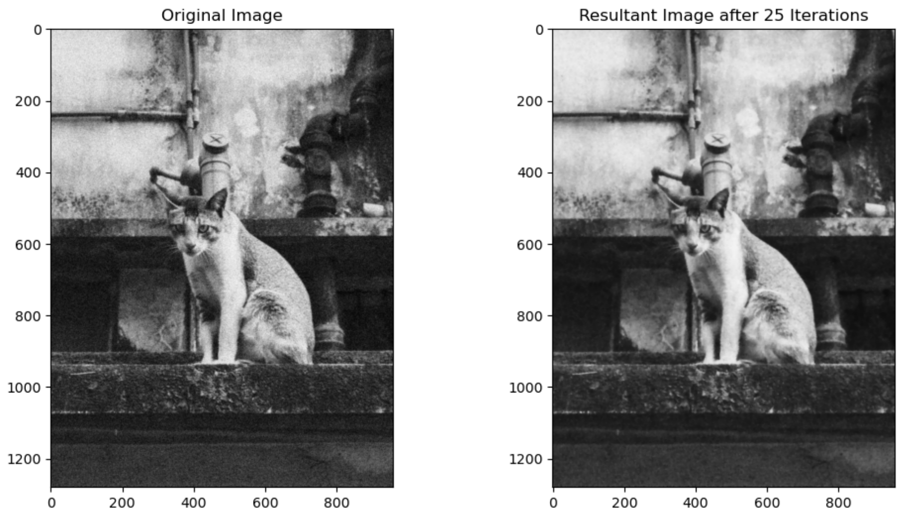


Figure 2: Resultant Image after 25 iterations with learning rate $\alpha = 0.005$

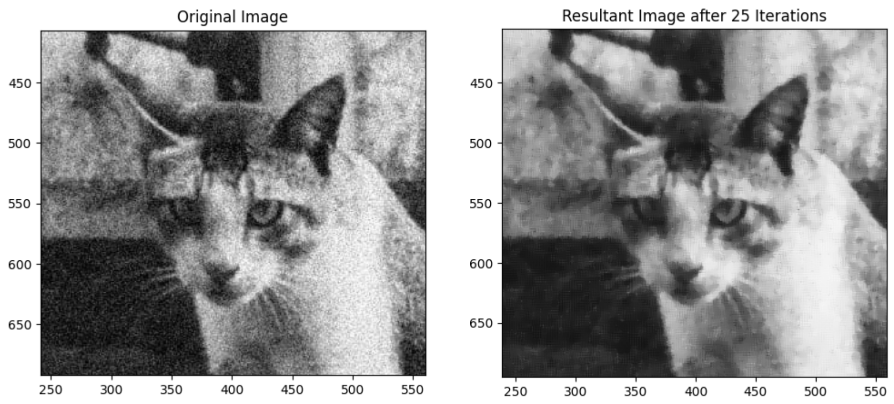


Figure 3: Zoomed Image for Clear View

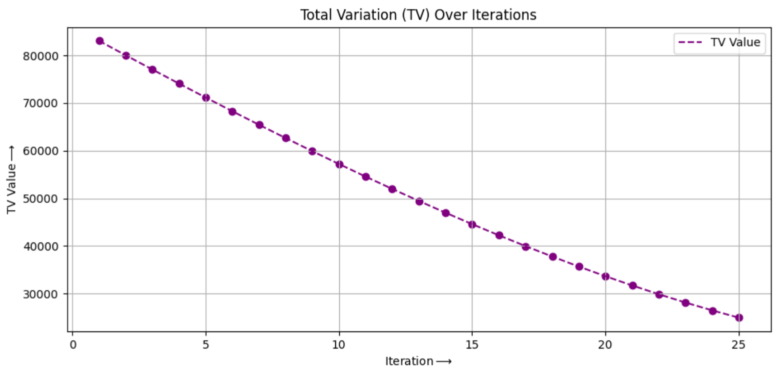


Figure 4: Total Variation in Each Epoch

Now after experimenting with the learning rate, we can observe the following results given below:

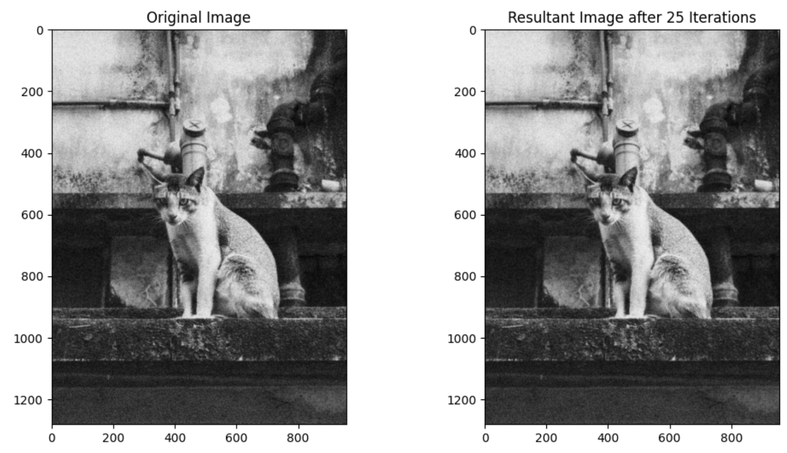


Figure 5: Result with learning rate $\alpha = 0.001$

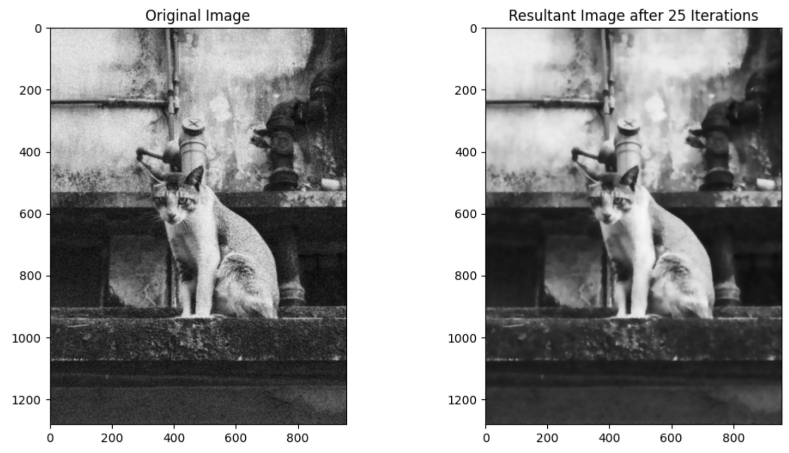


Figure 6: Result with learning rate $\alpha = 0.01$

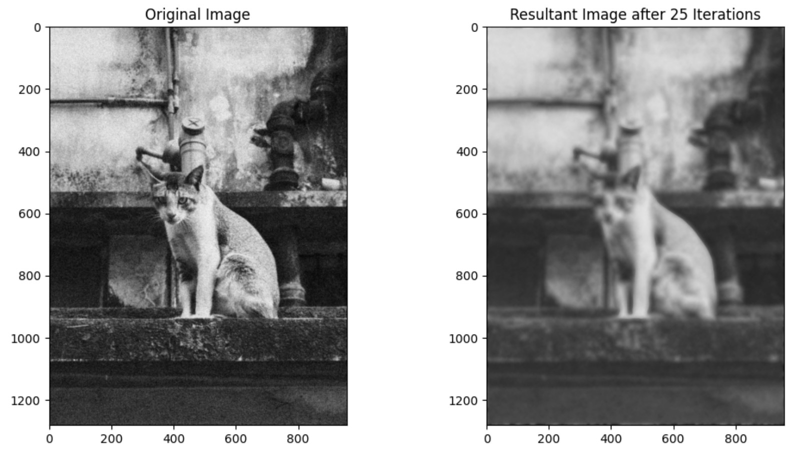


Figure 7: Result with learning rate $\alpha = 0.1$

The change in TV in epochs for different learning rates:

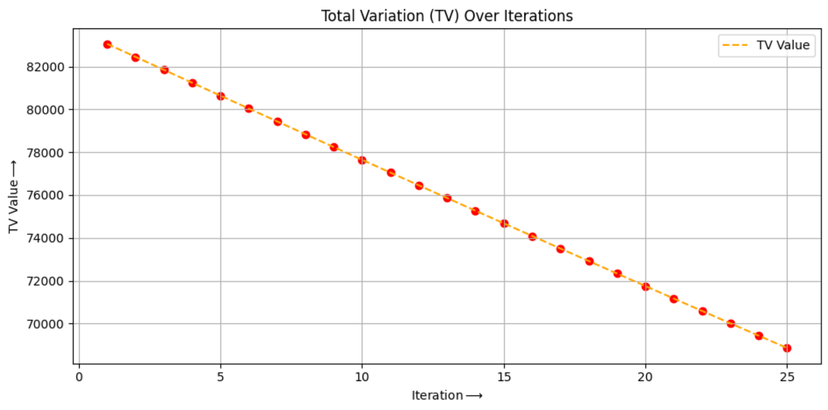


Figure 8: Change in TV with learning rate $\alpha = 0.001$

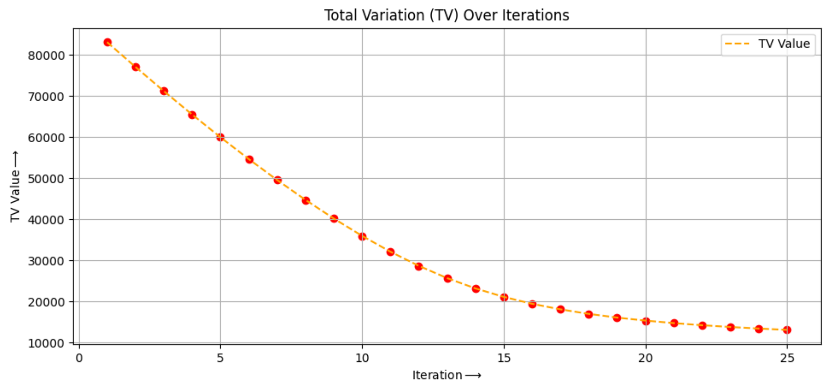


Figure 9: Change in TV with learning rate $\alpha = 0.01$

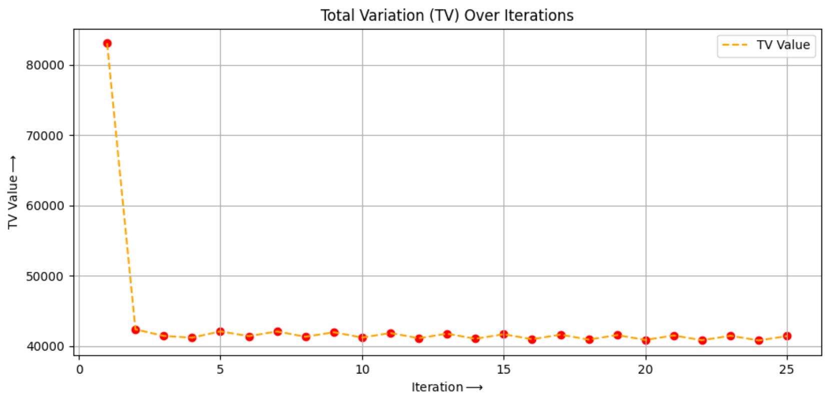


Figure 10: Change in TV with learning rate $\alpha = 0.1$

4 Discussion on Result

- **Optimization Process:**
  - 25 iterative updates were performed using the weight update rule.
  - The step size for the updates was $\alpha = 5 \times 10^{-3}$.
- **Resultant Images:**
  - After 25 iterations, the resultant image showed reduced noise and smoother features compared to the original Noisy image.
  - The zoomed view provided a clearer representation of the changes.
- **Effect of Learning Rates:**
  - Learning Rate $\alpha = 0.001$: In the image we are able to see minimal changes; the effect was subtle.
  - Learning Rate $\alpha = 0.01$: Noticeable improvement was observed in smoothing, with better feature preservation.
  - Learning Rate $\alpha = 0.1$: Significant smoothing observed but also lost a lot of details and we see oversmoothing in this case.
- **Effect of Number of epochs:**
  - Increasing the Number of epochs we can reach to an optimal minima. But it has to be chosen properly. One way to choose it is we keep running epochs until the change in TV becomes significantly lower or constant.

So we can conclude the whole analysis by saying, Smaller learning rates resulted in slower convergence and finer details.and Larger learning rates sped up convergence but risked oversmoothing, impacting image details of the image.

* Another observation was, if we change the sign of the weight update formula, instead of negative sign in between if we take positive sign then this becomes a Noising algorithm instead of Denoising algorithm.

$$g^{(n+1)} = g^{(n)} - \alpha \frac{\|g^{(n)}\|_2}{\|h^{(n)}\|_2} \cdot h^{(n)} \Longrightarrow DenoisingAlgorithm$$

$$g^{(n+1)} = g^{(n)} + \alpha \frac{\|g^{(n)}\|_2}{\|h^{(n)}\|_2} \cdot h^{(n)} \Longrightarrow NoisingAlgorithm$$

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from PIL import Image
```

Total Variation function

```
In [2]: def total_variation(g):
            grad_x, grad_y = np.gradient(g)
            TV = np.sum(np.sqrt(grad_x**2 + grad_y**2))
            return TV
```

Gradient of TV = => (∇TV(g))

```
In [3]: def grad_tv(g):
            grad_x, grad_y = np.gradient(g)
            norm_grad = np.sqrt(grad_x**2 + grad_y**2)
            norm_grad[norm_grad == 0] = 1  # To avoid division by zero
            h_x = grad_x / norm_grad
            h_y = grad_y / norm_grad
            h = np.gradient(h_x)[0] + np.gradient(h_y)[1]
            return h
```

Loading and preprocessing the image

```
In [4]: image_path = r'D:\1.Study stuff\SEM 3 @IITD\Optical Imaging\Assignment 2\Test.jpg'
        image = Image.open(image_path).convert('L')  # Converting to grayscale
        image = np.array(image) / 255.0  # Normalizing pixel values to [0, 1]

        # Making a copy for the image for processing on it
        g = image.copy()
```

Performing the optimization :

```
In [5]: TVs = []  # For storing total variation in each itterative loop
        for n in range(25): # Given no of itteration
            tv = total_variation(g)
            h = -grad_tv(g)
            step_size = 5e-3
            g = g - step_size * np.linalg.norm(g) * h / np.linalg.norm(h)
            TVs.append(tv)
```

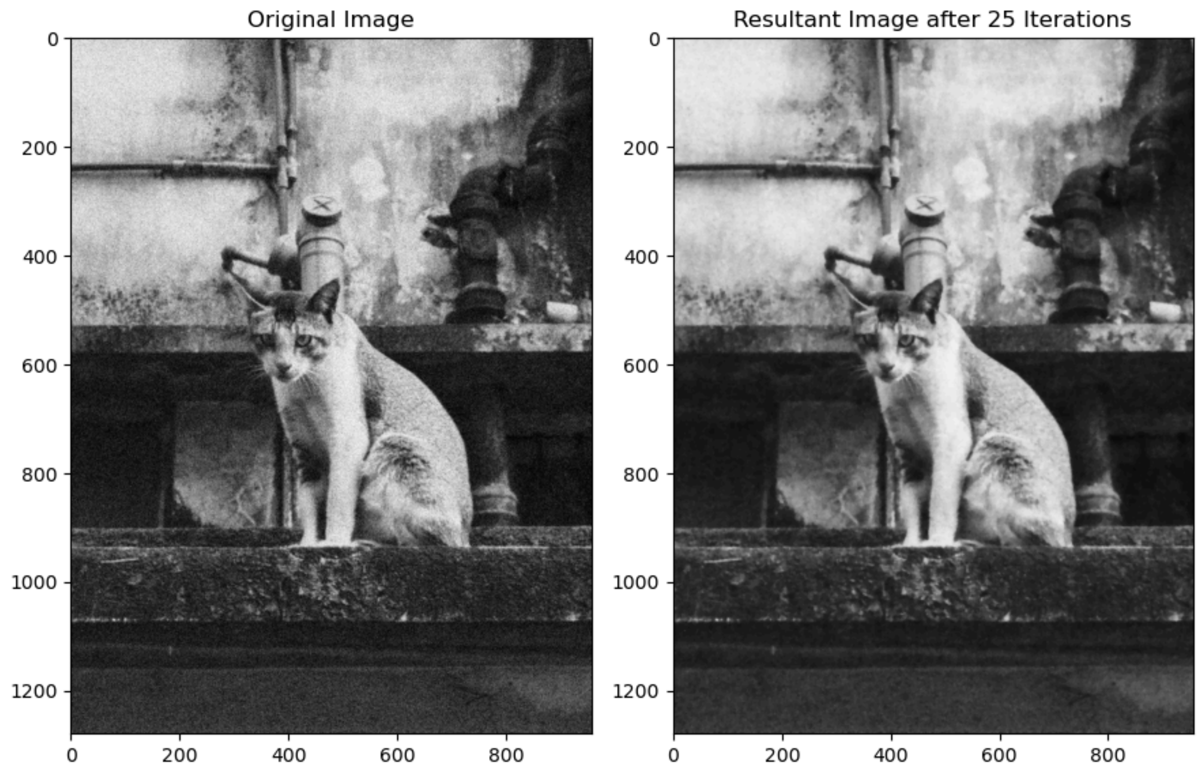
Printing the final TV and matrix h

```
In [6]: final_tv = total_variation(g)
        print(f" * Initial TV: {total_variation(image)}")
        print(f" * Final TV after 25 iterations: {final_tv}")

        # Printing the matrix h
        print("\nMatrix h = ∇TV(g):\n")
        print(grad_tv(g))
```

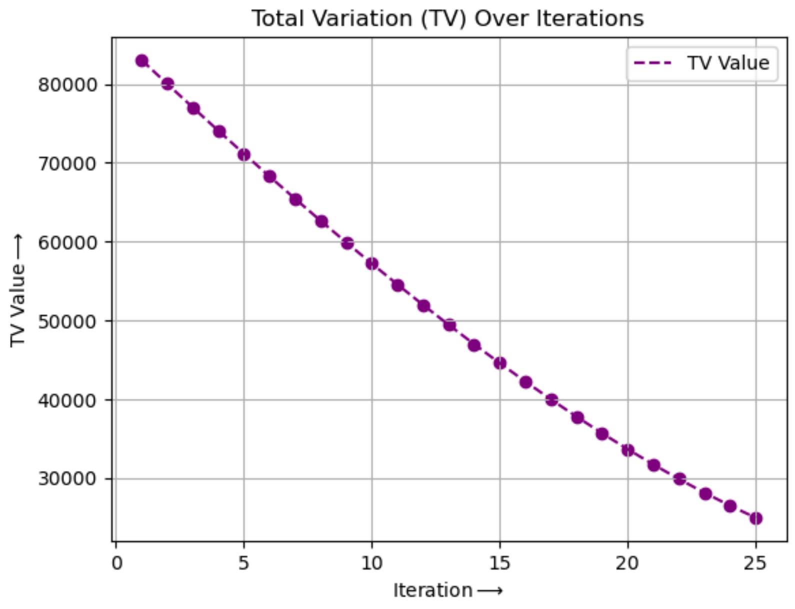
```
 * Initial TV: 83066.70843634507
 * Final TV after 25 iterations: 23536.249209381753

Matrix h = ∇TV(g):

[[ 0.37640345  0.05855352 -0.0676364  ... -0.59904383 -0.38597729
  -0.19871513]
 [ 0.63020582  0.54852162  0.49259555 ... -0.11293598  0.2409358
   0.92317496]
 [ 0.36500311  0.02932343 -0.69313673 ... -0.94282041 -0.63040425
   0.58786656]
 ...
 [-0.63662414 -0.00475128  0.24831612 ...  1.49655849  0.62316748
  -0.3700658 ]
 [ 0.22431272  0.24931516  0.22344397 ...  0.47478212 -0.09238439
   0.6253699 ]
 [ 0.0870555  -0.0263892   0.54403634 ... -0.57598897 -0.10288929
   0.5031889 ]]
```

Plotting the images

In [7]:
```
# Displaying the original image
plt.figure(figsize=(9, 6))
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

# Displaying the resultant image
plt.subplot(1, 2, 2)
plt.imshow(g, cmap='gray')
plt.title('Resultant Image after 25 Iterations')
plt.tight_layout()
plt.show()
```



In [8]:
```
# Plotting TV values
plt.plot(np.arange(1, len(TVs)+1), TVs, color='purple', linestyle='--', label='TV Value')
plt.scatter(np.arange(1, len(TVs)+1), TVs, color='purple')
plt.title('Total Variation (TV) Over Iterations')
plt.xlabel(r'Iteration$\longrightarrow$')
plt.ylabel(r'TV Value$\longrightarrow$')
plt.grid(True)
plt.legend()
plt.show()
```


Total Variation (TV) Over Iterations
TV Value
TV Value ⟶
Iteration ⟶