TOTAL VARIATION DENOISING

NATHAN S. WEATHERLY, RKW7RA

Contents

1. Introduction

1.1. The Denoising Problem.

This project set out to solve the problem of denoising noisy signals. There are many different types of noise that can be present within noisy signals but the most common is "white" noise. This form of noise is also known as additive Gaussian noise, where the original signal is distorted by adding a Gaussian random variable to each point. The process of denoising takes in a noisy signal and attempts to remove the noise and recover the original signal. The recovered signal will almost never be perfect, but the goal is to be able to reasonably distinguish between the noise and the underlying signal.

1.2. Total Variation.

One way in which we can attempt to measure the noise in a signal is through the Total Variation of a signal. In the continuous case with a continuous signal $y(t)$ that takes values on $[0, N]$, we define

$$TV(y) = \sup_{P \in \mathcal{P}} \sum_{i=0}^{n_P - 1} |y(t_{i+1}) - y(t_i)|.$$

Date: May 1, 2022.

Where $\mathcal{P}$ is the set of all partitions of $[0, N]$, n_P is the number of elements in the partition P and t_i is the ith element of the partition P. While the formula may seem complex, it essentially computes the amount of vertical movement of the signal $y(t)$ within the interval $[0, N]$. Furthermore, the sum is clearly maximized by continually adding points to the partition so in the discrete case we can reduce the formula to

$$TV(y) = \sum_{i=0}^{N-1} |y[n+1] - y[n]|.$$

Where $y[n]$ is a discrete time signal defined on $\{0, ..., N\}$. This formula follows from the fact that the partition over a finite discrete set with the maximum number of elements is exactly equal to the set itself. As in the continuous case, Total Variation represents the total vertical movement the signal has within the specified interval.

2. Total Variation Denoising

2.1. **Measuring Noise.**

One key observation that leads to the development of Total Variation Denoising is that the Total Variation of noisy signals is much higher than that of smooth and accurate signals. Additive Gaussian noise in particular introduces fluctuations at every time step that significantly increase the vertical movement of the noisy signal. This means that the Total Variation will be increased by adding noise to a signal and consequently it should be decreased by denoising a signal. The reason why Gaussian noise is so different from a standard signal is that successive values are completely independent random variables. In a standard signal, the value of that signal is highly correlated with the value of that signal at neighboring time steps. This means that the absolute difference between successive time steps should be relatively low and therefore the Total Variation should not be too high. In contrast, Gaussian noise will have fluctuations at almost every time step and so the Total Variation will be extremely high. Then when applying additive Gaussian noise to a signal the Total Variation should be significantly increased even if the relative amplitude of the noise is small in comparison to the entire signal. For example, consider the signal $y[n] = 10\sin(0.01n)$ and its noisy variant $\tilde{y}[n] = y[n] + e_n$ where e_i is a standard normal random variable for all i. Both signals are depicted in Figure 1 and using Python their calculated total variations were 65 for the original signal and 1086 for the noisy signal. The presence of noise in a signal and having a high Total Variation are seemingly correlated

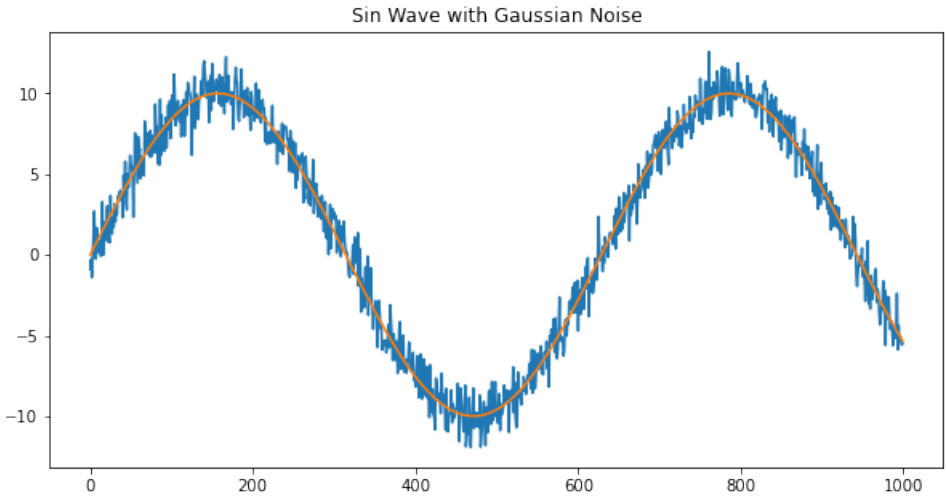

FIGURE 1. Sin Wave with Gaussian Noise

so if there were a method to minimize Total Variation it could help remove noise without destroying the underlying signal.

2.2. **Algorithm.** As was discussed in the previous section, reducing Total Variation could significantly help remove noise from a signal. However, any signal that is not constant will have some amount of natural Total Variation so we cannot only minimize Total Variation without any constraints. An ideal algorithm would keep the denoised signal as close to the noisy one as possible while at the same time significantly reducing the Total Variation of the noisy signal. To measure how close the two signals are we can use the squared L^2 norm of $y - x$ where y is the input signal and x is the output signal. Then we want to solve the minimization problem

$$\min_{x \in \mathbb{R}^{N+1}} \sum_{k=0}^{N} |y[k] - x[k]|^2 + \lambda \sum_{k=0}^{N-1} |x[k+1] - x[k]|.$$

Where λ is a parameter that determines how much Total Variation minimization is valued relative to the distance between the two signals. So a high λ would be useful for when noise is a large component of the signal and a lower λ would be useful for the opposite case. There are many ways of solving this minimization problem through iterative algorithms that use convex optimization but there also exists some direct approaches. The algorithm that was employed for this project was a direct algorithm that rephrases the minimization problem in terms of an equivalent problem known as the "dual" problem.

$$\min_{u\in\mathbb{R}^{N+1}} \sum_{k=1}^{N} |y[k] - u[k] + u[k-1]|^2$$

where $|u[k]| \leq \lambda$ and $u[0] = u[N] = 0$. Then given a solution to this problem we recover the Total Variation Denoising solution through the formula

$$x[n] = y[n] - u[k] + u[k-1].$$

Applying the Karush-Kuhn-Tucker conditions from nonlinear programming give us that for the optimal $u[n]$ and $x[n]$,

$$|u[n]| \leq \lambda \Leftrightarrow x[n] = x[n+1]$$

$$u[n] = -\lambda \Leftrightarrow x[n] < x[n+1]$$

$$u[n] = \lambda \Leftrightarrow x[n] > x[n+1].$$

Using this information, an algorithm can be formulated as follows: Start at some position k in the signal which is part of a constant line segment beginning at k_0 such that $x[k_0] = ... = x[k]$. We do not know the value of this segment but we know it is bounded by v_{min} and v_{max}. We can then calculate corresponding u_{min} and u_{max} using the previous relations. Attempt to extend the current value of the signal such that $x[k] = x[k+1]$. In the process of doing this, attempt to update the values of u_{min}, u_{max}, v_{min}, and v_max according to the previous relations. Then there are three cases. Either all of the updated values are valid or one of u_{min} and u_{max} is out of bounds. If u_{min} is too low then we must have a negative jump at the last point when $u_{min} = \lambda$. If u_{max} is too high then we must have a positive jump at the last point when $u_{max} = -\lambda$. In all three cases, we iterate the same procedure again either at the point after the jump or on the next point within the signal. If at any point the end of the signal is reached within an iteration, the boundary conditions of $u[N] = 0$ must be checked and if not satisfied there must be a jump as in the previous case. Once the boundary condition is satisfied, the optimal value of $x[n]$ has been constructed.

This algorithm is $O(n^2)$ because in the worst case, it traverses the entire signal on each iteration. The algorithm was implemented within a Python function that takes λ as a parameter and applied to various signals.

3. Results

To test the algorithm it was applied to various types of randomly generated piecewise constant signals. The signals used were progressively more difficult to denoise in order to test the capacity of the algorithm to differentiate between the noise and the underlying signal. First, the algorithm was applied to signals with random length intervals of value 1, 0, or -1. The algorithm was extremely accurate when applied to such signals and the output was almost always very close to the real signal. An example of such an application is in Figure 2.

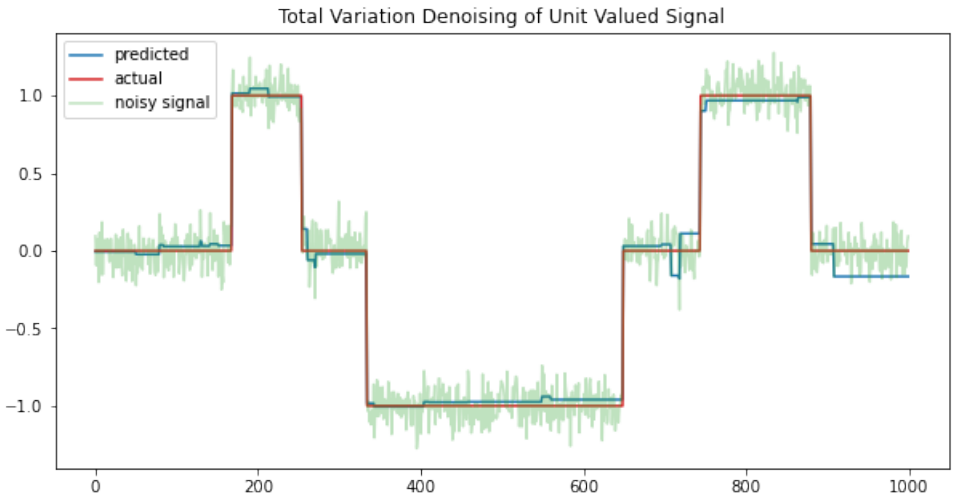

Figure 2. Unit Signal Denoising

The next step up from a unit-valued signal was a random signal that could take finitely many discrete values. The algorithm continued performing very well but as expected it was less accurate when the scale of the noise became similar to the scale of the random jumps. An example of such a case can be found in Figure 3.

The most difficult type of signal that the algorithm was applied to was a Levy process that was piecewise constant with a specified jump probability and a Gaussian random variable for jump distance. Remarkably the algorithm continued performing very well, although it became much more inconsistent between samples, most likely due to the wide variance in the difficulty of this type of signal. An example of the algorithm applied to a Levy process with difficult parameters can be found in Figure 4.

FIGURE 3. Discrete Signal Denoising

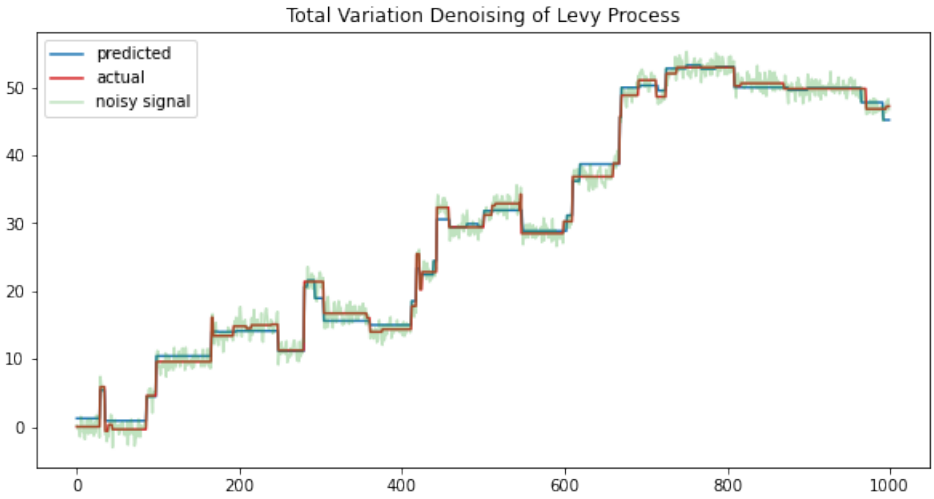

FIGURE 4. Levy Process Denoising

4. CONCLUSIONS

The most prominent conclusion that I formed about the Total Variation Denoising process is that there are two very important parts that ensure its success. First, applying the algorithm to the correct type of signal is absolutely necessary. Because the optimal solution to the Total Variation Denoising Problem is always a piecewise constant signal, attempting to recover a smooth function like a Sin wave will always result in a distorted signal. Other algorithms like smoothing filters are

likely to be much more efficient in those cases. However, in the case of signals with sharp edges, Total Variation Denoising will effectively capture edges at precise intervals without any smoothing which can make it very valuable in applications like bioinformatics. The other key part in ensuring that the algorithm performs as expected is choosing the λ parameter correctly. If the λ parameter is too high, the underlying signal will lose all of its structure but if the λ parameter is too low, the noise will permeate the result of the algorithm. However, even with the optimal λ parameter, the structure of the underlying signal is still the most important factor. An example of the distortion that happens to the Sin wave from Figure 1 is displayed in Figure 5.

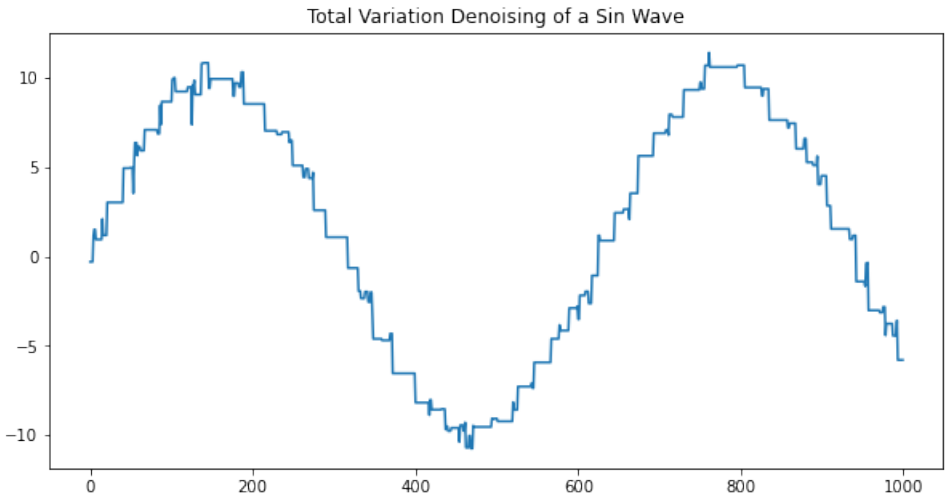

FIGURE 5. Sin Wave Denoising

With respect to challenges that occurred throughout this project, the main difficulty was that coding the primary algorithm in Python. The algorithm is very complex and the required code is very repetitive in nature so small bugs are hard to spot, especially without intimate knowledge of the mechanisms behind the algorithm.

There was only one attempted application of the algorithm that did not make it into this project. I knew that the discrete derivative (backward difference) of Brownian motion is Gaussian noise. Using this knowledge, I had the idea to take a piecewise linear signal with Brownian drift and attempt to remove this drift by taking the discrete derivative, applying the Total Variation Denoising to remove the Gaussian noise, and taking the cumulative sum to arrive back at the original signal with less drift. While in theory, this approach could

work, I found that it was extremely inconsistent and only worked effectively when the Brownian drift was extremely small relative to the signal. This is because when taking the discrete derivative, the magnitude of the noise relative to the magnitude of the underlying signal is significantly amplified. Therefore if the algorithm is to be applied successfully, the Brownian drift must be very small. I have left the results of this attempted application in the Python code for this project under the testing section as it was part of my work.

References

[1] Condat, Laurent. *A Direct Algorithm for 1D Total Variation Denoising.*, IEEE Signal Processing Letters, Institute of Electrical and Electronics Engineers, 2013

[2] Davies, P.L., Kovac, A. *Local Extremes, Runs, Strings and Multiresolution*, The Annals of Statistics, 2001

[3] Powell, MJD. *A 'taut string algorithm' for straightening a piecewise linear path in two dimensions*, IMA Journal of Numerical Analyis, 1998

[4] Rudin, L. I.; Osher, S.; Fatemi, E. *Nonlinear total variation based noise removal algorithms*, Physica D., 1992

[5] Selesnick, Ivan. *Total Variation Denoising*, New York University, 2012