Digital Signal Processing

Nabila Adawy
n.roshdy@innopolis.university
B19-RO-01

HA1_Report

Task 1: Noise Reduction

For the signal $y_\epsilon(t) = y(t) + \epsilon$, defined in $T = [0, 1]$, where $y(t) = sin(2\pi 50t) + sin(2\pi 120t)$:

- I used $np.random.normal(0.5, 1.8, 10000)$ to generate random noise ϵ (samples) from a normal (Gaussian) distribution. I chose the mean to be 0.5. And tried different numbers for the standard deviation until I reached a result similar to the one in the task description, which was the value 1.8. Then, I choose 10000 as a number of samples and a number of discretized time samples n. so I reached the following results.

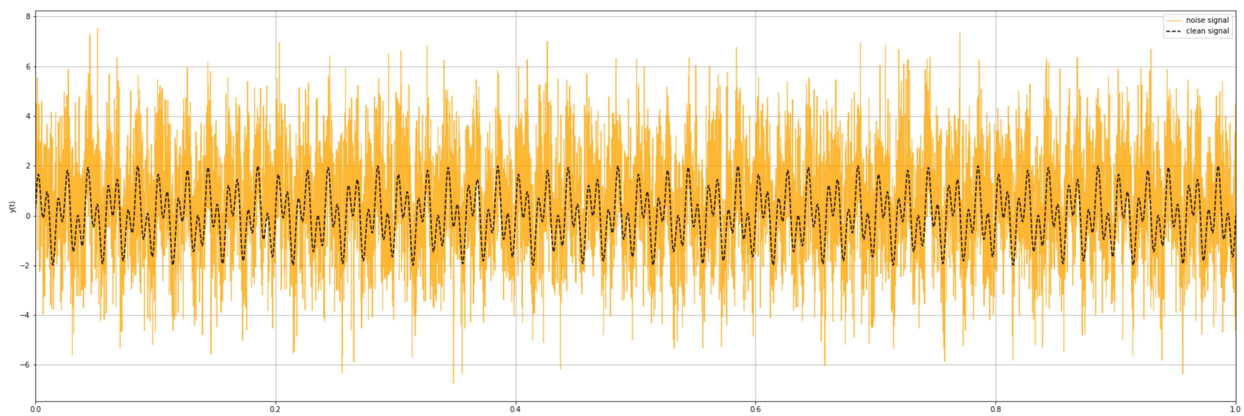

Figure 1: Clean and Noisy Signals

- I used $np.fft.fft$ to compute the Fast Fourier Transform of the discretized function $y_\epsilon(t)$ which equal to z then computed the $PDS = \frac{z\,\overline{z}}{n}$ and visualized it using $matplotlib.pyplot$ It looks as follows.

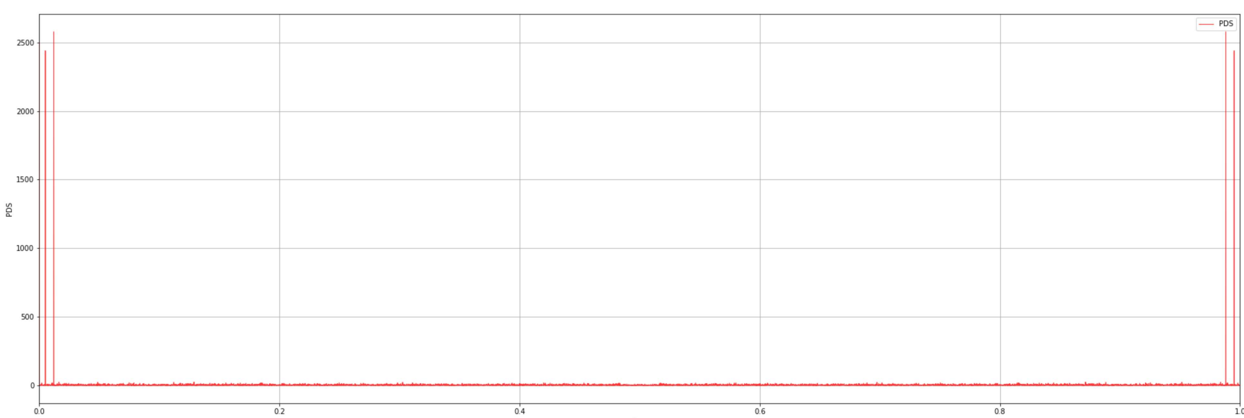

Figure 2: Power Density Spectrum

- From observations, the noise is in range where $PDS < 35$, so we can put the value of the threshold $\tau = 35$.

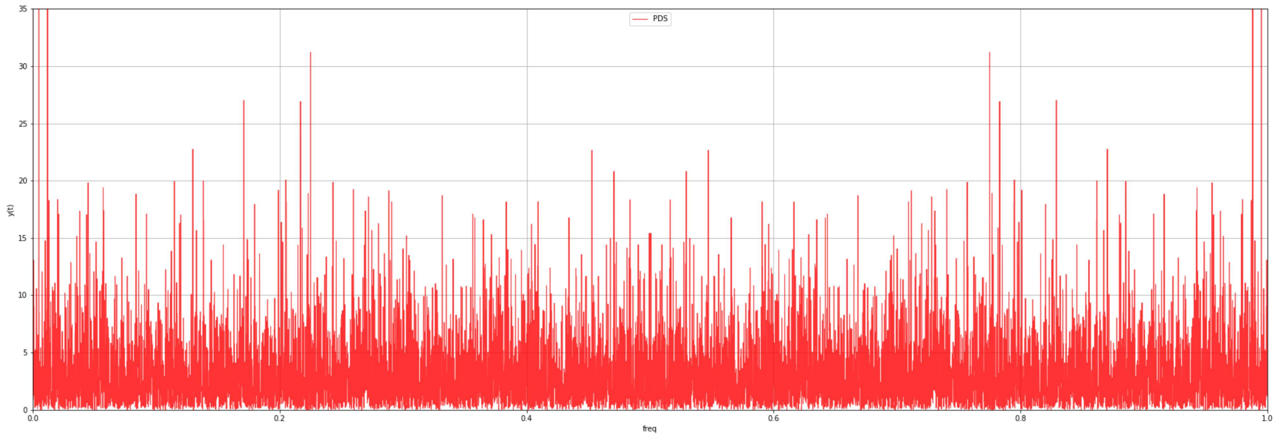

Figure 3: PDS closer view

- Keeping the elements that are larger than the threshold τ (filtering the PDS data from noise), and the corresponding indices, and visualizing the filtered PDS and the original one(with noise). we can obtain the following plot.

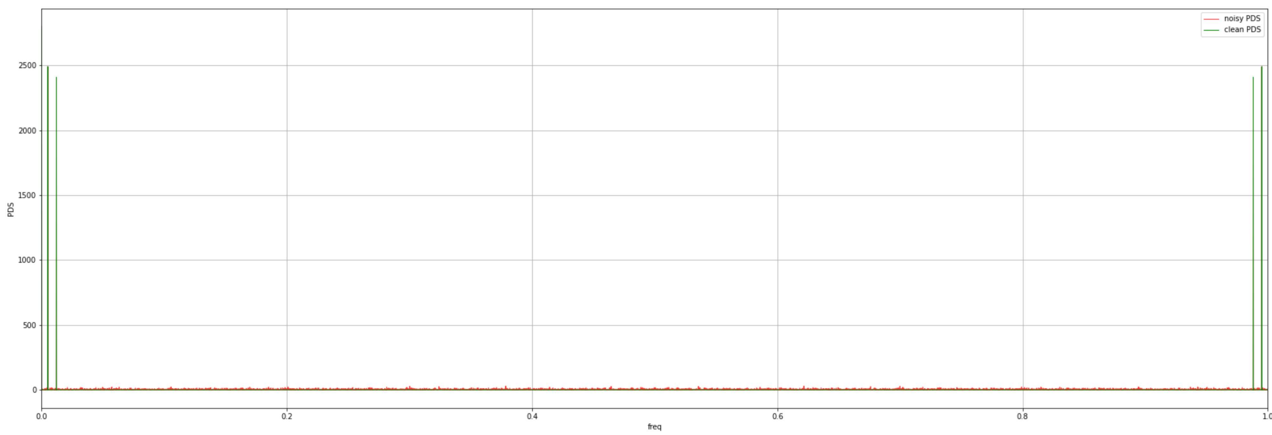

Figure 4: Noisy and clean PDS after filtering

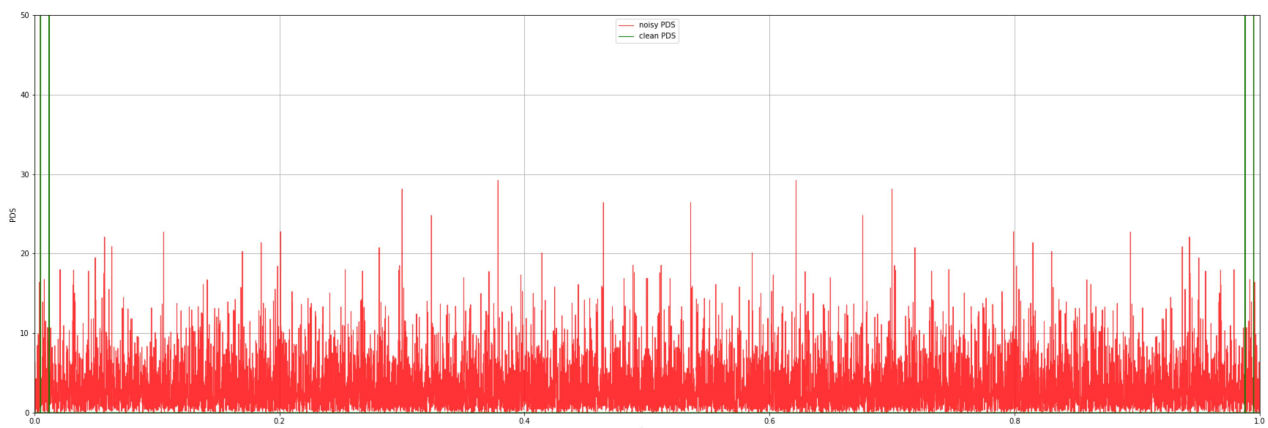

Figure 5: Closer view of the previous graph

- Filtering the discretized function $\widehat{y}_{\epsilon}(\omega)$ using the indices from the above step and computing the Inverse Fast Fourier Transform, I obtained the reconstructed function $y(t)$ as follows.

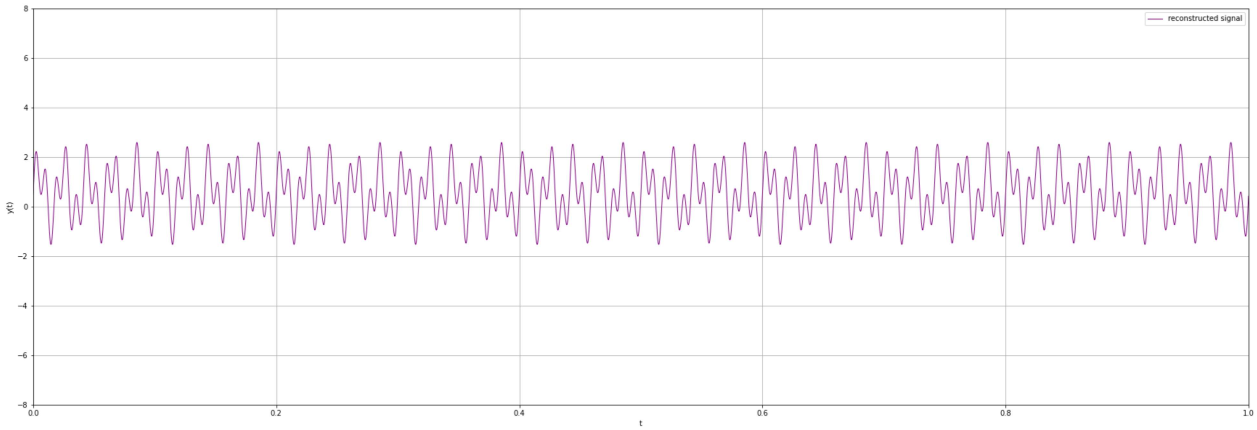

Figure 6: Noise Reduction, Reconstructed Function $y(t)$

*Note: all images have better resolution in the attached file.

Task 2: Image Compression

- For importing the image I used $matplotlib.image.imread$ then $matplotlib.pyplot.imshow$ to display the image. For testing, I used the following photo.

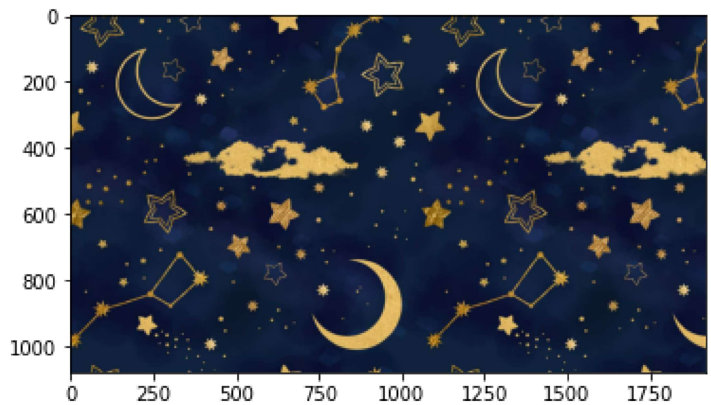

Figure 7: Original Testing Image

- Using the formula $Y' = 0.299\ R + 0.587\ G + 0.114\ B$ from $matplotlib$ I made a function that multiply each of RGB arrays of the image by those numbers then adding them to convert the RGB image to a gray scale format. Which means converting the image from $3D$ array into $2D$ array.

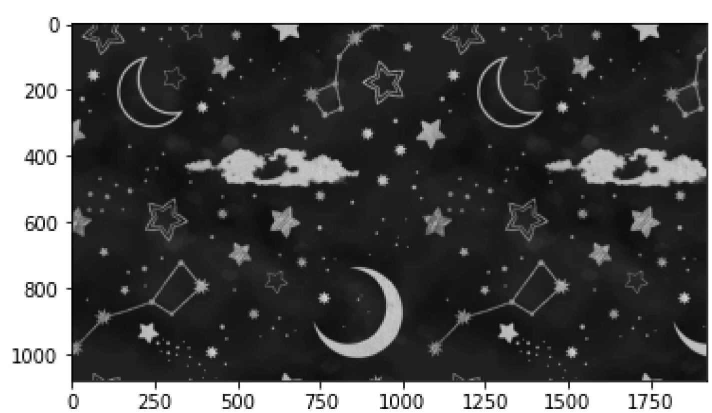

Figure 8: Image in gray scale format

- I used $fft2$ to compute the Fast Fourier Transform of the $2D$ array (gray image) to get its spectrum, then $fftshift$ to shift the the zero-frequency to the center of that spectrum.
- Computing the magnitude, adding 1 and computing the natural logarithm of the spectrum. $log\left(\mid F^2(A) \mid +1 \right)_{w \times h}$ we can obtain the following results.

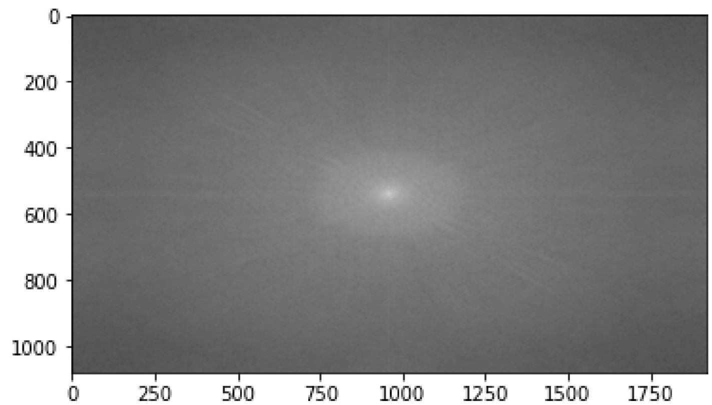

Figure 9: Image Spectrum

- I used $np.reshape$ to reduce the $2D$ array into $1D$ with $legnth = w*h$ then $np.abs$ to compute the magnitude and $np.sort$ for sorting the array $a = sort\left(\mid F^2(A) \mid \right)_{wh \times 1}$.

- Choosing $\tau = 0.1 \ll 1$, then computing $\lfloor (1 - \tau)wh \rfloor = b$, and defining the threshold as $threshold = a[b]$. From observations, the value $\tau = 0.1$ gave the best results for the reconstructed image (closer to the original one), decreasing τ more gives blur image.

- Now we can filter the spectrum of the shifted and reshaped array $\mid F^2(A) \mid_{wh \times 1} = A_f$ by keeping the elements that are bigger than the $threshold$ and saving the corresponding indices. we need to reshape the $1D$ array into $2D$ array to visualize the filtered spectrum.

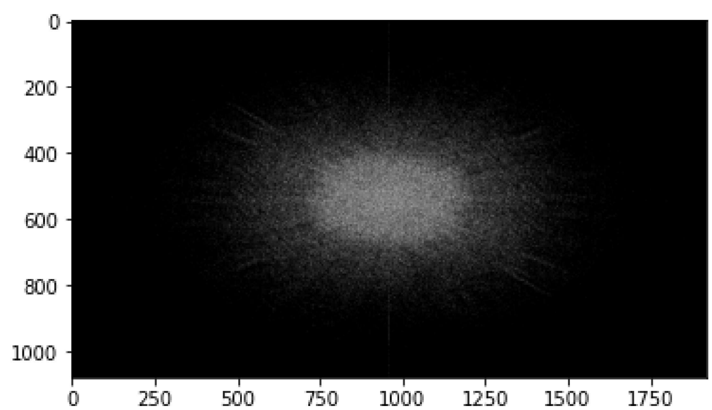

Figure 10: Filtered Spectrum

- Filtering the initial matrix $F^2(A)_{w \times h}$ using the indices from the above step. Then using $ifftshift$ and $ifft2$ to apply the shifting and Inverse Fast Fourier Transform to get the reconstructed image as follows.

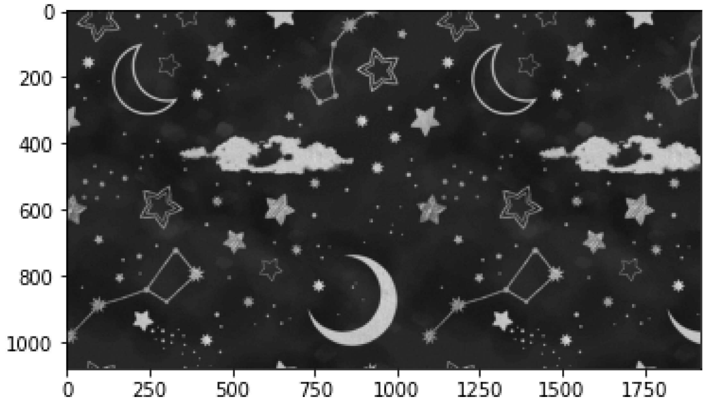

Figure 11: Reconstruced image

- Here is another testing Image.