Digital Signal Processing: Assignment 1

Ehsan Shaghaei 09.12.1999

April 10, 2022

1 Noise Reduction

In this task, we were supposed to simulated a function with random noise and reduce the noise in 5 steps as follows:

1.1 Simulate and plot the $y(t)$ and $y_\epsilon(t)$

In this Step, firstly I define the function $y(t)$ as described in the prompt [1]. Then the function *noise_channel* is defined for generating noise which basically produces a random integer in range $[-5, 5]$; This range were chosen randomly and it was not a bias choice. I have evaluated the model with other ranges such as $[-7, 7]$ or $[-3, 3]$ which does not affect the performance of the system. Afterwards,$y_\epsilon(t)$. were defined which is basically sum of the *noise_channel* and $y(t)$. Finally I plot the function $y(t), y_\epsilon(t)$ in time range of $[0, T]$ $S.T$ $T = 1$ with the sample rate of $N = 2^{10}$.

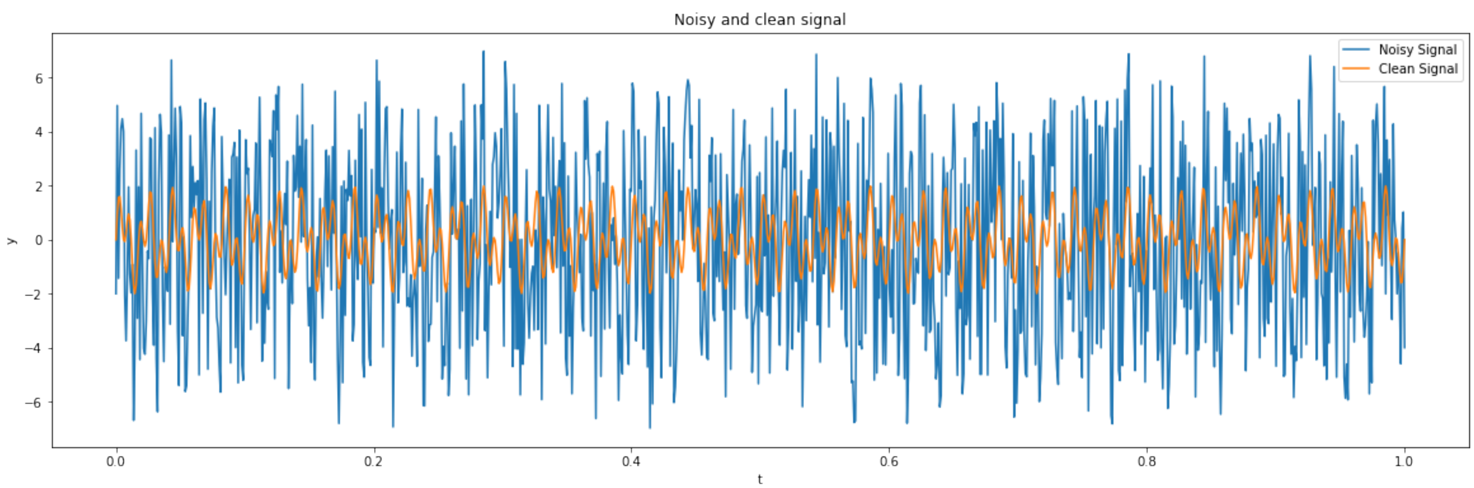

Figure 1: clean signal vs noisy signal

1.2 FFT

In this step a signal were generated from the noisy function in the interval $[0, 1]$ with sample rate $= 2^{10}$ which is a power of 2. A naive implementation of FFT were demonstrated and were used in order to obtain the Fourier transform of the signal as figure 2.

[1] Assignment Prompt

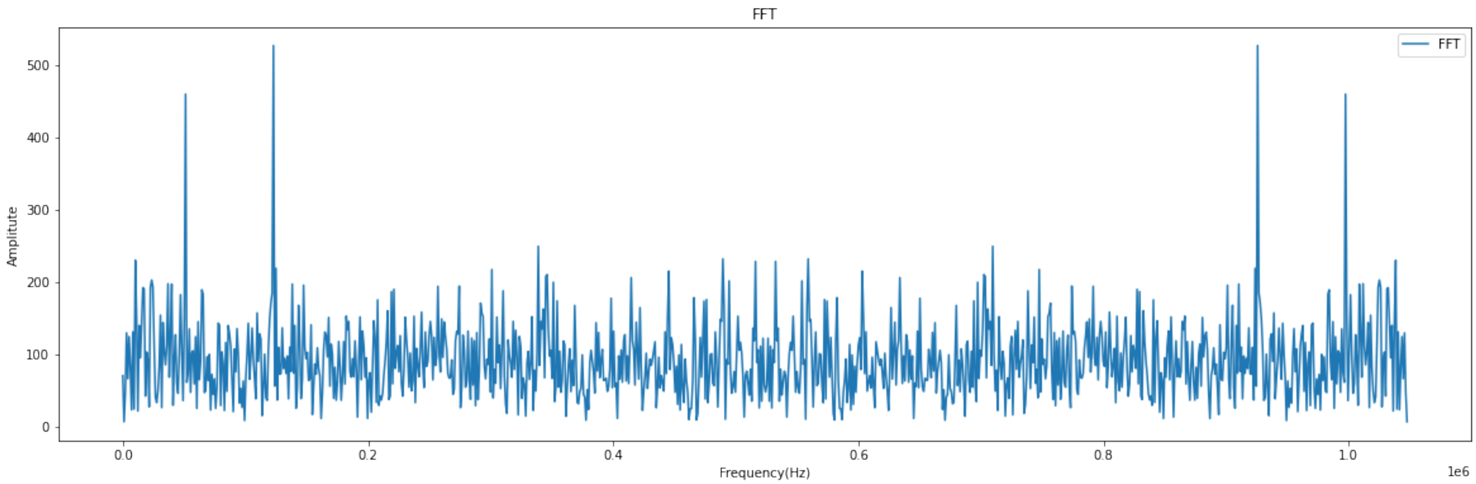

Figure 2: Noisy Signal FFT

1.3 Power Density Spectrum(PSD)

In this step, PSD of the noisy-signal were calculated and plotted as figure 3 using the function $FFT2PSD(F)$ which is implemented as following:

$$FFT2PSD(F) = \frac{F\hat{F}}{n(F)}$$

S.T n(F) is the number of the elements in the vector F

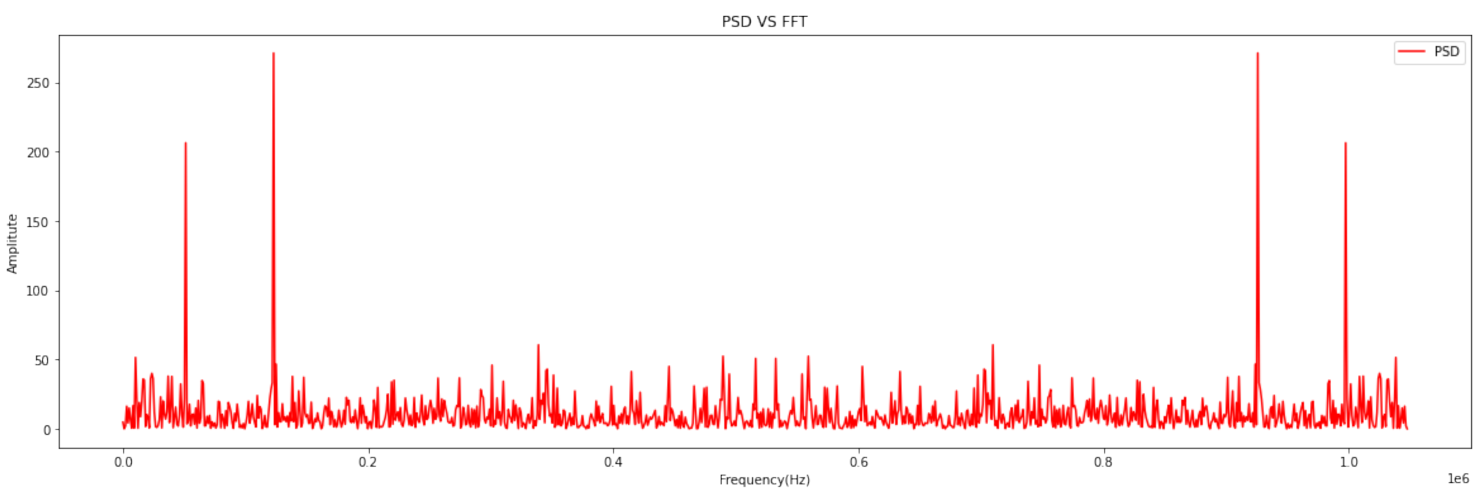

Figure 3: Noisy Signal PSD

1.4 Noise Reduction and Filtering

Let us define term "Bag ratio x of vector V" as the least $x \times 100$ percent of an accending sorted vector V.
In this stage, the threshold τ were chose by sorting the PSD vector and averaging over 4% of maximum amplitudes (bag ratio=0.04) and all the frequencies less than the Treshold were eliminated using the $filter$ funtion as following :

```
def filter(signal,psd,bag_ratio):
    sorted_signal = np.sort(signal)
    n_bag = int(len(signal)*bag_ratio)
    tau = abs(np.sum(sorted_signal[-1:-1-n_bag:-1])/n_bag)

    filtered_signal = []
    for i,sample in enumerate(psd):
        filtered_signal.append(0 if abs(sample)<tau else signal[i])
    return np.array(filtered_signal),tau
```

Listing 1: τ selection and filtering

The result of the filtering process over the fft of the noisy signal was as figure 4

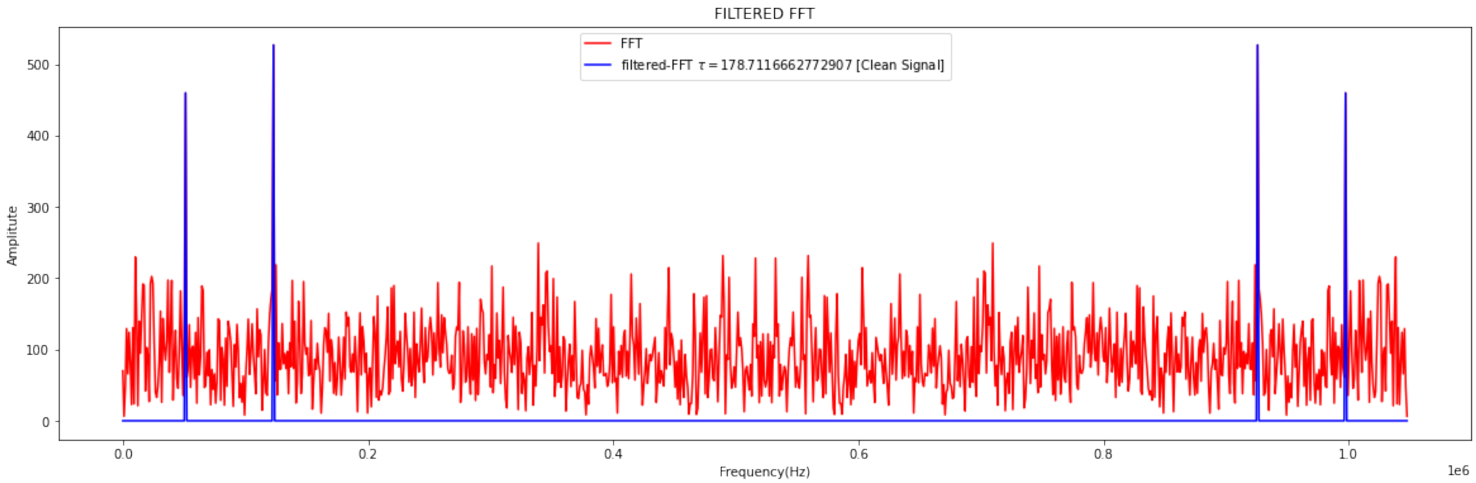

Figure 4: Noisy FFT vs Filtered FFT

1.5 Recovering the signal

In this step, the filtered FFT signal were recovered by Inverse Fourier Transform and were compared to the generated clean signal sample form $y(t)$. The results were as figure 5.

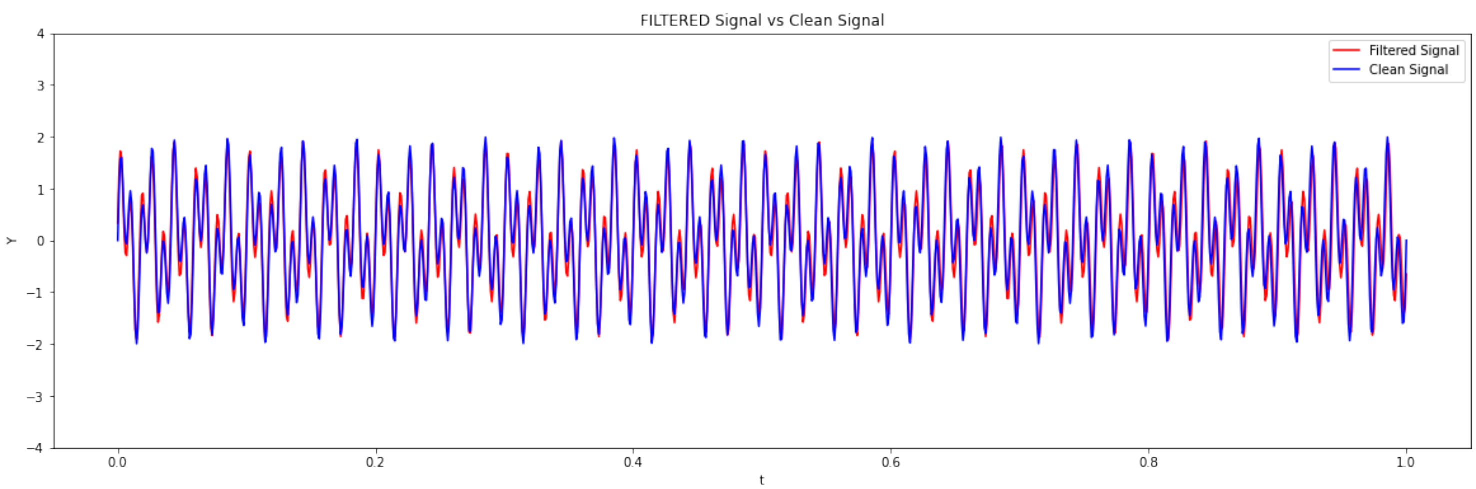

Figure 5: Filtered Signal vs Clean Signal

2 Image Compression

2.1 Pre-Processing

Initially, the image is loaded from local path, converted from RGB to Gray scale 2-D matrix.

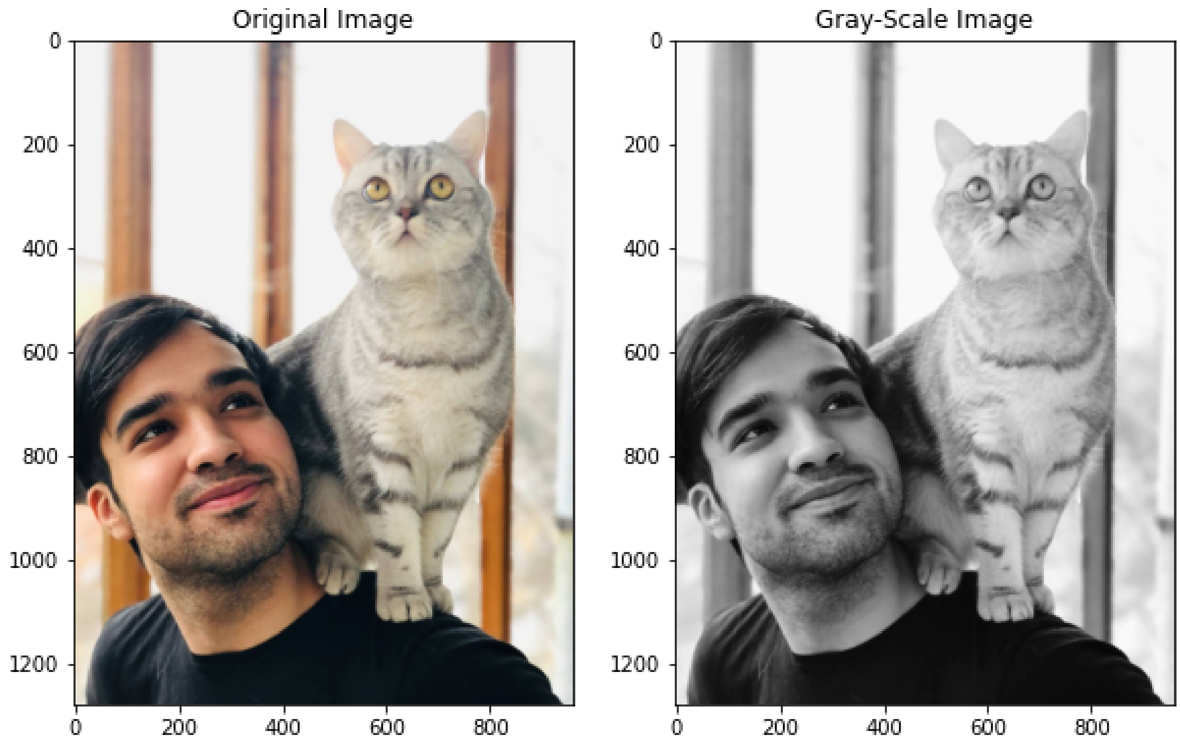

Figure 6: Orginal Image VS Grayscale Image

2.2 FFT-spectrum and Zero-Frequency Shift

In this step, first the FFT spectrum of the 2-D matrix signal plane were obtained and then the zero-frequency component were shifted to the center of the spectrum. using **numpy.fft** routines.

2.3 Spectrum magnitude

In this stage, The "modulus matrix" were generated by obtaining the amplitude of every element in the FFT matrix simply by **python** built-in *abs*() function which is supported by **numpy**.

2.4 Natural Logarithm of the signal plane matrix and spectrum plot demonstration

In this step, we shift the signal magnitude matrix by 1 unit, since the minimum possible value for magnitude is 0 and $log(0) \rightarrow UNDEFINED$. Now we generate the natural logarithm of the *shifted magnitude signal matrix*. Ploting the spectrum in figure 7

2.5 Vectorization

In this step the current signal plane matrix with dimensions 1280×960 is converted into a flat vector of length 1228800. Note that $1280 \times 960 = 1228800$.

2.6 Vector Magnitude

In this step, similar to step 2.3 we calculate the magnitude of vector elements.

2.7 Vector Sort

In this step, we sort the obtained vector from the previous step in ascending order using **numpy** built-in sort routine.

2.8 Picking the compression factor τ

In this step, the $\tau = 0.01$ were chose, which means we will keep $\tau \times 1280 \times 960 = 12288$ for image reconstruction.

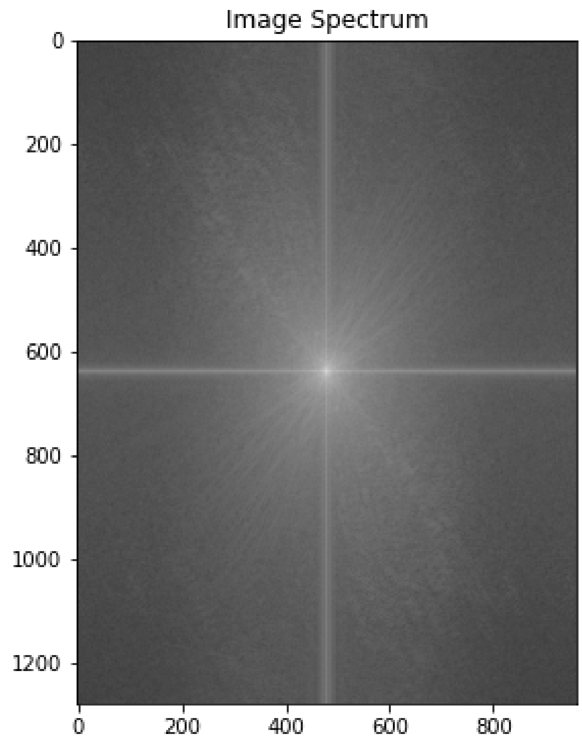

Figure 7: Natural logarithm shifted magnitude of FFT Spectrum

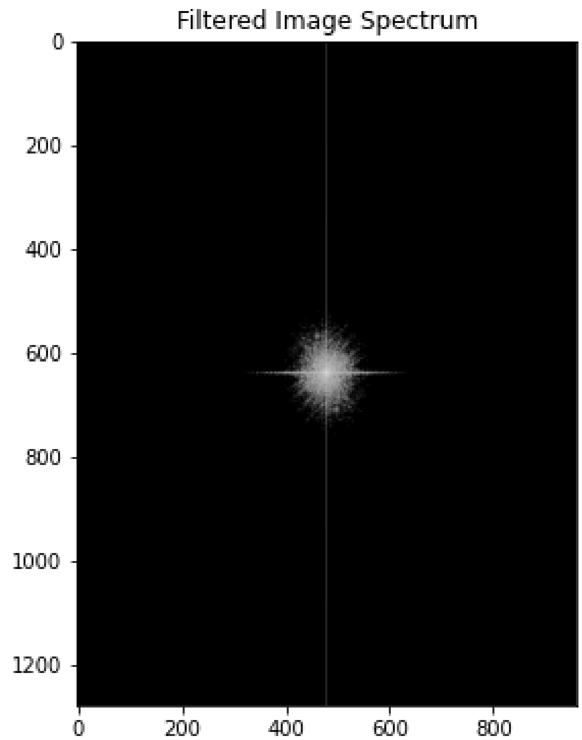

Figure 8: Compressed Spectrum

2.9 Threshold definition

We choose the threshold using the sorted vector in section 2.7 as following:

```
tau = 0.01
wh=len(sorted_spec)
b = np.floor((1-tau)*wh).astype(np.int32)
```

Listing 2: Threshold selection

2.10 Zeroing spectrum values below the threshold

All the natural logarithm shifted magnitude of FFT Spectrum7 values which were less than the threshold obtained in previous step were set to zero as in figure 8 and get all the indices $\mathcal{I}$ of the corresponding values above the threshold in the signal plane matrix.

2.11 Recovering the original image from the compressed data

In this step, we recover the image 9 from the compressed-data. The compressed data is the selected indices $\mathcal{I}$ of the *shifted-fft* which were obtained in previous section and section 2.2 respectively. The recovered image is inverse FFT of inverse Zero-Frequency Shift of natural logarithm of the compressed data matrix plane plus 1 as following:

$$RevoeredImage = IFFT(SHIFT - IFFT(log(CompressedData) + 1)$$

$$CompressedData = ShiftedFFT[\mathcal{I}]$$

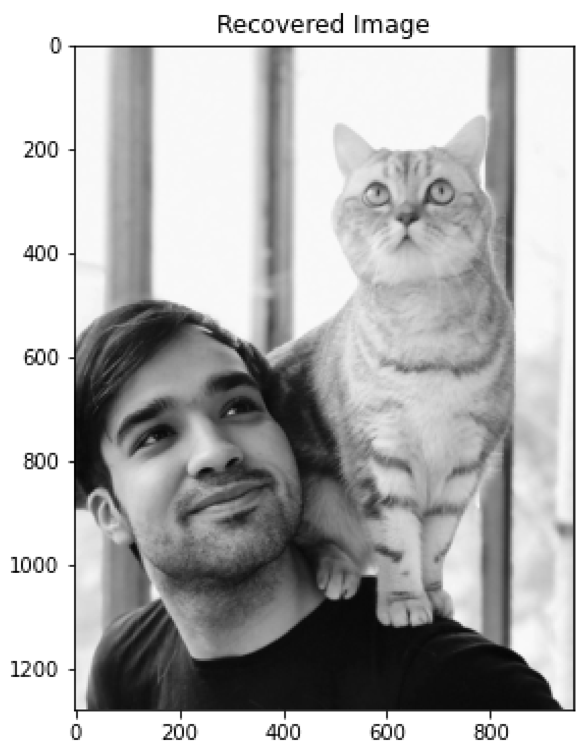

Figure 9: Recovered Image

3 Acknowledgement

I would like to take the opportunity to thank Imre Delgado for his efficiency in using the lab time, and Dr.Shilov for his phenomenal lectures.