Assignment 4

Shubham Sharma, 2021099

1) Amplitude from the raw CSI data.

[13]

```python
# code to find the amplitude and phase of every subscarrier
def amp_phase(df):
    from math import sqrt, atan2
    amp = []

    d = np.array(df)
    for j in range(len(d)):
        imaginary = []
        real = []
        amplitudes = []

        for i in range(len(d[j])):
            if i % 2 == 0:
                imaginary.append(d[j][i])
            else:
                real.append(d[j][i])
        for i in range(int(len(d[0]) / 2)):
            #write code here.
            amplitudes.append(sqrt(pow(float(real[i]), 2) + pow(float(imaginary[i]), 2)))

        amp.append(amplitudes)

    amp = pd.DataFrame(amp)
    amp = amp.reset_index(drop=True)

    print("amp_phase completed")

    return amp

```

[14]

```python
# Call Here
amp = amp_phase(matrix)
```

```
amp_phase completed
```

2) Optimal Value for Denoising filter

a) For Mayank_ISA

Window = **length / 70**

```
# outlier removal in amplitude
def hample_filter(input_matrix):
    input_matrix= np.asarray(input_matrix)
    n = input_matrix.shape[1]
    print(n)
    new_matrix = np.zeros_like(input_matrix)
    k = 1.4826  # scale factor for Gaussian distribution
    n_sigmas=1
    length = len(input_matrix)
    length = int(length/70)
    window_size=length # change the value here
    for ti in range(n):

        start_time = max(0, ti - window_size)
        end_time = min(n, ti + window_size)
        x0 = np.nanmedian(input_matrix[:, start_time:end_time], axis=1, keepdims=True)
        s0 = k * np.nanmedian(np.abs(input_matrix[:, start_time:end_time] - x0), axis=1)
        mask = (np.abs(input_matrix[:, ti] - x0[:, 0]) > n_sigmas * s0)
        new_matrix[:, ti] = mask*x0[:, 0] + (1 - mask)*input_matrix[:, ti]
    new_matrix = pd.DataFrame(new_matrix)
    return new_matrix

```

[62]

```
# # call Here
fine_df = hample_filter(norm_amp)
```

```
114
```

b) For Mayank_relation

Window = **length / 3000**

```
# outlier removal in amplitude
def hample_filter(input_matrix):
    input_matrix= np.asarray(input_matrix)
    n = input_matrix.shape[1]
    print(n)
    new_matrix = np.zeros_like(input_matrix)
    k = 1.4826  # scale factor for Gaussian distribution
    n_sigmas=1
    length = len(input_matrix)
    length = int(length/3000)
    window_size=length # change the value here
    for ti in range(n):

        start_time = max(0, ti - window_size)
        end_time = min(n, ti + window_size)
        x0 = np.nanmedian(input_matrix[:, start_time:end_time], axis=1, keepdims=True)
        s0 = k * np.nanmedian(np.abs(input_matrix[:, start_time:end_time] - x0), axis=1)
        mask = (np.abs(input_matrix[:, ti] - x0[:, 0]) > n_sigmas * s0)
        new_matrix[:, ti] = mask*x0[:, 0] + (1 - mask)*input_matrix[:, ti]
    new_matrix = pd.DataFrame(new_matrix)
    return new_matrix

```

[98]

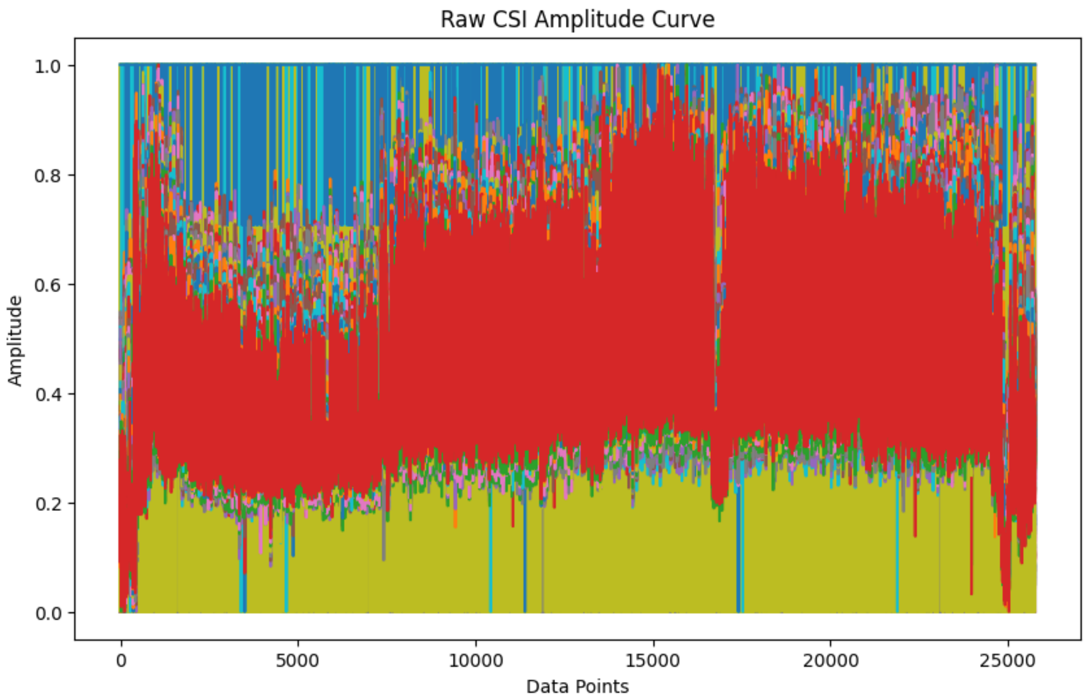
```
# # call Here
fine_df = hample_filter(norm_amp)
```

```
114
```

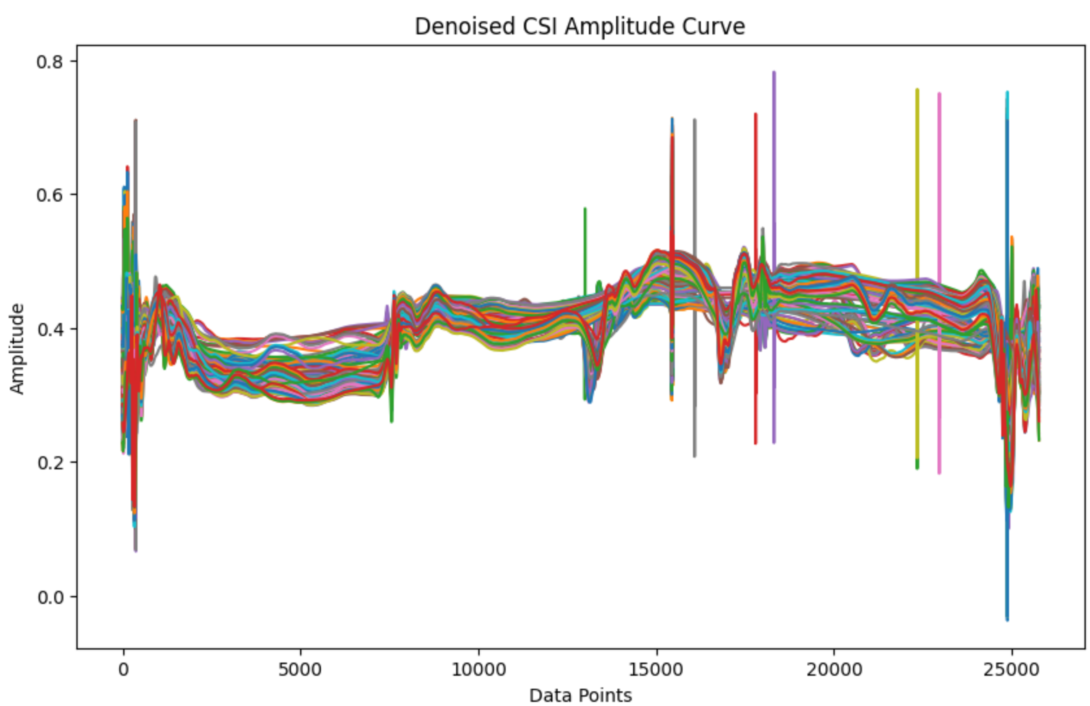
3) Visualization of Amplitude

a) For Mayank_ISA

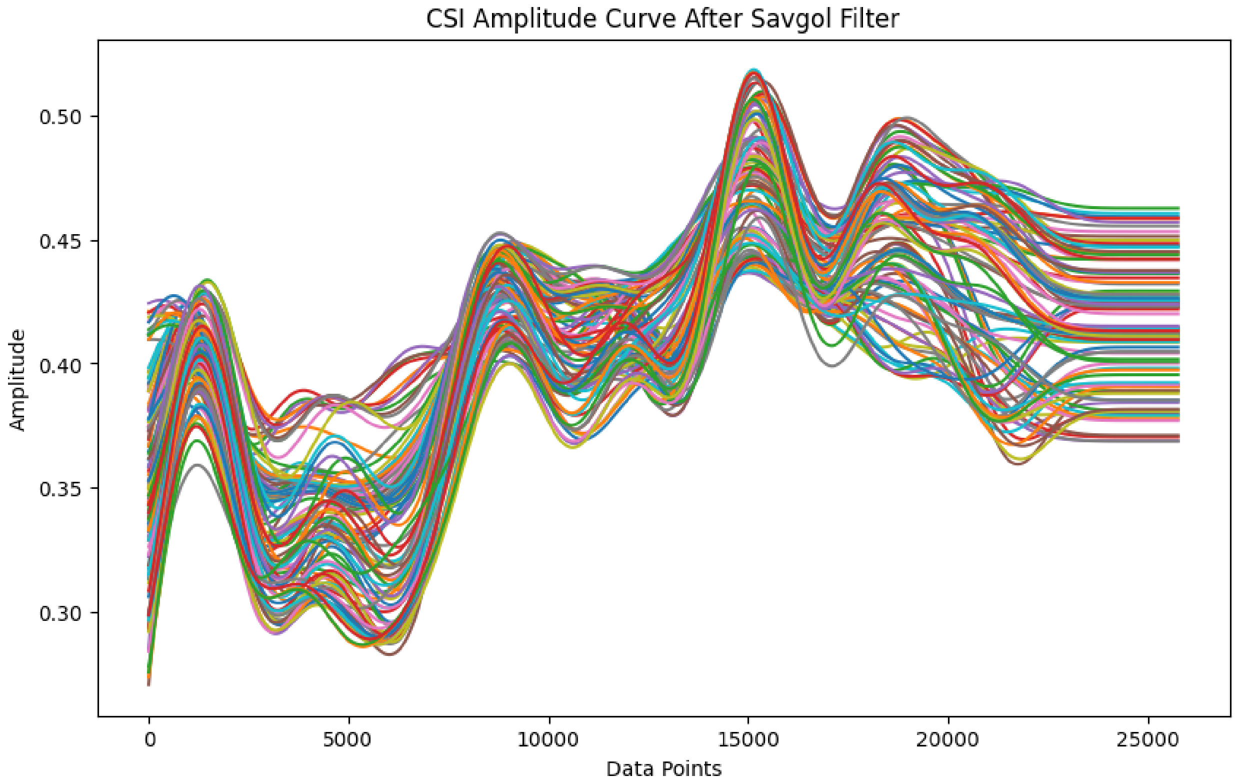
i) Raw CSI Amplitude Curve



ii) After Denoising

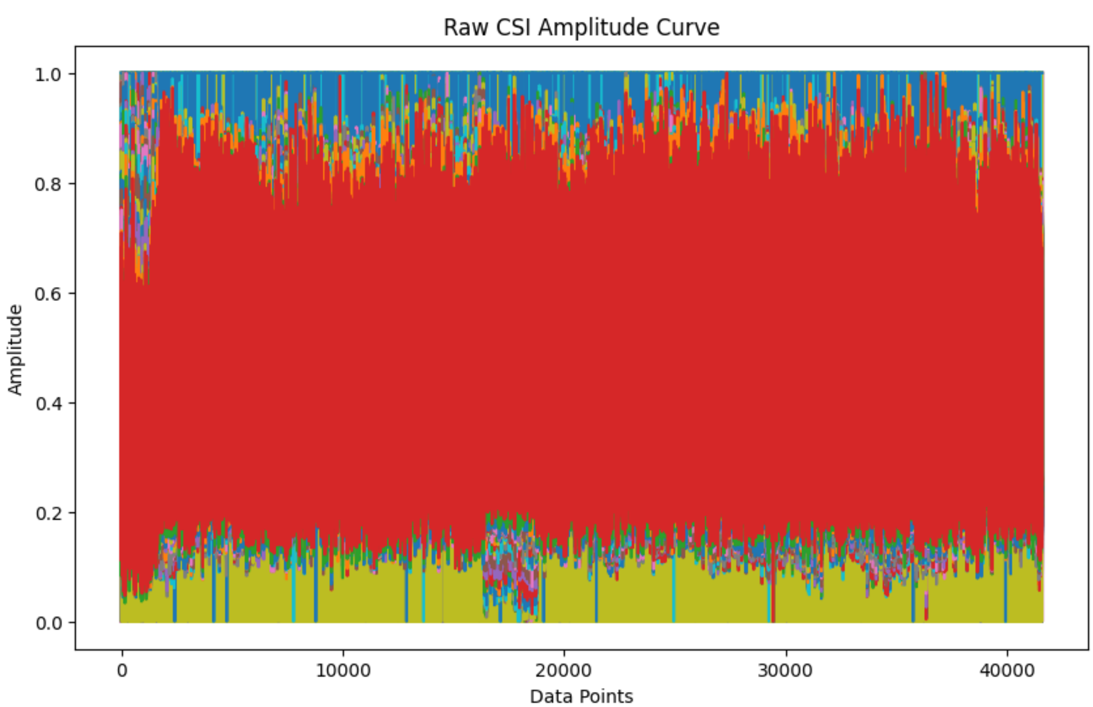


iii) After Savgol Filter

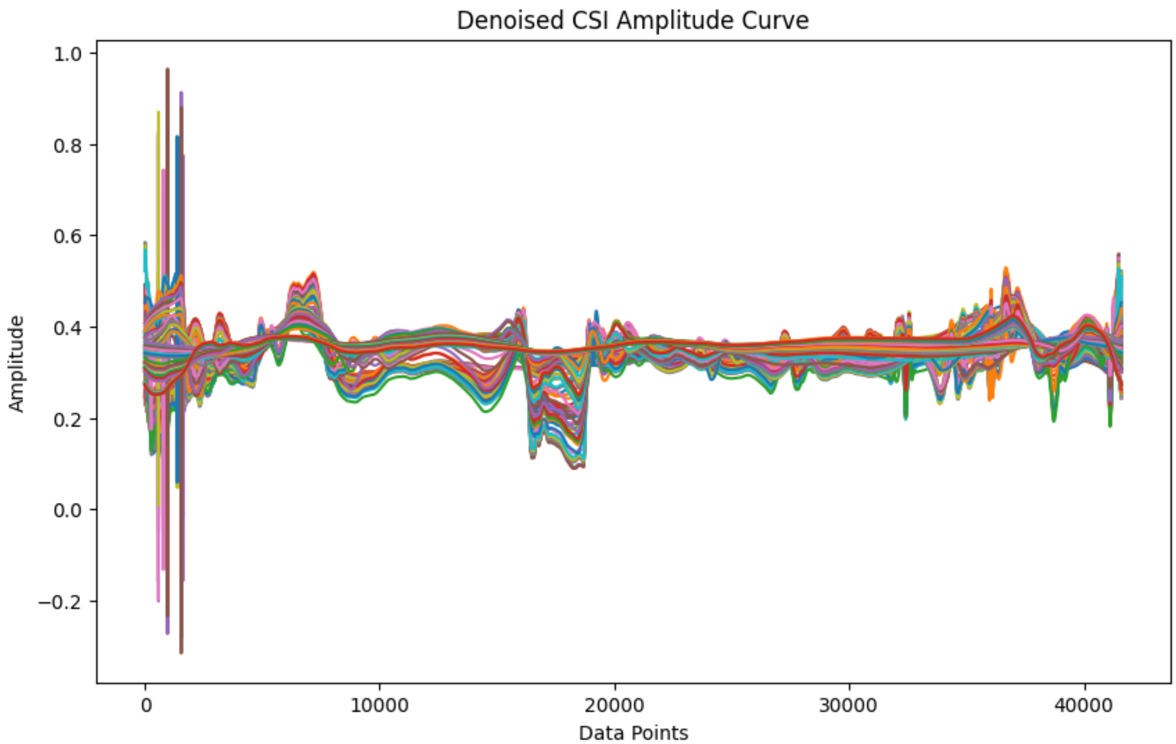


b) For Mayank_relation

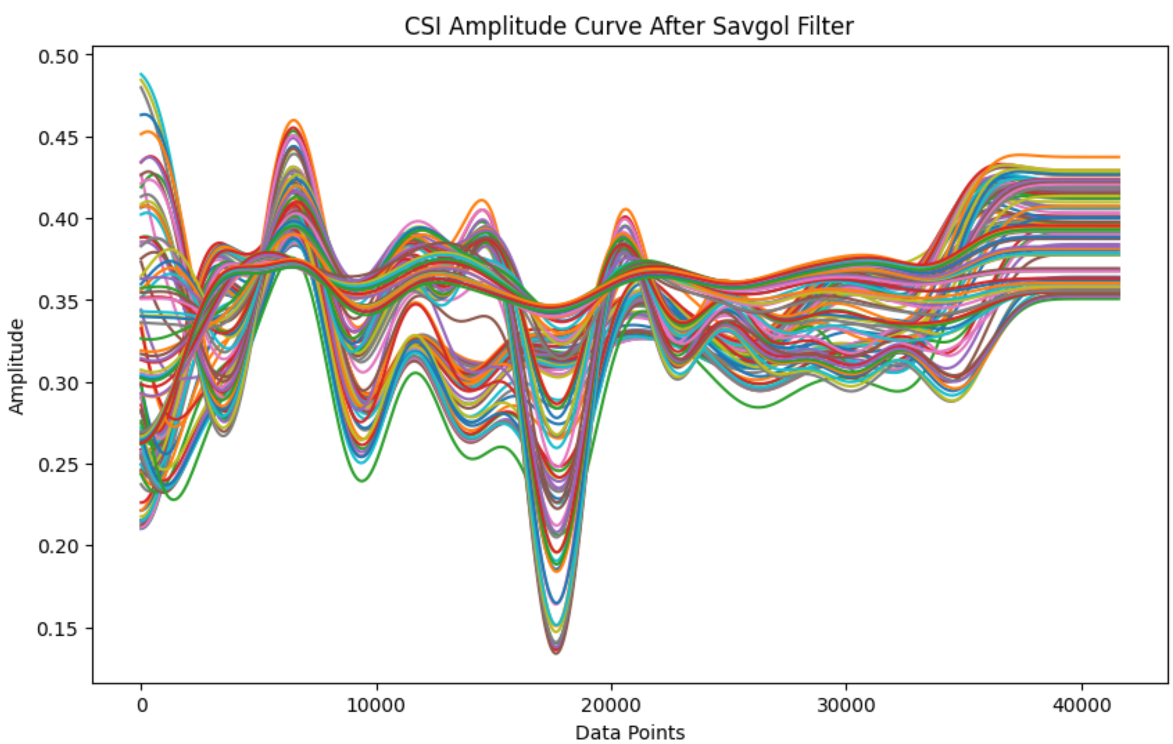
i) Raw CSI Amplitude Curve



ii) After Denoising



iii) After Savgol Filter



4) Dimensions of Input Data

```
[107]
# Make input shape as:
# First reduce the column dimension to 100
# Then make reshape to 10 X 10.
reduced_input = smooth.iloc[:, :100]
reshaped_input = reduced_input.values.reshape((-1, 10, 10))
```

5) Engagement Scores

a) For Mayank_ISA

```
# call Here
engagement_score(time2, t1)
```

```
threshold:  0.0007
0.7169811320754716
```

b) For Mayank_relation

```
[113]
# call Here
engagement_score(time2, t1)
```

```
threshold:  0.0007
0.8571428571428571
```

[Google Collab Link to Code]()