AMATH 582: Homework 1

Kutay Demiralay
Aeronautics and Astronautics Department Department, University of Washington, Seattle, WA

Introduction and Overview

We have an array of data with dimensions (262144, 49), where each column represents acoustic pressure measurements at a specific time. We have 49 time steps, where there is 30 minutes between each time step. The 262144 rows are reshaped into a cube, represented in (64,64,64) array. This way we can store the pressure measurements in 3 dimensions plus time dimension. The data is noisy, and our aim is to locate the submarine in puget sound using this raw data.

Theoretical Background

The Fourier transform (FT) is an integral transformation that converts a function in time domain, into a representation describing the frequencies inherent in the original function. Input of the transform continuous function in the time domain, Output of the transform is a complex or real valued function in the frequency domain. Eq. (1) is the general function for Fourier Transform.

$$\hat{F(k)}=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(x)\, e^{-ikt}\, dx \quad (1)$$

$$e^{ikx}=\cos(kx)+i\sin(kx) \quad (2)$$

In python Fourier Transform is done with single command np.fft.fftn, where n value represents the dimension fft. Np.fft.fftn command is always coupled with np.fft.fftshift and they are used together, the reason is that fftshift rearranges the output of fftn and moves zero frequency to the center of the array. It is good for visualization and interpolation.

Inverse Fourier Transform is an integral transformation that converts a function in the frequency domain, into a function in the time domain. Basically inversing the effects of Fourier Transformation over function. Eq. (3) is the general function for Inverse Fourier Transform. The python command for inverse fourier transform is np.fft.ifftn, with n being the dimension of inverse fourier transform.

$$\hat{F(k)}=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(x)\, e^{-ikt}\, dx \quad (3)$$

Central Frequency or center frequency, is the largest amplitude frequency in the domain. can be obtained by averaging the fourier transform.

Gaussian function, sometimes called a Gaussian distribution or Gaussian bell curve, is a mathematical formula that describes the pattern of a normal distribution of values. Three dimensional gaussian function, shifted with central frequency values, can be represented as in Eq(4). Where Eq.(5) denotes variance.

$$G(\kappa x\, ,\, \kappa y\, ,\kappa z\,) = e^{-\tau(\kappa x - \text{centralfreq}_x\,)^2 + \left(\kappa y - \text{centralfreq}_y\right)^2 + (\kappa z - \text{centralfreq}_z)^2)} \tag{4}$$

$$\tau = \frac{1}{2\,\sigma^2} \tag{5}$$

Eq.(6) represents a brief schema for arriving at filtered data from noisy raw data F(x). Note: Fourier Transform of Gaussian function is still a Gaussian function.

$$F(x) \xrightarrow{FFT} \hat{F}(k) \xrightarrow{Filter} \hat{F}(k)\hat{G}(k) \xrightarrow{IFFT} \text{Filtered data} \tag{6}$$

Algorithm Implementation and Development

All coding was done in the python programming language. Numpy, matplotlib.pyplot, scipy, pwt,mpl_toolkits.mplot3d and math libraries were used.

First Step: Finding The Central Frequency

Central Frequency is found by taking the fourier transform of raw signal (fftshif+ fftn) for each time step, averaging them, taking the absolute value to get rid of complex values and finding the maximum argument within this final array.

```
for time in range(49):
  rawsignal_at_time = np.reshape(d[:, time], (N_grid, N_grid, N_grid))
  fhat[time, :, :, :] = np.fft.fftshift(np.fft.fftn(rawsignal_at_time))
  sumfhat = sumfhat + fhat[time, :, :, :]

avgfhat=sumfhat/49
fhat_abs=np.abs (avgfhat)
centralfreq= np.unravel_index (np.argmax(fhat_abs, axis=None), fhat_abs.shape)
```

Second Step: Creating The Proper Gaussian Filter

I created a Gaussian filter using 3 for loops for each dimension, centered at the central frequency value found in the previous step, with sigma value equal to 3.

```
#create a shifted gaussian filter
for idx in range(64):
 for idy in range(64):
   for idz in range(64):
     gaussian_filter_shifted[idx,idy,idz]= (np.exp((-((idx-32-centralfreq[0])**2 +
(idy-32-centralfreq[1])**2 +
(idz-32-centralfreq[2])**2))/(2*sigma**2)))/(pow(2*np.pi,(3/2))*pow(sigma,3))
```

Third Step: Finding The Position vs. Time

I started with initializing the position array at first. Inside the loop I applied a gaussian filter to each function step in the frequency domain. Then using the inverse fourier transform (ifftn+ifftshit) I got the filtered data which contains position vs. time step data.

```
position= np.zeros(shape=(49,3))
# Apply the filter in the frequency domain
# Inverse FFT to filter the data
b=np.zeros((49,64,64,64),dtype=complex)
for time in range(49):
 a=fhat[time,:,:,:]
 fhat_filtered= a*gaussian_filter_shifted
 b[time2,:,:,:]=np.abs(np.fft.ifftn(np.fft.ifftshift(fhat_filtered)))
 c=b[time, :, :,:]
 position[time2,:] = np.asarray(np.unravel_index(np.argmax(c, axis=None),c.shape))
```

Fourth Step: Plotting the Results

Using Python libraries and various plotting techniques I could plot the results you will see below.

Computational Results

Central frequency values are indexed from -32 to 32.
Central Frequency coefficient values found: [7, 17, -22]

Gaussian Function is 3 dimensional. 2 Dimensional X slice of our Gaussian function, with sigma value equal to 3, centered at central frequency value, at frequency domain pşotted using plt.imshow function on matlab, can be seen in fig. (1) below.

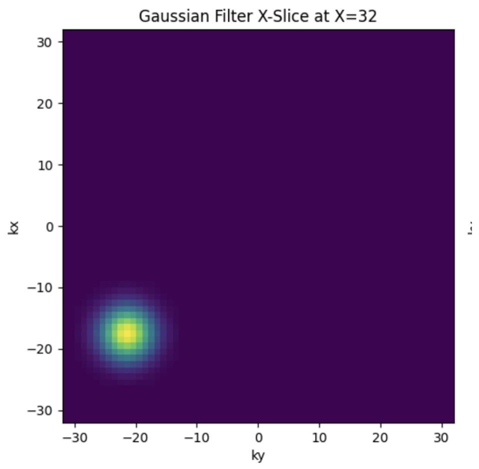

Figure 1: Visualization of Gaussian Filter X-Slice at X=32 in Frequency Domain

After fourier transforming of raw data, then gaussian filtering the raw data in frequency data, and then inverse fourier transforming the filtered data, I got filtered data back in time domain.The x and y position is indexed from -32 to 32 by my choice. In fig.(2) we can see the image of data slice along at a slice along the Z axis, at the same Z=32 value, using python's plt.imshow function. It is shown that the filtration works, the noise is canceled out. Except where the gaussian function works as a filter.

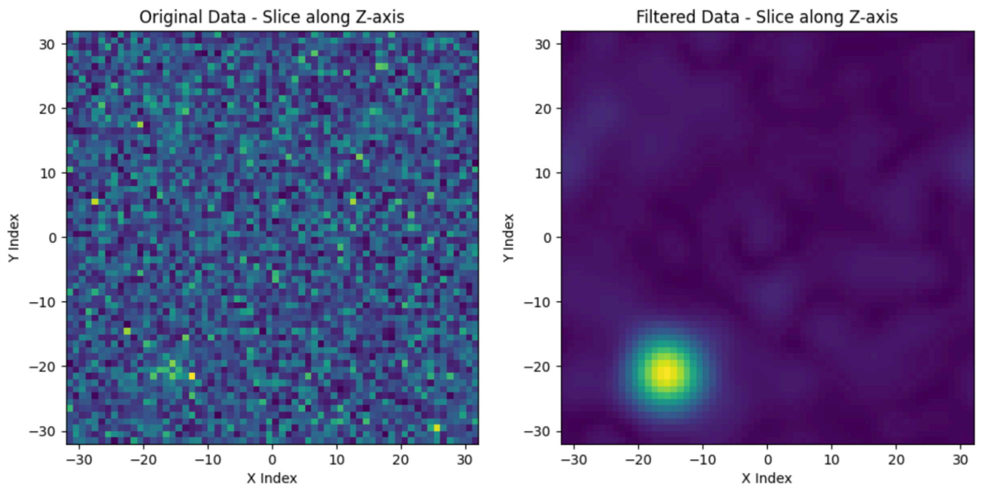

Figure 2: Visualization of Original Data and Filtered data in time domain, at the same Z axis slice.

After showing that the filtration works I plotted the position at 3 different dimensions of the submarine at 49 timesteps or 24 hour intervals with 3 different plots in fig(3). At each subplot I draw both the position values acquired using filtered data and raw data.

Positions are indexed from -32 to 32 by choice. Both plots are not perfectly smooth but it is evident that blue plots (plots using filtered data) are smoother, in all 3 subplots. This further verifies the success of filtration I did earlier.

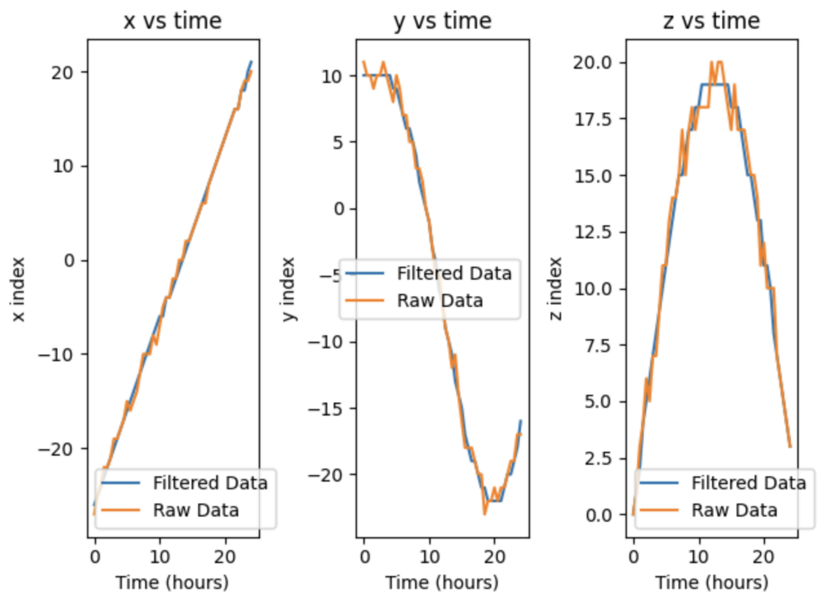

Figure 3: Position in 3 different dimensions vs. time graphs of submarine

I also plotted these 3 different plots in Figure 4,into a single 3d plot, using filtered data only this time The trajectory of the submarine along x,y,z dimensions are visualized. The indexing of x,y,z dimensions are from -32 to 32 by choice.

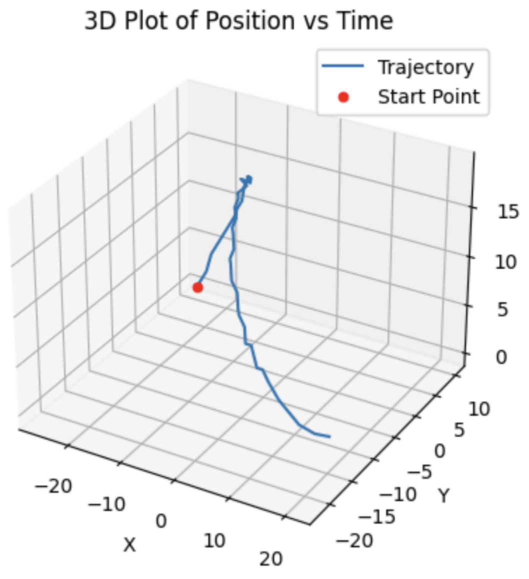

Figure 4: 3D position vs. time graph of submarine

Summary and Conclusion

I was able to plot the submarine's position through a time interval of 24 hours, from the pressure measurements taken in every 30 minute time steps. Through averaging of Fourier series I was able to find the central frequency generated by this submarine, and used this central frequency value to create a gaussian filter in order to denoise the raw measurement data. With denoised data, using fourier transformation I was able to find the path of the submarine in time domain.

There is still some noise in the data as can be seen in figures 2 and 3, additional filtering techniques would be useful. Plus the position plots are not perfectly smooth, even with filtering. By decreasing the time interval between each measurement, we could further smoothen the plots. If that's not possible, through interpolation techniques we could get more data points, with smaller time steps, to further smoothen the graphs.

Acknowledgements

I would like to thank my classmates Po, Shavey, Emoji for useful discussions and Professor Eli for his notes, lectures, Office Hours and code examples

References

[1] Eli Shlizerman (2024). 582 KutzBook. AMATH 582 Wi 24: Computational Methods For Data Analysis.