APPLICATIONS OF FOURIER TRANSFORM TO FILTER ACOUSTIC DATA

XUECHENG LIU

Applied Mathematics Department, University of Washington, Seattle, WA
xl0306@uw.edu

Abstract. A submarine emitting an unknown frequency was in the Puget Sound. By applying Fourier transformation to the 24-period acoustic data and averaging through time, submarine's unknown frequency was detected as $\vec{k} = (5.3, 2.2, -7.0)$. 3D Gaussian filter was applied to the acoustic data to find the path of the submarine in 3D spatial domain. Finally we project the path to 2D xy domain to allow a sub-tracking aircraft to follow.

1. Introduction and Overview

Our goal is to locate the path of a submarine using the nosing acoustic data. We can access a 3-dimensional acoustic data over the 24-hour period with half-hour increments. We first averaging the acoustic data in Fourier space over time to get the frequency signature. Then we apply 3D Gaussian filter with frequency signature as the mean to the acoustic data in Fourier space and transferring data back to the spatial domain to detect the location in each time snap. With this information, we can deploy an aircraft to keep track of the submarine in the future.

2. Theoretical Background

According to [Brunton and Kutz, 2019],the Fourier transform can be used to get spatial frequency information about the acoustic data. The transformation is given by in the 1D setting

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

The original signal can be transferred back via inverse operation which is given by

$$\check{g}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(k) e^{-ikx} \, dk$$

The Fourier transformation in the 3D setting can be generalized as

$$\hat{f}(k_x, k_y, k_z) = \frac{1}{\sqrt{(2\pi)^3}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y, z) e^{-i\pi(xk_x + yk_y + zk_z)} \, dx \, dy \, dz$$

Similarly, in the 3D setting, the original signal can be obtained via inverse transformation which is given by

$$\check{g}(x, y, z) = \frac{1}{\sqrt{(2\pi)^3}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x_k, y_k, z_k) e^{i\pi(xk_x + yk_y + zk_z)} \, dx_k \, dy_k \, dz_k$$

The first step is to average the acoustic data over time in the Fourier space to remove the white noise. Let $\vec{f}$ representing the acoustic data without noise and $\vec{\xi}$ representing the white noise in d-dimension. By linearity property of the Fourier transform, we have

Date: January 29, 2022.

$$\widehat{(\vec{f}+\vec{\xi})}^{(k)} \overset{d}{=} \hat{\vec{f}}^{(k)} + \hat{\vec{\xi}}^{(k)}$$

Take the sum of the Fourier transform and average over time will result in

$$\frac{1}{K}\sum_{k=0}^{K-1}\widehat{(\vec{f}+\vec{\xi})}^{(k)} \overset{d}{=} \frac{1}{K}\sum_{k=0}^{K-1}\hat{\vec{f}}^{(k)} + \frac{1}{K}\sum_{k=0}^{K-1}\hat{\vec{\xi}}^{(k)}$$

Since $\vec{\xi}$ follows normal distribution with mean $\vec{0}$,$\frac{1}{K}\sum_{k=0}^{K-1}\hat{\vec{\xi}}^{(k)}$ will be zero. Thus, we are left with the averaged acoustic data in the Fourier space without noise. The center frequency $k_x,\ k_y,\ k_z$ can be found by taking the index of the maximum value in the averaged data and apply the index to the frequency space.

A 3D Gaussian filter defined as

$$g(x, y, z, s) = e^{-((x-k_x)^2+(y-k_y)^2+(z-k_z)^2)/(2s^2)}$$

where $k_x,\ k_y,\ k_z$ is defined as above, is used to filter out the noise of data in the Fourier space at each time snap. Finally we inverse transfer the frequency data back to spatial data and pick the indices with the largest value to get the coordinate of the submarine in each time snap.

3. **Algorithm Implementation and Development**

The core package I used to apply the Fourier transformation is **Numpy** and the main package to visualize the data and path is **Matplotlib**. The following algorithm illustrates how to find central frequency.

Algorithm 1 Center frequency finding algorithm

import data from **submarine.npy**
for j = 1:49 **do**
  Reshape column j into 3D format
  Apply the Fourier Transformation to reshaped data
  Sum FFT of all column
end for
divide sum of FFT by 49 and find the indices with maximum value
Plug the indices to frequency domain for center frequency

The following algorithm illustrates how to find the location of the submarine over 49 time snaps.

Algorithm 2 Path locating algorithm

Define Gaussian filter with center frequency as mean
for j = 1:49 **do**
  Reshape column j into 3D format
  Apply the Fourier Transformation to reshaped data
  Apply Gaussian filter to Fourier-transformed data
  Apply inverse Fourier transform to the filtered data
  Find the indices with maximum value
  Store the coordinates by plugging the indices to the spatial domain
end for
Plot the path of the submarine

4. **Computational Results**

Figure 1 illustrates the visual inspection of the frequency signature:

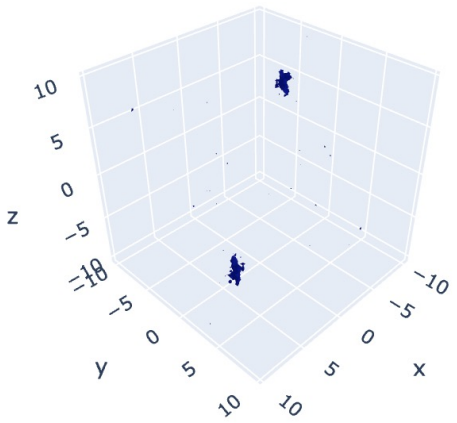

FIGURE 1. Center Frequency of Averaged Signal over Fourier Domain.

Figure 2 illustrates the comparison of the path with and without filtering the data.

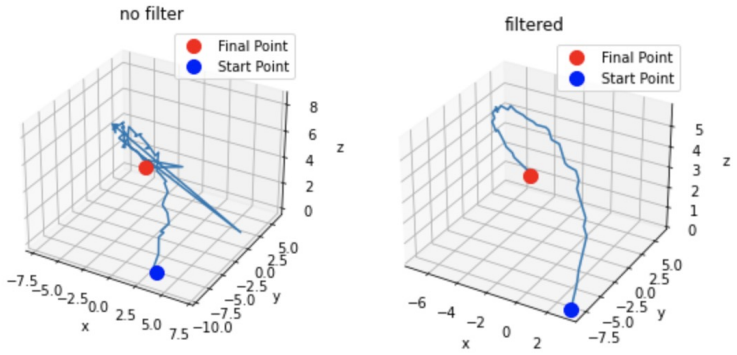

FIGURE 2. Path of the submarine without and with Gaussian filter

We can better visualize the path after filtering noise using the 3D contour plot (which could be rotated manually), as illustrated in Figure 3:

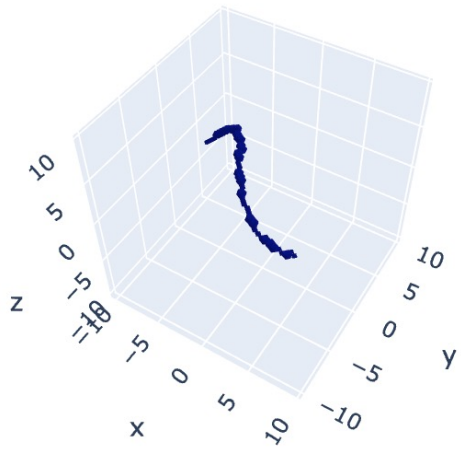

FIGURE 3. 3D Contour Plot of Submarine's Path.

Figure 4 shows the path of the submarine over 24 hour period in the x direction,y direction and x, y plane respectively:

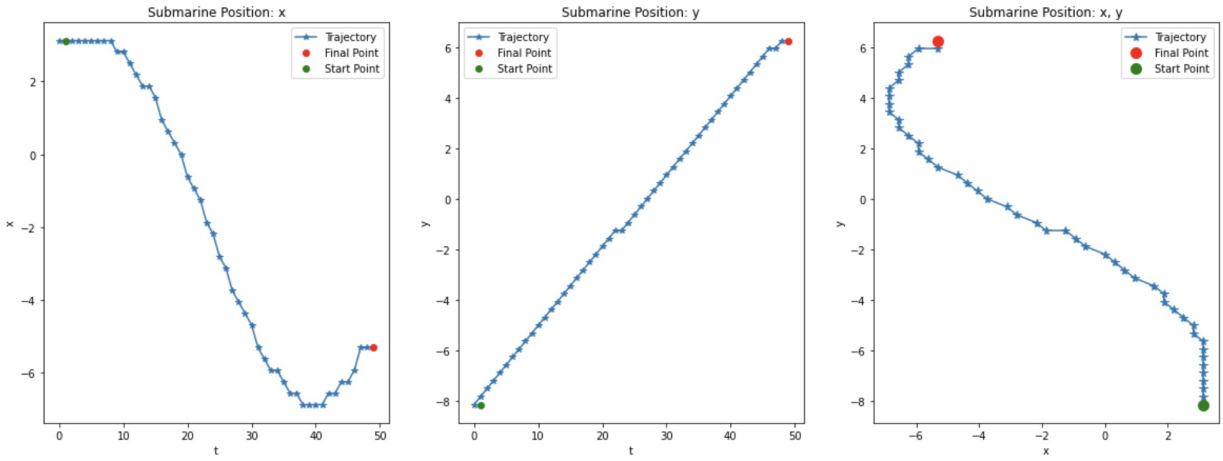

FIGURE 4. Path of submarine over x direction,y direction and x, y plane.

5. **Summary and Conclusions**

This study shows that Fourier transform is a powerful and efficient technique to resolve issues in high dimensions. It also shows that averaging the signal over time in Fourier domain can effectively remove the noise of data and construct a Gaussian filter with center frequency to better remove the noise in each time snap. The future work and be explored by applying different non-Gaussian filters and compare the results with Gaussian filters. The final location in x, y coordinate is $(-5.3, 6.25)$, where we can send an aircraft for future tracking.

Acknowledgement

The author is thankful to Prof. Bamdad Hosseini and Teaching Assistant, Katherine Grace Owens for useful discussions and helper visualizing function about the problem. I am also thankful to Qiyao Liu for providing charming selfies to help me practice the Fourier analysis in 2D settings.

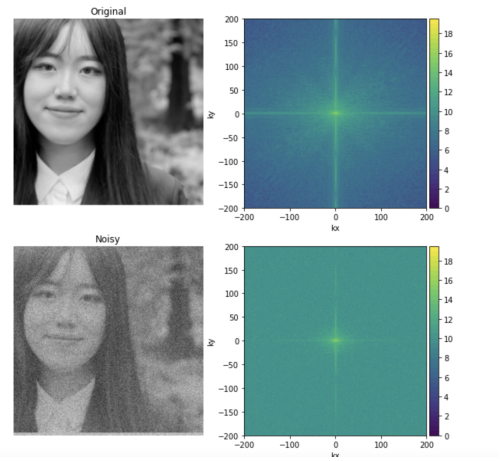

FIGURE 5. A beauty and her Fourier Transform.

REFERENCES

[Brunton and Kutz, 2019] Brunton, S. and Kutz, J. (2019). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press.

Appendix

Python code is attached as following:

```
import numpy as np

data_path = "subdata.npy" # make sure npy file is under the same directory
d = np.load(data_path) # huge matrix of size 262144 x 49 (columns contain flattened 3d matrix

# **Visualizing the dataset in the first timesnap**

# import libraries for plotting isosurfaces
import plotly
import plotly.graph_objs as go
# utility for clearing output of cell as loop runs in notebook
from IPython.display import clear_output

# plot the data in time

L = 10; # length of spatial domain (cube of side L = 2*10)
N_grid = 64; # number of grid points/Fourier modes in each direction
xx = np.linspace(-L, L, N_grid+1) #spatial grid in x dir
x = xx[0:N_grid]
y = x # same grid in y direction
z = x # same grid in z direction

K_grid = (2*np.pi/(2*L))*np.linspace(-N_grid/2, N_grid/2 -1, N_grid) # frequency grid for each

xv, yv, zv = np.meshgrid( x, y, z) # generate 3D meshgrid in spatial domain
```

```
# plot iso surfaces in the first timesnap, change the end value to see more timesnaps
for j in range(0,1,3):

  signal = np.reshape(d[:, j], (N_grid, N_grid, N_grid)) # reshape the jth column
  normal_sig_abs = np.abs(signal)/np.abs(signal).max() # normalize the data

  # generate data for isosurface of the 3D data
  fig_data = go.Isosurface( x = xv.flatten(), y = yv.flatten(), z = zv.flatten(),
                            value = normal_sig_abs.flatten(), isomin=0.7, isomax=0.7)

  # generate plots
  clear_output(wait=True) # need this to discard previous figs
  fig = go.Figure( data = fig_data )
  fig.show()


ave_fft = np.zeros((64,64,64))
for j in range(49):
    d_hat = np.fft.fftn(np.reshape(d[:, j], (64, 64, 64)))
    d_hat = np.fft.fftshift(d_hat)
    ave_fft = ave_fft + d_hat

ave_fft = ave_fft/49 # we have 49 timesnaps in total

# get the spatial coordinate of the submarine
idx = np.argmax(np.abs(ave_fft))
idc = np.unravel_index(idx,(64,64,64)) # unflaten the index of center frequency
print("The spatial coordinates are ",x[idc[0]],y[idc[1]],z[idc[2]])

kx,ky,kz = np.meshgrid(K_grid,K_grid,K_grid)# frequency grid
# get the center frequency
print("The center frequency is ",kx[idc],ky[idc],kz[idc])


# **Visualize the center frequency**

normal_ave_fft = np.abs(ave_fft)/np.real(np.max((ave_fft)))
fig_data = go.Isosurface( x = kx.flatten(), y = ky.flatten(), z = kz.flatten(),
                          value = normal_ave_fft.flatten(), isomin=0.5, isomax=1)

fig = go.Figure( data = fig_data )
fig.show()


# **We first track the data without filter. This is done by taking the coordinate with the gre

# **Clearly we could see there are outliers**

# In[7]:
```

```
import matplotlib.pyplot as plt
import numpy as np

xc = []
yc = []
zc = []

for i in range(49):
    temp = np.reshape(d[:, i], (64, 64, 64))
    idx = np.argmax(temp)
    id2 = np.unravel_index(idx, (64, 64, 64))

    xc.append(xv[id2])
    yc.append(yv[id2])
    zc.append(zv[id2])

ax = plt.axes(projection='3d');
ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_zlabel('z')
ax.plot(xc, yc, zc,'-');
plt.plot(xc[-1], yc[-1], zc[-1],
        'ro', markersize=10, label='Final Point');
plt.plot(xc[0], yc[0], zc[0],
        'bo', markersize=10, label='Start Point');
#ax.view_init(15,60);
ax.set_title("no filter");
plt.legend();


# **This time we apply a 3D Gaussion filter to the fourior transformation of data in each time

# In[8]:


def Gaussian(x,y,z,s):
    return np.exp( - ( ((x-kx[idc])**2 + (y-ky[idc])**2 + (z-kz[idc])**2))/(2*s**2))


# In[9]:


k_grid = np.linspace(-10, 10, 64)
k1, k2, k3 = np.meshgrid(K_grid, K_grid,K_grid)
sigma = 2
g_vals = Gaussian(k1, k2,k3, sigma)

xc = []
yc = []
zc = []
for i in range(49):
```

```
    d_hat = np.fft.fftn(np.reshape(d[:, i], (64, 64, 64)))
    d_hat = np.fft.fftshift(d_hat)
    d_hat_filtered = g_vals*d_hat
    d_filtered = np.abs(np.fft.ifftn(np.fft.ifftshift(d_hat_filtered)))

    idx = np.argmax(d_filtered)
    id = np.unravel_index(idx,(64,64,64))


    xc.append(xv[id])
    yc.append(yv[id])
    zc.append(zv[id])

ax = plt.axes(projection='3d')


ax.plot(xc, yc, zc,'-');
plt.plot(xc[-1], yc[-1], zc[-1],
        'ro', markersize=10, label='Final Point')
plt.plot(xc[0], yc[0], zc[0],
        'bo', markersize=10, label='Start Point')
ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_zlabel('z')
ax.autoscale();
#ax.view_init(100,60);
ax.set_title("filtered");
plt.legend()
plt.show()


# In[17]:


value = np.zeros((64,64,64))
for i in range(49):
    value[int(np.where(x == xc[i])[0])][int(np.where(y == yc[i])[0])][int(np.where(z == zc[i])

fig_data = go.Isosurface( x = xv.flatten(), y = yv.flatten(), z = zv.flatten(),
                          value = value.flatten(), isomin=0.01, isomax=1)

fig = go.Figure( data = fig_data )
fig.show()


# In[18]:


# 2D Projections
fig = plt.figure(figsize=(20, 7));

# x position
```

```
plt.subplot(1,3,1);
plt.plot(xc,'-*',label = 'Trajectory');
plt.plot(49,xc[-1],'ro',label='Final Point');
plt.plot(1,xc[0],'go',label='Start Point');
plt.xlabel('t'); plt.ylabel('x');
plt.title('Submarine Position: x');
plt.legend();

#y position
plt.subplot(1,3,2);
plt.plot(yc,'-*',label = 'Trajectory');
plt.plot(49,yc[-1],'ro',label='Final Point');
plt.plot(1,yc[0],'go',label='Start Point');
plt.xlabel('t'); plt.ylabel('y');
plt.title('Submarine Position: y');
plt.legend();


# x, y position
plt.subplot(1,3,3);
plt.plot(xc, yc, '-*', label='Trajectory', markersize=8);
plt.plot(xc[-1], yc[-1], 'ro',label='Final Point', markersize=10);
plt.plot(xc[0], yc[0], 'go',label='Start Point', markersize=10);
plt.xlabel('x'); plt.ylabel('y');
plt.title('Submarine Position: x, y');
plt.legend();

print("The end location of the submarine is (",xc[-1],",",yc[-1],")")
```