Acoustic Positioning With DFT Noise Filtering

Arnold Peter Ruymgaart*

University of Washington
apruymgaart@gmail.com githubudemy

March 20, 2021

Abstract

In this assignment we are provided with noisy acoustic submarine positioning data. The data comprises discrete time measurements made at half hour increments organized into a 4D array. Each time frame is a 3D array of cartesian coordinates containing the acoustically detected intensity volume. The data is transformed into reciprocal space by 3D FFT. Averaging the transformed signal increases the signal to noise ratio and allows discovery of the frequency "signature". In reciprocal space, the frequency volume containing the signal remains centered at the same position regardless of time index while the signal position in lab space follows a trajectory. The fixed location of the center frequency in reciprocal space allows placement of a Gaussian filter centered at the central frequency component. Each time frame is FFT 3D transformed, filtered in k space and subsequently transformed back. The resulting denoised echo is now positioned by a simple maximum allowing the recovery of the de-noised trajectory.

I. Introduction & Overview

SOund Navigation And Ranging (SONAR) or SONAR-like acoustic positioning systems can be used to discover and localize underwater targets such as submarines. As the name implies, SONAR utilizes sound waves to accomplish these tasks. SONAR systems can be active or passive. Active systems transmit a sound at a certain frequency and measure return signals (echoes) that are reflected back from targets. Passive systems listen and triangulate without transmission.

The principle of operation of active SONAR is as follows. First a sound signal of a certain frequency composition is transmitted. Upon transmission, a timer is started and the receiver component "listens" for any reflected return signal (an echo). The timing between the transmission and return determines the distance of the target object. More than a single sound detector allows triangulation and positioning rather than mere distance sounding. If the target emits an acoustic "signature" then the concept of passive sonar is similar. The position can be found by triangulation of timing data from multiple sensors.

II. Theoretical Background

We are provided with noisy data. Each time frame of the data contains a shape (a volume) in 3D Cartesian space the center of which is the location of the submarine (see Section IV). If the data were not noisy we could simply detect the maximum or center of mass directly (see Table 3) in this particular problem as discussed in Section V. So the main task at hand is de-noising the signal. In order to accomplish that we will find the center frequency and fliter around that center frequency in Fourier space. The following principles are needed and briefly reviewed:

A Complex numbers & their moduli

B Orthogonal system expansion & Fourier basis

C Wavelengths, wavenumbers & scaling

D Fourier transform (FT) & its inverse (IFT)

E Discrete Fourier transform (DFT)

F Gaussian filter

(A) Complex Numbers & Their Moduli. We briefly review complex numbers: A complex number $z = a + bi$ is a position in the complex plane. It has a real component a and a complex component b. While the L_1 norm or "length" or a real number $x \in \mathbb{R}$ is its absolute value $|x|$, the absolute value or "length" of a complex number (also called modulus or radius from origin) is $|z| = \sqrt{a^2 + b^2}$. When we plot a $1D$ curve of complex values we often plot the absolute values.

(B) Orthogonal Expansion & Fourier Basis. A vector can be represented as a linear combination of vectors

*Contact for further information

that are elements of a basis set of vectors. Likewise, a function f can often be expanded into a linear combination of orthogonal basis functions. This is what is done in Fourier series representation and Fourier transform. The orthogonal basis functions in this case are $\sin 2n\pi$ and $\cos 2n\pi$ for $n \in \mathbb{N}$. They are orthogonal on the interval $[0, 2\pi]$ or equivalently $[-\pi, \pi]$. But they are not orthogonal on any arbitrary interval. So we can expand the function we wish to represent only in this interval. If the interval we need is a different length, we need to scale it to $[-\pi, \pi]$. Because sine and cosine are periodic on 2π, representing an arbitrary function on interval $[a, b]$ as a Fourier series has the effect of repeating the curve in that interval indefinitely along the real line as illustrated in Figure 1.

The *coefficients are determined by the projection of f onto the basis functions*. Just like vector projections onto a basis set, we can use the inner product to calculate the magnitude of these projections. For a continuos function, the inner product on interval $[a, b]$ is defined as $\langle f(x), g(x)\rangle = \int_a^b f(x)g(x)dx$. For an expansion of function $f(x) = \sum c_n\psi_n$ onto general orthogonal basis $\{\psi_n\}$, the coefficient c_n on basis function ψ_n is found as follows:

$$c_n = \frac{\langle f, \psi_n\rangle}{\langle \psi_n, \psi_n\rangle} \tag{1}$$

The coefficients are the magnitude of that particular basis function required to construct the function.

If we take the coefficient c_n to be a complex number, we can use Euler's formula $e^{iy} = \sin y + i\cos y$ to write the Fourier series as a complex exponential. For $[a, b] = [-L, L]$:

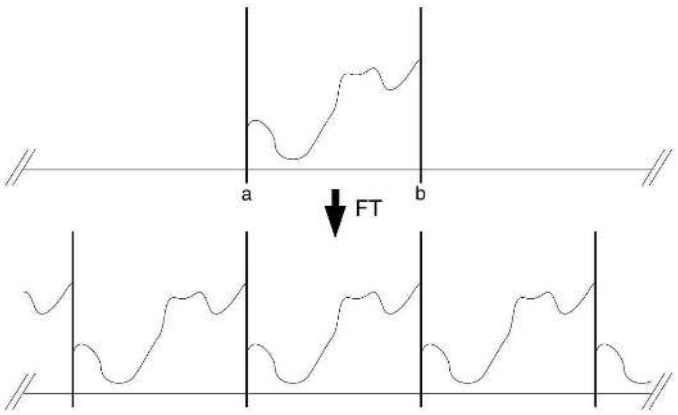
$$f(x) \sim \sum_{-\infty}^{\infty} \mathbf{c}_n e^{i\frac{n\pi}{L}x} dx \tag{2}$$

$$\mathbf{c}_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-i\frac{n\pi}{L}x} dx \tag{3}$$

where complex coefficient $\mathbf{c}_n = a_n + b_n i$ real component a_n is the coefficient on the basis cosine of $\frac{n\pi}{L}x$ while b_n is the coefficient on the sine. *Because sine is odd* $b_{-n} = -b_n$. Derivation of the complex exponential form from standard series is included in appendix C.

(C) Wavelengths, Wavenumbers & Scaling. *Wavelength* λ is the distance between crests. This is the distance after which the wave repeats. For the Fourier basis functions $\sin\left(\frac{2n\pi}{b-a}x\right)$ and $\cos\left(\frac{2n\pi}{b-a}x\right)$ the number of times the wave repeats is the wave index n (number of crests in the interval). For $[a, b] = [-L, L]$ we define *scaling factor*

Figure 1: *Representation In Fourier Basis*



Top: function f defined on the interval $[a, b] \subset \mathbb{R}$. The interval (a, b) must be scaled such that $b - a = 2\pi$ in order for the basis to remain orthogonal. Bottom: representation of f as Fourier series. Since since the basis functions are periodic on 2π the function segment in (a, b) when represented in the Fourier basis will repeat indefinitely in both directions. This is the case even if the function in (a, b) is not by nature a repeating function (e.g. it could be a segment of x^2 or e^x, etc. which are not repeating functions.

$$s = \frac{2\pi}{b-a} = \frac{2\pi}{2L} = \frac{\pi}{L} \tag{4}$$

This factor accomplishes scaling the system interval described in the previous section to the orthogonal Fourier domain. Now to get wavelength λ we divide the length $b - a$ of the interval $[a, b]$ the function was defined on by n. So $\lambda = \frac{b-a}{n}$ and in this problem equals $\frac{2L}{n} = \frac{2\pi}{ns}$.

We now define *wavenumber* as the scaled wave index:

$$k_n = \frac{2\pi}{\lambda} = ns = \frac{n\pi}{L} \tag{5}$$

The Fourier series in complex exponential form, substituting wavenumber into eqn. 2:

$$f(x) \sim \sum_{-\infty}^{\infty} \mathbf{c}_n e^{ik_n x} dx \tag{6}$$

(D) Fourier Transform & Its Inverse . The Fourier series in complex exponential form shown in equations 2 and 6 is defined on an interval $(-L, L)$ rescaled to $(-\pi, \pi)$ where $L < \infty$. The difference between 2 adjacent wavenumbers $\Delta k = k_{n+1} - k_n = \frac{(n+1)\pi}{L} - \frac{n\pi}{L} = \frac{\pi}{L} = s$. Then if we take the limit as $L \to \infty$ (and $\Delta k \to 0$ or equivalently $s \to 0$) we obtain the Fourier transform. Derivation included in Appendix C. Importantly, the frequency spectrum has become *continuous*: there are no gaps between adjacent frequencies. The Fourier transform $\mathscr{F} : L^2 \to L^2$ maps a *function* of variable x to another function of the reciprocal

variable $k \propto \frac{1}{x}$. Both the domain and codomain L^2 is the space of square integrable functions[1]. The Fourier $F(k) = \mathscr{F}[f(x)]$ transform of function $f(x)$ is defined as follows:

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ikx} dx \tag{7}$$

The function $f(x)$ can be recovered from $F(k)$ by the inverse Fourier transform

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(k) e^{ikx} dk \tag{8}$$

The Fourier transform of function $f(x, y, z)$ in 3 dimensions (with $\vec{k} \cdot \vec{x} = k_x x + k_y y + k_z z$)

$$F(k_x, k_y, k_z) = \frac{1}{\sqrt{2\pi}} \iiint_{-\infty}^{\infty} f(x) e^{-i(\vec{k} \cdot \vec{x})} dx dy dz \tag{9}$$

(E) Discrete Fourier Transform (DFT). In practice, computationally, we cannot represent $(-\infty, \infty)$ so we have an interval $[a, b]$ with finite limits. In addition, the number of points inside this interval is finite on the computer while in reality, even in an interval, the number of points is uncountable. The input function f on the computer is an array of values of $f(x)$ at discrete points $x_i \in [a, b]$. We will denote discrete f as $\vec{f}$. The transform of $\vec{f}$ calculated on the computer in this case is called the discrete Fourier transform. This transform is almost identical to a Fourier series approximation of $\vec{f}$. In fact, we can obtain the complex coefficients and wavenumbers directly from the (shifted) DFT result. For convenience, we define $DFT[\vec{f}] \equiv \vec{F}$ and indexed F_i indicates the i^{th} element of $\vec{F}$. We can represent $\vec{f}$ perfectly if we choose at least at total $N = |\vec{f}|$ Fourier modes (number of coefficients of DFT). Via FT of Dirac comb we can show that we can recover the Fourier series coefficients from the DFT: $\vec{c} = \frac{1}{N}\vec{F}$. If the length of discrete function array is even, (and integer power of 2) then wave indices $\vec{n} = \left[-\frac{N}{2}, \frac{N}{2} - 1\right] \cap \mathbb{Z}$ and wavenumbers $\vec{k} = s\vec{n}$. For example, if $N = 64 = 2^6$ then the *shifted* numbers are $\vec{n} = (-32, -31, \ldots, 31)$. And we have $\vec{k} = s\vec{n} = \frac{\pi}{L}\vec{n} = (-32s, -31s, \ldots, 31s)$.

We can reconstruct $\vec{f}$, effectively performing the inverse discrete transform, as follows:

$$f_i = \frac{1}{N} \sum_{i=1}^{N} \Re(F_i) \cos(k_i) + \Im(F_i) \sin(k_i) \tag{10}$$

In the process of converting the Fourier series from standard to complex exponential form (see Appendix C), the summation is doubled *and we now have negative wavenumbers*. Wave indices n now start at $-N/2$ rather than 0. Since cosine is an even function $\cos(x) = \cos(-x)$ and the spectrum is repeated twice with all negative wavenumbers. Since sine is odd we have $\sin(x) = -\sin(-x)$ and the sine coefficient $b_{-n} = -b_n$. An example reconstruction of a Gaussian from its DFT in Python is available here: reconstructGaussian.py

(F) Gaussian Filter. A Gaussian signal (or filter) g is produced by the function:

$$g(x, y, z) = e^{-r(x-p_x)^2 - r(y-p_y)^2 - r(z-p_z)^2} \tag{11}$$

where the tuple (x, y, z) is Cartesian (or k space) position and constant tuple (p_x, p_y, p_z) is the center position of the Gaussian. The radius is modulated by bandwidth variable r ([Kutz, 2013], Section 13.2). A Gaussian has the property of having the same shape as its Fourier transform. In k space, the filter is applied by multiplying it with the signal to be filtered:

$$S(k_x, k_y, k_z) \cdot g(k_x, k_y, k_z) \tag{12}$$

where $S(k_x, k_y, k_z)$ is the signal at position (k_x, k_y, k_z). This is equivalent to a convolution in lab space. Filter width should be taken into consideration. A wider filter in Fourier space is narrower in lab space.

III. Algorithm Implementation & Development

In the first portion of the script, the scale factor for the Fourier domain is calculated as described in Section II C. It is named *scale* in the Python code listed in Appendix B. The code further accomplishes the following (in order):

A. Loop over frames and calculate DFT as described in Section II E of each frame. In Python-numpy, the command $np.fft.fftn(\ldots)$ carries out the 3D DFT. The transformed signals are stored and an average transformed image is calculated.

B. Find the center frequency simply by looking for the maximum of the average FT image. The ND index of the maximum of an ND array is found with command $np.unravel\_index(np.argmax(\ldots), shape)$.

C. Make a filter in k space as described in Section II F, eqn. 11. Choice is a Gaussian filter centered at the center frequency.

D. Loop over the FT images that were stored in step 1 and apply the filter with Python-numpy $np.multiply(S, g)$. Then transform back to lab space

[1] AMATH 503 personal lecture notes

Table 1: *Center frequency*

	x	y	z
Wave index n_i	17	7	-22
Wavenumber $k_i = sn_i$	5.34	2.199	−6.912
Wavelength (units of L)	1.18	2.86	0.91

with $np.ifftn(\dots)$. The center of the sub (position) is the peak of the denoised lab space image.

The implementation of each of the above is included in appendix B. The above code sections are labeled (III.A) through (III.D).

IV. Computational Results

We are provided with noisy echo (or passively emitted) data in form of a signal intensity Cartesian space mesh $S(x, y, z)$ at 49 time points. Both S and time t are discrete and the shape of the data tensor is $|\vec{x}| \times |\vec{y}| \times |\vec{z}| \times |\vec{t}| = 64 \times 64 \times 64 \times 49$. The signal appears spherical when viewed with 3D isosurface plot. We therefore expect a single maximum to indicate target position.

The signal frames provided are presumed to be in lab space because the signal is clearly moving in position as the frames progress, something we would not expect in frequency space. The signal moving in space causes the averaged signal as collected to occupy an elongated volume which encloses the trajectory. This is visualized in Figure 2 A. Conversely, the average of the Fourier transforms of the signal is localized within a small stationary 3D k space volume as visualized in Figure 2 B.

The DFT of the signal at each time frame is stored and separately summed. The summation of the FT signal results in denoising as visualized in Figure 3. The latter denoising allowed location of the center frequency listed in Table 1 and visualized in Figure 4 by simple calculation of the global single maximum of the average.

A Gaussian filter was made in Fourier space at the signal maximum as visualized in Figure 2 C and stored. The previously stored FT signals are subsequently retrieved and the Gaussian filter is applied by multiplication in Fourier space as illustrated in Figure 4. After applying the filter to each signal frame, it is transformed back to real space by inverse DFT. Each resulting real space image appears similar in shape to a clean Gaussian (no visulaization included) with a clear maximum. This maximum is the target position and is stored for each frame as listed in Table 2 and plotted in Figure 5. For reference, position recovered from maxima in real space directly without any transforms or denoising is shown in Table 3.

Table 2: *Submarine Position*

Time	x	y	z
01 (00:00)	3.12	−8.12	0.00
02 (00:30)	3.12	−7.81	0.31
03 (01:00)	3.12	−7.50	0.62
04 (01:30)	3.12	−7.19	1.25
05 (02:00)	3.12	−6.88	1.56
06 (02:30)	3.12	−6.56	1.88
07 (03:00)	3.12	−6.25	2.19
08 (03:30)	3.12	−5.94	2.50
09 (04:00)	3.12	−5.62	2.81
10 (04:30)	2.81	−5.31	3.12
11 (05:00)	2.81	−5.00	3.44
12 (05:30)	2.50	−4.69	3.75
13 (06:00)	2.19	−4.38	4.06
14 (06:30)	1.88	−4.06	4.38
15 (07:00)	1.88	−3.75	4.69
16 (07:30)	1.56	−3.44	5.00
17 (08:00)	0.94	−3.12	5.00
18 (08:30)	0.62	−2.81	5.31
19 (09:00)	0.31	−2.50	5.31
20 (09:30)	0.00	−2.19	5.62
21 (10:00)	−0.62	−1.88	5.62
22 (10:30)	−0.94	−1.88	5.94
23 (11:00)	−1.25	−1.25	5.94
24 (11:30)	−1.88	−1.25	5.94
25 (12:00)	−2.19	−0.94	5.94
26 (12:30)	−2.81	−0.62	5.94
27 (13:00)	−3.12	−0.31	5.94
28 (13:30)	−3.44	0.00	5.94
29 (14:00)	−4.06	0.31	5.94
30 (14:30)	−4.38	0.62	5.94
31 (15:00)	−4.69	0.94	5.62
32 (15:30)	−5.31	1.25	5.62
33 (16:00)	−5.62	1.56	5.31
34 (16:30)	−5.94	1.88	5.31
35 (17:00)	−6.25	2.19	5.00
36 (17:30)	−6.25	2.50	5.00
37 (18:00)	−6.56	2.81	4.69
38 (18:30)	−6.56	3.12	4.38
39 (19:00)	−6.88	3.44	4.06
40 (19:30)	−6.88	3.75	3.75
41 (20:00)	−6.88	4.06	3.44
42 (20:30)	−6.88	4.38	3.44
43 (21:00)	−6.56	4.69	2.81
44 (21:30)	−6.56	5.00	2.50
45 (22:00)	−6.25	5.00	2.19
46 (22:30)	−6.25	5.62	1.88
47 (23:00)	−5.94	5.62	1.56
48 (23:30)	−5.31	5.94	1.25
49 (24:00)	−5.00	6.25	0.94

Table 3: *Unfiltered Position Sample*

Time	x	y	z
01 (00:00)	3.44	−8.44	0.00
02 (00:30)	3.12	−7.81	0.31
03 (01:00)	3.12	−7.50	0.94
04 (01:30)	2.81	−6.88	1.25
05 (02:00)	3.12	−6.88	1.88
06 (02:30)	3.12	−6.56	1.56
07 (03:00)	3.44	−5.94	2.19
08 (03:30)	3.12	−5.94	2.19
09 (04:00)	2.81	−5.62	2.81
10 (04:30)	2.50	−5.31	3.44

V. Summary & Conclusions

(1) Submarine Center Frequency. Denoising by averaging the FT of the signal was effective at increasing the signal to noise ratio (see Figure 3) and allowed recovery of the spectral pattern of the signal as visualized in Figure 4. The central wavelengths/wavenemubers are listed in Table 1. *Filter width*: various widths were tried in an attemtp to minimize noise in the result trajectory. Widths $r \in [0.1, 0.5]$ produced best results.

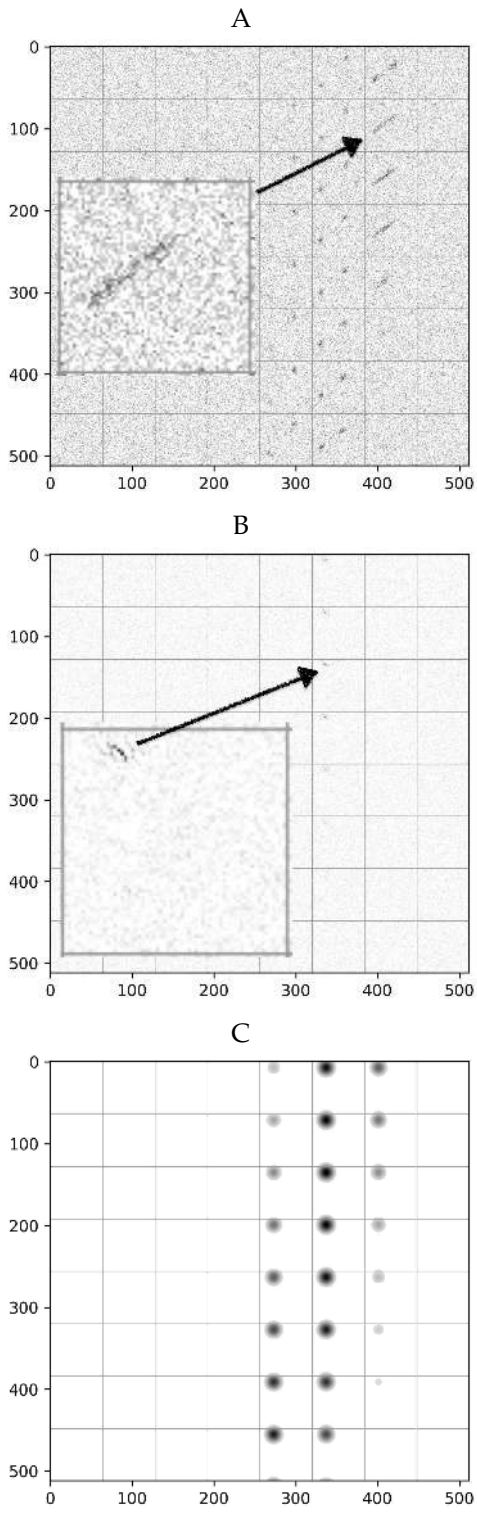
(2) Path. The target position trajectory could be recovered directly from unfiltered data. The signal is detectible not only in the denoised average as shown in panel A of Figure 2 but also directly in individual frames. The latter allowed direct calculation of maxima and a sample is provided in Table 3. The FT filtered results (Table 2) provide a much smoother trajectory. The denoised trajectory recovered is listed in Table 2 and can be compared to the noisy path. The denoised trajectory is plotted in Figure 5.

(3) Surveillance Aircraft. The aircraft should be sent to the x, y portion of the sub coordinates and times as listed in the respective columns of Table 2.

References

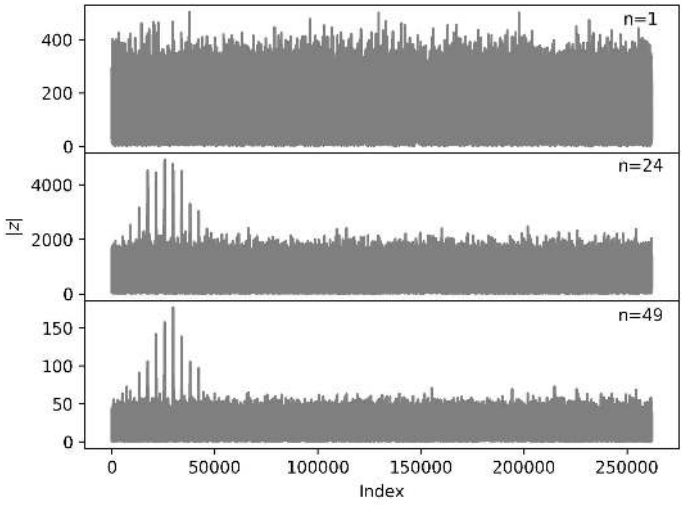
[Kutz, 2013] Kutz, J. Nathan (2013). Data-Driven Modeling & Scientific Computation: : Methods for Complex Systems & Big Data. *Oxford University Press, Inc. 198 Madison Ave. New York, NY, United States*, ISBN 978-0-19-966034-6.

Figure 2: *Flattened Averaged Signal & Filter Volumes*



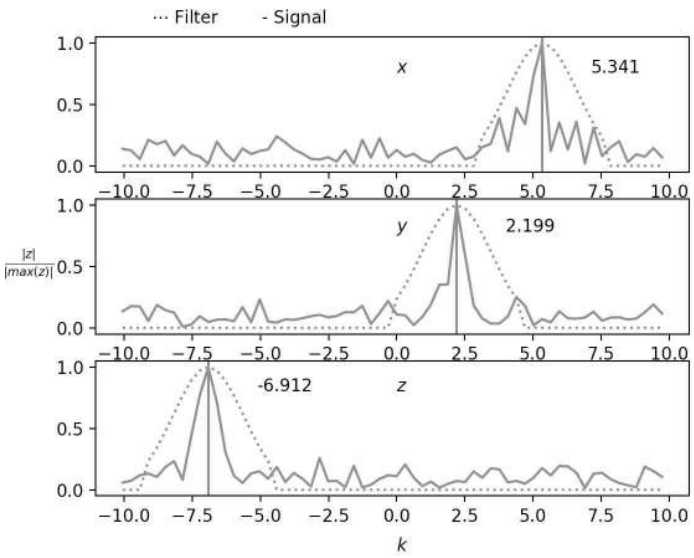
Images represent $3D$ volumes $64 \times 64 \times 64$ flattened by organizing the z dimension slices adjacent to one another in the plane $(64 \times 64) \times 8 \times 8$. (A) Averaged original signal. The elongated shape indicates movement in Cartesian space since the signals are summed. (B) Averaged Fourier transformed signal. The signal remains in the same general location in Fourier space. This corresponds to the bottom plot (at time step 49) in Figure 3. (C) Gaussian filter. The cells below the inset contain only noise. The center of the Gaussian is at the same k space position as the maximum of the FT signal in the center plot.

Figure 3: *Accumulating Sum Of Linearized FT Signal*



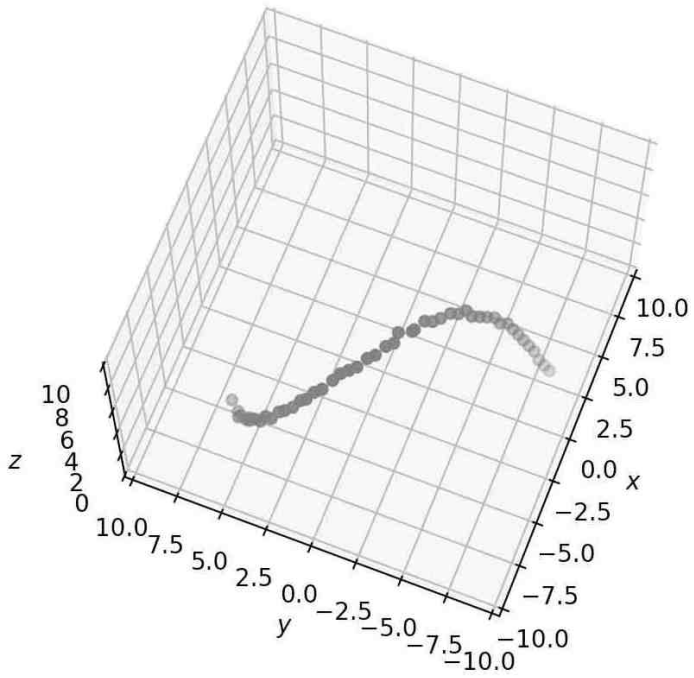
Accumulating sum of the linearized Fourier transform of the signal. We can see the signal to noise ratio increase as additional frames are added. This is because noise can be positive or negative with equal probability (zero centered) and therefore tends to average to zero. This is helpful only if the signal remains in the same space. Since we are in Fourier space here (frequency space AKA k space), the signal remains in the same position while it changes position in lab space.

Figure 4: *Acoustic Signature Of Submarine*



Acoustic signature of averaged normalized signal in k space. In the above plots, signal intensity is plotted along a line through the maximum parallel to each coordinate axis of shifted wavenumbers. Vertical lines are drawn at the maximum signal. The maximum signal is at $(k_x, k_y, k_z) = (5.341, 2.199, -6.912)$. The Gaussian filter applied is also plotted. Its maximum is placed at the same position as the signal maximum.

Figure 5: *Trajectory Of Submarine*



Trajectory recovered after application of filter and inverse transform. The center of each IFT signal is plotted with a grey dot. The z axis is depth, so the view is from the bottom up. The sub starts at the surface on the right side of the plot.

A. Python functions used

All below functions are for making plots. The first function provides cross-section lines through the maximum of a $3D$ volume. These volume crossing lines are parallel to the axis of the volume and can be used to generate $1D$ plots in each dimension. This was done to generate Figure 4

```
##===== return value lines (one parallel to each axis) through the maximum of F(x,y,z) =====##
def minMaxInfoScalarFuncOfVec(Mesh, F, verbose=False):
  if verbose: print('minMaxInfoScalarFuncOfVec')
  [X,Y,Z] = Mesh
  lmx = np.argmax(abs(F))
  argMx = np.unravel_index(lmx, F.shape)
  fx,fy,fz = F[argMx[0],:,argMx[2]], F[:,argMx[1],argMx[2]], F[argMx[0],argMx[1],:]
  gx,gy,gz = X[argMx[0],:,argMx[2]], Y[:,argMx[1],argMx[2]], Z[argMx[0],argMx[1],:]
  fMxLin = np.argmax(np.absolute(F))
  argMxF = np.unravel_index(fMxLin, F.shape)
  xIndMx = np.argmax(fx)
  yIndMx = np.argmax(fy)
  zIndMx = np.argmax(fz)
  if verbose: print('\t', xIndMx,yIndMx,zIndMx, fx[xIndMx], fy[yIndMx], fz[zIndMx], 'at', gx[xIndMx], gy[yIndMx], gz[zIndMx] )
  return [xIndMx,yIndMx,zIndMx,gx,gy,gz,fx,fy,fz]

##===== Plot flattened 3D image (into 2D plane) =====##
def flatImage(Z, thresh=0.2, title=None):
  mx = np.max(np.absolute(Z))
  print("image min=",np.min(np.absolute(Z)), "max=", mx, "avg=", np.average(np.absolute(Z)))
  if not thresh is None: Z[Z < thresh*mx] = 0.0
  Im = np.zeros((512,512))
  for i in range(8):
    for j in range(8):
      m = i + j*8
      Im[i*64:(i+1)*64,j*64:(j+1)*64] = np.absolute(Z[:,:,m])
      Im[:,j*64] = mx
      Im[i*64,:] = mx
  plt.imshow(Im)
  if not title is None: plt.title(title)
  plt.show()

##===== Plot 3 linearized, shared x  =====##
def plotSeriesLinearized(im1,im2,im3):
  fig, axs = plt.subplots(3, 1, sharex=True)
  fig.subplots_adjust(hspace=0)
  axs[0].plot(abs(im1.reshape(-1)))
  axs[1].plot(abs(im2.reshape(-1)))
  axs[2].plot(abs(im3.reshape(-1)))
  fig.text(0.85, 0.855, 'n=1', ha='center', va='center')
  fig.text(0.85, 0.600, 'n=24', ha='center', va='center')
  fig.text(0.85, 0.340, 'n=49', ha='center', va='center')
  axs[1].set(ylabel='$|z|$')
  axs[2].set_xlabel('Index')
  plt.show()
  plt.clf()

##===== Plot 3D trajectory =====##
def plot3dScatter(X):
  fig = plt.figure()
  ax = fig.add_subplot(111, projection='3d')
  a,b,c,d,e,f=-10,10,-10,10,0,10
  ax.axes.set_xlim3d(left=a, right=b)
  ax.axes.set_ylim3d(bottom=c, top=d)
  ax.axes.set_zlim3d(bottom=e, top=f)
  ax.set_xlabel('$x$')
  ax.set_ylabel('$y$')
  ax.set_zlabel('$z$')
  ax.scatter(X[:,0], X[:,1], X[:,2])
  plt.show()
```

B. Python codes

```
##===== Setup =====##
import os,sys,copy,numpy as np,scipy.io as sio,matplotlib.pyplot as plt
fft, fftshift, ifft = np.fft.fft, np.fft.fftshift, np.fft.ifft
abs, rndm, exp = np.absolute, np.random.normal, np.exp
np.set_printoptions(precision=3)
mat_contents = sio.loadmat('subdata.mat')
data = mat_contents['subdata']
L,n,pi = 10,64,np.pi
scale = (2*pi)/(2*L)
x2 = np.linspace(-L,L,n+1)
x = x2[0:n]
y,z = x,x
k = scale * np.append(np.arange(0,n/2),np.arange(-n/2,0))
ks = fftshift(k)
[X,Y,Z] = np.meshgrid(x,y,z)
[Kx,Ky,Kz] = np.meshgrid(k,k,k)
[Kxs,Kys,Kzs] = np.meshgrid(ks,ks,ks)
im1,im2 = None,None
ftImSeq = []
ftImAvg = np.zeros((n,n,n)).astype('complex')

##===== (III.A) Avg FT image =====##
for j in range(data.shape[1]):
  Un = data[:,j]
  ftIm = np.fft.fftn(Un.reshape(n,n,n)) # FFT 3D
  ftImAvg += ftIm
  ftImSeq.append(ftIm)
  if   j==0 :  im1 = copy.copy(ftImAvg)
  elif j==24 : im2 = copy.copy(ftImAvg)
ftImAvg = ftImAvg/len(ftImSeq)
ftImAvgS = fftshift(ftImAvg)

##===== (III.B) Find the signal (peak) in average FT image =====##
argMxLinS = np.argmax(abs(ftImAvgS))
argMxS = np.unravel_index(argMxLinS, ftImAvg.shape)
pkx,pky,pkz = ks[argMxS[1]], ks[argMxS[0]], ks[argMxS[2]]

##===== (III.C) Gaussian filter in k-space =====##
r = 0.1
filt = exp(-r*(Kx-pkx)**2 - r*(Ky-pky)**2 - r*(Kz-pkz)**2  )

##===== (III.D) Trajectory and LaTex table output =====##
mins, hours, trajectory = 0,0,[]
for j in range(len(ftImSeq)):
  Utn = ftImSeq[j]                # SELECT FT SIGNAL AT t=j
  UnFilt = np.multiply(Utn, filt) # APPLY FILTER
  U = np.fft.ifftn(UnFilt)        # INVERSE FFT 3D
  argMxLin = np.argmax(abs(U))    # PEAK = CENTER OF SUB
  argMx = np.unravel_index(argMxLin, U.shape)
  trajectory.append([ x[argMx[1]], y[argMx[0]], z[argMx[2]] ])
  szTexOut = "$%02d$ (%02d:%02d) & $%3.2f$ & $%3.2f$ & $%3.2f$ \\\\" % \
      (j+1, hours,mins, x[argMx[1]], y[argMx[0]], z[argMx[2]])
  if mins == 0: mins = 30
  else :
    mins = 0
    hours += 1
  print(szTexOut)

##===== plots & other output =====##
print('CENTER WAVENUMBERS', pkx, pky, pkz )
[xIndMxs,yIndMxs,zIndMxs,gxs,gys,gzs,fxs,fys,fzs] = \
  minMaxInfoScalarFuncOfVec([Kxs,Kys,Kzs], abs(ftImAvgS)/np.max(abs(ftImAvgS)))
flatImage(filt)
plotSeriesLinearized(im1,im2,ftImAvg)
plot3dScatter(np.array(trajectory))
```

C. Fourier Series & Fourier Transform

i. Fourier series in standard and complex exponential form

Fourier series in standard notation (not using complex exponential):

$$f(x) \sim \sum_{n=0}^{\infty} a_n \cos\left(\frac{n\pi}{L}x\right) + b_n \sin\left(\frac{n\pi}{L}x\right) \tag{13}$$

To get the series in complex exponential form using Euler's formula, we must extend the sum from (0 to ∞) to ($-\infty$ to ∞). Since cos is even ($\cos\left(\frac{-n\pi}{L}x\right) = \cos\left(\frac{n\pi}{L}x\right)$) we get double all basis waves. So the coefficient must be halved. The complex Fourier series definition:

$$f(x) \sim \sum_{n=-\infty}^{\infty} \mathbf{c}_n e^{-in\pi x/L}, \qquad -L < x < L \tag{14}$$

The complex coefficients $\mathbf{c}_n = a_n + ib_n$ are found by solving the integral

$$\mathbf{c}_n = \frac{1}{2L}\int_{-L}^{L} f(x) e^{in\pi x/L} dx = \frac{1}{2L}\int_{-L}^{L} f(x)\left[\cos\left(\frac{n\pi x}{L}\right) + i\sin\left(\frac{n\pi x}{L}\right)\right] dx \tag{15}$$

Where the RHS is obtained by substituting Eulers formula $e^{ix} = \cos(x) + i\sin(x)$. For the real Fourier series, we sum instead from 0 to ∞ to get $f(x) \sim \sum_{n=0}^{\infty} A_n \cos\left(\frac{n\pi x}{L}\right) + B_n \sin\left(\frac{n\pi x}{L}\right)$ and this leads to a factor of $\frac{1}{2}$ so we have half the sum. In addition, in the complex series, $A_n = A_{-n}$ and $B_{-n} = -B_n$. If we start the summation of the real series at $n = -\infty$ then we need the same factor of $\frac{1}{2}$ in front of the integrals and definition of B_{-n}. In the latter case, and we do not need the 1/2 when converting $\mathbf{c}_n = A_n + iB_n$.

$$A_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\left(\frac{n\pi x}{L}\right) \qquad B_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\left(\frac{n\pi x}{L}\right) \tag{16}$$

$$\mathbf{c}_n = \begin{cases} \frac{1}{2}(A_n + iB_n) & n \geq 0 \\ \frac{1}{2}(A_{-n} - iB_{-n}) & n < 0 \end{cases}$$

ii. Derivation of Fourier transform from complex exponential series

Coefficient $\mathbf{c}_n = 1/2L\int_{-\infty}^{\infty} f(x)e^{-ik_n}dx$ (eqn. 3) equals $\frac{1}{2L}F(k_n)$ by equation 7. Substituting back for cn into the series (eqn. 14)

$$f(x) = \sum_{-\infty}^{\infty}\left(\frac{1}{2L}F(k_n)\right)e^{ik_n x} \tag{17}$$

$$= \sum_{-\infty}^{\infty}\left(\frac{1}{2\pi}\cdot\frac{\pi}{L}F(k_n)\right)e^{ik_n x} \tag{18}$$

$$= \frac{1}{2\pi}\sum_{-\infty}^{\infty} F(k_n)e^{ik_n}\Delta k \qquad (i) \tag{19}$$

$$= \int_{-\infty}^{\infty} F(k)e^{ik}dk \qquad (ii) \tag{20}$$

(i) In section II D we had $\frac{\pi}{L} = \Delta k$.
(ii) In limit $\Delta k \to 0$ we get the Riemann integral.