AMATH 582 Homework One: Submarine Hunt

Daniel W. Crews

January 27, 2021

Abstract

Signal processing is used ubiquitously in science and engineering. This simple numerical experiment explores an important aspect of signal analysis, namely statistical averaging of a spectral time series to eliminate noise and thereby identify center frequencies. Here the time series consists of a spatial wavefield which is denoised via Gaussian filtering about the identified frequency to determine a trajectory. While the technique explored in this report is quite elementary compared to more sophisticated signal processing approaches, it conveys the essential elements of, for example, a radar or sonar tracking system.

1 Introduction

The premise of this analysis is to track the location of a submarine given a noisy spatial wavefield constructed somehow through acoustic measurements. Given that noise is random, it may be supposed (for instance via the central limit theorem) that it is normally distributed with zero mean and some variance. Therefore it is expected to vanish upon time average. As the submarine is moving, its location also smears over time average. Yet the submarine's spatial frequency is an analytic signature and remains upon time average, provided the average is over an interval small enough compared to the change in the submarine's shape or sound. This analysis assumes the target's signature to not change with time in the frequency domain.

This report describes an algorithm used to denoise the signal via a statistical method and to identify the submarine's trajectory. It begins with a brief theoretical overview of the methods used, and proceeds to a discussion of the details of the code used to implement the methods. There is then a discussion of results identifying the three-dimensional trajectory, along with a suggested search area for a submarine-tracking aircraft and the likely frequency of the submarine. The report is kept brief without sacrificing clarity.

2 Theoretical Background

2.1 Spectral analysis, discrete Fourier series, and the FFT

The Fourier series is an eigenfunction expansion of an integrable function $f(x)$ (or formally, satisfying the Dirichlet conditions) on a finite interval $x \in D$ of length $|D| = L$ into the series

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{ik_n x}, \qquad k_n = \frac{2\pi}{L} n \tag{1}$$

where the wavenumbers k_n are integer multiples of the fundamental domain harmonic $k_1 = \frac{2\pi}{L}$ and the coefficients c_n are given by the usual Fourier integrals [1]. Given functional values f_n on equally-spaced points $\{x_n\}|_{n=0}^{N-1}$, the "discrete Fourier transform" is a quadrature approximation of the Fourier integral,

$$\hat{f}_m = \sum_{n=0}^{N-1} f_n e^{-ik_m x_n}, \qquad f_n = \frac{1}{N}\sum_{m=0}^{N-1} \hat{f}_m e^{ik_m x_n} \tag{2}$$

The requirement that the points be equally spaced is quite important. The reason is revealed by supposing the functional values f_n to interpolate a piecewise constant function $\widetilde{f}(x)$ on the interval D, with each subinterval centered about each point x_n and with width Δx_n. The Fourier integral of $\widetilde{f}(x)$ yields coefficients

$$c_n = \frac{1}{L}\sum_{m=0}^{N-1} \frac{\Delta x_m}{2}\mathrm{sinc}\Big(\frac{\Delta x_m}{2}k_n\Big)e^{-ik_n x_m} f_m \tag{3}$$

where the cardinal sine function $\mathrm{sinc}(x) = \sin(x)/x$ is the spectral signature of the interval Δx_m. If the interval widths Δx_m are equal, then the sinc function factors out of the series and the expression may be inverted. Inversion takes place on account of the discrete orthogonality property of the circle functions,

$$\frac{\pi}{n}\sum_{\alpha=-n}^{n} \sin\Big(k\frac{\pi}{n}\alpha\Big)\sin\Big(m\frac{\pi}{n}\alpha\Big) = \delta_{nm}, \tag{4}$$

a property noted by at least 1759 by Lagrange, though he did not take the last steps to the full Fourier series as the appropriate limiting notions were not fully understood in those times [2].

This procedure results in a discrete transform pair provided that the intervals are equally spaced. If the spacings are not equal, each interval's spectral signature is tangled up with its phase information and such a simple discrete inversion is not possible. In such a case, one must interpolate onto equally-spaced intervals. Note that Eqn. 3 is the zeroth-order interpolant transform. The spectral signature of *e.g.* higher-order Legendre polynomial interpolants are the spherical Bessel functions, of which sinc is the first member.

The fast Fourier transform (FFT) is an algorithm to compute the discrete Fourier transform of Eqn. 2 on a set of N equally-spaced points in $\mathcal{O}(N\log(N))$ operations. If the set consists of $N = 2^M$ points then even more efficient transformation is possible. In Cartesian coordinates the multidimensional discrete Fourier transform is obtained by that of each dimension independently, as the kernel $e^{i\boldsymbol{k}\cdot\boldsymbol{x}}$ factorizes the Fourier integral into one-dimensional transforms[1]. In this analysis the three-dimensional FFT is computed using Numpy's np.fft.fftn() routine, and inverse transform via np.fft.ifftn().

2.2 Separation of mean and fluctuations

A goal of science and engineering is to separate the dynamics of a mean value from that of its fluctuations. The idea is that an interpretable signal X consists of an orderly mean part $\langle X\rangle$ which follows some understandable rule of evolution, and a fluctuating part δX whose precise nature is unknown (or even unknowable, if the intrinsic probability suggested by quantum mechanics is indeed true) [3]. The variable X is decomposed by

$$X = \langle X\rangle + \delta X. \tag{5}$$

It is important to note that the fluctuations δX may influence the observable evolution $\langle X\rangle$. Yet it is often the case that fluctuations are entirely independent of the mean dynamics. For instance, acoustic fluctuations in the Puget Sound due to wind and distant whale song do not change the sound of a submarine there. In such a case the goal is to eliminate the noise, and by the very definition of the splitting in Eqn. 5, averaging of the signal X over an appropriate window of samples will substantially reduce the presence of δX [4]. By this method all coherent sources will be determined and the fluctuations suppressed.

2.3 The exponential filter

The averaging procedure obscures information within the sampling window. Yet once the coherent source is identified through averaging, one may then filter the original data within the window to denoise and isolate the source. A common filter utilized is the exponential one in d-dimensions,

$$\mathcal{F}(\boldsymbol{k};\boldsymbol{k}_0) = \exp\Big(-\tau\sum_{i=0}^{d-1}(k_i - k_{0,i})^\sigma\Big) \tag{6}$$

where $\boldsymbol{k}_0$ is the center (vector) frequency, τ is a scaling (bandwidth) parameter, and σ an even parameter giving the filter strength. The Gaussian filter is utilized in this analysis, corresponding to $\tau = 0.5$ and $\sigma = 2$.

[1]Note that for more specialized geometries such as polar coordinates one may define discrete versions of the relevant Fourier transforms, e.g. the Hankel transform and its discrete counterpart. It is intriguing to observe that in polar coordinates the Fourier series arises even for infinite domains as the angular direction is naturally periodic.

3 Analysis and discussion

The data provided for analysis is supposed to be a spatial waveform determined through acoustic measurements, and consists of complex amplitudes on an evenly-spaced $N^3 = 64^3$ Cartesian grid discretizing a domain of $L = 10$ spatial units, with 49 time measurements taken in half-hour intervals. As an objective is to identify the frequency of the submarine, the averaging described in Section 2.2 is applied to the Fourier transform of the given data. By determining the submarine frequency, inversion of the filtered spectrum is used to detect the submarine's location for each of the 49 recorded half-hour steps.

3.1 Algorithm schematic

The following list gives a simple overview of the steps used for trajectory identification, along with the associated Numpy functions for implementation of the FFT:

1. **Initialize:** The data is read in ($n = 64$), and the spatial and spectral grids initialized;
   - The evenly-spaced spatial grid via `x=np.linspace(-L, L, num=n+1)`, and the corresponding wavenumbers (scaled to angular frequency by $\times 2\pi$, as the Numpy FFT returns frequencies in Hertz) using `np.fft.fftfreq` (see App. functions for details).
2. **Transform, time average:** The spectrum is found via FFT for each timestep. It is then averaged over a window of seven time samples, or three and a half hours (chosen as the square-root of 49; it was found to work in this case).
3. **Find center frequency:** The center frequency in each averaged time window is found by computing the mean wavenumber of all points greater than a 70% detection threshold.
4. **Spectral filter:** For each time t, the spectrum is denoised by applying a Gaussian filter (Eqn. 6 with $\tau = 0.5$ and $\sigma = 2$) about the center frequency k_0 determined in Step 3 for that window $t \in |T_i|$.
5. **Inverse transform and trajectory:** The denoised spectrum is transformed back to the spatial domain, and the mean location of all points above an 80% detection threshold is considered to be the submarine's location.

Note that the detection threshold is set by the operator, according to the degree of various signals persisting after time averaging. The 70% and 80% detection thresholds were seen to work quite well. For further information, Appendix A details some Python functions while Appendix B contains code implementation. Github repository for this work is available here.

3.2 Analysis results

Initialization according to Step 1 in the schematic allowed visualization of the dataset via an isosurface, as in Fig. 1. The signal is clearly visible to the eye in this plot, and was confirmed as a real object by time-averaging of the frequency spectrum over a set of seven time measurements. Time-averaging of the frequency is critical, as the object is moving so that its average position is not constant. However, the object's spectral signature is unchanged as it did not change shape (or did not change acoustic signature, as the nature of the data provided is obscure). Table 1 shows centers found by time averaging.

Time [hr]	0-3.5	3.5-7.0	7.0-10.5	10.5-14.0	14.0-17.5	17.5-21.0	21.0-24.5	All-time average
$\langle k_x \rangle_T$ [1/L]	0.814	0.661	0.784	0.760	0.810	0.728	0.802	0.766
$\langle k_y \rangle_T$ [1/L]	0.271	0.292	0.331	0.240	0.341	0.267	0.271	0.288
$\langle k_z \rangle_T$ [1/L]	-1.00	-1.03	-1.09	-1.06	-1.13	-1.07	-1.1125	-1.07

Table 1: Observed center frequencies of data following spectral averaging over time intervals of 3.5 hours, with seven samples per interval, then used as the filter frequencies k_0 in the Gaussian filter. The arbitrary spatial unit is given as L, not to be confused with domain length. The all-time average is given as well.

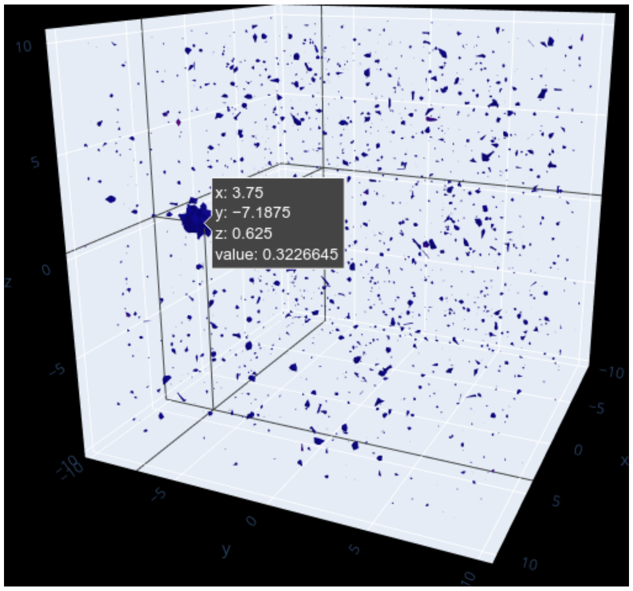
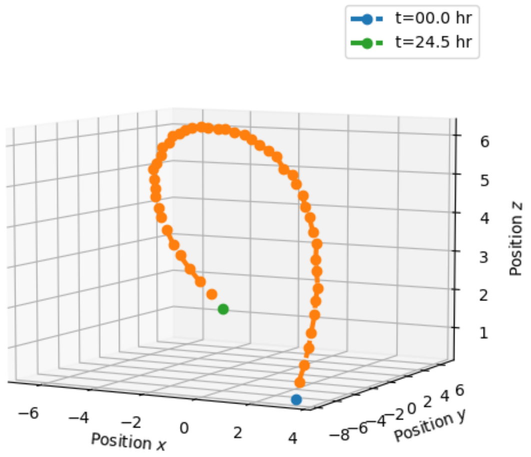


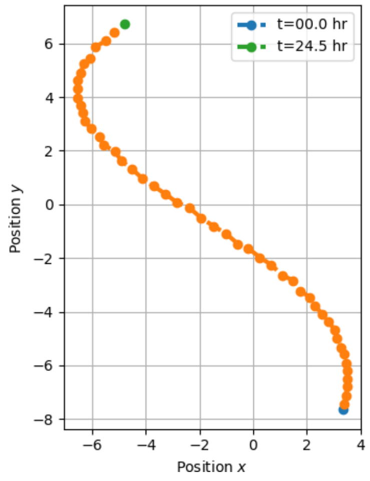
Figure 1: Isosurface at 50% maximum of acoustic data $|\psi(x, t = 0)|/|\psi|_\infty$, revealing a noisy background field and an apparent signal with coordinates highlighted. Upon time-averaging in the frequency domain, the signal was confirmed to be coherent, i.e. have non-zero mean.

Note that in Table 1 negative wavenumbers are used as the given spatial data is complex, meaning that the reality condition is not satisfied, instead $\hat{f}(-k) = \hat{f}^*(k)$ in this dataset. The absolute value is reported as "the frequency" of the submarine, however. This identifies the submarine's spectral signature as $\boldsymbol{k} \approx \{0.766 \pm 0.05, 0.288 \pm 0.03, 1.07 \pm 0.04\}$ [1/L] by mean and standard deviation of the window averages of Table 1, where L is an arbitrary space unit corresponding to that of the provided data, *not* the domain length. The width of the space or frequency "submarine Gaussian" was not measured for ease of analysis.

Having computed the center frequencies, the spectrum was filtered and the trajectory determined according to the schematic Steps 4 and 5. The resulting 3D trajectory and top-down position given in Figs. 2a, 2b respectively. This suggests the submarine is currently located around $\boldsymbol{x} \sim (-5, 6.5)$ in (x, y).



(a) Three-dimensional trajectory of submarine, showing a rise and dive maneuver.



(b) Top-down view of predicted trajectory. Ideal search area for submarine-tracking aircraft is around $\boldsymbol{x} \sim (-5, 6.5)$ and continuing north-east.

4 Summary and Conclusions

This numerical experiment combined Fourier spectral methods with statistical methods to analyze what is effectively sonar data by an elementary technique. Noise was eliminated by averaging of the frequency spectrum of a moving target, when averaging of the spatial position would be smeared away by motion. The trajectory was determined satisfactorily and the frequency signature of the object determined for future reference and classification. Finally, Python was used for implementation of this exercise, though Matlab may be used in the future if the required analysis is simpler that way.

References

[1] C. Lanczos. *Applied Analysis.* Dover, New York, 1958.

[2] C. Lanczos. *Discourse on Fourier Series.* SIAM Classics in Applied Mathematics, 2016.

[3] N.G. van Kampen. *Stochastic processes in physics and chemistry.* Elsevier, 3rd edition, 2007.

[4] J.N. Kutz. *Data-driven Modeling and Scientific Computation.* Oxford University Press, 1st edition, 2013.

Appendix A Python Functions

The following list compiles important Python functions used in implementation:

- `np.linspace(x1,x2,num=n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `np.meshgrid(x,y)` returns 2-D grid coordinates based on `x` and `y`, with 3-D available too.
- `np.fft.fftn(x)` computes n-dimensional Fourier transform of gridded data while `np.fft.ifftn(x)` does the inverse transform and `np.fft.fftfreq(n, d=dx)` computes the frequencies for a dataset of `n` points and sample spacing `dx` *in Hz*.
- `np.where(X > a)` determines a Boolean indexing array wherever the array `X` is greater than value `a`. Any Boolean statement may be put into the input. This is a quite useful function.

Appendix B Python Implementation

```
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import csv

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

### Read data
# Replace i by j (only need to run once)
# f_raw = open("subdata.csv", "r")
# f_raw = ''.join([i for i in f_raw]).replace('i', 'j')
# f_new = open("subdata2.csv", "w")
# f_new.writelines(f_raw)
# f_new.close()

# Read complex data
subdata = np.genfromtxt('subdata2.csv', delimiter = ',', dtype = np.complex128)
```

```
### Domain parameters
L = 10
n = 64
# Spatial domain
x2 = np.linspace(-L, L, num=n+1)
x = x2[1:n+1]
y = x
z = x
# Wavenumbers
k = 2.0*np.pi*np.fft.fftfreq(n, d=x[1]-x[0])
#ks = np.fft.fftshift(k)

# Grids
X, Y, Z = np.meshgrid(x, y, z)
#KSX, KSY, KSZ = np.meshgrid(ks, ks, ks)
KX, KY, KZ = np.meshgrid(k, k, k)

### Look at unfiltered data
sd = np.reshape(subdata[:, 0], (n,n,n))
M = np.amax(np.abs(sd))
vol = np.abs(sd)/M

### Isosurface plot at 50%
fig = go.Figure(data=go.Isosurface(
    x=X.flatten(),
    y=Y.flatten(),
    z=Z.flatten(),
    value=vol.flatten(),
    isomin=0.5,
    isomax=1,
    ))

fig.show()

### Average spectrum and determine center frequency
# Form seven step averages
avg_spec = np.zeros((n, n, n, 7))+0j
for i in range(7):
    for j in range(7):
        Unt = np.fft.fftn(np.reshape(subdata[:, 7*i + j], (n,n,n)))
        avg_spec[:,:,:,i] += Unt/7

# Center frequency = mean of where spectral density exceeds threshold (60% max)
center = np.zeros((3, 7))
for i in range(7):
    M = np.amax(np.abs(avg_spec[:,:,:,i]))
    idx = np.where(np.abs(avg_spec[:,:,:,i])/M > 0.7)
    center[0, i] = np.mean(KX[idx])
    center[1, i] = np.mean(KY[idx])
    center[2, i] = np.mean(KZ[idx])
    print(center[:,i]/(2.0*np.pi))

# For simplicity, define center as average of these window centers
```

```
c_k = np.mean(center, axis=1)/(2.0*np.pi)
c_l = np.std(center, axis=1)/(2.0*np.pi)
print('The all-time average center frequency is ' + str(c_k) + ' [1/m]')
print('The all-time st.d. is ' + str(c_l) + ' [1/m]')

### Filter and inverse transform
c_x = np.zeros((3, 49))
for i in range(7*7):
    # Discrete transform
    Unt = np.fft.fftn(np.reshape(subdata[:, i], (n,n,n)))
    # Denoise with gaussian filter
    s = 0.5 # filter strength
    mid = center[:, i//7]
    flt = np.exp(-s*((KX-mid[0])**2.0 + (KY-mid[1])**2.0 + (KZ-mid[2])**2.0))
    Unft = np.multiply(Unt, flt)
    # Reverse transform
    Unf = np.fft.ifftn(Unft)

    # Find center above threshold (80% max)
    idx = np.where(np.abs(Unf)/np.amax(np.abs(Unf)) > 0.8)
    c_x[0,i] = np.mean(X[idx])
    c_x[1,i] = np.mean(Y[idx])
    c_x[2,i] = np.mean(Z[idx])
    #print(c_x[:,i])


### 3D trajectory
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(c_x[0,0], c_x[1,0], c_x[2,0], 'o--', linewidth=3, label='t=00.0 hr')
ax.plot(c_x[0,1:-1], c_x[1,1:-1], c_x[2,1:-1], 'o--', linewidth=3)
ax.plot(c_x[0,-1], c_x[1,-1], c_x[2,-1], 'o--', linewidth=3, label='t=24.5 hr')
plt.legend(loc='best')
plt.grid(True)
ax.set_xlabel(r'Position $x$')
ax.set_ylabel(r'Position $y$')
ax.set_zlabel(r'Position $z$')
plt.tight_layout()

### 2D trajectory
plt.figure()
plt.plot(c_x[0,0], c_x[1,0], 'o--', linewidth=3, label='t=00.0 hr')
plt.plot(c_x[0,1:-1], c_x[1,1:-1], 'o--', linewidth=3)
plt.plot(c_x[0,-1], c_x[1,-1], 'o--', linewidth=3, label='t=24.5 hr')
plt.grid(True)
plt.legend(loc='best')
plt.xlabel(r'Position $x$')
plt.ylabel(r'Position $y$')
plt.gca().set_aspect('equal', adjustable='box')
plt.tight_layout()

plt.show()
```