Submarine Tracking Through Signal Processing

Elliot Jennis

January 2021

Abstract

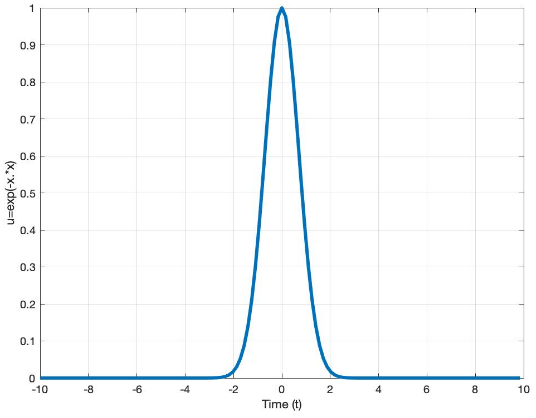
The Fourier transform is a powerful tool for interpreting and manipulating data. In this paper the Fourier transform, among other signal processing techniques, is used to identify the path of a submarine from the noise filled data collected by a sonar station. The practical application of this mathematical tool is an algorithm known as the Fast Fourier transform. This algorithm was used to transform the sonar data and then average it in the signal domain to reduce the noise. This averaging process enabled the identification of the submarine's signature frequency. A Gaussian filter was then applied at the location of the submarine's signal in frequency space and the filtered data was transformed back into the time domain. With the clear path of the submarine now in view, the final location of the craft was determined and the information relayed to nearby anti-submarine warfare aircraft.

Introduction and Overview

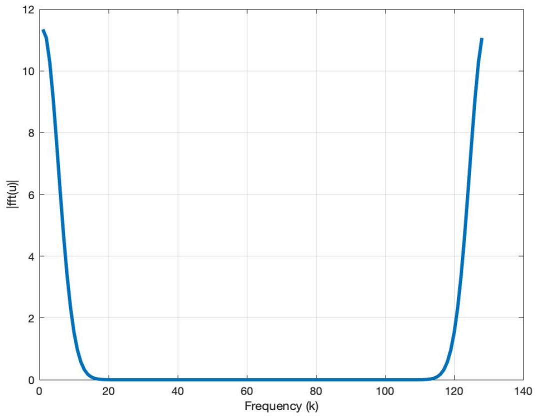
The first step into the world of signal processing and analysis must include a strong understanding of the Fourier transform (FT). Both its effectiveness as a data processing tool and its inherent drawbacks are best understood though applying the FT to solve a real world problem. A sonar station has collected noisy data over the course of a 24-hour time frame with data collection occurring every half hour. The primary task is to apply signal processing techniques that use FT's and other algorithmic tools to identify a specific frequency of unknown value within the noise filled data set. Once the signal has been identified, the data must be filtered to determine the x and y coordinates of the signal in the time domain. The final coordinates of the signal are the deliverable for this problem. A basic understanding of denoising algorithms and signal domain filters is required to successfully complete this task.

Theoretical Background

Understanding the implementation of the FT requires prior knowledge regarding manipulating and expanding functions. The french mathematician Joseph Fourier (whom the FT is named after) developed a method of representing functions as a series expansion of cosines



(a) Original Gaussian function



(b) Result of FFT command without any manipulation

Figure 1: Example of how the FFT algorithm can shift data in the signal domain.

and sines[1] (equation 1).

$$f(x) = \frac{a_0}{2} + \sum_{i=1}^{\infty} (a_n \cos nx + b_n \sin nx) \quad x \in (-\pi, \pi]. \tag{1}$$

Through further manipulation and the use of orthogonality properties, equations for a_n and b_n (also known as Fourier coefficients) can be explicitly defined[1]. The expansion can also be written in complex form along the real domain $x \in [-L, L]$ (equation 2) with $f(x)$ still being defined as a real function[1].

$$f(x) = \sum_{-\infty}^{\infty} c_n e^{in\pi x/L} \quad x \in [-L, L] \tag{2}$$

The Fourier coefficients can themselves be written in complex form (equation 3) along the $[-L, L]$ domain by applying Euler's formula[1]. With the addition of the complex kernel, the following can be concluded about the value of the Fourier coefficients: when the function $f(x)$ is even the Fourier coefficients are real, when it is odd the Fourier coefficients are imaginary[1].

$$c_n = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-in\pi x/L} dx \tag{3}$$

The FT is an integral transform that moves a function from the time domain t to its equivalent in the frequency domain k (equation 4)[1].

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \tag{4}$$

It can be inverted to bring a function from the frequency domain back into the time domain (equation 5)[1]. These two equations (equations 4 & 5) can be redefined on along $[-L, L]$

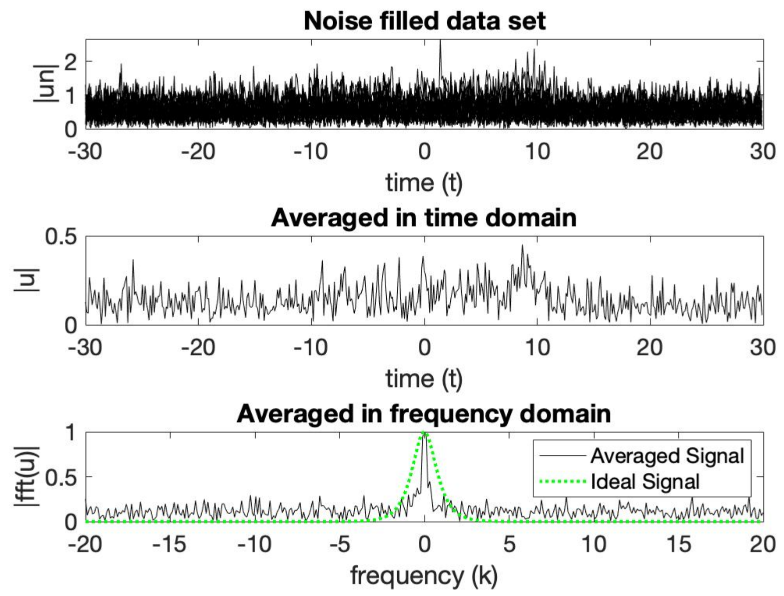


Figure 2: Graphical comparisons of using averaging technique on noisy data with a moving signal.

allowing it to be discretized for algorithmic purposes.

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k) dk \tag{5}$$

Challenging operations in the time domain can become trivial when applied in the frequency domain which is useful in the field of signal processing and analysis. This is the power of the FT. It is not without drawbacks however. Much like the Heisenberg uncertainty principle in quantum mechanics, there exists an uncertainty principle for the FT. Having a high degree of resolution in the time domain, equates to very low resolution in the frequency domain[1]. This fact must be taken into account when analyzing data sets.

Algorithm Implementation and Development

There are multiple algorithms in use within the proposed solution to this problem that require further explanation. To begin, the overall structure of solution must be articulated:

1. Reshape data into $n*n*n*R$ sized matrix with n being the number of Fourier modes and R being the number of realizations
2. Average the data to find the signature frequency
3. Filter the data at the signature frequency
4. Find the max value for each realization and the corresponding location in xyz coordinates of that value in the time domain.

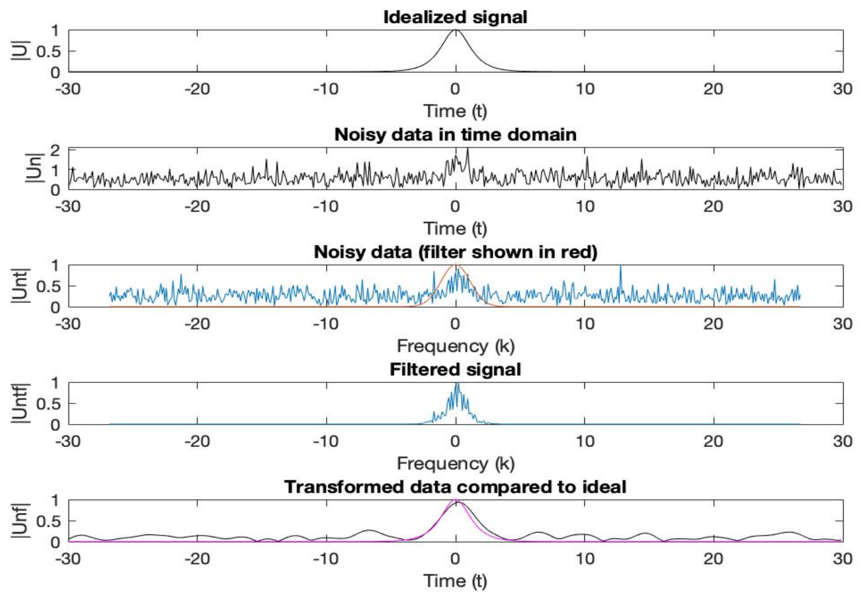
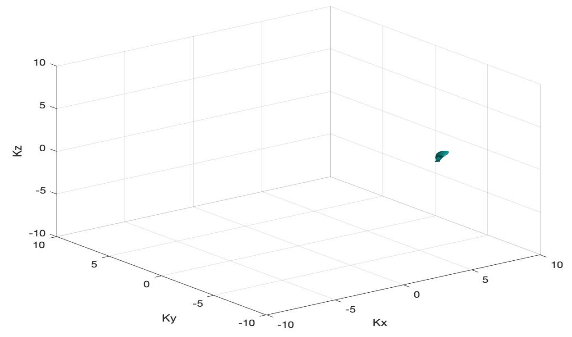


Figure 3: Effectiveness of using a Gaussian filter on noisy data.

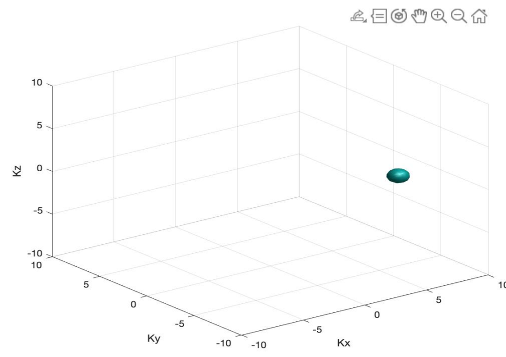
5. Plot the resulting data

The reshaped data is presented as a a $64*64*64*49$ matrix, 64 being the number of Fourier modes in three-dimensional space and 49 being the number of times data was collected corresponding to a collection period of every half hour over a 24 hour time frame (49 realizations). Using for-loops, the data can be split into 49 realizations of data with each realization being Fourier transformed and then added together. This final result is then averaged by dividing by the total number of realizations (49) revealing a signal located in frequency space. The coordinates in frequency space can then be coded into a three dimensional Gaussian filter with each value replacing k_0 for the respective coordinate frame. This filter is then multiplied with each realization of data in frequency space (not averaged) and then transformed back into the time domain. The maximum value of each realization is computed as well as the index of that value. A corresponding location in x, y, z space is determined using that index. To accomplish all of the above, individual algorithms must be explored in further detail.

The first algorithm of note is the Fast Fourier Transform (FFT). This algorithm was developed to computationally determine the Fourier transform of a given input. It has a very low operation count when compared to its contemporaries. The count goes like $O(NlogN)$. For large values of N, the operation count can be treated a linearly increasing value due to the extremely slow growth of $logN$. This low operating count is due in part to discretizing the broad range of $[-L, L]$ into 2^n points (also called Fourier modes) which must be clearly defined within any code using the FFT. Additionally, the FFT assumes solutions are periodic (within the domain $k \in [-\pi, \pi]$) due to the oscillatory behavior of the complex kernel. Code using the FFT must re-scale their wave numbers to reflect this assumption. Lastly, the FFT shifts data and multiplies every other mode by -1 (Figure 1). A shifting algorithm known as FFTshift (Appendix A) in MATLAB syntax as well as the absolute value function should



(a) Signature frequency



(b) Gaussian filter

Figure 4: Locations of both the signature frequency and the constructed filter in the frequency domain.

be applied to any data set brought into the frequency domain to correct for this.

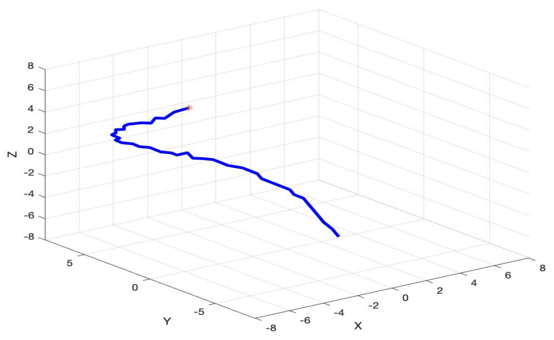
The next algorithm focuses on averaging to clean up the noise within a data set. Fundamentally, it splits the data into multiple slices of time or 'realizations'. The data in each realization is added on top of each other and then divided by the total number of realizations. Spikes in the averaged data indicate a signal. This technique is quite powerful and can often yield results within only a few realizations. However, there are some limitations to the use of this averaging technique. Noise present in the data must be white noise with zero mean, otherwise when the data is averaged the noise will not disappear, in fact it may amplify. Next, the signal must have a constant frequency in time. One problem with averaging data in the time domain is that a moving signal will not yield a clear result (Figure 2). Similarly when in the frequency domain, the signal must stay put at a specific value to yield a meaningful result.

The use of filtering is also required to solve the problem. Filters vary in type, but their general purpose is to attenuate frequencies above and/or below a certain central frequency. This is a very useful technique when trying to find a *known* signal within a noisy data set (Figure 3). In this application, a Gaussian filter of the form $F(k) = exp^{-\tau(k-k_0)^2}$ with τ being the bandwidth of the filter and k_0 being the location in frequency space of the signal[1]. Once transformed back to the time domain, the target path will emerge.

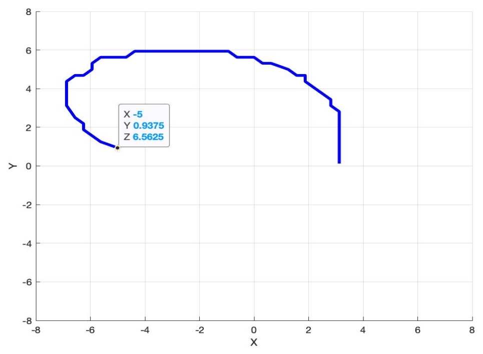
Computational Results

Signature Frequency

The averaging algorithm was used to reduce the white noise within the data set. This algorithm was put into place (Appendix B: Listing 2) to determine the location of the signature frequency of the submersible in the frequency domain. The averaged data is shown in Figure 4a. The location of this signature was determined graphically to be at $(5, -7, 2)$ in (Kx, Ky, Kz) space.



(a) 3D path visualization



(b) XY visualization

Figure 5: Final path constructed from the filtered data set indicating the location of the submarine over time.

Filter

With the location of the signal determined, a three dimensional Gaussian filter was constructed centered at the signature frequency with $\tau = 0.6$ (Figure 4b). The filter was applied to the transformed data and then inverseFFt'ed to return the newly filtered data sets to the time domain (Appendix B: Listing 3).

Time Domain Tracking

To find the path of the submarine from the filtered data, the max value for each realization was found and indexed. The location in XYZ space was found using the index of the max value (Appendix B: Listing 4). This value represents the strongest signal within each filtered data set indicating the approximate location of the submarine. This process was repeated for all 49 realizations and the results were plotted to form the path of the submarine (Figure 5a). Following the curve to its terminus at realization 49, the coordinates of the submarines last known location are shown to be at $X = 5, Y = 0.9375$ (Figure 5b).

Summary and Conclusions

Using the FFT algorithm helped determine the location of the submarine given very noisy data. While the signal itself was moving in the time domain, it was not moving in the frequency domain. This fact allowed for the simple averaging technique to reduce the white noise inherent in the data. It also justified the use of the FFT as time resolution was not a primary concern for this particular application. With a known frequency and its location determined from averaging, a stationary Gaussian filter was applied to view the data in the time domain. The filtered data was used to determine the path of the submarine.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.

A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[X,Y] = meshgrid(x,y)` returns 2-D grid coordinates based on the coordinates contained in the vectors `x` and `y`. X is a matrix where each row is a copy of `x`, and `Y` is a matrix where each column is a copy of `y`. The grid represented by the coordinates `X` and `Y` has `length(y)` rows and `length(x)` columns
- `X = zeros(sz1,...,szN)` returns an `sz1-by-...-by-szN` array of zeros where `sz1,...,szN` indicate the size of each dimension. For example, `zeros(2,3)` returns a 2-by-3 matrix.
- `B = reshape(A,sz1,...,szN)` reshapes A into a `sz1-by-...-by-szN` array where `sz1,...,szN` indicates the size of each dimension. You can specify a single dimension size of `[]` to have the dimension size automatically calculated, such that the number of elements in B matches the number of elements in A. For example, if A is a 10-by-10 matrix, then `reshape(A,2,2,[])` reshapes the 100 elements of A into a 2-by-2-by-25 array.
- `Y = fftn(X)` returns the multidimensional Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D transform is equivalent to computing the 1-D transform along each dimension of X. The output Y is the same size as X.
- `Y = fftshift(X)` rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array.
- `M = max(A,[],'all')` finds the maximum over all elements of A. This syntax is valid for `MATLAB®` versions R2018b and later.
- `fv = isosurface(X,Y,Z,V,isovalue)` computes isosurface data from the volume data V at the isosurface value specified in isovalue. That is, the isosurface connects points that have the specified value much the way contour lines connect points of equal elevation. The arrays X, Y, and Z represent a Cartesian, axis-aligned grid. V contains the corresponding values at these grid points. The coordinate arrays `(X, Y, and Z)` must be monotonic and conform to the format produced by meshgrid. V must be a 3D volume array of the same size as X, Y, and Z.
- `plot3(X,Y,Z,LineSpec)` creates the plot using the specified line style, marker, and color.

B MATLAB Code

```
clc;clear all; close all;
load subdata.mat; % Imports the data as a 262144x49 (space by time) matrix called subd
%% Initialzing data resolution and relevent variables

L = 10; % spatial domain
n = 64; % Fourier modes
x2 = linspace(-L,L,n+1); x = x2(1:n); y =x; z = x;
k = (2*pi/(2*L))*[0:(n/2 - 1) -n/2:-1]; ks = fftshift(k);

[X,Y,Z]=meshgrid(x,y,z);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);
Utave = zeros(n,n,n);
```

Listing 1: Initialization of variables and Fourier modes

```
%% Averaging Algorithm

for j=1:49

Un(:,:,:)=reshape(subdata(:,j),n,n,n); %spacial matrix
Ut = fftn(Un); %transform of data
Uf = fftshift(Ut);  %shifted data
Utave = Utave + Uf;  %averaging to remove noise

end
Mave = max(abs(Utave),[],'all');

isosurface(Kx,Ky,Kz,abs(Utave)/Mave,0.7)
axis([-10 10 -10 10 -10 10]), grid on;
xlabel('Kx');
ylabel('Ky');
zlabel('Kz');
hold on;
```

Listing 2: Averaging algorithm

```
%% Data filtering

Xfreq = 5;
Yfreq = -7;
Zfreq = 2;
band = .6;
%gaussian filter
gx=exp(-band*(Kx-Xfreq).^2);
gy=exp(-band*(Ky-Yfreq).^2);
gz=exp(-band*(Kz-Zfreq).^2);
filter = gx.*gy.*gz;
%plot of filter in signal domain
isosurface(Kx,Ky,Kz,filter,0.7);
axis([-10 10 -10 10 -10 10]), grid on;
xlabel('Kx');
ylabel('Ky');
zlabel('Kz');

xloc = (1:49);
yloc = (1:49);
zloc = (1:49);

for j=1:49
Un1(:,:,:)=reshape(subdata(:,j),n,n,n); %spacial matrix
Ut1 = fftshift(fftn(Un1));  %transform of data
Utf = Ut1.*filter;          %filtering data
Unf = ifftn(fftshift(Utf));
[M,Index] = max(abs(Unf(:))); %max value of filtered data and index of location
xloc(j) = X(Index);           %matricies of location data
yloc(j) = Y(Index);
zloc(j) = Z(Index);
end
```

Listing 3: Initialization of variables and fourier modes

```
figure(2)
plot3(xloc,yloc,zloc,'b','Linewidth',3); %3d visualization of sub path
grid on
axis([-8 8 -8 8 -8 8]);
hold on
plot3(xloc(49),yloc(49),zloc(49),'*'); %final location of sub
xlabel('X'), ylabel('Y'), zlabel('Z');
```

Listing 4: Plot of sub path