AMATH 582 Homework 1

Arvindh Sharma *

January 24, 2020

Abstract

An algorithm was developed to track the path of a marble based on 20 signals of noisy, spatial data using MATLAB. The signal from each time instance was transformed to spectral domain using Fast Fourier Transform (FFT). The transforms were averaged to extract the wavenumbers corresponding to the marble. Gaussian filters centered around the wavenumbers of interest were used to reduce the noise in the spectral domain. The filtered data was transformed back to the spatial domain with the white noise reduced. The marble path was traced from the 20 reduced-noise signals in the spatial domain.

1 Introduction and Overview

Fourier transformation is a powerful tool in signal analysis. This work uses the problem of identifying the path of a marble swallowed by Fluffy to explore some of the concepts behind spectral analysis. We are given ultrasound image signals from a narrow spatial region, and are asked to identify a marble swallowed by Fluffy by analyzing the signals. The challenges are that the data is inherently noisy and the marble is not stationary. Therefore, we employ Fourier transform to shift the analysis into the spectral domain where the marble movement in space does not alter its signature. The rest of the work focuses on reducing the noise so that the spatial location of the marble can be determined so that we can liberate Fluffy from the trouble with the marble.

2 Theoretical Background

The Fourier transformation and the filtering techniques used in this work are briefly described below [1].

2.1 Fourier series and FFT

Any periodic function, $f(x)$, can be represented using an infinite sum of sines and cosines, provided $f(x)$ is piecewise smooth. This representation is called a *Fourier series* and is shown in Eq. (1).

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

Since $f(x)$ is periodic, it is expected to repeat itself outside of the domain $(-\pi, \pi]$. It can be recognized that a function defined in an arbitrarily big domain $[-L, L]$ could be scaled to the Fourier domain, when the positional variable x is scaled to $\frac{\pi x}{L}$. The Fourier series for this domain is then given by Eq. (2).

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos \frac{k\pi x}{L} + b_n \sin \frac{k\pi x}{L} \right) \quad x \in [-L, L] \quad . \tag{2}$$

If $L \to \infty$, then the signal $f(x)$ does not necessarily need to be periodic since it is assumed to be periodic over an infinite domain for the purpose of representing it as a Fourier series. At the same time, we can replace the discrete Fourier sum with infinite continuous frequencies and write the series as an integral. This allows

*GitHub : `https://github.com/arvindhsharma`

us to define the Fourier transform pair given in Eq. (3), which contain the transformations to and back from the Fourier domain for $f(x)$.

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

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k) dk \quad . \tag{3b}$$

The FFT is an algorithm developed to accurately and quickly evaluate the Fourier transforms (and inverses). For a signal of length N, the number of operations using FFT is $O(N \log N)$. As N grows large, the number of operations grows almost linearly. In order to implement the algorithm, there is a requirement that the length of the input signal be a power of 2. Thus, any signal $f(x)$, periodic or non-periodic, can be transformed to the Fourier domain and back to its original domain (spatial, temporal, etc.) efficiently using FFT as long as its length is a power of 2. This allows us to study the signal in the domain of our choosing where trends can be identified more easily. FFT operates on a domain of length 2π, so if the input signal is of a different domain size, it will have to be scaled to 2π.

Another important concept is that when $f(x)$ is transformed to Fourier domain, the spectral signal corresponding to wave numbers does not change regardless of the order or location of the spectral content in the spatial domain. In other words, a single pulse of given frequency and amplitude in a signal $f(t) \quad t \in [0, 10]$, will have the same Fourier transform regardless of whether the pulse is centered at $t = 1$ or $t = 9$. So even though the signal generated by the marble will have different spatial response due to its changing location, its transform in the frequency domain can be expected to be unchanged.

2.2 Gaussian noise

The data to be analyzed for the marble location contains background white noise. It is known that white noise can be modeled as a Gaussian with zero mean and a unit variance being added to each frequency component in the frequency domain. This fact can be exploited to filter out the noise, since averaging the signal in frequency domain over many samples will retain the amplitude of the target frequencies while attenuating the amplitudes associated with the noise.

2.3 Gaussian filter

Gaussian filters are useful in preserving the signal strength in the spectral domain around the frequency of interest, k_0, while reducing the intensity everywhere else. It is simple to construct and in 3-dimensions, it has the added advantage of being spherically symmetric about the point of interest, $(k_{x,0}, k_{y,0}, k_{z,0})$. With the variance σ^2 acting as the factor defining the width of the filter, a simple Gaussian with unit maximum amplitude can be constructed as shown in Eq. (4) in 3-dimensions.

$$G(k_x, k_y, k_z; \sigma) = \exp(-\frac{(k_x - k_{x,0})^2 + (k_y - k_{y,0})^2 + (k_z - k_{z,0})^2}{2\sigma^2}) \quad . \tag{4}$$

To test how changing the variance σ^2 affects the signal strength in the transforms, a 1-D Gaussian filter was created as $G(k_x; \sigma) = \exp(-\frac{(k_x - k_{x,0})^2}{2\sigma^2}$. The filter was applied to a noisy test signal with a known pulse centered at $t = 0$ with various values of the standard deviation σ, as shown in Figure 1. It can be observed in Figure 1(c) that the peak of the intensity of the filtered signals transformed back into the time domain do not necessarily align at $t = 0$. This is the result of the unpredictability introduced when the noisy signal is filtered around $k_{x,0} = 0$, thereby retaining and losing some of the information from the spectral domain. In light of this observation, when using Gaussian filter in this work, a range of σ is employed and the results are checked for consistency.

3 Algorithm Implementation and Development

The algorithm detailed in Algorithm 1 is implemented in MATLAB to de-noise the signal and estimate the path of the marble from the twenty samples in the data. The high-level flow of execution is described below:

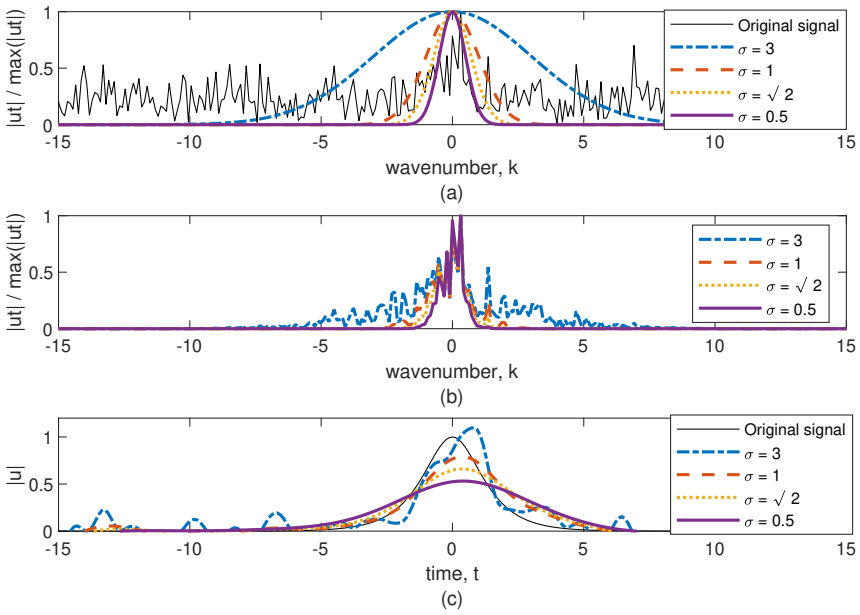

Figure 1: (a) The test signal in the spectral domain with Gaussian filters of varying σ superimposed on it; (b) Filtered test signal in the spectral domain; (c) Inverse FFT of the filtered signal in time domain.

3.1 Problem setup

The data is imported into the workspace from the file `Testdata.mat`. The data is in the form of 20 rows of spatial information obtained from the ultrasound, with both real and imaginary components, and is saved in the variable *Undata*. The spatial domain is predefined, and is known to span from -15 to 15 units in x, y, and z directions, and it is divided into 64 points along each direction. Thus the Fourier modes would number 64 after the data is transformed to spectral domain. Since FFT swaps the halves of spectral domain, the wavenumber vector k is defined to account for this swap. k also scales each wavenumber from the domain of length $2L$ to 2π by multiplying with $\frac{2\pi}{2L}$, since FFT operates only on a domain size of 2π. The spectral domain grid is built with ks, which is same as k that has been corrected for the frequency shift.

3.2 Averaging the signal in frequency domain to extract spectral signature

The raw data in *Undata* cannot be visualized to locate the marble due to its noisy nature. Since we already know that this white noise in the data is Gaussian, we can expect that averaging the Fourier transform of the signals would help in identifying the wavenumber corresponding to the ultrasound reflecting off the marble, as explained in Section 2.2.

In a loop, each of the 20 time slices of the data in *Undata* is processed thus:

- Each signal contained in *Undata* is first reshaped into a 3-dimensional matrix of length 64 in each dimension, and is stored in *Un*. Each point in *Un* is analogous to the ultrasound "pixel" value at that point in space in that time slice.
- Using MATLAB's implementation of FFT in n-dimensions(three in our case), the spatial data is transformed to the Fourier domain and saved in the 4-dimensional matrix *Unt_composite*.

Unt_composite contains the Fourier transformed information from the twenty signals, and we average the twenty signals and name it *Unt_ave*. Since the averaging process reduces the intensity of the background noise, only the wavenumbers corresponding to the signal bouncing off the marble will be unattenuated. Thus, we extract the index of the data point having the highest absolute value from the average of transformed signals in *Unt_ave*, and the corresponding wavenumbers $(k_{x,0}, k_{y,0}, k_{z,0})$ of the datapoint give the frequencies of signal reflected off the marble in x, y, and z directions respectively.

Algorithm 1: Algorithm to find the location of the marble from noisy data

Import data under the variable name $Undata$ from `Testdata.mat`
Define spatial and Fourier domains
for $j = 1:1:20$ **do**
  $Un \leftarrow$ Measurement j from $Undata$ reshaped to a matrix of size (64, 64, 64)
  $Unt\_composite(:,:,:,j) \leftarrow$ 3-D FFT of Un
end for
$Unt\_ave \leftarrow$ mean($Unt\_composite$) along its fourth dimension
$(k_{x,0}, k_{y,0}, k_{z,0}) \leftarrow$ Wavenumbers of the point where absolute value of Unt_ave is maximum
for $\sigma = [3, 1, \sqrt{2}, 0.5]$ **do**
  $filter\_3 \leftarrow$ Gaussian filter around $(k_{x,0}, k_{y,0}, k_{z,0})$, with the instance of σ as its standard deviation
  for $ii = 1:1:20$ **do**
    $Untf \leftarrow Unt\_composite(:,:,:,ii) \odot filter\_3$
    $Unf \leftarrow$ Inverse FFT of $Untf$
    $[x\_path(ii), y\_path(ii), z\_path(ii)] \leftarrow$ Point where absolute value of Unf is maximum
  end for
  $[x\_int, y\_int, z\_int] \leftarrow [x\_path(20), y\_path(20), z\_path(20)]$
  Print $[x\_int, y\_int, z\_int]$ to screen
end for
if $[x\_int, y\_int, z\_int]$ vary by less than 10^{-5} for the values of σ tested **then**
  Give the doctor $[x\_int, y\_int, z\_int]$ as location of the marble
end if

3.3 Signal Analysis with Gaussian filter

The next steps are geared towards getting back the spatial location of the marble in each of the twenty signals. To do so, we employ Gaussian filter to extract the spectral content from $Unt\_composite$ corresponding to the wave numbers $(k_{x,0}, k_{y,0}, k_{z,0})$ while attenuating the signal everywhere else. However, as seen in Section 2.3, the filter response can affect the reconstructed spatial information due to the presence of noise and cause uncertainties in the precise location of the marble. In order to get a measure of certitude, the following steps are repeated for Gaussian filters with different standard deviation values of $\sigma = [3, 1, \sqrt{2}, 0.5]$:

1. Construct a Gaussian filter in 3-dimensions around the wavenumbers of interest, $(k_{x,0}, k_{y,0}, k_{z,0})$. The filter will have a standard deviation of the instance of σ in the current iteration and will have a maximum amplitude of unity.

2. For each of the twenty 3-D slices of Fourier transformed data in $Unt\_composite$, the following manipulations are done:

   2.1. Element-wise multiplication with the Gaussian filter to yield the filtered matrix, $Untf$.

   2.2. $Untf$ undergoes inverse FFT to obtain the spatial data of the filtered spectral data.

   2.3. Since we expect the the location of the marble to manifest as the point with maximum intensity in $Untf$, we extract the maximum of the absolute value and identify its location in (x, y, z). This is stored in the array (x_int, y_int, z_int).

3. We are interested in where the marble is in the twentieth time instance, which is nothing but the last row of (x_path, y_path, z_path). We save these coordinates in (x_int, y_int, z_int).

3.4 Repeatability check

Finally, we have an estimate of the marble location in physical space (x, y, z) obtained with the Gaussian filters with varying σ. To check if there is any uncertainty in the marble location at the last time point, we see if the predicted position (x_int, y_int, z_int) estimated with the different filters are all falling within a small range, arbitrarily chosen in this case to be 10^{-5}. Once this is verified to be true, we can predict the marble location to be (x_int, y_int, z_int) to at least four significant digits after the decimal point with certainty.

4 Computational Results

Appendix B.1 shows the MATLAB code developed from the implementation of Algorithm 1. The code also includes methods to visualize the steps and results of execution to help follow along the process.

After loading the data from the file `Testdata.mat`, the spatial and Fourier domains are defined for their respective spans using the `meshgrid()` command. Note that the variable k, which defines the wavenumbers by transforming the interval from $(-L, L)$ on to the Fourier interval $(-\pi, \pi)$, has been declared such that the arrangement of frequencies is suitable for analysis with FFT. Therefore, in order to correct the order of wavenumbers, `fftshift()` is used on k to bring the center frequency back to the middle of the array and is stored as ks. The Fourier domain grid, (Kx, Ky, Kz), is then generated from ks.

Following this, a loop is implemented to get the average of the FFT transformed raw data so that the wave numbers from the reflected signal, $(k_{x,0}, k_{y,0}, k_{z,0})$, could be extracted by reducing the noise. The FFT transform is carried out in 3-dimensions using the `fftn()` command. The location of the maximum value from the 3-D grid of Unt_ave is extracted by first flattening it and then finding the index of the maximum absolute value using `max()` and `abs()` in conjugation. The visualization (Figure 3 in Appendix C) helps gain an intuition of the process.

Once $(k_{x,0}, k_{y,0}, k_{z,0})$ is known, Gaussian filters are created with $\sigma = [3, 1, \sqrt{2}, 0.5]$ to filter the data in the Fourier domain. Since the Fourier grid was formed from the FFT-shifted wavenumber vector ks, we need to apply inverse FFT-shift with `ifftshift()` on the Fourier grid (Kx, Ky, Kz) before the filter can be multiplied element-wise. The filtered signals are then transformed back to the spatial domain using `ifftn()`. A process similar to what was used to find the maximum in frequency domain is implemented to get the maximum absolute value in the spatial domain, which for each of the twenty data measurements gives the location of the marble at that time point. The results are stored in the vector $[x\_path, y\_path, z\_path]$, the twentieth point of which gives the location of the marble at time $t = 20$. We call this location (x_int, y_int, z_int).

Table 1 lists the results from the program, and it can be seen that (x_int, y_int, z_int) is consistent for different values of σ. Therefore, it can be assessed with certainty that the marble location at the twentieth data measurement is $(x, y, z) = (-5.625, 4.219, -6.094)$.

Wavenumbers		Results
$(k_{x,0}, k_{y,0}, k_{z,0})$		$(1.885, -1.047, 0)$
Terminal marble position	Filter parameter	Results
(x_int, y_int, z_int)	$\sigma = 3$	$(-5.625, 4.219, -6.094)$
	$\sigma = 1$	$(-5.625, 4.219, -6.094)$
	$\sigma = \sqrt{2}$	$(-5.625, 4.219, -6.094)$
	$\sigma = 0.5$	$(-5.625, 4.219, -6.094)$

Table 1: Computational results.

The path that the marble traverses can be plotted using `plot3()` in 3-dimensions. Figure 2 shows this path and the end location of the marble is indicated by the pentagonal marker.

5 Summary and Conclusions

This work utilizes the concepts of Fourier transforms and filtering to identify the location of a marble from noisy data. Since it is difficult to identify the marble location in spatial domain owing to the noise and its movement in space, the signal is transformed to spectral domain using FFT in MATLAB. The wavenumbers associated with the marble's position are identified by averaging the signal in the spectral domain. Having identified the wavenumbers, Gaussian filtering is used to remove the noise in the spectral domain. The filtered signal is then transformed back to the spatial domain for each of the twenty measurements to trace out the path of the marble. By employing filters of varying σ, the uncertainty associated with the filtering process is negated to an extent. The final location of the marble at the twentieth data measurement is determined to be $(x, y, z) = (-5.625, 4.219, -6.094)$.

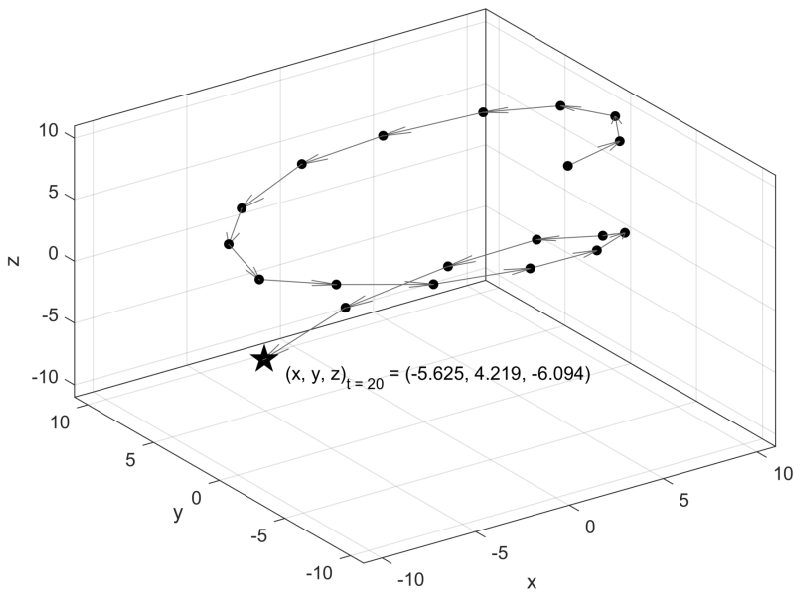

Figure 2: Visualization of the marble path over the twenty data measurements.

References

[1] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

Appendix A MATLAB Functions

Here is an overview of the major functions used in the program in Appendix B.1.

- `Y = fftn(X)` returns the N-dimensional FFT of the N-dimensional array `X`, equivalent to estimating FFT along each dimension of `X`. `Y` is of the same size as `X`.
- `Y = ifftn(X)` returns the discrete inverse Fourier transform of the N-dimensional array `X`.
- `Y = fftshift(X)` rearranges Fourier transform `X` so that the zero frequency component is at the center of the array. `X` can be a vector, a matrix, or higher-dimensional array.
- `X = ifftshift(Y)` performs the inverse of `fftshift()` and rearranges `Y` so that the frequency components are in the same order as the original FFT output.
- `[X, Y, Z] = meshgrid(x, y, z)` returns 3-D grid coordinates based on the coordinates of the vectors `x`, `y`, and `z`. The grid has a size of length(`y`) $\times$ length(`x`) $\times$ length(`z`).
- `B = reshape(A, sz)` reshapes `A` into matrix `B` with shape `sz`, which is the size vector. For example, if `sz` is $(3, 2)$, then `B` will have three rows and two columns.
- `plot3(X, Y, Z)` plots the coordinates represented by the vectors `X`, `Y`, and `Z` in 3-dimensions, provided they are of the same lengths.
- `y = linspace(x1, x2, n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.

Appendix B MATLAB Code

B.1 Main program

This is the script used to analyze the given data and find the path of the marble.

```
%% LOADING DATASET AND DEFINING DOMAINS

clear; close all; clc;
load Testdata

L = 15; % 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);
Unt_composite = nan(n, n, n, 20);

%% AVERAGING THE FF TRANSFORM TO EXTRACT FREQUENCIES OF INTEREST

for j = 1:20
    Un = reshape(Undata(j,:),n,n,n);
    Unt_composite(:, :, :, j) = fftn(Un); % 4-D matrix containing FFT of the 20 signals
end

Unt_ave = mean(Unt_composite, 4); % Average of 20 FFTs
Unts_ave = fftshift(Unt_ave);

[~, maxid] = max(abs(Unts_ave(:))); % Point where magnitude of FFT is maximum

k_x0 = Kx(maxid); k_y0 = Ky(maxid); k_z0 = Kz(maxid);

% Visualization of average FFT in frequency space
close all, isosurface(Kx, Ky, Kz, abs(Unts_ave) / max(abs(Unts_ave(:))), 0.65);
axis([-7 7 -7 7 -7 7]), drawnow
hold on
h2 = plot3(k_x0, k_y0, k_z0, 'r*', 'MarkerSize', 15);
ax = gca; ax.Box = 'on';
xlabel('Wavenumber in x-direction, k_x')
ylabel('Wavenumber in y-direction, k_y')
zlabel('Wavenumber in z-direction, k_z')
grid on
fig1 = gcf;
fig1.Units = 'inches';
fig1.Position = [-.1 1.8 6.75 5.0625];
fig1.PaperUnits = 'inches';
fig1.PaperSize = [6.75 5.0625];
% print('frequencies', '-depsc', '-r600')
% print('frequencies', '-dpng', '-r600')

fprintf('\nFrequencies of interest are:\nfx = %.3f\nfy = %.3f\nfz = %.3f\n', k_x0, k_y0, k_z0)

%% SIGNAL ANALYSIS
```

```
sig = [3, 1, (1 / sqrt(2)), .5]; % Vector with the standard deviations to be used in Gaussian filter
x_int = nan(1, length(sig)); y_int = nan(1, length(sig)); z_int = nan(1, length(sig));

for ii = 1:1:length(sig)

    %% FILTER CONSTRUCTION AND VISUALIZATION

    % 3-D Gaussian filter
    filter_3 = exp(-((((ifftshift(Kx) - k_x0) .^ 2 ...
        + (ifftshift(Ky) - k_y0) .^ 2 + (ifftshift(Kz) - k_z0) .^ 2)) / (2 * sig(ii) ^ 2)));

    % Filter visualization
    close all, isosurface(Kx, Ky, Kz, fftshift(filter_3), .5);
    axis([-7 7 -7 7 -7 7]), drawnow
    xlabel('Kx'), ylabel('Ky'), zlabel('Kz')
    fig2 = gcf;
    pause(1), close

    %% MARBLE PATH ESTIMATION

    x_path = nan(20, 1); y_path = nan(20, 1); z_path = nan(20, 1);
    for jj = 1: 1: 20
        Untf = Unt_composite(:, :, :, jj) .* filter_3;
        Unf = ifftn(Untf); % Inverse FFT of filtered signal in frequency domain

        [~, maxid1] = max(abs(Unf(:)));
        x_path(jj) = X(maxid1); y_path(jj) = Y(maxid1); z_path(jj) = Z(maxid1); % Estimated marble locat

    %     close all, isosurface(X, Y, Z, abs(Unf) / max(abs(Unf(:))), 0.9)
    %     axis([-20 20 -20 20 -20 20]), grid on, drawnow
    %     xlabel('x'), ylabel('y'), zlabel('z')
    %     hold on
    %     plot3(x_int(jj), y_int(jj), z_int(jj), 'r*', 'MarkerSize', 20)
    %     pause(1)
    %     close

    end

    x_int(ii) = x_path(end); y_int(ii) = y_path(end); z_int(ii) = z_path(end); % Marble location at the

    fprintf('\nIteration %d :', ii)
    fprintf('\nThe marble''s estimated location is:\n')
    fprintf('x = %.3f\ny = %.3f\nz = %.3f\n', x_path(end), y_path(end), z_path(end))

end

% Check to see if the marble location estimates are consistent

if range(x_int) < 1e-5 && range(y_int) < 1e-5 && range(z_int) < 1e-5
    fprintf('\nThe marble''s location where the acoustic beam is to be focused is:\n')
    fprintf('x = %.3f\ny = %.3f\nz = %.3f\n', x_int(end), y_int(end), z_int(end))
else
    warning('The marble location is uncertain. Verify before proceeding.')
end
```

```
%% VISUALIZATION OF THE PATH OF THE MARBLE


fig3 = figure;
fig3.Units = 'inches';
fig3.Position = [-.1 1.8 6.75 5.0625];
fig3.PaperUnits = 'inches';
fig3.PaperSize = [6.75 5.0625];
h3 = plot3(x_path, y_path, z_path);
h3.Color = 'k';
h3.LineStyle = ':';
h3.Marker = '.'; h3.MarkerSize = 18;
hold on
h4 = quiver3(x_path(1:19), y_path(1:19), z_path(1:19), diff(x_path), diff(y_path), diff(z_path));
h4.Color = [0.4 0.4 0.4]; h4.MaxHeadSize = 0.2;
h5 = plot3(x_path(end), y_path(end), z_path(end), 'k.', 'MarkerSize', 20);
h5.Marker = 'p'; h5.MarkerFaceColor = 'k'; h5.MarkerSize = 15;
xlabel('x'), ylabel('y'), zlabel('z')
axis([-11 11 -11 11 -11 11]), grid on
ax = gca; ax.Box = 'on'; ax.View = [-35.1 36.8];
% print('path', '-depsc', '-r600')
% print('path', '-dpng', '-r600')
```

B.2 Program to explore filter characteristics

This is the script used to check how changing σ^2 affects the Gaussian filter performance.

```
clear; close all; clc

%% NOISE ON SIGNAL

L = 30; % time slot to transform
n = 512; % number of Fourier modes 2^9
t2 = linspace(-L, L, n+1); t = t2(1:n); % time discretization
k = (2 * pi / (2 * L)) * [0:(n / 2 - 1) (-n / 2):-1]; % frequency components of FFT
u = sech(t); % ideal signal in the time domain

%%  SIGNAL AND SPECTRUM

noise = 10;
ut = fft(u);
unt = ut + noise * (randn(1, n) + 1i * randn(1, n));
un = ifft(unt);

%% GAUSSIAN FILTERS WITH DIFFERENT VARIANCES

colorvec = lines(4);
lnstyles = {'-.', '--', ':', '-'};
fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 4.5];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 4.5];
s1 = subplot(3, 1, 1);
```

```
s1.Box = 'on';
plot(fftshift(k), abs(fftshift(unt)) / max(abs(fftshift(unt))), 'k', 'LineWidth', 0.5)
hold on
axis([-18 18 0 1])
xlabel({'wavenumber, k', '(a)'}), ylabel('|ut| / max(|ut|)')
s2 = subplot(3, 1, 2);
s2.Box = 'on';
hold on
axis([-15 15 0 1])
xlabel({'wavenumber, k', '(b)'}), ylabel('|ut| / max(|ut|)')
s3 = subplot(3,1,3);
s3.Box = 'on';
plot(t, u, 'k', 'LineWidth', 0.5)
hold on
axis([-15 15 0 1.2])
xlabel({'time, t', '(c)'}), ylabel('|u|')
c1 = 0;
for sig = [3, 1, (1 / sqrt(2)), .5]
    c1 = c1 + 1;
    subplot(s1)
    filter = exp(-(k).^2 / (2 * sig ^ 2));
    unft = filter .* unt;
    unf = ifft(unft);
    plot(fftshift(k),fftshift(filter), 'Color', colorvec(c1, :), ...
        'LineStyle', lnstyles{c1}, 'Linewidth', 1.5)

    subplot(s2)
    plot(fftshift(k), abs(fftshift(unft)) / max(abs(fftshift(unft))),...
         'LineStyle', lnstyles{c1}, 'Color', colorvec(c1, :), 'LineWidth', 1.5)

    subplot(s3)
    plot(t, unf, 'Color', colorvec(c1, :), 'LineStyle', lnstyles{c1}, 'Linewidth', 1.5)
end
subplot(s1)
legend('Original signal', '\sigma = 3', '\sigma = 1', '\sigma = \surd 2', '\sigma = 0.5')
subplot(s2)
legend('\sigma = 3', '\sigma = 1', '\sigma = \surd 2', '\sigma = 0.5')
subplot(s3)
legend('Original signal', '\sigma = 3', '\sigma = 1', '\sigma = \surd 2', '\sigma = 0.5')
% print('filter_sig_variations', '-depsc', '-r600')
% print('filter_sig_variations', '-dpng', '-r600')
```

Appendix C Additional figure

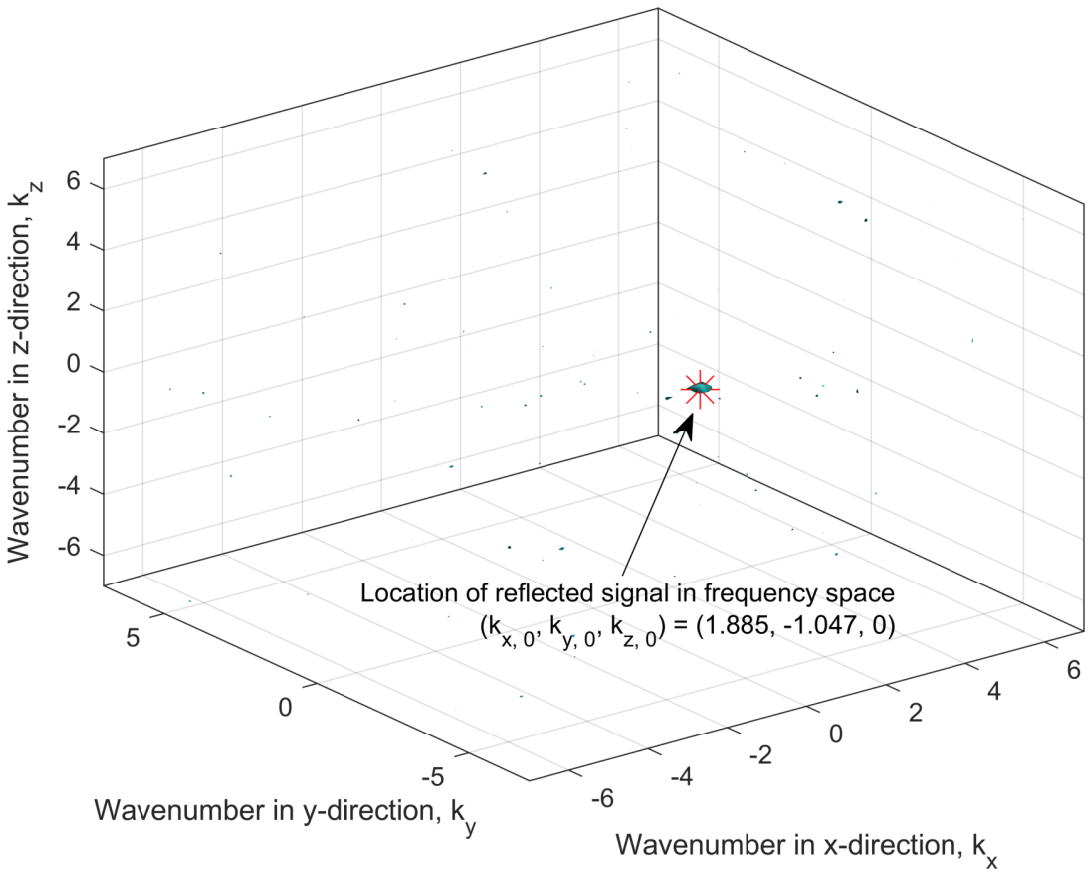

Figure 3: Plot indicating the wavenumbers corresponding to the marble location in spectral domain.