AMATH 580: Homework 1

Hyun Ah Lim
January 24, 2020

Abstract

Ultrasound readings of the abdominal cavity of a poor dog named Fluffy were taken in order to identify the location of a marble that she had recently ingested. The location of the said marble could be not determined in any of these measurements due to significant levels of noise contained in the data. Since the ultrasound device used to track the moving object had a fixed frequency, a spectral analysis was performed to denoise the data. Specifically, the discrete fast Fourier Transform routine was used to transform the spatial data into their frequency components where white noise could be removed, revealing the unknown central frequency at (Kx, Ky, Kz) = (1.8850, -1.0472, 0). Based on this center value, a 3D Gaussian filter was applied to each measurement in the spectral domain in order to clean the data. An inverse fast Fourier Transform then converted the cleaned data back to its original spatial domain, where the location of the marble was then identified in all 20 measurements. The last location, 20th measurement, was found to be (X, Y, Z) = (-5.625, 4.21875, -6.09375). An intense acoustic wave can now be directed to this location and the marble destroyed. Fluffy's (and any other so inclined dog's) life can be saved by spectral analysis.

1 Introduction and Overview

As an example of the uses of spectral analysis, and the fast Fourier Transform, 20 noise filled ultrasound measurements of the intestinal cavity of dog were provided. The objective was to denoise the data in order to detect the path of a marble that the dog had recently ingested. Most importantly, we had to find the last location of the marble so that an intense acoustic wave could be used to destroy it.

The measurements were provided as a 20 x 262144 2D matrix, each row representing a single measurement in the spatial domain. As stated earlier, significant noise clouded the actual location of the marble in this raw signal data. The ultrasound signal's frequency was unknown but fixed. Applying the fast Fourier Transform routine should transform the data to its frequency components and allow for the identification of the unknown center frequency. By identifying this center frequency, a 3D Gaussian filter can be created and applied to remove noise at frequencies unlikely to be part of the ultrasound signal. Then transforming the cleaned data back to the spatial domain should provide the marble's location, if the correct central frequency was identified and the 3D Gaussian filter correctly applied.

2 Theoretical Background

As explained in our textbook [1], Fourier introduced to us the concept that any signal can be represented as a series of sine and cosine functions (1):

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx) \; ; x \, \epsilon \, [-\pi, \pi]$$

(1)

Expanding on this idea, using the orthogonality properties of sine and cosine and Euler's identity ($e^{\pm ix} = \cos x \pm i \sin x$), we were shown in lecture that the Fourier Transform (2) along with its inverse (3) could be written as integral transforms:

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

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

Applied to a finite domain of $x \in [-L, L]$, it provides a discrete sum of the eigenfunctions and associated wavenumbers. In other words, it provides information about the frequency components of that function contained in that domain.

As shown to us in a demonstration during lecture, this is extremely useful when trying to locate a moving object whose signal is masked by white noise but transmits at a fixed frequency. We can transform the signal reading from the spatial domain to the frequency domain where we can then clean and filter the data to include only those components centered around that fixed frequency.

If the center frequency is unknown, masked by white noise, much of that noise can be removed by averaging the signal data in the frequency domain across multiple measurements, since its mean value is 0. Once the center frequency is identified, applying a Gaussian filter (4) around that center value can be used to clean the data for each measurement. This cleaned data can then be then transformed back to spatial domain, with significant removal of noise. In the example filter shown below c is our center frequency value, x is the variable for frequency, and τ tells us how wide our filter will be (indicator of the standard deviation of a normal distribution).

$$\text{Gaussian Filter} = e^{-(\tau(x-c)^2)} \tag{4}$$

The fast Fourier Transform routine, takes the spectral transformation method described above and provides additional efficiency by discretizing the frequency domain into 2^N points, reducing the processing resources required to $O(N\, log\, N)$ from the typical $O(N^2)$. In general, the transform does have a constraint in that it assumes the interval $x \in [-L, L]$ is periodic, which creates a boundary effect.

The routine also requires additional process steps to account for impacts to the data, as outlined in the textbook[1]:

1. The data is shifted so that $x \in [0, L] \rightarrow [-L, 0]$ and $x \in [-L, 0] \rightarrow [0, L]$
2. Every other mode is multiplied by -1
3. Assumes the interval is on a 2π periodic domain

These impacts can be resolved and accounted for and are documented in the Algorithm Implementation and Development section.

3 Algorithm Implementation and Development

As stated earlier, the image data was provided as a 20 x 262144 array (Testdata.mat), each row representing a 3D ultrasound image/measurement. The algorithms outlined below were developed using MATLAB.

3.1 Set up initial parameters based on the given domain and spatial and spectral resolution

- 3.1.1 Establish spatial domain [-L, L], where $L = 15$.
- 3.1.2 Define the Fourier Modes at $n = 64$. This is our spectral resolution, in that 2^n discrete frequencies will be measured.
- 3.1.3 Discretize spatial domain using n and L values. This is our spatial resolution.
- 3.1.4 Discretize the frequency domain based on Fourier Modes (n), and normalizing for the [-L, L] interval to Fourier's assumed [-π, π]. This is our spectral resolution (2^n frequencies).
- 3.1.5 Define the (X,Y,Z) 3D coordinate grid for the spatial domain. Each dimension interval is [-L, L – (2L/n)], with n points, each (2L/n) apart. Note that the Fourier Transform assumes a periodic function, so the last point in our domain is L – (2L/n), not L.
- 3.1.6 Define the (Kx, Ky, Kz) 3D coordinate grid for the frequency grids space. Each axis domain is [- (2π/2L)*(n/2), (2π/2L)*(n/2-1)] and each point is spaced 2π/2L apart. Spacing is normalized because the Fourier Transform assumes a 2π interval, and the last point is (2π/2L)*(n/2-1) and not (2π/2L)*(n/2-1) because it also assumes a periodic domain.
- 3.1.7 Create a zero filled 3D matrix for use in accumulating a running total in the following loop algorithm.
- 3.1.8 Create a zero filled 4D matrix to store each transformed iteration for later use.

3.2 Iteration: Reshape, Transform and Sum

- 3.2.1 Extract one measurement (i.e. row of data) from the original Testdata.mat file and reshape this 1 x 262144 vector as a 3D matrix for the spatial domain.
- 3.2.2 Transform the resulting 3D matrix using the fftn() function, an N-dimensional discrete fast Fourier Transform. In the same step, apply the fftshift() function to realign the transformed data to the k axes (since fftn(), flips the left and right halves of the spectrum). The data is now in the spectral domain for this measurement.
- 3.2.3 Add the transformed spectral data from the previous step, to the zero filled 3D matrix, to build a total at each frequency component.
- 3.2.4 Write transformed data to a separate 4D matrix for later use (there is no averaging here, we are just retaining each iteration of the transformed data).

3.3 Remove White Noise Based on Mean Value Property

After completion of the loop above, average the sum total of the frequency components by dividing the matrix created in step 3.2.3 by the total number of measurements, 20. Since white noise has a mean value = 0, taking the average across multiple measurements should reveal the fixed frequency of the ultrasound signal to be the only components that do not reduce near 0. Note that

the absolute value of the calculated average should be used since the Fourier Transform multiplies every other mode by -1.

3.4 Normalize the Average Value Matrix

Normalize the elements of the average value matrix produced in section 3.3 (divide all values by the maximum value) and visualize the result. A concentration of large values can be seen in this visualization (figure 3.4.1) indicating that a signature or center frequency does exist.

3.5 Build a 3D Gaussian filter based on the identified center frequency

- 3.5.1 Identify the maximum value and identify its location in the (Kx, Ky, Kz) coordinate system using the ind2sub() function. Note that this function will output the coordinates as (Ky, Kx, Kz).
- 3.5.2 Use this position as your (kmx, kmy, kmz) values to create a 3D Gaussian filter centered around this center frequency point. Note that the tau (τ) values determines of the tightness of the filter around the center frequency (indicator of the standard deviation of the distribution curve). A few values were tried, and a value of 4 was determined effective through trial and error.

$$\text{3D Gaussian Filter} = e^{-(\tau((x-x)^2(Ky-km\ )^2(Kz-km\ )^2))}$$

3.6 Apply 3D Gaussian Filter and identify (X,Y,Z) location in the spatial domain

- 3.6.1 Extract a measurement from the 4D matrix created in the first loop algorithm and use the squeeze() function to transform it to a 3D matrix.
- 3.6.2 Multiply this measurement by the 3D Gaussian filter created in the previous step (this removes signal data, that is not centered around the frequency of the ultrasound).
- 3.6.3 Shift the data back to the inverted fast Fourier Transform layout using iffshift().
- 3.6.4 Take the inverse fast Fourier Transform function, ifftn(), to transform the data back to the spatial domain.
- 3.6.5 Identify the maximum value and identify its location.
- 3.6.6 Translate the matrix location of the maximum value, to its (X,Y,Z) using the ind2sub() function. Note that this function will output the coordinates as (Y,X, Z).
- 3.6.7 Write these (X,Y,Z) coordinate positions to a series of vectors in order to track and visualize the path of the marble.

3.7 Identify the (X,Y,Z) coordinate values of the marble at the 20th measurement.

3.8 Visualize Marble Path and Locations

- 3.8.1 Use plot3() to plot the path of the marble as a line.
- 3.8.2 Use isosurface() to draw the location of the marble in each of the twenty measurements.

4 Computation Results

4.1 Center Frequency of the Ultrasound Transmission

Center Frequency found at (Kx, Ky, Kz) = (1.8850, -1.0472, 0)

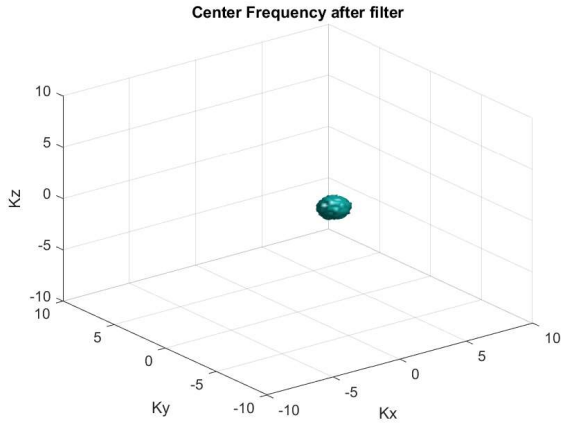

FIGURE 1. ISOSURFACE DEPICTION OF THE CENTER FREQUENCY AFTER AVERAGING, NORMALIZATION AND APPLICATION OF A 3D GAUSSIAN FILTER

4.2 Marble Location Coordinates and Path

The Spectral Analysis performed revealed the location of the marble at the following points during each of the 20 data measurements provided by ultrasound (Table 1). Also provided are the visual representations of these 20 locations and their path (figure 1 and figure 2).

Measurement	X	Y	Z
1st	4.6875	-4.21875	9.84375
2nd	8.4375	-2.8125	9.84375
3rd	9.84375	-0.46875	9.84375
4th	8.90625	2.34375	9.84375
5th	6.09375	4.21875	9.375
6th	1.40625	4.6875	8.90625
7th	-3.28125	4.6875	8.4375
8th	-7.96875	2.8125	7.5
9th	-9.84375	0.9375	7.03125
10th	-9.84375	-1.40625	6.09375
11th	-7.5	-3.28125	5.15625
12th	-2.8125	-4.6875	4.21875
13th	2.34375	-4.6875	3.28125
14th	6.5625	-3.75	2.34375
15th	9.375	-1.875	1.40625
16th	9.84375	0.9375	-0.46875
17th	7.96875	2.8125	-0.9375
18th	4.21875	4.21875	-2.8125
19th	-0.9375	4.6875	-4.21875
20th	-5.625	4.21875	-6.09375

TABLE 1: MARBLE COORDINATE LOCATIONS FOUND AFTER DENOISING OF DATA THROUGH SPECTRAL ANALYSIS

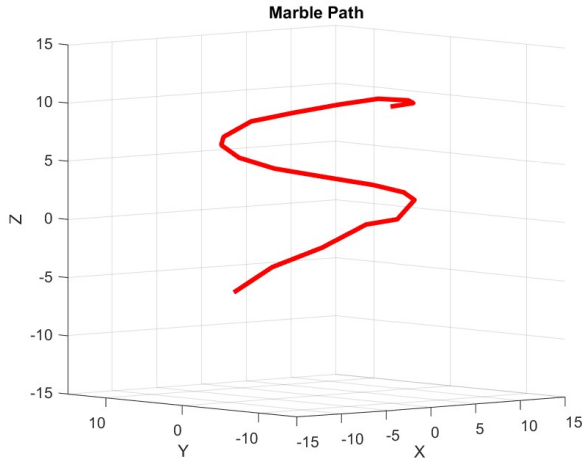

FIGURE 1: VISUALIZATION OF MARBLE PATH BASED ON 20 ULTRASOUND MEASUREMENTS.

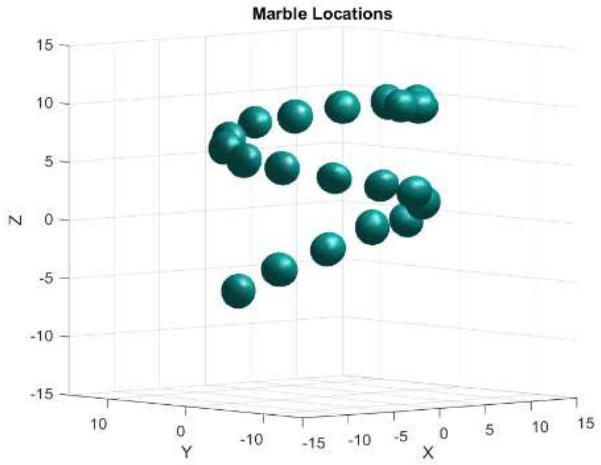

FIGURE 2: VISUALIZATION OF MARBLE LOCATION AT EACH OF THE 20 ULTRASOUND MEASUREMENTS.

4.3 Marble Last Location (20th measurement)

The last location of the marble (the 20th measurement) indicates that the intense acoustic wave should be aimed at (X, Y, Z) = (-5.625, 4.21875, -6.09375)

5 Summary and Conclusions

Spectral analysis is an effective method for denoising (i.e. cleaning) signal data. The MATLAB functions used to apply the fast Fourier Transform steps required for such an analysis (fftn(), fftshift, ifftn() and ifftshift()) proved accurate and concise in terms of the coding required. In this scenario, 20 ultrasound measurements were masked with a significant amount of noise and provided for analysis. The task was to track a moving object's fixed (yet unknown) frequency signal hidden in the data and find its final location. It is important to note that when the fixed frequency is unknown, a sufficient number of measurements must be taken in order to remove enough white noise to identify this unknown value. The 20 measurements proved adequate in this data set and the center frequency was identified. Then applying a Gaussian 3D filter centered around this frequency successfully denoised the data for each separate measurement.

The resulting denoised data was transformed back to the spatial domain and the location of the object (a marble) in each of the 20 measurements was found. It can be said with confidence that the method used in this analysis would have saved Fluffy's life.

References

[1] J. Nathan Kutz 2013, Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data, Oxford University Press

Appendix A: Relevant MATLAB functions

1. reshape(): The reshape function allows you to reshape the dimensions of a given matrix. In this instance it was used to reshape the initial measurement structured as a vector, into a 3D matrix representing a spatial domain.

2. fftn(): This an N dimensional discrete fast Fourier Transform function. Since we were working with 3D spatial data, fftn() was used rather than ft() to transform 3D spatial data to a 3D frequency domain

3. fftshift(): This function was is used to move the 0 values back to the center after the fftn() function was used. Does not affect processing but allows us to better visualize the frequency domain

4. ifftn(): This is the N dimensional Inverse discrete fast Fourier Transform function. This was used to transform data back from the spectral domain to the spatial domain.

5. ifftshift(): This reverses the fftshift() that was performed in the spectral domain, back to the original frequency coordinate order where the 0 values are at the outer edges. This should be performed prior to executing the ifftn() to transform back to the spatial domain.

6. squeeze(): This function removes any single element dimensions. Since the raw spectral measurements were retained in a 4D matrix. When we had to peel off a measurement at a time as a 3D measurement, this function was very useful.

7. ind2sub(): This function finds the {row, column, page} of a given value in a matrix. It was used to identify the central frequency in the spectral domain and the marble position in the spatial domain. Important note is that since it outputs {row, column, page}, you end up with (y,x,z) coordinates.

8. isosurface(): Provides surface plotting capability in 3D based on values contained in a 3D matrix. This was used to plot the 20 locations found for the marble.

9. plot3(): Function provides 3D plotting capabilities which were used to map the path of the marble.

10. print(): Generates image files. This was used to create figures used in this paper.

11. 'for loop' (honorable mention): This isn't a function but the 'for loop' allowed us to perform iterations across all 20 measurements with minimal coding.

Appendix B: MATLAB Code

```
clear all; close all; clc;
load Testdata

L=15; % spatial domain
n=64; % Fourier modes
x2=linspace(-L,L,n+1); % spatial discretization
x=x2(1:n);  % dimension x
y=x;  % dimension y
z=x;  % dimension z
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1];  % FFT frequency components
ks=fftshift(k);  % move 0 frequencies to center for visualization purposes

[X,Y,Z]=meshgrid(x,y,z);   % create spatial grid space
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);  % create frequency gridspace
Untp_sum = zeros(n,n,n);
Untp_all = zeros(length(Undata(:, 1,1,1)),n,n,n);

% Average data in frequency domain to remove white noise and identify central frequency
for jj = 1:length(Undata(:, 1,1,1))
      Un = reshape(Undata(jj,:),n,n,n);
      Untp = fftshift(fftn(Un));
      Untp_sum = Untp_sum + Untp ;
      Untp_all(jj,:,:,:) = Untp; % save this for filter/inverse iter
end
Utpave=abs(Untp_sum)/jj;
Utpave_norm = Utpave/max(Utpave(:));

% viz average in 3D after normalization without filter
close all
isosurface(Kx,Ky,Kz,abs(Utpave_norm), .55)
   axis([-10 10 -10 10 -10 10]), grid on, drawnow
xlabel('Kx'); ylabel('Ky'); zlabel('Kz')

% extract max value/indices
[M, I] = max(abs(Utpave(:)));
[kmy,kmx,kmz] = ind2sub(size(Utpave), I);
kx = ks(kmx); ky = ks(kmy); kz =  ks(kmz);

%  create 3D Gaussian filter given tau and CF
tau = 4; % standard dev for Gaussian filter (how narrow should the filter be)
filter = exp(-(tau.*(Kx - kx).^2 + tau.*(Ky - ky).^2 + tau.*(Kz - kz).^2));

% Apply filter to average signal for viz of cleaned CF
close all
Utpf = filter.*Utpave;
isosurface(Kx,Ky,Kz,abs(Utpf), .75)
```

```
   axis([-10 10 -10 10 -10 10]), grid on, drawnow
title('Center Frequency after filter')
xlabel('Kx'); ylabel('Ky'); zlabel('Kz')
print(gcf, '-dpng', 'id_cf.png')

% Generate cleaned Spatial Realizations
for mm = 1:jj
  Untp = squeeze(Untp_all(mm,:,:,:));
   Untpf = filter.*(Untp);
   Untf = ifftshift(Untpf);
   Unf = ifftn(Untf);
   Unf_all(mm,:,:,:) = Unf;
end

close all
%  Identify coordinates for each realization and create table figure
xs = zeros(jj, 1); ys = zeros(jj, 1); zs = zeros(jj, 1);
for pp = 1:jj
   real = squeeze(Unf_all(pp, :, :, :));
   [PM, PI] = max(abs(real(:)));
   [py,px,pz] = ind2sub(size(real), PI);
   xx = x(px); yy = y(py); zz =  z(pz);
   xs(pp, :) = x(px); ys(pp,:) = y(py); zs(pp,:) = z(pz);
end

% plot Isosurface for each realization
close all
for vv = 1:jj
 Unf = squeeze(Unf_all(vv, :, :, :));
 Unf_norm = abs(Unf)/(max(abs(Unf(:))));

 isosurface(X,Y,Z,abs(Unf_norm), .9)
 axis([-15 15 -15 15 -15 15]), grid on, drawnow
 hold on
end

title('Marble Locations')
view([-40.5 5]); xlabel('X'); ylabel('Y'); zlabel('Z')
print(gcf, '-dpng', 'marble_loc.png');

close all
plot3(xs, ys, zs, 'r', 'Linewidth', 3)
view([-40.5 5]);
xlim([-15 15]);  ylim([-15 15]); zlim([-15 15]);
grid('on'); xlabel('X'); ylabel('Y'); zlabel('Z')
title('Marble Path')
print(gcf, '-dpng', 'descent_path.png');
```

Appendix C: Github

https://github.com/Washington-Turtles/amath582.git