AMATH 482 - Computational Methods in Data Analysis: Implementing Fast Fourier Transform

Gargi Kher
Friday January 24th, 2020

Abstract: The Fast Fourier Transform has many applications in mathematics, science, and engineering. This theorem can take in a function on a particular interval and break it up into functions of sines and cosines, making it useful for analyzing the frequency of some piece of data. In this assignment, I take a noisy set of data from an ultrasound measuring the location of a marble over time and incorporate the Fast Fourier Transform into certain methods to denoise the data. I will discuss my process and results for obtaining the final position of the marble.

Sec. I. Introduction and Overview

In this assignment, we were tasked with denoising data for the position of a marble obtained from an ultrasound. The ultrasound measured the position of the marble at twenty instances in time. This data was given in the form of a two-dimensional array in a downloadable MATLAB file 'Testdata.mat'. This matrix represented the three-dimensional position data (x, y, and z coordinates), of the marble for all of the twenty different measurements, but was crowded with noise that made it impossible to see where the marble was at each measurement (timestep). It was our job to filter out the noise in the frequency domain in order to find the actual position of the marble inside the dog. Starter code was provided by the instructor for this assignment. All of the functions used in this assignment were related to the Fast Fourier Transform in some way, which will be explained in the next section.

Sec. II. Theoretical Background

The Fast Fourier Transform (Eqn.1) can take a discrete function in time space and convert it into frequency space by breaking up the function into sines and cosines. The transformed function can then be converted back into time space by applying an Inverse Fast Fourier Transform (Eqn.2).

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

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

While both equations are depicted as being on an infinite interval, when working with data, we only have a finite amount to analyze. These equations will still work for data on an interval defined as $x \in [-L, L]$. This theorem can solve operations on an order $Nlog(N)$ scale, making it faster than most other algorithms that are order N^2. Part of the reason this algorithm has a relatively fast computing time is due to dividing the defined interval into 2^n points, with n being any whole number. Looking at equation 1, we can see that some function on an interval (we will imagine the interval is finite) is taken in. This function is decomposed over its interval as a series of sines and cosines (e^{-ikx}, a complex value, can be written in terms of sines and cosines). The

Fourier Transform, now in the frequency domain, can be sent back to its original form, using equation 2.

One thing to note about the Fast Fourier Transform is that it follows the Heisenberg Uncertainty Principle. Objects are defined by their position and momentum. This principle states that the more you know about an object's position, the less you'll know about its momentum, and vise-versa. The Fast Fourier Transform can convert a function in the time domain into the frequency domain, but the resulting function will tell us nothing about the time, it will only tell us about the frequency. We need both the Fourier Transform and its inverse to obtain as much information as we can about an object.

Generally, when receiving some sort of a sound signal, there is a lot of noise in the background. For example, if you're trying to receive a signal from a plane flying overhead, the information you receive will likely be clouded with noise from the environment. Thus, you will have a lot of noisy data with no way to determine which wave corresponds to the frequency of the plane and what time that plane was flying overhead.

There are ways around this. One method, known as Averaging, averages a signal in the frequency domain. It's great to use when you have multiple measurements of a signal. Averaging will purposely add white noise to a signal in the frequency domain. This noise, which has a mean distribution of zero and affects all types of frequencies in the same way, can be added any number of times to the signal. The method will then take this sum and average it- the more noise you add the larger the number you will divide by. The resulting signal in one dimension will contain a single distinguishable peak- the central frequency. This tells you the frequency of the object of interest.

Filtering is another way to go about denoising data. This method applies a Gaussian filter (Eqn. 3) to a function in the frequency domain.

$$G(k) = e^{-\tau(k-k0)^2} \quad (3)$$

The Gaussian filter takes in some value τ that is the width of the filter and can be adjusted by trail-and-error, and $k0$, the central frequency of the data. The key difference here compared to the Averaging method is that the central frequency of the data must already be known. The function in the frequency domain, or in this case the Fast Fourier Transformed function, is multiplied by the filter, causing everything outside of the range of τ on either side of $k0$ to go to zero. In one-dimensional space the result of this should be a single peak on a graph. Taking the inverse transform of this result can then allow us to plot our data in the time domain, where we will be able to see a filtered version of the original signal.

In practical applications, it may not be possible to use either of the analysis methods by themselves because you simply aren't given enough information to implement them alone. Thus, you must use both. For instance, by applying the averaging method on a function, you can obtain the central frequency and then use that in your filter function to denoise your data. In the next section, I will be discussing how I used both methods to help denoise the signal we received.

Sec. III. Algorithm Implementation and Development

Some code was provided that helped us begin the assignment (Appendix B: Lines 5-20, all comments are mine). This code defined an interval $x \in [-L, L]$ as steps going from -15 to 15. The interval was split into 2^6 points, and was then used to create x, y, and z axes representing the 3D position space. A vector of frequencies, k, was defined, and had to be scaled by $2\pi/L$ because the Fast Fourier Transform theorem only works for functions on a 2π domain. "$Undata$", representing all of the measurements from the ultrasound at each timestep, was reshaped into an array called Un (Appendix B: Line 16) with dimensions that were the same as the position ($X, Y\ and\ \ Z$) and frequency ($Kx, Ky,$ and Kz) arrays that were defined above. This made it easier to work with the position data in the next steps. The isosurface plot depicted the position of the marble over all twenty measurements, but as expected, it was covered with noise (Sec. IV., Figure 1). I needed to get rid of this noise to determine the true position of the marble.

My first step was to take the noisy position data (the array Un) and find the center frequency of the data, or the frequency the marble emits. This was done by the Averaging method. Because Un was in the time domain and thus giving positions of the marble, it needed to be converted to the frequency domain. To do this, I summed all of the position data and then applied the Fast Fourier Transform to this value. Then, I took the average of this transform, resulting in a matrix that represented the average frequency of all of the position data collected over the twenty timesteps (Appendix B: Lines 25-33).

This average frequency was normalized by dividing by the maximum value in the array and then plotted with the points $X, Y,$ and Z defined in the starter code (Appendix B: Line 34). The resulting isosurface plot depicted the averaged frequency, which contained the central frequency (Sec. IV., Figure 2). When plotting the averaged frequency in one dimension, the central frequency is clearly identifiable as the frequency that corresponds to the highest, most distinguishable peak in the graph. However, because we were working in three dimensions, it wasn't clear which coordinates corresponded to the highest point in the graph. I created a for loop to find the indexes of the highest average frequency values, or those that corresponded to the maximum value in the average frequency array (Appendix B: Lines 41-52). These index values were used in the next part, the Filtering method.

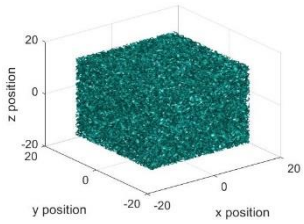
I defined my filter function similar to a standard Gaussian function. I had to take into account that we were working in three dimensions and essentially added the filter function to itself three times – one for frequencies in each of the three directions. I used the index values found before to obtain the central frequencies in the $x, y,$ and z directions from the arrays $Kx, Ky,$ and Kz defined in the started code (Appendix B: Lines 54-55). The value I obtained for τ was determined towards the end by trial-and-error, based on how clean my plot looked.

Once the filter function was created, it had to be applied to the function in the frequency domain, or the Fast Fourier Transform of the position data. An inverse Fast Fourier Transform was then applied to this result in order to convert the newly filtered frequency data back into position data. I performed these steps inside of a for loop that iterated through each timestep, so that the position data for each measurement of the marble could be filtered. Using the isosurface function, I was able to see where the marble was during all twenty measurements. To plot the path of the marble, I utilized the isosurface functions' capabilities to store the values of the

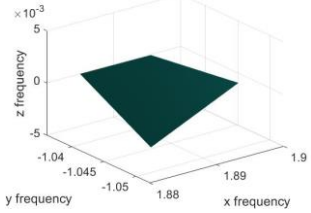
vertices of each point. For each iteration through the loop, I stored the first set of coordinates in the variable "v" into their respective x, y, or z vectors. Using the plot3 function, I was able to plot the approximate path of the marble using these vectors (Appendix B: Lines 54-77),(Sec. IV., Figure 3).

Sec. IV. Computational Results

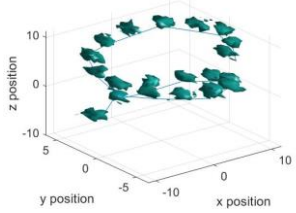
The central frequency, or frequency signature generated by the marble was located at the coordinates (-4.8171, 5.6549, -6.701) in the frequency space. This means that the marble was emitting a frequency at those specific locations in the x-, y- and z- frequency directions. The path of the marble showed that its 20th or last position was located at the coordinates (-4.688, 4.219, -7.096) in the position domain. Thus, if an intense wave was to be directed at that location, it would be able to break up the marble.



(1)



(2)



(3)

Figure 1: Plot of original unfiltered position data of the marble for all twenty measurements. Figure 2: Plot depicting average frequency for all twenty measurements. Figure 3: Plot depicting the approximate positions of the marble and the approximate path of the marble after filtering.

Sec. V. Summary and Conclusions

The Fast Fourier Transform seems to be an effective way to determine the frequency or position of a certain object and even filter out environmental noise that may be clouding the data. I was able to implement this theorem into two different methods (Averaging and Filtering) that helped denoise the data of an unreadable signal and produce a result that showed the position of the object I was looking for. The Fast Fourier Transform clearly shows strengths for applications requiring any kind of noise or position analysis.

Sec. VI. Sources/References

Kutz, Nathan J. (2013). *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*

"isosurface." *https://www.mathworks.com/help/matlab/ref/isosurface.html*. MathWorks, 2019.

Appendix A. MATLAB functions used and brief implementation explanation

$fftn(X)$ – Computes Fast-Fourier Transform of multidimensional object X.
$fftshift(Y)$ – Shifts a function Y, which is already the transformed function.
$ifftn(Y)$ – Takes the inverse Fast-Fourier Transform of a multidimensional object Y.
$isosurface(X,Y,Z,V,isovalue)$ – Plots the coordinates (X,Y,Z) of object V at a particular $isovalue$.
$[f,v] = isosurface(X,Y,Z,V,isovalue)$ –Stores the vertices (v) and the indexes of the vertices (f) of the isosurface plot.
$reshape(array,n,n,n)$ –Reshapes a matrix or array $array$ into an n x n x n matrix.
$find(condition)$ – Finds the indices at which a matrix or array meets a condition.
$max(array,[],'all')$- Finds the maximum value out of all the elements in an array.
$plot3(x,y,z)$- Plots three vectors or matrices $(x, y,$ and $z)$ as coordinates in three-dimensional space.
$meshgrid(x,y,z)$- Creates 3D grids based on the coordinates of $x, y,$ and z.

Appendix B. MATLAB codes

```
%loading test data file
clear; close all; clc;
load('Testdata.mat')
%% Starter code given by instructor
L=15; % defining spatial domain
n=64; % setting Fourier modes
x2=linspace(-L,L,n+1);
x=x2(1:n); %consider only the first n points
y=x; %y-axis will be defined the same way as the x-axis
z=x;%z-axis will be defined the same way as the x-axis
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1]; %k rescaled to 2pi domain
ks=fftshift(k); %shift Fourier transform
[X,Y,Z]=meshgrid(x,y,z); %x, y, and z axis
```

```
[Kx,Ky,Kz]=meshgrid(ks,ks,ks); %fourier transform shift of x, y and z axes on
Fourier transform
for j=1:20
Un(:,:,:)=reshape(Undata(j,:),n,n,n); %reshape Undata(j,:) into a nxnxn
matrix
close all, isosurface(X,Y,Z,abs(Un),0.4)
axis([-20 20 -20 20 -20 20]), grid on, drawnow
pause(1)
end
xlabel('x position')
ylabel('y position')
zlabel('z position')
set(gca,'Fontsize',16)
%%
test = zeros(64,64,64);
for jj = 1:20
    Un(:,:,:)=reshape(Undata(jj,:),n,n,n); %reshape Undata(j,:) into a nxnxn
matrix
    test = test+Un;
end
%% Averaging
ut = fftn(test); %taking FFT of the sum of all Un, that is 20 points
ave = abs(ut)/20; %averaging the fftshift of ut
isosurface(Kx,Ky,Kz,fftshift(ave)/fftshift(max(ave,[],'all')),.99)%Plotting
frequencies while scaling the average height.
xlabel('x frequency')
ylabel('y frequency')
zlabel('z frequency')
set(gca,'Fontsize',16)
grid on, drawnow
%Finding the central frequency
maxave = max(ave,[],'all');
for x1 = 1:64
    for y1 = 1:64
        for z1 = 1:64
            if ave(x1,y1,z1)==maxave
                k1 = x1;
                k2 = y1;
                k3 = z1;
            end
        end
    end
end
%% Filtering
tau = 2;
gfil  = exp(-tau*(((Kx-Kx(k1,k2,k3)).^2))) + exp(-tau*(((Ky-
Ky(k1,k2,k3)).^2))) + exp(-tau*(((Kz-Kz(k1,k2,k3)).^2)));
%Plotting the marble's positiion at each observation
Xvec = zeros(20,1);
Yvec = zeros(20,1);
Zvec = zeros(20,1);
for i = 1:20
Un(:,:,:)=reshape(Undata(i,:),n,n,n);
uft = gfil.*(fftn(Un)); %Apply filter to the signal in frequency space
ifftshift(ave)
uf = (ifftn(uft)); %Convert filtered signal back to time domain
isosurface(X,Y,Z,abs(uf),1)
```

```
[f,v] = isosurface(X,Y,Z,abs(uf),1); %Plotting path of marble against
filtered signal
Xvec(i,1) = v(1,1);
Yvec(i,1) = v(1,2);
Zvec(i,1) = v(1,3);
grid on, drawnow
hold on
end
xlabel('x position')
ylabel('y position')
zlabel('z position')
set(gca,'Fontsize',16)
%% Plotting the path
plot3(Xvec,Yvec,Zvec)
grid on, drawnow
xlabel('x position')
ylabel('y position')
zlabel('z position')
set(gca,'Fontsize',16)
```