AMATH 582 Homework 1: FFT Applications Including Filtering by Averaging and Gaussian Filtering

Gavin M. Chase

January 24, 2020

Abstract

This report seeks to clarify the use of fast Fourier transforms (FFT) to reduce noise in a given data set. Specifically, this report addresses frequency filtering through averaging a collection of data and the use of a Gaussian filters to isolate a stationary frequency moving in space. To demonstrate these methods, I searched through a highly noisy set of data for a marble travelling in a 3-dimensional space representing my dog's intestines. The marble's final position is $x = -5.156$, $y = 4.687$, and $z = -5.625$.

1 Introduction and Overview

1.1 Historical Significance of FFT

Fourier transforms seek to represent any function $f(x)$ as a trigonometric series of sines and cosines[2]. Thus, any representation of space or time can be transformed via this method to produce its spectral content. In 1965, Cooley and Tukey developed an algorithm that performs Fourier transforms at an operation count of $O(NlogN)$ [1][2]. This algorithm, known commonly as the fast Fourier transform (FFT), has proven highly useful in a number of signal processing applications because of it's speed. In an effort to gain a greater appreciation for FFT and related filtering methods, I applied them to the problem described in section 1.2.

1.2 The Marble Problem

I applied these methods to a problem concerning a marble. A fictitious fluffy swallowed a marble that has entered his intestines. The veterinarian takes 20 measurements with ultrasound of Fluffy's intestines to find it. Because Fluffy is uncomfortable, he would not stay still and generated enough white noise to mask the marble's position without some means of filtering.

It is worth mentioning the relevance of this particular problem given my actual dog, Trigger, is actively recovering from surgery to remove a struvite stone measuring 2" in diameter from her bladder. Like Fluffy, she is extremely jittery when visiting the veterinary hospital and caused the staff much consternation.

2 Theoretical Background

The Fourier series, shown in Equation 1, represents any function $f(x)$ as a trigonometric series of sines and cosines[2].

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

The Fourier transform and its inverse are

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

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

Like the Fourier Series, these transforms can be performed for the entire line $x \in [-\infty, \infty]$, but for computational purposes are only applied over the finite interval $x \in [-L, L]$. The solutions on this interval must have periodic boundary conditions because the integration kernel $\exp(\pm ikx)$ is oscillatory [2]. When used in this way, one can transform any real function $f(x)$ into the frequency domain using Equation 2, conduct any desired manipulations, and return the function back to its temporal or spatial form using Equation 3.

The FFT algorithm performs the transforms in Equations 2 and 3 with an operation count of $O(NlogN)$ because it shifts the data so that $x \in [0, L] \to [-L, 0]$ and $x \in [-L, 0] \to [0, L]$. It also multiplies every other mode by -1 and assumes a 2π periodic domain[2]. To maximize speed when using the FFT algorithm, one must discretize the range $x \in [-L, L]$ to 2^n points.

There are two filtering techniques closely linked with FFTs. The first method is averaging. If the particular signal one is interested in does not change, and the noise is white with zero mean, then averaging a set of signal detections will cause the noise values to approach zero while the signal remains intact[2]. Thus, even if the source of the signal moves in time or space and is impossible to identify from that perspective, it is still possible to identify the frequency once the data has been transformed into the frequency domain using Equation 2 or the FFT.

The second filtering method is the Gaussian filter: a filter designed to attenuate high frequencies and let the remainder pass.

$$\mathcal{F}(k) = exp(-\tau(k - k_0)^2) \tag{4}$$

In this Gaussian filter, τ determines the bandwidth of the filter and k is the wavenumber. One can optimize this filter to meet specific needs by adjusting the τ and k_0 values. Adjusting τ changes the width of the filter with smaller τ's resulting in wider filters. Adjusting k_0 changes the position of the filter. To perform this operation in three dimensions use

$$\mathcal{F}(k) = exp(-\tau(k_x - k_{x0})^2 + -\tau(k_y - k_{y0})^2 + -\tau(k_z - k_{z0})^2). \tag{5}$$

It is worth emphasizing the importance of filter positioning and optimization as it relates to decision making. Even if noise is white with zero mean, it is possible for a poorly placed filter to induce Type II error. In other words, it may return a strong signal that does not match what one is looking for. Adjusting the filter size allows users to set lower detection thresholds with greater confidence. If a filter is too narrow the filtered signal may accidentally be filtered out, which creates the illusion that no signal exists (Type I error). Converesly, a filter that is too wide will allow too much noise to remain, which can create false peaks and force the user to unnecessarily raise the decision threshold.

3 Algorithm Implementation and Development

Because the marble's position was masked by noise and the frequency signature was unknown, I started by identifying the frequency signature. This had three sub-steps: establish spatial and frequency domains, averaging the frequency content, and identify the peak frequency, which are shown in Algorithms 1, 2, and 3. They can also be found in Appendix B, Listing 1.

Algorithm 1: Establish Spatial and Frequency domains

Set the spatial domain L to 15
Set Fourier modes n to 2^6
Discretize the spatial domain from $-L$ to L with n points ensuring to account for boundary conditions
Duplicate spatial domain in y and z planes
Set frequency domain k on a 2π periodic domain using $k = \frac{2\pi}{2L} * [0 : (\frac{n}{2} - 1) - \frac{n}{2} : -1]$
Set new vector ks as the FFT shifted k

After visualizing the filtered signal, I set the detection threshold at 0.75 to find the frequency signature. I then created a Gaussian filter based on Equation 5, centered it on the wavenumbers identified in Algorithm 3, and set the bandwidth parameter to $\tau = 0.15$. This process is shown in the first three lines of Algorithm 4 and in Listing 2 of Appendix B. The resultant filter is depicted in Figure 1.

Algorithm 2: Averaging the Frequency Content

Create an initial matrix for the averaged signal as a collection of zeros
Import data from `Testdata.mat`
for $j = 1 : 20$ **do**
  Extract measurement j from `Undata`
  Perform FFT on measurement j in 3D
  Add transformed measurement to the previous averaged signal matrix
end for
Divide the FFT shifted sum of the measurements by j
Normalize the averaged signal by dividing |averaged signal| by the max value of |averaged signal|
Project the signal in a 3D ks space showing only points above the detection threshold 0.75

Algorithm 3: Identifying the Frequency Signature

Normalize the averaged signal by dividing |averaged signal| by the max value of |averaged signal|
Find the matrix position for the max value in the averaged signal matrix
Extract the `row`, `column`, and `page` positions for the max value matrix position
Scale `row`, `column`, and `page` positions as the k_x, k_y, and k_z positions of the Frequency Signature

After constructing the filter, I simply applied the filter to each measurement and collected the resultant positions in a vector describing the marble's path. This vector easily breaks into the x, y, and z components following the same method used in Algorith 3. The actual code for this step can be found in Listing 2 of Appendix B. To project the marble's position at each measurement, I applied a detection threshold of 0.5.

Algorithm 4: Filtering each Measurement's Frequency Content and Finding the Marble

Set bandwidth parameter $\tau = 0.15$
Create filter shown in Equation 5 with τ, k_x, k_y, and k_z positions as inputs
Project the filter in a 3D ks space
for j=1:20 **do**
  Extract measurement j from `Undata`
  Perform FFT on measurement j in 3D
  Multiply transformed measurement j by the filter
  Take absolute value of the inverse FFT of measurement j
  Add the matrix position for the max value of the filtered measurement j as a new value in a vector containing the marble's path
  Project the marble's position at measurement j showing only points above the detection threshold 0.5
end for
for j=1:20 **do**
  Extract the `row`, `column`, and `page` positions for all measurements from the "marble's path" vector
end for
Scale `row`, `column`, and `page` vectors as vectors containing x, y, and z coordinates for the marble
Project the collection of marble positions across time

4 Computational Results

As a reference point for the results, it is worth viewing Figure 4a to understand how powerful filtering and averaging with the FFT can be. This seemingly useless figure contains the marble's position at the 20^{th} measurement before filtering the frequency content.

After averaging the 20 measurements in the frequency domain, the frequency signature was found to be $k_x = 1.820$, $k_y = -1.021$, and $k_z = -0.007$. This is shown graphically in Figure 2 with a detection threshold

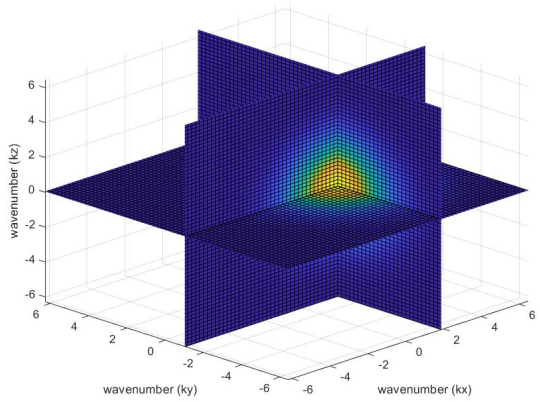


Figure 1: The Gaussian filtercreated in Algorithm 4 with $\tau = 0.15$

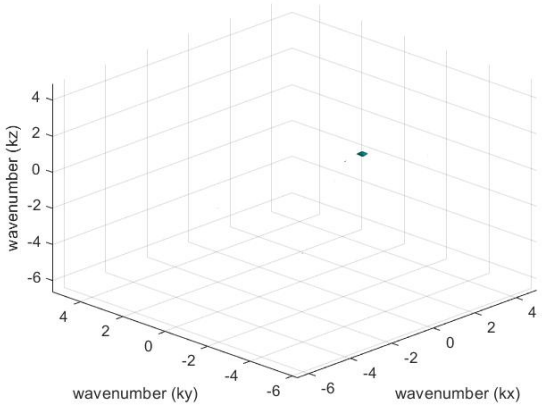


Figure 2: The frequency signature after averaging the measurements set to a detection threshold of 0.75.

of 0.75. This was then used to plot the filtered marble positions for each measurement. These positions can be found in Table 1, and are shown in Figure 3. The detection threshold for each marble position measurement was 0.5. The 20^{th} measurement at that threshold is shown in Figure 4b and was found to be $x = -5.156$, $y = 4.687$, and $z = -5.625$.

It is worth briefly noting that there is some inherent uncertainty in both the measurement of the frequency signature and the marble's position due to the discrete nature of the computations. Because the spatial domain L is divided into 2^6 discrete positions, the marble's position can only be found on one of those 2^6 positions. In the marble problem, each point happens to be 0.4688 from the next point in all directions. A similar problem exists for k but with a smaller gap of only 0.2094.

5 Summary and Conclusions

The purpose of this exercise was to demonstrate that if a particular signal is masked with white noise, applying FFT filtering methods can very efficiently, and with a high degree of confidence, reduce a noisy data set to something meaningful. The transformation shown in Figure 4 is clear evidence of this. Future iterations of this problem could include other filter types. This would expand my knowledge of filtering and allow me to select the most effective filter for a particular data set.

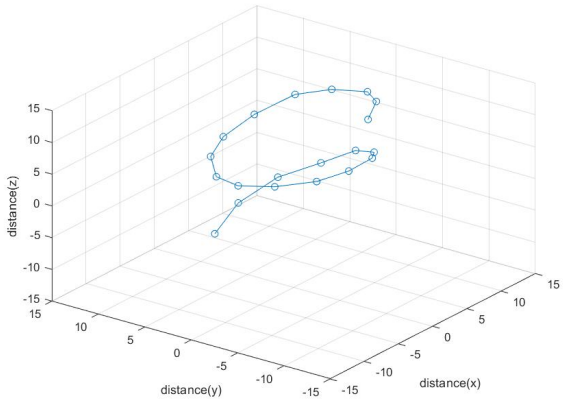


Figure 3: The marble's path across all 20 measurements.

	X Coord	Y Coord	Z Coord		X Coord	Y Coord	Z Coord
1	5.156	-4.218	10.312	11	-6.563	-2.812	5.625
2	8.906	-2.343	10.312	12	-2.344	-4.218	4.688
3	10.781	0	9.843	13	2.344	-4.218	3.750
4	9.375	2.812	9.843	14	7.031	-3.281	2.812
5	6.563	4.687	9.843	15	9.844	-1.406	1.406
6	1.875	5.625	8.906	16	10.313	0.938	0.468
7	-3.281	5.156	8.436	17	8.438	3.281	-1.406
8	-7.031	3.750	7.969	18	4.688	5.156	-2.343
9	-9.375	1.406	7.031	19	-0.469	5.625	-3.750
10	-9.375	-0.937	6.563	20	-5.156	4.687	-5.625

Table 1: Filtered marble positions across all 20 measurements

References

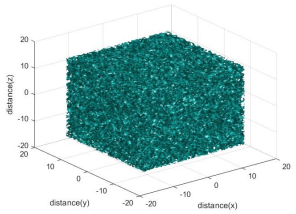
[1] James W Cooley and John W Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

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

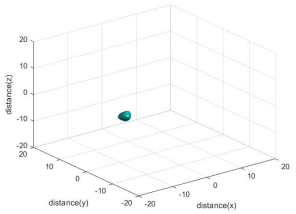
Appendix A MATLAB Functions

What follows is a list of MATLAB functions as they appear in Appendix B. Much of the text below with additional examples can be found in the Mathworks Documentation Library.

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `fftshift(x)` rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array. If X is a vector, then fftshift swaps the left and right halves of X. If X is a matrix, then fftshift swaps the first quadrant of X with the third, and the second quadrant with the fourth. If X is a multidimensional array, then fftshift swaps half-spaces of X along each dimension.
- `[X,Y,Z] = meshgrid(x,y,z)` returns 3-D grid coordinates based on the coordinates contained in the vectors `x`, `y` and `z`. X is a matrix where each row is a copy of `x`. The same is true for Y and Z. The grid is size `length(x)` by `length(y)` by `length(z)`.



(a) A seemingly useless 20^{th} measurement of the marble's position before filtering the frequency content.



(b) The marble's position on the 20^{th} measurement at a detection threshold of 0.5.

Figure 4: The transformation from noisy signal to filtered position

- `zeros(x)` creates a matrix of size `length x-length y-length z` containing only zeros.
- `reshape(x)` reshapes matrix V with size `length x-length y-length z`.
- `fftn(x)` detects the number of dimensions in x and conducts a multidimensional Fourier transform on `x` in the number of detected dimensions. It returns a matrix `y` with same size as `x`.
- `max(x, [],'all')` finds the maximum value in matrix `x` in all directions.
- `isosurface(X,Y,Z,V,isovalue)` computes isosurface data from the volume data V at the isosurface value specified in `isovalue` (isovalue can be a range). That is, the isosurface connects points that have the specified value much the way contour lines connect points of equal elevation.
- `slice(X,Y,Z,V,xslice,yslice,zslice)` draws slices in a Cartesian grid at locations in vectors `xslice`, `yslice` and `zslice` to show how the values change in `V`.
- `ifftn(x)` returns the discrete inverse Fourier transform using FFT. It detects the number of dimensions and conducts the transform in each dimension.
- `ind2sub(size,x)` returns the row, column, and page of `x` within a matrix of size `size`.
- `find(x)` returns the position of `x` within a matrix.
- `plot3(x,y,z)` plots vectors `x`, `y`, and `z`, as sets of `x`, `y`, and `z` coordinates, respectively. Thus, if each vector contains 20 values, the function will plot 20 points.

Appendix B MATLAB Code

This appendix includes the MATLAB code used to produce the results in this report. The code can also be found at `https://github.com/gchase15/AMATH-582`

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

%% Setup data
L=15; %spatial domain
n=64; %Fourier modes 2^6
x2=linspace(-L,L,n+1); %Space discretization
x=x2(1:n); %space I actually use
%(first point is the same as last and FFT works on periodic boundaries)
y=x; %the marble is moving in 3 dimensions
z=x;

k=(2*pi/(2*L))*[0:n/2-1 -n/2:-1]; %standard shifted frame
ks=fftshift(k);%shifting k to unshift it

[X,Y,Z]=meshgrid(x,y,z);
[Kxs,Kys,Kzs]=meshgrid(ks,ks,ks);

%% averaging the signal

Utave=zeros(n,n,n);
for j=1:20
   Un(:,:,:)=reshape(Undata(j,:),n,n,n);
   %this essentially takes each row of data and chops it into a cube
    Unt=fftn(Un(:,:,:));
    Utave=Utave+Unt;
end
Utave=fftshift(Utave)/j;
Utave=abs(Utave)/max(abs(Utave), [], 'all');
%scales everything as a ratio of the max value

 %if the max point in the cube is the signal,
 %after it is scaled to some portion of 1, the true signal
 %should lie around the area at or near the max of 1
 close all, isosurface(Kxs,Kys,Kzs,Utave, .75:1)
 %detection threshold is around .75
 axis([-7 7 -7 7 -7 7]), grid on, drawnow;
 xlabel ('wavenumber (kx)'), ylabel ('wavenumber (ky)'),zlabel ('wavenumber (kz)'),
 title('Averaged Signal')
 pause(2)

 freqpos=find((Utave/max(Utave,[],'all'))==1);
 [freqrow,freqcol,freqpage]=ind2sub(size(Utave),freqpos);

 xfreqpos=ks(1)+(freqcol/n)*(2*ks(64))%x wavenumber
 yfreqpos=ks(1)+(freqrow/n)*(2*ks(64))%y wavenumber
 zfreqpos=ks(1)+(freqpage/n)*(2*ks(64)) %z wavenumber
```

Listing 1: MATLAB code covering initial setup, averaging the spectral content, and determining the frequency signature.

```
 %% the filter

tau=0.15;
 filter=exp((-tau*(Kx-xfreqpos).^2)+(-tau*(Ky-yfreqpos).^2)+(-tau*(Kz-zfreqpos).^2));
 %gaussian filter centered on max value from averaging the noisy signal
 xslice=1.88;yslice=-1.05; zslice=0.05;
 %the cut marks to show the filter distribution
 close all, slice(Kxs,Kys,Kzs,fftshift(filter), xslice,yslice,zslice)
 axis([-7 7 -7 7 -7 7]), grid on, drawnow
 xlabel ('wavenumber (kx)'), ylabel ('wavenumber (ky)'),zlabel ('wavenumber (kz)'), title('Gaussian Fil

 %% finding the marble

 for j=1:20
   Un(:,:,:)=reshape(Undata(j,:),n,n,n);
   Unt=fftn(Un(:,:,:));
   Untf=filter.*Unt;
   Unf=abs(ifftn(Untf));
   path(j)=find((Unf/max(Unf,[],'all'))==1);
   close all, isosurface(X,Y,Z,Unf/max(Unf, [], 'all'), .5:1)
   %detection threshold of .5
   axis([-20 20 -20 20 -20 20]), grid on, drawnow
   xlabel ('distance(x)'), ylabel ('distance(y)'),zlabel ('distance(z)'),
   title('Marble Position')
   pause(.2)
 end

 %% plotting the marble's path
for j=1:20
 [mrow(j),mcol(j),mpage(j)]=ind2sub(size(Unf),path(j))
end
%this for loop deconstructs path into vectors containing x,y, and z
%coordinates
 xpath=-L+(mcol/n)*(2*L);% x positions
 ypath=-L+(mrow/n)*(2*L);% y positions
 zpath=-L+((mpage/n)*(2*L));% z positions
  plot3(xpath,ypath,zpath, '-o')
  grid on
 xlabel ('distance(x)'), ylabel ('distance(y)'),zlabel ('distance(z)'),
   title('Path of the Marble')
```

Listing 2: MATLAB code covering filtering and projecting the path of the marble.