HW 1: An Ultrasound Problem

Anamol Pundle

January 22, 2015

Abstract

The process and results of de-noising a dataset using a band-pass filter and time averaging are described in this report. The noisy data consists of a set of twenty measurements in time obtained using ultrasound directed towards a small area in the intestines of a canine specimen that has ingested a marble. The frequency of the signal reflected off of the marble is constant in time but noisy due to movement of the canid and internal fluid movement. The frequency signature of the marble is first identified by time averaging the frequency spectrum. A Gaussian filter in frequency domain is constructed using the center frequency, which is then applied to the noisy data in frequency domain at each time step. Conversion of the de-noised spectrum to time domain gives us the position of the marble at each time step. The trajectory of the marble is plotted, and the position of the focus of an intense acoustic wave to break up the marble is calculated.

1 Introduction and Overview

The Fast Fourier Transform (FFT) has revolutionized the field of signal processing. The key feature of this method is to use the FFT algorithm to decompose a signal into its frequency components, identify the desirable frequencies and attenuate the undesirable ones. In the problem at hand, we are provided with a set of twenty ultrasound measurements from the intestines of a dog, who has swallowed a marble. The data is noisy, due to the movement of the dog and its internal fluid movement. The goal is to de-noise this data and find the frequency signature of the set of signals, and subsequently the trajectory of the marble, so that an intense acoustic wave can be focussed on the marble in order to break it up at the final time step.

Several concepts are implemented in order to achieve this goal, including the FFT algorithm, averaging the frequency of the signals and Gaussian filtering. These are described in detail in the next section, followed by the a discussion of the algorithm used. The computational results of this exercise are then documented, followed by a list of MATLAB functions used in the computer program and the source code.

2 Theoretical Background

2.1 The Fourier Transform

The Fourier transform is a useful and efficient tool to solve several problems arising in the physical and biological sciences. The Fourier transform decomposes a signal in time into its frequency components. It is defined over the entire line, i.e., $x \in [-\infty, \infty]$ and is an integral transform. It is defined as

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

Its inverse is defined as

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

Our domain, however, is finite $x \in [-L, L]$, due to which the continuous eigenfunction expansion becomes a discrete sum of eigenfunctions and their eigenvalues. The Fourier transform has the added advantage of being linear. Another property of the Fourier transform is that a narrow signal in the time domain has a large spread in the frequency domain and vice-versa. Therefore, the Fourier transform provides no information about the positions in time of these frequencies. Techniques such as windowed Fourier transforms can be used to find both the frequency decomposition as well as the position of those frequencies in time.

2.2 The FFT Algorithm

The algorithms most frequently used to calculate the Fourier transform of a function (and its inverse) are known as the FFT Algorithm. The most common FFT algorithm is the Cooley-Tukey algorithm, developed by J.W. Cooley and John Tukey, which follows the general technique of divide and conquer. The algorithm divides the Fourier transform of the function into two smaller Fourier transforms recursively, resulting in a reduction of the computational time to $O(N\ logN)$. However, due to the recursive division by two, the algorithm requires the domain to be discretized into 2^N points. The algorithm also implies periodic boundary conditions on the boundaries, owing to the e^{ikx} term. The Cooley-Tukey FFT algorithm has excellent accuracy when compared to other discretization schemes, such as finite differencing.

2.3 Gaussian Filtering and Time Averaging

A commonly used technique used to filter data where the frequency spectrum does not change with time is Gaussian filtering. A Gaussian function centered on the frequency of choice is first constructed. The Gaussian function in one dimension is defined as

$$f(x) = A\ exp(-\left(\frac{(x-xo)^2}{2\sigma_x^2}\right) \tag{3}$$

where A is the amplitude, x_o is the centre and σ_x is the spread of the function. Multiplying this function with the frequency spectrum of the signal results in the attenuation of all frequencies away from the center of the Gaussian function and the isolation of the frequency near the center. The filter can be extended to higher dimensions by adding a term for each dimension inside the exponential function.

Time averaging the frequency spectrum of a signal over multiple time steps is an effective method to suppress white noise. White noise can be defined as a random signal equally distributed over all frequencies with zero mean and finite standard deviation. Since the mean of the signal is zero, adding a sufficient number of signals in the frequency domain should result in the white noise adding up to zero. This fact can be used to filter signals where the frequency of the signal is unknown but constant, as is done in this report. Once the frequency distribution of the signal has been isolated by time averaging, a Gaussian filter around the frequency can be constructed and applied to the all signals used in time averaging.

3 Algorithm Implementation and Development

This section lists the algorithm used to isolate the frequency signature of the signal, attenuate noise in the data and find the trajectory of the marble.

- Open `testdata.mat`
- Define domain size and grid vectors
  - The grid vectors for the frequency domain need to be multiplied by $\frac{2\pi}{L}$ since the FFT algorithm assumes 2π periodic signals. These grid vectors also need to be shifted using the `fftshift` function, since the FFT algorithm shifts the data such that $x \in$ [-L, 0]$\rightarrow$[0,L] and $x \in$ [0, L]$\rightarrow$[-L,0].
- Construct cartesian grid in 3D using `meshgrid`
- Reshape 1D data from `testdata.mat` into 3D using `reshape`
- Fourier transform each data signal using the `fftn` function and add together to variable `Utave`
- Normalize `Utave` using largest element in matrix. Take absolute value and `fftshift`. Visualize using `isosurface`
  - The absolute value of `Utave` needs to be taken because the FFT multiplies every other mode by -1.

- Find center frequency and its spread.
- Construct a 3D Gaussian function centered on the center frequency with spread covering the frequency signature.
- Multiply each signal in Fourier space by the Gaussian filter.
- Take the inverse FFT of each signal.
- Use `fftshift` to shift the data, normalize using the largest element in the matrix and the take the absolute value.
- Visualize using `isosurface` and `plot3` functions.
- Find the coordinates of position of the marble at the last time step.

The Gaussian filter used for the problem is

$$f(Kx, Ky, Kz) = exp\left(-\frac{\left((Kx - 1.885)^2 + (Ky + 1.047)^2 + Kz^2\right)}{2}\right) \quad (4)$$

where Kx, Ky and Kz are variables in the frequency domain. The center of the Gaussian function is found from the frequency signature, and the spread is found heuristically.

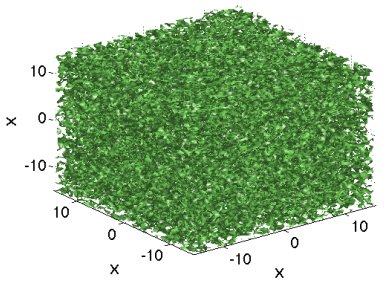
4 Computational Results

From the code stub provided with the problem, the spatial resolution of the ultrasound scanner is found to be 0.46875(units not mentioned) and the spectral resolution is found to be 1.000. Figure 1 shows an isosurface visualization (with threshold 0.7) of noisy data in both time and frequency domain. No discernible signal can be observed.

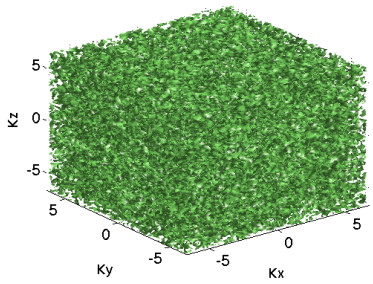
Figure 2a shows the frequency signature of the signals after time averaging. The frequency signature is concentrated at $(9, -5, 0)$ (after multiplying by $\frac{L}{2\pi}$). A visualization of the Gaussian function constructed at this center is shown in Figure 2b.

Figure 3a shows an isosurface visualization of the trajectory of the marble, obtained after applying the Gaussian filter to the signal at each time step. The marble seems to moving in a downward helical motion. Figure 3b shows the motion of the centre of the marble using the plot3 function.

Figure 4 shows an isosurface visualization of the final position of the marble. The final position is found to be $(-5.625, 4.2188, -6.0398)$.

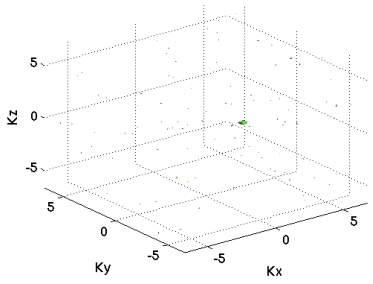


(a) Isosurface visualization in time domain

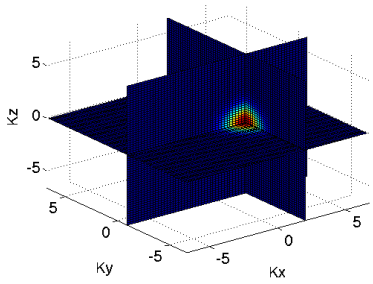


(b) Isosurface visualization in frequency domain

Figure 1: Visualization of noisy data signal

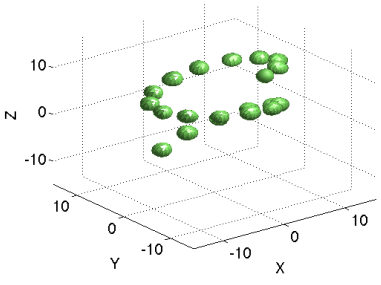


(a) Frequency signature of signals

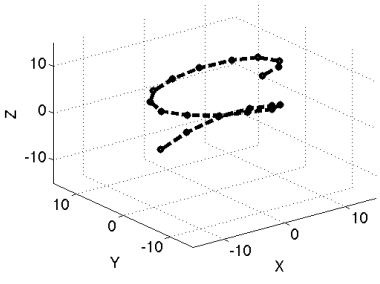


(b) Gauss Filter centered on center frequency

Figure 2: Frequency distribution of signal and Gaussian Filter



(a) Isosurface visualization



(b) Plot3 visualization

Figure 3: Visualization of trajectory of marble

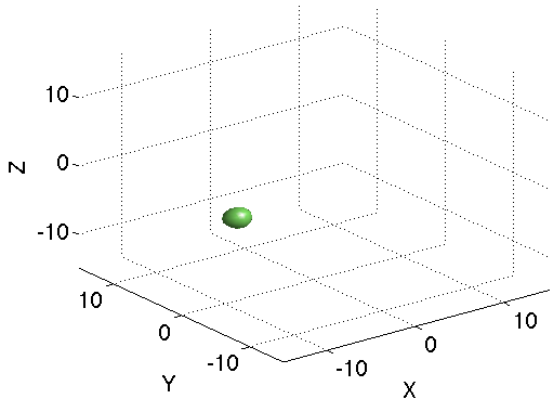


Figure 4: Isosurface visualization of final position of marble

5 Summary and Conclusions

The techniques of gaussian filtering and time averaging were successfully applied to the problem of de-noising data in 3D space. The frequency signature of the signals was identified by averaging the signals obtained through successive measurements. A gaussian filter was constructed around the frequency signature and applied to every signal. The trajectory of the marble was thus found, and the position of the marble at the final time step was found to be $(-5.625, 4.2188, -6.0398)$.

References

1. Kutz, J. Nathan, Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data, September 2013
2. http://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm
3. http://en.wikipedia.org/wiki/Fourier_transform
4. Matlab help

Appendix I: MATLAB Functions Used

`linspace`: generates linearly spaced vectors.

- `linspace(X1, X2, N)` generates N points between X1 and X2.

`meshgrid`: creates cartesian grid in 2D/3D space using grid vectors.

- `[X,Y,Z] = meshgrid(xgv,ygv,zgv)` replicates the grid vectors xgv, ygv, zgv to produce the coordinates of a 3D rectangular grid (X, Y, Z). The grid vectors xgv,ygv,zgv form the columns of X, rows of Y, and pages of Z respectively.

`zeros`: creates a zeros vector/matrix

- `zeros(M,N)` or `zeros([M,N])` is an M-by-N matrix of zeros.

`max`: finds the largest element in a vector/matrix

- `[Y,I] = max(X)` returns the indices of the maximum values in vector I.

`fftn`: N-dimensional discrete Fourier transform

- `fftn(X)` returns the N-dimensional discrete Fourier transform of the N-D array X.

`ifftn`: N-dimensional inverse discrete Fourier transform

- `ifftn(F)` returns the N-dimensional inverse discrete Fourier transform of the N-D array F.

`fftshift`: shifts data such that $x \in$ [`-L`, `0`]→[`0,L`] `and` $x \in [0, \mathrm{L}]\rightarrow$[-L,0]

- For vectors, `fftshift(X)` swaps the left and right halves of X.

`abs`: finds absolute value

- `abs(X)` is the absolute value of the elements of X.

`reshape`: reshapes array

- `reshape(X,M,N)` or `reshape(X,[M,N])` returns the M-by-N matrix whose elements are taken columnwise from X.

`isosurface`: Extracts an isosurface from a volume

- `FV = isosurface(X,Y,Z,V,ISOVALUE)` computes isosurface geometry for data V at isosurface value `ISOVALUE`. Arrays (X,Y,Z) specify the points at which the data V is given. The struct FV contains the faces and vertices of the isosurface

`plot3`: Plots lines and points in 3-D space.

- `plot3(x,y,z)`, where x, y and z are three vectors of the same length, plots a line in 3-space through the points whose coordinates are the elements of x, y and z.

Appendix II: MATLAB Code

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

L=15; % spatial domain
n=64; % Fourier modes

% define grid vectors, create 3D cartesian grid
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); % variable for averaged frequency

% calculate average of frequencies over all 20 signals
for j=1:20
      Un(:,:,:)=reshape(Undata(j,:),n,n,n);
      Utave = Utave + fftn(Un);
end

Utave = abs(fftshift(Utave)); % shift and take abs value of averaged

                              %frequency data

% find index of center frequency in Utave
maxi = 0;
for ii = 1:64
      for jj = 1:64
            for kk = 1:64
                  if Utave(ii, jj, kk) > maxi
                        maxi = Utave(ii, jj, kk);
                        a = ii; b = jj, c = kk;
                  end
            end
      end
end

% isosurface visualization of frequency signature
figure(1)
isosurface(Kx,Ky,Kz,abs(Utave)/max(Utave(:)),0.6, 'r');
set(gca, 'FontSize',18);
axis([ks(1) -ks(1) ks(1) -ks(1) ks(1) -ks(1)]), grid on;
xlabel('Kx'); ylabel('Ky'); zlabel('Kz');
```

```
% create Gauss filter
gaussfilter = exp(-((Kx-ks(b)).^2 + (Ky-ks(a)).^2 + (Kz-ks(c)).^2)/2);

% slice visualization of Gauss filter
slice(Kx,Ky, Kz, gaussfilter, 2,-1,0);
set(gca, 'FontSize', 20);
axis([ks(1) -ks(1) ks(1) -ks(1) ks(1) -ks(1)]);
xlabel('Kx'); ylabel('Ky'); zlabel('Kz');

% Apply Gauss filter to all 20 signals
maxi = 0; M = [];
for j = 1:20
    Un(:,:,:)=reshape(Undata(j,:),n,n,n);
    Utn = fftshift(fftn(Un));
    Newt = Utn.*gaussfilter;
    New= ifftn(Newt);

    % find index of maximum element in each 3D matrix
    for ii = 1:64
        for jj = 1:64
            for kk = 1:64
                if abs(New(ii,jj,kk)) > maxi
                    maxi = abs(New(ii,jj,kk));
                bb = X(1,ii,1); aa = Y(jj,1,1); cc = Z(1,1,kk);
                end
            end
        end
    end

    maxi = 0;
    M = [M ;[aa bb cc]]; % store indices of maximum elements
                              % of all 20 time steps in M

    % isosurface visualization of trajectory of marble
    figure(2)
    isosurface(X, Y, Z, abs(New)/max(abs(New(:))), 0.7);
    axis([-L L -L L -L L]), grid on;
    set(gca, 'FontSize',20);
    xlabel('X'); ylabel('Y'); zlabel('Z');
    pause(0.1);
end

% visualization of trajectory of marble using plot3
figure(3)
plot3(M(:,1), M(:,2), M(:,3),'k--o', 'LineWidth', 5);
```

```
axis([-L L -L L -L L]), grid on;
set(gca, 'FontSize',20);
xlabel('X'); ylabel('Y'); zlabel('Z');

% isosurface visualization of final position of marble
figure(4)
isosurface(X, Y, Z, abs(New)/max(abs(New(:))), 0.7);
set(gca, 'FontSize',20);
xlabel('X'); ylabel('Y'); zlabel('Z');
axis([-L L -L L -L L]), grid on;
```