3D Data Denoiser: An Ultrasound Problem

Peter Schultz
January 25, 2019

Abstract

This script determines important frequencies from a set of noisy spatial data at different time points, and filters the noisy data to extract useful information. The data arises from an ultrasound of my dog, who has swallowed a marble. We intend to reveal the path of the marble using filtering methods, and subsequently break up the marble using high intensity ultrasound.

I. Introduction and Overview

Filtering is a process that is ubiquitous in today's technology. Some examples include radar, audio recordings, medical imaging, etc. Herein we demonstrate a common process for filtering noisy data.

II. Theoretical Background

The Fourier transform is an extremely powerful tool that allows us to transform a set of time data into frequency data. We can subsequently remove unwanted frequencies (noise) from the data by constructing a filter which multiplies certain values by 0 or a small value. The Fourier transform is defined as the following:

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

In this equation, *k* represents the wave numbers. This operation is linear, which allows us to transform data, and inverse transform it to get back the original data (possibly with some modifications) trivially. An important assumption made in the procedure used in this script is that the noise that is present in the time data is random and normally distributed. This allows us to average the frequency data that is acquired through transforming the initial spatial data set, and pick out the important, persistent, frequency.

III. Algorithm Implementation and Development

The general method used is this:

1. Determine important frequencies in data
2. Compute frequency data using the Fourier transform

3. Construct filter that retains important frequencies and minimizes extraneous frequencies
4. Multiply frequency data with filter
5. Inverse transform filtered spectrum to get result

In this problem, we are given a set of 20 timepoints, each containing 3D spatial data of size 64x64x64. Initially, we create our axes data. Since each element in the 64x64x64 matrix needs an x, y, and z component, a 64x64x64 matrix must be created for each of the three dimensions. The same is true for the spectrum axes.
The function used to perform the Fourier transform in MATLAB is *fft()*, which stands for fast Fourier transform. This function performs the transform in an efficient way, with some downsides. The frequency result of the *fft* is always “shifted” - meaning that the two halves of the data are reversed with respect to the center. For instance, a vector that has [1 2 3 4 4 3 2 1] as the expected result, would instead be calculated as [4 3 2 1 1 2 3 4]. To mitigate this, the *fftshift()* command is used, which corrects the shift. Additionally, when creating the spectrum axes, the values must be multiplied by $\frac{2\pi}{2L}$, since MATLAB assumes that the input data is 2π periodic, and the limits are from $-\frac{L}{2}$ to $\frac{L}{2}$.
The first step is to transform each time point to the frequency domain and sum the resulting spectrums. Doing this will cause the frequencies from random noise to converge to zero, and cause the only important frequency, which will persist in each time point, to be the maximum. After computing the important frequency, a filter is generated, which is centered on the important frequency.

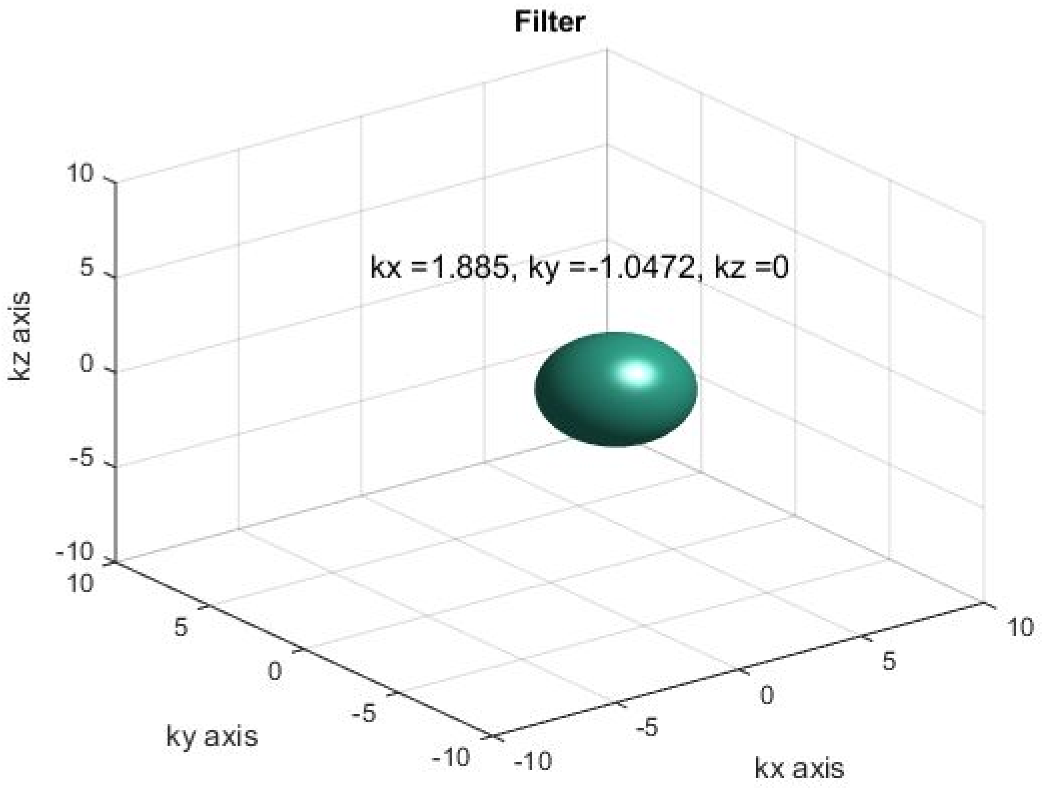


Figure 1: The filtered used in this assignment, centered around the important frequencies.

The filter used is shown in Figure 1 above; the center frequency generated by the marble is labelled on the plot. The function used to create this filter is:

$$filter = e^{-(a(k_x - k_1) + a(k_y - k_2) + a(k_z - k_3))}$$

Where $k_{x,y,z}$ corresponds to the frequency in the *x, y, z* direction at the current index, and $k_{1,2,3}$ corresponds to the center coordinates of the important frequency identified in the first step of the algorithm, and *a* is a width parameter. After the filter is created, it must be fftshift-ed, this is because when the data sets are transformed, the MATLAB fft function computes the spectrum in such a way that the result is shifted. Thus, the filter must be shifted to match the spectrum before they are multiplied together.

Next, this filter can be applied to each time point to, ultimately, find the location of the marble at that time. To do this, each data set is transformed into the frequency domain, multiplied by the filter, and inverse transformed back to the time domain. This allows us to pick out the location of the marble at each time point, which will be the maximum value in the data set.

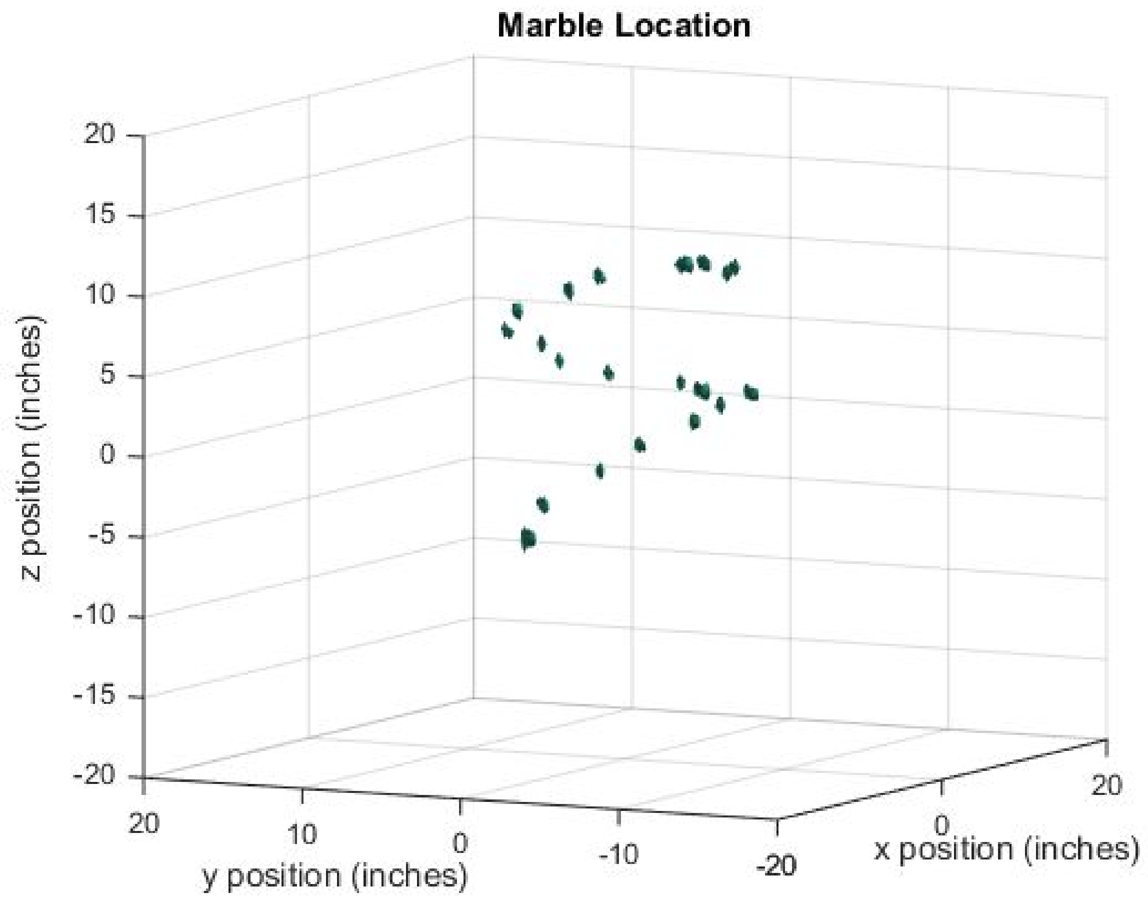
IV. Computational Results

The center frequency generated by the marble: $[kx, ky, kz] = [1.8850, -1.0472, 0]$, the units of these values are cycles/inch, assuming the original spatial data is in inches. The results of the procedure described above are shown below in Figure 2.

The path of the marble seems to be a downward spiral, coaxial with the z axis. The location of the marble at t=20: $[x, y, z] = [-5.625, 4.219, -6.094]$, where the units are inches.

Figure 2 was created by plotting the isosurface of each timepoint on the same plot. The isovalue was adjusted to be just below the max value in each data set (timepoint), by normalizing each dataset by its maximum.

Figure 2: Illustration of marble positions at 20 time points. The t=1 is at the top, and the points continue to t=20 at the bottom.



V. Summary and Conclusions

The filtering method described in the algorithm section was well suited for this data set; the result shown in Figure 2 is clear and noise-free. To break up the marble at t=20, the high intensity sound waves should be directed at the location described at the end of the previous section.

Appendix A MATLAB functions used and brief implementation explanation

fftn() - Computes fast Fourier transform, on N- dimensional data.
fftshift() - Shifts transform result.
ifftn() - Computes inverse transform, on N- dimensional data.
meshgrid() - Creates coordinates of a N-D grid.
reshape() - Rearranges a vector into specified shape/size.
ind2sub() - Converts an index of a vector to subscripts of a matrix of given size/shape.
isosurface() - Plots a surface in 3D with given isovalue.
max() - Determines max value in matrix, and index if warranted.