Save Fluffy

Aayush Chhabra

Abstract

We collected 3-D ultrasound data from the intestine of our dog (fluffy) who has eaten a stone. To extract meaningful information, I took the Fourier transform of this data and removed noise by averaging the signals across different time. Further, I designed and applied a Multivariate Gaussian filter (based on central frequency) on this data and inverted the data back to spatial domain to find the location of this stone. After finding the location of this stone at the final time, we used an acoustic wave to bombard the stone and save fluffy.

Introduction and Overview

Our dog fluffy has swallowed a marble which is now in its small intestine. We used ultrasound and obtained the data to find the location of stone in fluffy's intestine. Unfortunately, fluffy keeps moving and the internal fluid movement through the intestines generates highly noisy data.

With our ultrasound, we have recorded 20 different measurements in time. Each of these measurements describe the Ultrasound data as a 64 x 64 x 64 matrix representing spatial domain. To remove the noise, we are going to transform this data from spatial domain at each time to frequency domain. The benefit of taking everything in frequency domain is that now we don't have any time varying effects in our data. Further, we will take average of our data, as a result of which, noise in the data will start approaching zero. From this denoised information, we will find a central frequency around which we are going to build a filter. For this work, we are using a Multivariate Gaussian distribution as our filter. We will iterate through each time step in our data, transform data into frequency domain, take an element-wise product with the filter, and invert our fourier transform to find the positition of the stone.Finally, we will record our result for the 20^{th} time step which will be the position where we hit the acoustic wave.

Theoretical Background

0.1 Fourier Series

Fourier series is a way to represent functions in one domain as a sum of elementary functions in another domain. The most important Fourier series concern physical quantities with inverse relations like frequency and time.

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n cos(nx) + b_n sin(nx)$$
$$x \in [-\pi, \pi)$$
$$n \in Z \tag{1}$$

To find the coefficients of this expansion, we are going to use the orthogonality properties of trigonometric functions. After multiplying with the appropriate function and integrating, we find the coefficients in equation (1):

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)cos(mx)dx \qquad n \geq 0$$
$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)sin(mx)dx \qquad n > 0$$

0.2 Fourier Transform

Fourier transform is an extension of the Fourier series, where instead of just decomposing a signal into integral frequencies, we perform a continuous frequency decomposition. This transform comes along with its inverse and helps in transforming between time and frequency domains.

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

0.3 FFT

Fast Fourier Transform is an algorithmic implementation of the Fourier transform which operates in linearithmic time. This is one of the most important algorithms that has ever been developed and is at the heart of modern digital signal processing. In our work, we will be using the MatLab implementation of this algorithm. For reasons of efficieny, MatLab changes the relative position of coordinates in the input vector/matrix. Therefore, before plotting this data we will use the **fftshift** command to undo this change. Another important assumption made in this implementation is that the given function is periodic on a 2π interval. Look at Appendix B for more detailed information about the various MatLab functions used in our computation.

Algorithm Implementation and Development

1. Initial Data

We begin our analysis by looking at the initial noisy ultrasound data in the spatial domain. In the figure 1, we can see that there is nothing evident from this information and we will need to use some data analysis tools to remove the noise before proceeding further. Since, the reason for noise in our data is that fluffy is moving with time, we will use Fourier transform at this point to remove all the time information from our data.

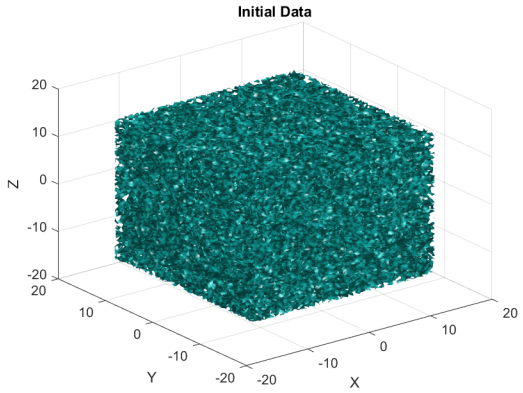

Figure 1: Initial Ultrasound Data - Spatial Domain

2. Fourier Transform

We will now iterate through each time step and Fourier transform the received signal. Then, we will add all the 20 signals in the frequency domain and normalize them. As a result, the noise in the data will approach zero and we will be able to see the information clearly. Plotting this information (Figure 2) and looking at relatively higher isovalues for the isosurface, we can see the information becoming more clear. Figure 3 shows the cross sectional view of the isosurface at isovalue = 0.6 from which we can find the central frequency of the data. To find the exact peak of this data, we used our MatLab code and found K_x = 1.8850, K_y = -1.0472, K_z = 0. These values clearly justify the obtained figure and will serve as the frequency around which we build our filter.

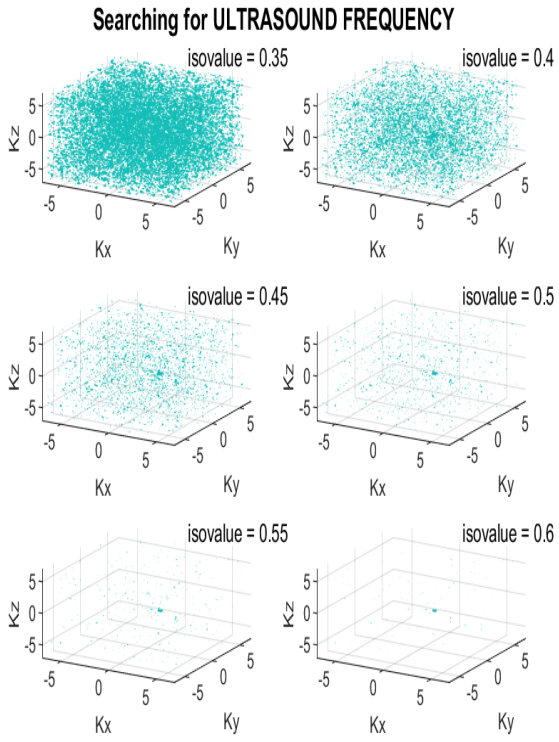

Figure 2: Averaging Signal in the frequency domain

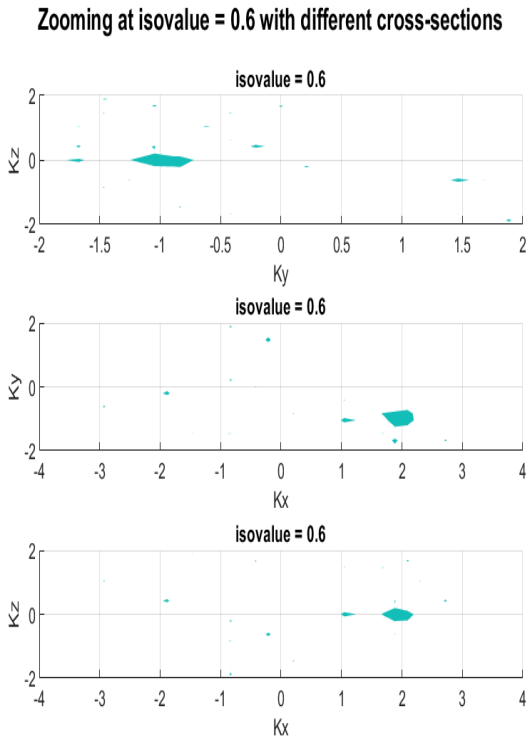

Figure 3: Cross Sectional View of central frequency

3. Filter Design

Now that we know the central frequency of the data, we can start designing a filter around this frequency to extract the location of the stone from this data. For this work, I decided to go with a Multivariate Gaussian Distribution with its mean located at the Central Frequency. The co-variance matrix for this distribution is a hyper-parameter that can be tuned within a few iterations as the data is not too sensitive to this information. I also kept equal variance in each direction to avoid any bias (we have no reasons to encode any bias). Figure 4 shows an isosurface of this filter.

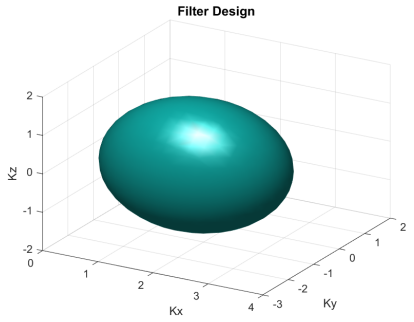

Figure 4: Filter Design: Isosurface View

4. Filter Applied on average

In the second step of this process, we took the Fourier transform of our signals and averaged them in the frequency domain. Now, we will apply our filter to this average signal in the frequency domain. This doesn't help us to predict the path/location of the stone (as this is the average signal) but it does help us to see whether our signal is working. This is an important insight to build confidence in our further analysis. From figure 5, we can see that the filter clears out almost all of the noise and leaves a significant trace of the stone.

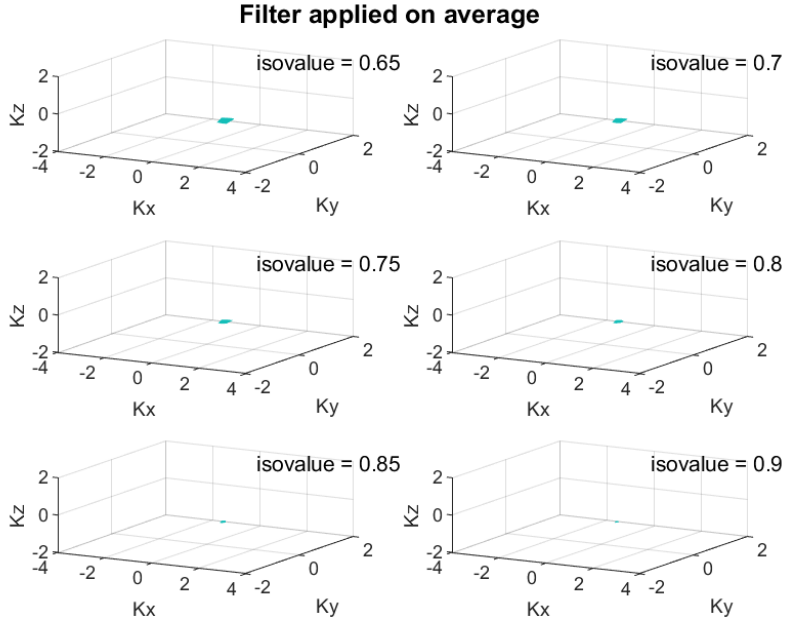

Figure 5: Filter applied on average signal

5. Filter Applied at each Time Slice

For our final step of the analysis, we will iterate through each slice of time and take Fourier transform of the sensor data at that time. In the frequency domain, we will perform an element-wise multiplication of the data with our filter to remove the noise from the data. After denoising the signal, we will invert it back to the time domain with spatial information. We will find the peak in this inverted signal and this peak is the location of stone at that instance of time.

Figure 6 shows the path of the stone which is moving through fluffy's intestine. Each subplot in this figure progressively shows the path after processing some number of time slices to present a sense of direction in which the stone is moving.

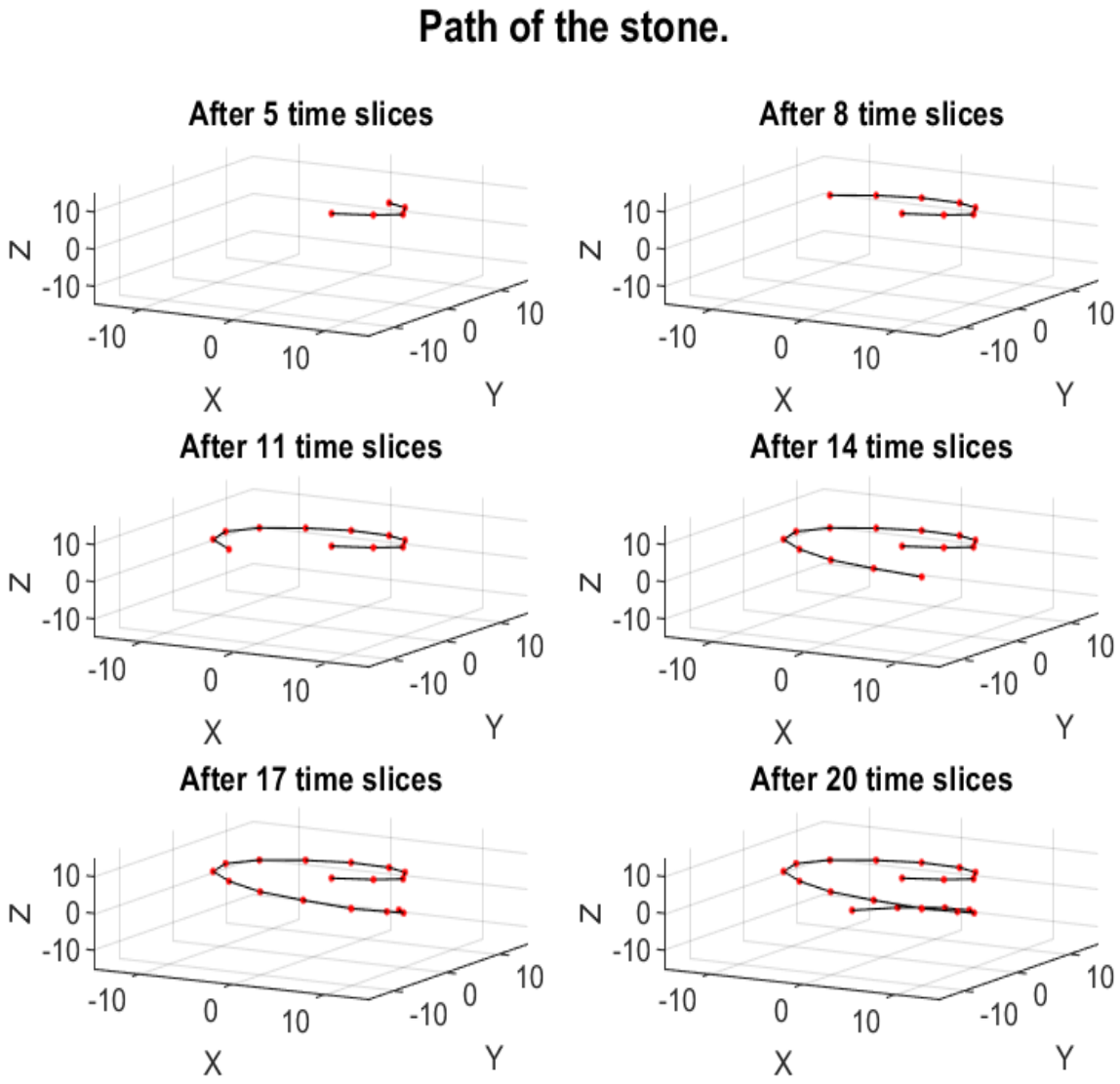

Figure 6: Path of the stone as a function of time.

Computational Results

From our analysis, we were able to find the path of the stone as it is moving through fluffy's intestine. Figure 7 shows the path of the stone after 20 time slices. After obtaining the entire path of the stone, we now need to find the final position of the stone at the 20^{th} time so that we can apply an acoustic wave to destroy the stone. With the help of our MatLab code, we can query the coordinates of the final position which turns out to be X = -5.6250, Y = 4.2188, Z = -6.0938.

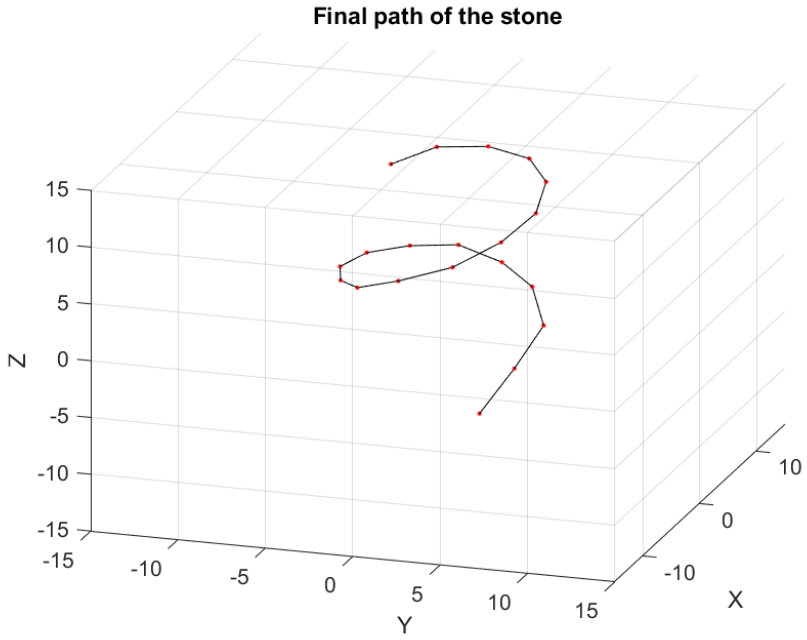

Figure 7: Final path of the stone.

Summary and Conclusions

We started with raw data from an ultrasound sensor. This data contained lot of noise because of the movement from internal fluids. To remove this noise, we took the Fourier Transform of the data and averaged the signals in the frequency domain. This averaging reduced the noise and we were able to pick a central frequency from the data. We then constructed a Multivariate Gaussian Filter around this central frequency. After designing the filter, we iterated through each slice of time, used fftn command to transform the data to frequency domain, applied the filter in frequency domain by element-wise multiplication, and inverted the data back with the ifftn command. This gave us the path of the stone moving through the fluff's intestine. Then we looked at the coordinates of the final position of the stone which we found out was X = -5.6250, Y = 4.2188, Z = -6.0938. An acoustic wave was applied at these coordinates to save fluffy. Fluffy is very happy now.

Appendix A

MATLAB functions glossary (official documentation):

Y = fftn(X) : returns the multidimensional Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D transform is equivalent to computing the 1-D transform along each dimension of X. The output Y is the same size as X.

Y = fftshift(X) : rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array. In our work, we have used this for transforming data before plotting it.

X = ifftn(Y) : returns the multidimensional discrete inverse Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D inverse transform is equivalent to computing the 1-D inverse transform along each dimension of Y. The output X is the same size as Y.

X = ifftshift(Y) : rearranges a zero-frequency-shifted Fourier transform Y back to the original transform output. In other words, ifftshift undoes the result of fftshift.

fv = isosurface(X, Y, Z, V, isovalue) : computes isosurface data from the volume data V at the isosurface value specified in isovalue. That is, the isosurface connects points that have the specified value much the way contour lines connect points of equal elevation.

Y = linspace(x_1, x_2, n) : generates n points. The spacing between the points is $\frac{(x_2 - x_1)}{(n-1)}$.

[X,Y,Z] = meshgrid(x, y, z) : returns 3-D grid coordinates defined by the vectors x, y, and z. The grid represented by X, Y, and Z has size length(y)-by-length(x)-by-length(z).

y = mvnpdf(X, mu, sigma) : returns pdf values of points in X, where sigma determines the covariance of each associated multivariate normal distribution.

APPENDIX B

Matlab Code

```
clear; close all; clc;

% Load the data and prepare it for further analysis
load Testdata
L=15; % spatial domain
n=64; % Fourier modes
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);

% Create mesh for spatial domain and freq domain
[X,Y,Z]=meshgrid(x,y,z);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);

% Since fluffy (the dog) is moving while ultrasound,
% we will fourier transform our signal. This will remove
% all the spatial information and leave us with just
% the freq information.

% To denoise the signal (noise due to the moving fluid),
% we will average the signal in the frequency domain.
% White noise will start shrinking and we will be able
% to find the dominant frequency in this way.
U_noisy_fft_avg = zeros(64,64,64); % Stores the avg in freq-domain
for j = 1:size(Undata,1) % go through each slice of time
    U_noisy(:,:,:) = reshape(Undata(j,:),n,n,n);
    % fft and add for avg
    U_noisy_fft_avg = U_noisy_fft_avg + fftn(U_noisy);
end
% Normalize for making average.
U_noisy_fft_avg = U_noisy_fft_avg/size(Undata,1);

% Let us now look at the data in freq domain
% so that we can find the freq of our ultrasound.
fig = figure(1);
title("Searching frequency")
iter = 1;
% only for plotting purposes
U_noisy_fft_avg_shift = fftshift(U_noisy_fft_avg);
for j = .35:.05:.6
    sub = subplot(3,2,iter);
    isosurface(Kx,Ky,Kz, ...
        abs(U_noisy_fft_avg_shift)/max(abs(U_noisy_fft_avg_shift(:))),j);
    axis([-7 7 -7 7 -7 7]); grid on; drawnow;
    text(2,4,10, strjoin(["isovalue =", j]))
    xlabel('Kx')
    ylabel('Ky')
    zlabel('Kz')
    view(30,30)
    iter = iter + 1;
end
```

```
sgtitle('Searching for ULTRASOUND FREQUENCY', 'FontSize', 12,...
    'FontWeight', 'bold');
print(fig, '-dpng', 'fig1')
%% Zooming to find out frequency

fig = figure(2);

subplot(3,1,1)
isosurface(Kx,Ky,Kz, ...
        abs(U_noisy_fft_avg_shift)/max(abs(U_noisy_fft_avg_shift(:))),j);
axis([-4 4 -2 2 -2 2]); grid on; drawnow;
title(strjoin(["isovalue =", j]))
xlabel('Kx')
ylabel('Ky')
zlabel('Kz')
view(90,0)

subplot(3,1,2)
isosurface(Kx,Ky,Kz, ...
        abs(U_noisy_fft_avg_shift)/max(abs(U_noisy_fft_avg_shift(:))),j);
axis([-4 4 -2 2 -2 2]); grid on; drawnow;
title(strjoin(["isovalue =", j]))
xlabel('Kx')
ylabel('Ky')
zlabel('Kz')
view(0,90)

subplot(3,1,3)
isosurface(Kx,Ky,Kz, ...
        abs(U_noisy_fft_avg_shift)/max(abs(U_noisy_fft_avg_shift(:))),j);
axis([-4 4 -2 2 -2 2]); grid on; drawnow;
title(strjoin(["isovalue =", j]))
xlabel('Kx')
ylabel('Ky')
zlabel('Kz')
view(0,0)

sgtitle('Zooming at isovalue = 0.6 with different cross-sections', 'FontSize',
    12,...
    'FontWeight', 'bold');
print(fig, '-dpng', 'fig2')

%% Filter Design

% After averaging out most of the noise and gradually increasing
% the isovalue, we can see that the dominant frequency.

% Now that we know the dominant frequency in the data, we will
% build a filter and extra act the location of the stone.

% We will now find the mean freq for the filter
[~,b] = max(abs(U_noisy_fft_avg_shift(:)));
mu_x = Kx(b);
mu_y = Ky(b);
```

```
mu_z = Kz(b);

% For our purposes, let's use a gaussian filter.
fig = figure(3);
mu = [mu_x mu_y mu_z];
sig = 1;
% width of the filter is a hyperparameter:
sigma = [sig 0 0; 0 sig 0; 0 0 sig];
filter = mvnpdf([Kx(:) Ky(:) Kz(:)],mu,sigma);
filter = reshape(filter,length(Kz),length(Ky),length(Kx));
isosurface(Kx,Ky,Kz,filter)
axis([0 4 -3 2 -2 2]); grid on; drawnow;
xlabel('Kx')
ylabel('Ky')
zlabel('Kz')
view(30,30)
title('Filter Design', 'FontSize', 12,...
    'FontWeight', 'bold');
print(fig, '-dpng', 'fig3')

%% Apply filter to average

% We will now apply this filter to our avg data in the freq domain.
fig = figure(4);
U_noisy_fft_avg_shift_filter = U_noisy_fft_avg_shift.*filter;
iter = 1;
for j = 0.65:0.05:.9
    sub = subplot(3,2,iter);
    isosurface(Kx,Ky,Kz, abs(U_noisy_fft_avg_shift_filter)/max(...
        abs(U_noisy_fft_avg_shift_filter(:))),j);
    axis([-4 4 -2 2 -2 2]); grid on; drawnow;
    text(1,1,2, strjoin(["isovalue =", j]))
    xlabel('Kx')
    ylabel('Ky')
    zlabel('Kz')
    view(30,30)
    iter = iter + 1;
end
sgtitle('Filter applied on average', 'FontSize', 12,...
    'FontWeight', 'bold');
print(fig, '-dpng', 'fig4')

%% Apply filter at each time slice
% We will now apply our filter to each time slice. This will help us
% to find the path of the particle.
position = zeros(20,3); % This will store the position of the particle
for j = 1:size(Undata,1) % go through each slice of time
    U_noisy(:,:,:) = reshape(Undata(j,:),n,n,n);
    U_noisy_fft = fftn(U_noisy);
    U_noisy_fft_shift = fftshift(U_noisy_fft);
    U_noisy_fft_shift_filter = U_noisy_fft_shift.*filter;
    U = abs(ifftn(ifftshift(U_noisy_fft_shift_filter)));
    [~,b] = max(U(:));
    position(j,:) = [X(b) Y(b) Z(b)];
```

```
158 end
159
160 %% Plot the path of the stone
161 fig = figure(5);
162 iter = 1;
163 for j=5:3:20
164     subplot(3,2,iter);
165     plot3(position(1:j,1),position(1:j,2),position(1:j,3), 'k-'); hold on;
166     plot3(position(1:j,1),position(1:j,2),position(1:j,3), 'r.');
167     axis([-15 15 -15 15 -15 15]); hold on;
168     view([30, 30])
169     iter = iter + 1;
170     title(strjoin(["After",j,"time slices"]))
171     xlabel('X')
172     ylabel('Y')
173     zlabel('Z')
174 end
175 sgtitle('Path of the particle.', 'FontSize', 12,...
176     'FontWeight', 'bold');
177 print(fig, '-dpng', 'fig5')
178
179 %% Final overall plot
180 fig = figure(6);
181 plot3(position(:,1),position(:,2),position(:,3), 'k-'); hold on;
182 plot3(position(:,1),position(:,2),position(:,3), 'r.');
183 axis([-15 15 -15 15 -15 15]); hold on;
184 view(71.9739,-25.8661)
185 title("Final path of the particle")
186 xlabel('X')
187 ylabel('Y')
188 zlabel('Z')
189 print(fig, '-dpng', 'fig6')
```