Spectrum analyzer documentation

Alaa Mohamed Roshdy 201600031

• Introduction

Spectrum analyzers are fundamental in digital signal processing (DSP). This spectrum analyzer processes signals and convolves them together. It also has the option to perform FFT on any audio file and view the resulting frequency domain graph. There is an option to compare the same signal at different DFT points, different window types and window lengths. There are built-in functions available to perform FFT on.

• Framework

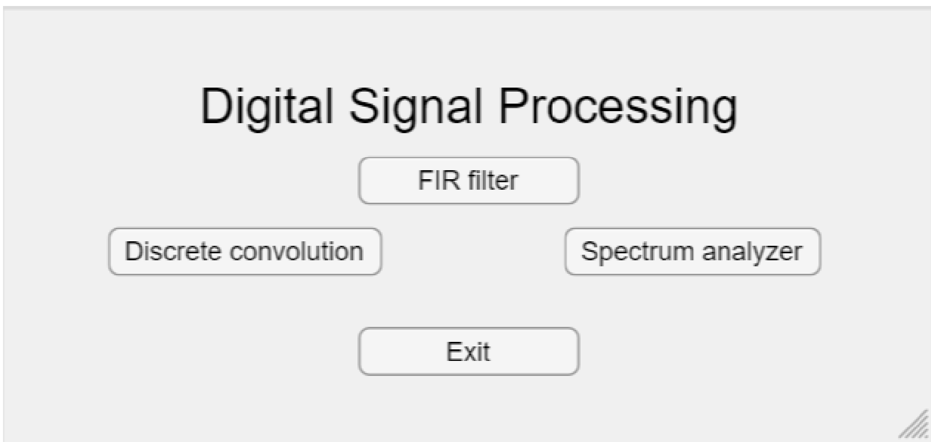

Figure 1. main window

The main framework of this program is divided into 3 main parts, discrete convolution, FIR filter and spectrum analyzer. When you open the program, all options will be available as buttons that will open a new window based on your choice. In all windows other than the main window, there is an exit and a back button.

• DISCRETE CONVOLUTION

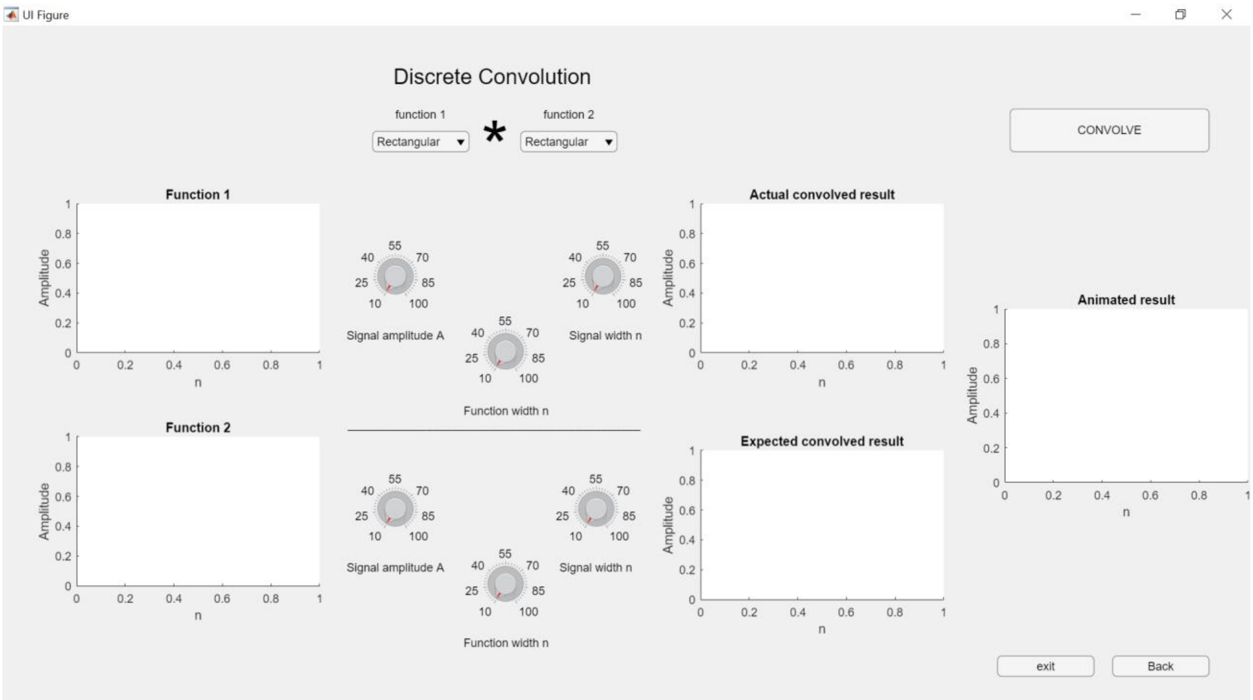

Figure 2. discrete convolution window

- Functions

There are 2 drop down lists for function 1 and function 2. The available functions that can be convolved together are rectangular, triangular, and ramp signals. The rectangular and triangular signals are generated using *tripuls* and *rectpuls* respectively. Both take in parameters *n* and *FunctionWidth* where n is the range of the signal on the axes and it is gotten from the parameter *SignalLength*. *n* is defined in such a way to make the signal centered around 0 in the x-axis (Discrete Time n). *FunctionWidth* and *SignalLength* are both parameters that the user chooses from the knobs in the program. This will be discussed later in functionalities section.
The ramp function is defined as $y = x$ for the range n. *n* is defined in such a way to ensure that the *SignalWidth* and *FunctionWidth* are centered around 0 in the x-axis (Discrete Time n).

- Functionalities

For each of the 2 functions, the user has the flexibility to choose the width of the signal, the width of the function and the amplitude of the signal from 3 different knobs. All 3 functionalities have a range from 10 to 100 points. To demonstrate each functionality, figure 1 shows function 1 as a rectangular signal having *SignalWidth* 25, *Amplitude* 10, and *FunctionWidth* 10.

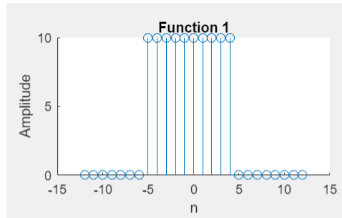

Figure 3. function 1 plot

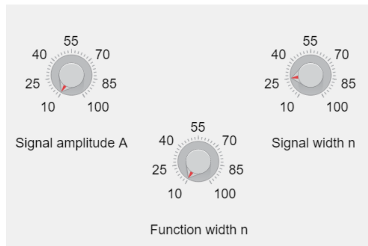

Figure 4. functionalities

The knobs shown in figure 4 are continuous, which consequently leads to fractional numbers. To avoid errors arising from fractions, all parameters are rounded up using *ceil* function.
Moreover, the *SignalWidth* can be odd or even numbered. The problem of even and odd arises when specifying the range *n* which, as previously mentioned, is the range of the signal (*SignalWidth*) centered around 0 in the x-axis (Discrete Time n). To resolve this issue, an if function is used to check if the *SignalWidth* is even using *rem(SignalWidth,2).*

n=-(SignalWidth)/2 +1: (SignalWidth)/2.	Odd signal
n = -(SignalWidth-1)/2:(SignalWidth-1)/2;	even signal

If the signal is even, the extra point that "will break" the symmetry of the signal will be on the negative x-axis of the signal.
For the ramp function, the *FunctionWidth* also plays a role in the issue of odd and even numbers. Unlike the rectangular and triangular functions, the ramp function is generated by first creating a function $y = x$ for the range of the function width, then appending zeros to the right and left of that of that function with a length *SignalWidth – FunctionWidth* on both the negative and positive sides to form the whole signal. Depending on whether the *FunctionWidth* is even or odd and whether the *SignalWidth* is even or odd, an extra 0 will be appended to the right or left of the signal. Refer to the code to find all 4 combinations (odd-odd, odd-even, even-odd, even-even) and their corresponding output array.

Both functions are then plotted on graph function 1 and graph function 2 accordingly, with the y-axis as an amplitude, and the x-axis the discrete time range n.

- ## Convolution and animated convolution process

  The convolve button calls a separate function *linconv* and 4 parameters are passed to it *y*, *y2*, *n* and *n2*. *y* and *y2* are function 1 and function 2 respectively, and *n* and *n2* are the ranges for *y* and *y2*. *y* and *y2* are zero padded to the right and left of the signal so that both signals are of equal lengths (output range should be = length of *y2* + length of *y* since this is a linear convolution) and centered around zero. The exact process is explained in the example below:

Function 1 range = [-4,6]
Function 2 range = [-3,3]
Convolution output range = [(-4) + (-3), (6) + (3)]

For function 1 and function 2 to have the same range as the convolution output, function 1 is appended the negative range of function 2 to the left of the signal and the positive range of function 2 to the right of the signal. The same is applied to y2. y is then flipped.

Y = [zeros(1, abs(min(n2))) y zeros(1,max(n2))];

To perform linear convolution on the processed functions, a matrix called *Y_matrix* is created from *Y* (processed version of y). Each row of this matrix will be multiplied by the function *Y2* (processed version of *y2*) and summed. To form *Y_matrix,* we first generate the first row of this matrix by shifting the function *Y* til the center of *Y* is the first element of the row. This array is called *firstrow*. This is done using *circshift(Y,min(n)+min(n2)). firstrow* is then multiplied by an array of ones and zeros to remove the unwanted values that appeared at the end of the array due to circular shift.

modified_firstrow = [ones(1,length(Y)-abs(min(n)+min(n2))) zeros(1,abs(min(n)+min(n2)))]
*firstrow = firstrow.*modified_firstrow;*

Now that the first row is generated, built-in matlab function *toeplitz* is used to generate *Y_matrix*. Toeplitz utilizes the fact that the negative side of the signal is the same as the positive side of the signal. The example below illustrates the process of generating *Y_matrix* from a *tripuls*.

Y = [0 0 1 2 3 4 3 2 1 0 0]
First row after circular shift = [4 3 2 1 0 0 0 0 1 2 3]
First row after modification = [4 3 2 1 0 0 0 0 0 0 0]

$$\text{Toeplitz of } y = \begin{bmatrix} 4 & 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 4 & 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 4 & 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

However, for the ramp function, the negative side of the signal has values different from the positive side of the signal by a factor of -1. To resolve this issue, the first column along with the first row of the signal needs to be generated. The negative values if the signal are stored in an array called *firstcolumn* using the same methodology that we used to get first row. The first row and column are then passed to the *toepltiz* function.
Once the matrix *Y_matrix* is generated, the convolution and animated convolution graphs are plotted using a for loop around the rows of *Y_matrix*. The expected convolution graph is computed using built-in function *conv().*
The following figure shows an example of the result of a convolution process.

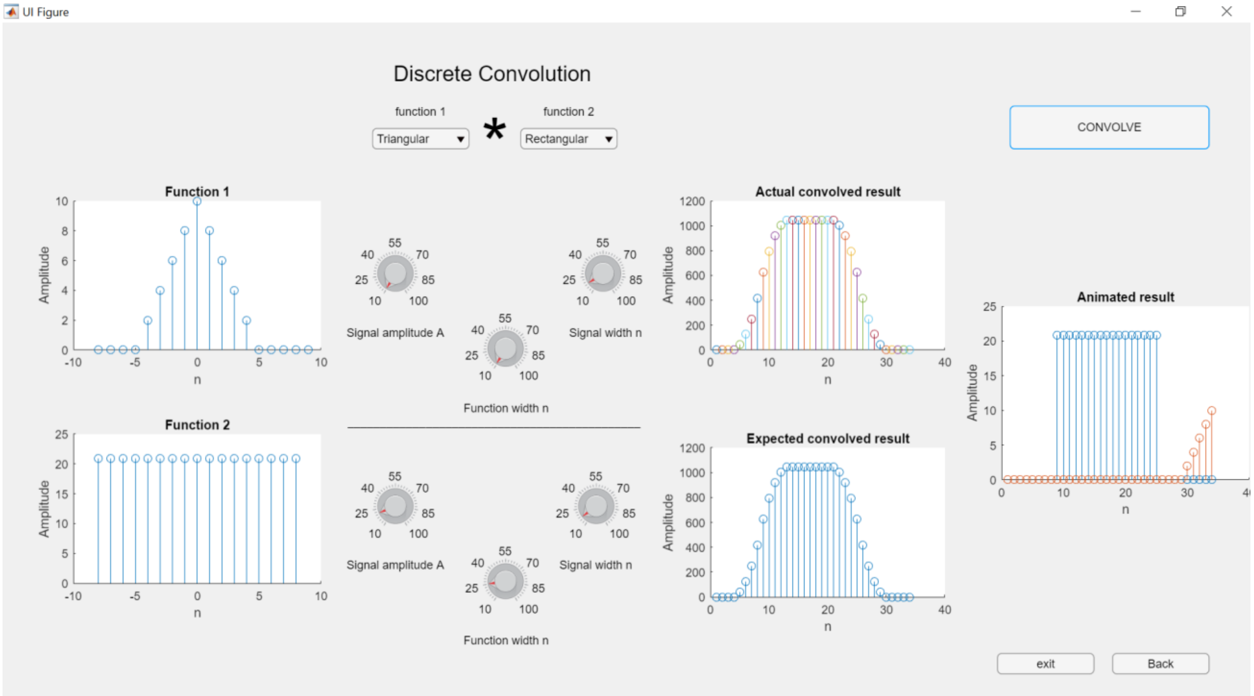

Figure 5. Result of Discrete convolution window with an arbitrary choice of knob values and functions

• SPECTRUM ANALYZER

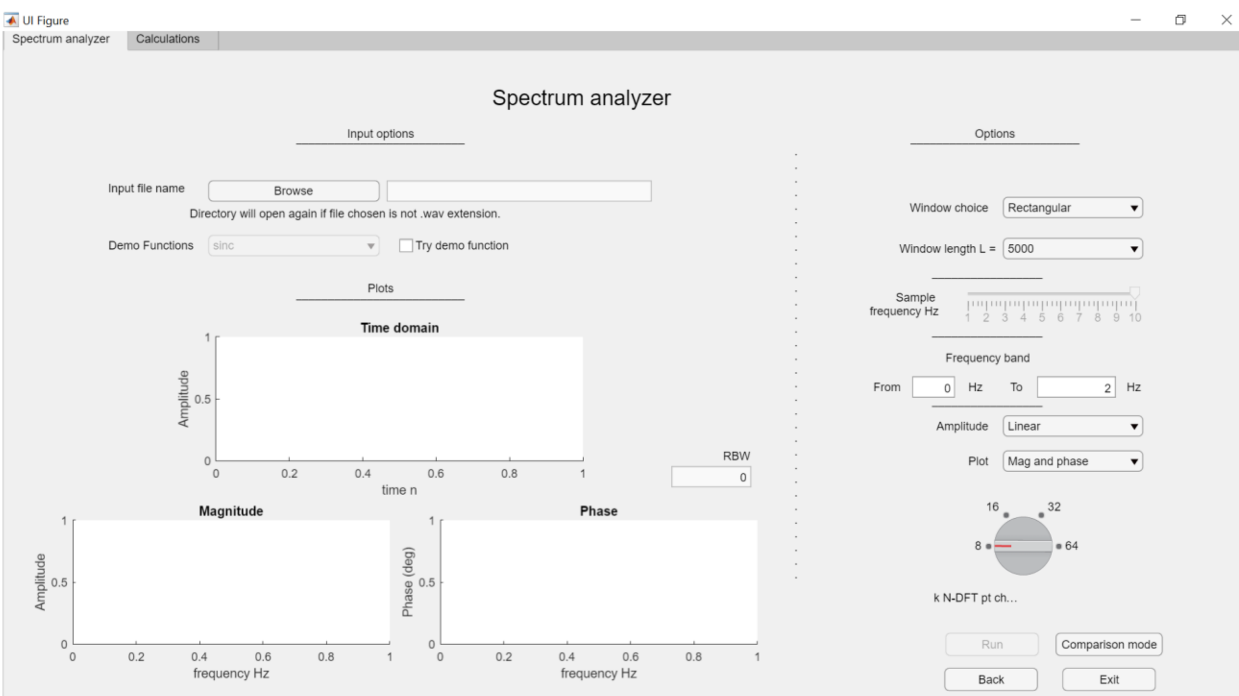

Figure 6. Spectrum analyzer window

The spectrum analyzer has 2 tabs, one for the DFT options and output, and another for the calculations resulting from the DFT output. First, we start with the spectrum analyzer tab. It has 2 options, either to input a file using the browse button to open a directory, or to use one of 3 available functions as examples to perform FFT on. The GUI adapts based on which option the user chooses through a check box called Try Demo Function. Unless the check box is checked, the N-DFT knob is read in thousands and the signal frequency is not accessible.

- Functionalities

The functionalities include, window choice, window length, N-DFT points, and sample frequency. Any inputted signal, whether from an audio file or an example file has a range equal to the N-DFT points *N* chosen. The window length *L*, which is in the form of a drop down list, changes according to the number of DFT points chosen so that it never exceeds it. It also varies depending on whether the input function is one of the demo examples or it's an input file. All lengths vary by a factor of 1000. See figure 7 below to show how window length list varies.

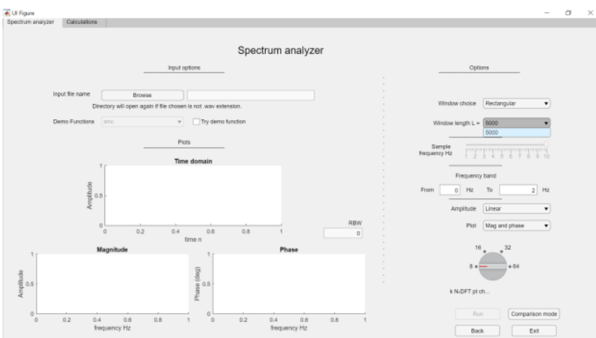

Figure 7a. Input function is an audio file, k DFT points is 8000

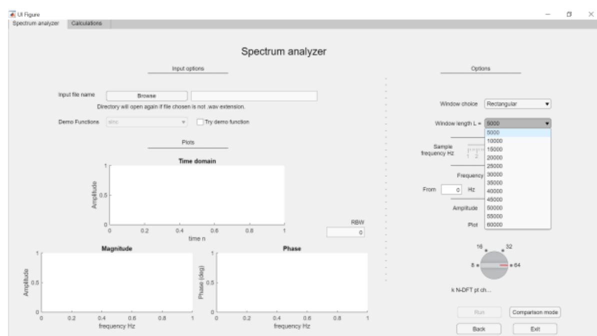

Figure 7b. Input function is an audio file, k DFT points is 64000

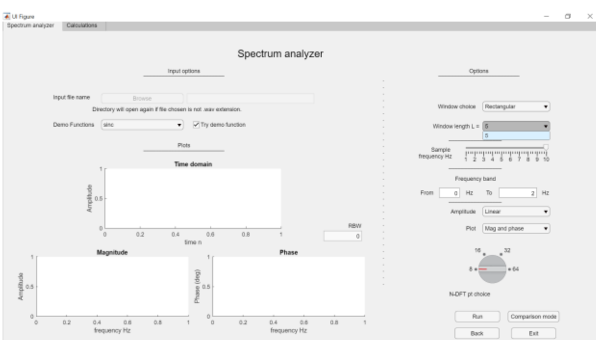

Figure 7c. Input function is an example, DFT points is 5

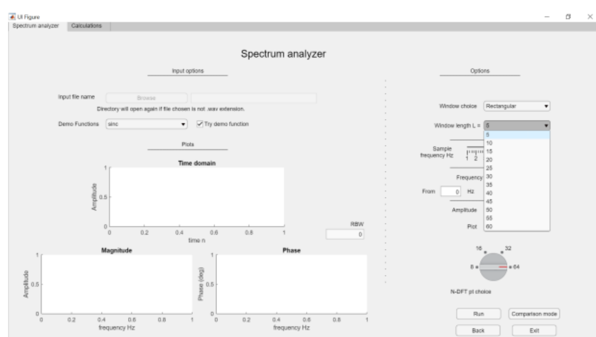

Figure 7d. Input function is an example, DFT points is 64

The window choices are rectangular, triangular, hamming and hanning windows and the are listed in the form of a drop down list. These choices are computed by first defining an array *w* of zeros with 1 row and *N* columns. The range *t* is *N* elements long and it represents time in seconds while *n* represents discrete time (has the same range). Then, depending on the window choice, the first *L* elements of *w* are modified.

$$w(1{:}L) = 1 \qquad \text{Rectangular window}$$

$$w\left(1:\frac{L}{2}\right) = \frac{t(1:\frac{L}{2})}{L/2}$$

$$w\left(\frac{L}{2}:L\right) = 0.2 - \frac{t(\frac{L}{2}:L)}{L/2}$$

Triangular window

$$w(1:L) = 0.5 - \cos\left(\frac{2\pi(1:L)}{L}\right)$$

Hanning window

$$w(1:L) = 0.54 - \cos\left(\frac{2\pi(1:L)}{L}\right)$$

Hamming window

The sample frequency fs is used for the demo functions and it ranges from 1 to 10 Hz. It is disabled when using the input audio file option.
The frequency span specifies the range of frequencies that the user wants to be displayed in the graph. It ranges from 0 till infinity. The value of "From" is always less than the value of "To" and "To" is assigned to be always greater than "From" and starts from 2 Hz. These are all restrictions to handle any errors that might arise from unwanted inputs by the user.

- Audio input

  The user inputs a file by clicking on browse button, where a directory opens to choose the desired file. This is done using *uigetfile()* built-in matlab function. The directory file will reopen again if the chosen file is not a .wav extension. Also, as long as no file is chosen or the demo function check box is not checked, the run button will remain disabled. Once the correct file extension is chosen, the input field will display the file name. To process the audio file, *audioread()* function is called and is passed 2 parameters, the audio file name and the range of samples to get (*N*) . The output is *x*, the main function, and *fs*, the sample frequency of the signal.

- Example functions

  There are 3 example functions, rect, sinc, and sine. Each have a length *n* where n is the range from 1 to *N*. Rect is formed by creating an array of ones, sinc is formed using the built-in *sinc()* function, and sine is formed using sin function.

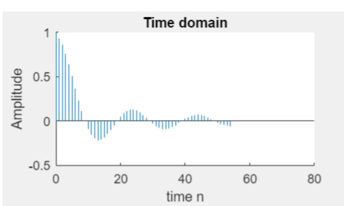

Figure 8a. sinc function

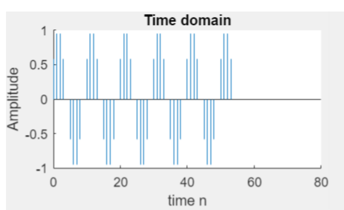

Figure 8b. sin function

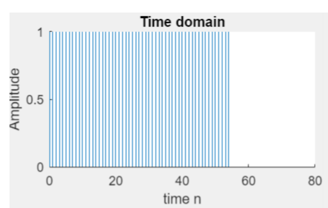

Figure 8c. rect function

- Calculations mode

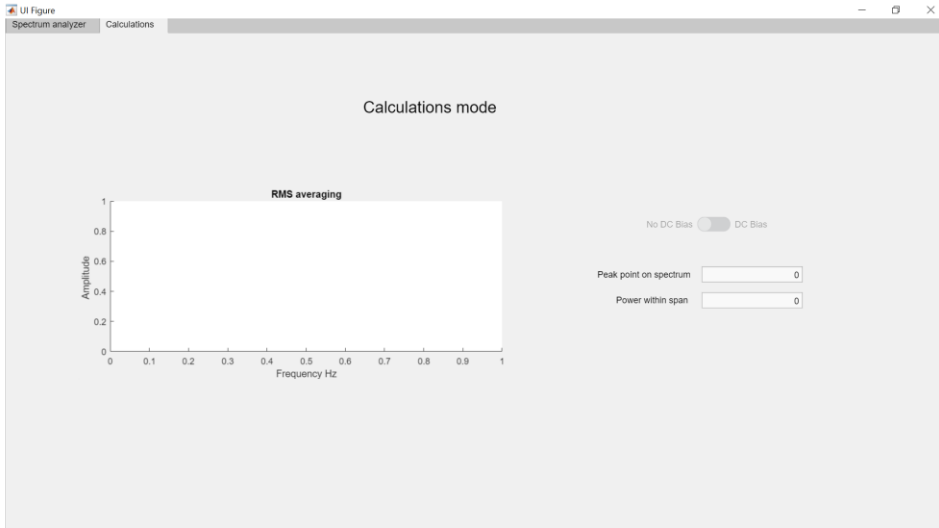

Figure 9. Calculations mode tab

Calculations tab has 2 graphs, one that plots the power spectral density of the specified frequency span range, and the other performs RMS averaging.
Also, there is a switch that indicates whether there is a DC bias or not, and a numeric field that displays the peak point on the frequency spectrum.
For the following figures, all the plots were based on the input sinc function, sample frequency 10 Hz and a rectangular window of length 55.

I. Power

powerband function was used to calculate the power within a span of frequencies.

II. RMS Averaging

For the RMS averaging, the time signal is divided into 4 ranges called *NoOfFFTs*, and fft is performed on each range independently as shown in the figure below. For this specific implementation, the range is specified beforehand and is not given as a choice to the user. This range is chosen to have the largest number of FFTs possible for this implementation. If the range was chosen to be greater than 4, say 8, the 8 N-DFT point would not be averaged because NoOfFFTs then would be 1.

The range, *NoOfPoints* is specified according to the total number of samples N and the number of ranges *NoOfFFTs.*

$$NoOfPoints = \frac{N}{NoOfFFTs}$$

The 2 variables, *first* and *last*, define the beginning and end of single range of samples in the time signal.

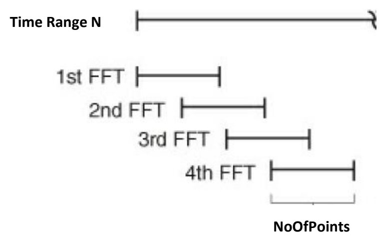

Figure 9. Illustrating RMS averaging steps

Once all variables are declared, each FFT output is stored in a separate row in a matrix called *FFTs*. Then, each row is summed and divided by NoOfPoints and the resulting vector is stored in a vector called RMS.

$$FFTs = \begin{matrix} F_1(1) & F_1(2) & F_1(3)\ \ldots\ldots \\ F_2(1) & F_2(2) & F_2(3)\ \ldots\ldots \\ \cdot & \cdot & \cdot \quad \ldots\ldots \\ F_{NoOfFFTs}(1) & F_{NoOfFFTs}(2) & F_{NoOfFFTs}(3)\ \ldots\ldots \end{matrix}$$

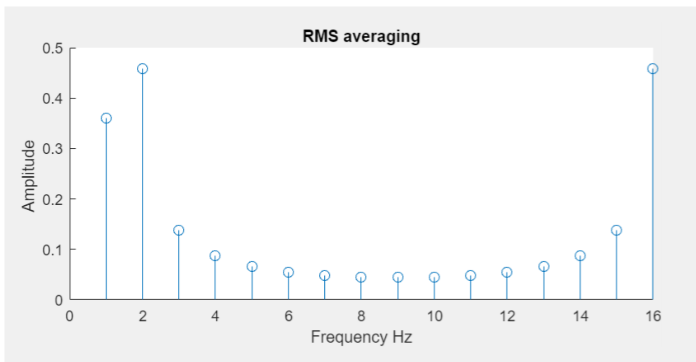

Figure 10. RMS averaging

III. DC Bias

The value *fftresult(1)* is checked. If the value is zero, then there is no DC bias, and the switch slides to the No DC bias side. If the value is not zero, the switch slides to the other side.

IV. Peak Point on spectrum

Regardless of the N-DFT point chosen by the user, the peak point is calculated based on the highest N-DFT point available called *maxN*.

It is calculated by finding the maximum value of the absolute *fftresult* vector.

- Plotting graphs

Run button processes the graphs to plot. To plot the input function *x*, the window function *w* is multiplied element-wise with *x*. The output is plotted on the range 0:*N-1* where *N* is the DFT points as previously mentioned. For the frequency domain output, *fft()* processes the Fourier transform of the function. The y-axis can be plotted in 2 different scales, in log or linear scale.

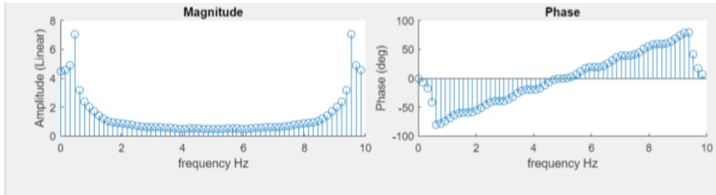

Figure 11 a. Linear scale for the fft plot

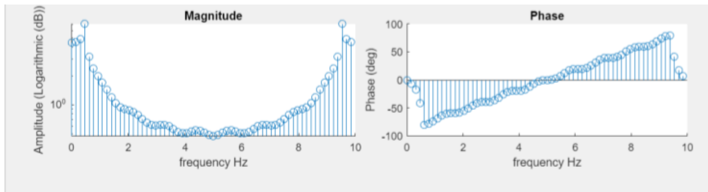

Figure 11 b. Logarithmic scale for the fft plot

As for the plot type, there are 2 options, either to plot the magnitude and phase or the real and imaginary.

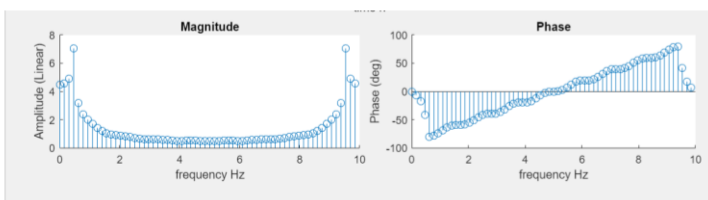

Figure 12 a. Magnitude and phase for the fft plot

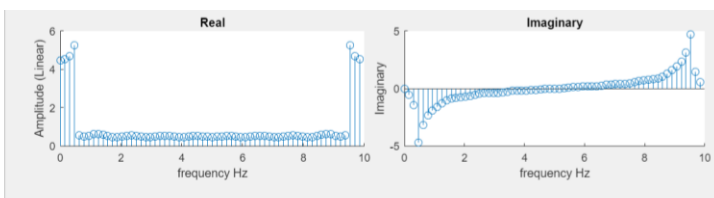

Figure 12 b. Real and imaginary for the fft plot

The output are plotted based on the frequency range span chosen by the user. Shown below are 2 plots for the sinc function, one with the span from 0 to 7 Hz and the other from 0 to 10 Hz.

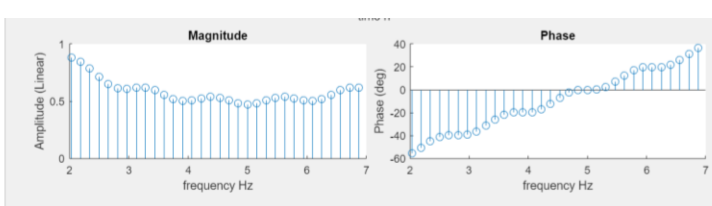

Figure 13 a. Frequency span from 0 to 7 Hz

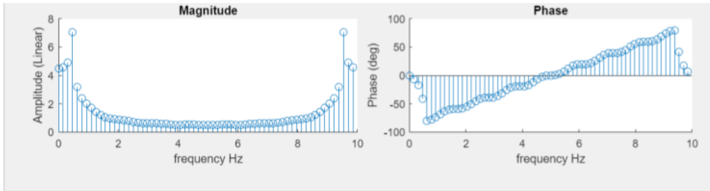

Figure 13 b. Frequency span from 0 to 10 Hz

Along with the output, a replicated figure of the magnitude/real plot is produced that has a markers option to view the x and y values of any sample.

- Comparison mode

Comparison mode button opens a new window that performs all the previously mentioned functions, only with the exception that the user will be allowed to choose 2 different values for the window length L, window choice, and N-DFT points. There are 4 different graphs, 2 of which compute the time domain output and the other 2 compute the frequency domain output. The first time domain and frequency domain graphs are controlled by drop down lists and discrete knobs separate from the other time domain and frequency domain graph.

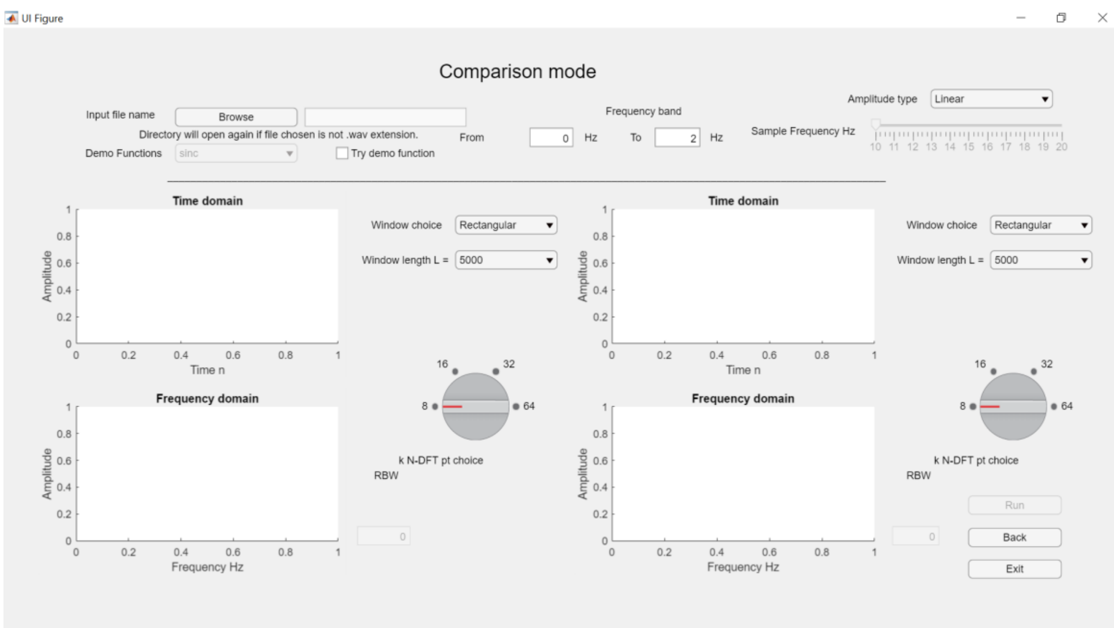

Figure 14. comparison mode window

• FIR FILTER

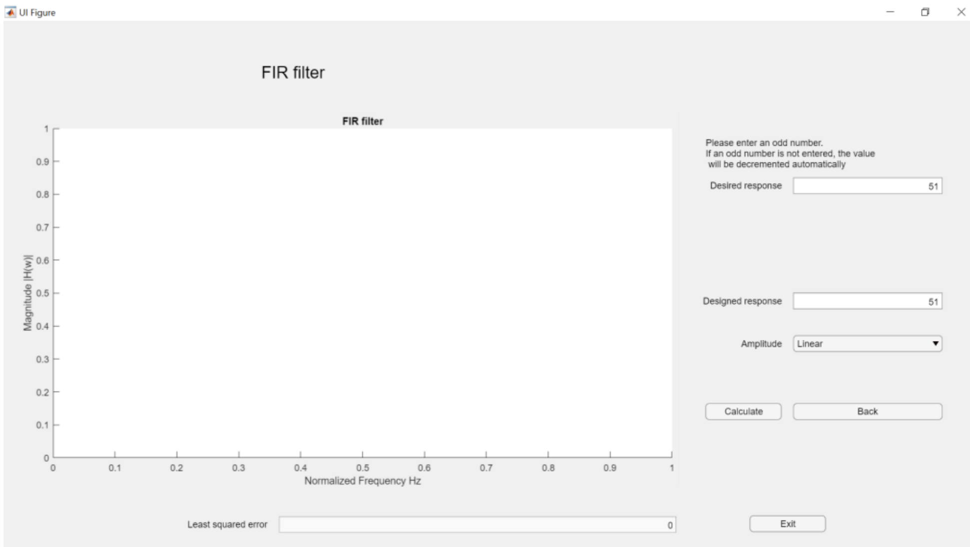

Figure 14. FIR filter window

The user inputs the desired filter response length *L* and the designed filter response *N*. Also, there is an option to view the resulting impulse response in log format using the drop down list of Amplitude type. The desired filter response is calculated by creating an array of zeros and ones. The ones are determined by *w* and w cut off w_c. The array is all ones from 0 til w_c, and the rest is zeros. w_c in this implementation is set as 0.25π because the filter is a low pass filter.

The designed, on the other hand, is calculated using a matrix F. The matrix F is formed by concatenating a row of length L of ones with a matrix of coefficients with dimensions (*L* x (*N*-1)/2). It is a matrix of cosines. The coefficients are all cosines because the designed impulse response h(n) is assumed to be even symmetric with odd number of taps, which makes the sines all cancel out. Once matrix *F* is formed, *h* can be easily calculated by taking the inverse of *F* multiplied by *HDesired*. However, the resulting *h* is even symmetric around zero, and the only visible range is the positive side. So, to have an even symmetric function that is fully visible, a new *h* is defined called *hsymmetric* which is the flipped version of *h.* Then, *hysmmetric* and *h* are concatenated to have a full, even symmetric signal centered around *N-1/2 + 1*. Now that *h* is finalized, *freqz()* is applied to *h* to get the *H*, frequency response of *h*.

The least squares error is calculated using

$$e^T e$$

Where e is

$$e = (F * h)^T - Hdesired$$

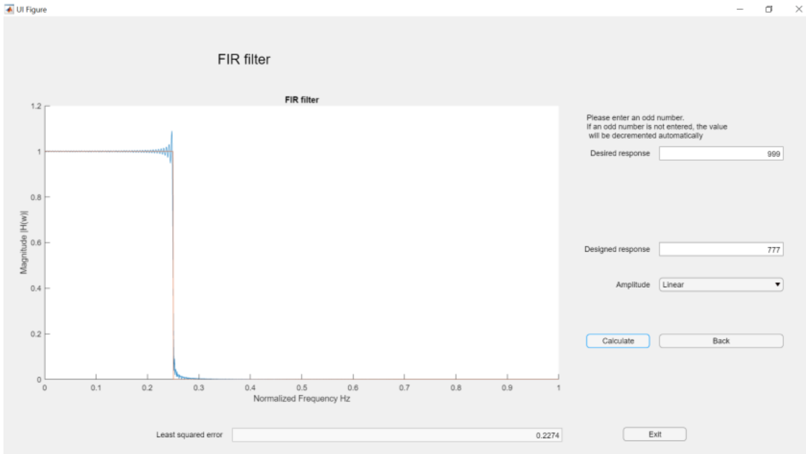

Figure 15. Example FIR filter response in linear scale

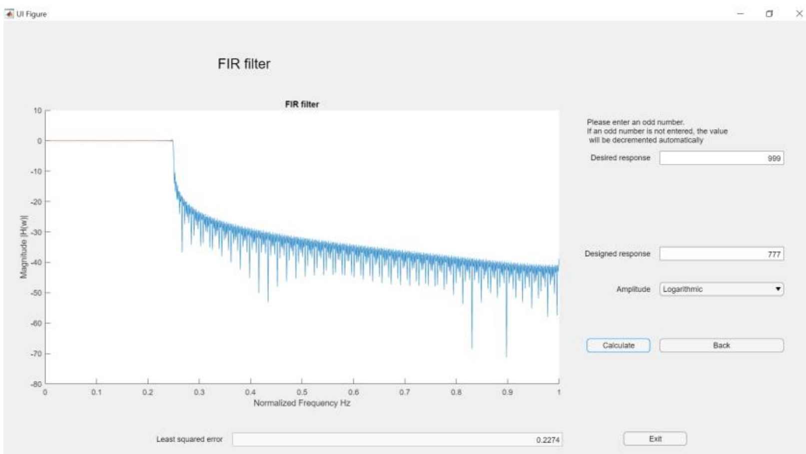

Figure 16. Example FIR filter response in log scale

Appendix

- **Discrete convolution window:**
  1. **Function to extract variables**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% Function that extracts the values from the knobs in the GUI.
        %These values will be sent to another function to be used in calculating the plot.
        function [Amplitude FunctionWidth SignalWidth] = variables(app,fn);
            if(fn == 1)
            %Initialize the variables
            Amplitude = app.SignalamplitudeAKnob.Value;
            FunctionWidth = ceil(app.FunctionwidthnKnob.Value); %Ceil is used to ensure that the function
width is an integer
            SignalWidth = ceil(app.SignalwidthnKnob.Value); %Ceil is used to ensure that the signla width
is an integer
            else
            Amplitude = app.SignalamplitudeAKnob_2.Value;
            FunctionWidth = ceil(app.FunctionwidthnKnob_2.Value); %Ceil is used to ensure that the
function width is an integer
            SignalWidth = ceil(app.SignalwidthnKnob_2.Value); %Ceil is used to ensure that the signl width
is an integer
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% This function calculates the Plot for the 2 functions plotted on the left side of the GUI.
        function [y n] = CalculatePlot(app, Amplitude, FunctionWidth, SignalWidth, fn)
            %%%%%%%%%%%
            %First, we determine which function is passed as a parameter through "fn". Function 1 or 2.
            if(fn == 1)
                 DropDownValue = app.function1DropDown.Value;
                 axes = app.UIAxes;
             else
                 DropDownValue = app.function2DropDown.Value;
                 axes = app.UIAxes_2;
            end
            %%%%%%%%%%%%
            %calculate n, the range of samples in the signal.
            %If the signal width is odd numbered,
            if(rem(SignalWidth,2))
                n = -(SignalWidth-1)/2:(SignalWidth-1)/2;
            else
                %if the signal width is even
                %the extra point that destroyed the symmetry of the signal will be put on the negative
side of the signal.
                %This is just to be consistent with what happens when rectpuls plots an even signal.
                n=-(SignalWidth)/2 +1: (SignalWidth)/2;
            end
            %%%%%%%%%%%%%%%%%%%%%%
```

```
            switch DropDownValue

                case 'Rectangular'
                    %in rectpuls, even signals result in an extra pulse on the negative side of the 
signal.
                    y = Amplitude*rectpuls(n,FunctionWidth);
                    stem(axes,n,y);

                case 'Triangular'
                    %tripuls is symmetric whether the functionwidth is even or odd, so no modifications 
need to be done.
                    y = Amplitude*tripuls(n,FunctionWidth);
                    stem(axes,n,y);
                case 'Ramp'
                    %%%%%%%%%%%%
                    %Handling the error that would arise if the function width is chosen to be greater 
than the signal width
                    if(FunctionWidth>SignalWidth)
                        SignalWidth=FunctionWidth;
                    end
                    %%%%%%%%%%%%
                        %caculate the values of the ramp function.
                    x = [-Amplitude:Amplitude/((FunctionWidth-1)/2):Amplitude];
                    y = x;
                    %%%%%%%%%%%%
                    %if the signal width is odd
                    if(rem(SignalWidth,2))
                        %%%%%%
                        %if the function width is odd, append zeros to the right and
                        %left of the function equally to form the signal.
                        if(rem(FunctionWidth,2))
                            y = [zeros(1,(SignalWidth-FunctionWidth)/2) y zeros(1,(SignalWidth-
FunctionWidth)/2)];

                        %if the function width is even, append zeros to the right and left of
                        %the function. The right will be appended an extra zero point than the
                        %left.
                        else
                            y = [zeros(1,(SignalWidth-FunctionWidth-1)/2) y 0 zeros(1,(SignalWidth-
FunctionWidth-1)/2)];
                        end
                        %%%%%%
                        %If the function width was greater than the signal width, recalculate n with the 
new value assigned to Signal width at the beginning of the case.
                        n = -(SignalWidth-1)/2:(SignalWidth-1)/2;
                        stem(axes,n,y);

                     %if the signal is even,
                     else
                        %%%%%%
```

```
                        %if the function width is odd
                        if(rem(FunctionWidth,2))
                        y = [zeros(1,(SignalWidth-FunctionWidth-1)/2) y 0 zeros(1,(SignalWidth-
FunctionWidth-1)/2)];
                        else
                        y = [zeros(1,(SignalWidth-FunctionWidth)/2) y zeros(1,(SignalWidth-
FunctionWidth)/2)] ;
                        end
                        %%%%%%
                        %If the function width was greater than the signal width, recalculate n with the
new value assigned to Signal width at the beginning of the case.
                        n=-(SignalWidth)/2 +1: (SignalWidth)/2;
                        stem(axes,n,y);
                    end
                    %%%%%%%%%%%%
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2. Linear convolution function

```
        function linconv(app,y,y2,n,n2)
            %expected convolved result
            stem(app.UIAxes5,conv(y,y2));
            %y will be flipped
            y =  fliplr(y);
            %y and y2 need to be zero-padded from the left and right
            Y = [zeros(1, abs(min(n2))) y zeros(1,max(n2))];
            Y2 = [zeros(1, abs(min(n))) y2 zeros(1,max(n))];

              %%%%%%%%%%%%
              %to perform linear convolution, a matrix called Y_matrix is created from function Y.
              %Each row of this matrix will be multiplied by the function Y2 and summed.
              %To form Y_matrix,


              %Get the first row of Y_matrix.
              %The first row of Y shifted to the left by min(n)+min(n2) which means that
              %The function Y got shifted until its center became at the beginning of the row.
              firstrow = circshift(Y,min(n)+min(n2));
              %Multiply this first row by an array of ones and zeros to remove the unwanted
              %values that appeared at the end of the array due to circular shift.
              %The length of ones and zeros is chosen to specifically only delete the values at the end
              %of the array of Y.
              modified_firstrow = [ones(1,length(Y)-abs(min(n)+min(n2))) zeros(1,abs(min(n)+min(n2)))]
              %Now multiply modified_firstrow by firstrow to get the desired row.
              firstrow = firstrow.*modified_firstrow;
              %%%%%
              %Now perform toeplitz to get the desired matrix.
```

```
            %However, this following condition is specific for the ramp function.
            %Since the ramp function has negative values, and these negative values were all deleted
            %due to the last few steps, we need to preserve the negative values in an array
            %called firstcolumn.
            if(min(Y)<0)
                %both functions are even
                if(~rem(length(y),2) && ~rem(length(y2),2))
                firstcolumn = circshift(Y,-(max(n)+max(n2)-1))
                %both functions are odd
                else if ((rem(length(y),2)) && rem(length(y2),2))
                      firstcolumn = circshift(Y,-(max(n)+max(n2)+1))
                      else
                     %if either functions is even and the other is odd
                      firstcolumn = circshift(Y,-(max(n)+max(n2)))
                      end
                end

            %Multiply this first column by an array of ones and zeros to remove the unwanted
            %values that appeared at the beginning of the array due to circular shift.
            %The length of ones and zeros is chosen to specifically only delete the values at the
beginning
            %of the array of Y.
            modified_firstcolumn = [zeros(1,length(Y)-abs(min(n)+min(n2))) ones(1,abs(min(n)+min(n2)))]
            %Now multiply modified_firstcolumn by firstcolumn to get the desired row.
            firstcolumn = firstcolumn.*modified_firstcolumn
                %Get Y_matrix
                Y_matrix = toeplitz(flip(firstcolumn),firstrow);
            else
            Y_matrix = toeplitz(firstrow);
            end
            %%%%%
            cla(app.UIAxes_3);
            %for loop to plot each row in the matrix individually multiplied and summed with Y2.
            for i = 1:size(Y_matrix,1)
                %plot the convolution
                stem(app.UIAxes_3,i,sum(Y_matrix(i,:).*Y2));
                hold(app.UIAxes_3,'on');

                %plot the animation
                cla(app.UIAxes_4);
                %Y2 will be the same throughout the animation. Y will slide over it.
                stem(app.UIAxes_4,Y2);
                hold(app.UIAxes_4,'on');
                stem(app.UIAxes_4,Y_matrix(i,:));
                hold(app.UIAxes_4,'on');

            pause(0.1);
            end

        end
```

```
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3. Spectrum analyzer main function

```
function DFT(app)
            % Initializing variables that are common between the examples and input files
            window = app.WindowchoiceDropDown.Value;        %window type
            L = str2num(app.WindowlengthLDropDown.Value);   %window length
            N = str2num(app.kNDFTptchoiceKnob.Value);        %number of samples
            maxN = 64;                                  %This will be used for calculating the peak value of
the spectrum
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Choosing the function that will be used.
            % If the check box is checked, the examples will be used instead of input file
            if(app.TrydemofunctionCheckBox.Value)
                %%%%
                % Initializing the variables
                fs = app.SamplefrequencyHzSlider.Value;         %sample frequency
                ts = 1/fs;                                      %sample to sample width
                t = (1:N)*ts;                                   %Time in s
                n = int8(t/ts);                                 %Time in n
                % Get RBW
                app.RBWEditField.Value = fs/N;
                func = app.DemoFunctionsDropDown.Value;
                %%%%
                x = zeros(1,length(n));
                switch func
                    case 'sinc'
                    x(n) = sinc(t);
                    case 'sine'
                    x(n) = sin(2*pi*t);
                    case 'rect'
                    x(n) = ones(1,length(t));
                end
                %%%%
            else
                %%%%
                % Initializing the variables
                N = N*10^3;                                 %N-DFT choice
                maxN = maxN*10^3;                               %This will be used for calculating the peak
value of the spectrum
                [x,fs] = audioread(app.InputfilenameEditField.Value,[1,N]); %inputting audio file
                app.RBWEditField.Value = fs/N;
                ts = 1/fs;                                  %Width between samples in seconds
                t = (1:N)*(ts);
                n = int32(t/ts);                            %Range in discrete time n
```

```
                x = x';     %audioread returns a column vector and all the coming operations are row
vectors so I took the transpose of the signal.
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Switch cases for window type
            w = zeros(1,length(t));
            cla(app.UIAxes);
            switch window
                case 'Rectangular'
                    w(1:L) = 1;
                case 'Triangular'
                    w(1:L/2) = t(1:L/2)/(L/2);
                    w(L/2:L) = 0.2 - t(L/2:L)/(L/2);
                case 'Hanning'
                    w(1:L) = 0.5 - 0.5*cos(2*pi*(1:L)/L);
                case 'Hamming'
                    w(1:L) = 0.54 - 0.46*cos(2*pi*(1:L)/L);
            end
            result = x.*w;
            fn = stem(app.UIAxes,0:N-1,result);
            set(fn, 'marker','none');

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Plotting the graphs for DFT
            cla(app.UIAxes2);
            %%%%%%%%%%%%%%%%%%%%
                PlotType = app.PlotDropDown_2.Value;
                fftResult = (fft(result,N));
                AmplitudeType = app.AmplitudeDropDown.Value;

                %Set the plot type
                switch PlotType
                    case 'Mag and phase'

                            %Plot the magnitude and phase
                            AmplitudeType = app.AmplitudeDropDown.Value;
                            %Set the y axis scale
                                switch AmplitudeType
                                    case 'Linear'
                                    stem(app.UIAxes2,(0:N-1)*fs/N,abs(fftResult));
                                    case 'Logarithmic'
                                     stem(app.UIAxes2,(0:N-1)*fs/N,pow2db(abs(fftResult)));
                                end
                             stem(app.UIAxes2_2,(0:N-1)*fs/N,angle(fftResult)*180/pi);

                            %Set the freqency span
                            xlim(app.UIAxes2,[app.FromEditField.Value app.ToEditField.Value]);
                            xlim(app.UIAxes2_2,[app.FromEditField.Value app.ToEditField.Value]);
```

```
                %Markers task implemented on the magnitude plot
                fig = figure('Name','For markers task')
                stem((0:N-1)*fs/N,abs(fftResult));
                xlim([app.FromEditField.Value app.ToEditField.Value]);
                datacursormode on;

        case 'Real and Im'

                %Plot the real and imaginary axes
                switch AmplitudeType
                        case 'Linear'
                        stem(app.UIAxes2,(0:N-1)*fs/N,real(fftResult));
                    case 'Logarithmic'
                        stem(app.UIAxes2,(0:N-1)*fs/N,pow2db(abs(real(fftResult))));
                end
                stem(app.UIAxes2_2,(0:N-1)*fs/N, imag(fftResult));

                %Set the frequency span
                xlim(app.UIAxes2,[app.FromEditField.Value app.ToEditField.Value]);
                xlim(app.UIAxes2_2,[app.FromEditField.Value app.ToEditField.Value]);

                %Markers task implemented on the real plot
                fig = figure('Name','For markers task')
                stem((0:N-1)*fs/N,real(fftResult));
                xlim([app.FromEditField.Value app.ToEditField.Value]);
                datacursormode on;

    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculations tab starts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Indicate whethe there is a DC bias
    if(fftResult(1) == 0)
        app.Switch.Value = 'No DC Bias';
    else
        app.Switch.Value = 'DC Bias';
    end

%Calculate peak point
app.PeakpointonspectrumEditField.Value = max(abs(fft(result,maxN)));
%%%%%%%%%%%%%%
%Calculate the PSD
if (~app.TrydemofunctionCheckBox.Value)
    power = bandpower(fftResult,fs,[app.FromEditField.Value app.ToEditField.Value]);
app.PowerwithinspanEditField.Value = power;
end
%%%%%%%%%%%%%%
%Calculate RMS averaging
```

```
        %Initialize the number of FFTs to be made for the signal
        NoOfFFTs = 4;
        %Calculate the number of points taken for each FFT operation
        NoOfPoints = N/NoOfFFTs;
        %N/4 because it is the largest value that will divide up
        %the signal into seveal FFTs. i.e. if I chose NoOfPoints
        %to be N/8, for the 8 point DFT the number of FFTs made will be
        %1 which makes RMS averaging unnecessary.

        %Initialize the range of values that we will perform FFT on
        first=1;
        last=NoOfPoints;

        %Perform FFT 4 times, each time taking a different range
        for k = 1:NoOfFFTs
        FFTs(k,:)  = abs(fft(result(first:last),NoOfPoints))
        %increment the range
        %Let there be 50% overlap between the ranges
        %NoOfPoints/2 makess this overlap happen
        first = first + NoOfPoints/2;
        last = last + NoOfPoints/2;
        end
        %Take the average of the solution
        RMS = sum(FFTs,1)./NoOfPoints;
        %Plot RMS
        stem(app.UIAxes4,RMS);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    end
end
```

4. Comparison mode main function

```
function DFT(app)
        % Initializing variables that are common between the examples and input files
        %signal 1
        window = app.WindowchoiceDropDown.Value;
        L = str2num(app.WindowlengthLDropDown.Value);
        N = str2num(app.kNDFTptchoiceKnob.Value);
        %signal 2
        window2 = app.WindowchoiceDropDown_2.Value;
        L2 = str2num(app.WindowlengthLDropDown_2.Value);
        N2 = str2num(app.kNDFTptchoiceKnob_2.Value);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Choosing the function that will be used.
        % If the check box is checked, the examples will be used instead of input file
        if(app.TrydemofunctionCheckBox.Value)
            %%%%
            % Initializing the variables of the example function
            fs = app.SamplefrequencyHzSlider.Value;         %sample frequency
```

```
            ts = 1/fs;                                              %sample to sample width
            %signal 1
            t = (1:N)*ts;                                           %Time in s
            n = int8(t/ts);                                       %Time in n
            % Get RBW
            app.RBWEditField.Value = fs/N;
            %signal 2
            t2 = (1:N2)*ts;
            n2 = int8(t2/ts);
            % Get RBW
            app.RBWEditField_2.Value = fs/N2;
            func = app.DemoFunctionsDropDown.Value;
            %%%%
            x = zeros(1,length(n));
            x2 = zeros(1,length(n2));
            switch func
                case 'sinc'
                x(n) = sinc(t);
                x2(n2) = sinc(t2);
                case 'sine'
                x(n) = sin(2*pi*t);
                x2(n2) = sin(2*pi*t2);
                case 'rect'
                x(n) = ones(1,length(t));
                x2(n2) = ones(1,length(t2));
            end
            %%%%
        else
            %%%%
            % Initializing the variables
            N = N*10^3;
            N2 = N2*10^3;
            [x,fs] = audioread(app.InputfilenameEditField.Value ,[1,N]); %inputting audio file
            [x2,fs] = audioread(app.InputfilenameEditField.Value ,[1,N2]); %inputting audio file
            app.RBWEditField.Value = fs/N;
            app.RBWEditField_2.Value = fs/N2;
            ts = 1/fs;
            t = (1:N)*(ts); %Take the first N points
            t2 = (1:N2)*ts;
            n = int32(t/ts);
            n2 = int32(t/ts);
            x = x'; %Since the output x is a column vector, it should be changed to row vector for
coming operations.
            x2 = x2';
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Switch cases for window 1 and window 2
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Switch cases for window 1
        w = zeros(1,length(t));
```

```matlab
cla(app.UIAxes);
switch window
    case 'Rectangular'
        w(1:L) = 1;
    case 'Triangular'
        w(1:L/2) = t(1:L/2)/(L/2);
        w(L/2:L) = 0.2 - t(L/2:L)/(L/2);
    case 'Hanning'
        w(1:L) = 0.5 - 0.5*cos(2*pi*(1:L)/L);
    case 'Hamming'
        w(1:L) = 0.54 - 0.46*cos(2*pi*(1:L)/L);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plotting the graph
% for window 1
result = x.*w;
fn = stem(app.UIAxes,0:N-1,result);
set(fn, 'marker', 'none');
%specify the yscale
amptype = app.AmplitudetypeDropDown.Value;
switch amptype
    case 'Linear'
    stem(app.UIAxes2,((1:N)*fs/N),abs(fft(result,N)));
    case 'Logarithmic'
    stem(app.UIAxes2,((1:N)*fs/N),pow2db(abs(fft(result,N))));
end
%frequency span
xlim(app.UIAxes2,[app.FromEditField.Value app.ToEditField.Value]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Switch cases for window 2
w2 = zeros(1,length(t2));
cla(app.UIAxes_2);
switch window2
    case 'Rectangular'
        w2(1:L2) = 1;
    case 'Triangular'
        w2(1:L2/2) = t(1:L2/2)/(L2/2);
        w2(L2/2:L2) = 0.2 - t(L2/2:L2)/(L2/2);
    case 'Hanning'
        w2(1:L2) = 0.5 - 0.5*cos(2*pi*(1:L2)/L2);
    case 'Hamming'
        w2(1:L2) = 0.54 - 0.46*cos(2*pi*(1:L2)/L2);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plotting the graph
% for window 2
result2 = x2.*w2;
fn2 = stem(app.UIAxes_2,0:N2-1,result2);
set(fn2, 'marker','none');
```

```
        %specify the y scale
        amptype = app.AmplitudetypeDropDown.Value;
        switch amptype
            case 'Linear'
            stem(app.UIAxes2_2,(0:N2-1)*(fs/N2),abs(fft(result2,N2)));
            case 'Logarithmic'
            stem(app.UIAxes2_2,(0:N2-1)*(fs/N2),pow2db(abs(fft(result2,N2))));
        end
        %frequency span
        xlim(app.UIAxes2_2,[app.FromEditField.Value app.ToEditField.Value]);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end

end
```

5. FIR Filter main function

```
function results = calculate(app)
        cla(app.UIAxes5);
        %Initialize the variables

        %desired and designed filter response length
        L = app.DesiredresponseEditField.Value;
        N = app.DesignedresponseEditField.Value;
        %Cut off frequency for low pass filter
        w_c = 0.25*pi;
        %Shift the signals by half the length of the response
        halfN = (N-1)/2;
        % Initialize the w for the matrix
        wL = pi*(0:L-1)/L;
        %%%%%%%%%%%%%%%%%%%%%%%%
        %Calculate Hdesired
        %Get the amplitude of the Hdesired signal
        %It is an array of ones and zeros
        %The ones are specified based on the cutoff frequency
        %Ones at w less than w_c
        Hdesired = (wL<=w_c);
        %%%%%%%%%%%%%%%%%%%%%%%%
        %Calculat Hdesigned
        %Create matrix F with dimensions (L x ((N-1)/2 +1) )
        F = [ones(L,1) 2*cos((1:halfN).*wL')];
        h = pinv(F)*Hdesired';

        wfine = linspace(0,pi,L);
        %Modify the h to make it even symmetric flipping the signal and
        %concatinating the flipped signal with the original signal
        flipH=fliplr(h')
        hsymmetric = [ flipH(1:length(flipH)-1) , h']
        %get the freq response of the h
```

```
        H = freqz(hsymmetric,1,wfine);

        %%%%%%%%%%%%%%%%%%%%%%
        %Specify the type of the y-axis and plot the graphs
        AmplitudePowType = app.AmplitudeDropDown.Value;
        %plot the graph for the designed h
            %Set the y axis scale
                switch AmplitudePowType
                    case 'Linear'
                            plot(app.UIAxes5, wfine/(pi),abs(H));
                            hold(app.UIAxes5, 'on');
                            plot(app.UIAxes5,wfine/(pi), abs(Hdesired));

                    case 'Logarithmic'
                            HLOG = pow2db(abs(H));
                            HdesiredLOG = pow2db(abs(Hdesired));
                            plot(app.UIAxes5, wfine/(pi),(HLOG));
                            hold(app.UIAxes5, 'on');
                            plot(app.UIAxes5,wfine/(pi),(HdesiredLOG));

                end
        %%%%%%%%%%%%%%%%%%%%%
        %Calculate least squares error
        e = (F*h)' - Hdesired;
        app.LeastsquarederrorEditField.Value = abs((e)*(e)');
    end

end
```