Lab #7 FIR Filter Design

Table of Contents

Lab Objectives

- You have learned how to create a simple moving average filter that has a number of coefficients that are all the same value and you have built low pass filters using Windowed SINC impulse responses. But, the MAV filters are not particularly selective and their selectivity cannot be easily be modified. There are also instances when we need to reject low frequency content or pass signals with frequencies within a range or reject signals within a specific range. Therfore, we will learn how to create high pass filters, bandpass and band reject filters from low pass prototypes using techniques discussed in class and from the Smith Text.
- You will also learn how to get the frequency response of a filter from its impulse response using the Fast Fourier Transform FFT.
- You will learn to test and characterize sinusoidal input filter performance in the time domain.
- You previously designed a stable, recursive method to compute the running standard deviation of a signal, but we don't always need the standard deviation from the beginning of time. Therefore, you will learn how to create and implement a faster responding approach to calculating the standard deviation.

Reading

Chp 14, 16, and 19 in Smith.   http://Arduino.cc

Intended Learning Outcomes:

- Demonstrate high pass filtering of a signal using an FIR filter implemented in a microprocessor
- Demonstrate band pass filtering of a signal using an FIR filter implemented in a microprocessor
- Demonstrate band stop filtering of a signal using an FIR filter implemented in a microprocessor
- Practice communicating your results and your work in writing

Section 1 – FIR Sinc Filters: Low Pass Filter (LPF) and High Pass Filter (HPF)

Overview

In this section, you will create an FIR low pass filter with a cutoff frequency of 50 breaths per minute (bpm), and then transform the low pass filter into a high pass filter by spectral inversion. The sample frequency of the system is 10 Hz, or 600 bpm. The fractional cutoff frequency is then 50/600 = 0.0833.

The high pass FIR filter is created by starting with a low pass windowed sinc filter kernel at a cutoff frequency of Fc=0.08333 fractional frequency (fraction of the sample rate). Once the kernel is complete, invert the sign of each point in the impulse response of the filter, and then add 1.0 to the middle (M/2) point. Kernels are implemented in fixed-point integer form, so the value of "1" is represented by the value of the kernel scaling fixed point (HFXPT). The kernel is the impulse response of the high pass filter.

For this section of the lab use the Serial Monitor to view and copy the Arduino output. You will copy some of this information into MATLAB for analysis.

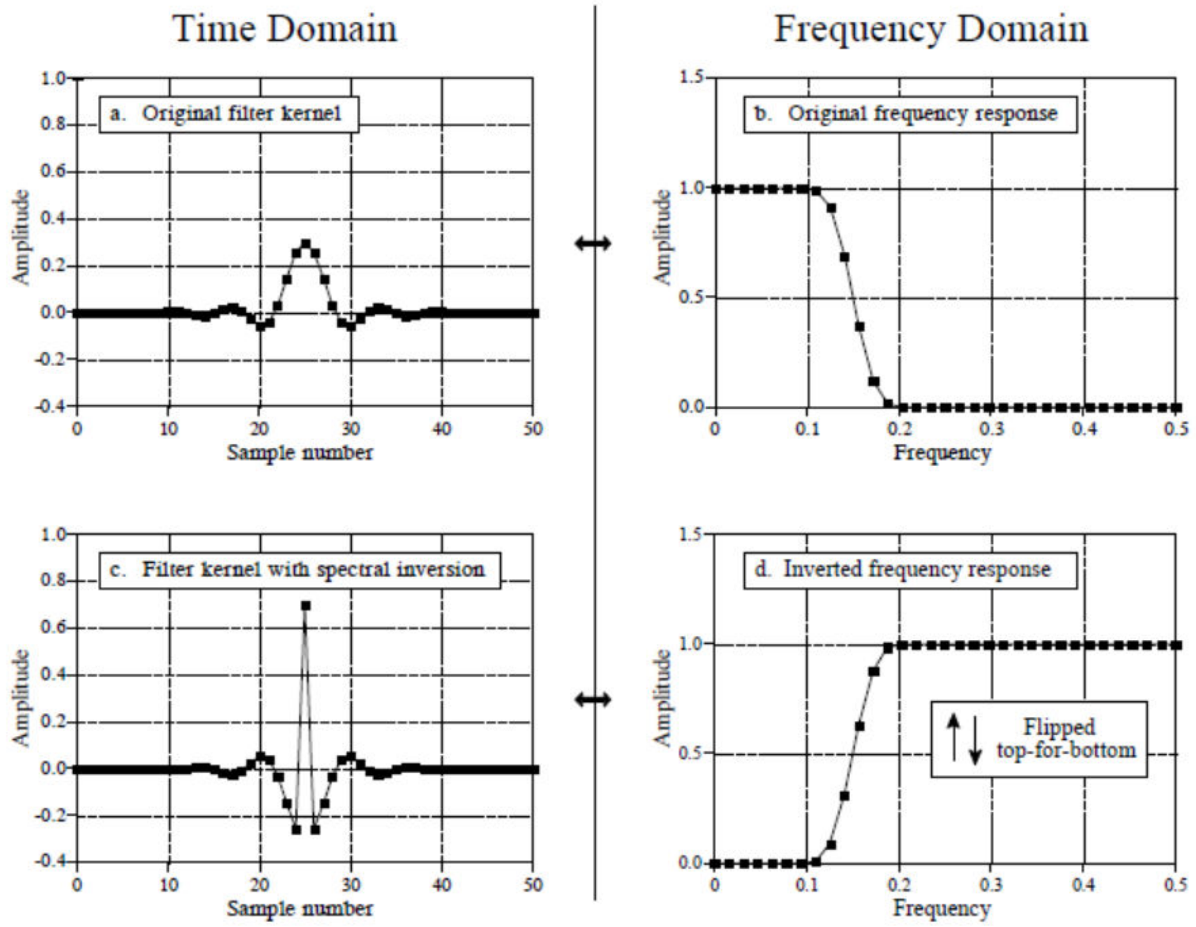


Figure 1. Conversion of a low pass FIR filter kernel into a high pass filter kernel by spectral inversion. Note how the HPF impulse response terms have negative sign for all terms and also the center point of the impulse response is both inverted and has the value 1.0 added to it. In the frequency domain, this results in the spectrum being flipped top for bottom.

Description of the code

Use the file Lab7_FIR_Section_01_Base_Code.ino for this part of the lab.

The code first creates the windowed sinc Low Pass Filter (LPF), normalizes it for unity DC gain, and maps it to a fixed-point representation. The impulse response is printed out so that you can find the frequency response in MATLAB. The sketch also prints out a C style declaration section that will be used to create filters in following sections.

Refer to the comments in the C code for details on what the code is doing. Become familiar with how the code works.

For this section of the lab use the Serial Monitor to view and copy the Arduino output. You will copy some of this information into MATLAB for analysis.

Procedure

1) Upload the sketch and modify the corner frequency of the kernel to 50 Breaths Per Minute (BPM).

2) Modify the code in the spectralInvertKernel( ) function in the Arduino C-code to convert the low pass filter kernel to a high pass filter kernel using the spectral inversion method.. Verify that you have created of a correct HPF kernel by comparison to the figure above from the text. Note: the kernels are stored in fixed-point variables and are scaled to retain their resolution. Recall that a spectrally inverted filter can be created from the LPF impulse response by negating each coefficient of the filter except for the middle sample. Create the middle sample of the HPF kernel by setting the coefficient to HFXPT minus the middle coefficient of the LPF. The value of HFXPT represents the value of 1 using fixed point scaling.

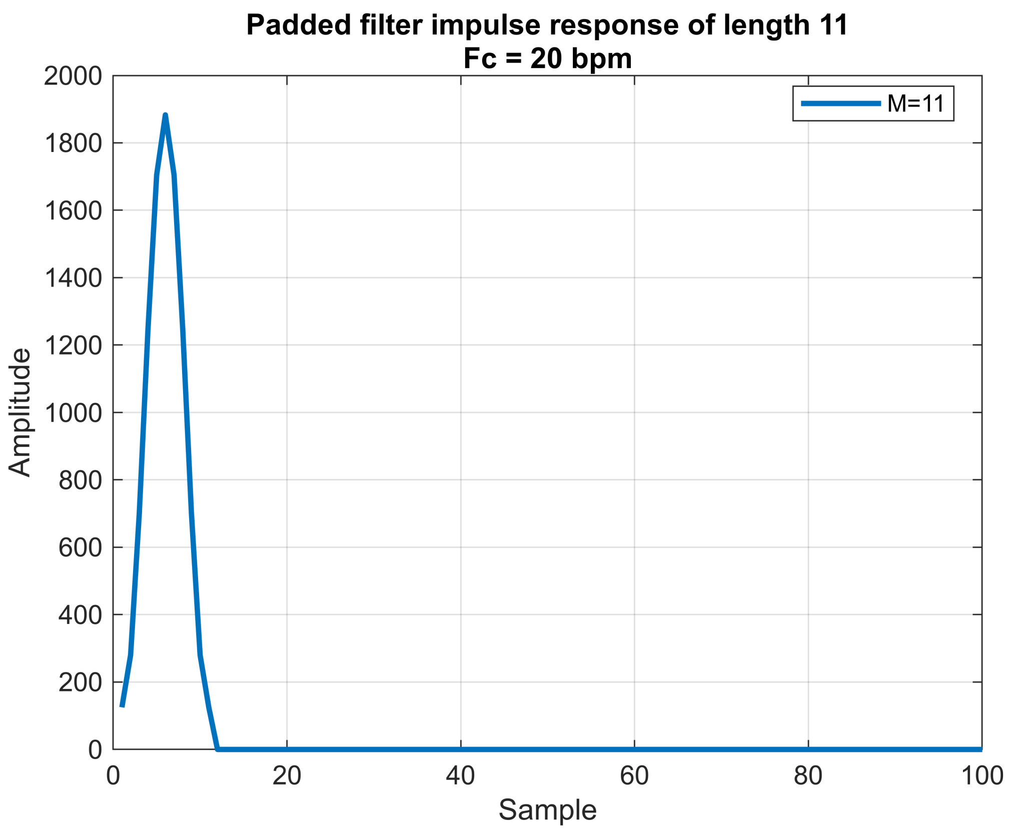
3) Change the impulse response lengths to 21, 41 and 81. For each kernel length copy the impulse response for the LPF and the HPF to an appropriately named variable in MATLAB. Copy these values from the Serial Monitor into this MATLAB Live Script. <u>Don't use the CaptureArduinoData.m function</u>.

The Discrete Fourier Transform (DFT) implemented by the FFT function in MATLAB will be used to find the frequency response of the filter. To improve the resolution the impulse response will be padded with zeros to a length of 256 samples. This will change the frequency resolution to

$f_{res} = \frac{f_{sample}}{N}$ **where N is the number of points in the impulse response plus the zero-padding.**

4) For each impulse response variable in MATLAB pad each variable with enough zeros to make the length of the impulse response 256. This can be done by concatenating a vector of zeros to the impulse response. An easy way to create a vector of zeros is to use the "zeros" function in MATLAB. I have done this in an example below:

```
%  Example Code
%
% Impulse response for LPF of Fc = 20 BPM MFILT = 11 (copied from the
% serial monitor after running the C code on the Arduino).
%

MFILT = 11;
hLPF_11 = [125,280,701,1247,1705,1883,1705,1247,701,280,125];

hLPF_11_padded = [hLPF_11, zeros( 1, 256-MFILT )];
%  Plot the impulse response of the first 101 samples
figure
plot(hLPF_11_padded,'LineWidth',2)
grid on
title(sprintf('Padded filter impulse response of length 11\nFc = 20 bpm'));
xlabel('Sample');
ylabel('Amplitude')
xlim([0,100])

legendStr = {'M=11'};
legend(legendStr,'Location',"best");
```



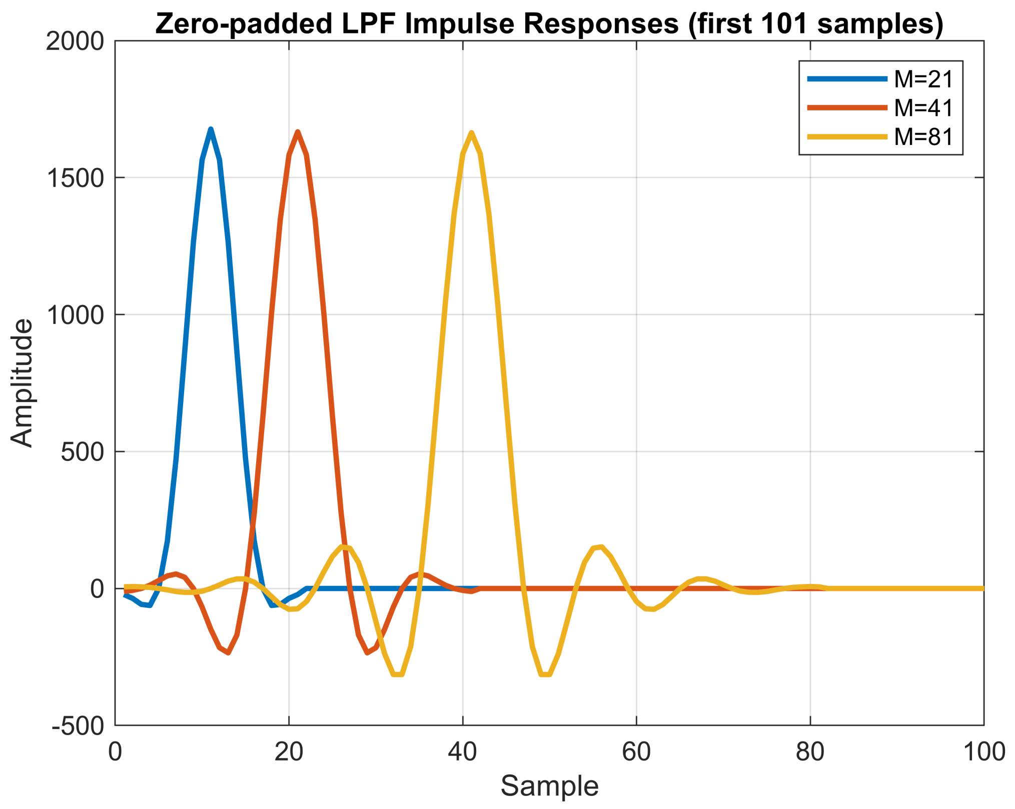
5) Plot P1 and P2 -- Zero pad each impulse response of length 21, 41 and 81 to a length of 256 samples (lowpass and highpass responses). Make 1 plot with all three impulse responses for the LPF overlaid on top of one another and another plot with all three impulse responses for the high pass filters overlaid on top of one another. Only plot the first 101 samples of the impulse responses. Include both plots P1 and P2 in your report.

6) Question Q1 -- In the narrative of your report in your own words (don't just repeat this question) answer which one of the three filters will introduce the most delay in the output signal? Why?

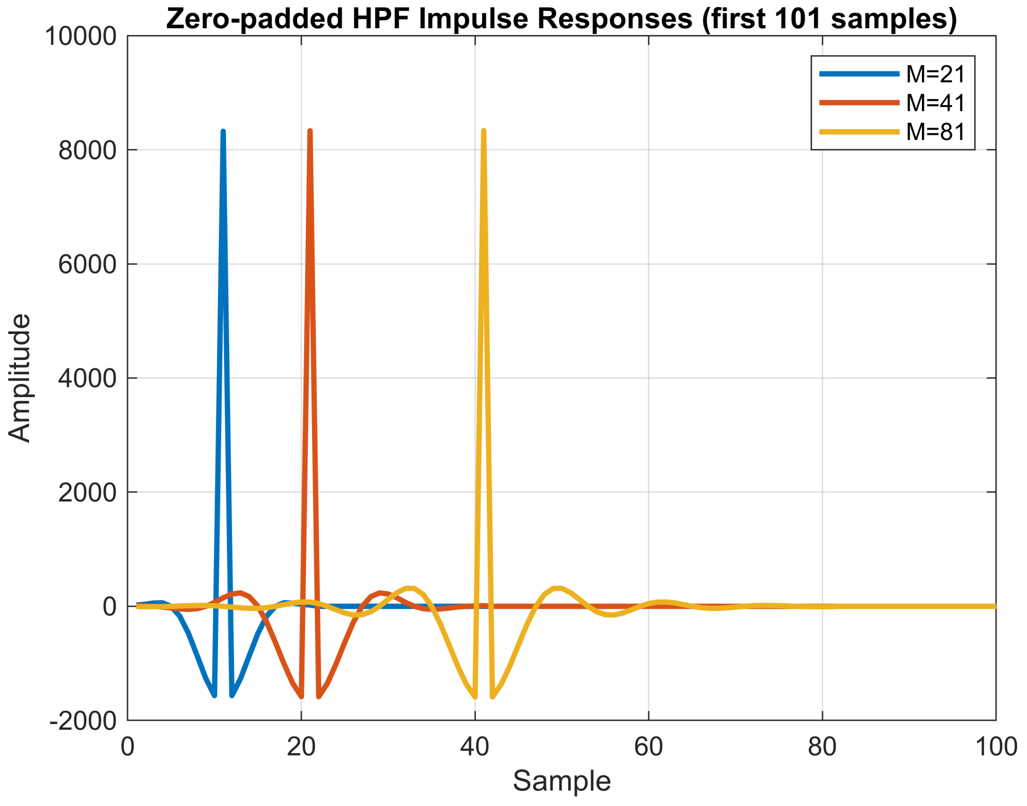
Answer Q1: Based on the impulse-response plots, the filter with the longest length of samples (M = 81) clearly shifts its main lobe the farthest in time and therefore introduces **the most delay** in the output signal. In general, for a FIR filter, the more impluse response a filter has, the longer its midpoint and its effective time shift.

```
%  Zero pad each impulse responseof length 21, 41 and 81 to a length of 256
%  samples.  Plot all three impulse responses on a single graph.  Only show
%  the first 101 samples of each (use the xlim([start, stop]) to limit the
%  x-axis length

%  Put your solution here
hLPF_21 =
[-22,-36,-58,-62,0,173,473,865,1265,1565,1677,1565,1265,865,473,173,0,-62,-58,-36,-2
2];
hLPF_21_padded = [hLPF_21, zeros( 1, 256-21 )];
```

```
hHPF_21 =
[22,36,58,62,0,-173,-473,-865,-1265,-1565,8323,-1565,-1265,-865,-473,-173,0,62,58,36
,22];
hHPF_21_padded = [hHPF_21, zeros( 1, 256-21 )];

hLPF_41 =
[-11,-7,0,12,29,46,53,41,0,-68,-149,-216,-235,-170,0,275,629,1008,1348,1583,1667,158
3,1348,1008,629,275,0,-170,-235,-216,-149,-68,0,41,53,46,29,12,0,-7,-11];
hLPF_41_padded = [hLPF_41, zeros( 1, 256-41 )];
hHPF_41 =
[11,7,0,-12,-29,-46,-53,-41,0,68,149,216,235,170,0,-275,-629,-1008,-1348,-1583,8333,
-1583,-1348,-1008,-629,-275,0,170,235,216,149,68,0,-41,-53,-46,-29,-12,0,7,11];
hHPF_41_padded = [hHPF_41, zeros( 1, 256-41 )];

hLPF_81 =
[6,7,6,4,0,-5,-11,-14,-14,-10,0,13,27,35,35,23,0,-30,-59,-76,-74,-48,0,61,117,152,14
7,95,0,-121,-238,-314,-314,-212,0,307,673,1046,1368,1587,1664,1587,1368,1046,673,307
,0,-212,-314,-314,-238,-121,0,95,147,152,117,61,0,-48,-74,-76,-59,-30,0,23,35,35,27,
13,0,-10,-14,-14,-11,-5,0,4,6,7,6];
hLPF_81_padded = [hLPF_81, zeros( 1, 256-81 )];
hHPF_81 =
[-6,-7,-6,-4,0,5,11,14,14,10,0,-13,-27,-35,-35,-23,0,30,59,76,74,48,0,-61,-117,-152,
-147,-95,0,121,238,314,314,212,0,-307,-673,-1046,-1368,-1587,8336,-1587,-1368,-1046,
-673,-307,0,212,314,314,238,121,0,-95,-147,-152,-117,-61,0,48,74,76,59,30,0,-23,-35,
-35,-27,-13,0,10,14,14,11,5,0,-4,-6,-7,-6];
hHPF_81_padded = [hHPF_81, zeros( 1, 256-81 )];

figure
plot(hLPF_21_padded,'LineWidth',2); hold on;
plot(hLPF_41_padded,'LineWidth',2);
plot(hLPF_81_padded,'LineWidth',2);
legend('M=21','M=41','M=81'); xlabel('Sample'); ylabel('Amplitude');
title('Zero-padded LPF Impulse Responses (first 101 samples)');
grid on; xlim([0 100]); hold off;
```



```
figure
plot(hHPF_21_padded,'LineWidth',2); hold on;
plot(hHPF_41_padded,'LineWidth',2);
plot(hHPF_81_padded,'LineWidth',2);
legend('M=21','M=41','M=81'); xlabel('Sample'); ylabel('Amplitude');
title('Zero-padded HPF Impulse Responses (first 101 samples)');
grid on; xlim([0 100]); hold off;
```



7) Find the frequency response of the LPF for each filter length using the DFT. For a length N sample sequence, the DFT results in $N/2+1$ samples in the frequency domain. These magnitudes represent frequencies from 0 to the Nyquist frequency of $f_{sample}/2$. There are two sequences that result, one from the magnitude of the COSINE terms and one from the magnitude of the SINE terms.

The 'fft' function in MATLAB computes the discrete fourier transform but for a sequence of N samples it produces N complex output samples. These represent the frequencies from 0 to f_{sample}. The terms in the output of the FFT are complex values. The real part of the value represents the magnitude of the COSINE term in the DFT. The imaginary part of the output is the magnitude of the SINE term in the DFT.

The syntax for the FFT function is:

```
%  m = fft( h );
```

The fft function can also zero pad the input by adding a second argument that determines the length of the FFT.

```
%  FFT with zero padding
%
%  m = fft(h, totalNumberSamples);
```

However, we have already padded the sequence, so use your zero padded sequence.

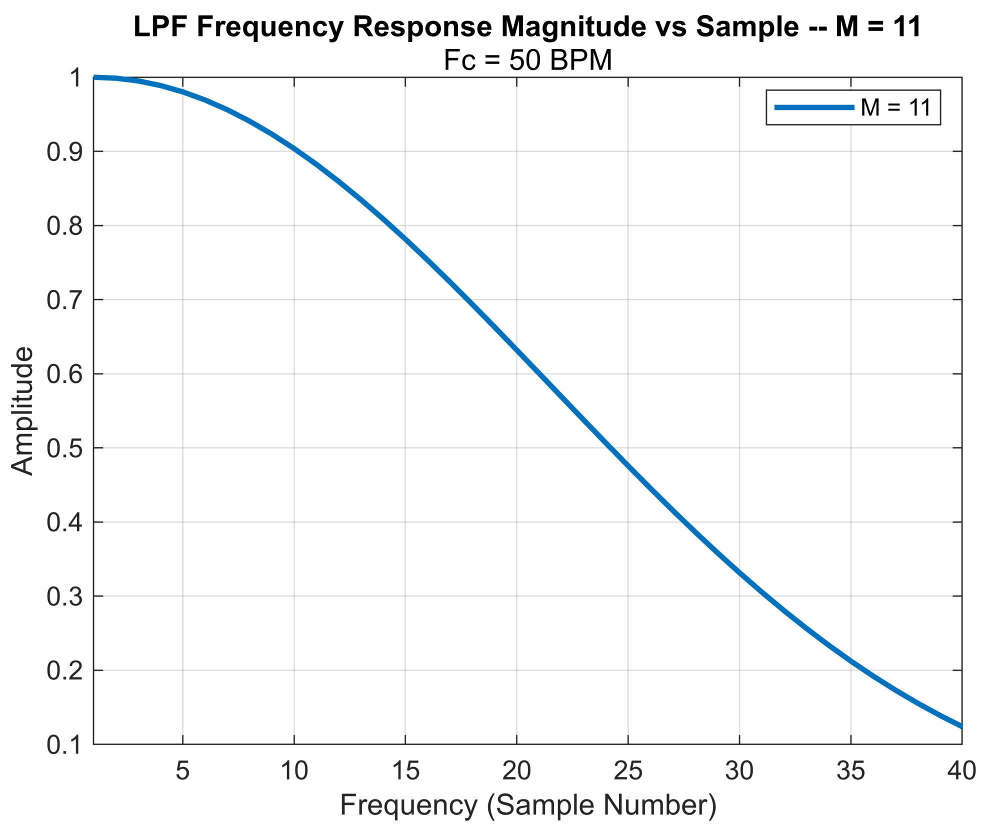
The magnitude of the frequency response is found using the MATLAB function abs.

```
% mag = abs(m)
```

Recall that the impulse response is scaled by HFXPT in order to use fixed point values. In order to get the frequency response in the desired units, scale each impulse by 1/HFXPT prior to computing the FFT.

An example of computing the FFT of the impulse response and plotting the frequency response is shown below for the M = 11, Fc = 20 BPM FIR low pass filter.

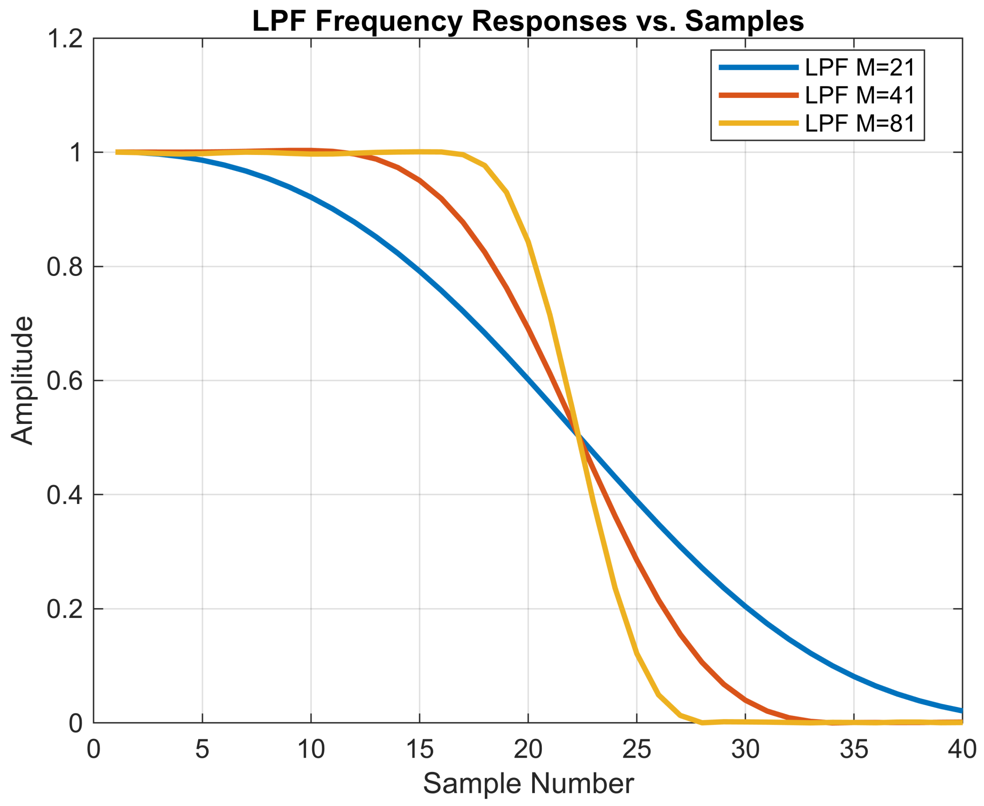
```
%  Example
%
%  Plot the impulse response of the LPF.  Scale the impulse response by the
%  value of HFXPT

HFXPT_LPF = 10000;


%  Find the frequency response.  Scale the impulse response by the value of
%  HFXPT

[mLPF_11] = fft( hLPF_11_padded/HFXPT_LPF );


%  Plot the magnitude of the frequency response versus sample number.
%  Change the X-axis so that it shows just the first 40 frequency bins or samples

figure
plot(abs(mLPF_11),'LineWidth',2);
grid on
title('LPF Frequency Response Magnitude vs Sample -- M = 11', 'Fc = 50 BPM')
xlabel('Frequency (Sample Number)');
ylabel('Amplitude');

%  Show just the first 40 frequency bins or samples
xlim([1,40])

legendStr = {'M = 11'};
legend(legendStr,'Location',"best")
```



4) PLOT P3 --- Plot the magnitude of the frequency response for all three low pass filters of length 21, 41, 81 on the same graph below (don't use decibels). Plot the frequency responses vs the sample number Limit the x-axis to a range from 0 to 40. This can be done using the xlim([0,40]) command after plotting the graph.

```
%
%  Find and plot the magnitude of the frequency response of the low pass filters of
length 21, 41, 81, Fc = 50 BPM using the fft function in
%  MATLAB.
%

%  Place your solution here
HFXPT_LPF = 10000;
[mLPF_21] = fft(hLPF_21_padded/HFXPT_LPF);
[mLPF_41] = fft(hLPF_41_padded/HFXPT_LPF);
[mLPF_81] = fft(hLPF_81_padded/HFXPT_LPF);

figure;
plot(abs(mLPF_21),'LineWidth',2); hold on;
plot(abs(mLPF_41),'LineWidth',2);
plot(abs(mLPF_81),'LineWidth',2);
legend('LPF M=21','LPF M=41','LPF M=81','Location','best');
xlim([0,40]);
xlabel('Sample Number'); ylabel('Amplitude');
```

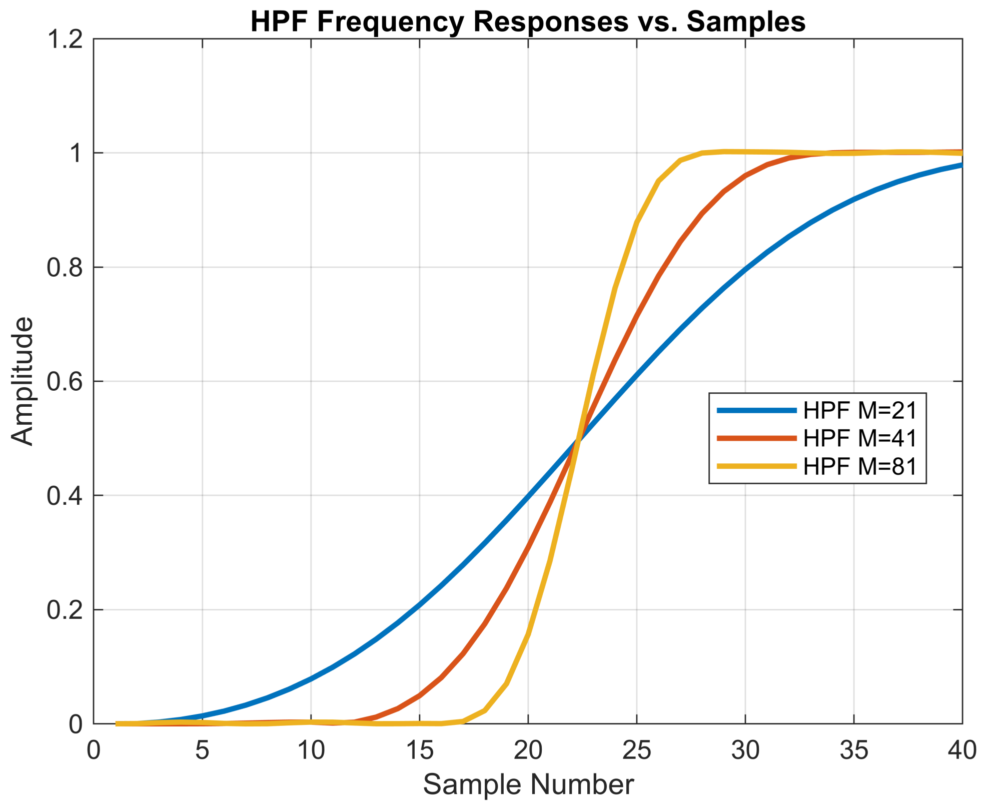
```
title('LPF Frequency Responses vs. Samples');
grid on; hold off;
```



7) PLOT P4 -- Plot the magnitude of the frequency response for all three high pass filters of length 21, 41, 81 on the same graph below (don't use decibels). Plot the frequency responses vs the sample number Limit the x-axis to a range from 0 to 40. This can be done using the xlim([0,40]) command after plotting the graph

```
%
%  Find and plot the magnitude of the frequency response of the high pass filters
of length 21, 41, 81, Fc = 50 BPM using the fft function in
%  MATLAB.
%

%  Place your solution here
HFXPT_HPF = 10000;
[mHPF_21] = fft(hHPF_21_padded/HFXPT_HPF);
[mHPF_41] = fft(hHPF_41_padded/HFXPT_HPF);
[mHPF_81] = fft(hHPF_81_padded/HFXPT_HPF);

figure;
plot(abs(mHPF_21),'LineWidth',2); hold on;
plot(abs(mHPF_41),'LineWidth',2);
plot(abs(mHPF_81),'LineWidth',2);
```

```
legend('HPF M=21','HPF M=41','HPF M=81','Location','best');
xlim([0,40]);
xlabel('Sample Number'); ylabel('Amplitude');
title('HPF Frequency Responses vs. Samples');
grid on; hold off;
```



8) The length of the impulse response is padded to 256 samples and the FFT is computed using all 256 samples. The sampling frequency of the system is 10 Hz (600 bpm). Each sample is taken at a 10 Hz rate (0.1 Sec/sample). The 256 point FFT splits the frequency dimension into 256 bins starting at 0 Hz and extending to the sample rate of 10 Hz. Each bin in the DFT is then 10Hz/256 wide (except the first and the last).

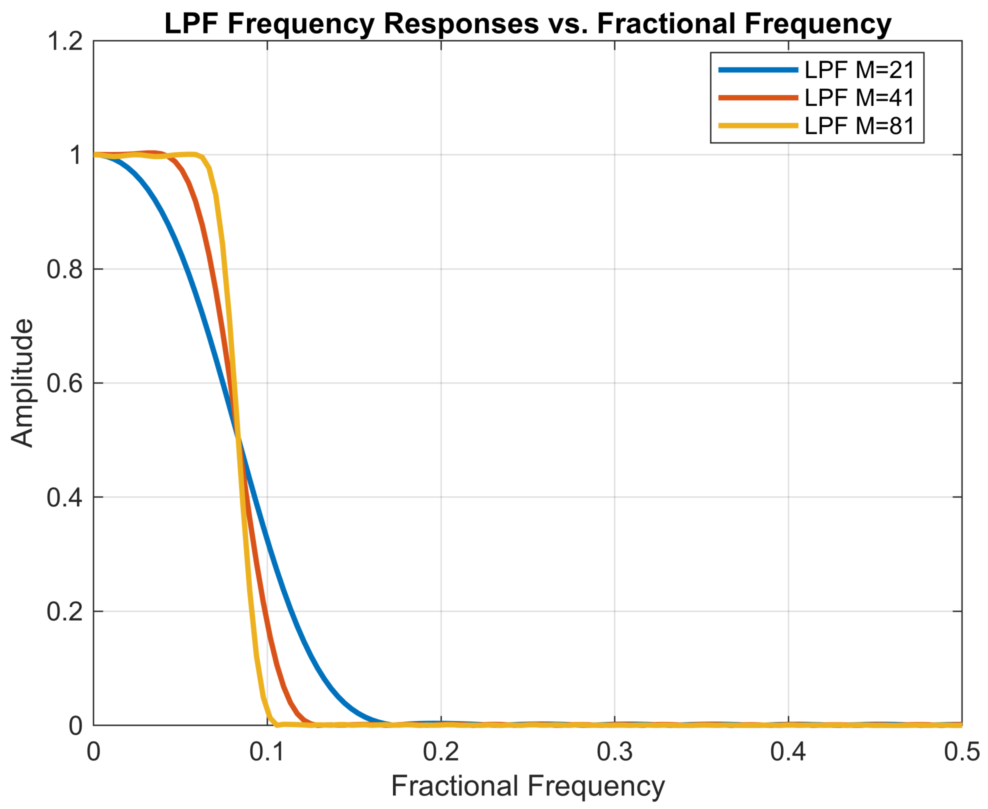
9) In MATLAB create a row vector of length 256 that represents the fractional frequency for each point in the FFT. That is, the frequency as a fraction of the sampling frequency. In MATLAB you can easily create a row vector with incrementing value using the syntax below. This will create a row vector from 0 to 10 in increments of 1 and a row vector from 0 to 10 in increments of 0.2

```
%  variable = [0:10];   %  row vector with elements 0 to 10 in increments of 1

%  variable2 = [0:.2:10];  %  row vector with elements from 0 to 10 in increments
of 0.2
```

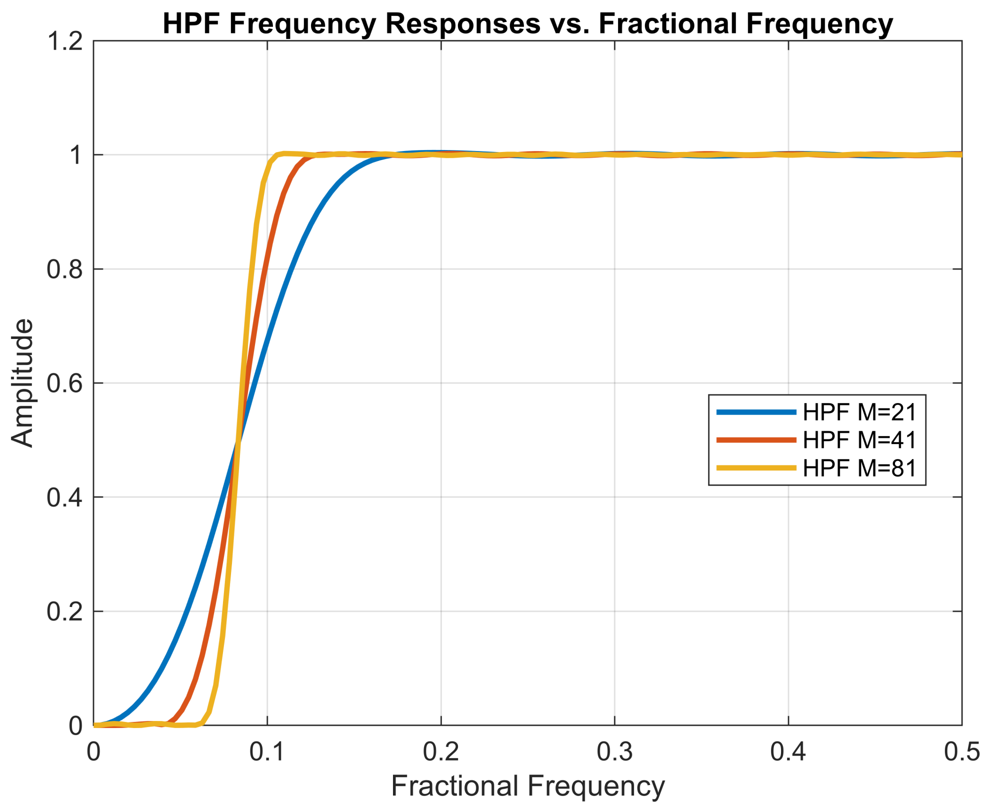
The fractional frequency can be found if you assume that the first frequency is 0 and the last frequency is 255 (256 elements). The last frequency corresponds to the sample frequency or a fractional frequency of 1.0

10) PLOT P5 and P6. Repeat the plots of the LPF frequency response and the HPF frequency response but use the fractional frequency for the X-axis. Limit the x-axis to a range from 0 to 0.5, the Nyquist rate.

```
%
%  Create a row vector that represents the fractional frequency from 0 to 1
%  of the 256 element DFT. Re-plot the LPF and HPF frequency responses vs
%  the fractional frequency.

%  Place your solution here
N = 256;
frac = (0:N-1)/N;
figure;
plot(frac, abs(mLPF_21),'LineWidth',2); hold on;
plot(frac, abs(mLPF_41),'LineWidth',2);
plot(frac, abs(mLPF_81),'LineWidth',2);
legend('LPF M=21','LPF M=41','LPF M=81','Location','best');
xlim([0,0.5]);
xlabel('Fractional Frequency'); ylabel('Amplitude');
title('LPF Frequency Responses vs. Fractional Frequency');
grid on; hold off;
```



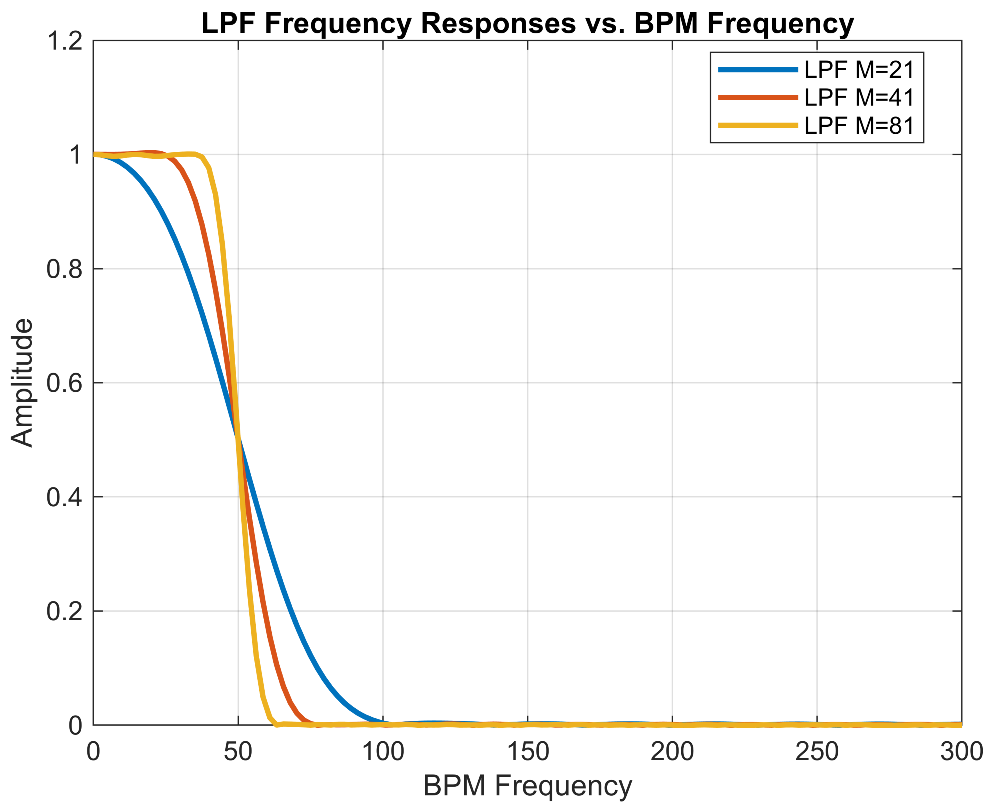
```
figure;
plot(frac, abs(mHPF_21),'LineWidth',2); hold on;
plot(frac, abs(mHPF_41),'LineWidth',2);
plot(frac, abs(mHPF_81),'LineWidth',2);
legend('HPF M=21','HPF M=41','HPF M=81','Location','best');
xlim([0,0.5]);
xlabel('Fractional Frequency'); ylabel('Amplitude');
title('HPF Frequency Responses vs. Fractional Frequency');
grid on; hold off;
```



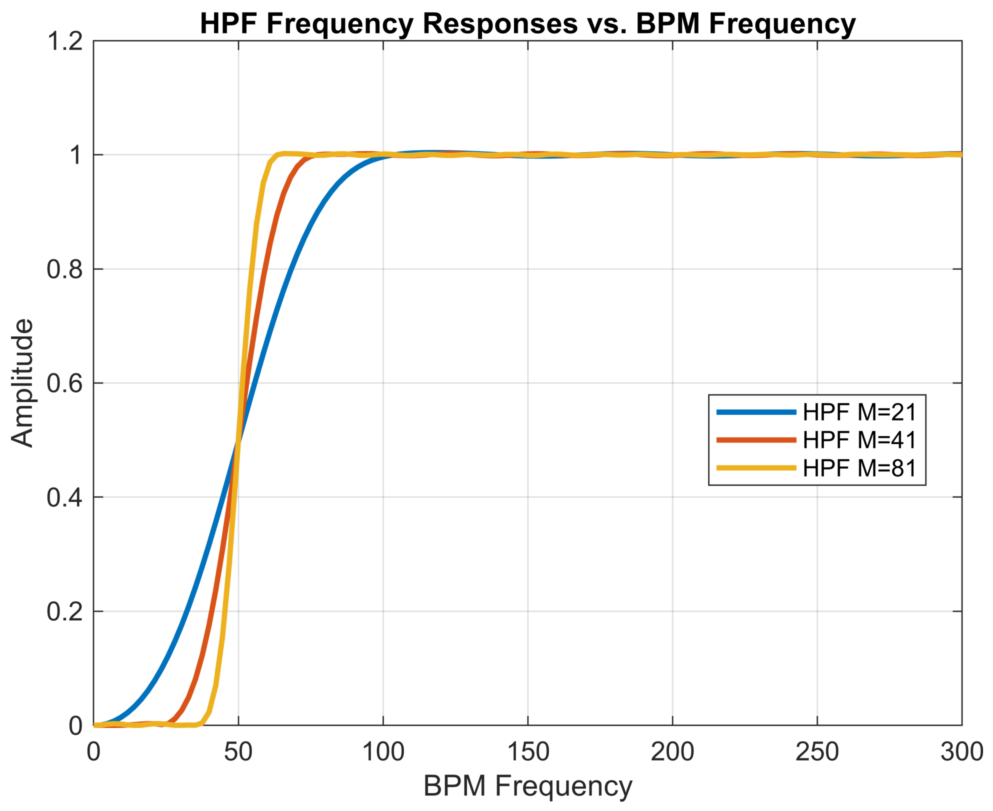
11) In MATLAB create another row vector that represents the frequency in BPM for each point in the DFT. Recall that the sample rate in BPM is 600.

12) PLOT P7 and P8: Repeat the plots from above but use the frequency in BPM for the X axis. Limit the range of the x-axis to 0 to 300 bpm.

```
%
%  Create a row vector that represents the frequency in BPM from 0 to 600
%  of the 256 element DFT. Re-plot the LPF and HPF frequency responses vs
%  the frequency in BPM.

%  Place your solution here
bpmAxis = (0:N-1)*(600/N);

figure;
plot(bpmAxis, abs(mLPF_21),'LineWidth',2); hold on;
plot(bpmAxis, abs(mLPF_41),'LineWidth',2);
plot(bpmAxis, abs(mLPF_81),'LineWidth',2);
legend('LPF M=21','LPF M=41','LPF M=81','Location','best');
xlim([0,300]);
xlabel('BPM Frequency'); ylabel('Amplitude');
title('LPF Frequency Responses vs. BPM Frequency');
grid on; hold off;
```



```
figure;
plot(bpmAxis, abs(mHPF_21),'LineWidth',2); hold on;
plot(bpmAxis, abs(mHPF_41),'LineWidth',2);
plot(bpmAxis, abs(mHPF_81),'LineWidth',2);
legend('HPF M=21','HPF M=41','HPF M=81','Location','best');
xlim([0,300]);
xlabel('BPM Frequency'); ylabel('Amplitude');
title('HPF Frequency Responses vs. BPM Frequency');
grid on; hold off;
```



11) Question Q2: Answer the following question in the narrative of your report. As the filter length of the FIR filter increases, what is the impact on the sharpness of the filter's cutoff between passband and stop band? What is the 10% to 90% transition bandwidth of each filter? This is the bandwidth between the points that are at 10% of the magnitude and 90% of the final magnitude (also See Smith for definition)

Answer Q2: When the filter length is increased, from 21 to 41 to 81, the sharper the filter's cutoff becomes when it transitions from the passband to the stopband. For when the length is M = 21, the transition bandwidth is more gradual taking a larger chunk of the frequency axis, sitting around 56 points of magnitude difference. M = 41, narrows the transition bandwidth, sitting around 26 points of magnitude difference. M = 81, is the narrowest bandwidth transition sitting around around approximately 12 points of magnitude difference.

Section 2 – FIR Sinc Filters: Band Pass Filter (BPF) and Band Stop Filter (BSF)

Overview

NOTE: In this section you will use the CaptureArduinoData.m function so you can save the output from the Arduino to MATLAB.

For breathing rate analysis, we may wish to identify if a signal has frequencies within a safe range. We may also be interested if a signal has frequencies outside a safe range.

To identify frequencies within a range we can use a Band Pass Filter (BPF) that passes only the frequencies within an acceptable range for analysis. We can then analyze what is coming through the filter and determine if the signals are at an acceptable level.

To identify frequencies that are outside of a safe range we can use a Band Stop Filter (BSF). A band stop filter will attenuate all signals within a range of frequencies and allow all others to pass through. If we attenuate the desired frequencies and allow the undesired frequencies we can analyze whether we have an unacceptable level of undesired frequencies.

Bandpass filters may be constructed by placing a lowpass filter and a highpass filter in cascade (series) with one another.as shown from the following figures from Smith. Recall that for a BPF the lowpass filter must have a corner frequency that is higher than the corner frequency of the highpass filter. Otherwise nothing will pass through the filter. The lowpass filter defines the UPPER cutoff frequency of the BPF. The highpass filter defines the LOWER cutoff frequency of the BPF.

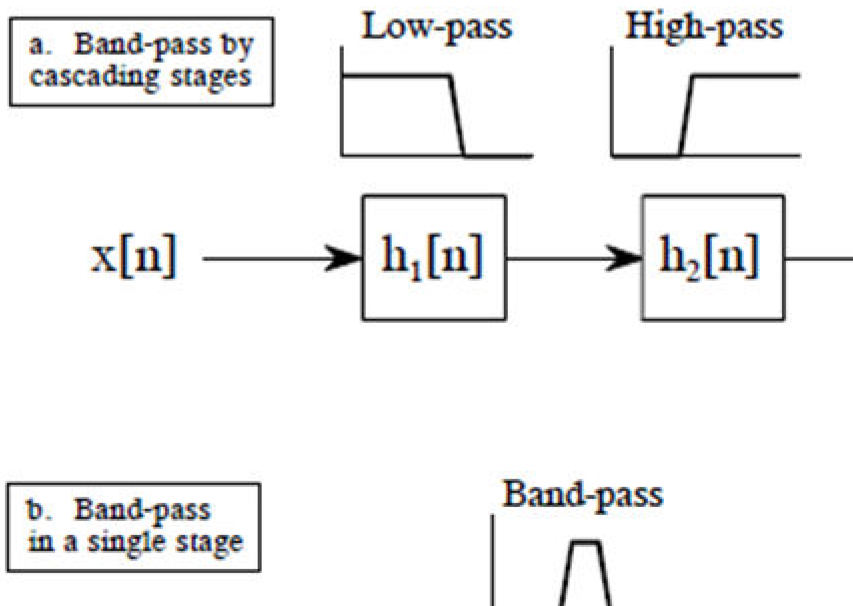


FIGURE 14-8
Designing a band-pass filter. As shown in (a), a band-pass filter can be formed by cascading a low-pass filter and a high-pass filter. This can be reduced to a single stage, shown in (b). The filter kernel of the single stage is equal to the *convolution* of the low-pass and high-pass filter kernels.

Bandstop filters may be constructed by placing a lowpass filter and highpass filter in parallel with one another and summing their outputs. These are also shown in the figure below from Smith. Recall that for a BSF the lowpass filter must have a corner frequency that is lower than the corner frequency of the highpass filter. Otherwise nothing will attenuated by the filter.

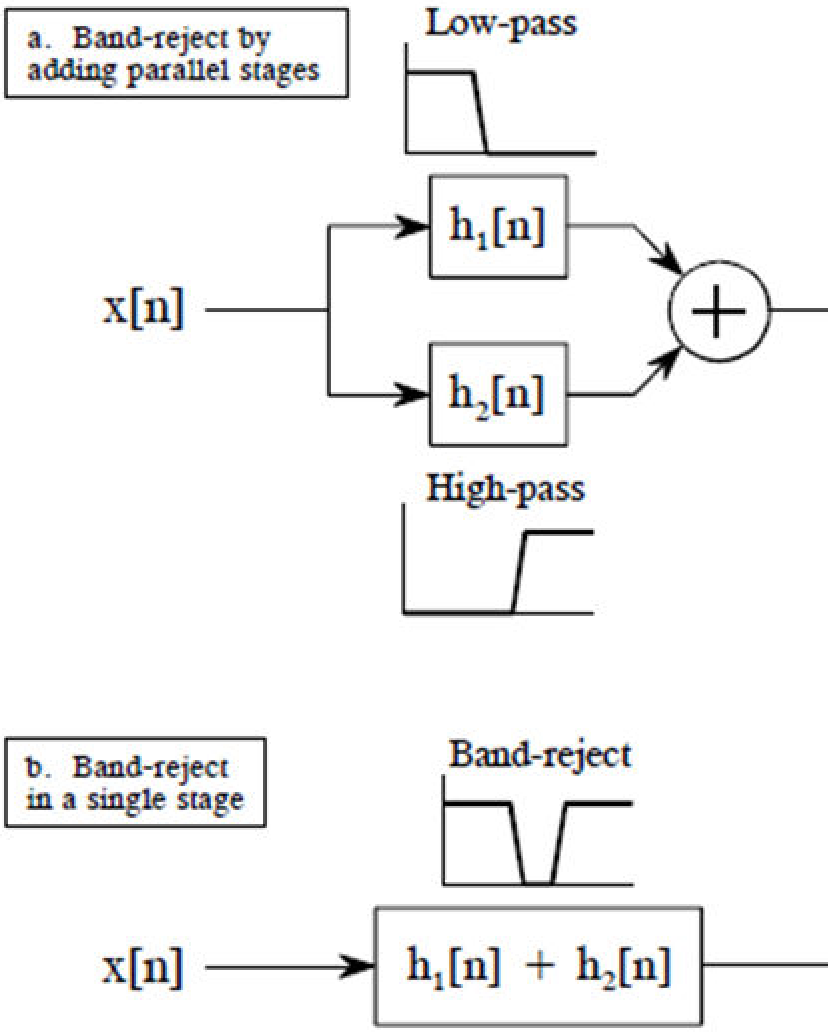


FIGURE 14-9
Designing a band-reject filter. As shown in (a), a band-reject filter is formed by the parallel combination of a low-pass filter and a high-pass filter with their outputs added. Figure (b) shows this reduced to a single stage, with the filter kernel found by *adding* the low-pass and high-pass filter kernels.

The code in the code base for Lab 7 section 02 implements prototype LPF and HPF sections in fixed point math by using the windowed SINC filters as in Section 1. You can use the MATLAB FIR_Designer function to create impulse responses for the LPF and HPF filters. The function will print out a C style header with the impulse response. You can copy this code into the appropriate location in the sketch.

Using the MATLAB FIR_Designer.m function you can create impulse responses for LPF and HPF filters with different corner frequencies and impulse response lengths. You can adjust the corner frequencies and transition bandwidths (by adjusting the impulse response lengths) of the LPF and HPF sections as needed and then combine the filters to build the specified BPF and BSF functions.

Load and execute the code from the file "Lab7_FIR_Section_02_Base_Code.ino". Use it to build and test BPF and BSF filters per the following procedure.

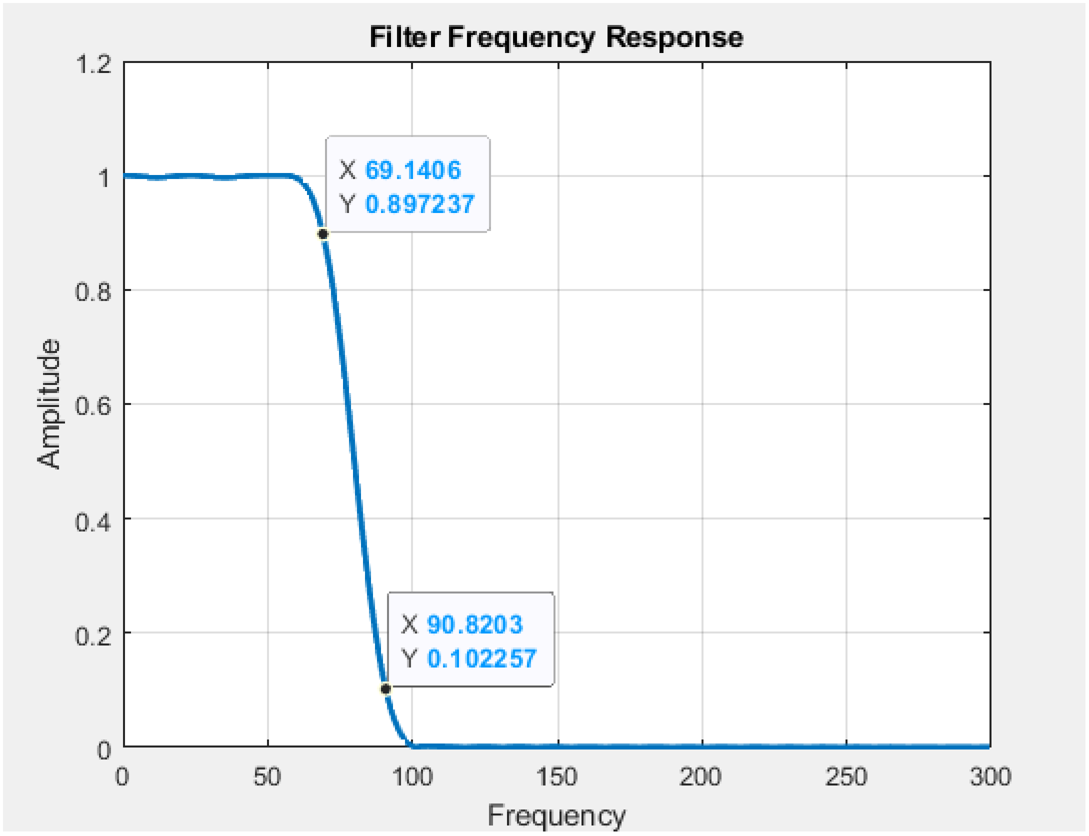
Procedure/Reporting

1) Using the FIR_Designer MATLAB function tool, design and test the FIR BPF whose specifications are in the code comments and repeated below.

Filter Specifications

Filter 1 -- Bandpass Filter

BPF with a lower corner frequency of 15 bpm and a high corner frequency of 50 bpm. The approximate 10% to 90% transition BW = 10 bpm The transition BW is defined as the width (in BPM) of the filter from the 10% magnitude frequency to the 90% magnitude frequency. The example below shows a LPF with a 10% to 90% transition BW of 90.8 - 69.1 = 21.7 BPM



<u>You will modify the the filters using FIR_Designer tool to create the individual LPF and HPF filters and then combine them appropriately to create a BPF response. This will be an iterative process using MATLAB FIR_Designer tool. See the figures above on how to combine the filters to create a bandstop response.</u>

2) Create the individual LPF and HPF filters required to make a BPF using the MATLAB FIR_Designer tool. In MATLAB combine the two filter impulse responses appropriately to create the desired filter response. Recall that for a BPF the lowpass filter must have a corner frequency that is higher than the

corner frequency of the highpass filter. Otherwise nothing will pass through the filter. The lowpass filter defines the UPPER cutoff frequency of the BPF. The highpass filter defines the LOWER cutoff frequency of the BPF.

The FIR_Designer tool will print out the value of HFXPT and the impulse response in a format that is easy to copy into MATLAB.

As a reminder the call to the FIR_Designer function is shown below. The order of the filter is defined using the Name/Value pair 'nOrder' (e.g. 'nOrder',30). The 50% cutoff frequency of the filter is defined by the Name/Value pair 'cutBPM' (e.g 'cutBPM',50). In order to have the C-code header printed you need to specify the argument 'FxdPoint' to true (e.g. 'FxdPoint',true).

```
%  Example call to FIR_Designer function
%  h = FIR_Designer('nOrder',31,'cutBPM',50,'FxdPoint',true,'PlotResponses',false);
```

NOTE: You may run the FIR_Designer tool in the the MATLAB Live Script. HOWEVER, be sure to use the Name/Value pair 'PlotResponses',false. This will prevent the tool from putting plots in the script! Also place a semicolon after the command to suppress printing the values of the impulse response in the script (there will still be one output with the impulse response however).

I suggest that you compute all three frequency responses (LPF, HPF and BPF) and plotting them on the same graph for the best insight.

The impulse response of the BPF is computed by convolving the LPF impulse response with the HPF impulse response. Use the MATLAB conv function to perform the convolution as shown in the example

$$h_{BPF}[n] = h_{LPF}[n] * h_{HPF}[n]$$

```
%  h_BPF = conv(h_LPF, h_HPF)
```

```
%  Put your solution here.  Use the FIR_Designer tool, you may run it in the
%  the Live Script but be sure to set 'PlotResponses' to false and add a
%  semicolon to the end of the command (you'll still get one print out).
%
%  This will make it easier to iterate the filter parameters to get the
%  BPF response that you want.
%
%  Try computing all three frequency responses (LPF, HPF and BPF) and plotting
%  them all on the same graph for best insight.  Then adjust your
%  parameters
%


%  Place your solution here
```

```
h_LPF = FIR_Designer('nOrder',61, 'cutBPM',50, 'Type','lpf', 'FxdPoint',true,
'PlotResponses',false);
```

```
    // LPF FIR Filter Coefficients MFILT = 61, Fc = 50
    const int HFXPT = 4096, MFILT = 61;
    int h[] = {0, 2, 4, 5, 5, 4, 0, -6, -12, -17, -17, -12, 0, 17,
    35, 47, 47, 32, 0, -43, -87, -117, -119, -82, 0, 122, 270, 424,
    558, 649, 681, 649, 558, 424, 270, 122, 0, -82, -119, -117, -87, -43,
    0, 32, 47, 47, 35, 17, 0, -12, -17, -17, -12, -6, 0, 4,
    5, 5, 4, 2, 0};

 Fixed Point scale and FIR Filter Coefficients for easy copy to MATLAB

HFXPT = 4096;

h = [0, 2, 4, 5, 5, 4, 0, -6, -12, -17, -17, -12, 0, 17,...
35, 47, 47, 32, 0, -43, -87, -117, -119, -82, 0, 122, 270, 424,...
558, 649, 681, 649, 558, 424, 270, 122, 0, -82, -119, -117, -87, -43,...
0, 32, 47, 47, 35, 17, 0, -12, -17, -17, -12, -6, 0, 4,...
5, 5, 4, 2, 0];
```

```
h_HPF = FIR_Designer('nOrder',61, 'cutBPM',15, 'Type','hpf','FxdPoint',true,
'PlotResponses',false);
```

```
    // HPF FIR Filter Coefficients MFILT = 61, Fc = 15
    const int HFXPT = 2048, MFILT = 61;
    int h[] = {2, 2, 2, 2, 2, 3, 3, 3, 2, 1, 0, -2, -5, -8,
    -12, -17, -23, -29, -36, -43, -51, -59, -67, -75, -82, -88, -94, -98,
    -102, -104, 1944, -104, -102, -98, -94, -88, -82, -75, -67, -59, -51, -43,
    -36, -29, -23, -17, -12, -8, -5, -2, 0, 1, 2, 3, 3, 3,
    2, 2, 2, 2, 2};

 Fixed Point scale and FIR Filter Coefficients for easy copy to MATLAB

HFXPT = 2048;

h = [2, 2, 2, 2, 2, 3, 3, 3, 2, 1, 0, -2, -5, -8,...
-12, -17, -23, -29, -36, -43, -51, -59, -67, -75, -82, -88, -94, -98,...
-102, -104, 1944, -104, -102, -98, -94, -88, -82, -75, -67, -59, -51, -43,...
-36, -29, -23, -17, -12, -8, -5, -2, 0, 1, 2, 3, 3, 3,...
2, 2, 2, 2, 2];
```

Once you have an LPF and HPF impulse response then plot the impulse response of the BPF and plot the BPF frequency response for your report

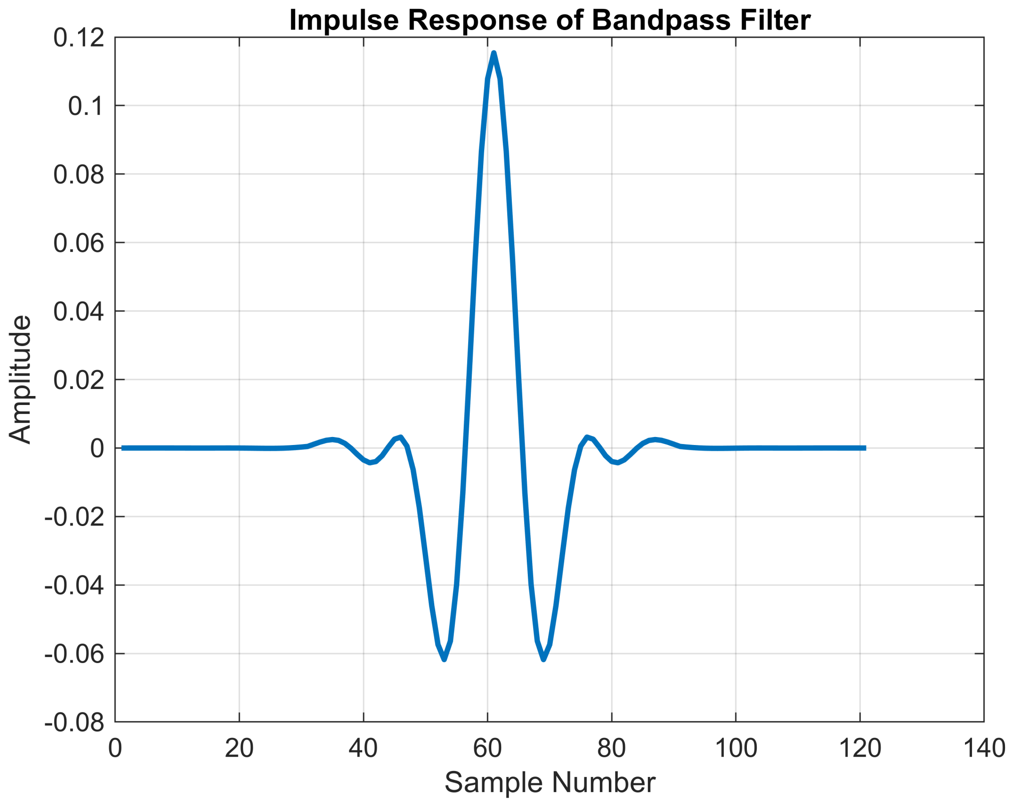
3) PLOT P9 -- Plot the impulse response of the BPF.

4) PLOT P10 -- Find the frequency response of the BPF using an FFT of length 256. Plot the frequency response versus frequency in BPM

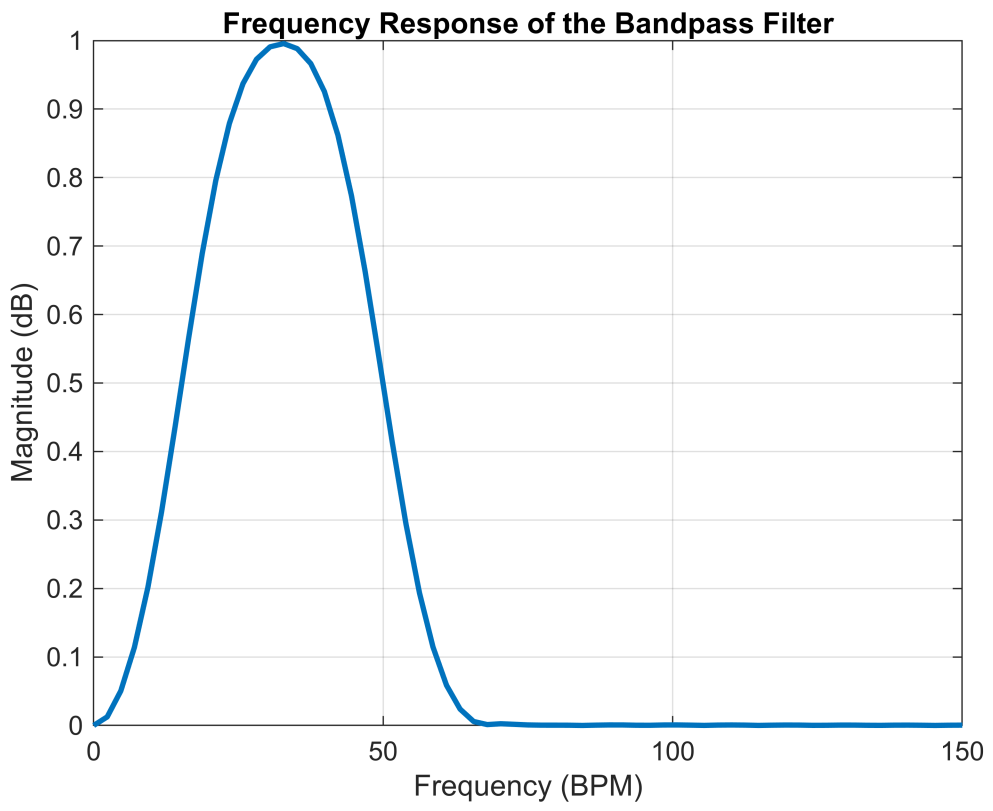
Now print out the C-header by setting the Name/Value pair 'PrintHeader' to true and the Name/Value pair 'FxdPoint' to true. Copy your LPF and HPF commands below with the arguments set correctly.

Be sure to set the parameters to the values you chose above. The C-Header can be copied into the Arduino C-Code to run the filter on the Arduino

```
%
h_BPF = conv(h_LPF, h_HPF);
figure
plot(h_BPF, 'LineWidth',2)
grid on
title('Impulse Response of Bandpass Filter')
xlabel('Sample Number')
ylabel('Amplitude')
```



```
[m_BPF] = fft(h_BPF, 256);
freqAxis = (0:256-1)*(600/256);
figure
plot(freqAxis, abs(m_BPF), 'LineWidth',2)
grid on
title('Frequency Response of the Bandpass Filter')
xlabel('Frequency (BPM)')
ylabel('Magnitude (dB)')
xlim([0 150])
```



2) Once the design is completed in MATLAB copy the C code header that is printed by the FIR_Designer tool for the LPF and the HPF filter into the generic filter convolution functions FIR_A and FIR_B that are in the code. In the "loop" portion of the code modify the code so that a BPF is implemented. Refer to the block diagrams at the beginning of this section to create a BPF.

!!!! IMPORTANT !!!!

In the C-code the two impulse responses are NOT convolved to create the BPF response. Rather the two filters (LPF and HPF) are run in series with the output of one filter feeding the input of the second filter. This is equivalent to passing the input to the BPF impulse response created through convolution.

3) In your report include the following:

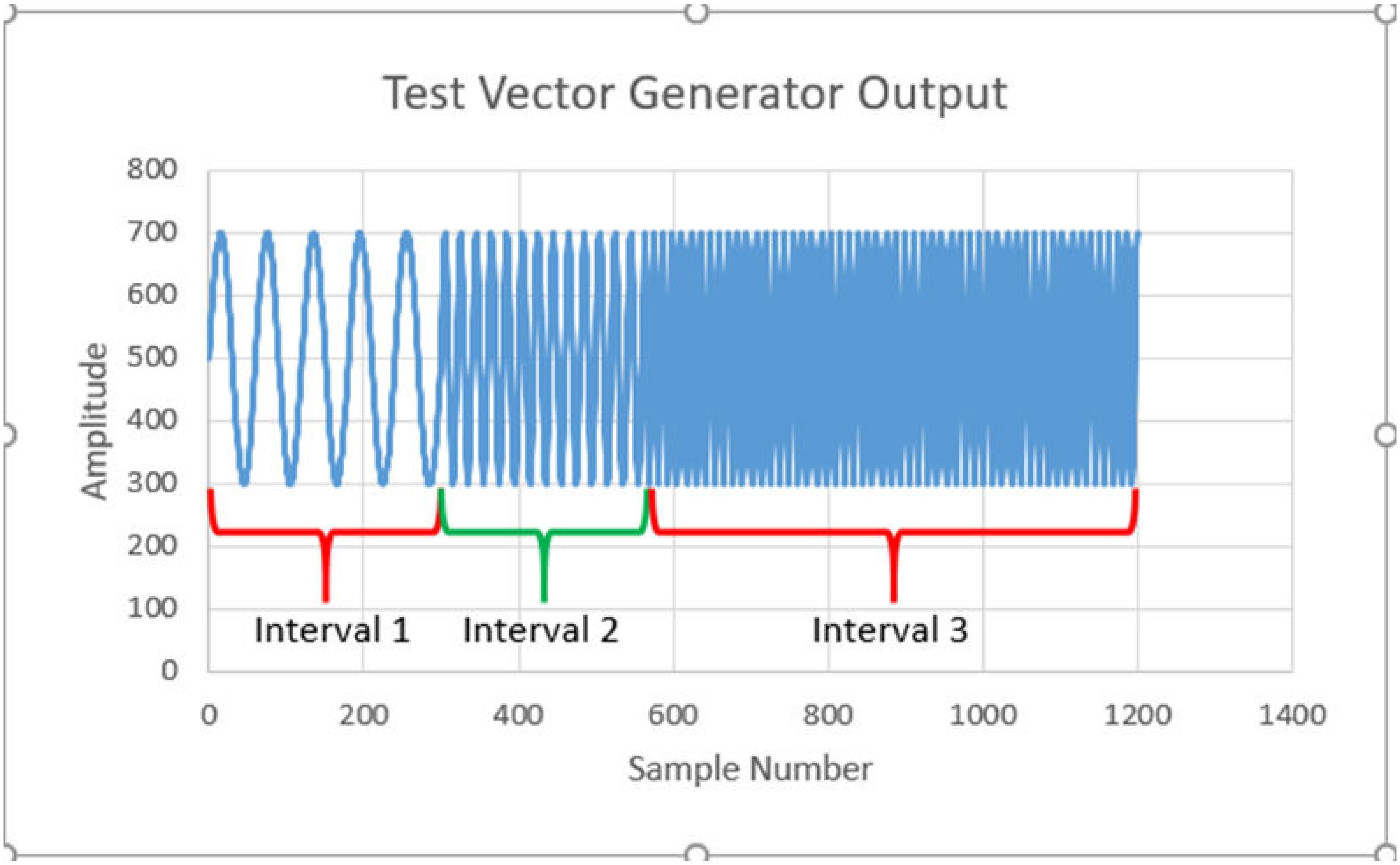
a) The impulse response of the BPF filter

b) The frequency response of the BPF filter using the FFT

c) The 10% to 90% transition bandwidth of the BPF filter

4) A test vector generator function is written in the C code to allow testing of the filters using the Arduino. Test your filter using the built-in test vector generator. The test vector generator creates three sinusoids in three intervals of time. They may have different amplitudes if desired. The parameters of the test vector are adjusted in the code as shown below.

```
// Variable rate periodic input
// Specify segment amplitude, bpm rate, interval seconds.
// Intervals trimmed for nearest cycle ending zero crossing.

const float AMP1 = 2.0, BPM1 = 10.0, TSEC1 = 24.0;
const float AMP2 = 2.0, BPM2 = 30.0, TSEC2 = 24.0;
const float AMP3 = 2.0, BPM3 = 50.0, TSEC3 = 24.0;
```



5) Choose frequencies and interval lengths in the test vector generator to validate BPF filter performance. Think about what frequencies should be used to verify that your design meets the filter specifications. You want to verify that the filter passes a certain frequency range. You will want to verify

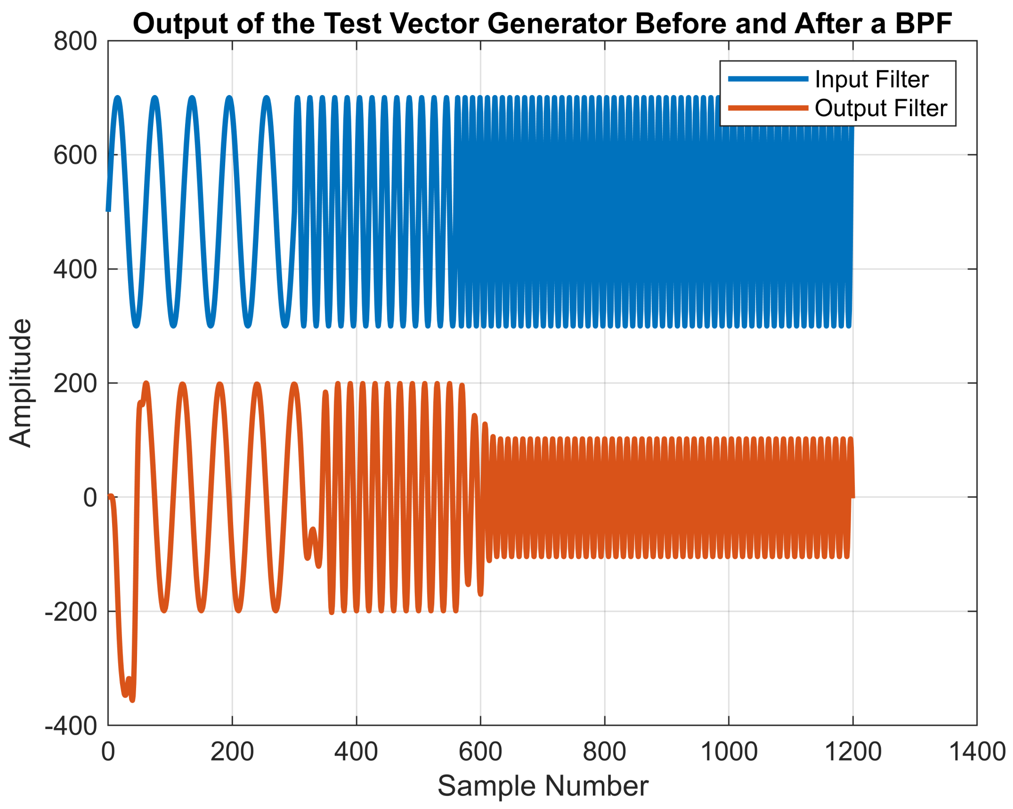
that the filter also attenuates a particular frequency range. You may want to test your filter with different sets of the three frequencies.

Use the CaptureArduinoData function to capture data from the output of the Arduino and save to a file.

PLOT P11 Make a single plot of the input and the output of the bandpass filter.

6) Compute the attenuation of the frequencies that you used in your test vector generator. Include a well-annotated chart of these results in your report showing the test frequencies and the attenuation achieved. Note the stopband attenuation and the passband gains as measured in the time domain.

```
%
%  Capture the output of the Arduino using the test vector generator
%  output (Use CaptureArduinoData function).  Plot both the input and the
%  output of the filter
%
load("data/section2DataBPF.mat");
seconds = outData(:,1);
inputFilter = outData(:,2);
outputFilter = outData(:,3);

figure;
plot(seconds, inputFilter, 'LineWidth',2); hold on;
plot(seconds, outputFilter, 'LineWidth',2);
legend('Input Filter','Output Filter'); xlabel('Sample Number');
ylabel('Amplitude');
title("Output of the Test Vector Generator Before and After a BPF");
xlim([0 1400]);grid on; hold off;
```



BandStop Filter Design

7) Build and test the FIR BSF whose specifications are in the code comments and repeated below.

!!!! IMPORTANT !!!!

In order to add the two impulse responses together, the fixed point scale value HFXPT must be the same for both impulse responses. In the FIR_Designer tool you can set the value of the fixed point scale value by adding the Name, Value pair "FxdPointScale" followed by the value that is desired. Example syntax is show below:

```
% h = FIR_Designer('nOrder', 21,'cutBPM',400, 'Type','HPF','FxdPoint',true,
'FxdPointScale',4096);
```

Filter 2 -- Bandstop Filter

BSF with a lower corner frequency of 15 bpm and a high corner frequency of 50 bpm. The 10% to 90% transition BW = 20 bpm

!! STRONG SUGGESTION !!

Copy the sketch that you used for the BPF and rename it with an appropriate filename to indicate that it is for a bandstop filter. This will allow you to go back and use your bandpass sketch if needed

Using the FIR_Designer tool in MATLAB, design an LPF and HPF to meet the requirements of the bandstop filter. Recall that for a BSF, the LPF will have a corner frequency that is LOWER than the HPF.

You will modify the code in the sketch to include the individual LPF and HPF filters as you did with the BPF. Then combine them appropriately in the 'loop' section of the code to create a BSF response. See the figures above on how to combine the filters to create a bandstop response.

PLOT P12, P13 -- Plot the impulse response and the frequency response of the band stop filter and include them in the your report.

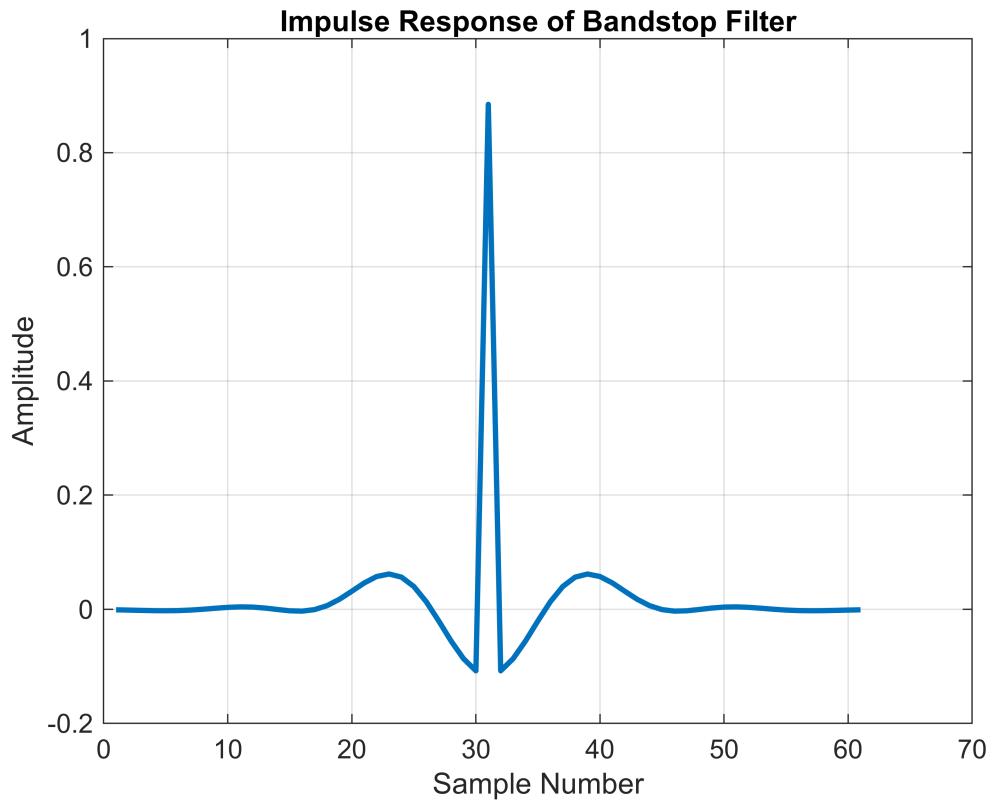
```
%  PUT YOUR SOLUTION HERE
%
%  Use the FIR_Designer tool, you may run it in the
%  the Live Script but be sure to set 'PlotResponses' to false and add a
%  semicolon to the end of the command (you'll still get one print out).
%
%  This will make it easier to iterate the filter parameters to get the
%  BPF response that you want.
%
%  Try computing all three frequency responses (LPF, HPF and BPF) and plotting
%  them all on the same graph for best insight.  Then adjust your
%  parameters
%



%  Place your solution here
hLPF_bsf = FIR_Designer('nOrder',61, 'cutBPM',15, 'Type','lpf',
'PlotResponses',false, 'FxdPoint',true);
```

```
    // LPF FIR Filter Coefficients MFILT = 61, Fc = 15
    const int HFXPT = 2048, MFILT = 61;
    int h[] = {-2, -2, -2, -2, -2, -3, -3, -3, -2, -1, 0, 2, 5, 8,
    12, 17, 23, 29, 36, 43, 51, 59, 67, 75, 82, 88, 94, 98,
    102, 104, 104, 104, 102, 98, 94, 88, 82, 75, 67, 59, 51, 43,
    36, 29, 23, 17, 12, 8, 5, 2, 0, -1, -2, -3, -3, -3,
    -2, -2, -2, -2, -2};

 Fixed Point scale and FIR Filter Coefficients for easy copy to MATLAB

HFXPT = 2048;

h = [-2, -2, -2, -2, -2, -3, -3, -3, -2, -1, 0, 2, 5, 8,...
12, 17, 23, 29, 36, 43, 51, 59, 67, 75, 82, 88, 94, 98,...
102, 104, 104, 104, 102, 98, 94, 88, 82, 75, 67, 59, 51, 43,...
36, 29, 23, 17, 12, 8, 5, 2, 0, -1, -2, -3, -3, -3,...
-2, -2, -2, -2, -2];
```

```
hHPF_bsf = FIR_Designer('nOrder',61, 'cutBPM',50, 'Type','hpf',
'PlotResponses',false, 'FxdPoint',true);
```

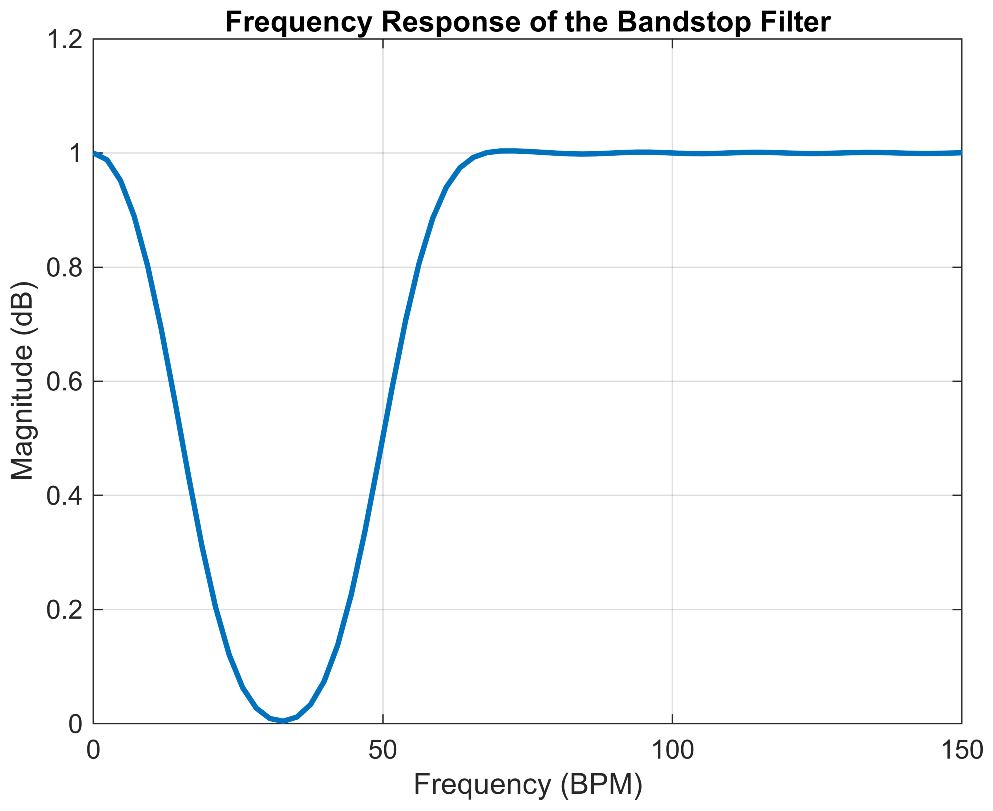
```
    // HPF FIR Filter Coefficients MFILT = 61, Fc = 50
    const int HFXPT = 4096, MFILT = 61;
    int h[] = {0, -2, -4, -5, -5, -4, 0, 6, 12, 17, 17, 12, 0, -17,
    -35, -47, -47, -32, 0, 43, 87, 117, 119, 82, 0, -122, -270, -424,
    -558, -649, 3415, -649, -558, -424, -270, -122, 0, 82, 119, 117, 87, 43,
    0, -32, -47, -47, -35, -17, 0, 12, 17, 17, 12, 6, 0, -4,
    -5, -5, -4, -2, 0};

 Fixed Point scale and FIR Filter Coefficients for easy copy to MATLAB

HFXPT = 4096;

h = [0, -2, -4, -5, -5, -4, 0, 6, 12, 17, 17, 12, 0, -17,...
-35, -47, -47, -32, 0, 43, 87, 117, 119, 82, 0, -122, -270, -424,...
-558, -649, 3415, -649, -558, -424, -270, -122, 0, 82, 119, 117, 87, 43,...
0, -32, -47, -47, -35, -17, 0, 12, 17, 17, 12, 6, 0, -4,...
-5, -5, -4, -2, 0];
```

```
h_BSF = conv(hLPF_bsf, [1]) + conv(hHPF_bsf, [1]);
figure
plot(h_BSF, 'LineWidth',2)
grid on
title('Impulse Response of Bandstop Filter')
xlabel('Sample Number')
ylabel('Amplitude')
```



```
[m_BSF] = fft(h_BSF, 256);
freqAxis = (0:256-1)*(600/256);
figure
plot(freqAxis, abs(m_BSF), 'LineWidth',2)
```

```
grid on
title('Frequency Response of the Bandstop Filter')
xlabel('Frequency (BPM)')
ylabel('Magnitude (dB)')
xlim([0 150]);
```



Now print out the C-header by setting the Name/Value pair 'PrintHeader' to true and the Name/Value pair 'FxdPoint' to true. Repeat your LPF and HPF commands below with the arguments set correctly.

Be sure to set the parameters to the values you chose above. The C-Header can be copied into the Arduino C-Code to run the filter on the Arduino

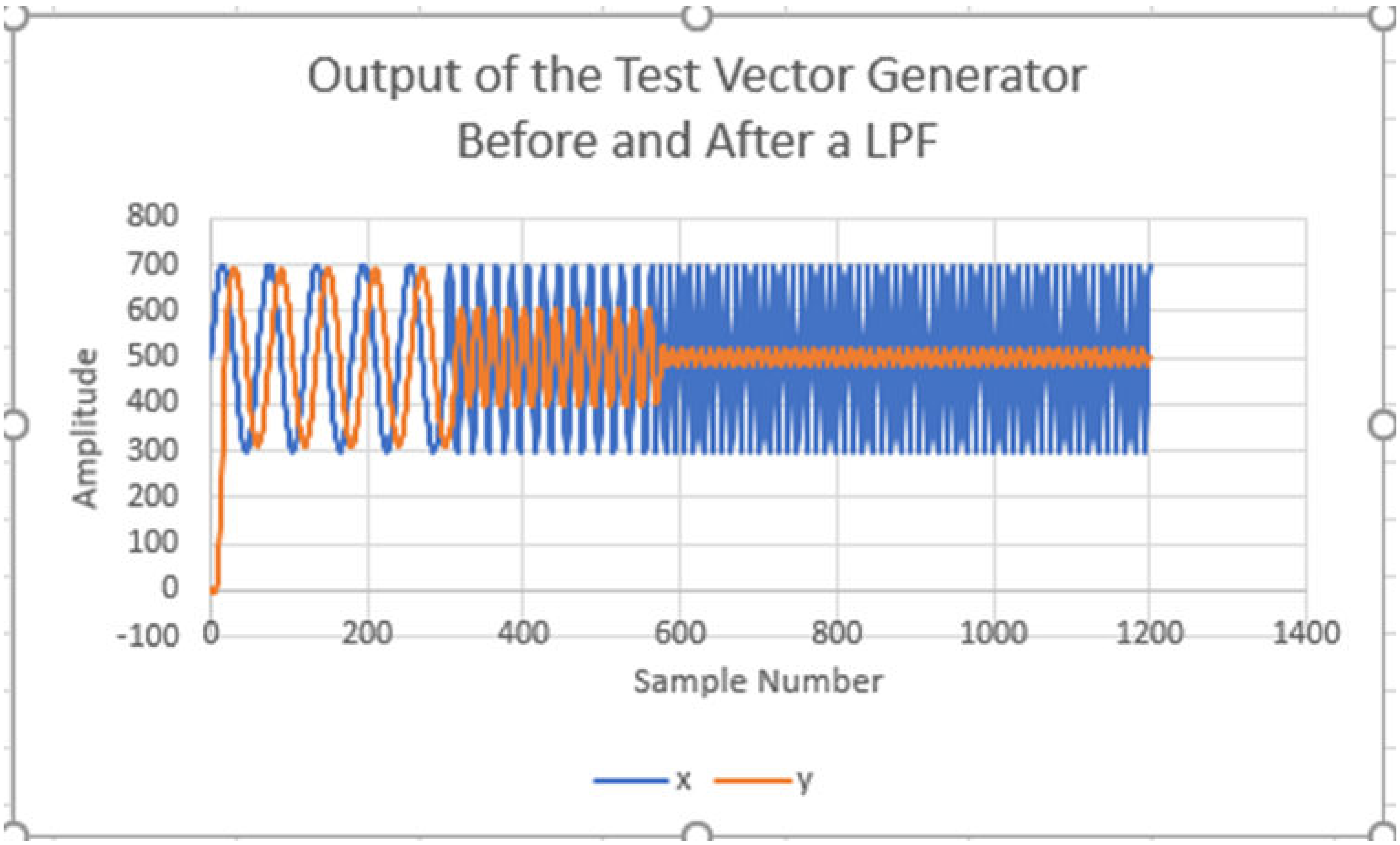
```
%
% hLPF = FIR_Designer('nOrder',
XX,'cutBPM',XX,'PlotResponses',false,'PrintHeader',true, 'FxdPoint',true);
% hHPF = FIR_Designer('nOrder',
XX,'cutBPM',XX,'Type','hpf','PlotResponses',false,'PrintHeader',true, 'FxdPoint',
true);
```

8) Test your BSF running on the Arduino using the output of the test vector generator as before. Choose appropriate frequencies for the generator to properly test the filter performance.

PLOT P14 -- Plot the input and output of the filter using the test vector generator on the same graph and include the plot in your report.

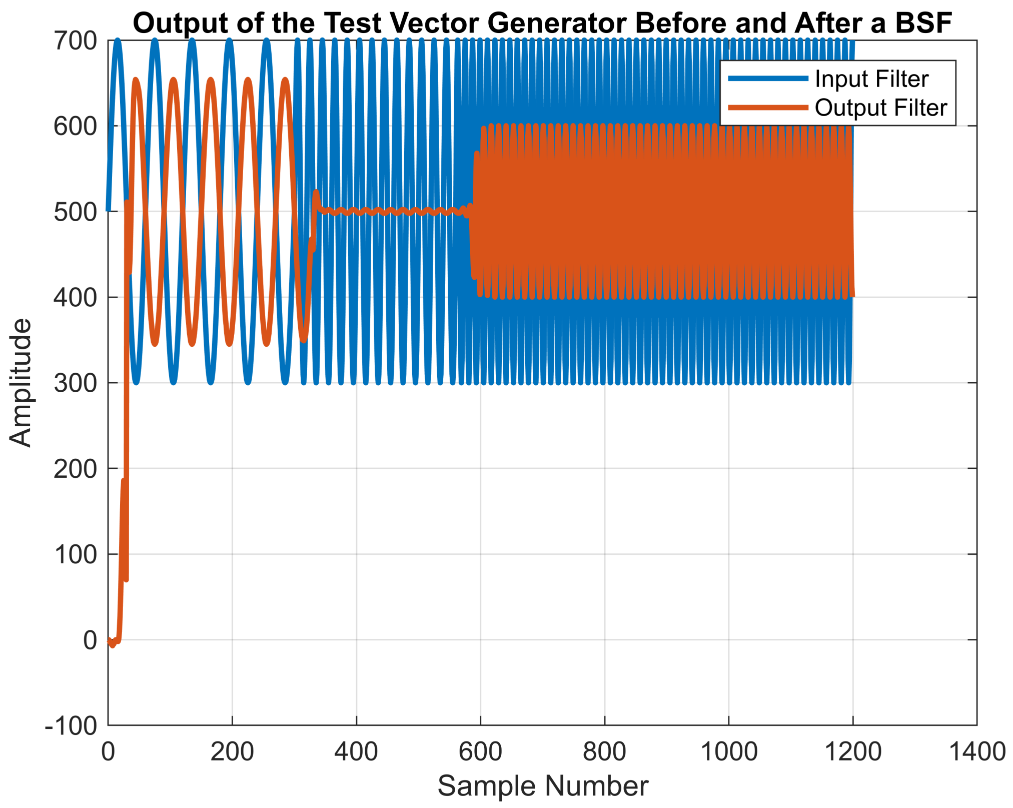
9) Demonstrate appropriate band stop performance in the time domain. Include a well-annotated chart of the result in your report. Note the stopband and passband gains as measured in the time domain.

Here is an example of the output of the test vector generator and the output of a LPF that follows the test vector generator. Note that as the frequency of the output increases, the low pass filter attenuates the signal.



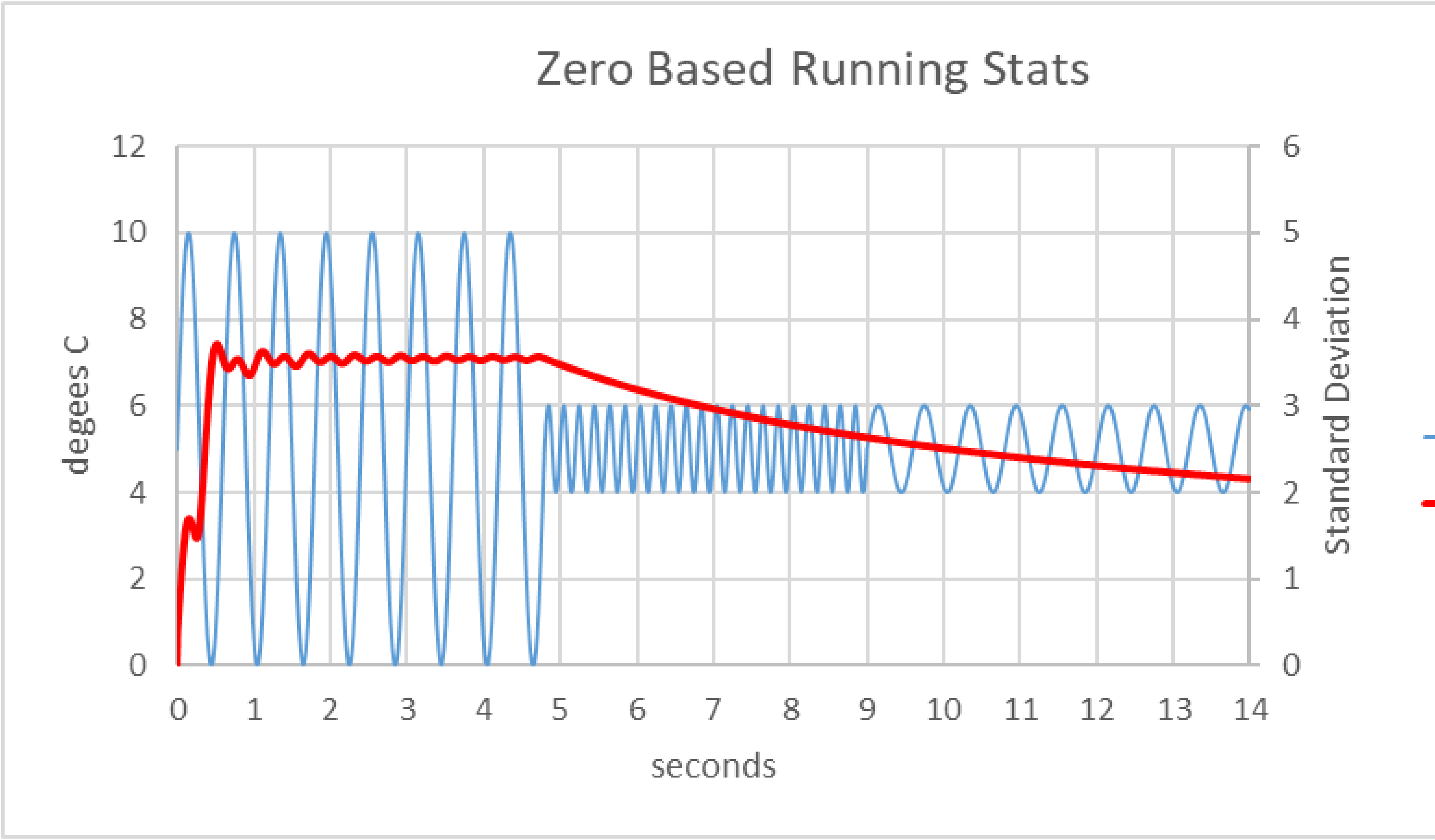
```
%
%  Capture the output of the Arduino using the test vector generator
%  output (Use CaptureArduinoData function).  Plot both the input and the
%  output of the filter
%


%  Place your solution here
load("data/section2DataBSF.mat");
```

```
seconds = outData(:,1);
inputFilter = outData(:,2);
outputFilter = outData(:,3);

figure;
plot(seconds, inputFilter, 'LineWidth',2); hold on;
plot(seconds, outputFilter, 'LineWidth',2);
legend('Input Filter','Output Filter'); xlabel('Sample Number');
ylabel('Amplitude');
title("Output of the Test Vector Generator Before and After a BSF");
xlim([0 1400]);grid on; hold off;
```



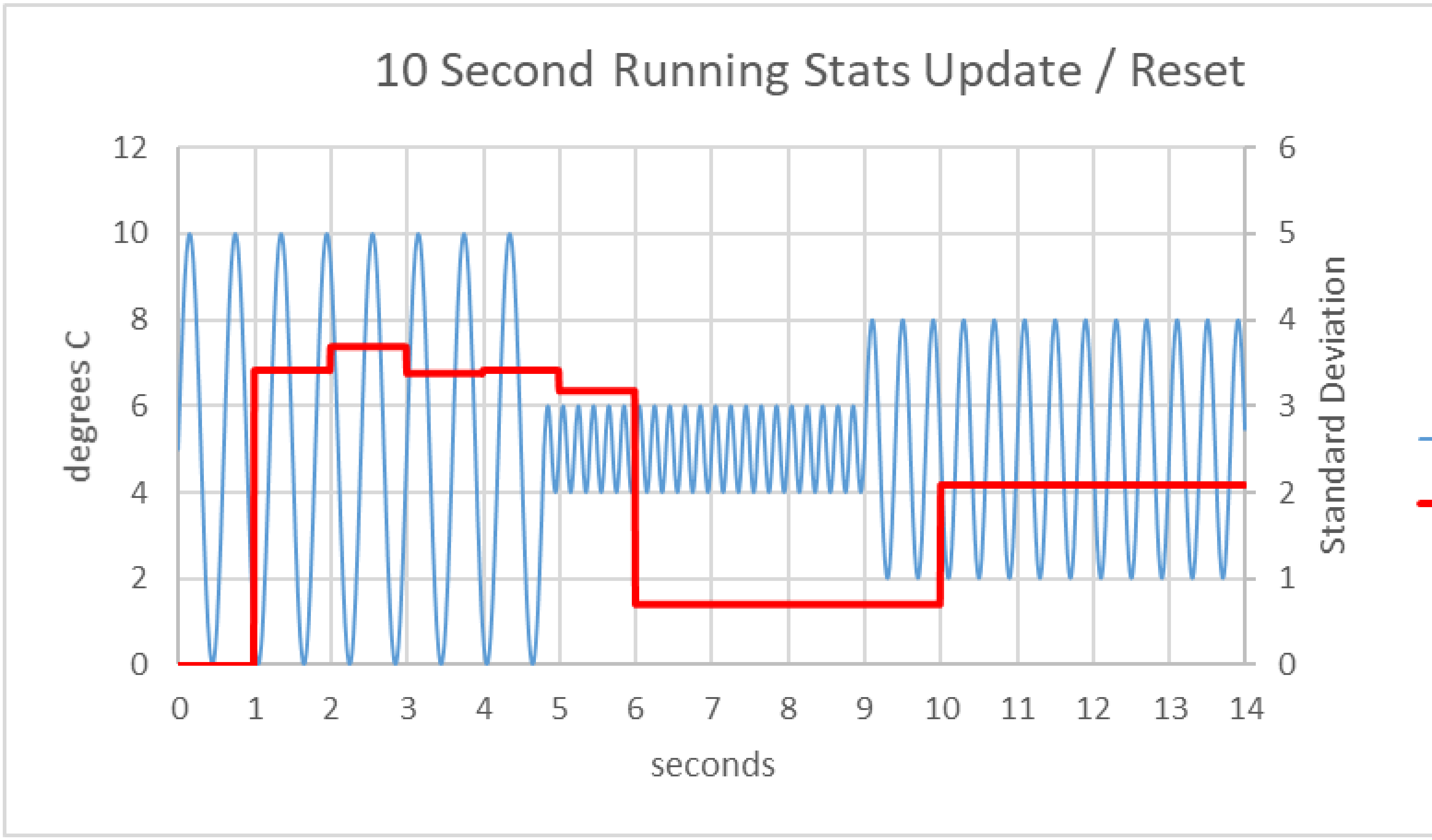
Section 3 - Resettable Recursive Variance Calculations

In Lab 2, you implemented a numerically stable, recursive algorithm to calculate the running variance of a signal. See a typical result below. Note the lengthy decay toward the true variance of 0.707 after the signal amplitude transition. Run the sketch from the code base file "Lab7_FIR_Section_03_Base_Code.ino" to reacquaint yourself with the algorithm.



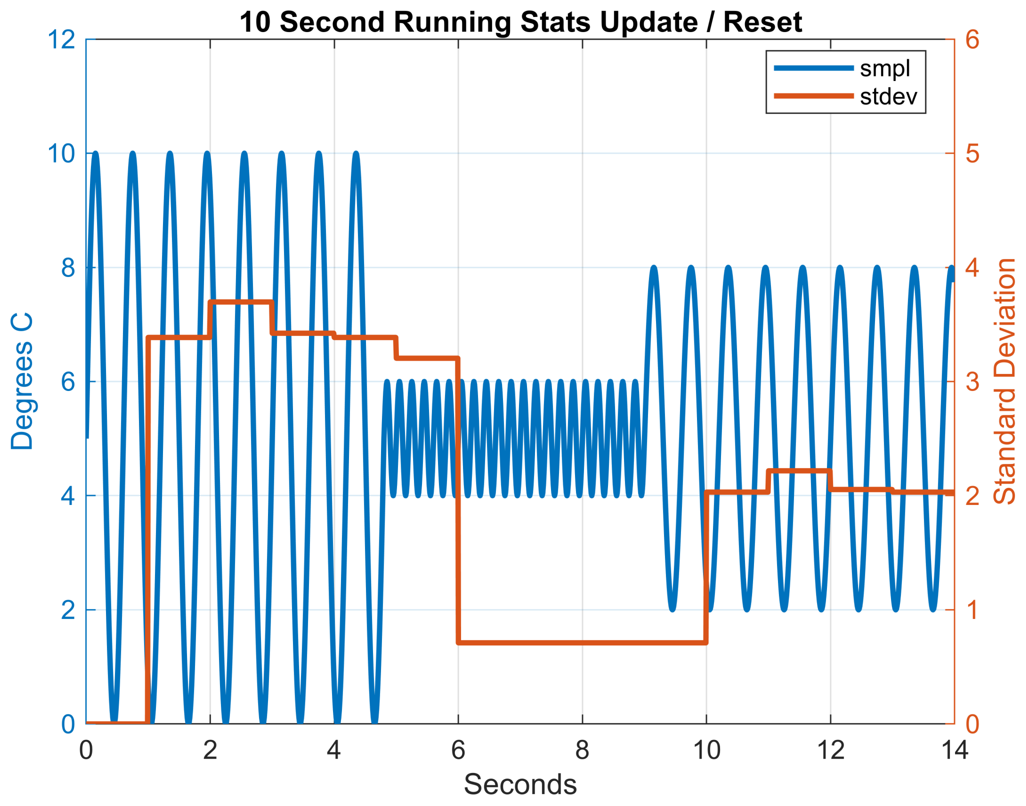
For the application of a breathing monitor, we are not interested in signal energy as measured by the standard deviation from the beginning of time. Rather, we are interested in the recent standard deviation. Modify the code to report and hold the running variance every ten seconds (i.e., 100 samples). Reset the variance and commence a new calculation after each report while holding the last value. See the stats update example below for varying amplitude and frequency data. Replicate this chart with your modified code, but reverse the order of modeled amplitude levels. Include a snip of your code modifications in your report.

PLOT P15 - Include a plot of your data in your report along with a code snippet in an included appendix.



```
%
%  Capture the output of the Arduino after modifying the code to implement
%  the resettable statistics.  Plot your results below
%
load("data/section3Data513.mat")
seconds = outData(:,1);
smpl = outData(:,2);
stdev = outData(:,3);

figure;
yyaxis left
plot(seconds/100, smpl, 'LineWidth',2);
xlabel('Seconds'); ylabel('Degrees C');
ylim([0 12]);

yyaxis right
plot(seconds/100, stdev, 'LineWidth',2);
ylabel('Standard Deviation');
ylim([0 6]);

title("10 Second Running Stats Update / Reset");
legend({'smpl','stdev'}, 'Location','best');
```

```
grid on;
```



Lab Write Up

For your lab write up include the plots as described in the procedure. Write your report as a story that you are telling someone. Discuss what you did and the observations that you made. Write the answers to the questions from the procedure within the narrative. Don't just repeat the question, write it in your own words along with your answer to the question. Be sure to use the IEEE format from the Word document. Don't change any of the fonts, margins, spacings etc..

Grading Rubric

Lab Section	Result Description	Points Available	Points Achieved
Abstract	Abstract and Introduction to the labe explaining what is your objective in doing the lab and outline how it is going to be done.	10	
1	Include Plots P1 to P8. Explain what you did in this section. Write narrative answers to questions 1 and 2. What is the impact of changing the filter length? What is the transision BW of the filter. Include any other observations based on your data.	20	
2	Include Plots P9 - P14 Include your tables of attenuation values versus frequency for the BPF and BSF Explain what you did in the procedure. Explain how you made a BPF and a BSF out of individual LPF and HPF filters. Include any other observations based on your data.	20	
3	Include Plot P15 of your resettable Standard Deviation calcultion. Include a snip of your code modificatioins. Describe how you modified the code to perform the reset and hold.	20	
Conclusion	In your own words summarize what you did in the lab and what you learned. Make it a part of the story that you are telling in the rest of the lab.	20	
Report Formatting	Correct use of the IEEE format. Titles and labels on all graphs. Appropriate descriptive captions.	10	