OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG EIT FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Laboratory

Digital Information Processing

Report

Names	Harish Kongari [Mr. Nr. 214586] Harikrishna Polavarapu [Mr. Nr. 218060]
Experiment	Filter I
Date	07/06/2018

Contents

1. Content of the Experiment

In this experiment, you will get to know what filters are and how they can be applied to signals. You will also implement some filters in Matlab.

2. Preparatory Exercises

Exercise 1.1: Filtering

What are filters? What different kinds are there? What are the differences? Give some examples and explain how they work. What are they used for in signal processing?

Answer:

Filters are the devices or processes which carries signals by removing unwanted components or features from signals.

In signal processing, filter is a process that perform filtering of signal that usually removes the unwanted components from the signal. Most often, the term filtering refers to removing of some frequencies.

As defined above that filter performs operations on signal frequency components. There are different kinds of filters based on the frequency response of the filters. They are described by their frequency bands that the filter allows to pass (Passband) and by rejection (Stopband) [1]:
Low-pass filter – Only low frequencies are passed, whereas high frequencies are attenuated.
High-pass filter – Only high frequencies are passed, whereas low frequencies are attenuated.
Band-pass filter – Only allows frequencies that are in the frequency band to pass.
Band-stop filter - Only frequencies that are in the frequency band are attenuated.
Notch filter – Only rejects a narrow (specific) frequency band and rest of the frequencies are allowed.
Comb filter – Consists only of a series of regularly spaced notches, giving the appearence of a comb.
All-pass filter – All frequency components are passed, but changes the phase relationship among various frequencies.

There are also different bases of classification of filters and these overlap in many different ways, there is no simple hierarchial categorization. So filters are also of Linear or Non-linear, Time varient or Time invarient, Causal or Non-causal, Analog or Digital, Discrete-time or Continuos-time, Active and passive, Infinite impulse response filter (IIR filter) or Finite impulse response filter (FIR filter).

Filters may be specified by family and bandform. A filter's family is specified by the approximating polynomial used and each lead to certain characteristics of the transfer function of the filter. Some common filter families and their particular characteristics are:

Butterworth filter – no gain ripple in pass band and stop band, slow cutoff .
Chebyshev filter (Type I) – no gain ripple in stop band, moderate cutoff .
Chebyshev filter (Type II) – no gain ripple in pass band, moderate cutoff.
Bessel filter – no group delay ripple, no gain ripple in both bands, slow gain cutoff.
Elliptic filter – gain ripple in pass and stop band, fast cutoff .
Optimum "L" filter.
Gaussian filter – no ripple in response to step function.
Hourglass filter.
Raised-cosine filter.

Examples: (By using Butterworth filter)
Let a signal as shown in *Figure 1.1.1* be our Original randomly generated signal [2].

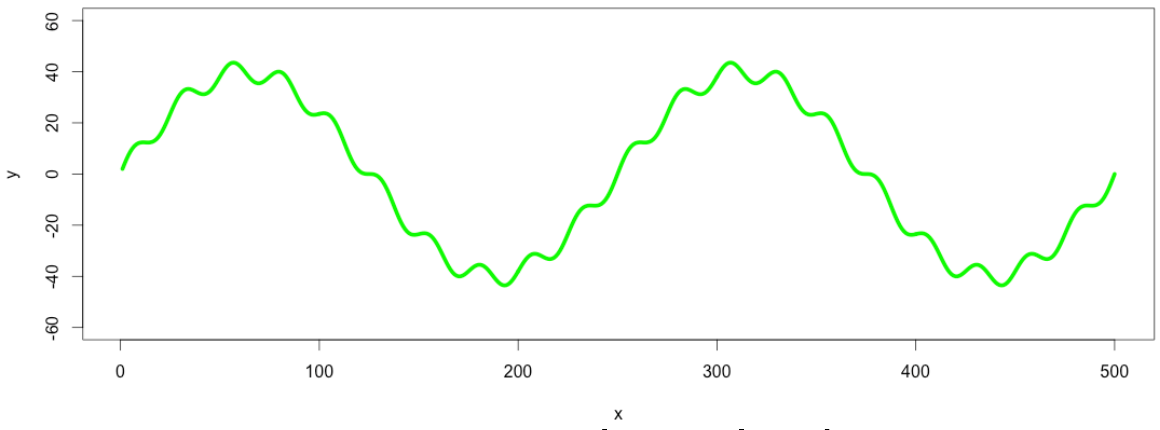


Figure 1.1.1: Original generated signal.

When the signal is processed bu using the Low-pass filter operation. In the figure below we can see the red line representes the filter response corresponding to Low-pass filteration. The signal is smoothed by removing high frequency components.

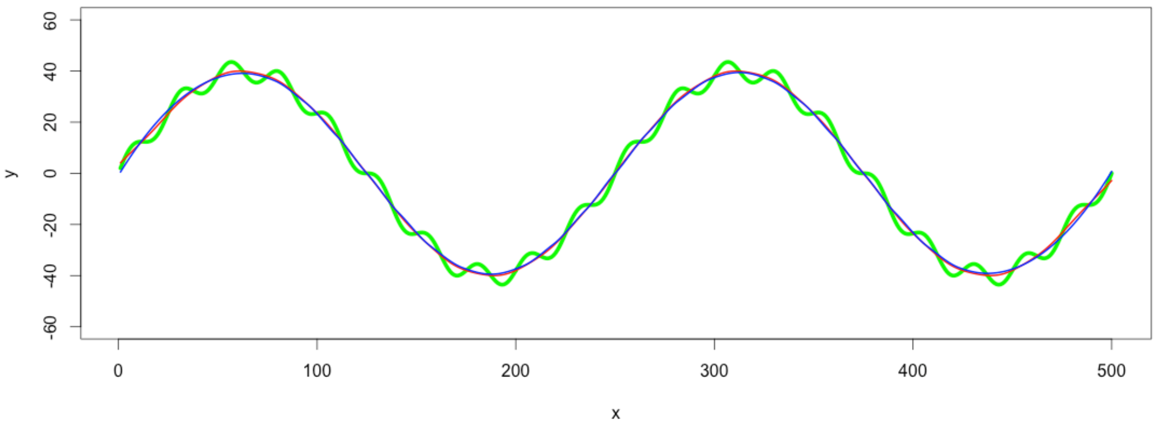


Figure 1.1.2: Original signal with Low-pass output (Red).

Figure 1.1.3 represents the filter response of a High-pass filter along with the previous results.We can observe that the outputis simply the substraction of Low-pass filtered values from the original signal.

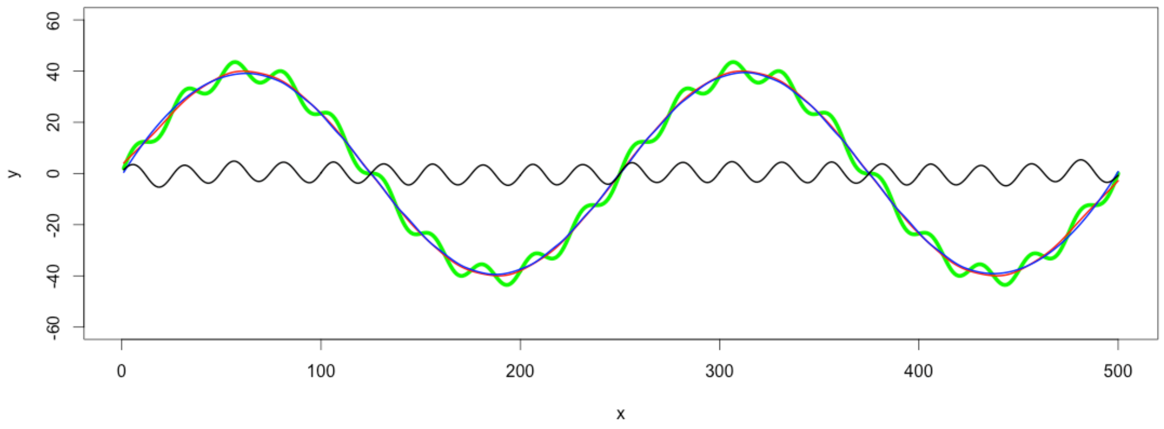


Figure 1.1.3: Original signal with Low-pass output (Red) and High-pass output (Black).

Exercise 1.2: Low-pass Filter

Explain how a low-pass filter works using an appropriate example.

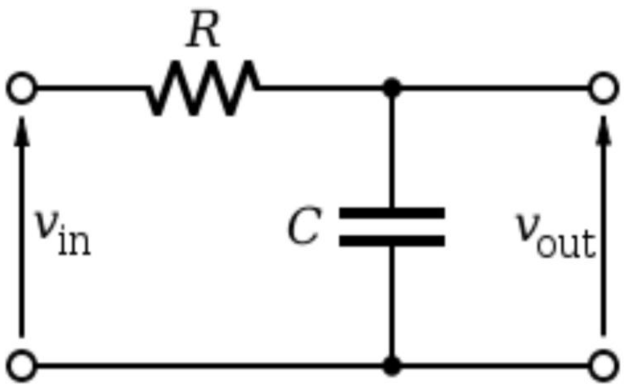
Answer:

A Low-pass filter [3] is a filter that allows signals below a cutoff frequency (known as the passband) and attenuates signals above the cutoff frequency (known as the stopband). By removing some frequencies, the filter creates a smoothing effect. That is, the filter produces slow changes in output values to make it easier to see trends and boost the overall signal-to-noise ratio with minimal signal degradation.

Low-pass filters, especially moving average filters or Savitzky-Golay filters, are often used to clean up signals, remove noise, perform data averaging, design decimators and interpolators, and discover important patterns.

Other common design methods for low-pass FIR-based filters include Kaiser Window, least squares, and equiripple. Design methods for IIR-based filters include Butterworth, Chebyshev (Type-I and Type-II) and elliptic.

For Example: RC Passive low-pass Filter



Filters can be divided into two distinct types: active filters and passive filters. Active filters contain amplifying devices to increase signal strength while passive do not contain amplifying devices to strengthen the signal. As there are two passive components within a passive filter design the output signal has smaller amplitude than its corresponding input signal, therefore passive RC filters attenuate the signal and have a gain of less than one, (unity).

Exercise 1.3: High-pass Filter

Explain how a high-pass filter works using an appropriate example.

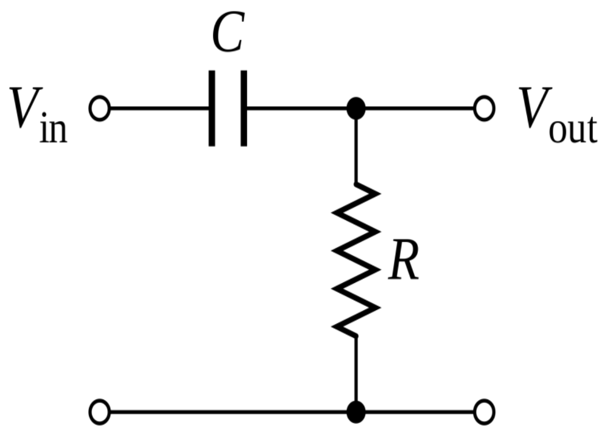
Answer:

A High-pass filter [4] (also known as a bass-cut filter) attenuates signals below a cutoff frequency (the stopband) and allows signals above the cutoff frequency (the passband). The output of this filter is directly proportional to rate of change of the input signal.

High-pass filters are often used to clean up low-frequency noise, remove humming sounds in audio signals, redirect higher frequency signals to appropriate speakers in sound systems, and remove low-frequency trends from time series data thereby highlighting the high-frequency trends.

Common design methods for high-pass FIR-based filters include Kaiser Window, least squares, and equiripple. Design methods for IIR-based filters include Butterworth, Chebyshev (Type-I and Type-II) and elliptic.

For Example: RC high-pass Filter



In this circuit, the reactance of the capacitor is very high at low frequencies so the capacitor acts like an open circuit and blocks any input signals at Vin until the cut-off frequency point (fc) is reached. Above this cut-off frequency point the reactance of the capacitor has reduced sufficiently as to now act more like a short circuit allowing the entire input signal to pass directly to the output.

3. Experiments with results

Exercise 2.1: Moving average

a) **Given the periodic signal $y = \sin(x) + 0.2 \cdot \sin(50 \cdot x)$ implement a matlab function *mov_av.m*, which performs a moving average on the signal. A moving average of order *n* for data point *t* is defined as the sequence of means of the last *n* consecutive data points. Try out different values for *n* > 1.**

$$\mathbf{m}_{MA}^{(n)}(t) = \frac{1}{n}\sum_{i=0}^{n-1} x(t-i)$$

What type of filter does the moving average implement? Describe the influence of the order *n*.

b) **Further, implement a weighted moving average, which decreases the influence the further the data point lies in history.**

$$\mathbf{m}_{LWMA}^{(n)}(t) = \frac{2}{n(n+1)}\sum_{i=1}^{n} i \cdot x(t-n+i)$$

Try out the same values for *n* as above. Describe the difference.

c) **A widely used smoothing method is the exponential smoothing. Again the influence of data point decreases the further they lie in history.**

$$\mathbf{m}_{EMA}^{(n)}(t) = \alpha \cdot x(t) + (1-\alpha) \cdot \mathbf{m}_{EMA}^{(n)}(t-1)$$

Try out different values for α. Compare to a) and b).

For a), b) and c) provide appropriate plots which show the influence of the filter.

a) Answer:

The Moving Average Filter (MA Filter) is a simple Low-pass FIR (Finite Impulse Response) filter. Which is commonly used for smoothing array of sampled signal. The MA filter involves in following functions [5]:

1. It takes 'M' input samples and computes the average of those 'M' samples from the average it produces a single output sampe.
2. When dealing with large amount of input samples which takes high computational efforts, the filter introduces a definite amount of delay.

The MA filter is the most common filter in DSP, mainly because it is the easiest digital filter to understand and use. In spite of its simplicity, the MA filter is optimal for a common task like smoothing (reducing random noise while retaining a sharp step response). This makes it the premier filter for time domain encoded signals.

However, the MA filter is the worst filter for frequency domain encoded signals, with little ability to separate one band of frequencies from another. Relatives of the MA filter include the Gaussian, Blackman, and multiplepass moving average. These have slightly better performance in the frequency domain, at the expense of increased computation time.

Matlab Code:

```
function mov_av(n)                    % Defining function with input variable 'n'
x= 0:1/1000:2*pi;                     % Defining time vector
y= sin(x)+0.2*sin(50*x);              % Given input signal
f = zeros(1,numel(y)-n+1);            % Preallocation for improving performance
for i=1: numel(y)-n+1                 % Defining the number of iterations of for loop
f(i)=sum(y(i:n+i-1))./n;              % Calculating moving average
end                                   % Ends the for loop
figure;                               % Opens a figure window
subplot(2,1,1);                       % Creating subplot
plot(x,y);                            % Plot corresponding to input signal
xlabel('time')                        % Labels the X-axis
ylabel('Amplitude')                   % Labels the Y-axis
title('Original signal');             % Adds the title to the figure
subplot(2,1,2);                       % Creating subplot
plot(f);                              % Plot corresponding output signal
xlabel('time')                        % Labels the X-axis
ylabel('Amplitude')                   % Labels the Y-axis
title(['Moving Average for order n = ',num2str(n)]);        % Adds the title to the figure
end                                   % Ends the function
```

In the above Matlab code, We used the the periodic signal '$y = \sin(x) + 0.2 \cdot \sin(50 \cdot x)$' to implement a matlab function '***mov_av.m***', that performs a moving average on the signal. A moving average of order 'n' for the data point 't' is defined as the sequence of means of the last 'n' consecutive data/sample points. Here we tried for different values of 'n' ranging from 2 to 1000 (n>1). The outputs corresponding to the code with different values of 'n' (n=2,20,100,200,400,600,800,1000) are as shown from *Figure 2.1.1* to *Figure 2.1.8* below.

Calling the function in command window, mov_av(2);

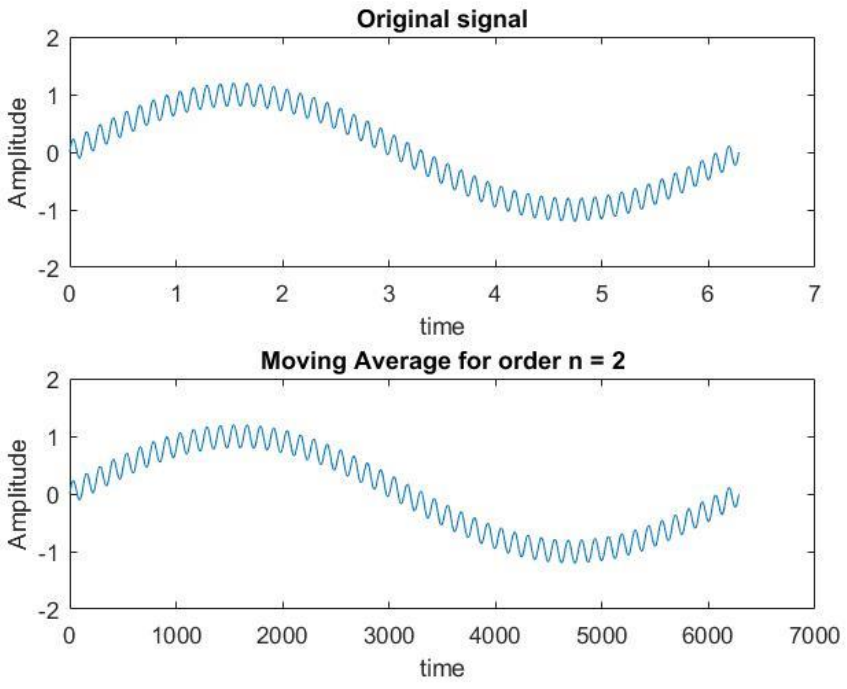


Figure 2.1.1: MA of signal of order n=2.

Calling the function in command window, mov_av(20);

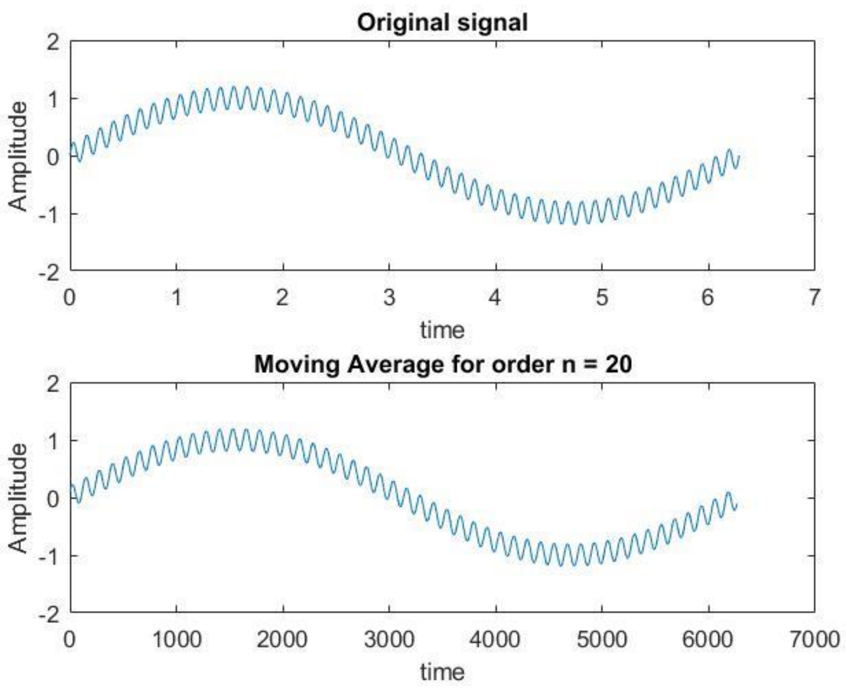


Figure 2.1.2: MA of signal order n=20.

Calling the function in command window, mov_av(100);

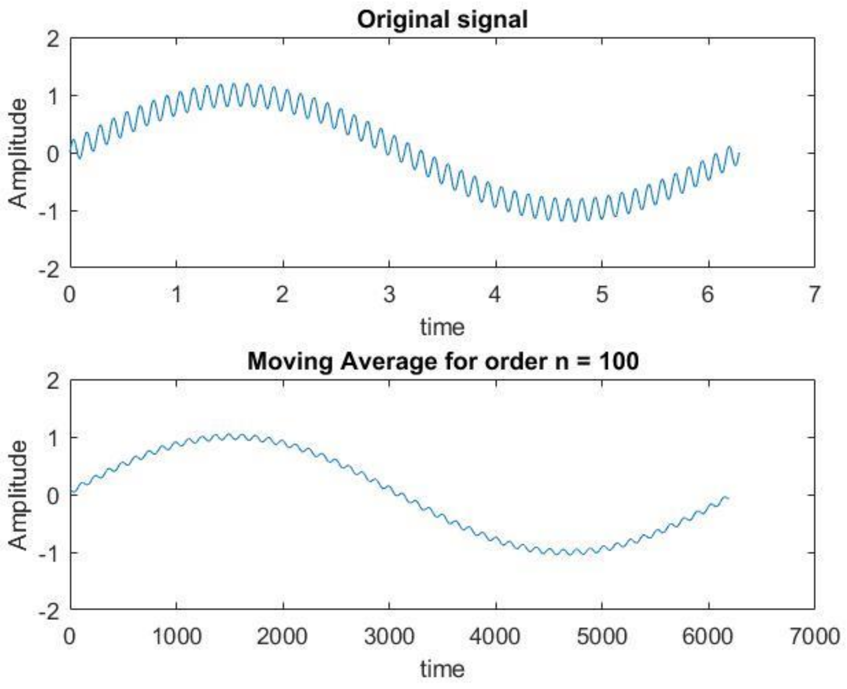


Figure 2.1.3: MA of signal of order n=100.

Calling the function in command window, mov_av(200);

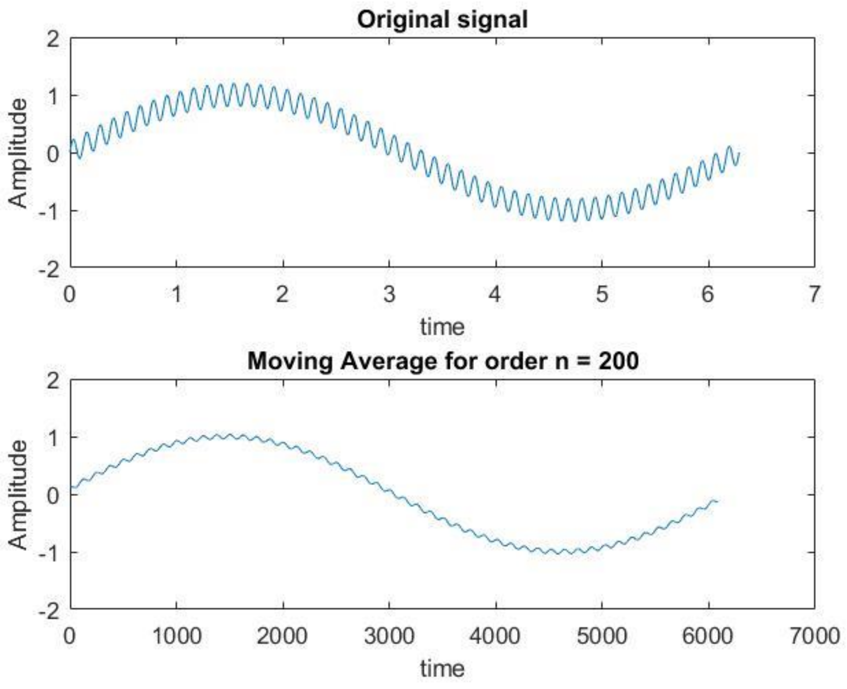


Figure 2.1.4: MA of signal of order n=200.

Calling the function in command window, mov_av(400);

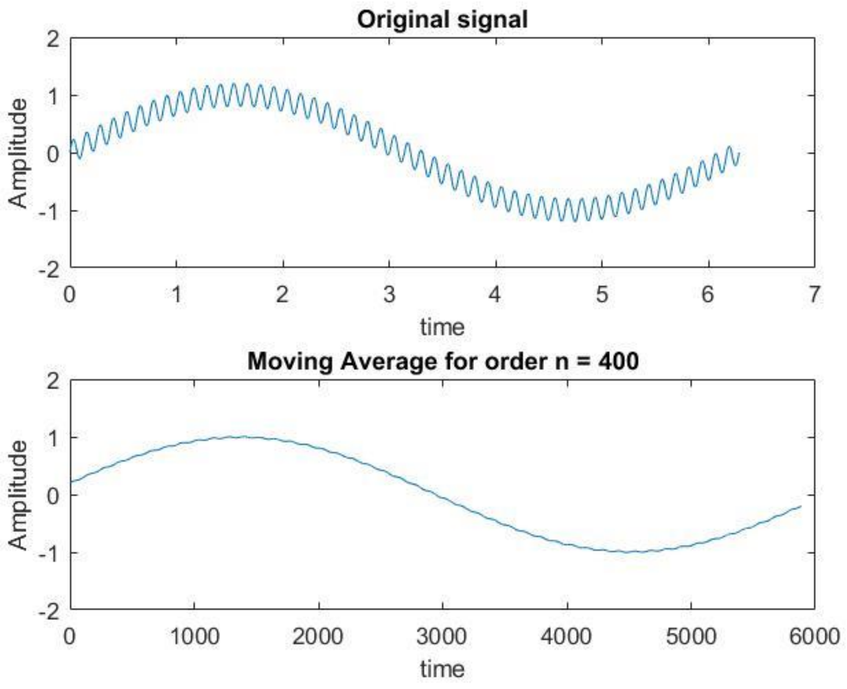


Figure 2.1.5: MA of signal of order n=400.

Calling the function in command window, mov_av(600);

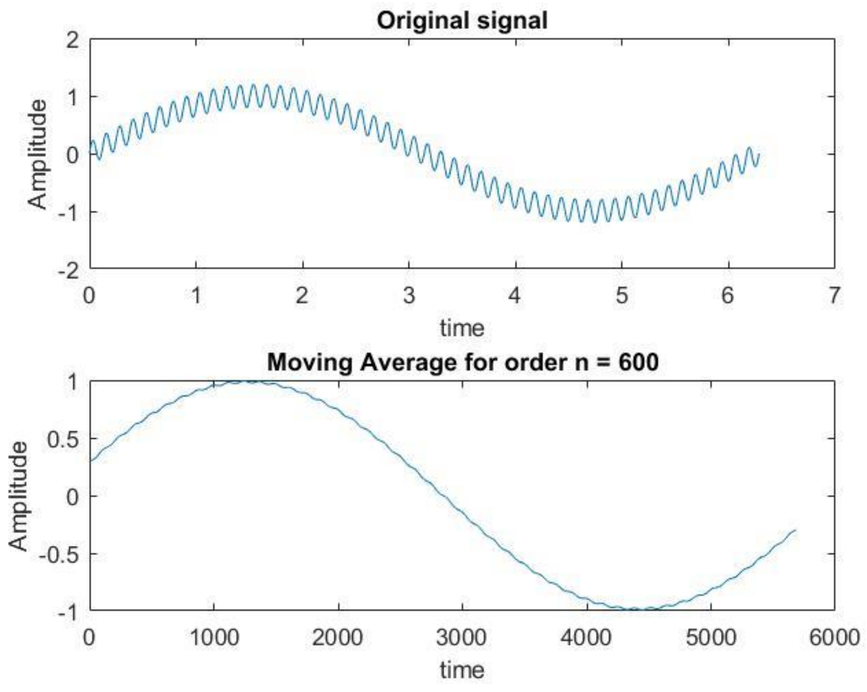


Figure 2.1.6: MA of signal of order n=600.

Calling the function in command window, mov_av(800);

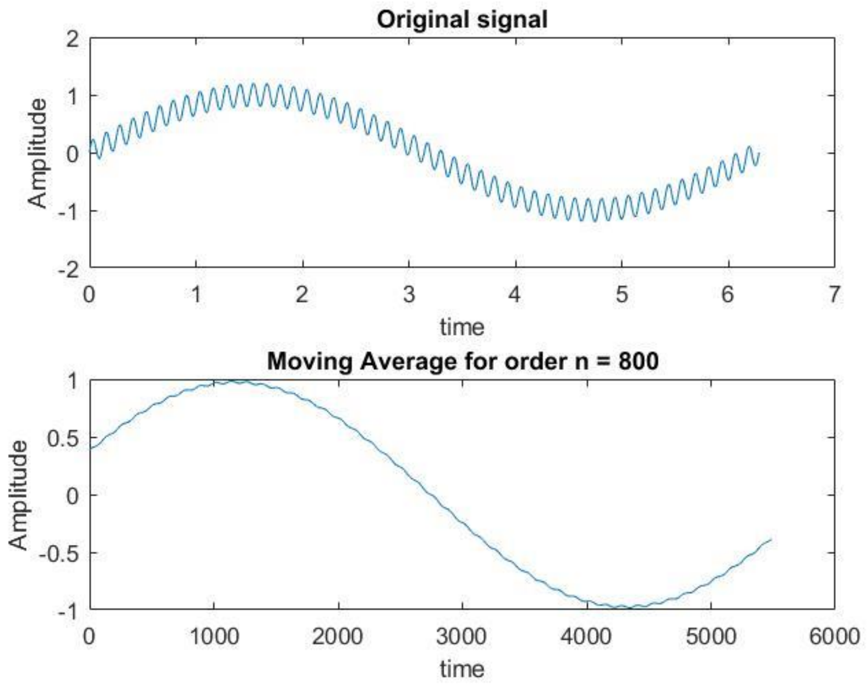


Figure 2.1.7: MA of signal of order n=800.

Calling the function in command window, mov_av(1000);

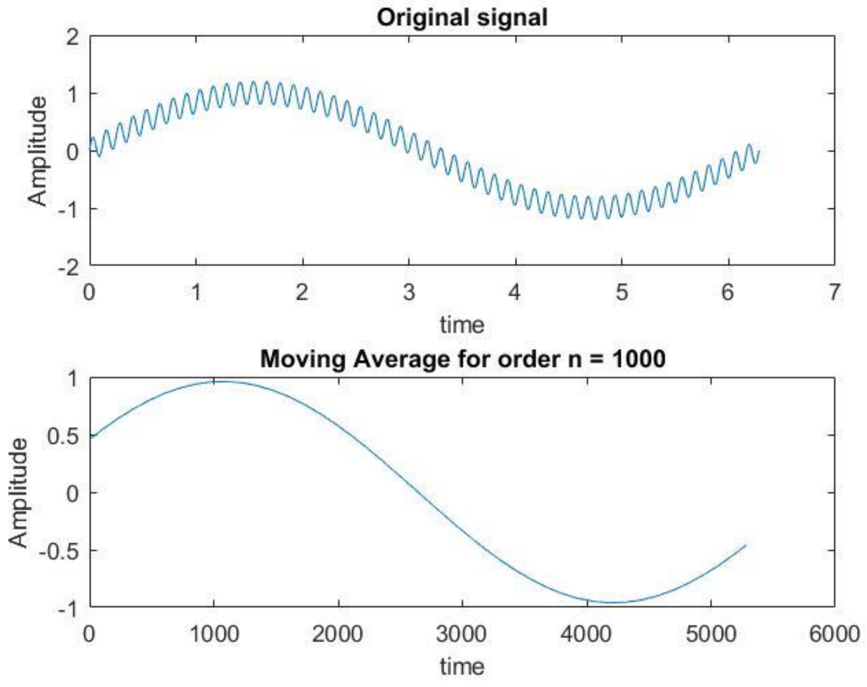


Figure 2.1.8: MA of signal of order n=1000.

From the above figures we can observe that the signal ripples are more MA of n=2 and these ripples gradually decreased (almost no ripples) by MA of n=1000. As we mentioned in introduction of this task, MA filter is a type of FIR filter. Since the MA filter is best known for smoothing of signal by reduction of random noise. There is significant inflence of the order 'n' that improves the output with the increase in order 'n'. i.e., The amount of noise reduction is equal to the square-root of the number of points in the average. For example, a 100 point moving average filter reduces the noise by a factor of 10.

b) Answer:

A Weighted Moving average is an average that that has multiplying factor to give different weights to the data at different positions in the sample window. Instead of averaging all the pixel values in the window, give the closer-by pixels higher weighting, and far-away pixels lower weighting.

Matlab Code:
```
function weighted_average(n)   % Defining function with input variable 'n'
x= 0:1/1000:2*pi;              % Defining time vector
y= sin(x)+0.2*sin(50*x);       % Given input signal
f = zeros(1,numel(y)-n+1);     % Preallocation for improving performance
for i=1: numel(y)-n+1          % Defining the number of iterations of for loop
f(i)=sum((1:n).*y(i:n+i-1))*(2./n*(n+1));  % Weighted moving average for n data points
end                            % Ends the for loop
figure;                        % Opens a figure window
subplot(2,1,1);                % Creating subplot
plot(x,y);                     % Plot corresponding to input signal
xlabel('time')                 % Labels the X-axis
```

```
ylabel('Amplitude')                % Labels the Y-axis
title('Original signal');          % Adds the title to the figure
subplot(2,1,2);                    % Creating subplot
plot(f);                           % Plot corresponding filtered signal
xlabel('time')                     % Labels the X-axis
ylabel('Amplitude')                % Labels the Y-axis
title(['Weighted moving Average for order n = ',num2str(n)]);  % Adds the title to the figure
end                                % Ends the function
```

In the above Matlab code, We used the the periodic signal '$y = \sin(x) + 0.2 \cdot \sin(50 \cdot x)$' to implement a weighted moving averages, which decreases the influence the further the data point lies in the history. Here we tried for different values of 'n' ranging from 2 to 1000 (n>1).

The outputs corresponding to the code with different values of 'n' (n=2,20,100,200,400,600,800,1000) are as shown from *Figure 2.1.9* to *Figure 2.1.16* below.

Calling the function in command window, weighted_average(2);

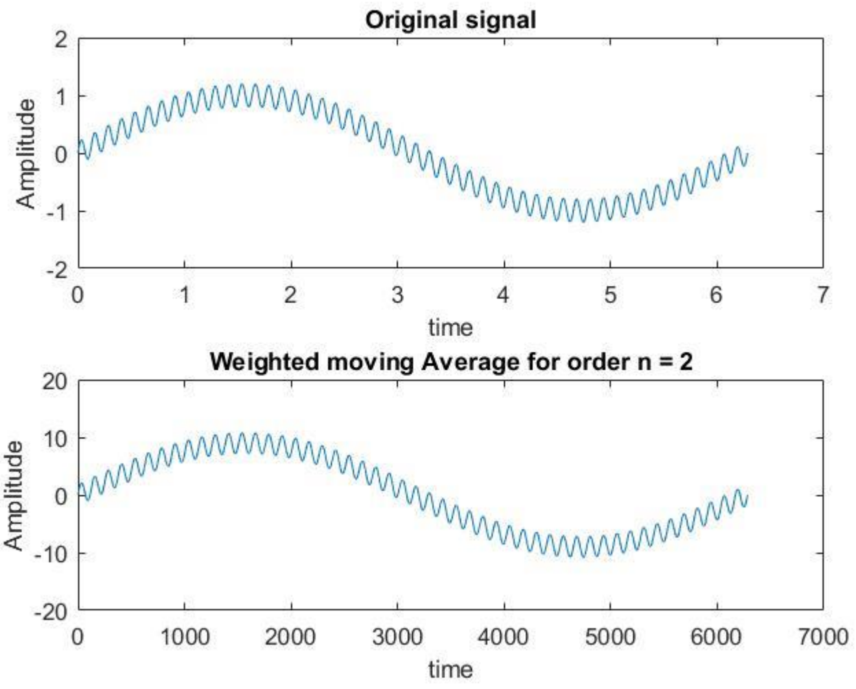


Figure 2.1.9: Weighted MA of signal of order n=2.

Calling the function in command window, weighted_average(20);

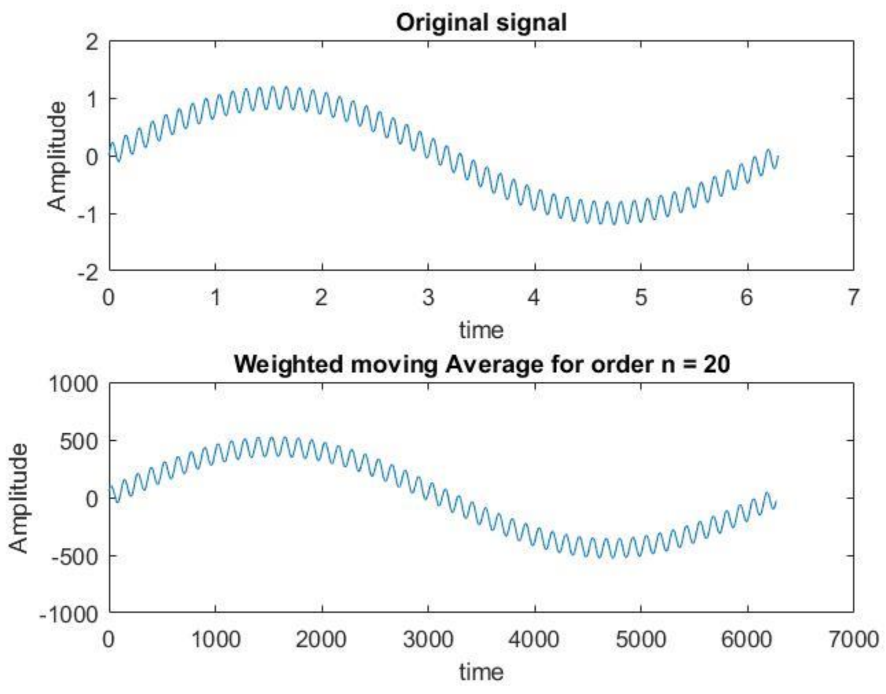


Figure 2.1.10: Weighted MA of signal of order n=20.

Calling the function in command window, weighted_average(100);

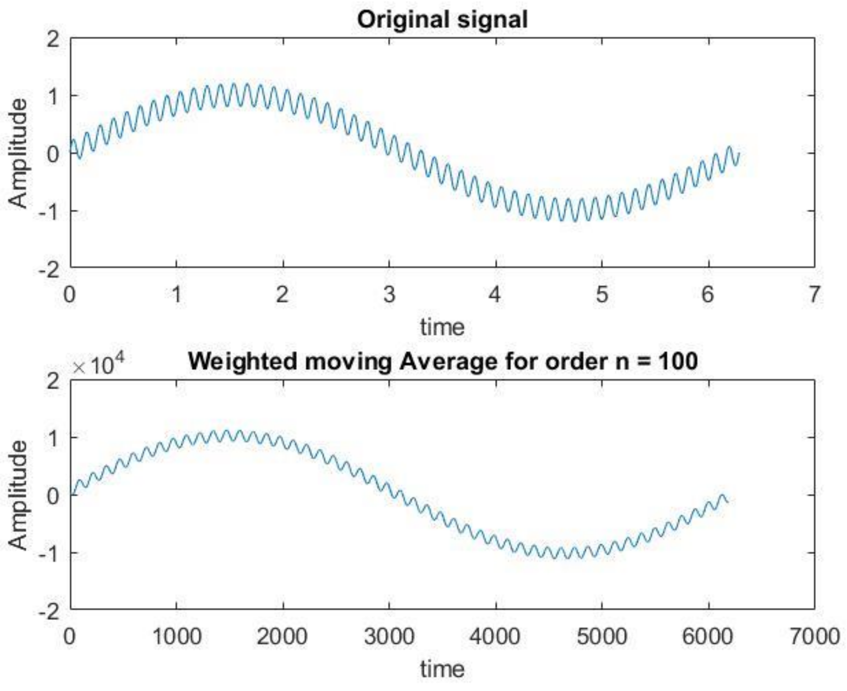


Figure 2.1.11: Weighted MA of signal of order n=100.

Calling the function in command window, weighted_average(200);

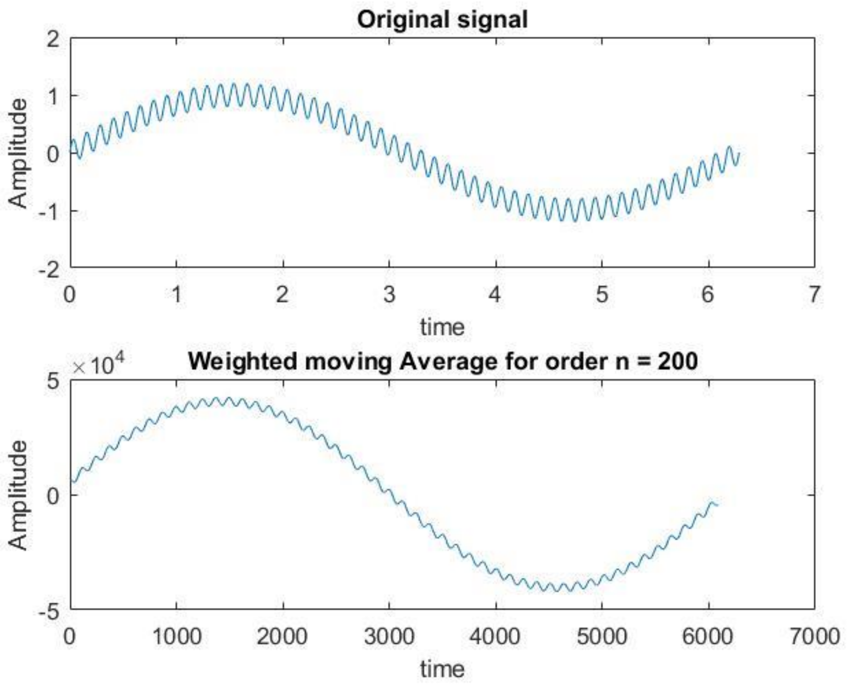


Figure 2.1.12: Weighted MA of signal of order n=200.

Calling the function in command window, weighted_average(400);

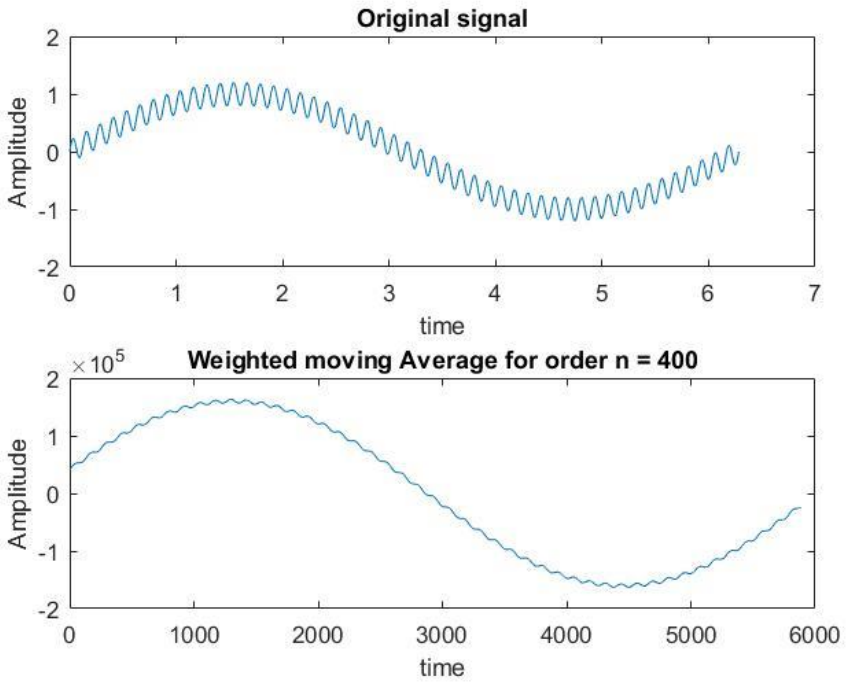


Figure 2.1.13: Weighted MA of signal of order n=400.

Calling the function in command window, weighted_average(600);

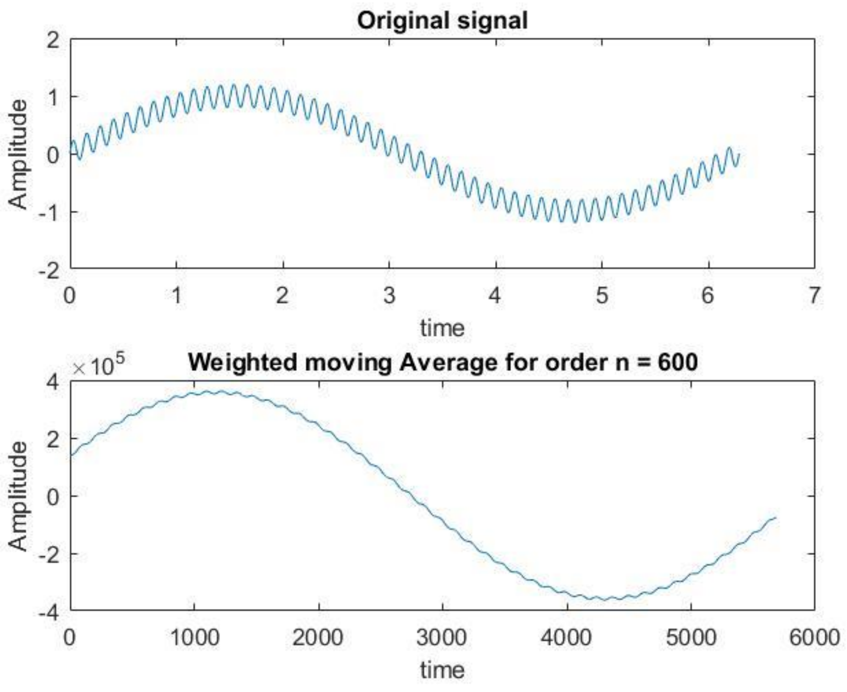


Figure 2.1.14: Weighted MA of signal of order n=600.

Calling the function in command window, weighted_average(800);

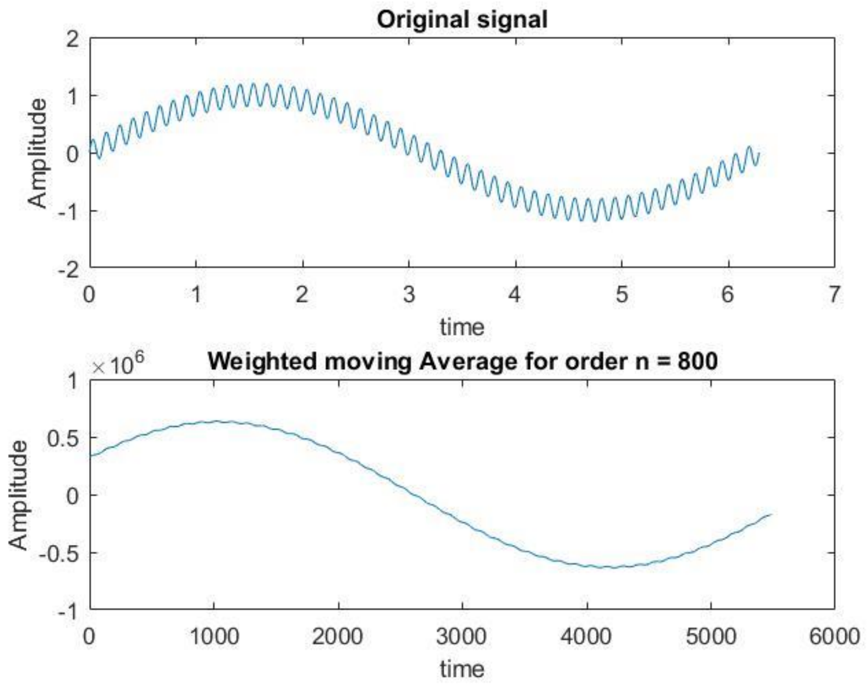


Figure 2.1.15: Weighted MA of signal of order n=800.

Calling the function in command window, weighted_average(1000);

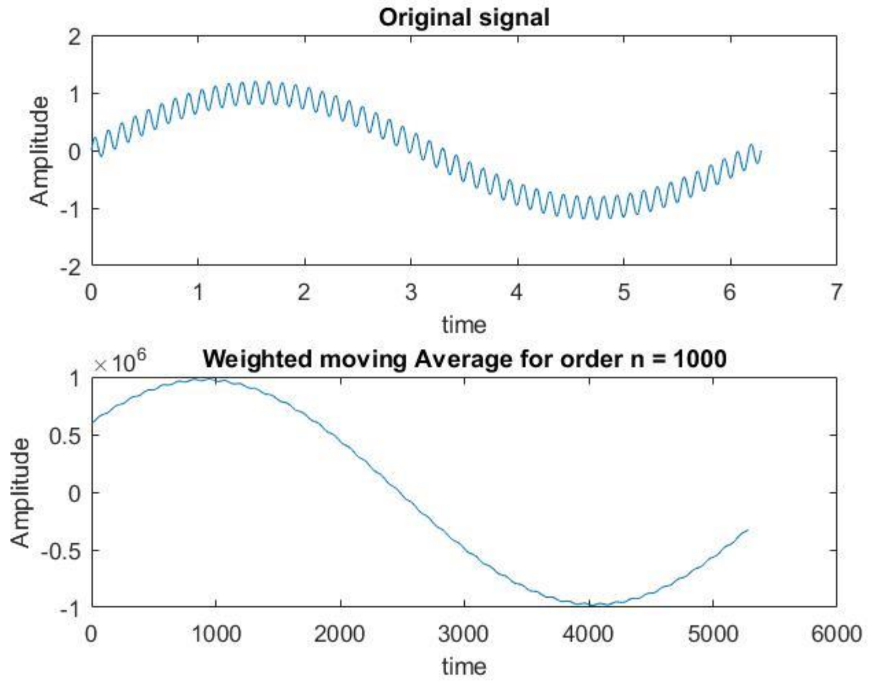


Figure 2.1.16: Weighted MA of signal of order n=1000.

From the above figures we can observe that the signal ripples are more weighted MA of n=2 and these ripples gradually decreased by weighted MA of n=1000. We can say for n=1000 the signal is smoother when compared to values of n=2,20,100,200,400,600,800.

c) Answer:

Exponential smoothing is one of many window functions commonly applied to smooth data in signal processing, acting as low-pass filters to remove high frequency noise.

In this task the exponential smoothing is given by:

$$\mathrm{m}_{EMA}^{(n)}(t) = \alpha \cdot x(t) + (1-\alpha)\cdot \mathrm{m}_{EMA}^{(n)}(t-1)$$

Where, the $\mathrm{m}_{EMA}^{(n)}(t)$ is the simple weighted average of the current observation $x(t)$ and the previous value of $\mathrm{m}_{EMA}^{(n)}(t)$ i.e., $\mathrm{m}_{EMA}^{(n)}(t-1)$. 'α' is called smoothing factor ($0<\alpha<1$). The smoothing factor 'α' close to one have less of a smoothing effect and give greater weight to recent changes in the data, while values of 'α' closer to zero have a greater smoothing effect and are less responsive to recent changes. There is no formally correct procedure for choosing 'α'.

Matlab Code:

```
function exponential_smoothening(alpha) % Defining function with input variable 'α'
x= 0:1/10:2*pi;                         % Defining time vector
y= sin(x)+0.2*sin(50*x);                % Given input signal
f = zeros(1,numel(y));                  % Preallocation for improving performance
f(1)=y(1);                              %  Assigning the value
for i=2:numel(y)                        % Defining the number of iterations of for loop
f(i)=  alpha*y(i) + (1-  alpha)*f(i-1);  % Exponential smoothing depending on factor 'α'
```

```
end                                          % Ends the for loop
figure;                                      % Opens a figure window
subplot(3,1,1);                              % Creating subplot
plot(x,y);                                   % Plot corresponding to input signal
xlabel('time')                               % Labels the X-axis
ylabel('Amplitude')                          % Labels the Y-axis
title('Original signal');                    % Adds the title to the figure
subplot(3,1,2);                              % Creating subplot
plot(x, f, 'r');                             % Plot corresponding output signal
xlabel('time')                               % Labels the X-axis
ylabel('Amplitude')                          % Labels the Y-axis
title(['Exponential smoothening for alpha = ',num2str(alpha)]);          % Adds the title
subplot(3,1,3)                               % Creating subplot
plot(x,y,x,f)                                % Plot corresponding to input and output signal
legend('Original signal','exponential smoothing')                        % Adds the legend
xlabel('time')                               % Labels the X-axis
ylabel('Amplitude')                          % Labels the Y-axis
title('Comparison of Original signal and exponential smoothed signal');  % Adds the title
end                                          % Ends the function
```

In the above Matlab code, We used the the periodic signal '$y = \sin(x) + 0.2 \cdot \sin(50 \cdot x)$' to implement a widely used smoothing method that is the exponential smoothing, In which the influence of the further the data point decreases the further they lie in the history. Here we tried for different values of smoothing constant 'α' (a number between 0 and 1) ranging from 0.2 to 1. The outputs corresponding to the code with different values of 'α' (α=0.2,0.4,0.5,0.6,0.8,1) are as shown from *Figure 2.1.17* to *Figure 2.1.22* below.

Calling the function in command window, exponential_smoothening(0.2);

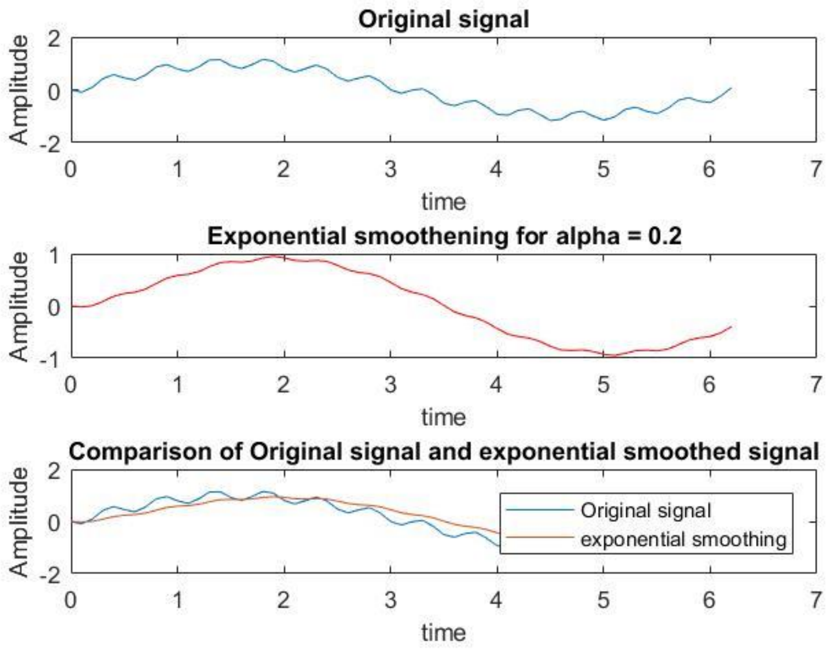


Figure 2.1.17: Exponential smoothing with α=0.2.

Calling the function in command window, exponential_smoothening(0.4);

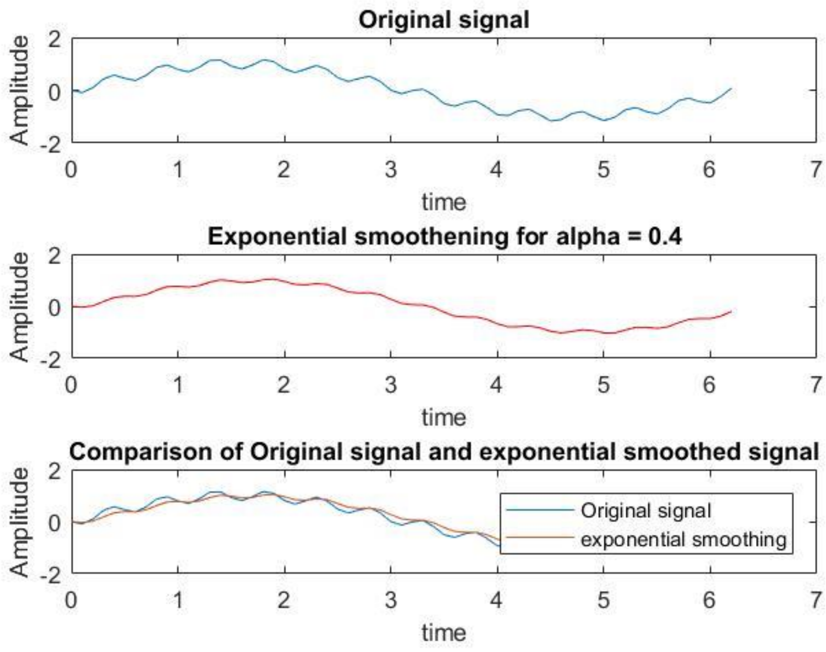


Figure 2.1.18: Exponential smoothing with α=0.4.

Calling the function in command window, exponential_smoothening(0.5);

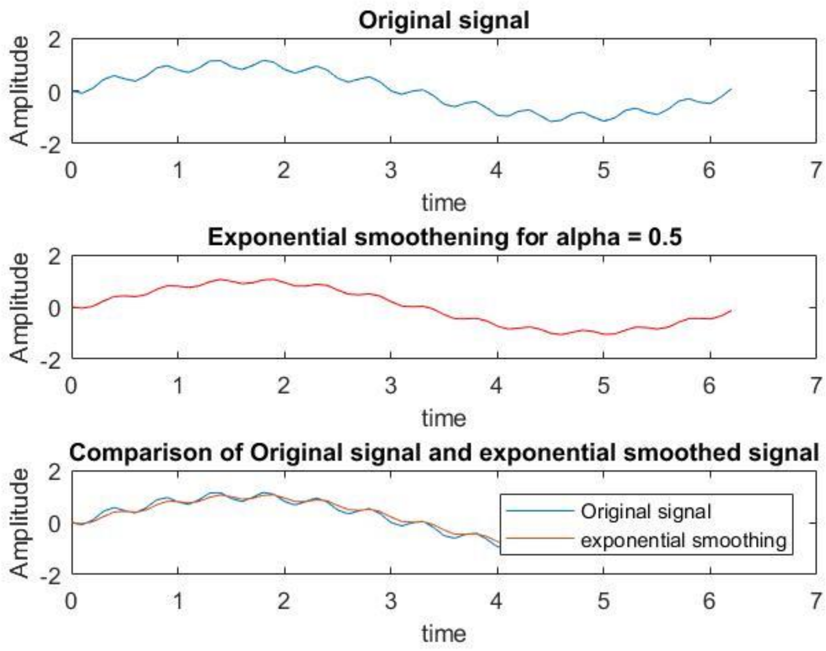


Figure 2.1.19: Exponential smoothing with α=0.5.

Calling the function in command window, exponential_smoothening(0.6);

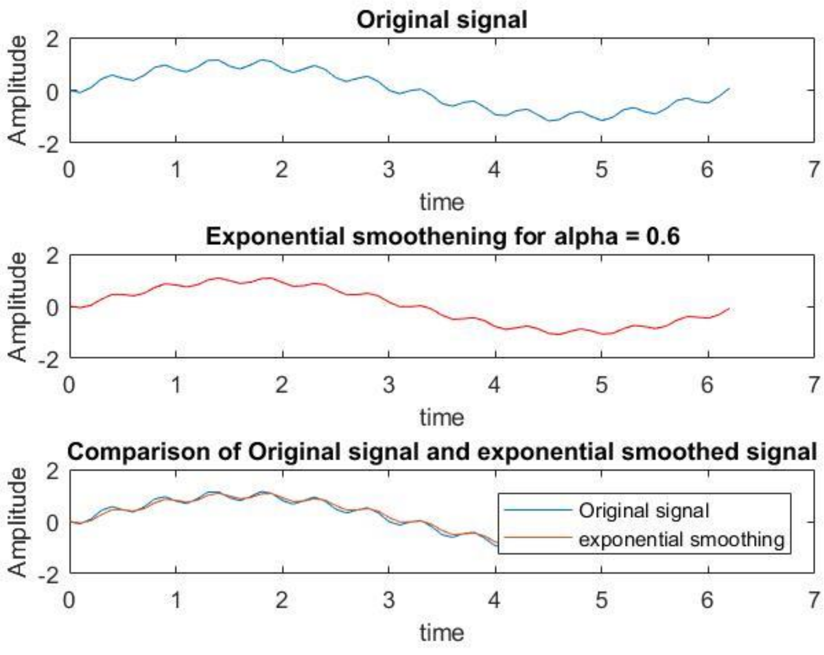


Figure 2.1.20: Exponential smoothing with α=0.6.

Calling the function in command window, exponential_smoothening(0.8);

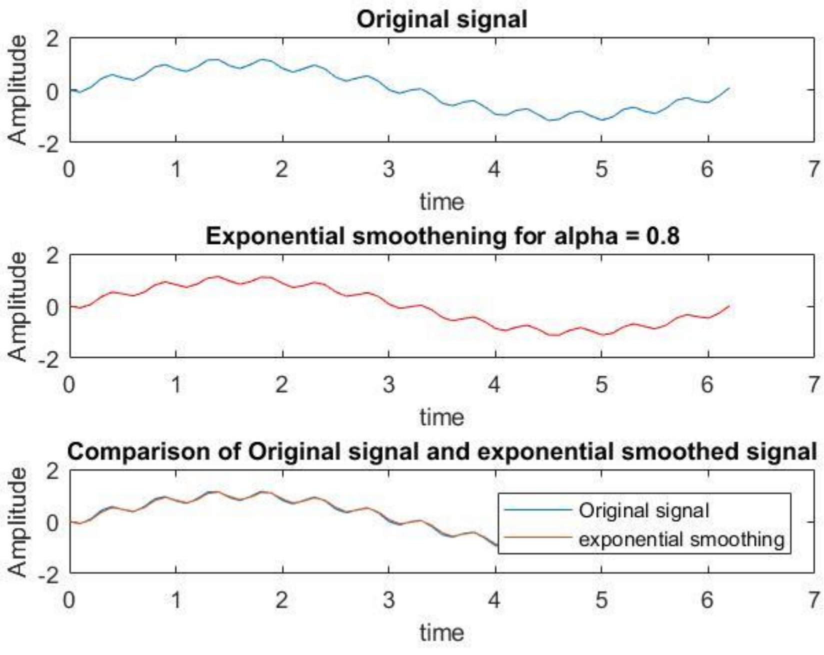


Figure 2.1.21: Exponential smoothing with α=0.8.

Calling the function in command window, exponential_smoothening(1);

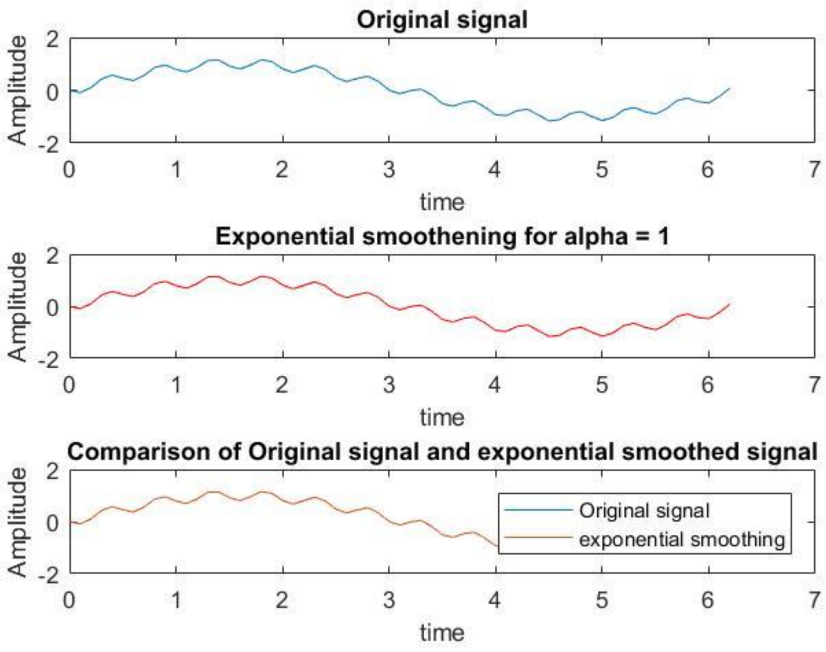


Figure 2.1.22: Exponential smoothing with α=1.

From the above *Figure 2.1.17* to *Figure 2.1.22* we can observe that for *α*= 0.2,0.4,0.5,0.6,0.8 the output of the exponential smoothing performance (Smoothing) decreses with the increase in the value 'α'. This type of filtering is called 'exponential', because the weighting factor of previous inputs decreases exponentially. So at α=1, the output is just equal to the input and no filtering takes place.

Exercise 2.2: Matlab Filter Butter

What is a Butterworth filter? Find out how to use the matlab command 'butter'. Implement a low-pass and a high-pass filter and apply them to the signal y.

Answer:

The Butterworth filter [6] is a type of signal processing filter designed to have as flat a frequency response as possible in the passband. It is also referred to as a maximally flat magnitude filter (i.e., has no ripples) in the passband and rolls off towards zero in the stopband. The Butterworth filter are fast and simple to use.Since this filter is frequency-based, the effect of filtering can be easily understood and predicted.

In Matlab in order to use Butterworth filter the following command can be used to design the filter "**[b,a] = butter(n,Wn,ftype)**". Here 'b', 'a' are the transfer function coefficients. The transfer function coefficients of the filter, returns as row vectors of the length n+1 for Low-pass and High-pass filters. 'n' is used to define the filter order, the filter order is specified as an integer scalar. 'Wn' corresponds to the cutoff frequency, specified as a scalar that designs a Low-pass or High-pass filter with cuffoff frequency. And ' ftype' is to specify the filter type. i.e., 'low' specifies a Low-pass filter with cutoff frequency 'Wn'. 'high' specifies a High-pass filter with cutoff frequency 'Wn'.

<u>Matlab Code</u>:

```
function butterworth_filter(n1,wn1)          % Defining function with input variables
fs = 1000;                                   % Defining sampling frequency
x = 0:1/fs:2*pi;                             % Defining the time interval
y = sin(x) + 0.2*sin(50*x);                  % Defining input signal
figure;                                      % Opens a figure window
plot(x,y)                                    % Plot corresponding to input signal
title('Original Signal')                     % Adds the title to the figure
xlabel('t')                                  % Labels the X-axis
ylabel('Amplitude')                          % Labels the Y-axis
[b,a] = butter(n1,wn1,'low');                % Defining a Low-pass butterworth filter
figure;                                      % Opens a figure window
freqz(b,a);                                  % Frequency response of the filter
title('Lowpass butterworth filter')          % Adds the title of the figure
Lowfilter=filter(b,a,y);                     % Filters the input data y with coeff. b and a
figure;                                      % Opens a figure window
plot(x,Lowfilter)                            % Plot corresponding Low-pass filter output
title('Lowpass filter Signal')               % Adds the title to the figure
xlabel('t')                                  % Labels the X-axis
ylabel('Amplitude')                          % Labels the Y-axis
figure;                                      % Opens a figure window
plot(x,y);                                   % Plot corresponding to input signal
hold on;                                     % Holds the figure to make further changes
plot(x, Lowfilter);                          % Plot corresponding Low-pass filter output
title('Original signal vs Lowpass butterworth filter')    % Adds the title to the figure
legend('Original signal', 'Lowpass Butterworth filter')  % Adds the legend to the figure
xlabel('t')                                  % Labels the X-axis
ylabel('Amplitude')                          % Labels the Y-axis
hold off;                                    % Turning OFF hold on function
[b1,a1] = butter(n1,wn1,'high');             % Defining a High-pass butterworth filter
figure;                                      % Opens a figure window
freqz(b1,a1)                                 % Frequency response of the filter
title('Highpass butterworth filter')         % Adds the title to the figure
Highfilter=filter(b1,a1,y);                  % Filters the input data y with coeff. b and a
figure;                                      % Opens a figure window
plot(x,Highfilter)                           % Plots corresponding High-pass filter output
title('Highpass filter Signal')              % Adds the title to the figure
xlabel('t')                                  % Labels the X-axis
ylabel('Amplitude')                          % Labels the Y-axis
figure;                                      % Opens a figure window
plot(x,y);                                   % Plot corresponding to input signal
hold on;                                     % Holds the figure to make further changes
plot(x, Highfilter);                         % Plot corresponding High-pass filter output
title('Original signal vs Highpass butterworth filter')    % Adds the title to the figure
legend('Original signal', 'Highpass Butterworth filter')  % Adds the legend to the figure
xlabel('t')                                  % Labels the X-axis
ylabel('Amplitude')                          % Labels the Y-axis
hold off;                                    % Turning OFF hold on function
end                                          % Ends the function
```

In the above MATLAB code, At first we defined sampling frequency and then the Cutoff frequency 'Wn' which must be 0.0 < Wn < 1.0. Later implemented Low-pass and High-pass butterworth filters, the frequency response and the filtered signals were obtained for the input signal 'y'. Here, We considered the sampling frequency 'fs' in the program to be 1000Hz and the cutoff frequency 'fc' to be 300 Hz. The normalised frequency can be calculated as wn = fc/(fs/2) and for this task it is found to be wn = 300/(1000/2) = 0.6. We can observe from the outputs that the filtered signal have as flat a frequency response as possible in the passband region of the original signal. And also they can be easily understood from the filtered responses. The outputs corresponding to the above code are as shown from *Figure 2.2.1* to *Figure 2.2.7* below.

Calling the function in command window, butterworth_filter(2, 300/(1000/2));

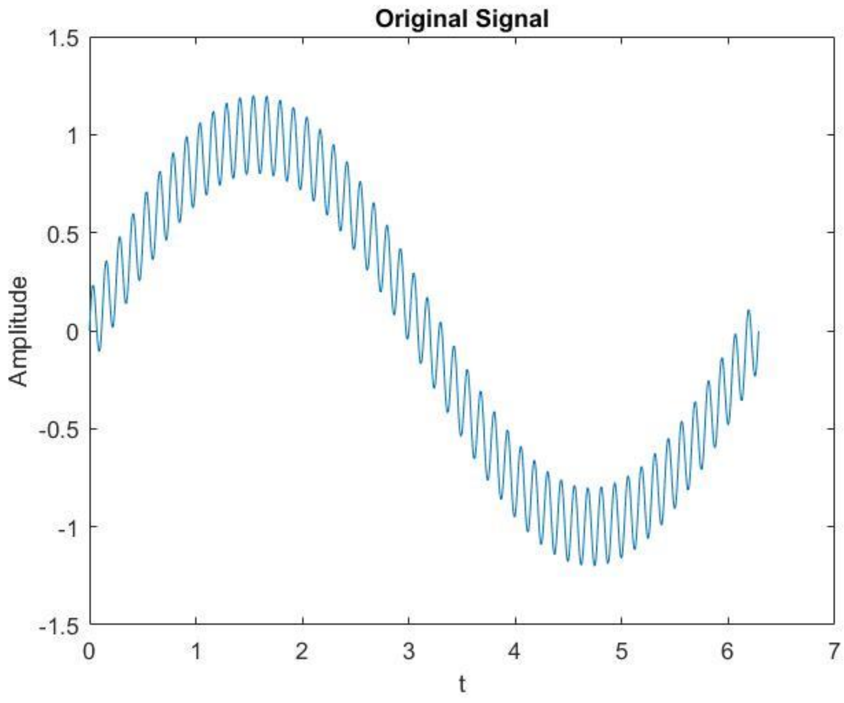


Figure 2.2.1: Original input signal.

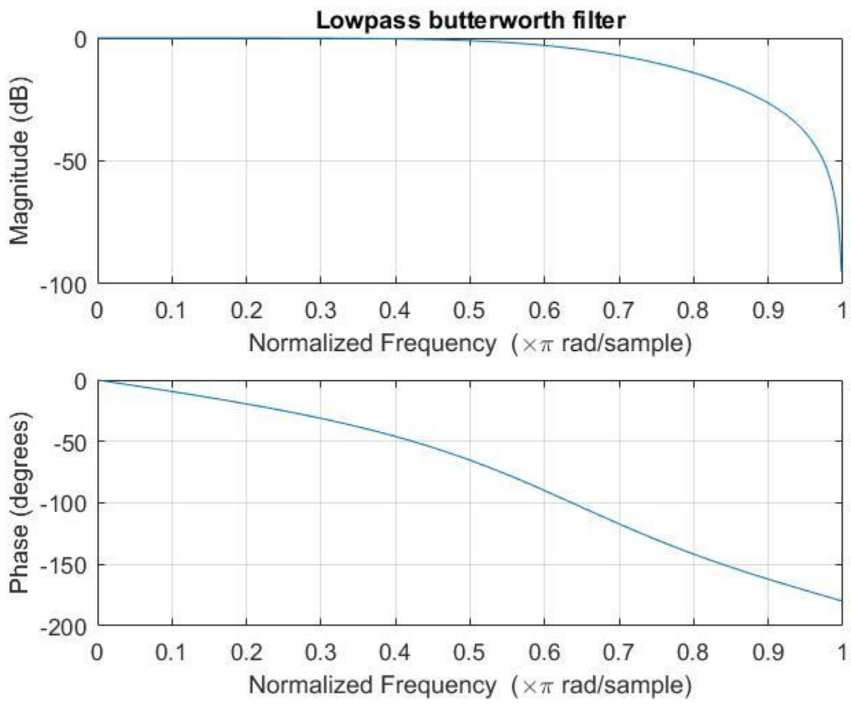


Figure 2.2.2: Low-pass frequency response of the signal.

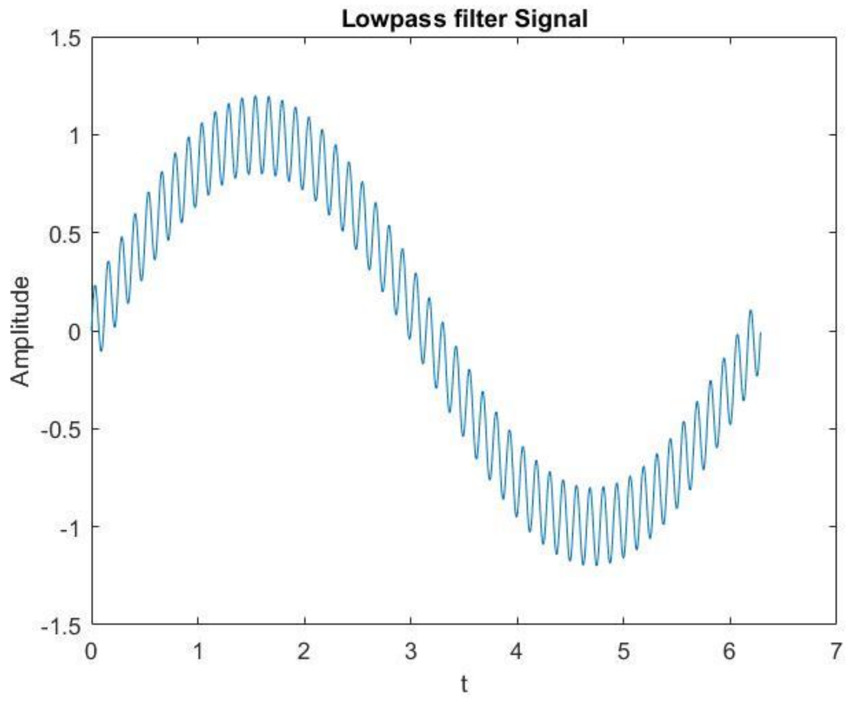


*Figure 2.2.3: Low-pass response of the signal by using '**butter**' command.*

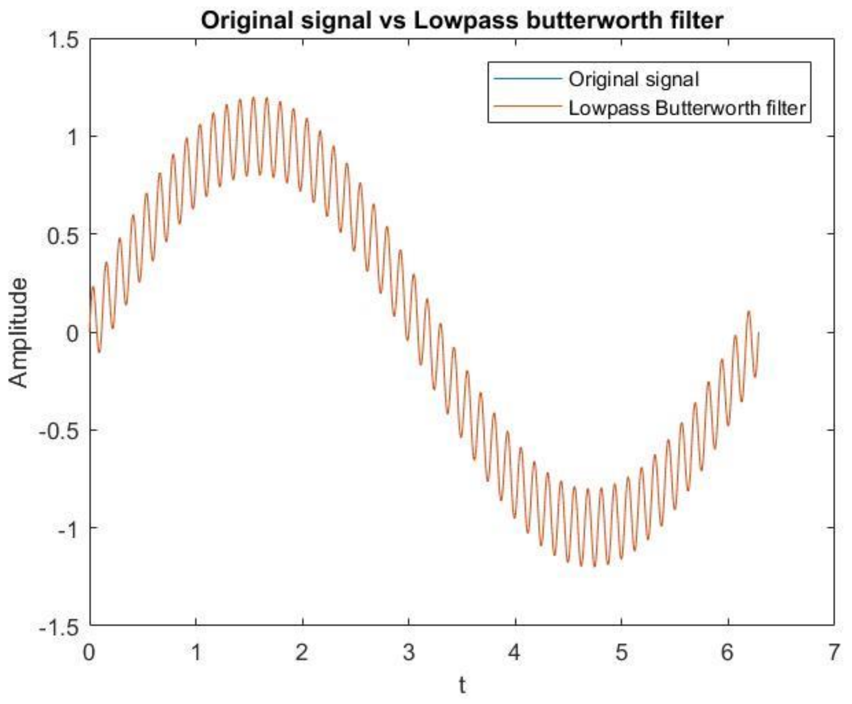


Figure 2.2.4: Comparision of Original and Low-pass butterworth filter signals.

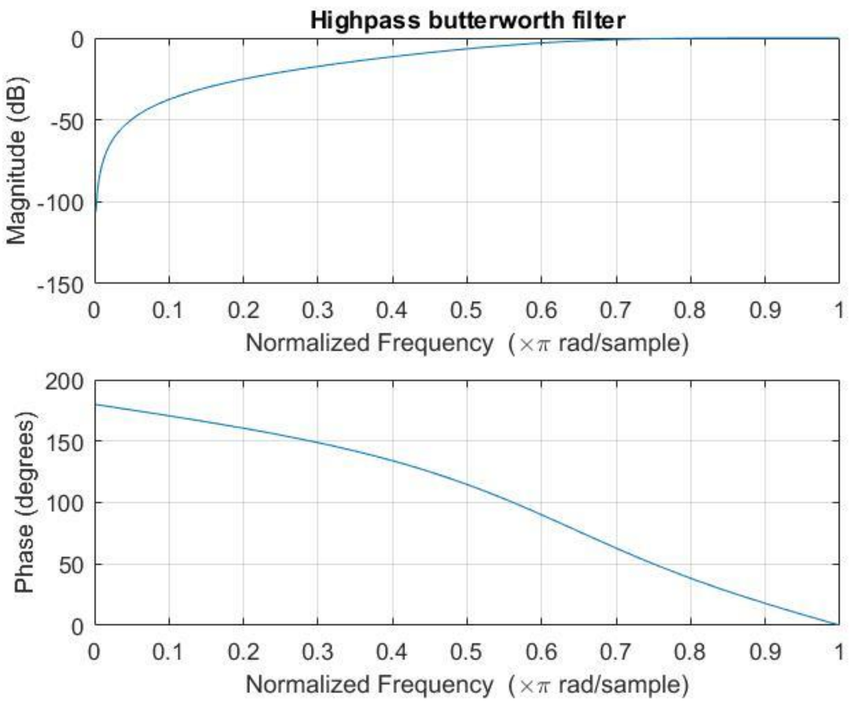


Figure 2.2.5: High-pass frequency response of the signal.

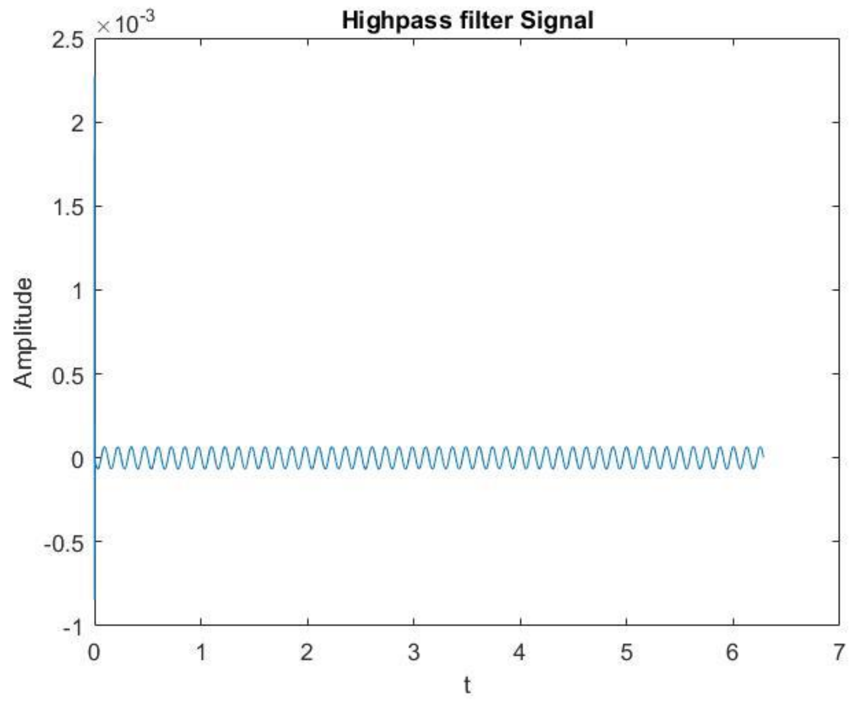


*Figure 2.2.6: High-pass response of the signal by using '**butter**' command.*

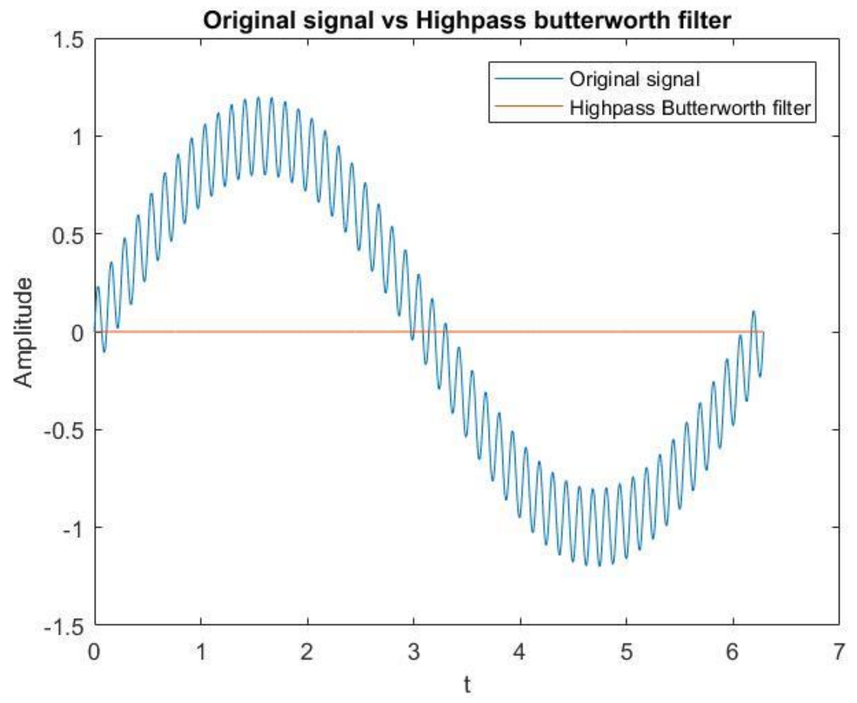


Figure 2.2.7: Comparision of Original and How-pass butterworth filter signals.

4. References

[1] Filter (signal processing) - https://en.wikipedia.org/wiki/Filter_(signal_processing).

[2] Filter (examples) - https://stackoverflow.com/questions/7105962/how-do-i-run-a-high-pass-or-low-pass-filter-on-data-points-in-r.

[3] Low-pass filter - https://de.mathworks.com/discovery/low-pass-filter.html.

[4] High-pass filter - https://de.mathworks.com/discovery/high-pass-filter.html.

[5] Moving Average Filters - http://www.analog.com/media/en/technical-documentation/dsp-book/dsp_book_Ch15.pdf.

[6] Butterworth filter - https://en.wikipedia.org/wiki/Butterworth_filter.