

ELEC 421

Digital Signal and Image Processing



Siamak Najarian, Ph.D., P.Eng.,
Professor of Biomedical Engineering (retired),
Electrical and Computer Engineering Department,
University of British Columbia

Course Roadmap for DSP

Lecture	Title
Lecture 0	Introduction to DSP and DIP
Lecture 1	Signals
Lecture 2	Linear Time-Invariant System
Lecture 3	Convolution and its Properties
Lecture 4	The Fourier Series
Lecture 5	The Fourier Transform
Lecture 6	Frequency Response
Lecture 7	Discrete-Time Fourier Transform
Lecture 8	Introduction to the z-Transform
Lecture 9	Inverse z-Transform; Poles and Zeros
Lecture 10	The Discrete Fourier Transform
Lecture 11	Radix-2 Fast Fourier Transforms
Lecture 12	The Cooley-Tukey and Good-Thomas FFTs
Lecture 13	The Sampling Theorem
Lecture 14	Continuous-Time Filtering with Digital Systems; Upsampling and Downsampling
Lecture 15	MATLAB Implementation of Filter Design

Lecture 15:

MATLAB Implementation of Filter Design

Table of Contents

- IIR vs. FIR lowpass filters in DSP
- FIR lowpass filter
- Designing IIR and FIR lowpass filters
- Smoothness, sharpness, and ripple
- Butterworth, Chebyshev, and Elliptic filters
- Design of lowpass Butterworth digital filter using bilinear transformation
- MATLAB demo for filter design
- Using filterDesigner GUI
- How to export designed filter from filterDesigner GUI
- How to use an exported filter file in MATLAB

IIR vs. FIR lowpass filters in DSP

IIR vs. FIR Low Pass Filters in DSP:

- Both **IIR** (Infinite Impulse Response) and **FIR** (Finite Impulse Response) filters are commonly used for low-pass filtering in Digital Signal Processing. Here is a breakdown of their characteristics and key differences:

IIR Low-Pass Filter:

- **Impulse Response:** Infinite duration (the filter's output persists even after the input signal ceases).
- **Design:** Uses mathematical functions to define the filter's frequency response. Popular methods include Butterworth, Chebyshev, and Elliptic.
- **Advantages:**
 - Can achieve steeper roll-off (faster transition from passband to stopband) for a given filter order compared to FIR filters.
 - Requires fewer coefficients for similar performance, leading to potentially lower computational cost.
- **Disadvantages:**
 - Can be unstable if not designed carefully.
 - Non-linear phase response (distorts the order of frequencies in the signal).

FIR lowpass filter

FIR Low-Pass Filter:

- **Impulse Response:** Finite duration (the filter's output eventually reaches zero after the input signal stops).
- **Design:** Uses windowing techniques (e.g., rectangular, Hanning) on ideal impulse responses to achieve the desired frequency response.
- **Advantages:**
 - Always stable (guaranteed not to oscillate).
 - Linear phase response (preserves the order of frequencies in the signal).
 - Easier to design for specific phase requirements.
- **Disadvantages:**
 - Requires more coefficients for similar performance compared to IIR filters, potentially increasing computational cost.
 - Generally has a less steep roll-off for a given filter order.

Designing IIR and FIR lowpass filters

Choosing Between IIR and FIR:

- The choice between IIR and FIR filters depends on your specific application:
 - If steep roll-off and low computational cost are priorities, and non-linear phase is acceptable, consider an IIR filter.
 - If stability, linear phase, or precise control over the phase response is crucial, choose an FIR filter.

Designing IIR and FIR Low-Pass Filters:

General Steps:

1. **Define Specifications:** Determine the desired cutoff frequency (where the filter starts attenuating frequencies), stopband attenuation (amount of reduction in unwanted frequencies), and passband ripple (allowed variation in the passband).
2. **Choose Filter Type:** Decide between IIR or FIR based on your needs (refer to the comparison above).

Designing IIR and FIR lowpass filters

Detailed Steps:

Designing an IIR Low-Pass Filter:

1. **Select a design method:** Choose a method like Butterworth, Chebyshev, or Elliptic based on your *desired roll-off characteristics*, i.e., the desired transition between the passband and stopband frequencies of your filter.
2. **Use design tools:** Utilize software tools or design tables that provide filter coefficients based on your specifications and chosen method.
3. **Implement the filter:** Translate the obtained coefficients into a digital filter structure (e.g., biquad sections) for implementation in your DSP application.

Designing an FIR Low-Pass Filter:

1. **Calculate filter order:** Use formulas or software tools to determine the minimum filter order needed for your specifications.
2. **Design the ideal impulse response:** Create a digital impulse response with the desired frequency response (all ones in the passband, zeros in the stopband).
3. **Apply windowing:** Multiply the ideal impulse response with a window function (e.g., rectangular, Hanning) to achieve a finite impulse response and trade-off between filter performance and transition width.
4. **Implement the filter:** Use the resulting coefficients from the windowed impulse response in a convolution operation for filtering your signal.

Smoothness, sharpness, and ripple

Smoothness, Sharpness, and Ripple:

- In the context of IIR filter design methods like Butterworth, Chebyshev, and Elliptic, here is a breakdown of the key terms related to the filter's frequency response:

Smoothness:

- This refers to **how flat and consistent** the filter's response is **within the passband** (the range of frequencies it allows to pass through).
- A smooth filter response minimizes any unwanted fluctuations or variations in the amplitude of the desired frequencies.
 - **Butterworth filters** prioritize smoothness, aiming for a maximally flat passband response.

Sharpness:

- This describes **how quickly** the filter's response transitions from the passband to the stopband (the range of frequencies it attenuates).
- A sharper filter has a steeper roll-off, meaning it transitions from allowing frequencies to blocking them more abruptly.
 - **Chebyshev** and **Elliptic filters** prioritize sharpness, achieving a steeper roll-off compared to Butterworth filters.

Ripple:

- This refers to any **deviations from the ideal flat response** in either the passband or stopband.
- It manifests as fluctuations in the filter's amplitude response for specific frequencies within these bands.
 - **Chebyshev filters** introduce some passband ripple in exchange for a sharper roll-off compared to Butterworth filters.
 - **Elliptic filters** allow for the sharpest roll-off but introduce significant ripple in both the passband and stopband.

Smoothness, sharpness, and ripple

Analogy:

- Imagine a filter response as a graph showing how much the filter weakens (attenuates) different frequencies.
 - **Smoothness:** Imagine a smooth, straight line in the passband region of the graph. This represents a filter that allows all desired frequencies with minimal variation in their amplitude.
 - **Sharpness:** Imagine a steep cliff in the graph between the passband and stopband regions. This represents a filter that transitions quickly from allowing to blocking frequencies.
 - **Ripple:** Imagine a wavy line instead of a straight line in the passband or stopband regions. These waves represent fluctuations in the filter's response, allowing or attenuating certain frequencies more than others within those bands.

Choosing the Right Method:

- The best filter design method depends on your application's needs:
 - If you prioritize a clean and consistent passband response without significant variations, choose Butterworth for its smoothness.
 - If a faster transition between passband and stopband is crucial, and some passband ripple is acceptable, choose Chebyshev for its sharpness.
 - If an extremely sharp cutoff is absolutely necessary, even with significant ripple in both passband and stopband, Elliptic might be the choice.

Conclusion: Smoothness, sharpness, and ripple are all interrelated characteristics of an IIR filter's frequency response. Understanding these trade-offs allows us to select the most appropriate design method for our specific signal processing task.

Butterworth, Chebyshev, and Elliptic filters

Butterworth, Chebyshev, and Elliptic Filters:

- In IIR filter design, choosing a method like Butterworth, Chebyshev, or Elliptic refers to selecting a specific approach to achieve the desired transition between the passband and stopband frequencies of your filter (also known as the ***roll-off characteristic***). Here is a breakdown of these methods and their impact on roll-off:

1. Ideal Filter Response:

Imagine an ideal low-pass filter. It perfectly allows all frequencies below a certain cutoff frequency (passband) and completely blocks all frequencies above it (stopband). The transition between these regions is instantaneous (infinitely steep roll-off).

2. Real-World Limitations:

Unfortunately, such an ideal filter cannot be implemented in practice. Real-world filters require a gradual transition between passband and stopband. The rate of this transition, or steepness of the roll-off, is a crucial design parameter.

3. Filter Design Methods:

The methods of Butterworth, Chebyshev, Elliptic are different approaches to designing IIR filters with varying roll-off characteristics.

Butterworth, Chebyshev, and Elliptic filters

- **Butterworth Filter:**
 - Offers a maximally flat passband response (minimizes ripple within the passband).
 - Has a gradual roll-off from passband to stopband.
 - This method is a good choice when a *smooth transition* is more important than a *sharp cutoff*.
- **Chebyshev Filter:**
 - Allows for a sharper roll-off compared to Butterworth filters for the same filter order.
 - Introduces some ripple within the passband (deviations from the ideal flat response).
 - This method is useful when a steeper cutoff is crucial, even if a slight variation in the passband is acceptable.
- **Elliptic Filter:**
 - Provides the most aggressive roll-off for a given filter order, achieving a very sharp transition between passband and stopband.
 - Introduces significant ripple in both the passband and stopband.
 - This method is used when an extremely sharp cutoff is essential, and some ripple in both passband and stopband is tolerable.

Choosing the Right Method:

- The best method for you depends on your specific needs:
 - **Prioritize flat passband and smooth transition:** Choose Butterworth.
 - **Need a sharper cutoff and can tolerate some passband ripple:** Choose Chebyshev.
 - **Absolutely require the sharpest possible cutoff, even with significant ripple:** Choose Elliptic.

Conclusion: These filter design methods offer different ways to control the roll-off characteristic of your IIR filter. By understanding the trade-offs between smoothness, sharpness, and ripple, you can choose the method that best suits your application's requirements.

Design of lowpass Butterworth digital filter using bilinear transformation

- **Step 1:** Compute digital frequency, ω_c in radians.
- **Step 2:** Compute pre-warped analog frequency, Ω_c radians/sec. Here, T is the sampling period in seconds.

$$\Omega_c = \frac{2}{T} \tan\left(\frac{\omega_c}{2}\right)$$

- **Step 3:** Compute normalized transfer function, $H_N(s)$, (also called prototype transfer function) for the given filter order, N .

$$H_N(s) = \frac{1}{B_N(s)} \quad ; \quad B_N(s) = \prod_{k=0}^{N-1} (s - p_k) \quad ; \quad p_k = e^{j\left(\frac{\pi}{2} + \frac{(2k+1)\pi}{2N}\right)}, \quad k = 0, 1, 2, \dots, N-1$$

- **Step 4:** Compute analog lowpass filter transfer function, $H(s)$. Substitute s/Ω_c for s in $H_N(s)$.

$$H(s) = H_N\left(\frac{s}{\Omega_c}\right)$$

- **Step 5:** Compute digital filter, $H(z)$, in standard form, by using the following bilinear transformation:

$$H(z) = H(s) \quad \text{where} \quad s = \frac{2}{T} \cdot \frac{z-1}{z+1} \quad \longrightarrow \quad H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_N z^{-N}}$$

Design of lowpass Butterworth digital filter using bilinear transformation

IIR Filter Topologies:

- An IIR filter has feedback, thus the general form of an **N**-order IIR filter is:

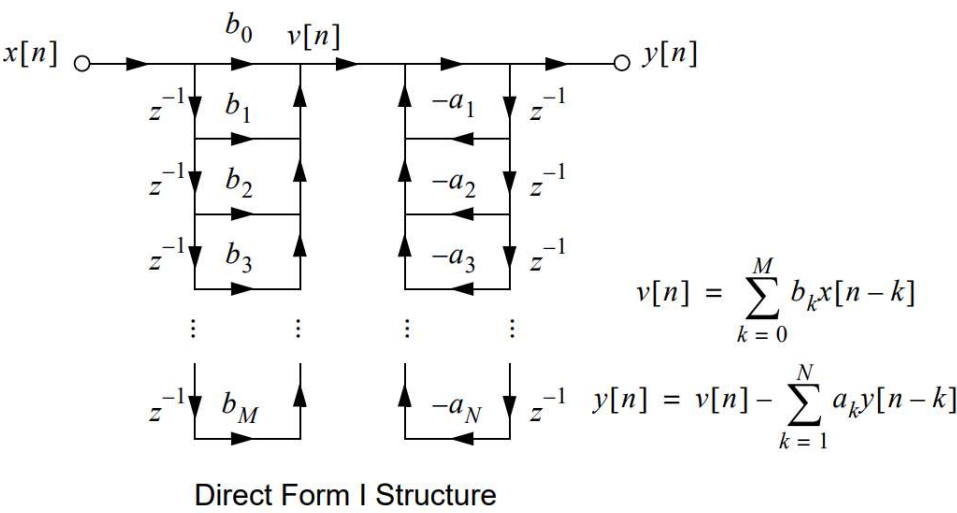
$$y[n] = - \sum_{k=1}^N a_k y[n-k] + \sum_{r=0}^M b_r x[n-r]$$

- By **z**-transforming both sides of the above equation and using the fact that $H(z) = Y(z)/X(z)$, we can write:

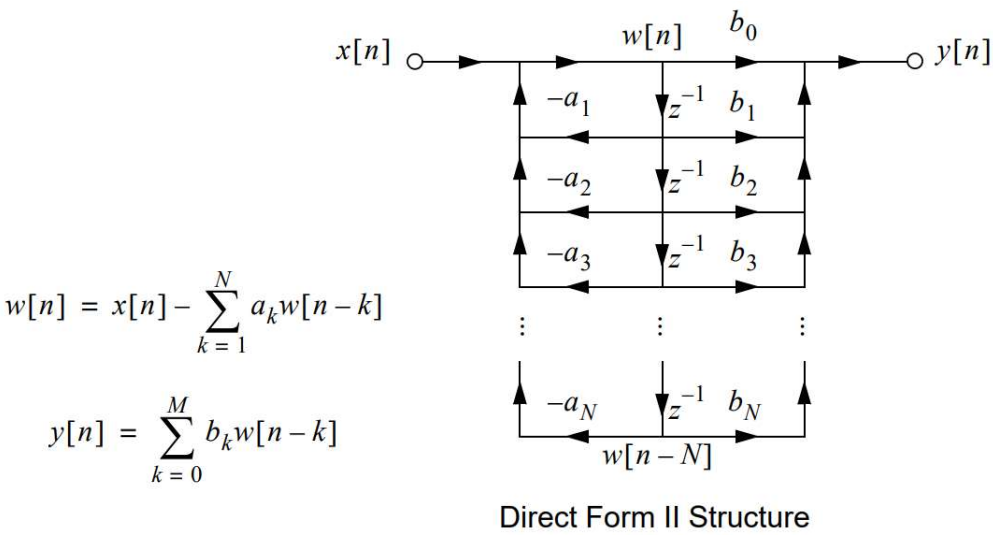
$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_N z^{-N}}$$

- IIR filters can be implemented in a variety of topologies, the most common ones are direct form I, II, cascade, and parallel.

Direct Form I Structure:

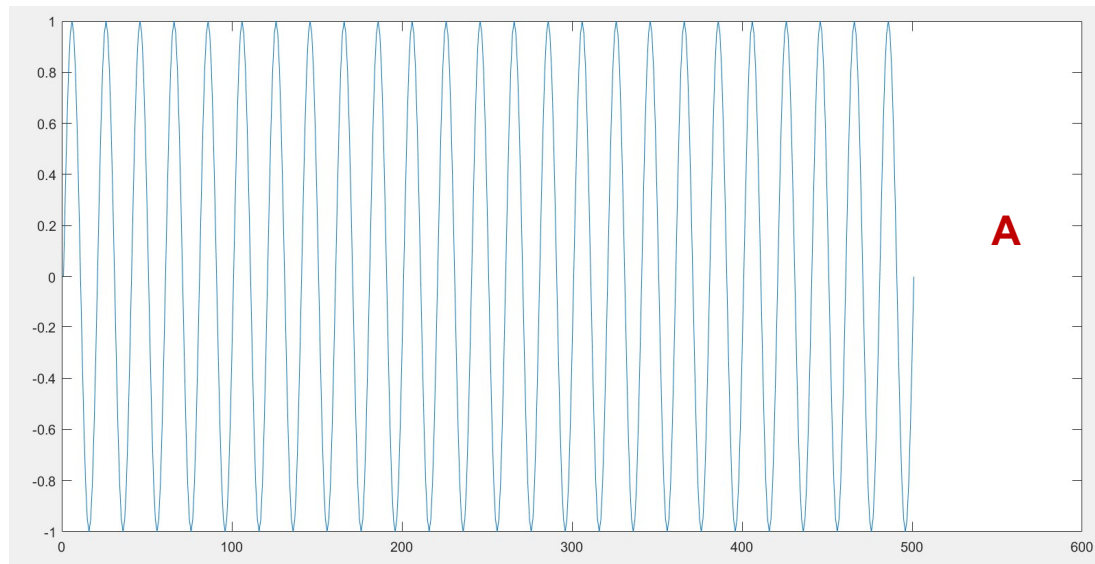


Direct Form II Structure:



MATLAB demo for filter design

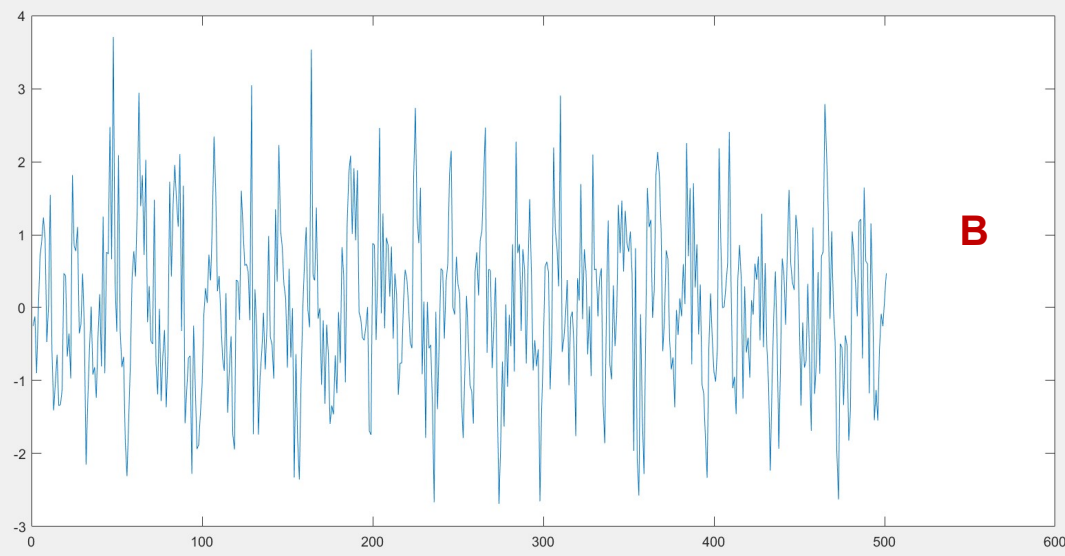
```
>> fs = 100; f = 5; t = 5 ; n = [0:1/fs:t] ; x = sin(2*pi*f*n); plot(x)
```



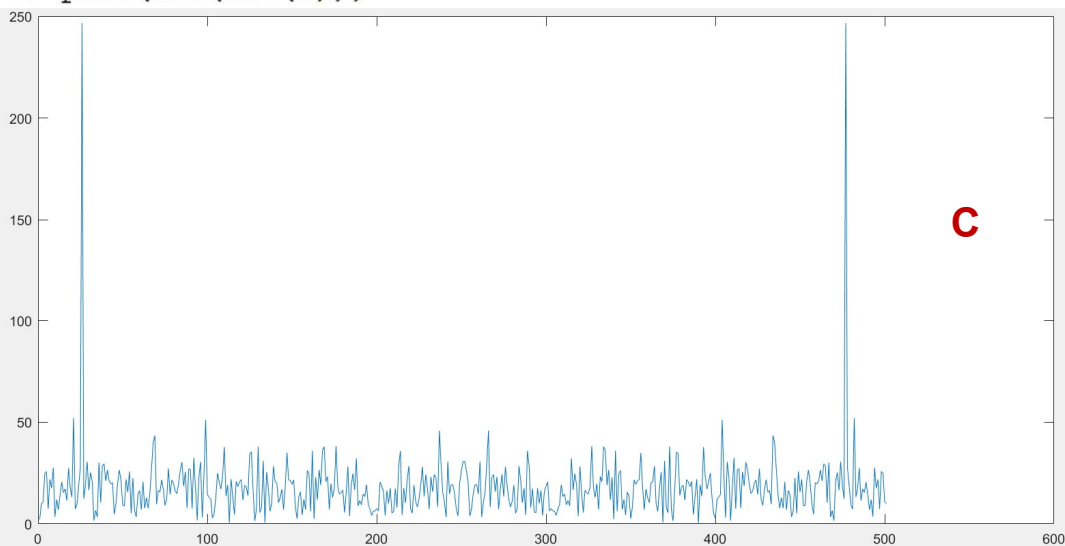
- In MATLAB, first, we need to define the signal which needs to be filtered out. We define the sampling frequency, **fs = 100**, and the signal frequency, **f = 5**. We are generating a signal for **t = 5** seconds. Then, **n = [0:1/fs:t]** will generate the time axis. Next, we generate a sinusoid signal, **x = sin(2*pi*f*n)** and then we plot it using **plot(x)**.
- Graph **A** shows our sinusoidal signal of **5** Hertz frequency with a sampling rate of **100**. Note that the graph looks continuous, but it is actually a discrete function. This is due to the large number of sampling rate. Some plotting libraries in MATLAB automatically use the number of data point on the **x**-axis instead of time.

MATLAB demo for filter design

```
z=awgn(x,1);plot(z)
```

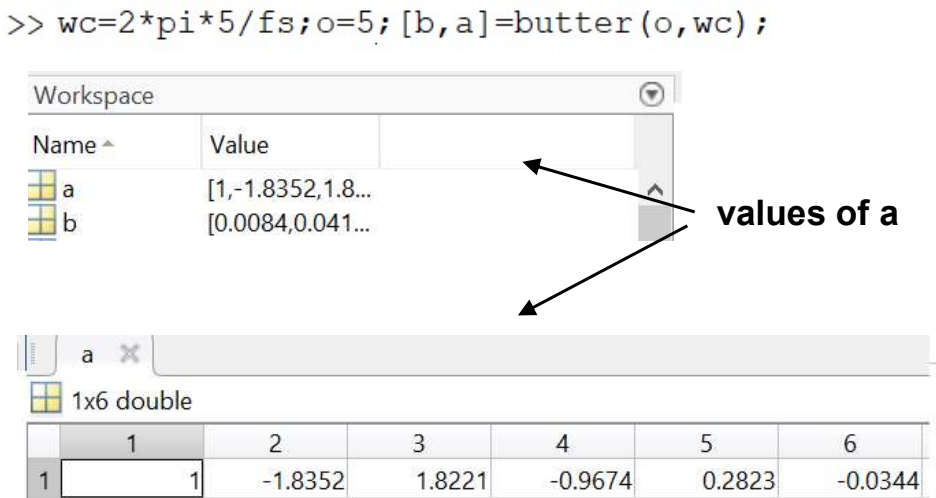


```
>> plot(abs(fft(z)))
```



- Let us now add noise into the signal. We use $z = \text{awgn}(x,1)$, which is an additive white Gaussian noise. The code $z = \text{awgn}(x, 1)$ in MATLAB adds white Gaussian noise (AWGN) to the signal x with a specific Signal-to-Noise Ratio (SNR) of 1 dB.
- Graph **B** shows our sinusoidal signal after adding noise into it. The noise has distorted most of the properties of the signal.
- Let us now find the Fourier transform or the spectrum of this noisy signal, We use $\text{plot}(\text{abs}(\text{fft}(z)))$ to graph the FFT. As we can see in **C**, all the frequency range are filled with the noise and it is quite high.

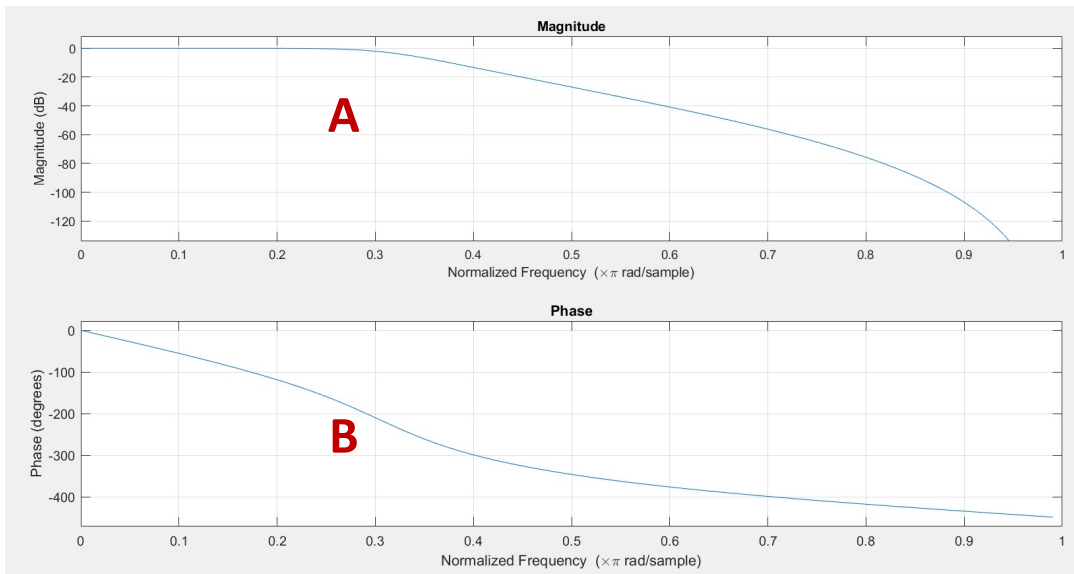
MATLAB demo for filter design



- Let us now design our digital filter. We will use an IIR lowpass filter, **Butterworth filter**. Here, this filter passes all the frequency which is up to the cutoff frequency and it stops all the frequency which is outside the cutoff frequency. The order of the filter determines how sharp there will be a transition from passband to stopband. Keeping all this in mind, first, we define the order of the filter to be 5, **o = 5**. Next, we define the cutoff frequency, **wc = (pi*f)/(fs/2) = 2*(pi*f/fs)** or in this case, **wc = 2*(pi*5/100) = 10*(pi/100)**. Here, we have **wc = 0.3141**.
- Because our signal is of 5 Hertz, so it is better to have a cutoff frequency of 10 Hertz so that it is well within the range of the passband. If we take the cutoff frequency of 5 Hertz, our signal frequency which we want to filter out will be very near to the edge of the cutoff and that may create a problem. So, we will take additional 5 Hz band in order to make sure signal frequency is well within the range of the passband. In our case, let us use **wc = 0.3141**.
- The command **[b,a] = butter(o, wc)** defines a Butterworth filter and gives us the numerator and denominator coefficient terms. Here, **b** is the numerator coefficient and **a** is the denominator coefficient of the **IIR transfer function**. The numerical values of these two coefficients can be seen by clicking on them in workspace, as shown by **A**.

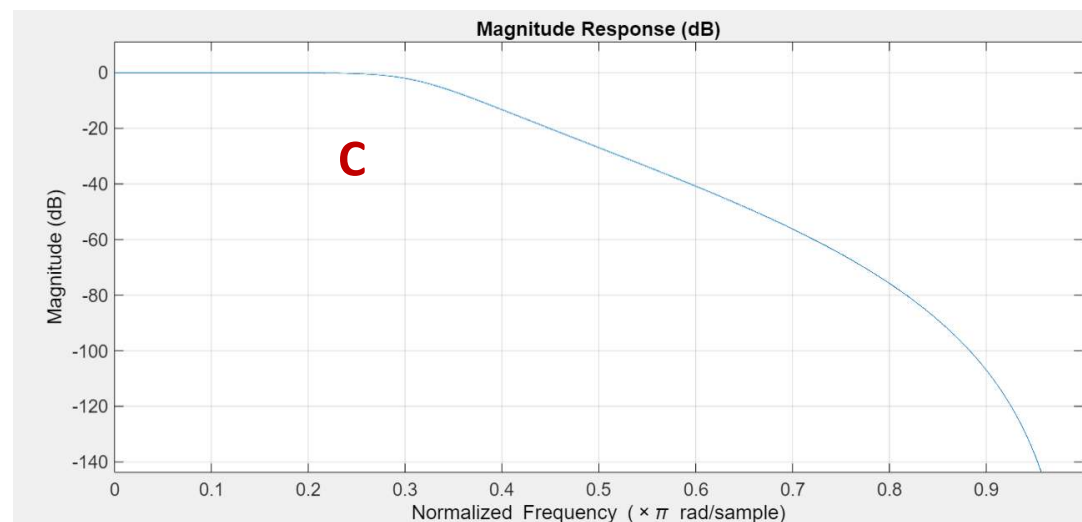
MATLAB demo for filter design

```
>> freqz(b,a)
```



- If we are interested in looking at the frequency response of the filter to see whether our design criteria has been met, we can use **freqz(b,a)**. These plots are shown in **A** and **B**. Our cutoff is about **$0.3141 \times \pi$** radians per sample. In **A**, between **0** and about **0.3141** , we have our passband and after that, we have our stopband.
- There is another tool we can use as well in order to display the frequency response of the filter. It is **fvtool(b,a)**. The filter response is shown in **C**, which is the same as **A**.

```
>> fvtool(b,a)
```

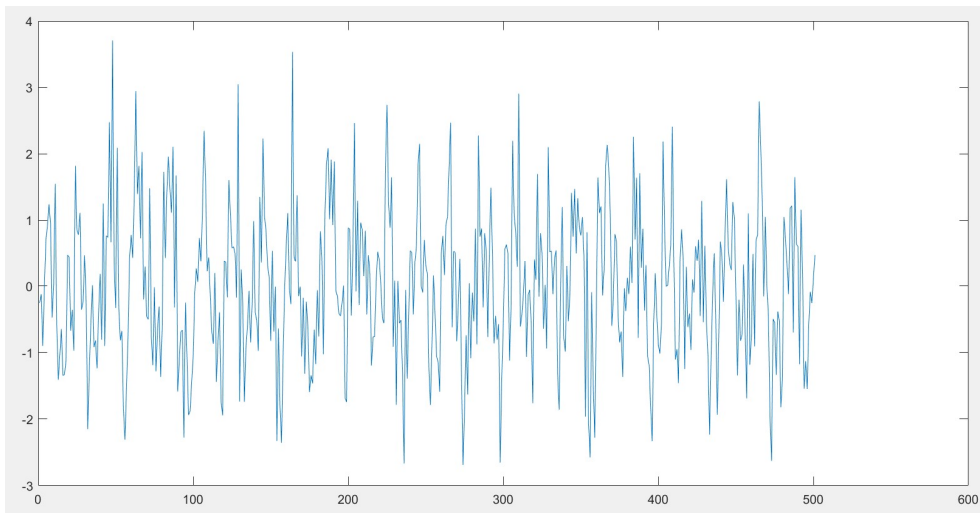


MATLAB demo for filter design

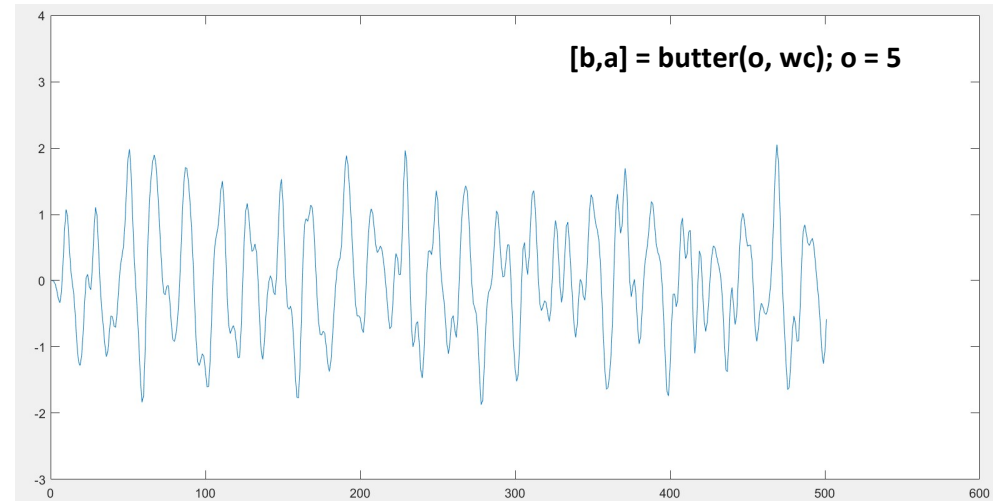
- Now, we can use the **filter command** in MATLAB, which is $\mathbf{x_f} = \text{filter}(\mathbf{b}, \mathbf{a}, \mathbf{z})$, to filter out the noise to some extent and improve the noisy signal. The plot of $\mathbf{x_f}$ is shown in **B**. Here, we can compare the noisy signal, **A**, with the denoised signal, **B**, side-by-side. Our filtered signal shows some improvement over the noisy signal.
- Note that to change the range of **x**- and **y**-axis in MATLAB plots, we can do this: Axis Properties → Rulers → Xlim or Ylim. Then, by clicking on the three vertical dots icon, we can change the minimum and maximum range for **x** and **y** values on the axes.

```
>> x_f=filter(b,a,z); plot(x_f)
```

A: noisy signal

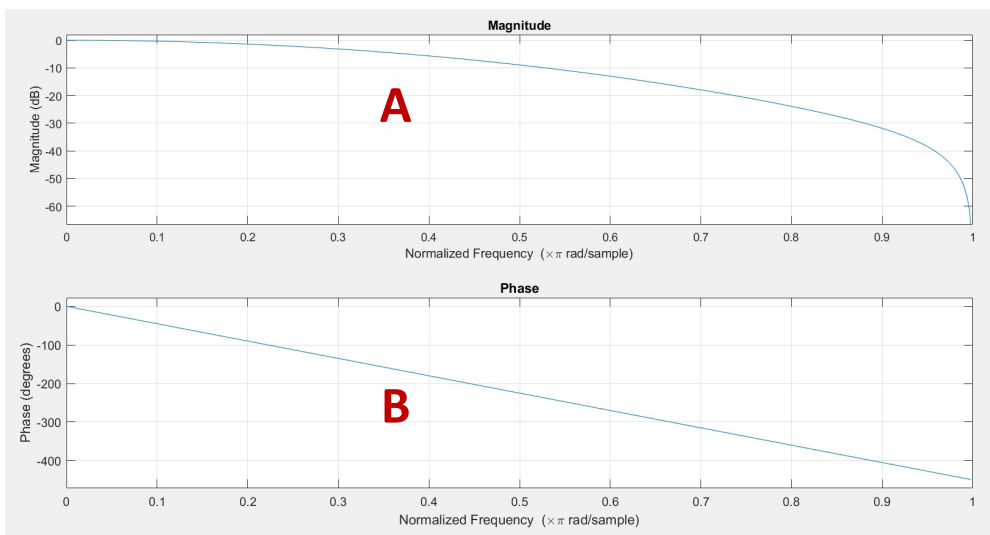


B: denoised signal

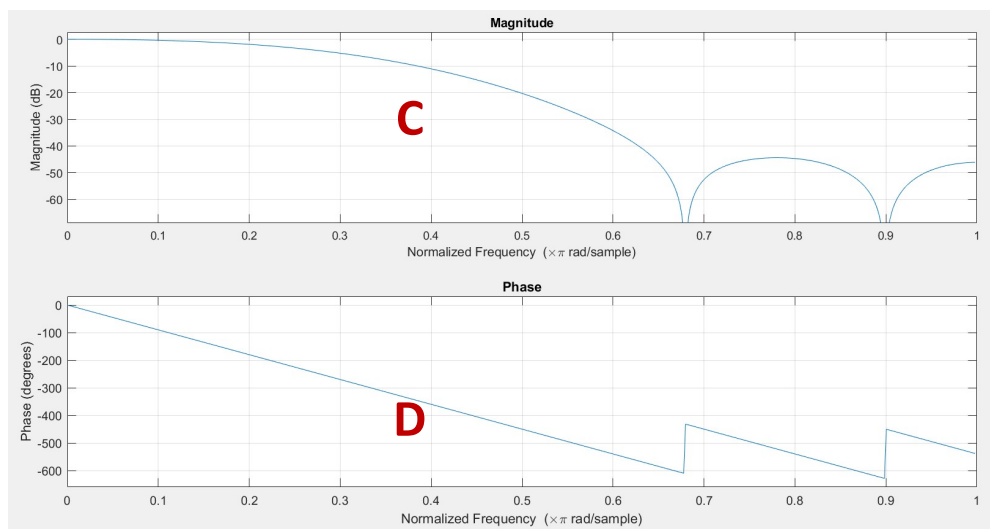


MATLAB demo for filter design

>> [b,a]=fir1(o,wc) For o = 5.



>> [b,a]=fir1(o,wc) For o = 10.

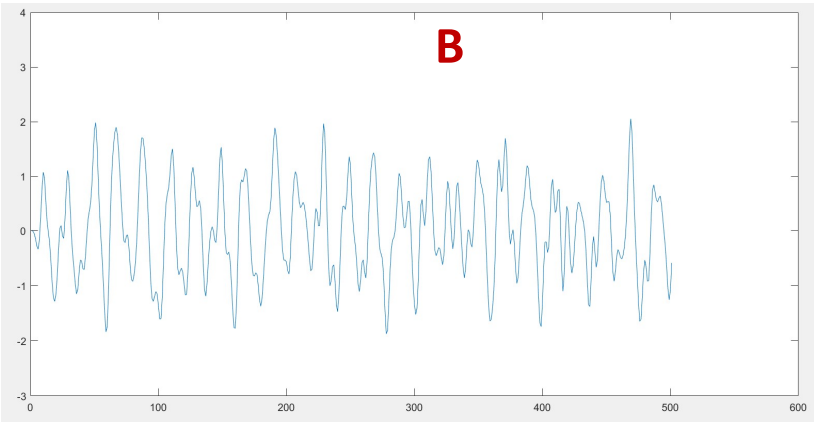
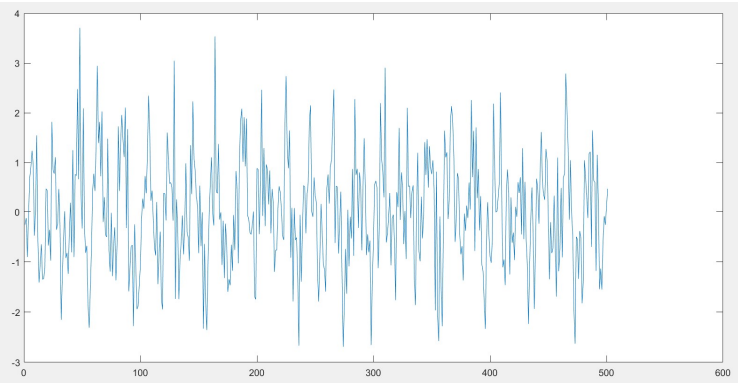


- Similarly, we can also design an **FIR filter**. First, we will use the same order, **o = 5**, and a cutoff frequency, **wc = 2*pi*5/fs**. The command for an FIR filter is **[b,a] = fir1(o,wc)**. Graphs **A** and **B** show the frequency response of our FIR filter for an order of 5.
- Now, let us try to change the order of the filter, because, generally, FIR requires higher order than that of the IIR filter. Here, we use **o = 10**. We can see that now, the frequency response is slightly better than the previous one, as shown in **C** and **D**.
- One of the reasons why we use FIR is due to its phase response. Here, in both **B** and **D**, we see a **linear phase response** in the filter output. So, we expect for the distortion to be less in the output signal. But in the case of IIR filters, they do not have a linear phase, and hence, most of the times, the output signal gets distorted due to this nonlinear phase property of the IIR filters.

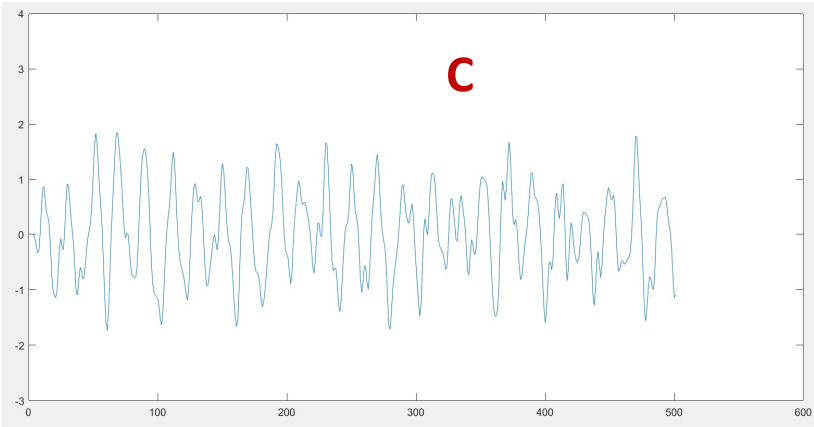
MATLAB demo for filter design

- Graph **A** shows the noisy signal, while Graphs **B**, **C**, and **D** show the denoised signal with various types of filters. In this case, using a slightly higher order of **10** does not lead to a major improvement.

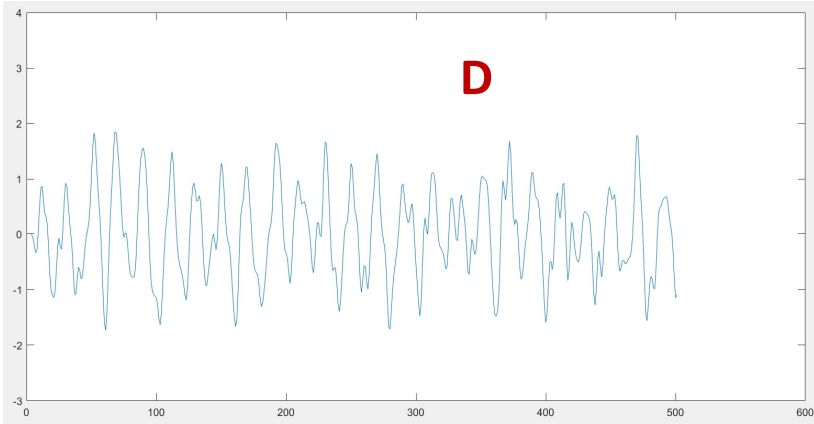
A: noisy signal



`[b,a] = butter(o, wc); o = 5`



`[b,a] = fir1(o, wc); o = 5`

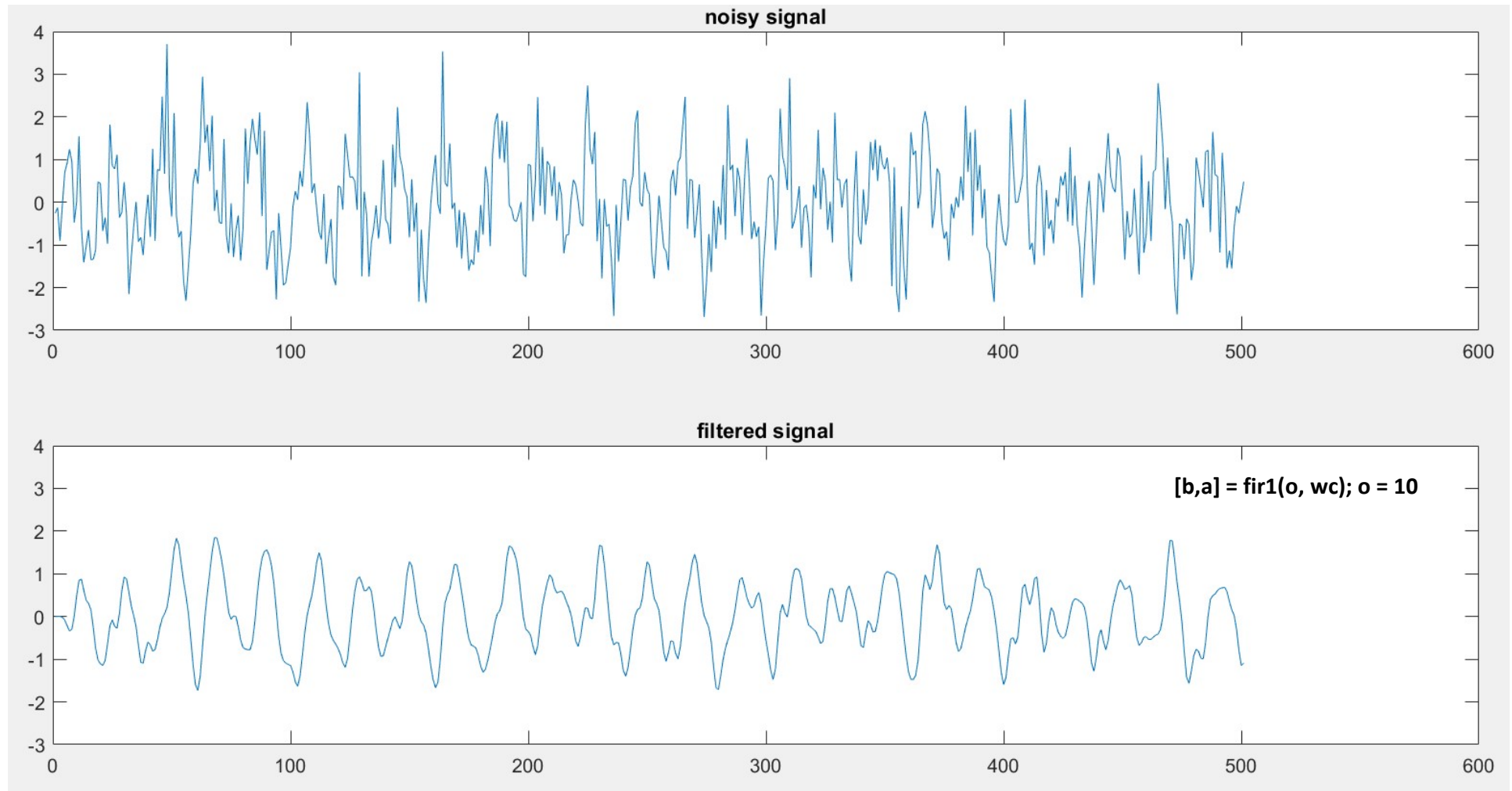


`[b,a] = fir1(o, wc); o = 10`

MATLAB demo for filter design

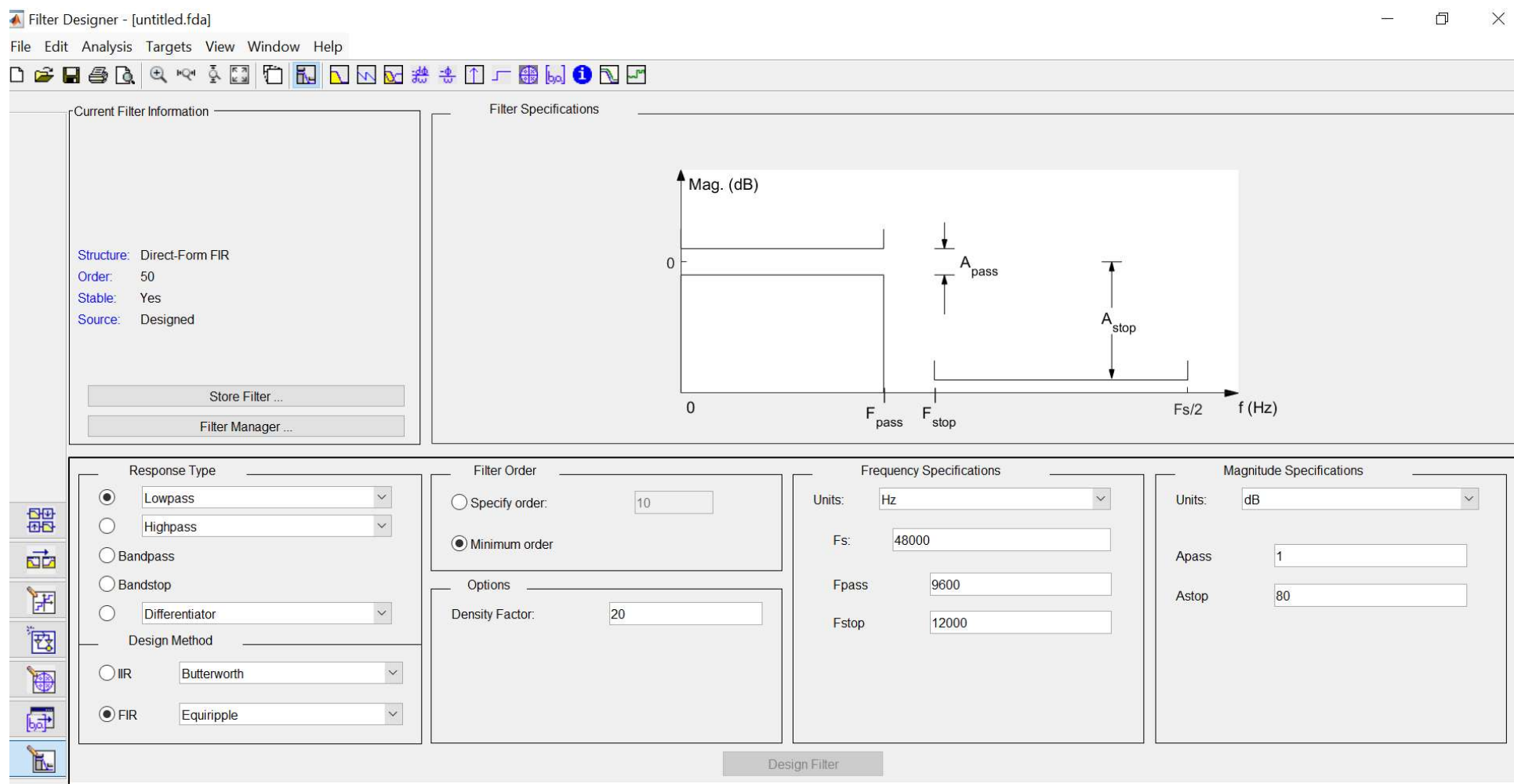
```
>> subplot(2,1,1);plot(z);title('noisy signal');  
>> subplot(2,1,2);plot(x_f); title('filtered signal');
```

- Clearly, we succeeded in removing most of the noise from the signal, as shown in the **filtered signal** graph.



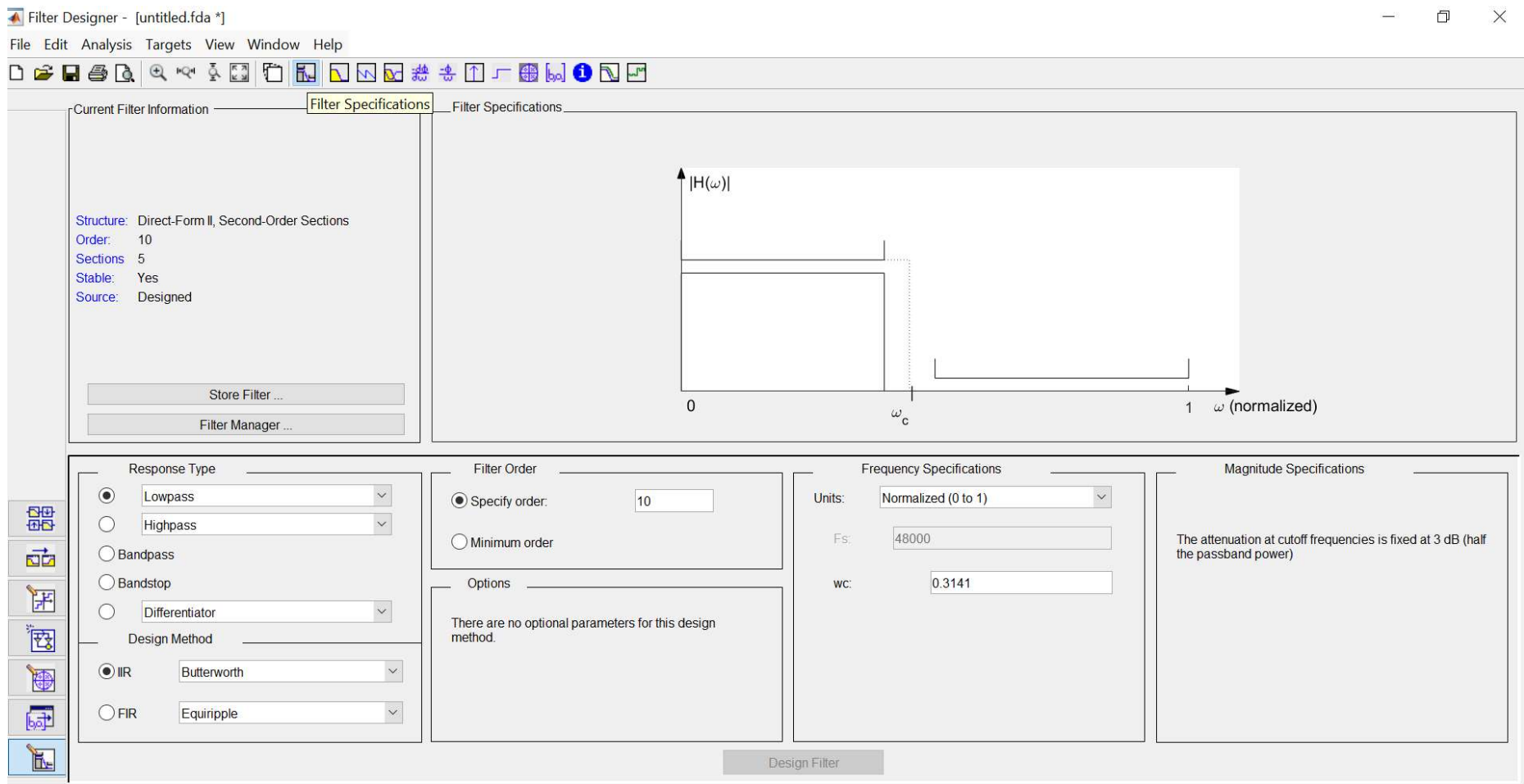
Using filterDesigner GUI

- Now, we will be focusing on how to effectively **design the FIR and IIR filters** of various kinds and **export** them by using MATLAB code. First, we need to open the graphical user interface or GUI to design the filters. This GUI is called **filter designer** and the command is “**filterDesigner**”. After typing filterDesigner and pressing enter, MATLAB will open the tool. The main interface is shown below.



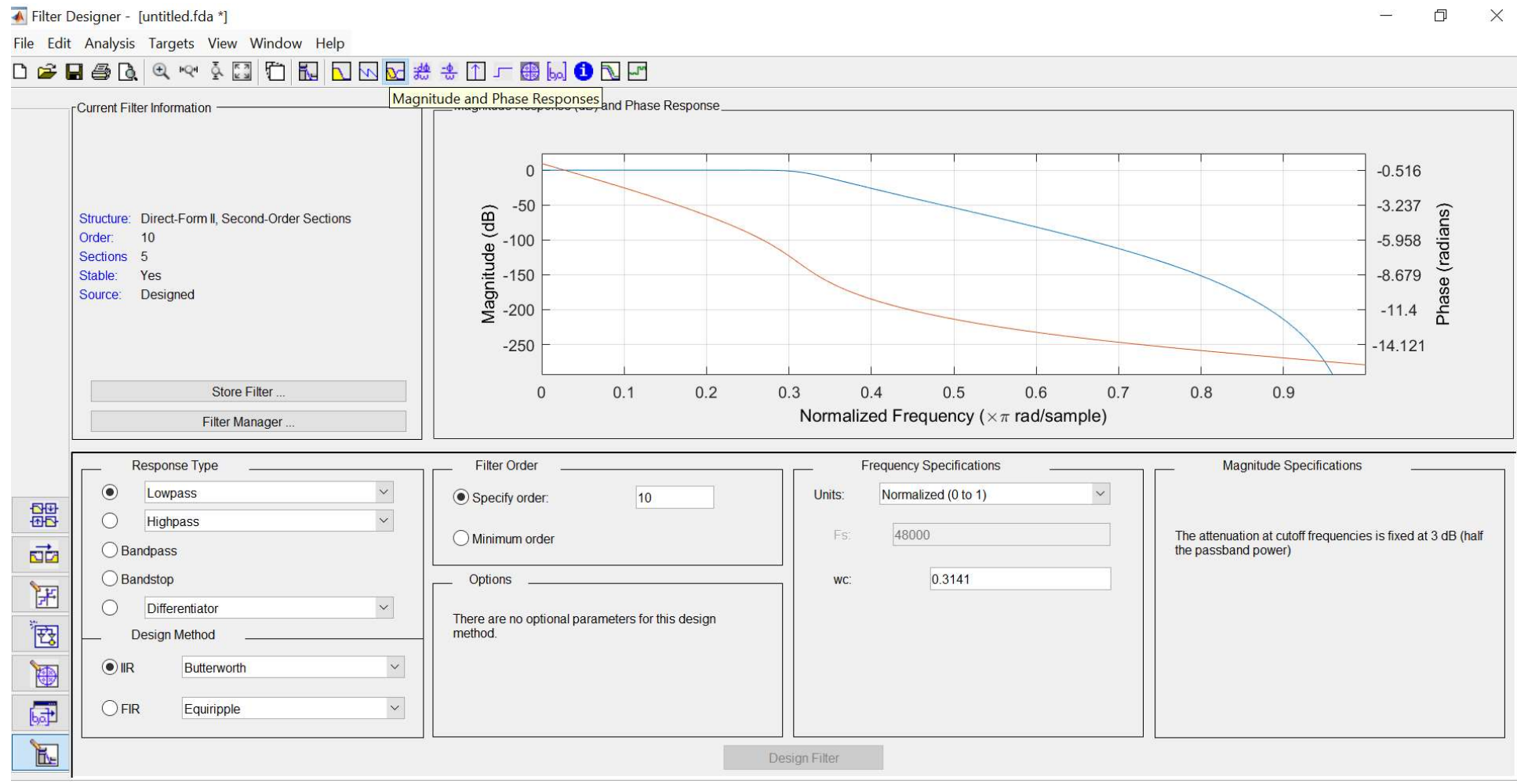
Using filterDesigner GUI

- Once the main interface opens, we are in the **Filter Specifications** window. Start with **Response Type**, and then select **Lowpass**. Next, for **Design Method**, select **Butterworth**. For **Filter Order**, select **10**. For **Frequency Specifications**, for **Units**, select **Normalized (0 to 1)**, and for **wc**, type in **0.3141**. Now, press **Design Filter** at the bottom of the window.



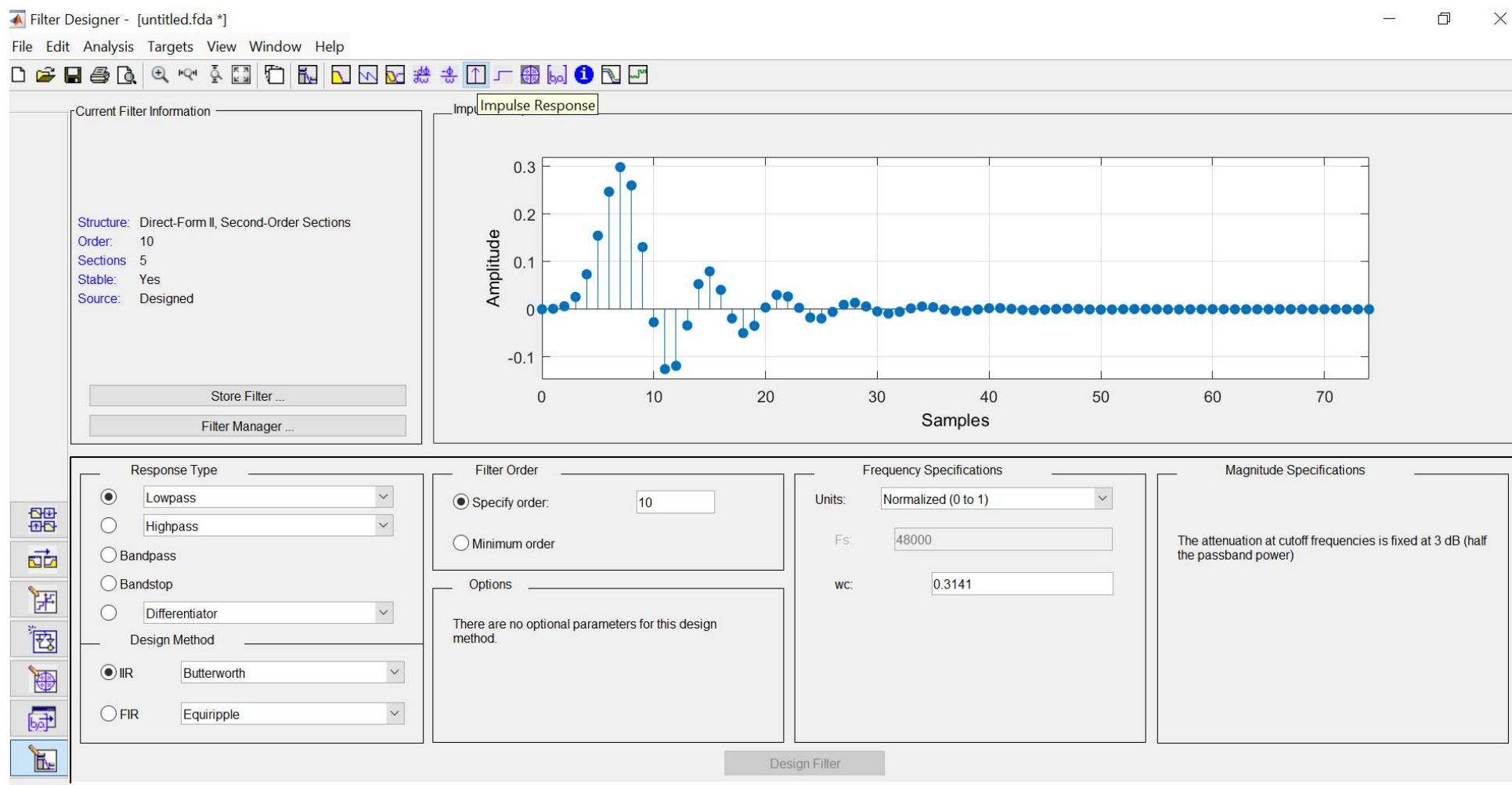
Using filterDesigner GUI

- By clicking on **Magnitude and Phase Response** icon in the top ribbon, we can see a plot of both responses.



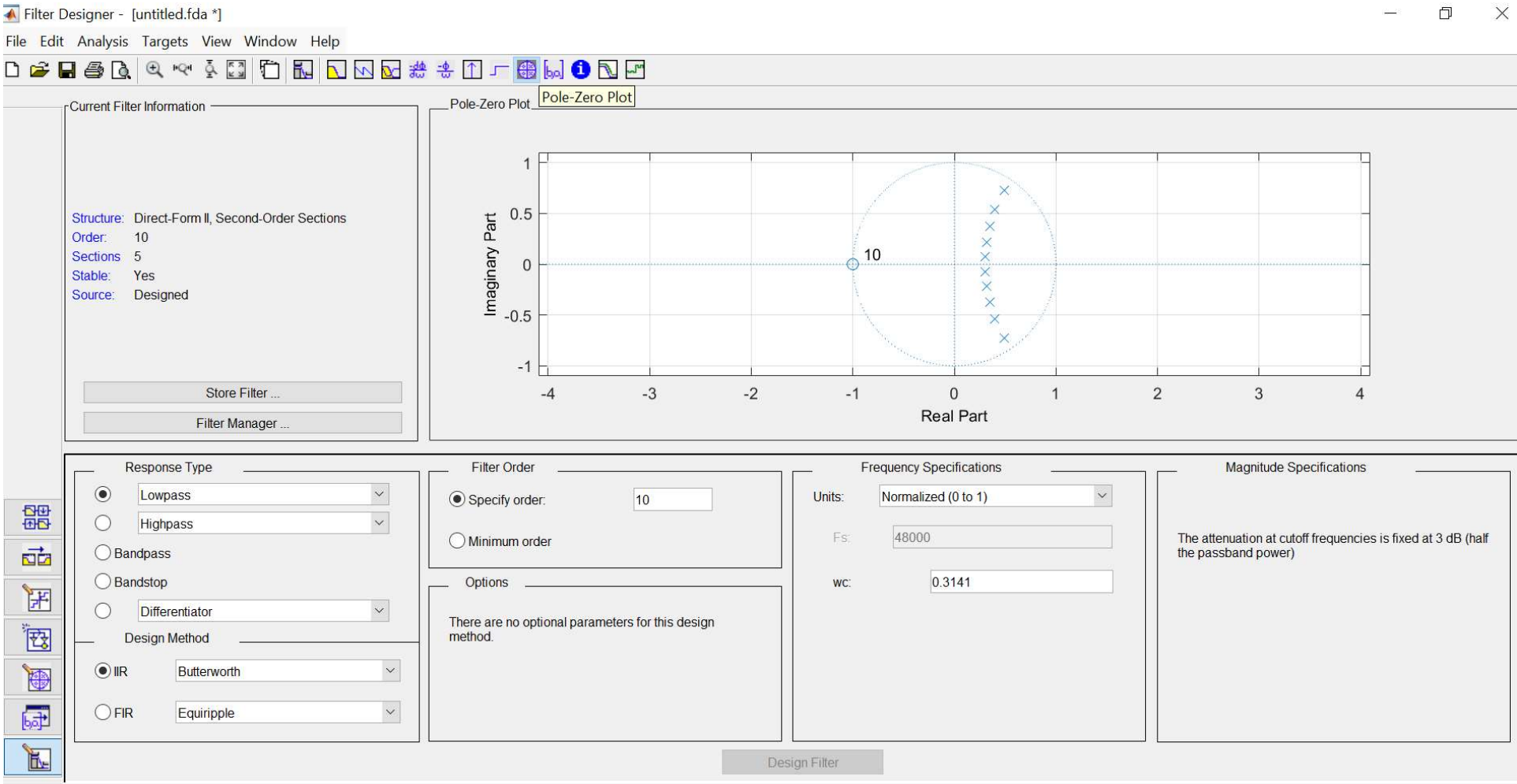
Using filterDesigner GUI

- By clicking on **Impulse Response** icon in the top ribbon, we can see the impulse response in the form of a plot of Amplitude vs. Samples.



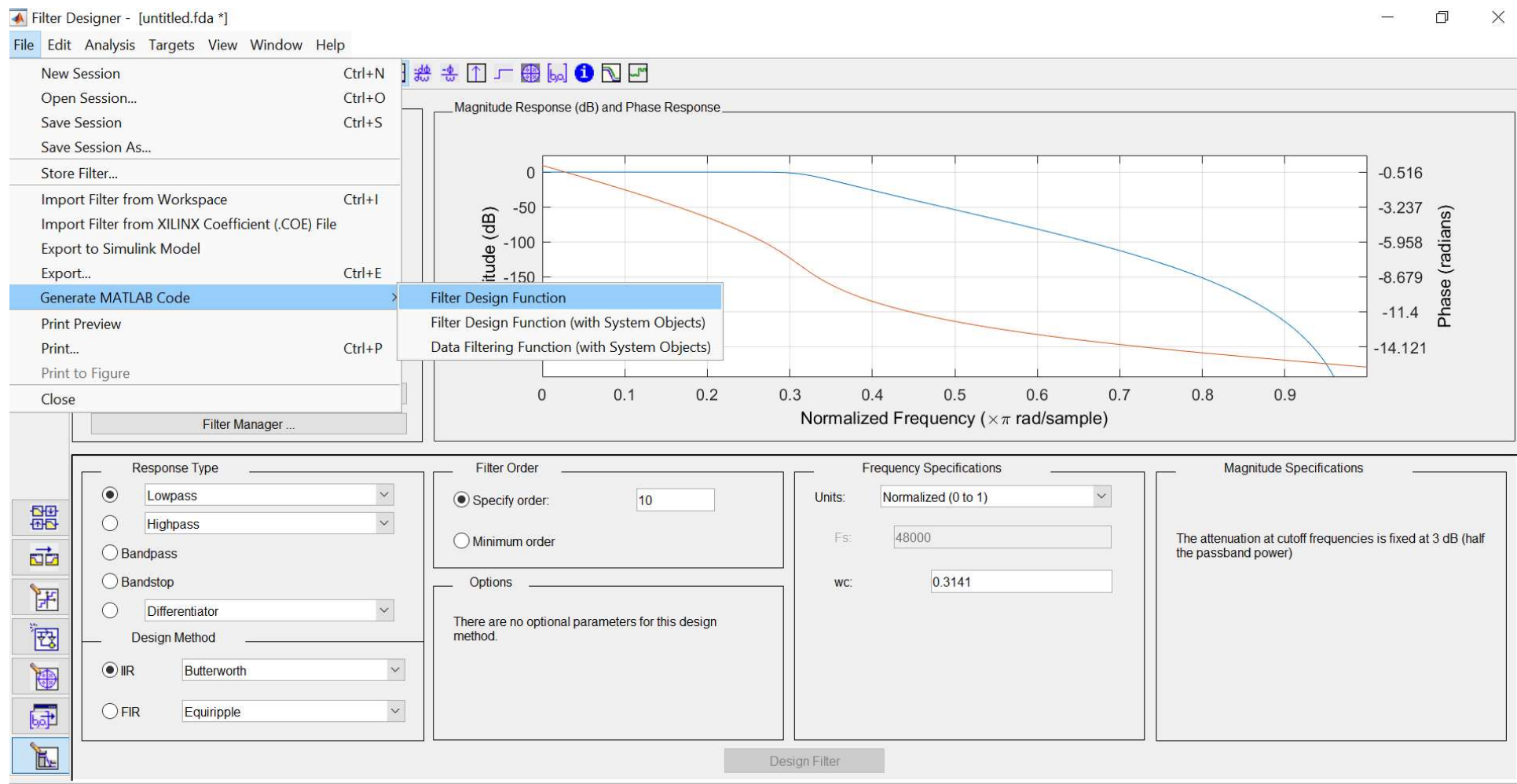
Using filterDesigner GUI

- By clicking on **Pole-Zero Plot** icon in the top ribbon, we can see the pole-zero plot in the s-plane.



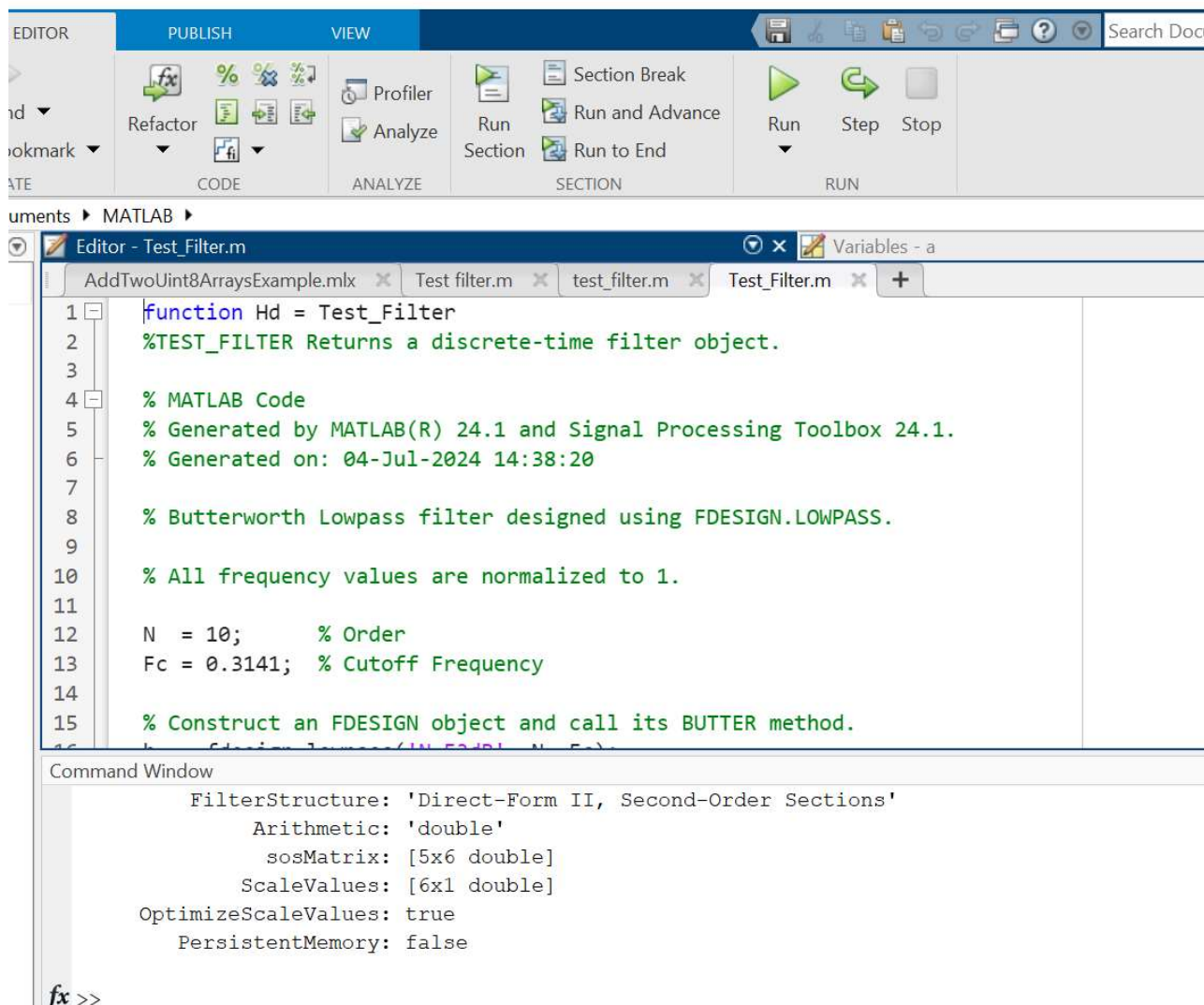
How to export designed filter from filterDesigner GUI

- **How to export the designed filter?** Follow this sequence of steps: click on **Generate MATLAB Code**, **Filter Design Function**, and then save the file with the name **Test_Filter** in MATLAB folder.



How to export a designed filter from filterDesigner GUI

- Now, click on **Run** key at the top of MATLAB main window and the filter is ready to be used whenever we need it. Before using the filter, type in **Hd = ans** as our ready-to-be-used designed filter.



The screenshot displays the MATLAB Editor interface. The top toolbar includes the 'Run' button (a green play icon). Below the toolbar, the Editor window shows a script named 'Test_Filter.m'. The script defines a function 'Hd = Test_Filter' that generates a Butterworth Lowpass filter. The Command Window at the bottom shows the output of the function, displaying the filter structure and parameters.

```
function Hd = Test_Filter
%TEST_FILTER Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 24.1 and Signal Processing Toolbox 24.1.
% Generated on: 04-Jul-2024 14:38:20

% Butterworth Lowpass filter designed using FDESIGN.LOWPASS.

% All frequency values are normalized to 1.

N = 10;      % Order
Fc = 0.3141; % Cutoff Frequency

% Construct an FDESIGN object and call its BUTTER method.
h = fdesign('butter', 'N, Fc', N, Fc);
Hd = buttap(h);
```

Command Window

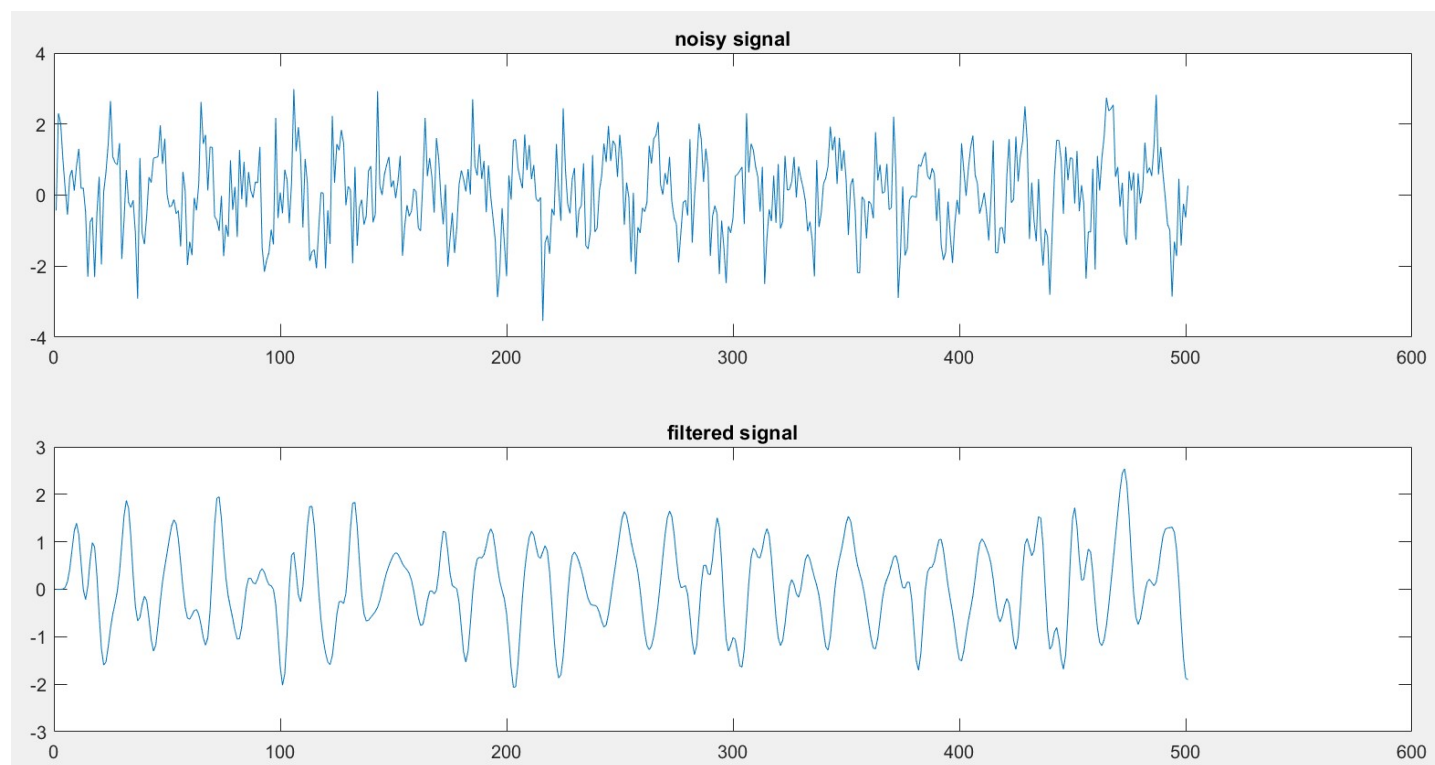
```
FilterStructure: 'Direct-Form II, Second-Order Sections'
Arithmetic: 'double'
sosMatrix: [5x6 double]
ScaleValues: [6x1 double]
OptimizeScaleValues: true
PersistentMemory: false
```

fx >>

How to use an exported filter file in MATLAB

- **How to use this filter for our data?** We will use the same data we had before, i.e., the original noisy sinusoid, and then we will filter it. Our filtered signal is now $\mathbf{x}_f = \text{filter}(\mathbf{H}_d, \mathbf{z})$. Finally, we can plot the noisy and denoised signals, as shown below:

```
>> Hd =ans;  
>> fs=100;f=5;t=5;n=[0:1/fs:t];x=sin(2*pi*f*n);z=awgn(x,1);wc=2*pi*5/fs;x_f=filter(Hd,z)  
>> subplot(2,1,1);plot(z);title('noisy signal');subplot(2,1,2);plot(x_f); title('filtered signal');
```



Course Roadmap for DSP

Lecture	Title
Lecture 0	Introduction to DSP and DIP
Lecture 1	Signals
Lecture 2	Linear Time-Invariant System
Lecture 3	Convolution and its Properties
Lecture 4	The Fourier Series
Lecture 5	The Fourier Transform
Lecture 6	Frequency Response
Lecture 7	Discrete-Time Fourier Transform
Lecture 8	Introduction to the z-Transform
Lecture 9	Inverse z-Transform; Poles and Zeros
Lecture 10	The Discrete Fourier Transform
Lecture 11	Radix-2 Fast Fourier Transforms
Lecture 12	The Cooley-Tukey and Good-Thomas FFTs
Lecture 13	The Sampling Theorem
Lecture 14	Continuous-Time Filtering with Digital Systems; Upsampling and Downsampling
Lecture 15	MATLAB Implementation of Filter Design

End of Lecture 15