College of Electronics Engineering,
Biomedical Engineering Department
Semester 6
2025-2026
Biomedical Signal Analysis Laboratory

Experiment [2-1]: *Digital Filter*

A **digital filter** is a central component in signal processing systems used to suppress or eliminate unwanted artifacts from signals to enhance their quality. Mathematically, they are considered **linear time-invariant (LTI) systems** that take an input signal, pass it through a transfer function (often involving an impulse response), and produce a processed output signal.

Benefits and the Shift from Analog

The transition from analog to digital filters was driven by several key advantages:

- **Mathematical Simplicity:** In the time domain, filtering involves **convolution**, which is a complex mathematical process. Digital signal processing allows us to convert signals into the **frequency domain** using the **Fourier Transform**, where convolution becomes simple **multiplication**.
- **Tunability and Flexibility:** Unlike fixed analog components, digital filters can be **tunable at runtime**. You can modify parameters like bandwidth or center frequency during a simulation without needing to redesign physical circuits or update model diagrams.
- **Precision and Performance:** Digital filters can achieve very high performance. For example, a tunable backward filter can track a varying signal (like a chirp) and achieve a **Signal-to-Noise Ratio (SNR)** of approximately 90 dB, significantly cleaner than raw or poorly filtered signals.
- **Iterative Design:** Digital tools allow for a **trial-and-error approach**, which is beneficial when the exact noise parameters are unknown beforehand.

Characteristics of Digital Filters

Several fundamental traits categorize digital filters:

- **Impulse Response Types:**
    - **FIR (Finite Impulse Response):** These filters have an impulse response of finite length and are theoretically simpler.
    - **IIR (Infinite Impulse Response):** These are more complex theoretically but can achieve the same results with a much lower **filter order**, saving memory and computational resources.
- **Filter Order:** This is defined by the order of the filter's transfer function. A higher order generally means a sharper filter but requires more memory and hardware resources.

- **Standard Responses:**
    - **Low Pass:** Preserves low frequencies and attenuates high-frequency noise.
    - **High Pass:** Preserves high frequencies and attenuates low frequencies.
    - **Band Pass:** Preserves a specific frequency range.
    - **Band Stop:** Attenuates a specific frequency range.

Working with Filters in MATLAB

MATLAB provides both graphical and programmatic ways to design and implement filters:

1. **Filter Designer Tool:** This is a graphical user interface (GUI) that allows you to design filters using dropdown menus, text boxes, and radio buttons for specifications like response type and filter order. You can use this tool to observe response types such as **magnitude response, phase response, and pole-zero plots**.
2. **Implementation Functions:**
    - **fft:** Used to convert signals from the time domain to the frequency domain for analysis.
    - **filter:** This function applies a designed filter object to a signal.
    - **fir1:** A common function used to design FIR filter impulse responses.
3. **Simulink Integration:** You can export filters from the Filter Designer directly into **Simulink** as blocks. Alternatively, you can use the **Digital Filter Design block** within the DSP System Toolbox to design filters directly inside a model.
4. **Hardware Deployment:** Once a filter is designed, MATLAB and Simulink provide options to **generate code** (C or HDL) for various hardware targets, allowing you to deploy your filter to real-world sensors or robots.

To continue the explanation of designing digital filters in MATLAB, the process typically follows an **iterative, multi-step workflow** that moves from signal analysis to interactive design, and finally to implementation or hardware deployment.

1. Signal Analysis and Frequency Conversion

The first step in designing an effective filter is understanding the frequency characteristics of your signal and its noise.

- **Time vs. Frequency Domain:** While signals are often recorded in the time domain, analyzing them in the frequency domain makes it easier to identify unwanted components.
- **The fft Function:** In MATLAB, you use the **Fast Fourier Transform (fft)** to convert a signal from the time domain to the frequency domain.
- **Magnitude Analysis:** Because fft returns complex numbers, you must use the abs function to extract the **magnitude** of the frequency components. This allows you to see distinct pulses (desired signals) versus bands of dense frequency components (noise).

2. Interactive Design Using the Filter Designer Tool

MATLAB provides a dedicated **Filter Designer tool** (formerly FDATool) that allows for a graphical, "trial-and-error" approach to design.

- **Specifications:** You can define the filter using radio boxes and dropdown menus for the **response type** (low pass, band pass, etc.), **design method** (FIR or IIR), and **filter order**.
- **Visualizing Responses:** The tool allows you to observe various response types before applying the filter, including **magnitude response, phase response, impulse response, and pole-zero plots**.
- **Iterative Refinement:** If a simple low pass filter fails to remove all noise (such as low-frequency motor noise), you can quickly switch to a **band pass filter** and adjust the passband and stopband edges until the desired signal is isolated.

3. Programmatic Design and Application

Once you have determined your filter requirements, you can implement them via scripts:

- **Coefficient Calculation:** For FIR filters, the fir1 function is commonly used to design the impulse response based on the filter order and a **normalized cutoff frequency**. The cutoff frequency must be normalized to the **Nyquist frequency** (half the sampling frequency).

- **Applying the Filter:**
    - If you designed the filter in the Designer Tool, you can **export it as an object** to the workspace and use the filter function: y = filter(filterObject, inputSignal).
    - Alternatively, you can use the **convolution** technique in the time domain by convolving the noisy signal with the filter's impulse response using conv(signal, impulseResponse).

4. Optimization: FIR vs. IIR

A critical part of the design procedure is balancing performance with computational cost:

- **FIR Filters:** These are theoretically simpler and often preferred for their stability, but they may require a very **high filter order** (e.g., 300+) to achieve a sharp cutoff.
- **IIR Filters:** These are more complex but can achieve the same performance as an FIR filter with a significantly **lower order** (e.g., reducing an order from 304 to 20), which saves memory and hardware resources. The **Elliptic filter** is often a good choice for minimizing filter order.

5. Implementation in Simulink

For system-level simulation, filters can be integrated into Simulink models:

- **Digital Filter Design Block:** This block provides a GUI identical to the MATLAB Filter Designer tool, allowing you to design and update filter parameters directly within the model.
- **Exporting from MATLAB:** You can also use the **"Realize Model"** option in the Filter Designer tool to automatically generate a Simulink filter block from your MATLAB design.
- **Handling Vectors:** Because the FFT process requires a vector of information while Simulink typically runs on sample time (scalars), you must use a **buffer** and a **zero-order hold** to store signals into a vector before performing frequency analysis.

6. Code Generation and Deployment

Finally, MATLAB and Simulink allow you to deploy your designs to real-world applications. Within the Filter Designer tool, the **Targets pane** offers options to generate **C or HDL code** for various hardware targets, removing the need to manually write standalone code for filter coefficients.

Designing a digital filter in Simulink involves modeling the system's difference equation or transfer function using specialized blocks. While the provided manuals focus heavily on MATLAB code-based design, they highlight that **Simulink** is used for real-time realization and system modeling. Based on standard engineering practices and the signal processing principles described in the documents, the procedures are as follows:

1. Define Filter Specifications

Before opening Simulink, identify the parameters required for the filter design:

- **Filter Type:** Decide between **IIR** (Infinite Impulse Response) or **FIR** (Finite Impulse Response).
- **Response Type:** Low-pass, high-pass, band-pass, or band-stop.
- **Frequencies:** Define the pass-band and stop-band frequencies (f_p, f_s) and the sampling frequency (F_s).

- **Attenuation:** Determine the allowable pass-band ripple (R_p) and stop-band attenuation (R_s).

2. Design the Filter Coefficients (MATLAB Integration)

It is often efficient to calculate the filter coefficients in MATLAB first using functions like buttord (Butterworth order) or bilinear (Bilinear transformation).

- Example: Use [b, a] = butter(n, Wn) to generate the numerator (b) and denominator (a) coefficients.

3. Build the Simulink Model

1. **Open Simulink:** Type Simulink in the MATLAB Command Window and create a new model.
2. **Add Input Source:** From the "Sources" library, add a block like **Sine Wave** or **Random Number** to act as the input signal.
3. **Implement the Filter:**
    - **Discrete Filter Block:** Search for the "Discrete Filter" block in the Simulink library. Double-click it and enter your b and a coefficient in the numerator and denominator fields.
    - **Direct Realization:** Alternatively, you can build the filter manually using **Unit Delay** (z^{-1}), **Gain**, and **Sum** blocks to recreate the difference equation (e.g., y[n] = 0.2x[n] + 0.52y[n-1]).

+1

4. **Add Visualization:** From the "Sinks" library, add a **Scope** or **Spectrum Analyzer** to the output of the filter to view the signal in the time or frequency domain.

4. Configure and Run Simulation

1. **Solver Settings:** Go to "Modeling Settings" and ensure the solver is set to **Fixed-step** (standard for digital/discrete systems).
2. **Set Sample Time:** The sample time in your blocks must match the sampling period used in your design ($T_s = 1/F_s$).
3. **Run:** Click the "Run" button to execute the simulation.

5. Verification

Compare the Simulink output with your theoretical expectations or MATLAB plots (using freqz or zplane) to ensure the filter suppresses the target frequencies as intended.

A Step-by-Step Guide to Digital Filter Design in MATLAB

PHASE 1: PROBLEM SETUP

Define Sampling Parameters

Set the sampling frequency (F_s) at 500 Hz and the sampling period (T) at $1/F_s$ (2 ms) to satisfy the Shannon Sampling Theorem.

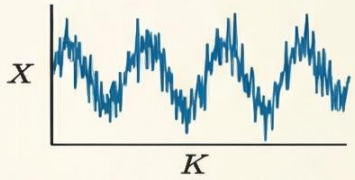
Construct the Noisy Signal

$$A\sin(2\pi \cdot 10 \cdot t) + \text{random noise (randn)}$$

The Shannon Constraint: Ensure F_s is at least twice the highest frequency component (10 Hz) to prevent aliasing.

PHASE 2: INITIAL ANALYSIS

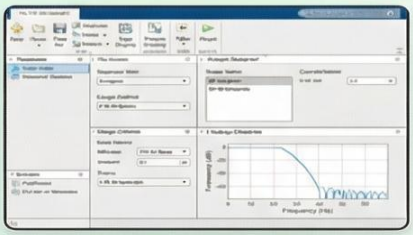
Visualize the Time Domain



Frequency Domain Transformation (FFT)

Identify Noise Grains: In the frequency spectrum, the "grains" between the main spikes represent the noise that needs to be filtered out.

PHASE 3: FILTER DESIGN VIA APP



Launch Filter Designer

Open the "Filter Designer" app from the MATLAB Apps tab to access the graphical design interface.

Select Response & Method

Choose a response type (e.g., Lowpass) and a design method (e.g., FIR Window/Rectangular or IIR Butterworth).

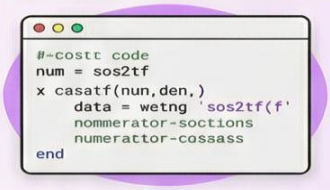
Set Cutoff Frequency

Define the cutoff frequency (F_c) near 10–12 Hz to preserve the 10 Hz signal while eliminating higher-frequency noise.

PHASE 4: IMPLEMENTATION & CODING

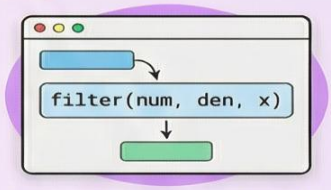
Export Coefficients

Export the design to the Workspace as Numerator (for FIR) or SOS Matrix/Scale Values (for IIR).



Convert IIR to Transfer Function

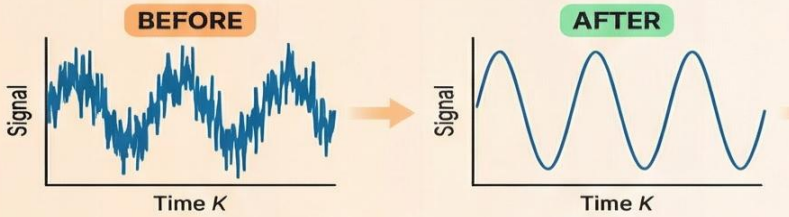
If using IIR, use the `sos2tf` function to convert second-order sections into numerator/denominator coefficients.



Create the System Model

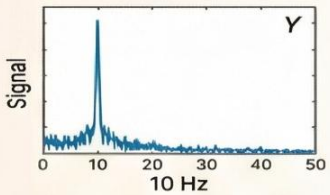
Use the `tf(num, den, T)` command to generate the digital transfer function of the filter.

PHASE 5: VERIFICATION & RESULTS



Compare Time/Frequency Plots

`filter(num, den, x)` generates output, *Y*



Confirm Signal Integrity

Check the frequency spectrum of the output to ensure the 10 Hz spike remains while the noise "grains" are removed.

Design and Implement a Filter

Design a digital low-pass filter using the Digital Filter Design block and incorporate it into your model to simulate the presence of low-frequency noise.

You can design lowpass, highpass, bandpass, and bandstop filters using the Digital Filter Design block. These blocks can compute filter coefficients for various filter structures. This topic uses the Digital Filter Design block to convert white noise to low-frequency noise so you can simulate its effect on your system.

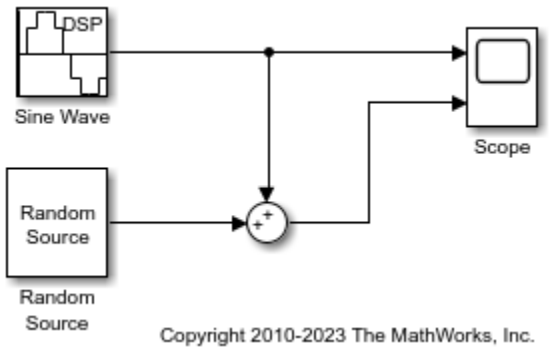
As a practical application, suppose a pilot is speaking into a microphone within the cockpit of an airplane. The noise of the wind passing over the fuselage is also reaching the microphone. A sensor is measuring the noise of the wind on the outside of the plane. You want to estimate the wind noise inside the cockpit and subtract it from the input to the microphone so that only the pilot's voice is transmitted.

In the first section of this topic, you learn how to model the low-frequency noise that is reaching the microphone. In the second section of the topic, you learn how to remove this noise so that only the pilot's voice is heard.

Design a Digital Filter in Simulink

In this section, you use a Digital Filter Design block to create low-frequency noise, which models the wind noise inside the cockpit.

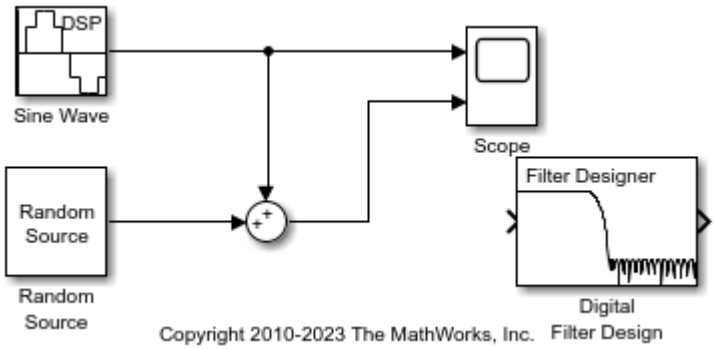
Open the ex_gstut3 model. This model contains a Scope block that displays the original sine wave and the sine wave with white noise added.



Open the DSP System Toolbox™ library by typing dsplib at the MATLAB® command prompt. Convert white noise to low-frequency noise by introducing a Digital Filter Design block into your model. In the airplane scenario, the air passing over the fuselage creates white noise that is measured by a sensor. The Random Source block models this noise. The fuselage of the airplane converts this white noise to low-frequency noise, a type of colored noise, which is heard inside the cockpit. This noise contains only certain frequencies and is more difficult to eliminate. In this example, you model the low-frequency noise using a Digital Filter Design block. This block uses the

functionality of the Filter Design and Analysis Tool (FDATool) to design a filter.

Double-click the Filtering library, and then double-click the Filter Implementations sublibrary. Click-and-drag the Digital Filter Design block into your model.

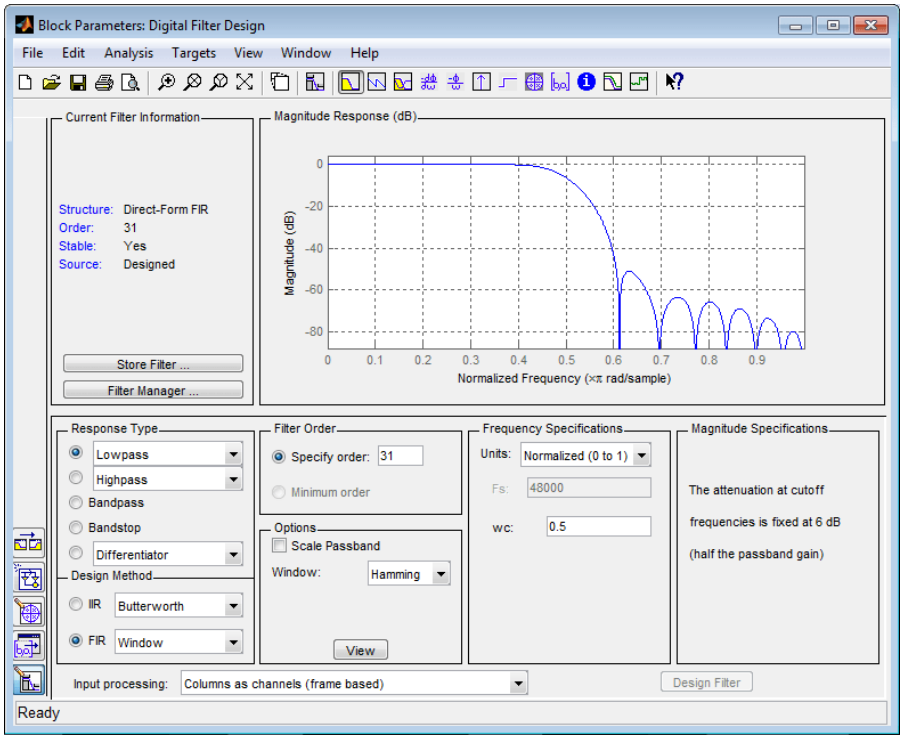


Set the Digital Filter Design block parameters to design a lowpass filter and create low-frequency noise. Open the block parameters dialog box by double-clicking the block. Set the parameters as follows:

- **Response Type** = Lowpass
- **Design Method** = **FIR**. From the list, choose Window.
- **Filter Order** = **Specify order** and enter 31.
- **Scale Passband** is cleared.
- **Window** = Hamming
- **Units** = Normalized (0 to 1)
- **wc** = 0.5

Based on these parameters, the Digital Filter Design block designs a lowpass FIR filter with 32 coefficients and a cutoff frequency of 0.5. The block multiplies the time-domain response of your filter by a 32 sample Hamming window.

Click **Design Filter** at the bottom center of the dialog box to view the magnitude response of your filter in the **Magnitude Response** pane. The Digital Filter Design dialog box should now look similar to the following figure.



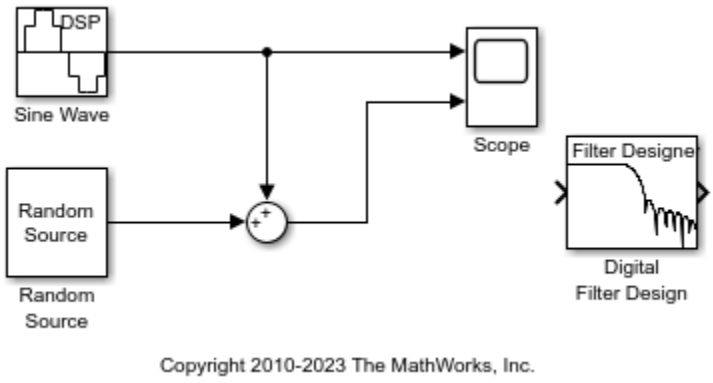
You have now designed a digital lowpass filter using the Digital Filter Design block.

You can experiment with the Digital Filter Design block in order to design a filter of your own. For more information on the block functionality, see the Digital Filter Design block reference page.

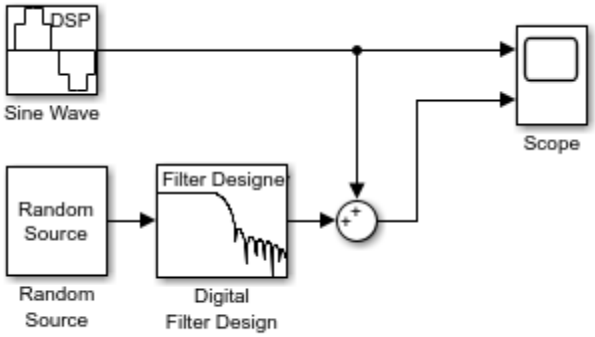
Add a Digital Filter to Your Model

In this section, you add the lowpass filter you designed in Design a Digital Filter in Simulink to your block diagram. Use this filter, which converts white noise to colored noise, to simulate the low frequency wind noise inside the cockpit:

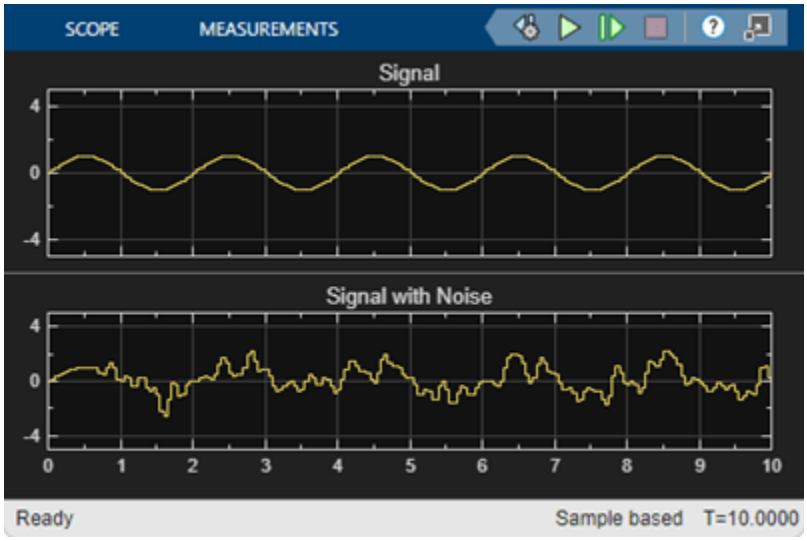
If the model you created in Design a Digital Filter in Simulink is not open on your desktop, open ex_gstut4 which is an equivalent model.



Incorporate the Digital Filter Design block into your block diagram by placing it between the Random Source block and the Sum block.

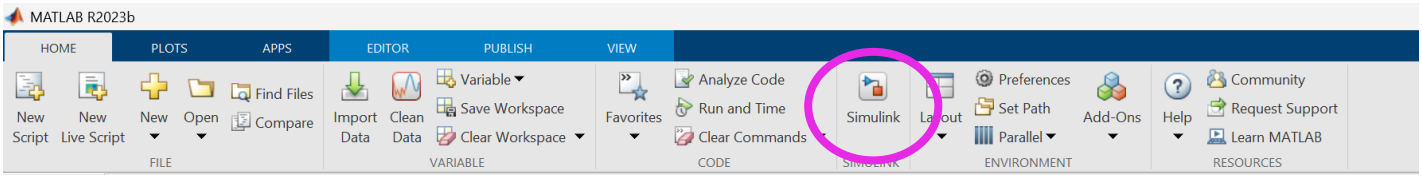


Run your model and view the results in the Scope window. This window shows the original input signal and the signal with low frequency noise added to it.

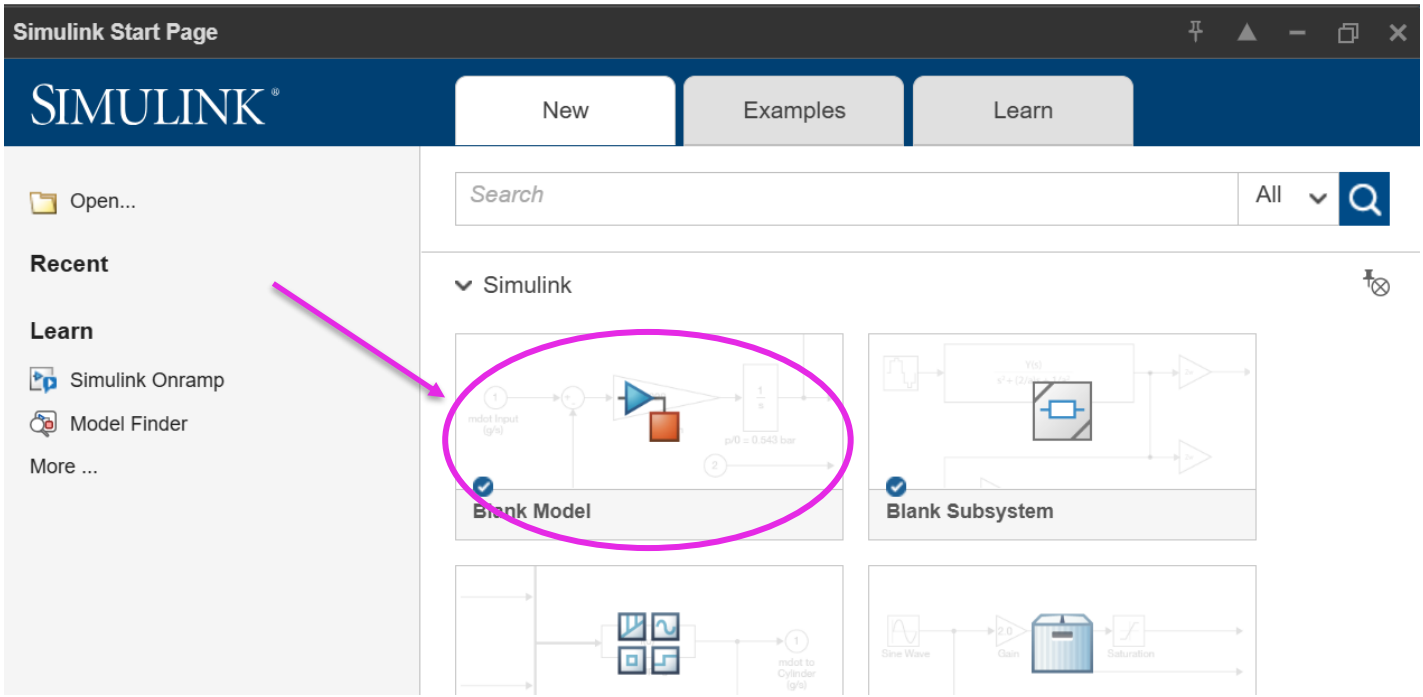


You have now built a digital filter and used it to model the presence of colored noise in your signal. This is analogous to modeling the low frequency noise reaching the microphone in the cockpit of the aircraft. Now that you have added noise to your system, you can experiment with methods to eliminate it.

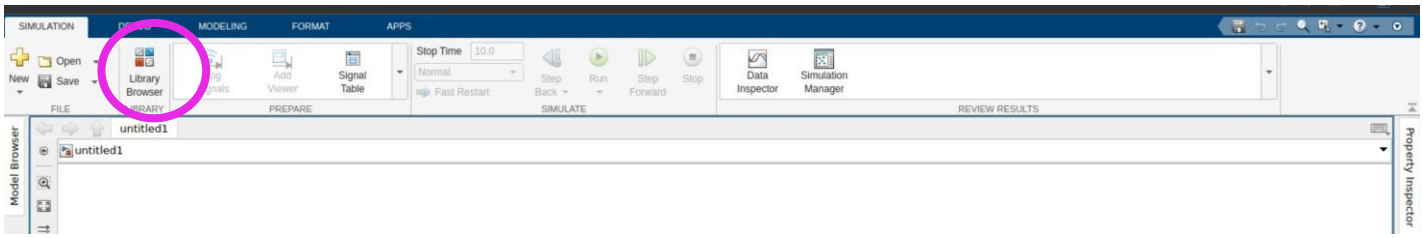
1. Open MATLAB
2. From home – click to Simulink



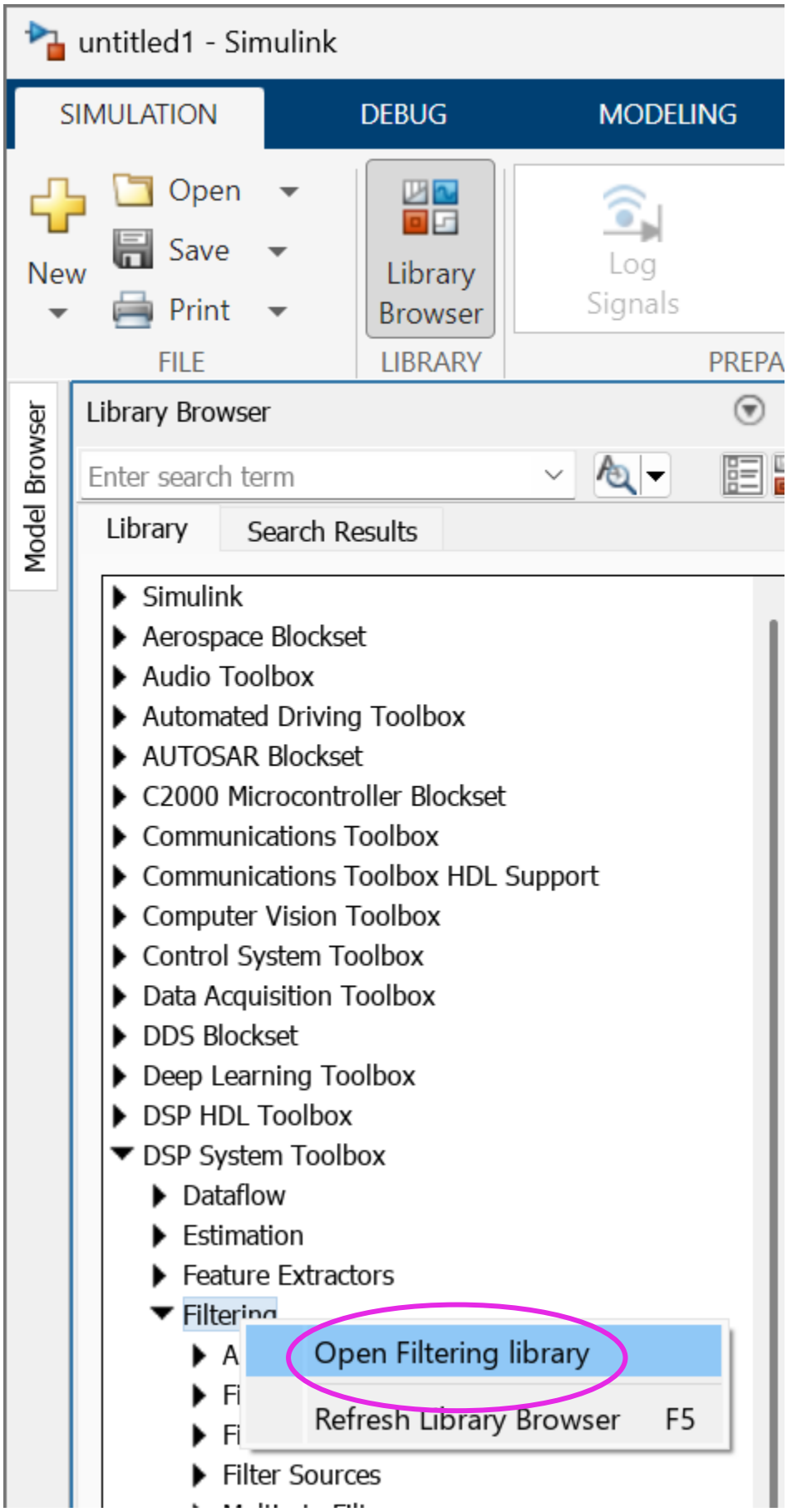
3. Blank model



4-



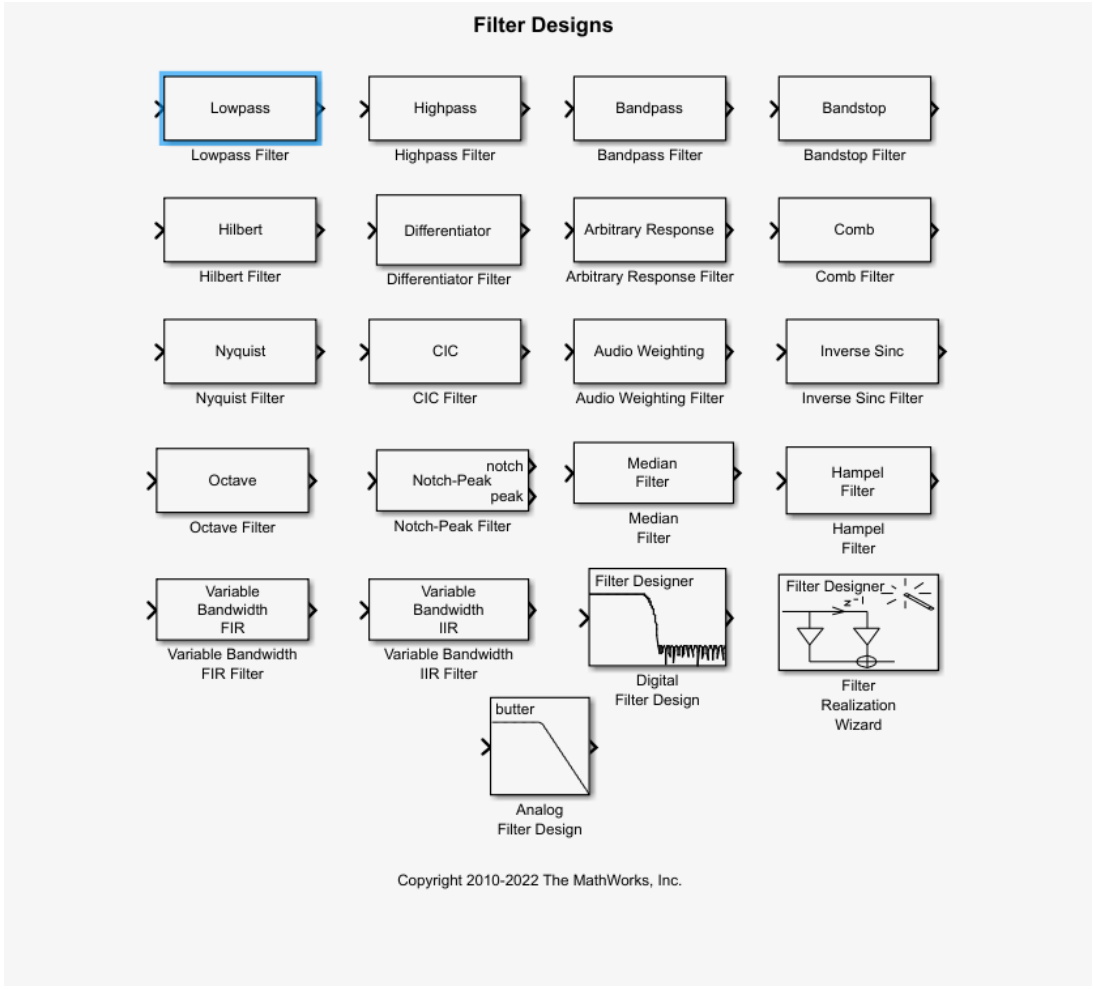
5-


untitled1 - Simulink
SIMULATION
DEBUG
MODELING
New
Open
Save
Print
FILE
Library Browser
LIBRARY
Log Signals
PREPA
Model Browser
Library Browser
Enter search term
Library
Search Results
Simulink
Aerospace Blockset
Audio Toolbox
Automated Driving Toolbox
AUTOSAR Blockset
C2000 Microcontroller Blockset
Communications Toolbox
Communications Toolbox HDL Support
Computer Vision Toolbox
Control System Toolbox
Data Acquisition Toolbox
DDS Blockset
Deep Learning Toolbox
DSP HDL Toolbox
DSP System Toolbox
Dataflow
Estimation
Feature Extractors
Filtering
Open Filtering library
Refresh Library Browser F5
Filter Sources

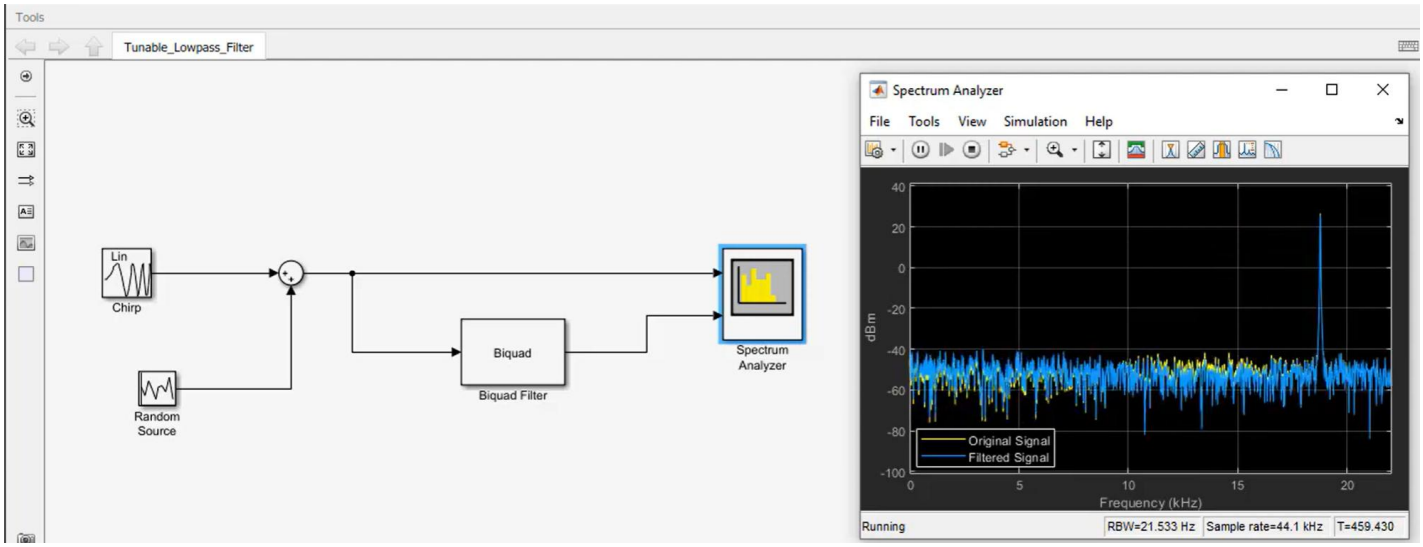
6


Library: dspfiltering - Simulink
LIBRARY
DEBUG
MODELING
FORMAT
APPS
New
Open
Save
Print
FILE
Library Browser
LIBRARY
Log Signals
Signal Table
Viewers Manager
PREPARE
Locked Library
Lock Links
PROTECT
Show Annotations in Library Browser
ANNOTATION
dspfiltering
dspfiltering
Filtering
Filter Implementations
Adaptive Filters
Multirate Filters
Filter Designs
Filter Sources

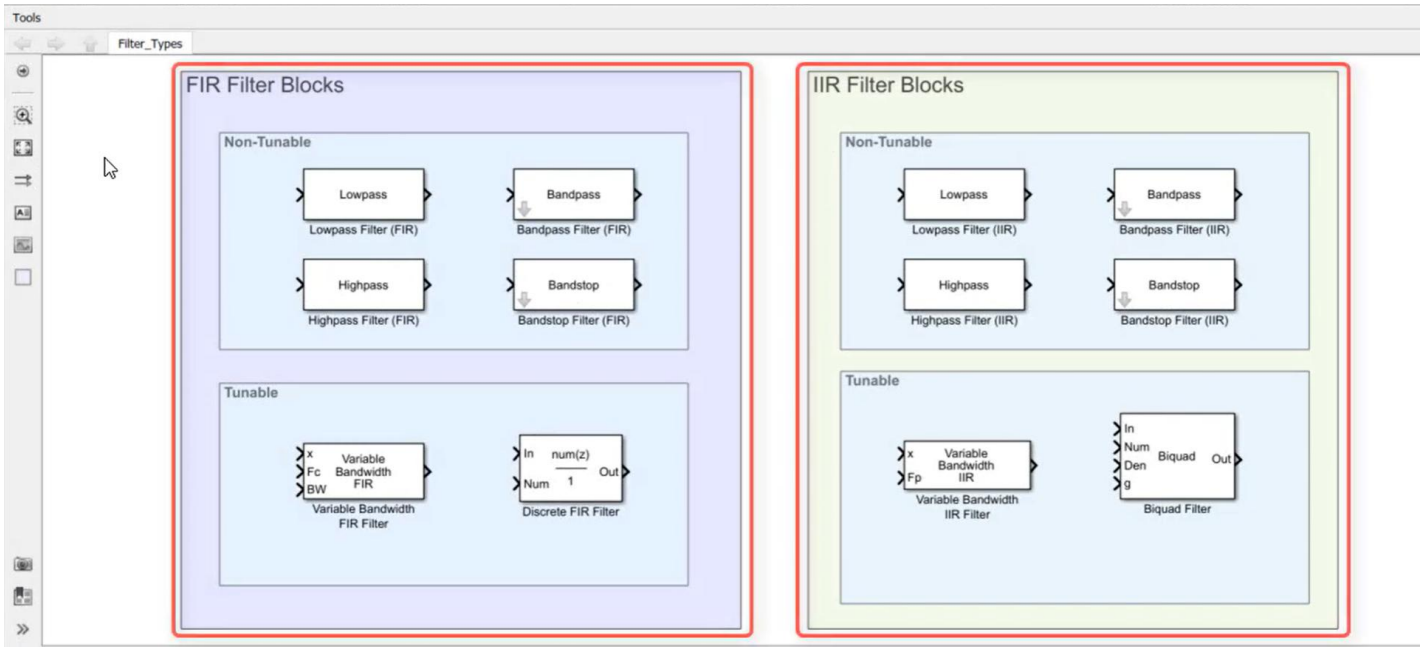
7


Filter Designs
Lowpass
Lowpass Filter
Highpass
Highpass Filter
Bandpass
Bandpass Filter
Bandstop
Bandstop Filter
Hilbert
Hilbert Filter
Differentiator
Differentiator Filter
Arbitrary Response
Arbitrary Response Filter
Comb
Comb Filter
Nyquist
Nyquist Filter
CIC
CIC Filter
Audio Weighting
Audio Weighting Filter
Inverse Sinc
Inverse Sinc Filter
Octave
Octave Filter
notch
Notch-Peak
peak
Notch-Peak Filter
Median Filter
Median Filter
Hampel Filter
Hampel Filter
Variable Bandwidth FIR
Variable Bandwidth FIR Filter
Variable Bandwidth IIR
Variable Bandwidth IIR Filter
Filter Designer
Digital Filter Design
Filter Designer
Filter Realization Wizard
butter
Analog Filter Design
Copyright 2010-2022 The MathWorks, Inc.

8



9



10– noice cancele

Non-Tunable Bandpass Filtering of Noisy Sine Wave