Lab 1: Implementing an IIR Filter

Contents

Lab Materials/Downloads

For this lab, you can optionally use the following downloadable items.

1. FilterFramework.slx: Simulink model containing easy access to some Simulink blocks for this lab. If you are using an older version of Matlab (at home), then you can instead use FilterFrameworkOlderVersionsMatlab.mdl. These Simulink files contain the Simulink blocks used in this lab and reduce the need to hunt through the Simulink `Library Browser` to find the blocks.

Lab 1 Overview

This course will use three main tools to explore and learn about digital signal processing (DSP).

1. Matlab - a high-level computation and programming environment that will allow DSP systems to be designed, prototyped, and tested.
2. Simulink - graphical blocks on top of Matlab that allow block-based DSP system design and simulation.
3. ST32H735G-DK Discovery Kit - microcontroller board with analog-to-digital converters

Skip to main content

implementations of DSP algorithms in C. Basic knowledge of C programming is a prerequisite.

The goal of the first few labs is to become familiar with the tools and to do so you will implement an example filter in each of these enviroments so that you can see how a simple DSP system could be implemented and to begin to see how these three tools can be used for DSP systems. Python would be an interesting alternative that may be used in future offerings of this course. These tools will be used throughout many of the labs and assignments in this course.

For this first lab, you will start to become familiar with Matlab and Simulink for DSP. You will use the STM32H735G-DK boards in the next lab.

Here are three specific goals for learning Matlab in this lab:

1. Learn how an example signal can be setup in Matlab.
2. Implement an example filter in Matlab.
3. See how filtering can be done and the results displayed.

Here are three specific goals for learning Simulink in this lab:

1. Learn how Simulink models can be built for performing DSP.
2. Setup Simulink to simulate a discrete-time system that includes a filter.
3. See how to interface Simulink with Matlab to perform some basic filtering operations.

Pre-Lab: Matlab Primer

Before getting into the implementation of a DSP filter for this lab, a Matlab Primer is provided to give you useful background for using Matlab for this course. Go through the Matlab Primer to ensure you have some of the basic knowledge of the environment.

Filter Background

The primary goal of this lab is to familiarize yourself with some of the tools that will be used in this course. Towards that goal, in this lab you are to do the following.

- Implement a filter in Matlab.

Skip to main content

Since this is a first lab, the details of the design and theory of filters will come later in the course. For this lab, you will go through the steps of taking an already designed filter and see a few ways in which this filter can be realized on different platforms. While this process should give you some observable and understandable results, a deeper understanding of the details of filters, signal processing, and signal analysis won't come until later in the course.

The filter considered for this lab is the one introduced during the first lecture on Slide 1.15 and Slide 1.16 of the lecture notes. That filter is a 4^{th}-order infinite impulse response (IIR) filter as shown in Fig. 20 (see Slide 1.15).

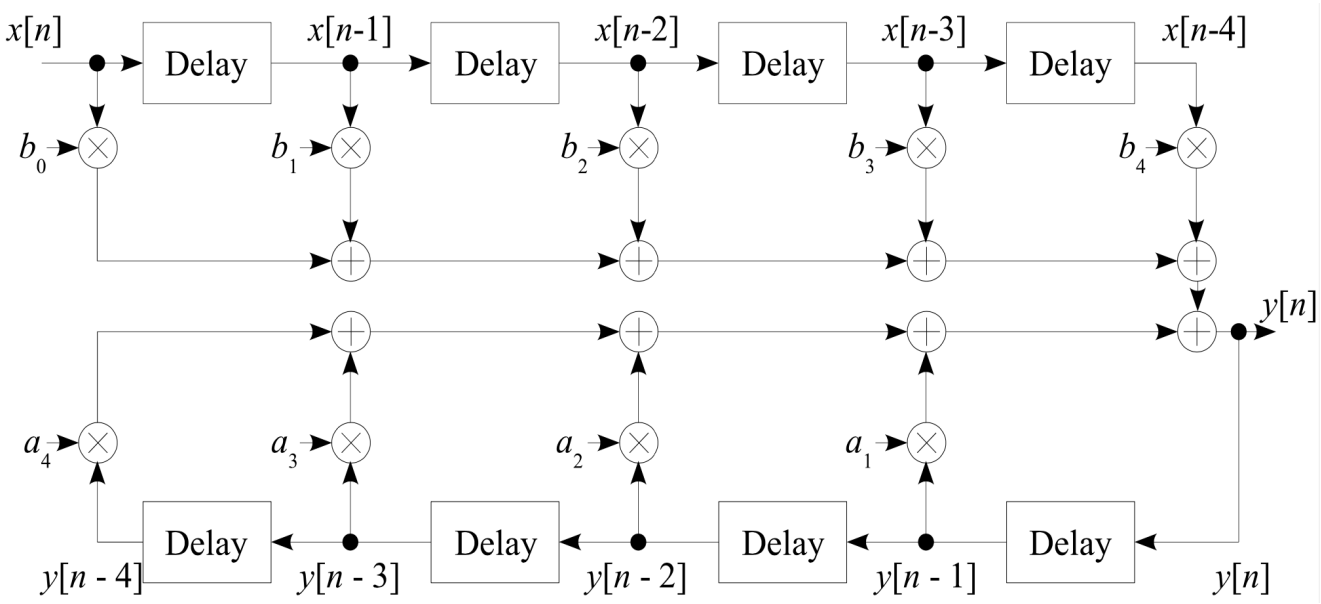

Fig. 20 4^{th}-order IIR filter structure.

Note

Don't worry yet if you do not comprehend what an IIR filter is and what you can do with such a filter. Those details will be learned as the course progresses. For now, the goal is to become familiar with some aspects of the lab environment and the theory will be discussed later in the course.

This IIR filter structure is a realization of the difference equation

$$y[n] = \sum_{} a_k y[n-k] + \sum_{}^{4} b_k x[n-k] \tag{1}$$

Skip to main content

where $x[n]$ is the input sequence to the filter, $y[n]$ is the output sequence of the filter, and b_k and a_k are sets of filter coefficients. You should be able to compare this difference equation with what is shown in Fig. 20 to see how the two represent the same concept.

> **Warning**
>
> If equations are not showing properly, sometimes you need to refresh the webpage. The Mathematical Markup Language (MathML) used contacts an external server that isn't always quick to respond.

While a number of different types of filters could be implemented, the goal for the filter you will implement is for a discrete-time lowpass filter with a frequency response as shown in Fig. 21 designed for a signal sampling rate of $f_s = 16$ kHz. We can see from this magnitude of the frequency response that frequencies below ~2 kHz should be passed nearly unchanged (multiplied by ~1) while frequencies greater than ~3 kHz should be attenuated significantly (multiplied by ~0). Matlab code is also provided for how this magnitude frequency response is generated. More will be learned on that later.

Matlab plot | Matlab code

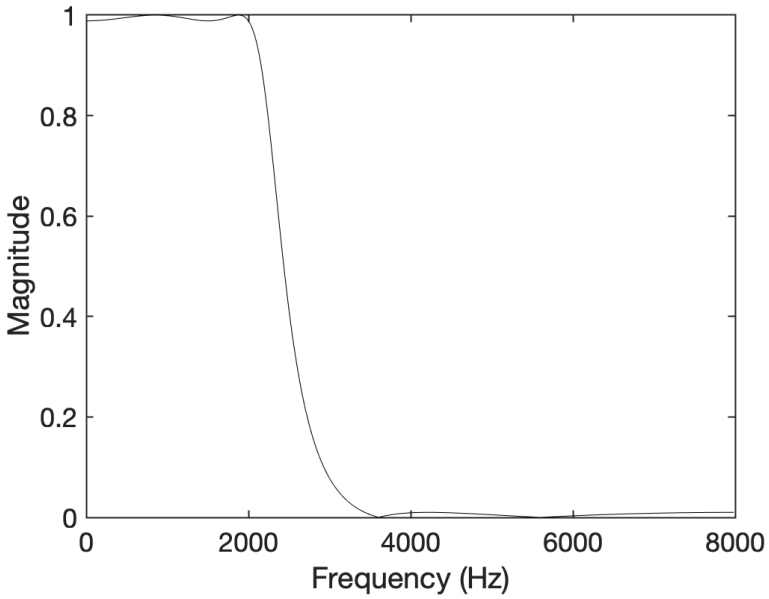

Fig. 21 Magnitude of the frequency response for a given lowpass filter for a sampling rate of $f_s = 16000$ Hz.

Skip to main content

The filter in Fig. 21 can be implemented using the following filter coefficients as given on Slide 1.16.

$$\begin{aligned} b_0 &= +0.0355267 \\ b_1 &= +0.0306650 \\ b_2 &= +0.0581951 \\ b_3 &= +0.0306650 \\ b_4 &= +0.0355267 \\ a_1 &= +2.1485451 \\ a_2 &= -2.2390905 \\ a_3 &= +1.1509858 \\ a_4 &= -0.2532257 \end{aligned} \tag{2}$$

> **Note**
>
> Do not worry if you don't understand what these numbers mean at the moment. Again, details of how to *design* your own filters and determine your own filter coefficients will be discussed later in the course. For now, simply accept that these filter coefficients will form the required lowpass filter as shown in Fig. 21. These numbers appear in the Matlab code used for generating the magnitude frequency response in Fig. 21.

Filtering with Matlab/Simulink

This portion of the lab provides some examples of implementing a filter in Matlab and Simulink. Since this is a first lab, the details of the design and theory of filters will come later in the course. In this section, you will implement a 4^{th}-order infinite impulse response (IIR) filter as shown in Fig. 20. This IIR filter is also given in the lecture notes on Slide 1.15 which with the proper selection of filter coefficients was made into a lowpass filter as given in Slide 1.16.

Generating an Example Noisy Sine Wave Signal

Before proceeding, let's first generate a simple signal to act as the input to the lowpass filter. While we could use a recorded signal from a real-world source, for this section let's keep it

Skip to main content

simple and generate a 150 Hz sine wave with a number of interfering signals superposed with the sine wave.

We will generate a *noisy* sine wave similar to that on Slide 1.16 of the lecture slides and discussed during the first lecture. An example input signal can be generated with the following Matlab code to form the input vector `sigsum`.

```
fs = 16000;                     % sampling frequency
t = 0:1/fs:0.02;                % start : time step : end
s1 = 1.00 * sin(2*pi*150 * t);  % desired 150 Hz sine wave
s2 = 0.05 * sin(2*pi*3304 * t); % interfering noise (3304 Hz)
s3 = 0.02 * sin(2*pi*4000 * t); % more interfering noise (4000 Hz)
s4 = 0.04 * sin(2*pi*5722 * t); % more interfering noise (5722 Hz)
s5 = 0.07 * sin(2*pi*7302 * t); % more interfering noise (7302 Hz)
sigsum = s1 + s2 + s3 + s4 + s5; % final superposed noisy signal
```

This Matlab code sets the sampling frequency to $f_s = 16,000$ Hz and performs sampling over the time period from 0 secs to 0.02 secs, for 321 samples in total (use `whos` to see the vector sizes). Notice that the vector `sigsum` is made up of a number of sinusoids of different amplitudes and frequencies. You can use the `plot(t, sigsum)` command to verify that you have generated a noisy sine wave similar to Slide 16.1.

```
plot(t, sigsum)
```

Performing Spectral Analysis on the Signal in Matlab

We will now take the signal `sigsum` from the previous section and perform frequency spectral analysis on the signal to determine its frequency content. Many ways exist to compute a frequency spectrum of a signal, and not all of them are equivalent, so it takes some explanation to understand the differences (which we won't fully cover in this introductory lab).

One approach to looking at the frequency spectrum of a signal is to calculate the discrete Fourier transform of the signal using the fast Fourier transform (FFT). A quick plot of the magnitude of the frequency spectrum of `sigsum` can be done as follows.

```
plot(abs(fft(sigsum)))
```

Skip to main content

Missing in this simple plot is proper labeling of the axes, the sampling frequency f_s is not incorporated, and we often prefer to look at the spectrum shifted and centred around the zero frequency.

> **Note**
>
> The FFT will give the positive frequencies in the first half (on the left) with zero (DC) on the far left and the negative frequencies in the second half (on the right). To put the zero frequency in the middle, we can use Matlab's `fftshift()` function as follows.
>
> ```
> plot(abs(fftshift(fft(sigsum))))
> ```

The following Matlab function includes labeling and shifting to make the plot more readily understandable. Save this function into a file named `plotspectrum.m`. This function will be useful for other labs and assignments in this course, so it is recommended that you keep it for later.

```
function plotspectrum(x,fs)
% This function plots the magnitude frequency spectrum of x.
%     x: data
%     fs: sampling rate (samples/second)

y = fft(x);                     % Perform a fast Fourier transform
n = length(y);
freqrange = (0:n-1)*fs/n-fs/2;  % Determine frequency index in Hz

if mod(n,2) ~= 0.     % check if n is odd
   % n is odd, so we must shift each index by half a spacing to
   % have zero lined up
   freqrange = freqrange + fs/(2*n);
end

plot(freqrange, fftshift(abs(y))/n);  % Plot the analog frequency spectrum
title('Spectrum of signal');
xlabel('Frequency (Hertz)');
ylabel('Magnitude');
```

You can then use this function to plot the spectrum of `sigsum` using

```
plotspectrum(sigsum,fs)
```

Skip to main content

> **Warning**
>
> To run any function, such as the `plotspectrum()` function that you saved in `plotspectrum.m`, the path for your Matlab **Command Window** should be the same as the function. Alternatively, use `addpath` to add the path where `plotspectrum.m` resides.

Other ways of determining a power spectral density (PSD) estimate (i.e., a frequency spectrum) for a signal is to use the `pspectrum`, `periodogram`, or `pwelch` functions in Matlab as follows.

```
figure(1); pspectrum(sigsum,fs)
figure(2); periodogram(sigsum)
figure(3); pwelch(sigsum)
```

The primary difference with these functions compared to the `plotspectrum()` function can be given as follows.

1. The power spectrum (*i.e.*, magnitude-squared) is shown instead of the magnitude spectrum
2. The plot is in decibels (*i.e.*, $decibels = 20\log_{10}(magnitude)$).
3. During computation, the signal is handled differently, such as partitioned into smaller pieces, the spectrum calculated for each piece, and then the results averaged. These extra steps can help when there is noise, such as additive white Gaussian noise, or in handling other issues, like statistical consisency in the power spectral density estimate.

These functions have numerous optional arguments that can be used to adjust the way that the power spectral density estimate is calculated for the spectrum.

> **Note**
>
> For `periodogram` and `pwelch`, the frequency axis is labeled as **normalized frequency** instead of in Hz. We will discuss what is meant by normalized frequency later in the course, but in brief the range of 0 to 8 kHz is normalized as 0 to π radians/sample.

Example Filter in Matlab

Once we have a signal in Matlab, filtering the signal can be done easily. Matlab is a good

Skip to main content

implementation, such as on the STM32H735G-DK board.

You can set up a filter in Matlab by first setting the coefficients used for the filter that were given in Equation (2) (repeated below for convenience) using the following Matlab code.

$$
\begin{aligned}
b_0 &= +0.0355267 \\
b_1 &= +0.0306650 \\
b_2 &= +0.0581951 \\
b_3 &= +0.0306650 \\
b_4 &= +0.0355267 \\
a_1 &= +2.1485451 \\
a_2 &= -2.2390905 \\
a_3 &= +1.1509858 \\
a_4 &= -0.2532257
\end{aligned}
$$

```
% Define our lowpass filter
format long
b = [0.0355267, 0.0306650, 0.0581951, 0.0306650, 0.0355267];
a = [1, -2.1485451, 2.2390905, -1.1509858, 0.2532257];
```

Skip to main content

Warning

While we haven't yet learned about what the numbers in `b` and `a` do, you may have noticed that the numbers in the Matlab vector `a` have the opposite sign as compared to Equation (2) and that there is also a `1` at the beginning of the Matlab vector `a`.

The values for `b` and `a` in Matlab come from a slightly rearranged version of Equation (1) (repeated here for convenience)

$$y[n] = \sum_{k=1}^{4} a_k y[n-k] + \sum_{k=0}^{4} b_k x[n-k]$$

written instead as

$$a_0 y[n] + \sum_{k=1}^{4} (-a_k) y[n-k] = \sum_{k=0}^{4} b_k x[n-k] \tag{3}$$

with $a_0 = 1$. In this form, $-a_k$ is used instead of a_k in the Matlab vector `a`. It can be a bit confusing for when to use a_k and when to use $-a_k$, but $-a_k$ is always used for Matlab's built-in functions like `filter()`, `freqz()`, etc. when the Matlab vectors `b` and `a` vectors are specified. So, the Matlab vectors are always formed as follows.

```
b = [b0, b1, b2, b3, ...];
a = [a0, -a1, -a2, -a3, ...];
```

With Matlab vectors `b` and `a` defined, we should double check that these filter coefficients define the desired lowpass filter. You can plot the frequency response of the filter in Matlab using the `freqz()` function as follows.

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

This command will plot the magnitude (in dB) and phase of the frequency response. If you wish more control over the plot, you can use Matlab code like the following.

```
w = 0:pi/10000:pi;          % Define frequency range
```

Skip to main content

```
plot(W*fs/(2*pi),abs(H));  % Plot magnitude of frequency response
title('Magnitude of Frequency Response');
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude of H');
```

Double check frequency response

Double check that the values for `b` and `a` give a magnitude of the frequency response that matches Fig. 21.

With the filter coefficients `b` and `a` defined, we can now filter `sigsum` using Matlab's `filter()` function and plot the results as follows.

```
% Filter the signal sigsum
y = filter(b, a, sigsum);       % filter sigsum

plot(t, sigsum, 'b'); hold on;  % plot sigsum in blue
plot(t, y, 'r'); hold off;      % plot y in red
ylabel('Amplitude'); xlabel('Time (secs)'); grid on;
title('Plot of sigsum (blue) and lowpass filtered output (red)');
```

Lab Submission Considerations

1. Plot, display, and save the plot for the input and the output of the filter. See Saving Matlab Figures on how to save Matlab figures.
2. What are your main observations for the filtered output signal versus the input signal?
3. Plot and save the frequency spectrum for the input signal as well as the filtered output signal. Compare the two spectra and make observations for the results of the filtering.

Using Simulink for DSP

In Simulink, we can perform similar processing as we did in Matlab, but instead by building a system with blocks. To start Simulink, type the following in the Matlab **Command Window**.

Skip to main content

Simulink allows for a number of different configurations of models. To create a blank model that is configured with typical settings for a DSP system, select **DSP System Toolbox → DSP System** from list on the right of the **Simulink Start Page** window. This **DSP System** configuration will set Simulink's solver options for a discrete-time system instead of the default continuous-time solver options (such as useful for SYSC 3600, SYSC 3610, SYSC 3500, or SYSC 3501). Some of the configuration options are as follows (see under **Modeling → Model Settings** in the toolbar ribbon at the top):

1. **Type:** fixed-step
2. **Solver:** discrete (no continuous states)
3. **Fixed step size (fundamental sample time):** auto

For most DSP models, you would use these settings.

For this lab, you can use the FilterFramework.slx model on the course webpage to copy-and-paste blocks into your model. The incomplete model in FilterFramework.slx has some of the blocks required for the filter implementation, which will save you time instead of having to search through the `Library Browser` to find the needed blocks.

> **Tip**
>
> If you wish to look for blocks yourself, you can find all of the Simulink blocks under **Simulation → Library Browser** as shown in Fig. 22 and then drag-and-drop them into your model. There are many Simulink blocks, so it may be easier to enter text in the search bar to find the blocks you need.

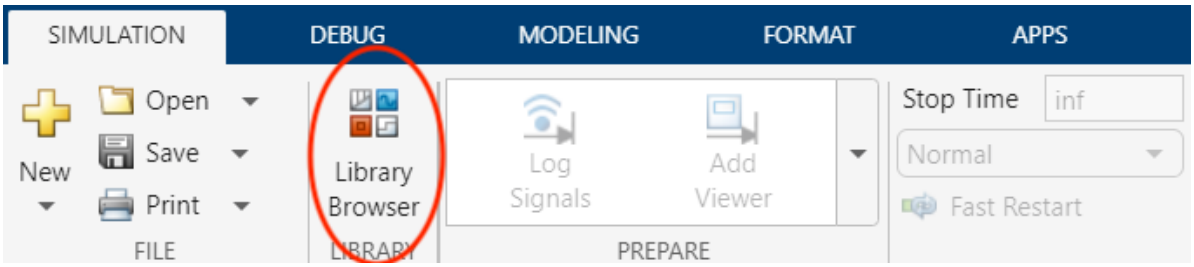

Fig. 22 Location of the Library Browser button.

Skip to main content

> **Note**
>
> It is assumed that you already have some exposure to Simulink through previous courses. Full documentation for Simulink can be found at
>
> - http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/

Using Simulink for Spectral Analysis

Similar to Matlab, you can also perform spectral analysis of a signal in Simulink. A few approaches exist, which include the `Spectrum Analyzer` under the **DSP System Toolbox → Sinks** section. There are also a number of spectral estimation techniques listed under the **DSP System Toolbox → Estimation → Power Spectrum Estimation** section of the **DSP System Toolbox** section.

Within Simulink, construct the model as shown in Fig. 23. The `Signal From Workspace` block can be found under the **DSP System Toolbox → Sources** section of the **Library Browser** and the other two blocks are found under the **DSP System Toolbox → Sinks** section of the **Library Browser**.

Skip to main content

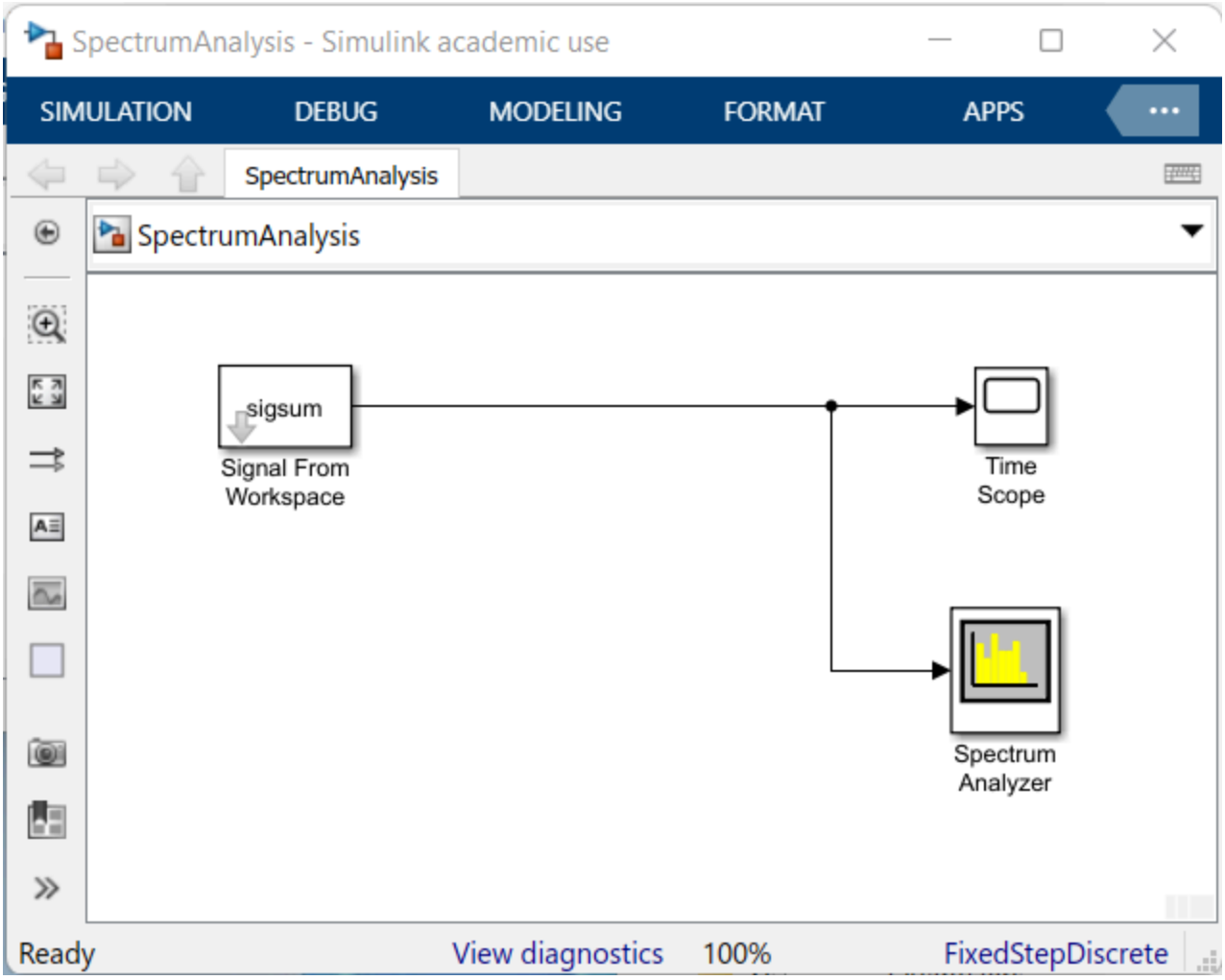

Fig. 23 Simulink model for the spectrum analysis of the noisy sine wave `sigsum`.

> **Warning**
>
> As only recently learned, the `Spectrum Analyzer` block in Matlab R2023a and Matlab R2023b has been updated significantly from previous versions of Matlab. Unfortunately, there appears to be a memory leak with this updated `Spectrum Analyzer` block that sometimes crashes Matlab. If this is happening for you, an alternative is shown in Fig. 24 where the `Spectrum Analyzer` block is replaced with a `To Workspace` block. The `To Workspace` block can be found in the **Library Browser** under **DSP System Toolbox → Sinks**. In this example use of the `To Workspace` block, after simulating the system in Simulink, you can switch back to the Matlab **Command Window** and use the command `pspectrum(out.yout,fs)` to view the spectrum (or similarly, `plotspectrum()`, `periodogram()`, or `pwelch()` as discussed earlier).

Skip to main content

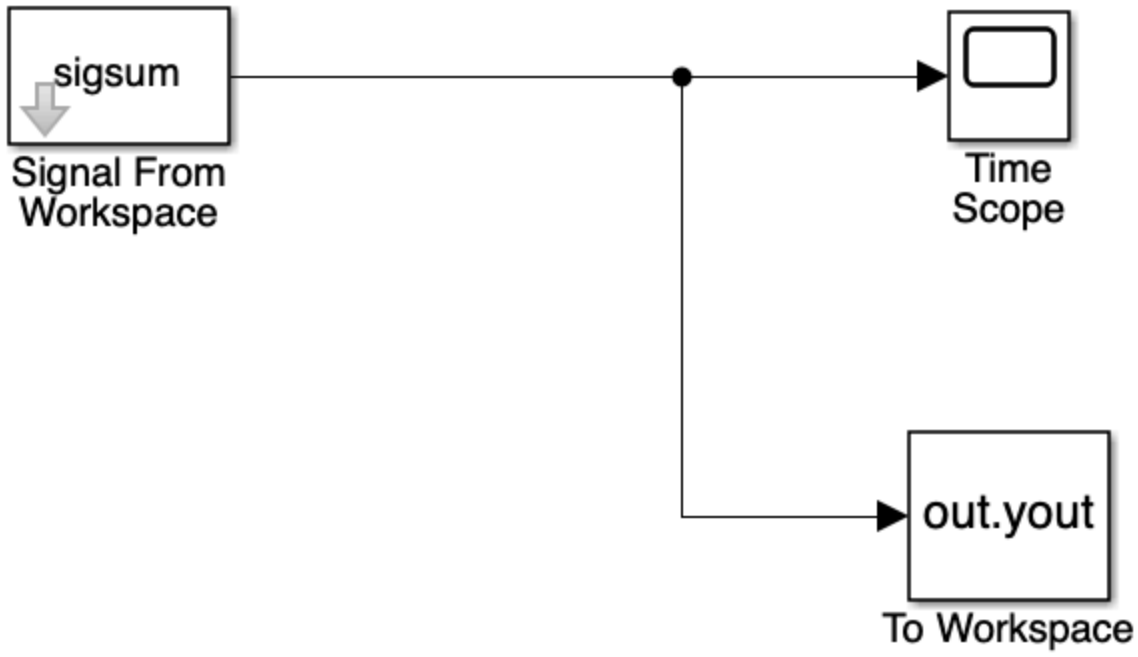

Fig. 24 Simulink model if the spectrum analysis is causing Matlab/Simulink to crash. This model replaces the `Spectrum Analyzer` block with a `To Workspace` block available in the **Library Browser** under **DSP System Toolbox → Sinks**.

Once you have implemented Fig. 23, you need to configure the blocks as follows.

- Double-click on the `Signal From Workspace` block and, following Fig. 25, set the input `Signal` to the input vector `sigsum` defined in Matlab, set the `Sample Time` to `1/fs`, and set the `Form output after final data value by` to `Cycle repetition`. These options will set the block to use the noisy sine wave and the sampling rate defined. You should have `sigsum` and `fs` still defined within Matlab from the previous sections.

Skip to main content

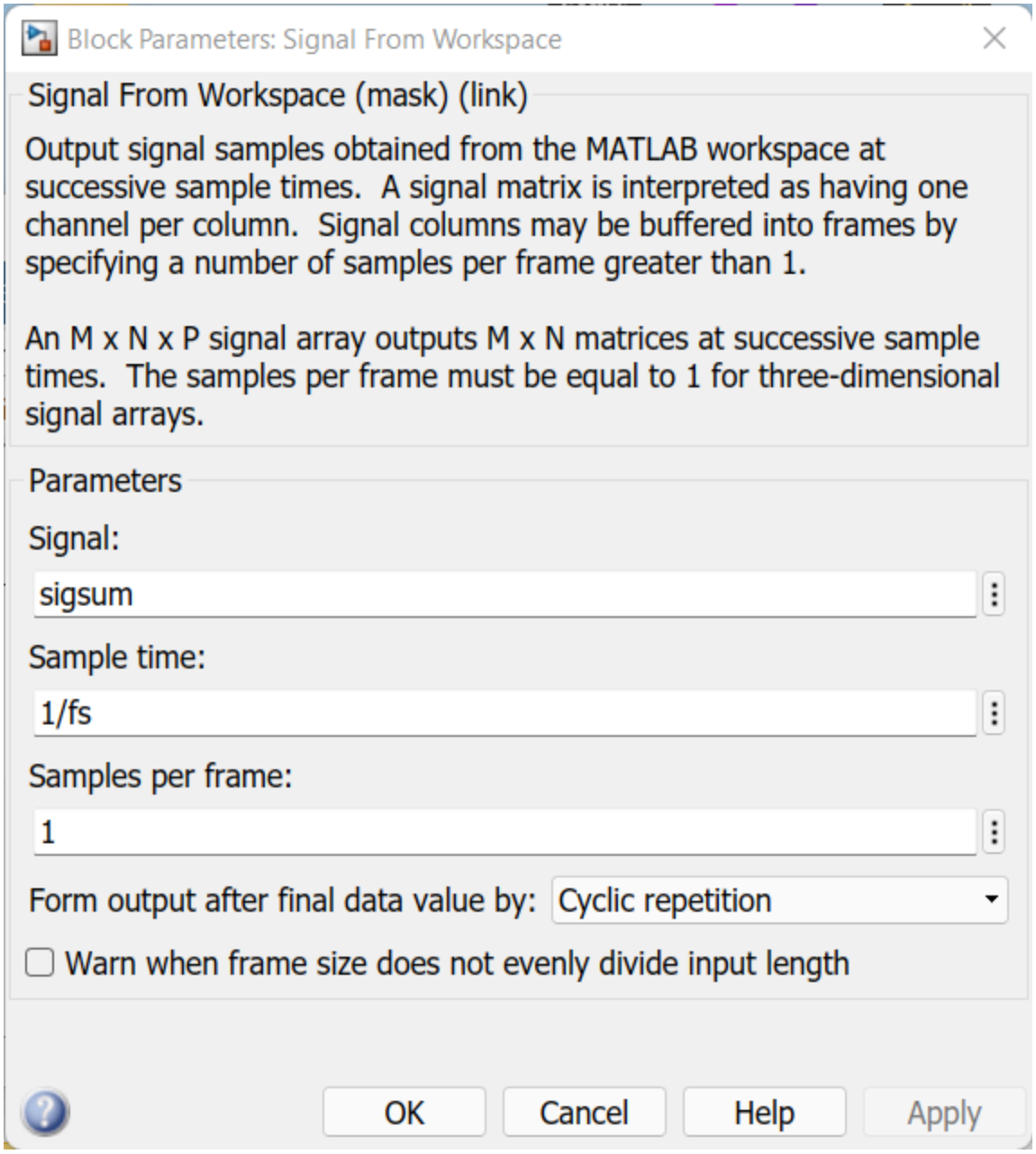

Fig. 25 Parameters for the `Signal From Workspace` block for `sigsum`.

- Double-click on the `Spectrum Analyzer` block and select **Analyzer → Spectrum → Power Density** so that the spectral analysis method is similar to those that you computed in Matlab. The rest of the default spectrum settings are sufficient for the purpose of this lab, but recognize that you could adjust some aspects of how the spectrum analyzer works.

> **Note**
>
> This `Spectrum Analyzer` is for Matlab R2023a in the lab. If you are using an older version of Matlab, then the spectrum analyzer will look a little different.

Skip to main content

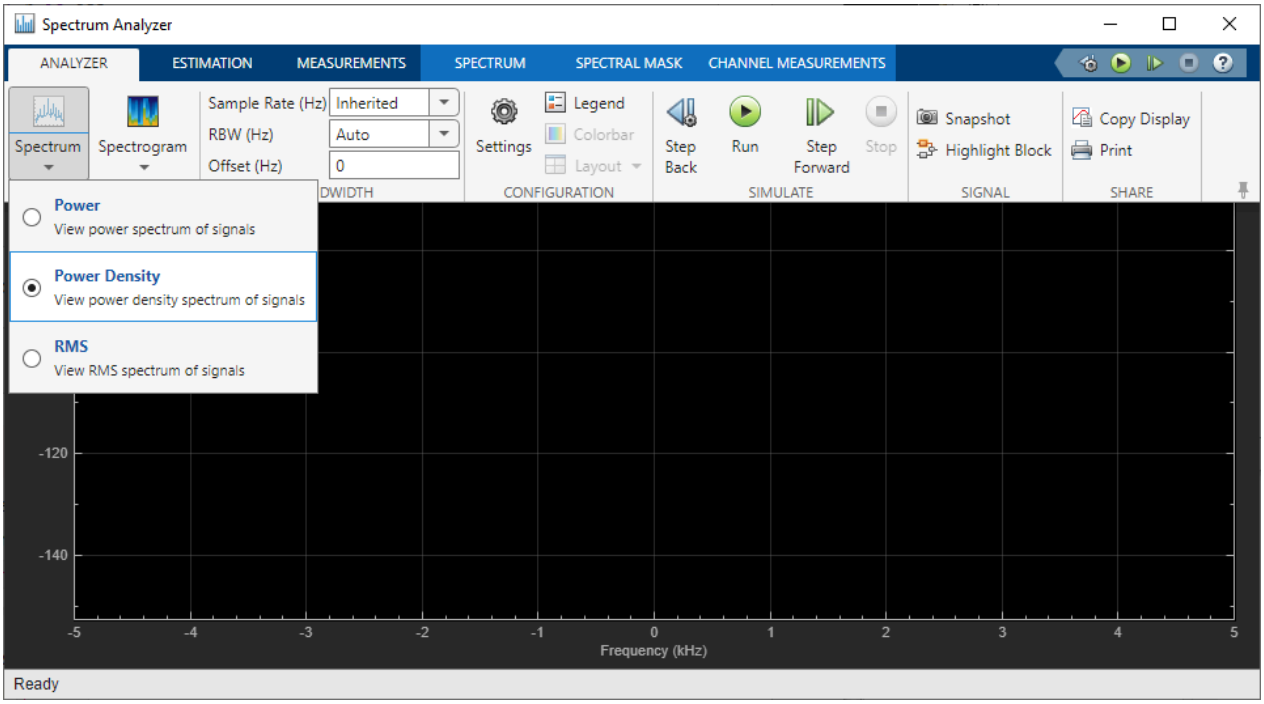

Fig. 26 `Spectrum Analyzer` block in Matlab R2023a.

- Double-click on the `Time Scope` to open a virtual oscilloscope as shown in Fig. 27.

Skip to main content

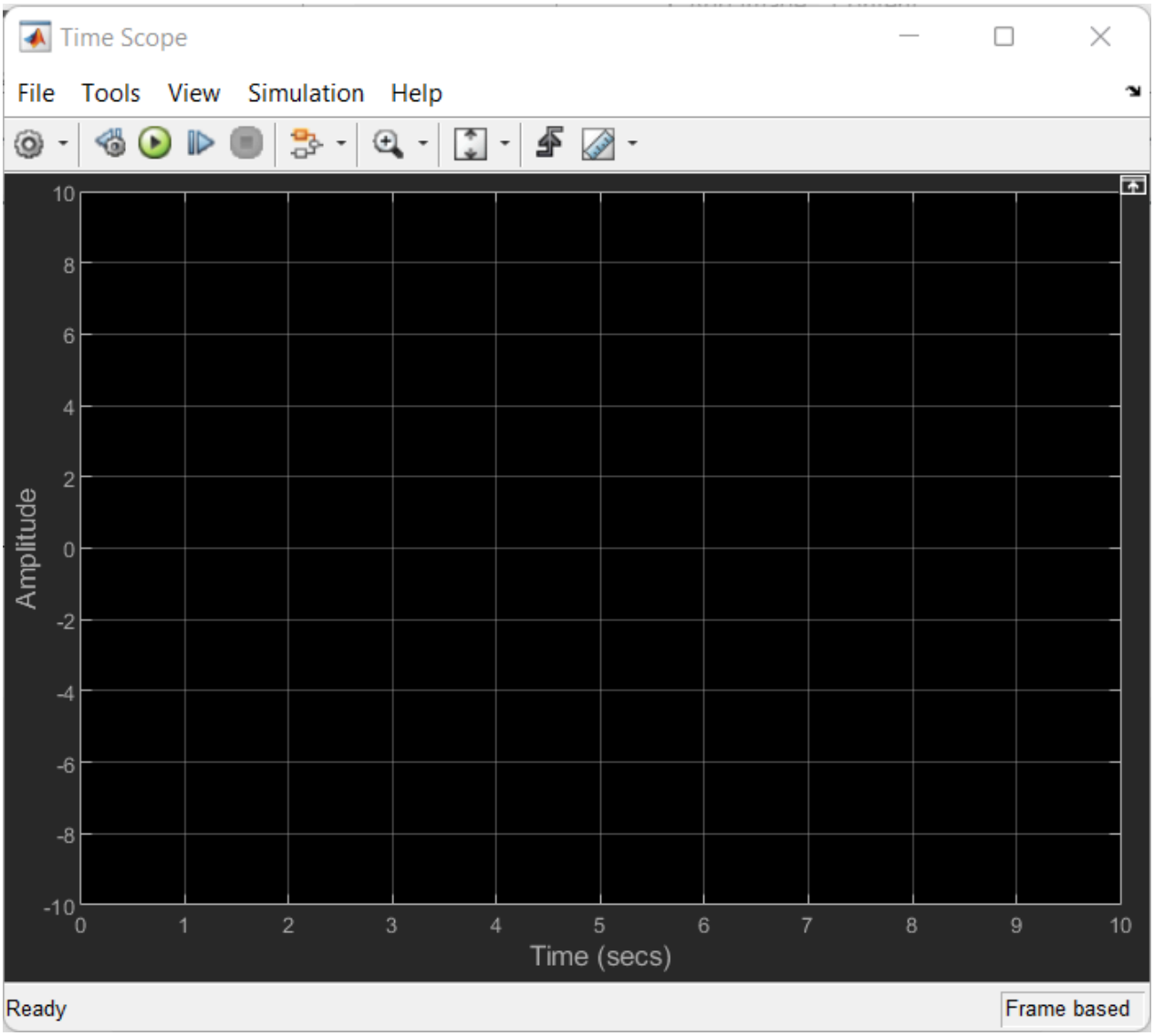

Fig. 27 Window for `Time Scope`.

Once everything is configured in your model, before simulating it change the `Stop Time` to 0.2 seconds (instead of `inf`) as shown in Fig. 28. Then, you can click the green `Run` button and then the `Stop` button, which appears under the `Simulation` tab of your model's window as shown in Fig. 28. Similarly, each of the `Time Scope` and `Spectrum Analyzer` windows also have a run button and stop button for convenience.

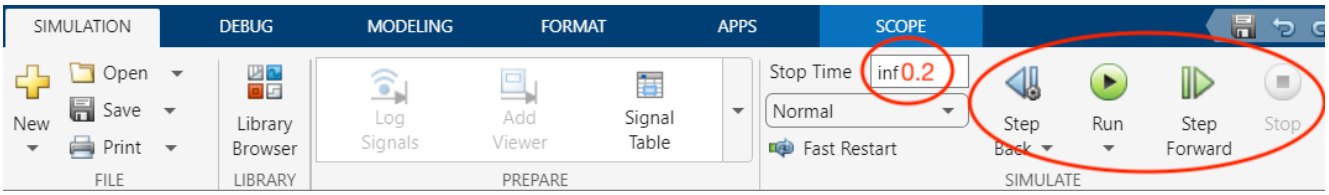

Fig. 28 `Run` and `Stop` buttons under `Simulation` tab.

Skip to main content

⚠ Lab Submission Considerations

Observe the simulation results in the `Time Scope` and in the `Spectrum Analyzer`. You may wish to save snapshots of the results. For the various Simulink scopes, you can select **File** → **Print to Figure** and then follow Saving Matlab Figures to save the figure.

1. Why do the `Time Scope` results not look perfectly sinusoidal?
2. How are the `Spectrum Analyzer` results the same and/or different from those you obtained in Matlab?

Implementing the 4^{th}-order IIR Lowpass Filter

Now that a suitable input signal `sigsum` is available and you understand its spectrum, you can begin to implement the 4^{th}-order IIR filter given in Fig. 20.

The model to implement the filter is given in Fig. 29 with the 4^{th}-order IIR filter subsystem given in Fig. 30. You can find the various blocks within the `FilterFramework.slx` model provided or in the **Library Browser** as follows:

- `Subsystem` block for the IIR filter under **Simulink** → **Ports & Subsystems**. You can double-click on the `Subsystem` block to see the model inside. Be sure to have the `In` and `Out` ports connected to the IIR filter.
- `Delay` block under **DSP System Toolbox** → **Signal Operations**. For the `Delay` blocks, you can *flip* the block in the model by right-clicking on it and selecting **Format** → **Flip Block**.
- `Sum` block and `Gain` block (multiplier for coefficients) under **Simulink** → **Math Operations**. Note that for the `Sum` block you can change the position of the inputs by double-clicking on the block and setting the **List of signs** to `++|` for the upper `Sum` blocks and to `|++` for the lower `Sum` blocks. For the `Gain` blocks, you can *rotate* the block in the model by right-clicking on it and selecting **Format** → **Rotate Block**.

Skip to main content

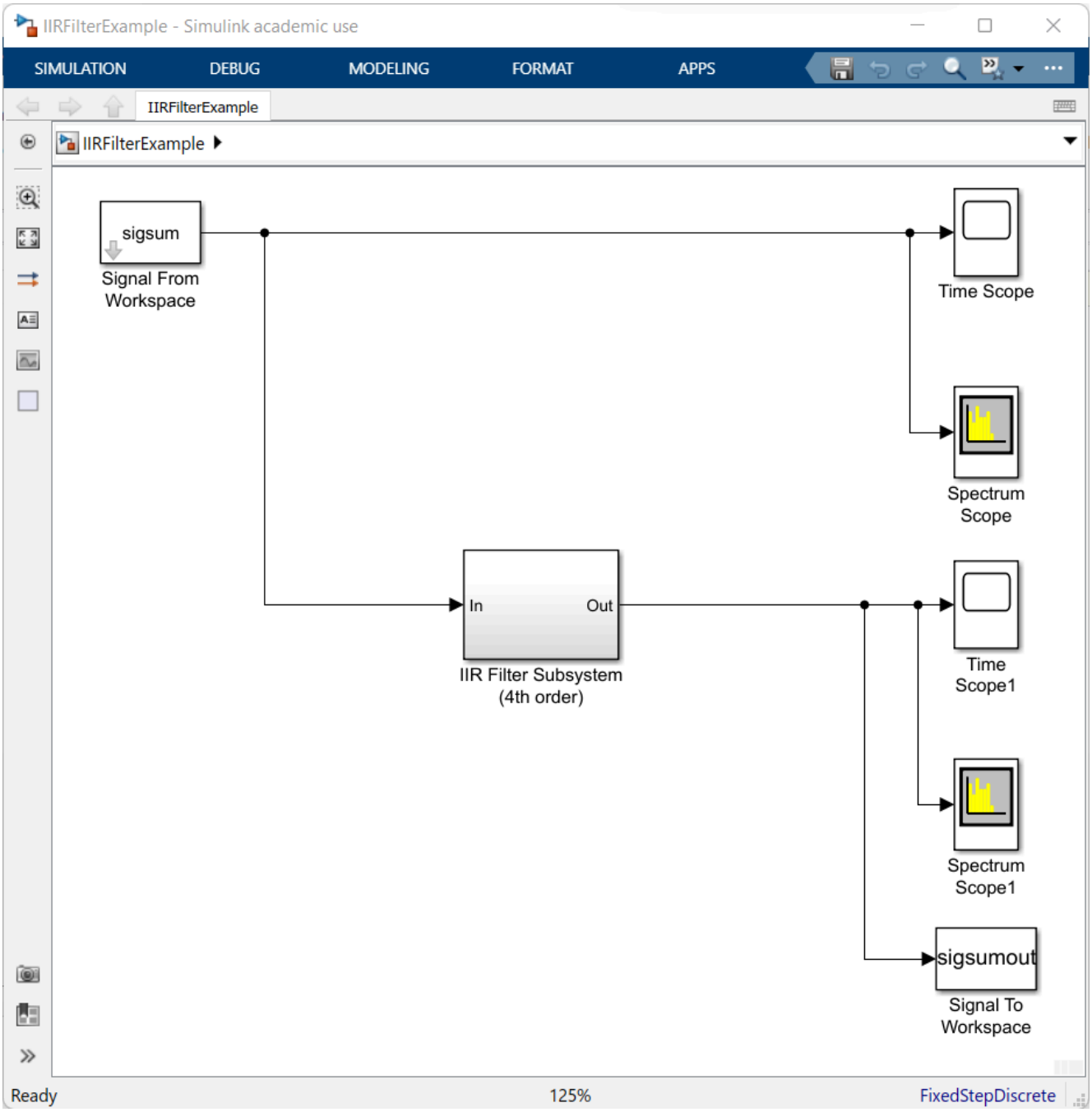

Fig. 29 Simulink model to analyse noisy sine after being filtered.

Skip to main content

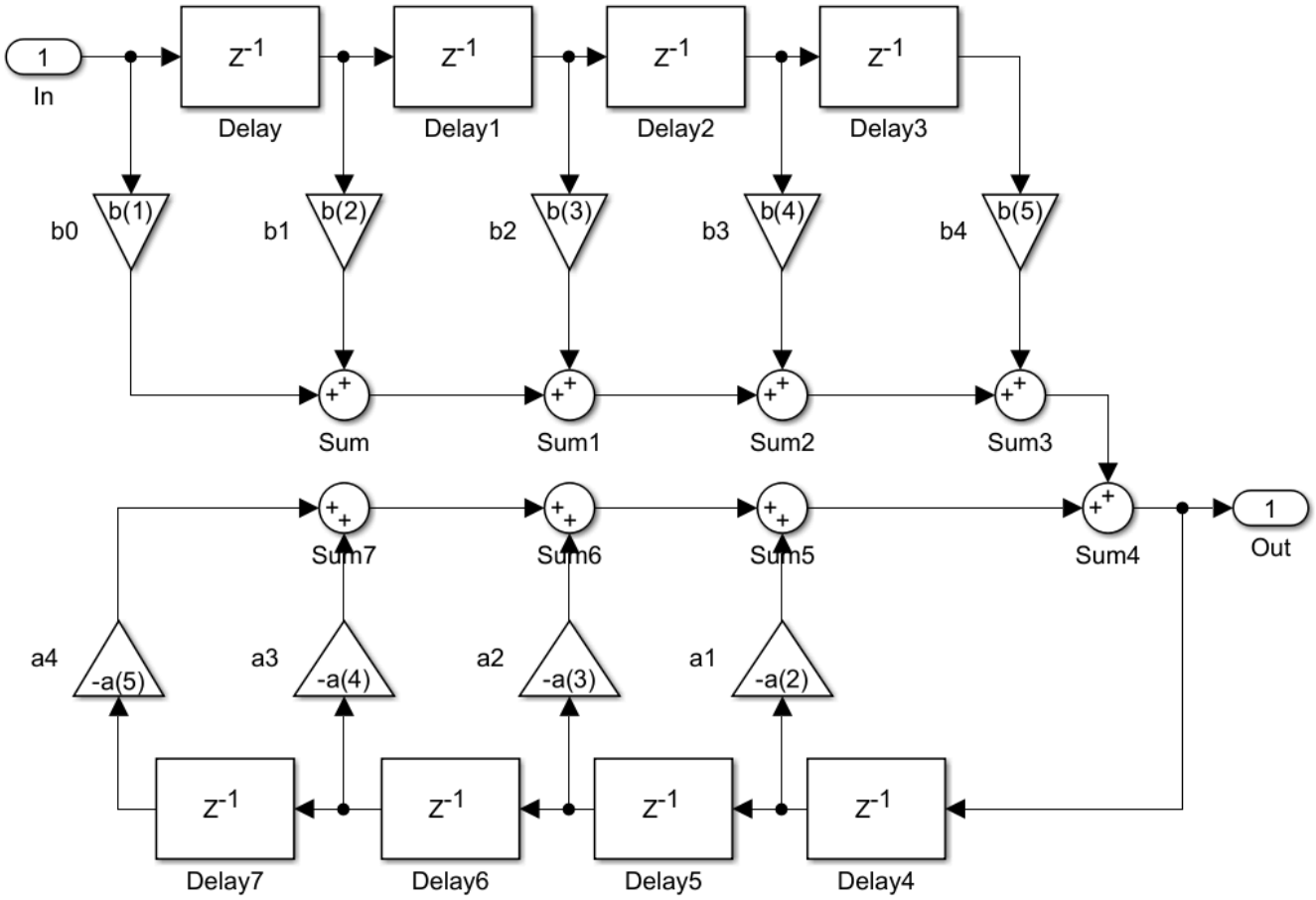

Fig. 30 4^{th}-order IIR filter in a Simulink subsystem.

For the `Gain` blocks in your IIR filter, be sure to enter the filter coefficient values for all of the b_k and a_k constants as given in (2). You need to be careful that the values for the a_k are written with the correct signs as in (2), or if using the `a` vector in Matlab that the $-a_k$ is used. In addition, note that the b_k and a_k indices are shifted by one since Matlab vectors start at 1 and not 0.

Once you have implemented the IIR lowpass filter, you can open up the scopes by double-clicking on them and then start the simulation. Note that the model shown in Fig. 29 will also output the filter results to the Matlab variable `sigsumout` with the inclusion of the `Signal To Workspace` block.

Skip to main content

Lab Submission Considerations

You may wish to save snapshots of the results. For the various Simulink scopes, you can select **File** → **Print to Figure** and then follow Saving Matlab Figures to save the figure.

1. How has the lowpass filter changed the input signal `sigsum`.
2. Is only the 150 Hz sinusoid left in the output signal after being passed through the lowpass filter?
3. What has happened to the higher frequency components in `sigsum` after filtering?

Implementing the 4^{th}-order IIR Lowpass Filter (Take Two)

Within the **DSP System Toolbox**, there is an easier way of implementing a filter by using the `Discrete Filter` block, which can be found under **DSP System Toolbox** → **Filtering** → **Filter Implementations** in the **Library Browser**. Implement the Simulink model given in Fig. 31 by replacing the previous filter with the `Discrete Filter` block.

Warning

You may want to save your model into a different file to keep a copy of your previous model.

Skip to main content

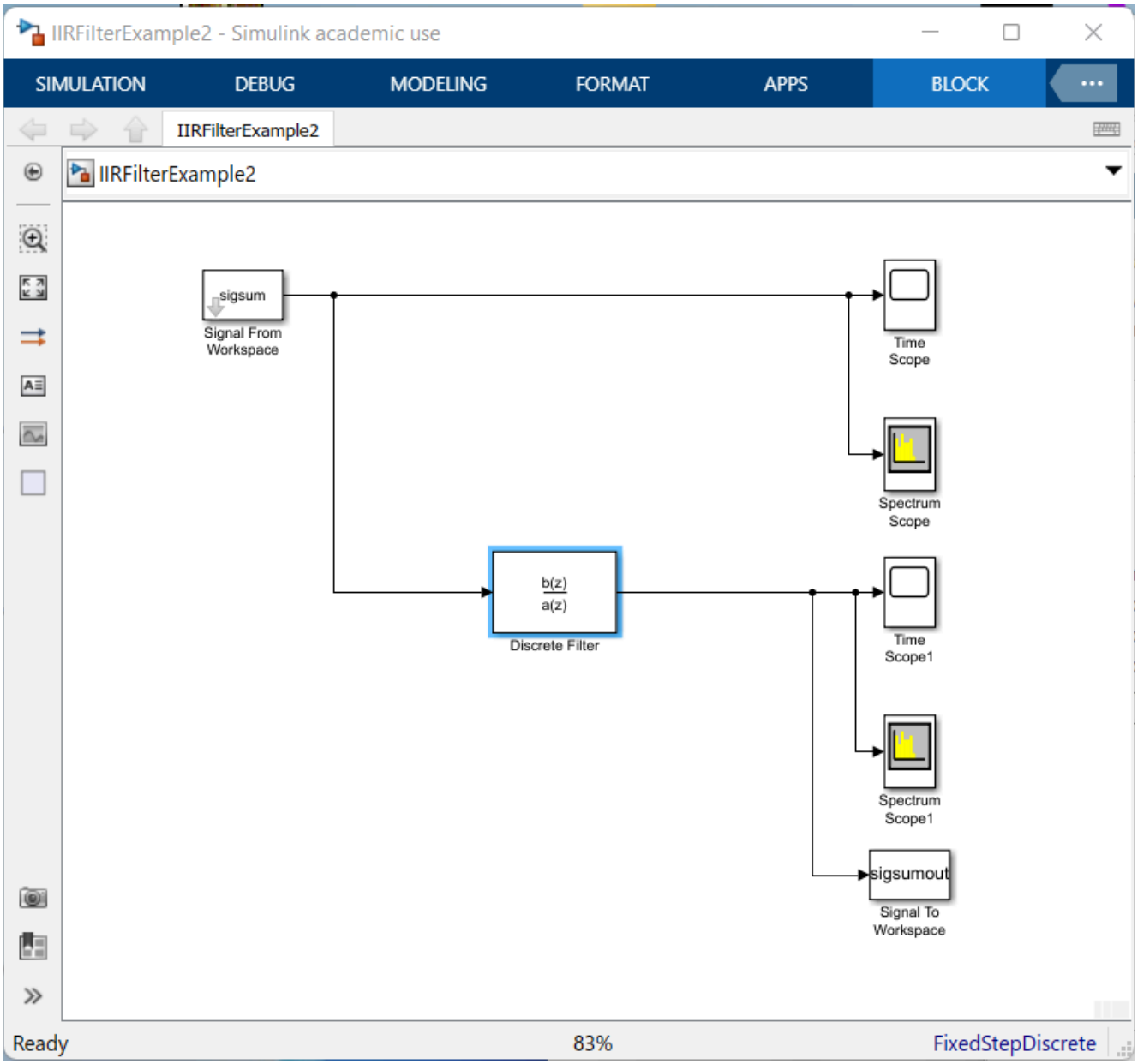

Fig. 31 Simulink model to analyse noisy sine after being filtered using `Discrete Filter` block.

An easy way to use the `Discrete Filter` block is to use the filter coefficients that we set in the Matlab vectors `b` and `a`. Double-click on the `Discrete Filter` block and set the `Numerator` to `b` and the `Denominator` to `a` as shown in Fig. 32. If you wish, you can also set the **Filter structure** to `Direct form I` which will mimic the filter structure you implemented in Fig. 30 (though, any of the filter structures will give the same results, ignoring numerical roundoff differences).

Skip to main content

Note

We will soon learn why these are called the numerator and denominator. Think of transfer functions in the Laplace domain, where we will see similar transfer functions in the z-transform domain.

We will also learn much later in the course about some of the different types of filter structures, including the various direct forms I and II.

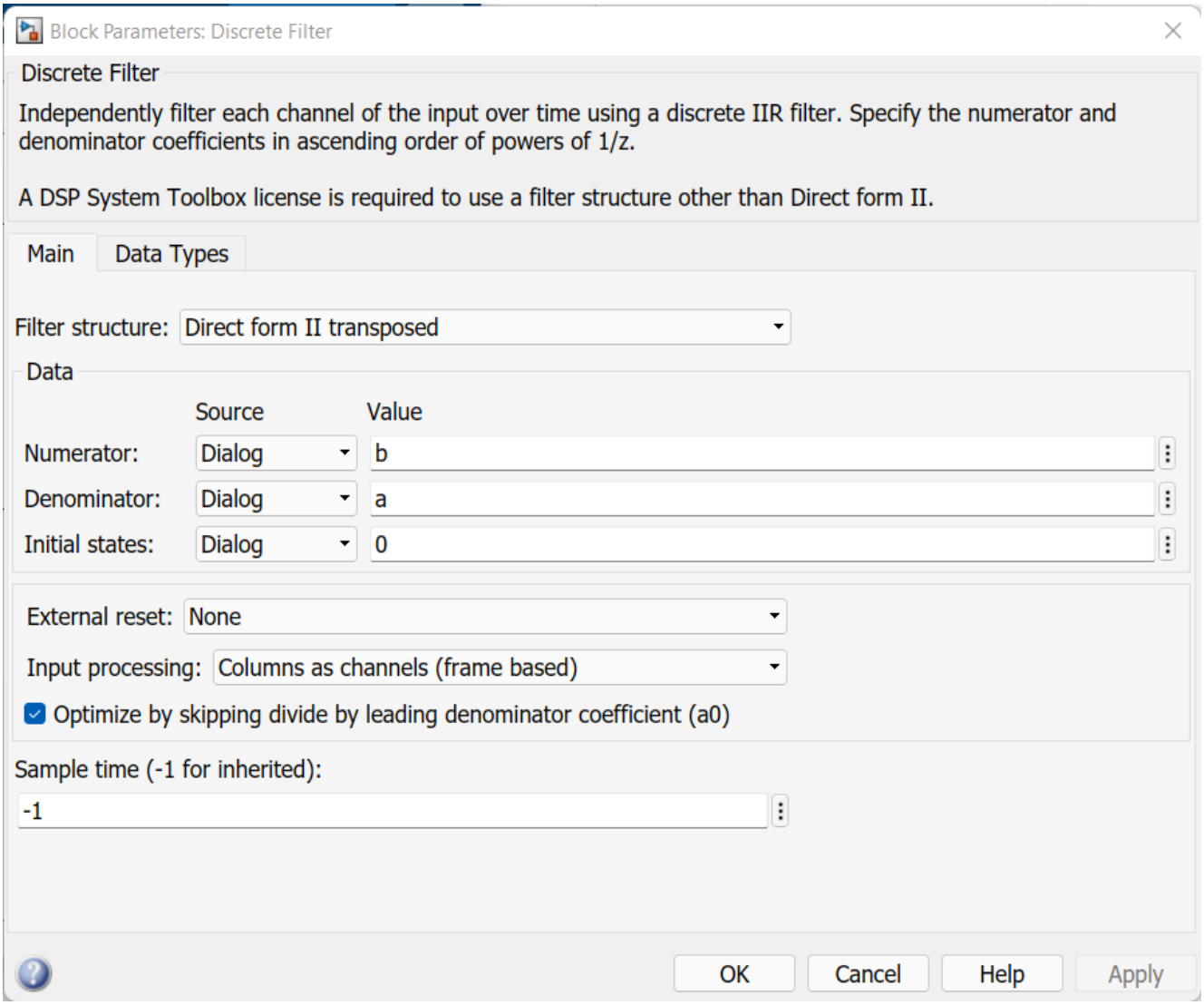

Fig. 32 Parameters for the `Discrete Filter` block to implement the IIR lowpass filter.

Now, simulate the Simulink model.

Skip to main content

⚠ Lab Submission Considerations

You may wish to save snapshots of the results. For the various Simulink scopes, you can select **File** → **Print to Figure** and then follow Saving Matlab Figures to save the figure.

1. Do you get the same results with the `Discrete Filter` block as with the hand-built filter structure in the previous section?