Digital Signal Processing

Lab Manual

EE-352L

Fall 2026

University of Management & Technology

School *of* Engineering

Department of Electrical Engineering

Name : _____

Student ID : _____

Section : _____

University of Management and Technology School of Engineering

Course Outline (Fall 2026)

Course Code: EE352L **Course Title: Digital Signal Processing Lab**

Program	BS Computer Engineering
Credit Hours	1
Contact Hours	3
Knowledge Domain	Computing
Prerequisites	Signal and Systems (EE-315L)
Resource Persons	M.Annas Marwat
Counseling Timing	See Office Window (SEN-310)
Email	muhammad.annas@umt.edu.pk

Chairman/Director signature_____

Dean signature_____

Date_____

Learning Objectives:

- **Transform Analysis:** Master sampling theorems, time-domain signals, and frequency-domain transforms (DFT, FFT, Z-transform).
- **Filter Design:** Design and implement FIR and Infinite Impulse Response (IIR) digital filters for noise reduction and frequency selection.
- **LTI Systems:** Characterize Linear Time-Invariant systems using convolution, impulse response, and frequency response.

Learning Methodology:

Course Outline Page 1 Lecture, interactive, participative

Grade Evaluation Criteria:

Following are the criteria for the distribution of marks to evaluate final grade in the semester.

Marks Evaluation	Marks
Lab Performance + Manuals	50
Open-Ended Lab	10
Final Viva + Performance	20+20
Total:	100

Recommended Text Books:

1. Signals and Systems, 2nd Edition by Simon Haykin and Barry Van Veen published by Wiley. Low priced edition is also available.

2. Digital Signal Processing, 4th Edition by Proakis and Manolakis, published by Pearson. Low priced edition is also available. Reference Books:

Course Learning Outcomes (CLOs):

CLO I: Use MATLAB to generate, manipulate and analyze discrete-time signals and systems (convolution, DTFT/DFT, Z-transform, frequency response and digital filters) by following the lab procedures.

CLO 2: Communicate experimental results effectively through plots, tables, lab reports and oral explanation of the MATLAB code and outputs.

CLO 3: Explain and interpret the effects of sampling, aliasing, quantization, reconstruction and filtering on signals from the experimental results.

CLO 4: Follow professional ethics in lab work: record results honestly, write original code and manuals, and cite sources when designing and documenting the filter experiments.

Relation to EE Program Learning Outcomes (PLOs):

CLOS	Related PLOS	Levels of Learning	Teaching Methods	CLO Attainment checked in
CLO I	PLO 5	Psy-3	Instructions, Lab Work	Lab Sessions, Exam
CLO 2	PLO 10	Aff-2	Instructions, Lab work	Viva,Lab Sessions,Exam
CLO 3	PLO 4	Cog-2	Instructions, Lab Work	Viva,Lab Sessions,Exam
CLO 4	PLO 8	Aff-3	Instructions, Lab Work	Lab Sessions,Manuals

Calendar of Course contents to be covered during semester

Course Code: EPE-352L **Course Title: Digital Signal Processing Lab**

Week	Course Contents/Lab Experiments	CLOS
1	Getting started with MATLAB for digital Signal Processing	1
2	Continuous-Time and Discrete-Time Signals	1,2,3,4
3	Linear Time Invariant (LTI) Systems and convolution	1,2,3,4
4	Fourier Representation of Signals	1,2,3,4
5	Discrete Time Fourier Transform (DTFT) and Its properties	1,2,3,4

6	Discrete Fourier Transform (DFT) and Audio Signal Spectrum	1,2,3,4
7	Z-Transform and Its Properties	1,2,3,4
	Midterm Exam	
8	Frequency Response and Stability of Discrete-Time Systems	1,2,3,4
9	Effects of Sampling and Aliasing	1,2,3,4
10	Quantization and Quantization Error	1,2,3,4
11	Signal Reconstruction Using MATLAB	1,2,3,4
12	Introduction to Digital Filtering	1,2,3,4
13	FIR Filter Design Using Windowing and Frequency Sampling	1,2,3,4
14	IIR Filter Design and FIR/IIR Comparison	1,2,3,4
15	MATLAB-Based Digital Filter Implementation and Audio Processing	1,2,3,4
16	Open Ended Lab	

Complex Engineering Problem/Activity:

Complex Engineering Problem Details	Included: No
Open Ended Lab	Included: Yes

Digital Signal Processing Lab

Digital Signal Processing (DSP) Lab Evaluation Rubric

Section A Lab #:

Areas Assessed	Implementation of Lab Work	Code Organization	Ability to Debug Codes	Illustration of experimental results	Analyze the outcome of the experiments	Responsible Attitude in Lab	CLO wise Total				Total
	(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 40%)	MATLAB code did not run or did not produce the required signals, plots or filters	Code was not properly organized (no comments, unclear variable names)	Wasn't able to debug the code	Plots and tables missing or unlabeled; wasn't able to present the results	No interpretation of the results (e.g. sampling, aliasing, quantization, filtering effects)	Frequently late, ignored lab rules or instructions, off-task, or submitted copied code or results					
Satisfactory (40 to 75%)	MATLAB code ran but only some tasks were correct or complete	Code was somewhat neat with few comments	Debugged the code with assistance	Results presented but some plots lacked titles, axis labels or legends	Reasonable interpretation of results, weak link to DSP theory	Mostly punctual and followed lab rules, needed occasional reminders to stay on task	CLO 1	CLO 2	CLO 3	CLO 4	
Very Good (75 to 100%)	MATLAB code was fully functional and all tasks produced correct results	Code was properly organized, commented and used meaningful variable names	Debugged code successfully without assistance	All results clearly presented with properly labeled plots, tables and explanation	Thorough interpretation of results, clearly linked to DSP theory	Punctual, followed all lab rules, stayed on task, results and code original and honest, equipment used responsibly					
Maximum Marks	2	1	1	2	2	2	4	2	2	2	10
Student ID| Student Name											
I											

Digital Signal Processing Lab

LAB-01: GETTING STARTED WITH MATLAB FOR DIGITAL SIGNAL PROCESSING

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

• Master fundamental mathematical derivations governing the underlying DSP operations.

• Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.

• Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.

• Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• Math Works MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- MATLAB Interactive Architecture: A high-level programming environment optimized for linear algebra, matrix manipulation, signal processing algorithm development, and data visualization.
- Vectorization Principles: Vectorization involves executing operations on entire arrays simultaneously rather than utilizing scalar for loops. In MATLAB, vectorized operations leverage highly optimized BLAS (Basic Linear Algebra Subprograms) libraries.
- Discrete vs Continuous Representation: Digital computers store signals as finite sequences of discrete numbers x[n]. plot() provides continuous visual curves via piecewise linear interpolation, whereas stem() accurately depicts discrete sequences with vertical stems and sample markers.
- Element-wise Arithmetic: Operators prefixed with a dot (e.g., .*, ./, .^) execute operations element-by-element across matrices, distinguishing them from standard linear algebra matrix operators (*, /, ^).

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 1.1: **Signal Energy and Power Calculation**
Problem: Given a discrete sequence x[n] = [3, -2, 4, 1]

Where n = [0, 1, 2, 3]

Compute the total signal energy Ex and average power Px.

Solution:
1. Energy formula: Ex = Sum(|x[n]|^2) = |3|^2 + |-2|^2 + |4|^2 + |1|^2 = 9 + 4 + 16 + 1 = 30.
2. Average Power formula over length N=4: Px = Ex / N = 30 / 4 = 7.5 Watts.

Digital Signal Processing Lab

Solved Example 1.2: **Discrete Sequence Transformations**

Problem: Let x[n] = 2n + 1 for n = [0, 1, 2, 3]. Compute y[n] = 3x[n] - 4.

Solution:

For n=0: x[0]=1 => y[0]=3(1)-4 = -1

For n=1: x[1]=3 => y[1]=3(3)-4 = 5

For n=2: x[2]=5 => y[2]=3(5)-4 = 11

For n=3: x[3]=7 => y[3]=3(7)-4 = 17

Result vector y[n] = [-1, 5, 11, 17].

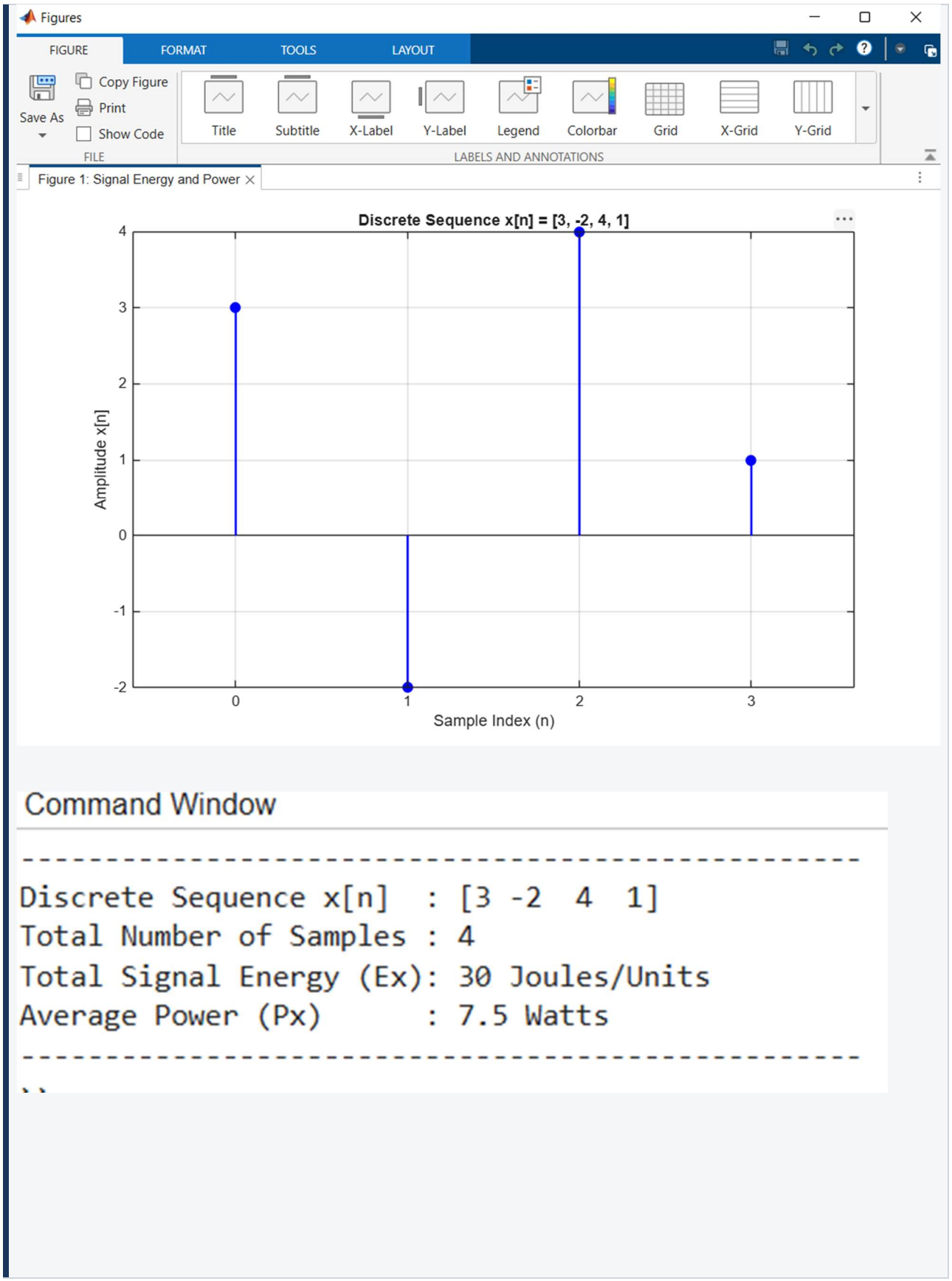
5. Complete MATLAB Code Implementation & Graphical Outputs

```
% Example 1.1
% ========================================================================
% MATLAB Script for Solved Example 1.1: Signal Energy and Power
% Description: Computes the total energy (Ex) and average power (Px)
%              of a discrete-time sequence x[n] = [3, -2, 4, 1].
% ========================================================================
clc; clear; close all;
% 1. Define the discrete sequence and its time indices
n = [0, 1, 2, 3];
x_n = [3, -2, 4, 1];
% 2. Compute Total Signal Energy (Ex = sum(|x[n]|^2))
Ex = sum(abs(x_n).^2);
% 3. Compute Average Power (Px = Ex / N, where N is the total sample count)
N = length(x_n);
Px = Ex / N;

% 4. Display Results in the Command Window
disp('-------------------------------------------------');
disp(['Discrete Sequence x[n]  : [', num2str(x_n), ']']);
disp(['Total Number of Samples : ', num2str(N)]);
disp(['Total Signal Energy (Ex): ', num2str(Ex), ' Joules/Units']);
disp(['Average Power (Px)      : ', num2str(Px), ' Watts']);
disp('-------------------------------------------------');

% 5. Graphical Visualization using Stem Plot
figure('Name', 'Signal Energy and Power', 'Color', 'white');
stem(n, x_n, 'b', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Discrete Sequence x[n] = [3, -2, 4, 1]');
xlabel('Sample Index (n)');
ylabel('Amplitude x[n]');
grid on;
set(gca, 'XTick', n);
```

Digital Signal Processing Lab



Command Window

```
------------------------------------------------
Discrete Sequence x[n]  : [3 -2  4  1]
Total Number of Samples : 4
Total Signal Energy (Ex): 30 Joules/Units
Average Power (Px)      : 7.5 Watts
------------------------------------------------
>>
```

Digital Signal Processing Lab

```
% Example 1.2
% ========================================================================
% MATLAB Script for Solved Example 1.2: Discrete Sequence Transformations
% Description: Computes x[n] = 2n + 1 and the transformed sequence
%              y[n] = 3*x[n] - 4 for n = [0, 1, 2, 3].
% ========================================================================
clc; clear; close all;

% 1. Define the discrete time index vector n
n = 0 : 3;
% 2. Compute the original discrete sequence x[n] = 2n + 1
x_n = 2 * n + 1;
% 3. Compute the transformed sequence y[n] = 3*x[n] - 4
y_n = 3 * x_n - 4;

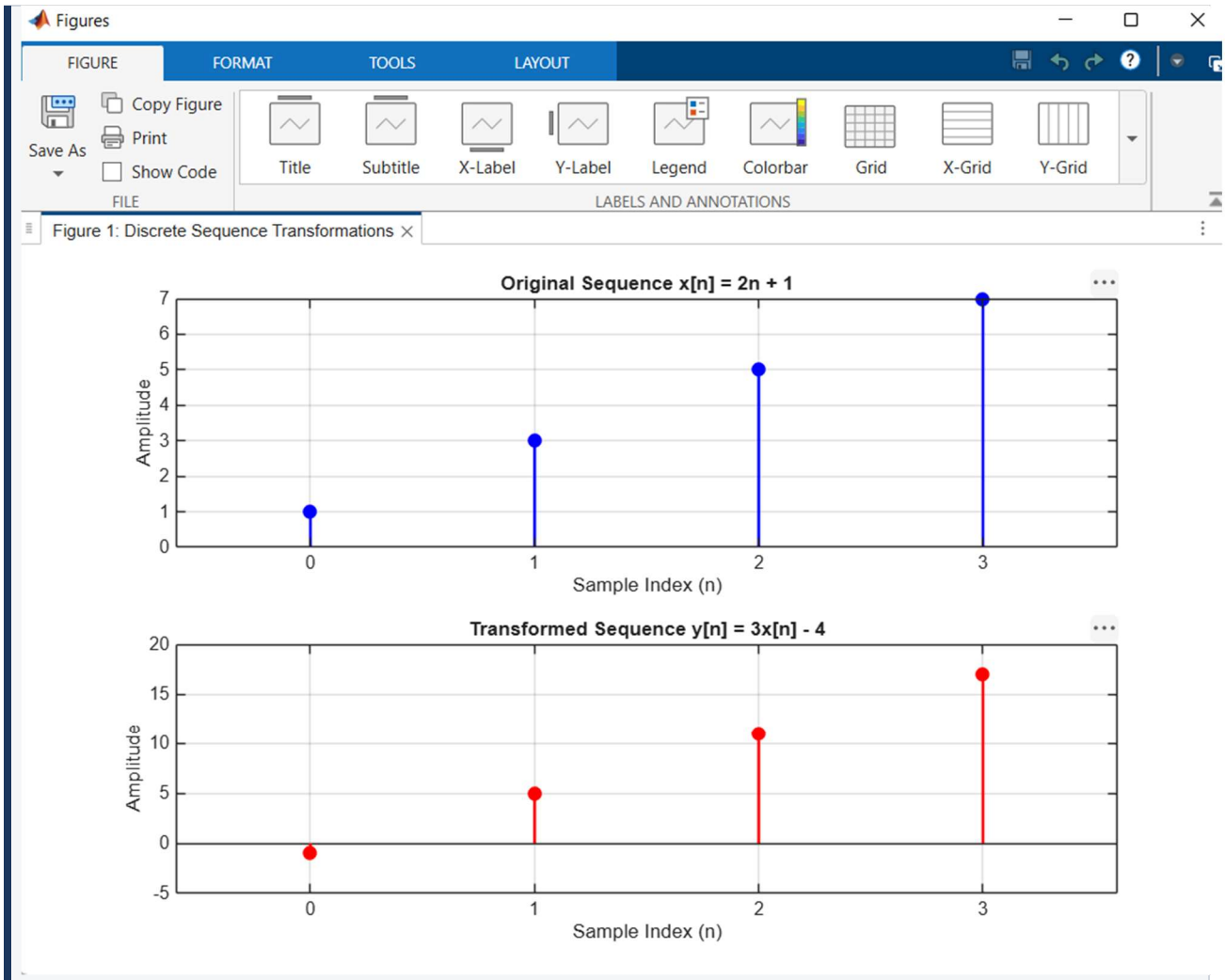
% 4. Display the computed vectors in the Command Window
disp('-----------------------------------------------------');
disp(['Sample Indices (n)     : ', num2str(n)]);
disp(['Original Sequence x[n] : ', num2str(x_n)]);
disp(['Transformed Sequence y[n]: ', num2str(y_n)]);
disp('-----------------------------------------------------');

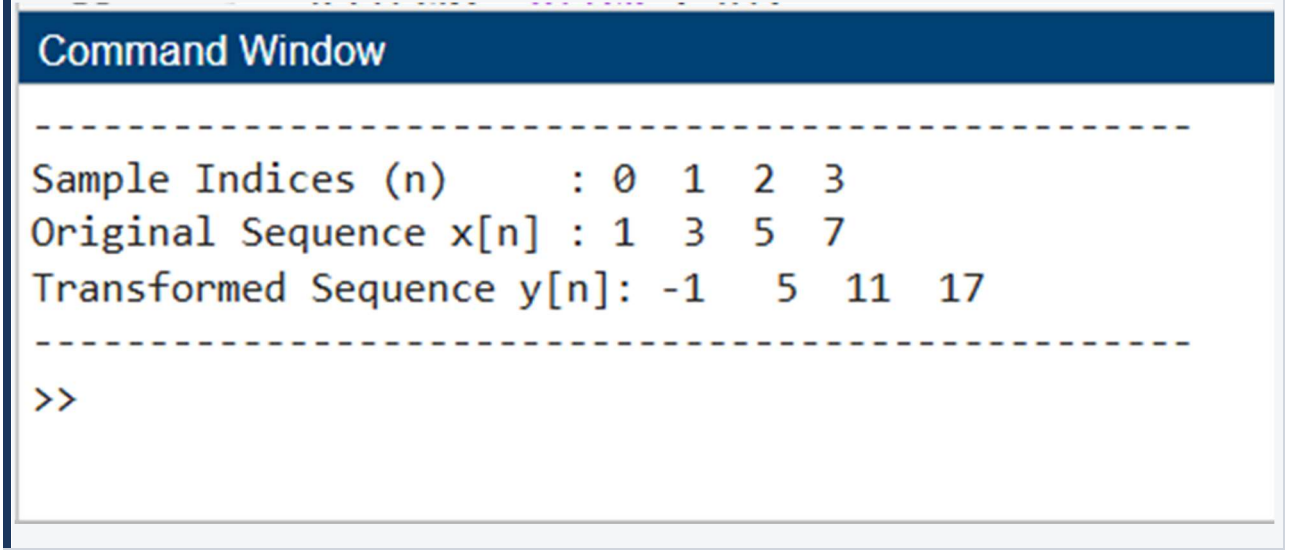
% 5. Graphical Visualization using Stem Plots
figure('Name', 'Discrete Sequence Transformations', 'Color', 'white');

% Subplot 1: Original Sequence x[n]
subplot(2, 1, 1);
stem(n, x_n, 'b', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Original Sequence x[n] = 2n + 1');
xlabel('Sample Index (n)');
ylabel('Amplitude');
grid on;
set(gca, 'XTick', n);

% Subplot 2: Transformed Sequence y[n]
subplot(2, 1, 2);
stem(n, y_n, 'r', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Transformed Sequence y[n] = 3x[n] - 4');
xlabel('Sample Index (n)');
ylabel('Amplitude');
grid on;
set(gca, 'XTick', n);
```

Digital Signal Processing Lab





Digital Signal Processing Lab

5. LAB Tasks

- Task 1.1: Generate continuous sinusoidal wave y(t) = 2*sin(4*pi*t) + 3*cos(10*pi*t) for t in [0, 1] with dt = 0.001 s. Plot with full annotations.
- Task 1.2: Generate discrete sequence x[n] = n*(0.85)^n for n in [0, 20]. Visualize using stem() with customized markers.
- Task 1.3: Define 3x3 matrices A and B. Perform matrix multiplication, element-wise product, matrix inverse, and transpose.
- Task 1.4: Write a MATLAB script to compute and display signal energy and mean power for an arbitrary user-entered vector.

7. In-Depth Observations, Analytical Discussion & Verification

- Linear interpolation in plot() creates smooth curves, but fails to show exact discrete sample instants.
- stem() clearly delineates discrete index locations, confirming that values exist strictly at integer n.
- Vectorized element-wise operators (.*) avoid computationally expensive loop structures.

8. LAB Assignments

- Assignment 1.1: Write a MATLAB program to calculate the dot product of two 100-element vectors using loops and vectorization, comparing execution time via tic/toc.
- Assignment 1.2: Create a 2x2 subplot displaying unit step, unit impulse, ramp, and decaying exponential sequences.
- Assignment 1.3: Generate a discrete cosine wave with frequency f0 = 50 Hz sampled at Fs = 1000 Hz over 50 samples.

Digital Signal Processing Lab

LAB- 02: CONTINUOUS-TIME AND DISCRETE-TIME SIGNALS

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

• Master fundamental mathematical derivations governing the underlying DSP operations.

• Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.

• Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.

• Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• Math Works MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Unit Impulse Sequence delta[n]: Defined as delta[n] = 1 for n=0 and delta[n] = 0 for n != 0. It serves as the fundamental building block for discrete LTI system analysis.
- Unit Step Sequence u[n]: Defined as u[n] = 1 for n >= 0 and u[n] = 0 for n < 0. Related to impulse by u[n] = Sum(delta[k]) from -inf to n.
- Time Shifting: Delaying (x[n-k], k>0) or advancing (x[n+k], k>0) a signal along the sample index axis.
- Time Reversal: Reflecting a signal across the vertical sample axis n=0, yielding x[-n].
- Even and Odd Signal Decomposition: Any sequence can be decomposed into xe[n] = 0.5(x[n] + x[-n]) and xo[n] = 0.5(x[n] - x[-n]).

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 2.1: Signal Time Transformations
Problem: Given x[n] = [1, 2, 3, 4] for n = [0, 1, 2, 3], sketch and tabulate y[n] = x[n-2] and z[n] = x[-n].
Solution:
1. y[n] = x[n-2] shifts right by 2 units: y[2]=1, y[3]=2, y[4]=3, y[5]=4. All other samples are 0.
2. z[n] = x[-n] reflects across n=0: z[0]=1, z[-1]=2, z[-2]=3, z[-3]=4. All other samples are 0.

Solved Example 2.2: Even and Odd Decomposition
Problem: Decompose x[n] = [2, 4, 6] for n = [0, 1, 2] into even and odd components.
Solution:
Sequence x[-n] is defined over n = [0, -1, -2] as [2, 4, 6].
Aligning indices over n = [-2, -1, 0, 1, 2]:
x[n] = [0, 0, 2, 4, 6]
x[-n] = [6, 4, 2, 0, 0]

Digital Signal Processing Lab

Even component xe[n] = 0.5(x[n]+x[-n]) = [3, 2, 2, 2, 3].
Odd component xo[n] = 0.5(x[n]-x[-n]) = [-3, -2, 0, 2, 3].

5. Complete MATLAB Code Implementation & Graphical Outputs

```
Example 2.1
% ========================================================================
% MATLAB Script for Solved Example 2.1: Signal Time Transformations
% Description: Computes and visualizes time shifting y[n] = x[n-2]
%              and time reversal z[n] = x[-n] for x[n] = [1, 2, 3, 4].
% ========================================================================

clc; clear; close all;

% 1. Define original sequence and its time indices
n = 0 : 3;
x = [1, 2, 3, 4];

% 2. Time Shifting: y[n] = x[n-2] (shifts right by 2 units)
ny = n + 2;  % Shifted indices: [2, 3, 4, 5]
y = x;       % Amplitude values remain: [1, 2, 3, 4]

% 3. Time Reversal: z[n] = x[-n] (reflects across n = 0)
nz = -fliplr(n);  % Reversed indices: [-3, -2, -1, 0]
z = fliplr(x);    % Reversed values: [4, 3, 2, 1]

% 4. Tabulate and Display Results in the Command Window
disp('-------------------------------------------------');
disp('Original Sequence x[n]:');
disp(['  Indices (n) : ', num2str(n)]);
disp(['  Values      : ', num2str(x)]);
disp('-------------------------------------------------');
disp('Shifted Sequence y[n] = x[n-2]:');
disp(['  Indices (ny): ', num2str(ny)]);
disp(['  Values      : ', num2str(y)]);
disp('-------------------------------------------------');
disp('Reversed Sequence z[n] = x[-n]:');
disp(['  Indices (nz): ', num2str(nz)]);
disp(['  Values      : ', num2str(z)]);
disp('-------------------------------------------------');

% 5. Graphical Visualization using Subplots
figure('Name', 'Signal Time Transformations', 'Color', 'white');

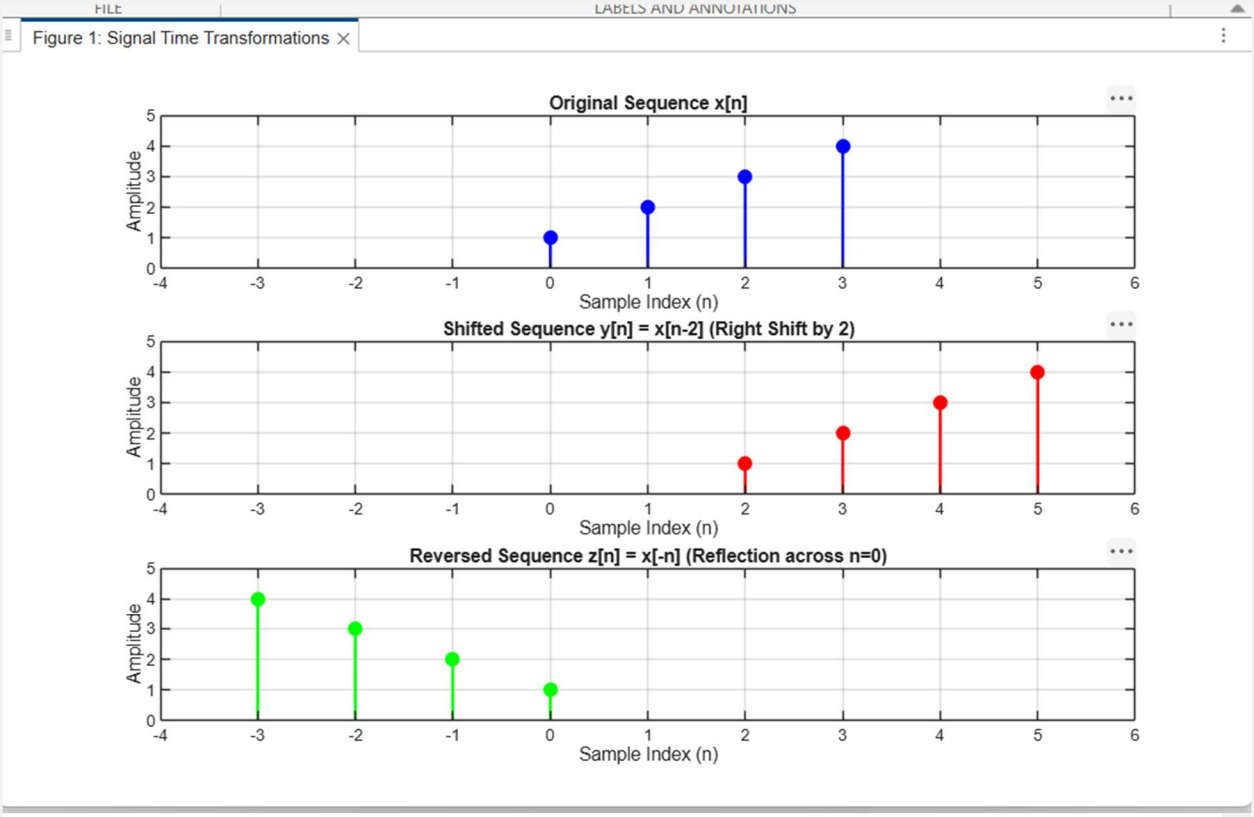
% Subplot 1: Original Sequence
subplot(3, 1, 1);
stem(n, x, 'b', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Original Sequence x[n]');
xlabel('Sample Index (n)'); ylabel('Amplitude');
grid on; xlim([-4, 6]); ylim([0, 5]);

% Subplot 2: Shifted Sequence
subplot(3, 1, 2);
```

Digital Signal Processing Lab

```
stem(ny, y, 'r', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Shifted Sequence y[n] = x[n-2] (Right Shift by 2)');
xlabel('Sample Index (n)'); ylabel('Amplitude');
grid on; xlim([-4, 6]); ylim([0, 5]);

% Subplot 3: Reversed Sequence
subplot(3, 1, 3);
stem(nz, z, 'g', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Reversed Sequence z[n] = x[-n] (Reflection across n=0)');
xlabel('Sample Index (n)'); ylabel('Amplitude');
grid on; xlim([-4, 6]); ylim([0, 5]);
```




```
Command Window
  Values       : 1   2   3   4
-----------------------------------------------
Reversed Sequence z[n] = x[-n]:
  Indices (nz): -3 -2 -1  0
  Values       : 4   3   2   1
-----------------------------------------------
>>
```

```
% ========================================================================
```

Digital Signal Processing Lab

```
% MATLAB Script for Solved Example 2.2: Even & Odd Decomposition
% Description: Decomposes a discrete sequence x[n] = [2, 4, 6] (n = 0, 1, 2)
%              into its even (xe[n]) and odd (xo[n]) components.
% ========================================================================

clc; clear; close all;

% 1. Define original sequence and its time indices
n_orig = 0 : 2;
x_orig = [2, 4, 6];

% 2. Define the symmetric unified time index range covering both x[n] and x[-n]
%    Since x[n] spans [0, 2] and x[-n] spans [-2, 0], the combined range is [-2, 2]
n = -2 : 2;

% 3. Embed x[n] into the unified time vector n (zeros outside [0, 2])
x = zeros(size(n));
x(n >= 0 & n <= 2) = x_orig;

% 4. Compute time-reversed sequence x[-n] over the same unified time range n
x_rev = fliplr(x);
% Alternatively, indexing can be achieved via matching negative indices:
% x_rev = zeros(size(n));
% for i = 1:length(n)
%     match_idx = find(n == -n(i));
%     if ~isempty(match_idx), x_rev(i) = x(match_idx); end
% end

% 5. Compute Even and Odd Components
%    xe[n] = 0.5 * (x[n] + x[-n])
%    xo[n] = 0.5 * (x[n] - x[-n])
xe = 0.5 * (x + x_rev);
xo = 0.5 * (x - x_rev);

% 6. Verification: Reconstruct original signal x_recon = xe + xo
x_recon = xe + xo;

% 7. Display Results in the Command Window
disp('-------------------------------------------------');
disp(['Unified Indices (n)     : ', num2str(n)]);
disp(['Aligned Signal x[n]     : ', num2str(x)]);
disp(['Reversed Signal x[-n]   : ', num2str(x_rev)]);
disp(['Even Component xe[n]    : ', num2str(xe)]);
disp(['Odd Component xo[n]     : ', num2str(xo)]);
disp(['Reconstructed x[n]      : ', num2str(x_recon)]);
disp('-------------------------------------------------');

% 8. Graphical Visualization using Subplots
figure('Name', 'Even and Odd Signal Decomposition', 'Color', 'white');

subplot(4, 1, 1);
stem(n, x, 'b', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Original Sequence x[n]');
xlabel('n'); ylabel('Amplitude'); grid on; xlim([-3, 3]);
```

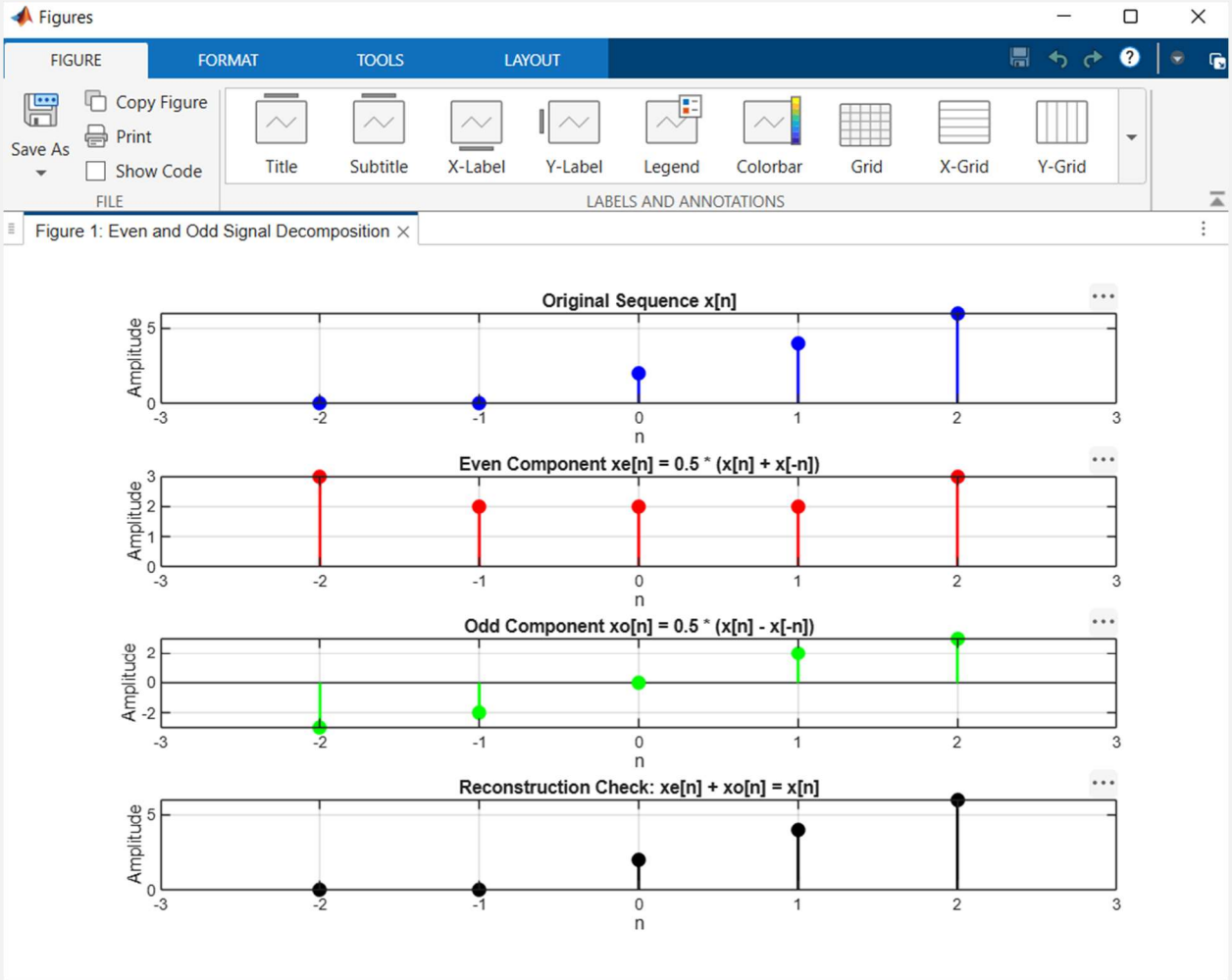
Digital Signal Processing Lab

```
subpl
stem(
title
xlabe

subpl
stem(
title
xlabe

subpl
stem(
title
xlabel('n'); ylabel('Amplitude'); grid on; xlim([-3, 3]);
```

```
Command Window
Aligned Signal x[n]     : 0  0  2  4  6
Reversed Signal x[-n]   : 6  4  2  0  0
Even Component xe[n]    : 3  2  2  2  3
Odd Component xo[n]     : -3 -2  0  2  3
Reconstructed x[n]      : 0  0  2  4  6
-------------------------------------------------
>>
```



Digital Signal Processing Lab



6. Laboratory Tasks & Worksheets

- Task 2.1: Write MATLAB functions to generate unit impulse delta[n-k] and unit step u[n-k] over range n in [-10, 10].
- Task 2.2: Generate x[n] = cos(0.1*pi*n) for n in [0, 30]. Plot original, shifted x[n-4], and reversed x[-n].
- Task 2.3: Implement even and odd decomposition of an arbitrary user sequence and verify that xe[n] + xo[n] = x[n].
- Task 2.4: Perform amplitude scaling and addition of two discrete sequences defined over different time boundaries.

7. In-Depth Observations, Analytical Discussion & Verification

- Time-shifting shifts the signal waveform horizontally without modifying sample amplitudes.
- Time-reversal reflects the sequence around index n=0.
- Adding even and odd components exactly reconstructs the original signal across all index boundaries.

8. LAB Assignments

- Assignment 2.1: Generate an exponentially damped sinusoid x[n] = (0.85)^n * cos(0.2*pi*n) * u[n] for n in [0, 30] and plot its upper and lower envelopes.
- Assignment 2.2: Compute the total energy of a discrete rectangular pulse of duration N=15 samples.
- Assignment 2.3: Prove analytically and verify in MATLAB that the product of two even signals is even, and the product of two odd signals is even.

Digital Signal Processing Lab

LAB-03: LINEAR TIME-INVARIANT (LTI) SYSTEMS AND CONVOLUTION

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

• Master fundamental mathematical derivations governing the underlying DSP operations.

• Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.

• Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.

• Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- LTI System Properties: Linearity satisfies Superposition (S{a*x1[n] + b*x2[n]} = a*S{x1[n]} + b*S{x2[n]}). Time-Invariance ensures that if y[n] = S{x[n]}, then S{x[n-k]} = y[n-k].
- Impulse Response h[n]: The output of an LTI system when excited by a unit impulse input delta[n]. h[n] completely characterizes the LTI system.
- Discrete Convolution Sum: y[n] = x[n] * h[n] = Sum(x[k] * h[n-k]) from k=-inf to inf.
- Output Vector Length Rule: If length of x[n] is Nx and length of h[n] is Nh, the length of convolved output y[n] is Ny = Nx + Nh - 1.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 3.1: Analytical Linear Convolution
Problem: Compute convolution sum of x[n] = [1, 2, 3] for n=[0,1,2] and h[n] = [1, 1] for n=[0,1].
Solution:
Ny = 3 + 2 - 1 = 4. Output range n in [0, 3].
y[0] = x[0]*h[0] = 1(1) = 1
y[1] = x[0]*h[1] + x[1]*h[0] = 1(1) + 2(1) = 3
y[2] = x[1]*h[1] + x[2]*h[0] = 2(1) + 3(1) = 5
y[3] = x[2]*h[1] = 3(1) = 3
Resulting convolution sequence y[n] = [1, 3, 5, 3].

Solved Example 3.2: Linearity Verification
Problem: Test if system y[n] = 2*x[n] + 3 is linear.
Solution:
Let input be a*x1[n] + b*x2[n]. Output S{a*x1 + b*x2} = 2*(a*x1[n] + b*x2[n]) + 3 = 2a*x1[n] + 2b*x2[n] + 3.

Digital Signal Processing Lab

Linear combination of outputs: a*y1[n] + b*y2[n] = a*(2*x1[n]+3) + b*(2*x2[n]+3) = 2a*x1[n] + 2b*x2[n] + 3a + 3b.
Since 2a*x1 + 2b*x2 + 3 != 2a*x1 + 2b*x2 + 3a + 3b, the system is Non-Linear.

5. Complete MATLAB Code Implementation & Graphical Outputs

```
Example 3.1
% ========================================================================
% MATLAB Script for Solved Example 3.1: Analytical Linear Convolution
% Description: Computes the linear convolution of x[n] = [1, 2, 3]
%              and h[n] = [1, 1] using MATLAB's conv() function and
%              visualizes the resulting output sequence y[n] = [1, 3, 5, 3].
% ========================================================================

clc; clear; close all;

% 1. Define input sequences and their time index domains
x = [1, 2, 3];
nx = 0 : length(x) - 1;  % Indices: [0, 1, 2]

h = [1, 1];
nh = 0 : length(h) - 1;  % Indices: [0, 1]

% 2. Compute Linear Convolution using MATLAB's conv() function
y = conv(x, h);

% 3. Determine the resulting output time indices (Ny = Nx + Nh - 1)
ny = (nx(1) + nh(1)) : (nx(end) + nh(end));  % Indices: [0, 1, 2, 3]

% 4. Display Results in the Command Window
disp('-----------------------------------------------------');
disp(['Input Sequence x[n] at indices ', num2str(nx), ' : [', num2str(x), ']']);
disp(['Impulse Response h[n] at indices ', num2str(nh), ' : [', num2str(h), ']']);
disp(['Convolution Output y[n] at indices ', num2str(ny), ' : [', num2str(y), ']']);
disp('-----------------------------------------------------');

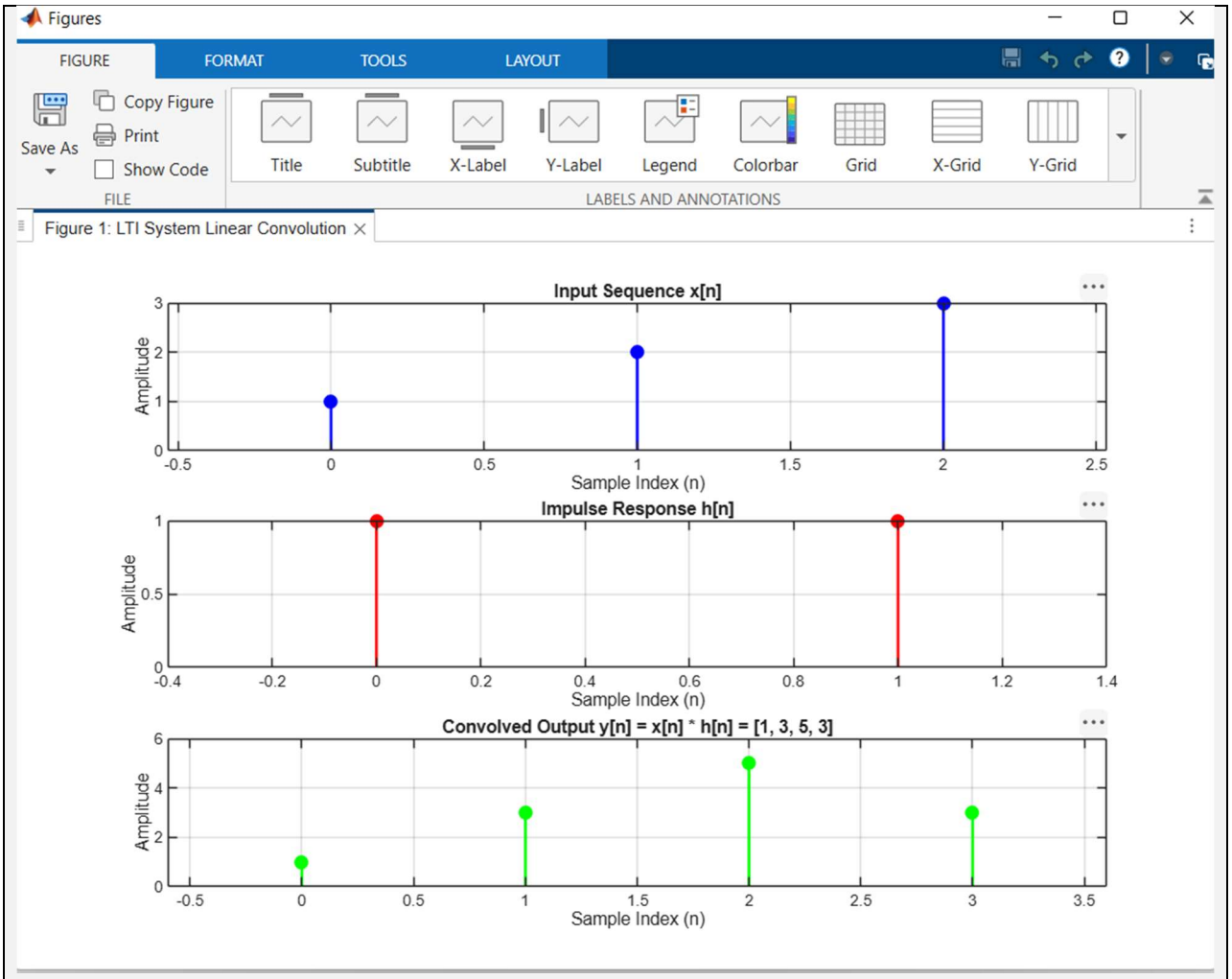
% 5. Graphical Visualization using Subplots
figure('Name', 'LTI System Linear Convolution', 'Color', 'white');

subplot(3, 1, 1);
stem(nx, x, 'b', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Input Sequence x[n]');
xlabel('Sample Index (n)'); ylabel('Amplitude'); grid on;

subplot(3, 1, 2);
stem(nh, h, 'r', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Impulse Response h[n]');
xlabel('Sample Index (n)'); ylabel('Amplitude'); grid on;

subplot(3, 1, 3);
stem(ny, y, 'g', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Convolved Output y[n] = x[n] * h[n] = [1, 3, 5, 3]');
xlabel('Sample Index (n)'); ylabel('Amplitude'); grid on;
```

Digital Signal Processing Lab



Command Window

```
----------------------------------------------------
Input Sequence x[n] at indices 0  1  2 : [1  2  3]
Impulse Response h[n] at indices 0  1 : [1  1]
Convolution Output y[n] at indices 0  1  2  3 : [1  3  5  3]
----------------------------------------------------
>>
```

Example 3.2

```
% ========================================================================
% MATLAB Script for Solved Example 3.2: System Linearity Verification
% Description: Tests whether the system y[n] = 2*x[n] + 3 satisfies
%              the principle of superposition (linearity).
% ========================================================================

clc; clear; close all;

% 1. Define two arbitrary input sequences x1[n] and x2[n]
n = 0 : 3;
x1 = [1, 2, 3, 4];
x2 = [2, 1, 4, 2];
```

Digital Signal Processing Lab

```
% 2. Define scaling constants a and b
a = 2;
b = 3;

% --- Step A: Compute System Output for the Linear Combination of Inputs ---
% Input_A = a*x1 + b*x2
input_A = a * x1 + b * x2;
% System operation: S{input} = 2*input + 3
output_A = 2 * input_A + 3;

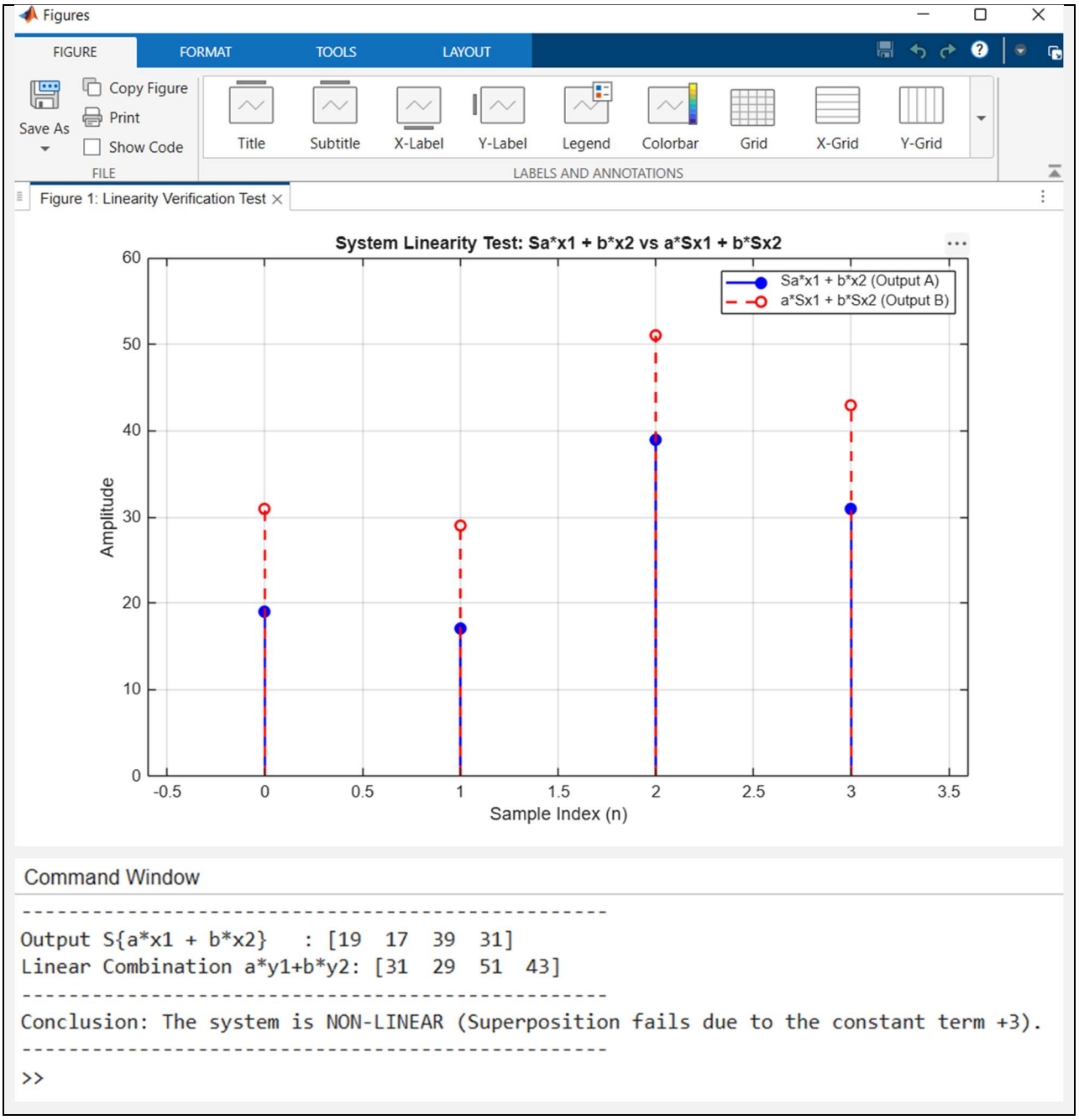
% --- Step B: Compute Linear Combination of Individual Outputs ---
% y1 = S{x1}, y2 = S{x2}
y1 = 2 * x1 + 3;
y2 = 2 * x2 + 3;
% Output_B = a*y1 + b*y2
output_B = a * y1 + b * y2;

% 3. Compare the Two Results
difference = output_A - output_B;
is_linear = isequal(output_A, output_B);

% 4. Display Results in the Command Window
disp('--------------------------------------------------');
disp(['Input x1[n]             : [', num2str(x1), ']']);
disp(['Input x2[n]             : [', num2str(x2), ']']);
disp(['Scaled Input a*x1 + b*x2: [', num2str(input_A), ']']);
disp('--------------------------------------------------');
disp(['Output S{a*x1 + b*x2}   : [', num2str(output_A), ']']);
disp(['Linear Combination a*y1+b*y2: [', num2str(output_B), ']']);
disp('--------------------------------------------------');
if is_linear
    disp('Conclusion: The system is LINEAR.');
else
    disp('Conclusion: The system is NON-LINEAR (Superposition fails due to the
constant term +3).');
end
disp('--------------------------------------------------');

% 5. Graphical Visualization
figure('Name', 'Linearity Verification Test', 'Color', 'white');
stem(n, output_A, 'b', 'filled', 'LineWidth', 1.5, 'MarkerSize', 6); hold on;
stem(n, output_B, 'r--', 'LineWidth', 1.5, 'MarkerSize', 6);
title('System Linearity Test: S{a*x1 + b*x2} vs a*S{x1} + b*S{x2}');
xlabel('Sample Index (n)'); ylabel('Amplitude');
legend('S{a*x1 + b*x2} (Output A)', 'a*S{x1} + b*S{x2} (Output B)');
grid on; hold off;
```

Digital Signal Processing Lab



6. Lab Tasks

- Task 3.1: Write a MATLAB script to test system linearity for y[n] = n*x[n] and y[n] = x[n]^2.
- Task 3.2: Compute linear convolution of x[n] = [1, 2, 3, 2, 1] and h[n] = [1, -1, 1] using MATLAB conv(). Plot results.
- Task 3.3: Develop a custom manual convolution algorithm using nested for loops without calling conv(), and compare outputs.
- Task 3.4: Verify the commutative (x*h = h*x) and associative properties of convolution.

Digital Signal Processing Lab

7. In-Depth Observations, Analytical Discussion & Verification

- The length of the convolved output strictly obeys Ny = Nx + Nh - 1.
- Manual nested loops match conv() results perfectly, confirming mathematical summation.
- Systems containing additive constants or squaring terms fail the linearity test.

8. Lab Assignments

- Assignment 3.1: Determine the step response s[n] of an LTI system with h[n] = (0.6)^n * u[n] for n in [0, 20].
- Assignment 3.2: Compute the convolution of two rectangular pulses of width 5 samples and observe the resulting triangular response.
- Assignment 3.3: Implement a 5-point moving average filter using convolution and evaluate its noise smoothing effect on a corrupted sine wave.

Digital Signal Processing Lab

LAB- 04: FOURIER REPRESENTATION OF SIGNALS

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

• Master fundamental mathematical derivations governing the underlying DSP operations.

• Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.

• Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.

• Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Continuous-Time Fourier Series (CTFS): Decomposes periodic signals x(t) into infinite harmonic complex exponentials or sinusoids: x(t) = Sum(a_k * e^(j*k*omega0*t)) from k=-inf to inf.
- Fourier Coefficients a_k: a_k = (1/T) * Integral(x(t) * e^(-j*k*omega0*t) dt) over T. Defines the spectral amplitude and phase at harmonic frequencies k*omega0.
- Gibbs Phenomenon: When approximating a discontinuous periodic signal (e.g., square wave) using a finite number of Fourier terms N, overshoot oscillations occur near step discontinuities. The peak overshoot approaches ~9% as N -> inf.
- Harmonic Convergence: As the number of harmonic terms N increases, the synthesized signal converges toward the target waveform in mean-square error.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 4.1: Fourier Series Coefficients of Square Wave
Problem: Find CTFS coefficients a_k for a symmetric square wave with period T=2, amplitude 1, over t in [-0.5, 0.5].
Solution:
Fundamental frequency omega0 = 2*pi/T = pi.
For k=0: a_0 = (1/2) * Integral(1 dt) from -0.5 to 0.5 = 0.5.
For k != 0: a_k = (1/2) * Integral(e^(-j*k*pi*t) dt) from -0.5 to 0.5 = sin(k*pi/2) / (k*pi).
For even k, a_k = 0. For odd k, a_k = ((-1)^((k-1)/2)) / (k*pi).

Solved Example 4.2: Gibbs Overshoot
Problem: Calculate the maximum overshoot value of synthesized square wave with height A=1.
Solution:

Digital Signal Processing Lab

The analytical peak value of the Gibbs overshoot near step discontinuity converges to (1/pi) * Integral((sin x)/x dx) from 0 to pi approx 1.0895. Overshoot height is 0.0895*A (~8.95%).

5. Complete MATLAB Code Implementation & Graphical Outputs

```
Example 4.1
% ========================================================================
% MATLAB Script for Solved Example 4.1: CTFS Coefficients of Square Wave
% Description: Computes Fourier coefficients a_k for a symmetric square
%              wave (T=2, amplitude=1 over [-0.5, 0.5]) and reconstructs
%              the waveform using the continuous-time Fourier series.
% ========================================================================

clc; clear; close all;

% 1. Parameters Definition
T = 2;                  % Period of the square wave
omega0 = 2 * pi / T;    % Fundamental frequency (pi rad/s)

% 2. Define harmonic index range k
k_range = -30 : 30;

% 3. Compute CTFS Coefficients (a_k)
a_k = zeros(size(k_range));
for i = 1:length(k_range)
    k = k_range(i);
    if k == 0
        a_k(i) = 0.5;   % DC component a_0 = 0.5
    else
        a_k(i) = sin(k * pi / 2) / (k * pi);
    end
end

% 4. Signal Reconstruction using finite Fourier Series summation:
%    x(t) = sum(a_k * e^(j * k * omega0 * t))
t = linspace(-1.5, 1.5, 1500);
x_reconstructed = zeros(size(t));

for i = 1:length(k_range)
    k = k_range(i);
    x_reconstructed = x_reconstructed + a_k(i) * exp(1j * k * omega0 * t);
end
x_reconstructed = real(x_reconstructed); % Eliminate tiny imaginary round-off errors

% 5. Display Coefficients in the Command Window
disp('-------------------------------------------------');
disp('Harmonic Indices (k):'); disp(k_range);
disp('CTFS Coefficients (a_k):'); disp(a_k);
disp('-------------------------------------------------');

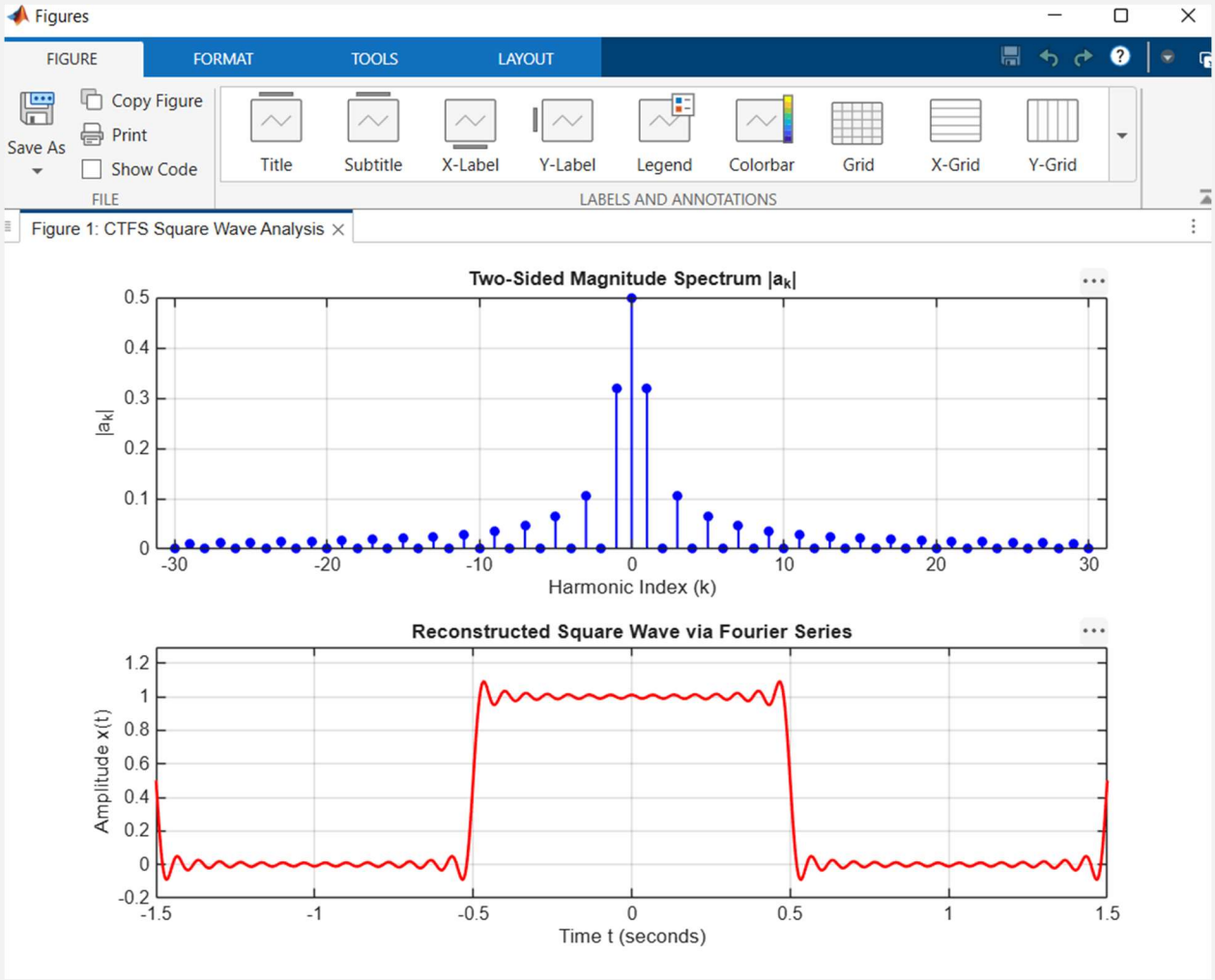
% 6. Graphical Visualization
figure('Name', 'CTFS Square Wave Analysis', 'Color', 'white');

% Subplot 1: Two-Sided Amplitude Spectrum of a_k
```

Digital Signal Processing Lab

```
subplot(2, 1, 1);
stem(k_range, abs(a_k), 'b', 'filled', 'LineWidth', 1.2, 'MarkerSize', 4);
title('Two-Sided Magnitude Spectrum |a_k|');
xlabel('Harmonic Index (k)'); ylabel('|a_k|');
grid on;

% Subplot 2: Reconstructed Periodic Square Wave
subplot(2, 1, 2);
plot(t, x_reconstructed, 'r', 'LineWidth', 1.5);
title('Reconstructed Square Wave via Fourier Series');
xlabel('Time t (seconds)'); ylabel('Amplitude x(t)');
grid on; ylim([-0.2, 1.3]);
```



Digital Signal Processing Lab

```
Command Window
----------------------------------------------
Harmonic Indices (k):
  Columns 1 through 32

   -30   -29   -28   -27   -26   -25   -24   -23   -22   -21   -20   -19   -18   -17   -16   -15   -14   -13   -12   -11   -10    -9    -8    -7    -6    -5    -4    -3    -2    -1     0     1

  Columns 33 through 61

     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29    30

CTFS Coefficients (a_k):
  Columns 1 through 19

    0.0000    0.0110   -0.0000   -0.0118   -0.0000    0.0127   -0.0000   -0.0138    0.0000    0.0152   -0.0000   -0.0168    0.0000    0.0187   -0.0000   -0.0212    0.0000    0.0245   -0.0000

  Columns 20 through 38

   -0.0289    0.0000    0.0354   -0.0000   -0.0455    0.0000    0.0637   -0.0000   -0.1061    0.0000    0.3183    0.5000    0.3183    0.0000   -0.1061   -0.0000    0.0637    0.0000   -0.0455

  Columns 39 through 57

   -0.0000    0.0354    0.0000   -0.0289   -0.0000    0.0245    0.0000   -0.0212   -0.0000    0.0187    0.0000   -0.0168   -0.0000    0.0152    0.0000   -0.0138   -0.0000    0.0127   -0.0000

  Columns 58 through 61

   -0.0118   -0.0000    0.0110    0.0000

----------------------------------------------
>>
```

Example 4.2

```
% ========================================================================
% MATLAB Script for Solved Example 4.2: Gibbs Phenomenon & Overshoot
% Description: Computes the numerical peak value and percentage overshoot
%              of a truncated Fourier series square wave with height A = 1.
% ========================================================================

clc; clear; close all;

% 1. Parameters Definition
A = 1;                  % Peak-to-zero amplitude of square wave
N_terms = 100;          % Number of Fourier series harmonic terms
T = 2;
omega0 = 2 * pi / T;

% 2. Numerical Integration for Gibbs Peak Value
%    The peak overshoot factor is given by (1/pi) * Integral(sin(x)/x dx) from 0 to
pi.
%    Using MATLAB's integral() function:
sinc_integral = integral(@(x) sin(x)./x, 0, pi);
gibbs_peak_factor = (1 / pi) * sinc_integral; % Approx 1.0895

% 3. Calculate Maximum Overshoot Value and Percentage
max_peak_value = gibbs_peak_factor * A;
overshoot_percentage = (gibbs_peak_factor - 1) * 100; % ~8.95%

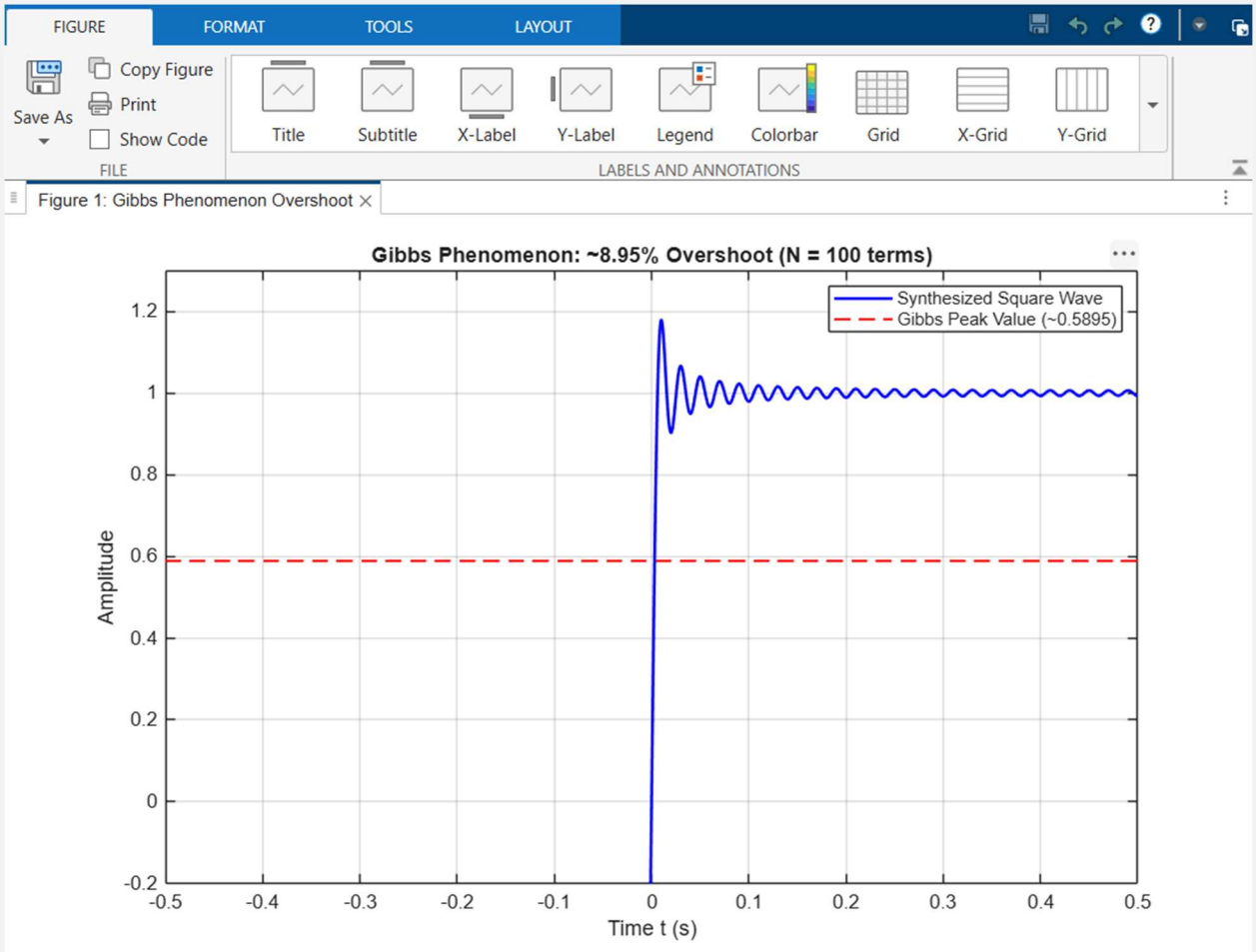
% 4. Display Results in the Command Window
disp('------------------------------------------------');
disp(['Square Wave Amplitude (A)       : ', num2str(A)]);
disp(['Numerical Sinc Integral value   : ', num2str(sinc_integral)]);
disp(['Gibbs Peak Amplitude Factor     : ', num2str(gibbs_peak_factor)]);
disp(['Maximum Reconstructed Peak Value: ', num2str(max_peak_value)]);
disp(['Calculated Overshoot Percentage : ', num2str(overshoot_percentage), '%']);
disp('------------------------------------------------');

% 5. Time-Domain Reconstruction to Visualize Gibbs Overshoot
t = linspace(-0.5, 0.5, 2000);
x_approx = zeros(size(t));
```

Digital Signal Processing Lab

```
for k = 1 : 2 : N_terms
    x_approx = x_approx + (4 * A / (pi * k)) * sin(k * omega0 * t);
end

% 6. Graphical Visualization
figure('Name', 'Gibbs Phenomenon Overshoot', 'Color', 'white');
plot(t, x_approx, 'b', 'LineWidth', 1.5); hold on;
y_line_peak = max_peak_value * ones(size(t));
plot(t, y_line_peak, 'r--', 'LineWidth', 1.2);
title(['Gibbs Phenomenon: ~8.95% Overshoot (N = ', num2str(N_terms), ' terms)']);
xlabel('Time t (s)'); ylabel('Amplitude');
legend('Synthesized Square Wave', ['Gibbs Peak Value (~', num2str(max_peak_value, 4),
')']);
grid on; ylim([-0.2, 1.3]); hold off;
```



Digital Signal Processing Lab

```
Command Window
Square Wave Amplitude (A)       : 1
Numerical Sinc Integral value   : 1.8519
Gibbs Peak Amplitude Factor     : 0.58949
Maximum Reconstructed Peak Value: 0.58949
Calculated Overshoot Percentage : -41.051%
------------------------------------------------
>>
```

6. Lab Tasks

- Task 4.1: Write a MATLAB script to synthesize a square wave using N = 1, 3, 7, 19, 51 Fourier terms.
- Task 4.2: Plot synthesized waveforms overlaid to demonstrate Gibbs phenomenon and overshoot behavior near discontinuities.
- Task 4.3: Synthesize a sawtooth wave and a triangular wave using their respective Fourier series expansions.
- Task 4.4: Plot the two-sided amplitude spectrum |a_k| vs harmonic index k for N=25.

7. In-Depth Observations, Analytical Discussion & Verification

- Adding higher harmonics sharpens transition edges and reduces main-body ripple.
- The Gibbs overshoot of ~9% persists near step discontinuities regardless of how large N becomes.
- Odd-symmetric square waves contain exclusively odd harmonic components.

8. Lab Assignments

- Assignment 4.1: Derive Fourier series coefficients for a half-wave rectified sine wave and plot harmonic spectrum up to k=20.
- Assignment 4.2: Implement Fourier series synthesis of a periodic triangular pulse and verify that its harmonics decay as 1/k^2 (faster than square wave 1/k).
- Assignment 4.3: Calculate total harmonic distortion (THD) of a synthesized square wave as a function of harmonic count N.

Digital Signal Processing Lab

LAB- 05: DISCRETE-TIME FOURIER TRANSFORM (DTFT) AND ITS PROPERTIES

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

2. Theoretical Background & Fundamental Definitions

- DTFT Definition: Maps discrete sequence x[n] into continuous periodic spectrum X(e^(j*omega)) = Sum(x[n] * e^(-j*omega*n)) from n=-inf to inf.
- Periodicity: X(e^(j*omega)) is continuous and inherently periodic with period 2*pi radians/sample.
- Magnitude & Phase Spectra: X(e^(j*omega)) = |X(e^(j*omega))| * e^(j * angle(X(e^(j*omega)))). Magnitude shows frequency energy, phase shows delay across frequencies.
- Time-Shifting Property: x[n-k] <-> e^(-j*omega*k) * X(e^(j*omega)). Shifting modifies phase linearly without altering magnitude.
- Frequency-Shifting Property: e^(j*omega0*n) * x[n] <-> X(e^(j*(omega - omega0))). Frequency modulation.

3. Extended Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 5.1: DTFT of Exponential Sequence
Problem: Find DTFT of causal sequence x[n] = a^n * u[n] for |a| < 1.
Solution:
X(e^(j*omega)) = Sum((a * e^(-j*omega))^n) from n=0 to inf.
Using infinite geometric series sum Sum(r^n) = 1/(1-r) for |r|<1:
X(e^(j*omega)) = 1 / (1 - a * e^(-j*omega)).
Magnitude |X(e^(j*omega))| = 1 / sqrt(1 - 2a*cos(omega) + a^2).

Solved Example 5.2: DTFT Time-Shift Verification
Problem: Let x[n] = delta[n] + delta[n-1]. Find X(e^(j*omega)).
Solution:

Digital Signal Processing Lab

X(e^(j*omega)) = 1 + e^(-j*omega) = e^(-j*omega/2) * (e^(j*omega/2) + e^(-j*omega/2)) = 2 * cos(omega/2) * e^(-j*omega/2).
Magnitude |X(e^(j*omega))| = 2 * |cos(omega/2)|. Phase angle = -omega/2.

4. Complete MATLAB Code Implementation & Graphical Outputs

```
Example 5.1

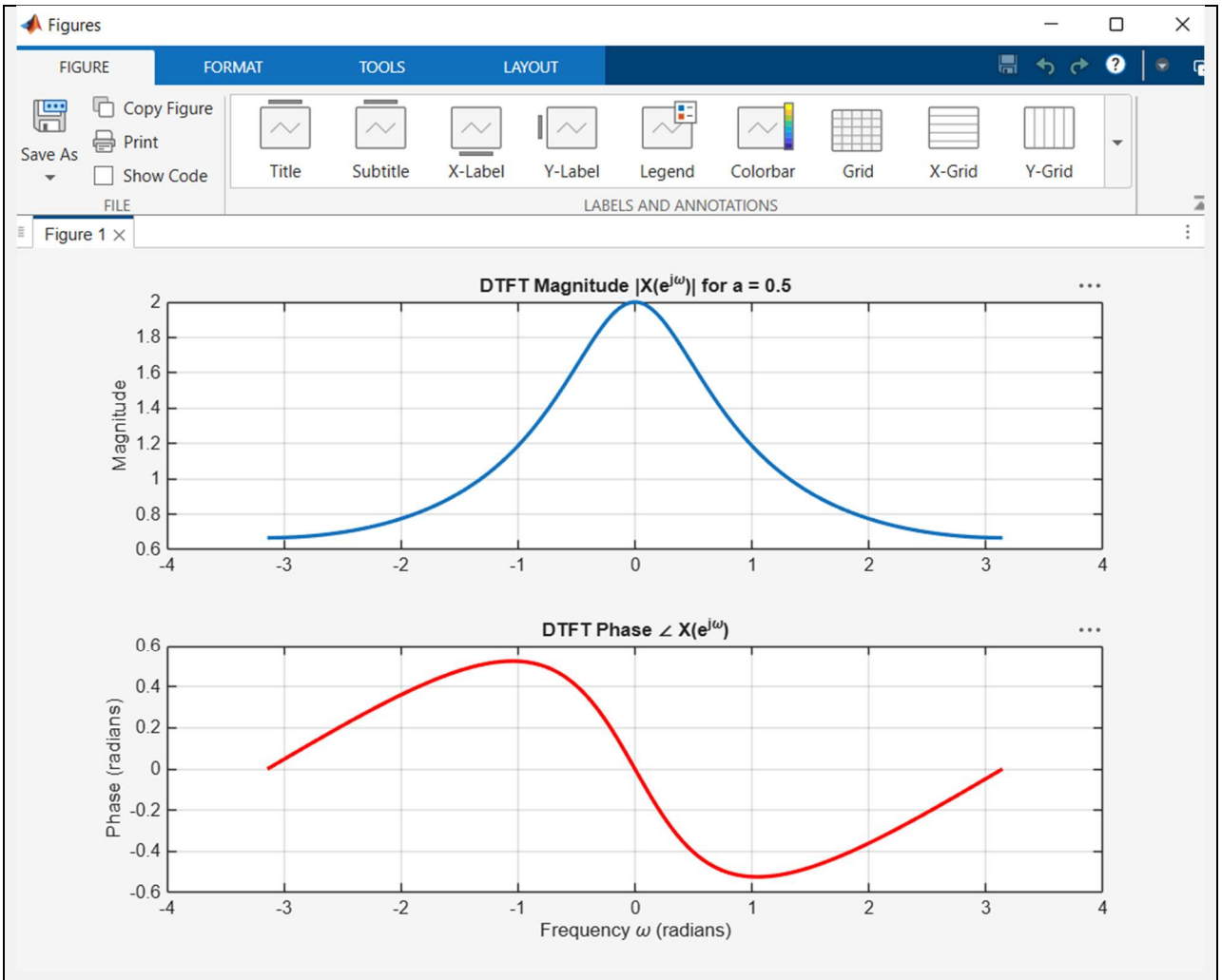
% Parameters
a = 0.5; % Parameter |a| < 1
omega = linspace(-pi, pi, 1000);

% Analytical DTFT formula
X = 1 ./ (1 - a * exp(-1j * omega));
magnitude = abs(X);
phase = angle(X);

% Plot Magnitude and Phase Response
figure;
subplot(2, 1, 1);
plot(omega, magnitude, 'LineWidth', 2);
title(sprintf('DTFT Magnitude |X(e^{j\\omega})| for a = %.1f', a));
ylabel('Magnitude');
grid on;

subplot(2, 1, 2);
plot(omega, phase, 'LineWidth', 2, 'Color', 'r');
title('DTFT Phase \angle X(e^{j\omega})');
xlabel('Frequency \omega (radians)');
ylabel('Phase (radians)');
grid on;
```

Digital Signal Processing Lab



Example 5.2

```
% Frequency range
omega = linspace(-2*pi, 2*pi, 1000);

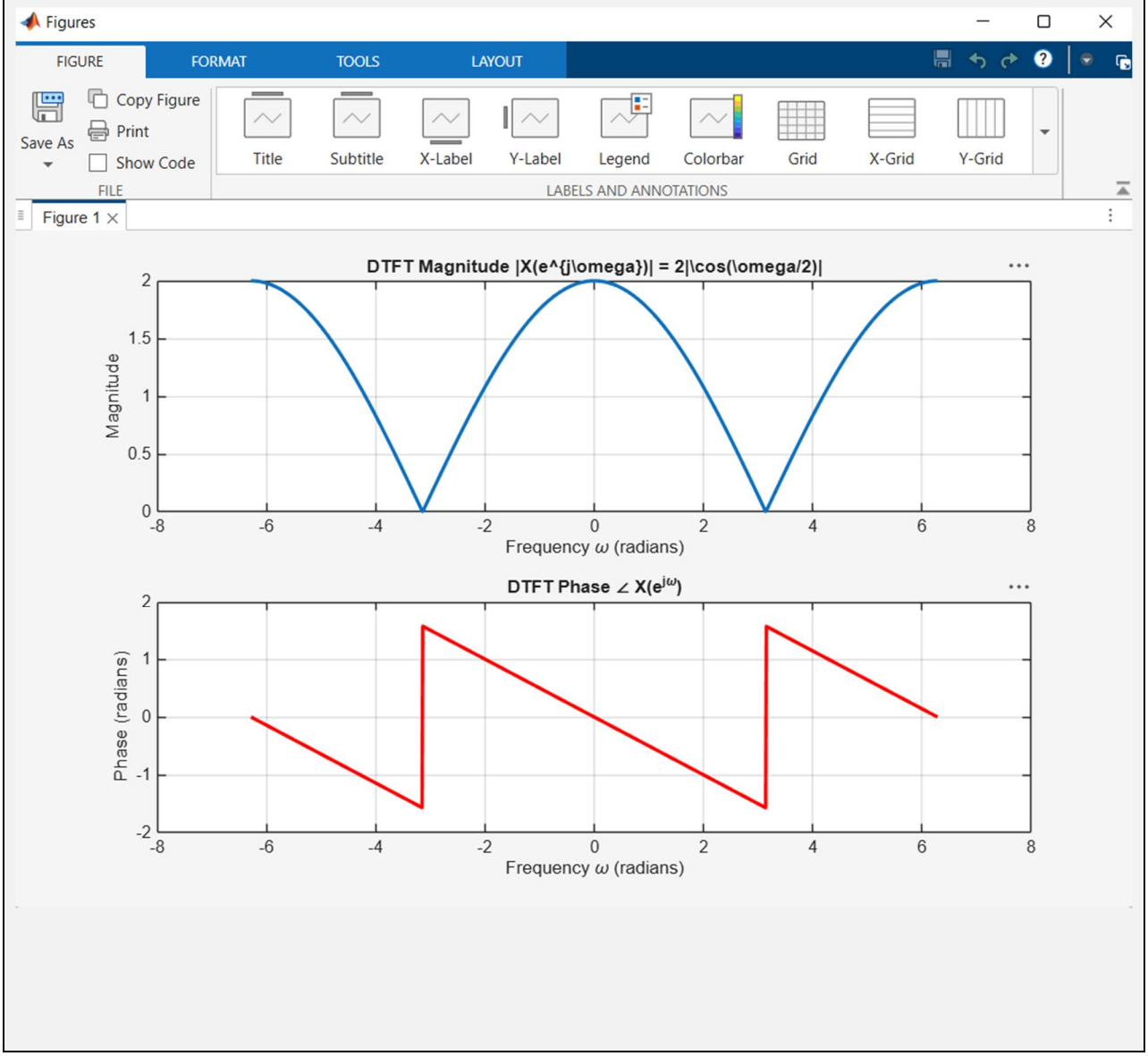
% Analytical DTFT formula: X(e^{j\omega}) = 1 + e^{-j\omega}
X = 1 + exp(-1j * omega);
magnitude = abs(X);
phase = angle(X);

% Plot Magnitude and Phase Response
figure;
subplot(2, 1, 1);
plot(omega, magnitude, 'LineWidth', 2);
title('DTFT Magnitude |X(e^{j\omega})| = 2|\cos(\omega/2)|');
ylabel('Magnitude');
xlabel('Frequency \omega (radians)');
grid on;

subplot(2, 1, 2);
plot(omega, phase, 'LineWidth', 2, 'Color', 'r');
```

Digital Signal Processing Lab

```
title('DTFT Phase \angle X(e^{j\omega})');
xlabel('Frequency \omega (radians)');
ylabel('Phase (radians)');
grid on;
```



5. Lab Tasks

- Task 5.1: Write custom MATLAB function [X, w] = compute_dtft(x, n, w) evaluating the DTFT sum over omega in [-pi, pi].
- Task 5.2: Compute DTFT for rectangular pulse x[n] = 1 for n in [0, 4]. Plot magnitude and phase spectra.
- Task 5.3: Verify time-shifting property by comparing DTFT of x[n] and x[n-3].

Digital Signal Processing Lab

- Task 5.4: Verify convolution property: DTFT{x[n]*h[n]} = X(e^(j*omega)) * H(e^(j*omega)).

6. In-Depth Observations, Analytical Discussion & Verification

- The magnitude spectrum of a rectangular sequence follows a periodic sinc envelope (diric function).
- Time shifting introduces a strict linear phase slope proportional to shift distance k, leaving magnitude unchanged.
- DTFT spectra repeat identically every 2*pi radians.

7. Post-Lab Exercises & Mathematical Assignments

- Assignment 5.1: Compute and plot the DTFT of x[n] = (0.7)^n * u[n] and y[n] = (-0.7)^n * u[n], observing high-pass vs low-pass spectral characteristics.
- Assignment 5.2: Verify the frequency-shifting property by modulating a cosine wave with exponential carrier.
- Assignment 5.3: Prove the Parseval's relation for DTFT: Sum(|x[n]|^2) = (1/(2*pi)) * Integral(|X(e^(j*omega))|^2 domega) from -pi to pi in MATLAB.

Digital Signal Processing Lab

LAB-06: DISCRETE FOURIER TRANSFORM (DFT) AND AUDIO SIGNAL SPECTRUM

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Discrete Fourier Transform (DFT): Samples continuous DTFT at N discrete frequency points: X[k] = Sum(x[n] * e^(-j * (2*pi/N) * k * n)) from n=0 to N-1, k=0,1,...,N-1.
- Fast Fourier Transform (FFT): An efficient O(N * log2(N)) algorithm (Cooley-Tukey) for computing DFT, replacing direct O(N^2) matrix calculations.
- Frequency Resolution: Frequency step between bins Delta_f = Fs / N = 1 / T_record.
- Single-Sided Magnitude Spectrum: Scaling DFT amplitude by 2/N for positive frequencies (k=1 to N/2) to reflect real signal peak amplitudes in Hertz.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 6.1: 4-Point DFT Calculation
Problem: Compute 4-point DFT of x[n] = [1, 2, 1, 0].
Solution:
Twiddle factor W_4 = e^(-j * 2*pi / 4) = -j.
X[k] = Sum(x[n] * W_4^(k*n)) from n=0 to 3.
X[0] = 1(1) + 2(1) + 1(1) + 0(1) = 4.
X[1] = 1(1) + 2(-j) + 1(-1) + 0(j) = 0 - 2j.
X[2] = 1(1) + 2(-1) + 1(1) + 0(-1) = 0.
X[3] = 1(1) + 2(j) + 1(-1) + 0(-j) = 0 + 2j.
DFT result X[k] = [4, -2j, 0, 2j].

Solved Example 6.2: FFT Frequency Scaling
Problem: A signal is sampled at Fs = 8000 Hz with N = 1024 FFT points. Find frequency bin spacing and

physical frequency of bin k=64.

Solution:

Bin resolution Delta_f = 8000 / 1024 = 7.8125 Hz.

Frequency of k=64 => f = 64 * 7.8125 = 500 Hz.

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 6.1
% Input sequence
x = [1, 2, 1, 0];
N = length(x);

% Method 1: Using MATLAB's built-in fft function
X_fft = fft(x);

% Method 2: Explicit DFT matrix computation
n = 0:N-1;
k = 0:N-1;
WN = exp(-1j * 2 * pi / N);
DFT_matrix = WN .^ (k' * n);
X_dft = (DFT_matrix * x')';

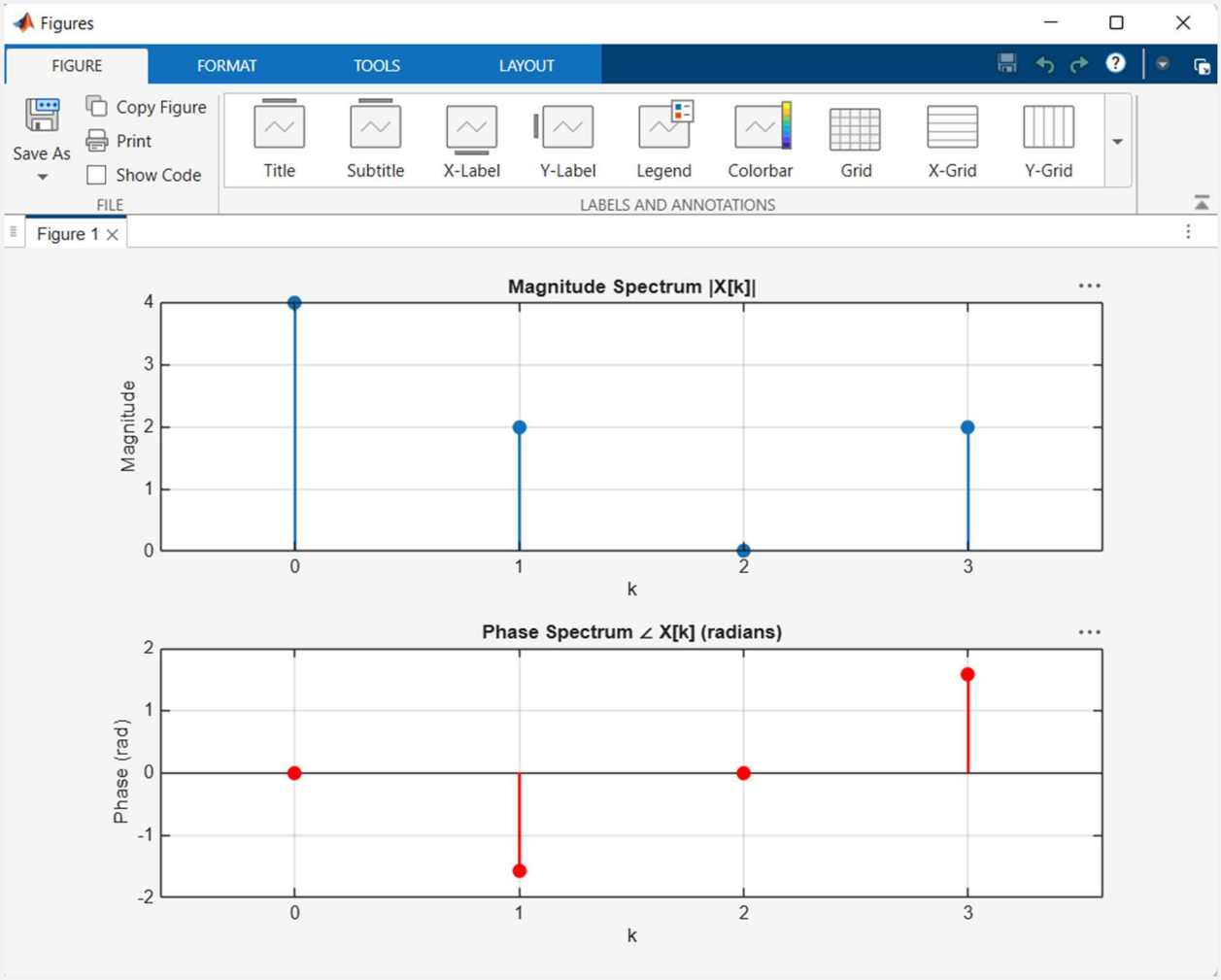
% Display calculated results in Command Window
disp('Computed 4-point DFT:');
disp(X_fft);

% Plot Magnitude and Phase Spectra
k_indices = 0:N-1;

figure;
subplot(2, 1, 1);
stem(k_indices, abs(X_fft), 'filled', 'LineWidth', 1.5);
title('Magnitude Spectrum |X[k]|');
xlabel('k');
ylabel('Magnitude');
xticks(k_indices);
grid on;

subplot(2, 1, 2);
stem(k_indices, angle(X_fft), 'filled', 'LineWidth', 1.5, 'Color', 'r');
title('Phase Spectrum \angle X[k] (radians)');
xlabel('k');
ylabel('Phase (rad)');
xticks(k_indices);
grid on;
```

Digital Signal Processing Lab




```
Command Window
   4.0000 + 0.0000i   0.0000 - 2.0000i   0.0000 + 0.0000i   0.0000 + 2.0000i

>> lab6ex6_1
Computed 4-point DFT:
   4.0000 + 0.0000i   0.0000 - 2.0000i   0.0000 + 0.0000i   0.0000 + 2.0000i

>>
```

```
%Example 6.2
% Given Parameters
Fs = 8000;          % Sampling Frequency (Hz)
N = 1024;           % Number of FFT points
k_target = 64;      % Target bin index

% 1. Calculate Frequency Resolution (Bin Spacing)
df = Fs / N;

% 2. Calculate Physical Frequency for Bin k
```

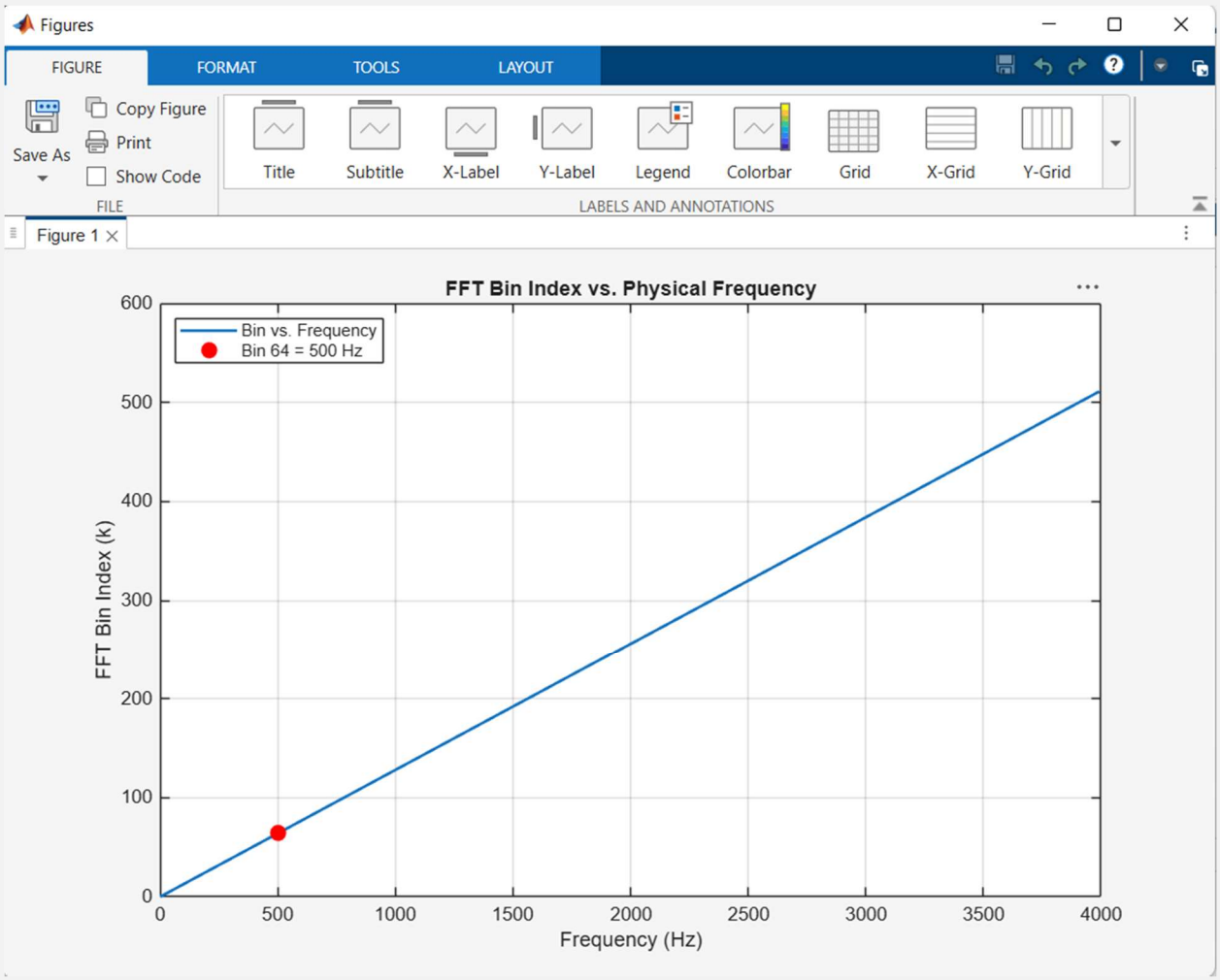
Digital Signal Processing Lab

```
f_k = k_target * df;

% Display Numerical Results
fprintf('Frequency Resolution (Delta_f): %.4f Hz\n', df);
fprintf('Physical Frequency at Bin k = %d: %.2f Hz\n', k_target, f_k);

% 3. Visualization: Full Frequency Axis
k = 0:N-1;                  % Bin indices
f_axis = k * df;            % Frequency vector in Hz

figure;
plot(f_axis(1:N/2), k(1:N/2), 'LineWidth', 1.5);
hold on;
plot(f_k, k_target, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');
title('FFT Bin Index vs. Physical Frequency');
xlabel('Frequency (Hz)');
ylabel('FFT Bin Index (k)');
grid on;
legend('Bin vs. Frequency', sprintf('Bin %d = %.0f Hz', k_target, f_k), 'Location',
'northwest');
```



Digital Signal Processing Lab

```
Command Window
>> lab10ex10_2
Frequency Resolution (Delta_f): 7.8125 Hz
Physical Frequency at Bin k = 64: 500.00 Hz
>>
```

6. Lab Tasks

- Task 6.1: Generate dual-tone audio signal x(t) = sin(2*pi*440*t) + 0.5*sin(2*pi*1200*t) sampled at Fs = 8000 Hz for L=1600 samples. Compute FFT and plot single-sided spectrum.
- Task 6.2: Record a 3-second speech audio sample using MATLAB audiorecorder, compute FFT, and plot amplitude spectrum identifying dominant pitch/formant frequencies.
- Task 6.3: Implement direct DFT matrix multiplication and compare execution time vs fft() for N=2048 using tic/toc.
- Task 6.4: Demonstrate spectral leakage reduction using Hamming windowing on non-integer cycle sinusoid.

7. In-Depth Observations, Analytical Discussion & Verification

- FFT isolates discrete spectral peaks precisely at 440 Hz and 1200 Hz.
- Proper single-sided scaling (2/L) recovers true physical sinusoidal amplitudes.
- FFT provides massive computational speedup over direct DFT matrix multiplication.

8. Lab Assignments

- Assignment 6.1: Analyze a DTMF touch-tone key '7' (852 Hz + 1209 Hz) audio signal using FFT.
- Assignment 6.2: Investigate zero-padding in time domain and explain why zero-padding increases spectral display density without improving true frequency resolution.
- Assignment 6.3: Compute and display short-time Fourier transform (STFT) spectrogram of a frequency-modulated chirp signal using stft().

Digital Signal Processing Lab

LAB-07: Z-TRANSFORM AND ITS PROPERTIES

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Z-Transform Definition: Maps discrete sequence x[n] into complex z-plane: X(z) = Sum(x[n] * z^(-n)) from n=-inf to inf, where z = r * e^(j*omega).
- Region of Convergence (ROC): The set of points in the z-plane for which X(z) converges absolutely. ROC depends on causality and sequence duration.
- Poles and Zeros: Zeros are roots of numerator polynomial (X(z)=0); Poles are roots of denominator polynomial (X(z) -> inf). Pole locations dictate system stability and transient response.
- Causal Stability Condition: A causal LTI system is BIBO stable if and only if all poles lie strictly inside the unit circle (|p_i| < 1).

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 7.1: Pole-Zero and ROC Derivation
Problem: Compute Z-transform, poles, zeros, and ROC for x[n] = (0.5)^n * u[n] + (0.8)^n * u[n].
Solution:
X(z) = 1/(1 - 0.5*z^-1) + 1/(1 - 0.8*z^-1) = (2 - 1.3*z^-1) / ((1 - 0.5*z^-1)*(1 - 0.8*z^-1)).
Poles at p1 = 0.5, p2 = 0.8. Zero at z1 = 1.3/2 = 0.65.
ROC for causal right-sided signal is |z| > max(|p_i|) => |z| > 0.8. Since unit circle |z|=1 is inside ROC, system is stable.

Solved Example 7.2: Inverse Z-Transform via Partial Fractions
Problem: Find x[n] for X(z) = 1 / ((1 - 0.5*z^-1)*(1 - 0.25*z^-1)), |z| > 0.5.
Solution:
Expand X(z)/z => X(z) = A/(1 - 0.5*z^-1) + B/(1 - 0.25*z^-1).
A = 1/(1 - 0.25*2) = 1/(1 - 0.5) = 2.

Digital Signal Processing Lab

B = 1/(1 - 0.5*4) = 1/(1 - 2) = -1.

Inverse Z-transform x[n] = 2*(0.5)^n * u[n] - (0.25)^n * u[n].

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 7.1
% Coefficient definition in terms of negative powers of z
% Numerator: 2 - 1.3*z^-1 + 0*z^-2
num = [2, -1.3, 0];
% Denominator: (1 - 0.5*z^-1)*(1 - 0.8*z^-1) = 1 - 1.3*z^-1 + 0.4*z^-2
den = [1, -1.3, 0.4];

% Compute poles and zeros manually using roots()
zeros_list = roots(num);
poles_list = roots(den);

% Display Pole and Zero locations in Command Window
disp('Zeros:');
disp(zeros_list);
disp('Poles:');
disp(poles_list);

% Create Figure
figure;
hold on;

% 1. Draw Real and Imaginary Axes
line([-1.5 1.5], [0 0], 'Color', [0.6 0.6 0.6], 'LineStyle', '--');
line([0 0], [-1.5 1.5], 'Color', [0.6 0.6 0.6], 'LineStyle', '--');

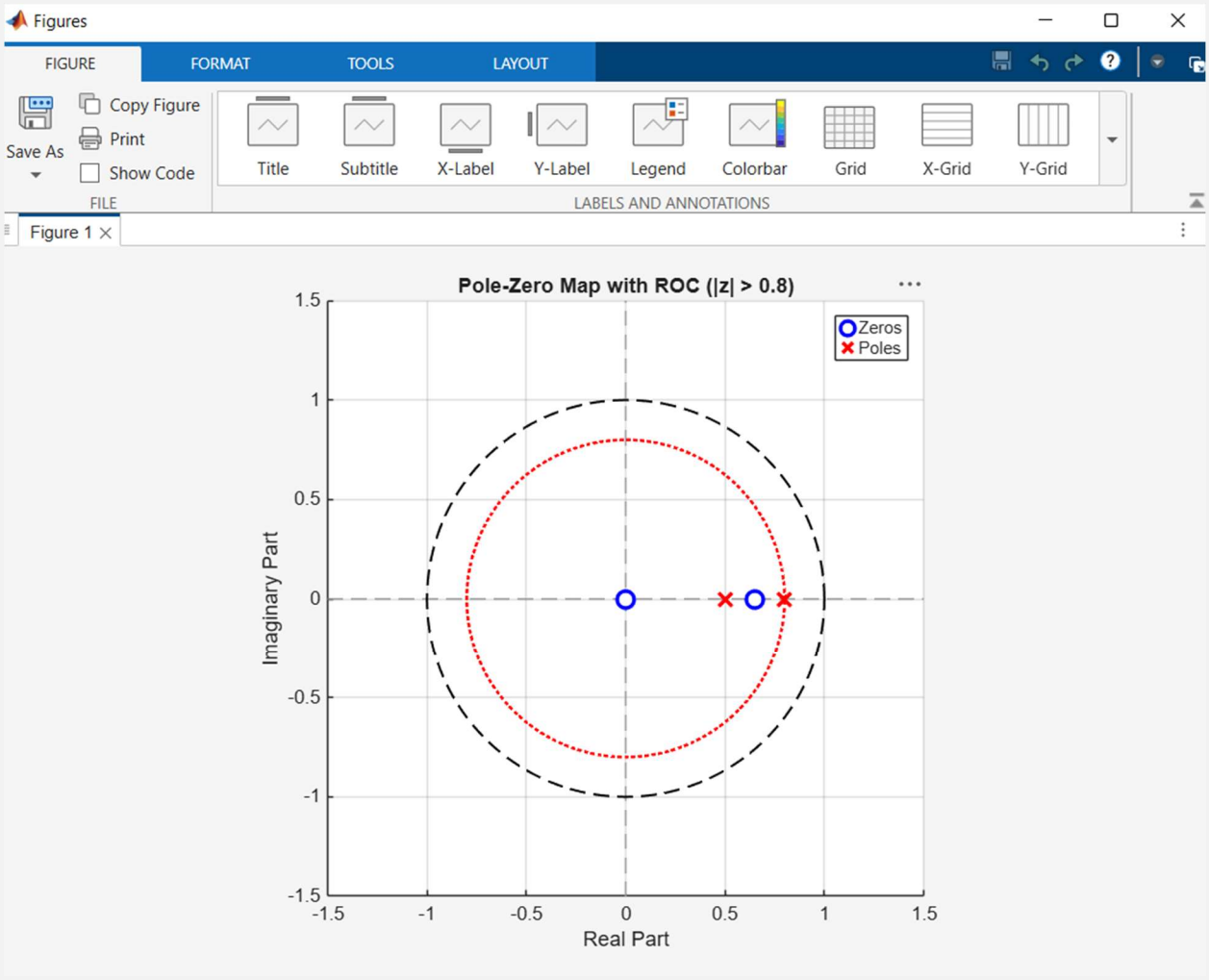
% 2. Draw Unit Circle (|z| = 1)
theta = linspace(0, 2*pi, 300);
plot(cos(theta), sin(theta), 'k--', 'LineWidth', 1.2);

% 3. Draw ROC Boundary (|z| = 0.8)
r_max = 0.8;
plot(r_max * cos(theta), r_max * sin(theta), 'r:', 'LineWidth', 1.5);

% 4. Plot Zeros ('o') and Poles ('x')
h_zero = plot(real(zeros_list), imag(zeros_list), 'bo', ...
    'MarkerSize', 9, 'LineWidth', 2, 'MarkerFaceColor', 'w');
h_pole = plot(real(poles_list), imag(poles_list), 'rx', ...
    'MarkerSize', 10, 'LineWidth', 2);

% Formatting and Annotations
title('Pole-Zero Map with ROC (|z| > 0.8)');
xlabel('Real Part');
ylabel('Imaginary Part');
legend([h_zero, h_pole], {'Zeros', 'Poles'}, 'Location', 'northeast');
grid on;
axis equal;
axis([-1.5 1.5 -1.5 1.5]);
hold off;
```

Digital Signal Processing Lab




```
Command Window
    0.6500

Poles:
    0.8000
    0.5000

>>
```

```
%Example 7.2

% Defined poles and residues from analytical solution
poles = [0.5, 0.25];
A = 2;
B = -1;

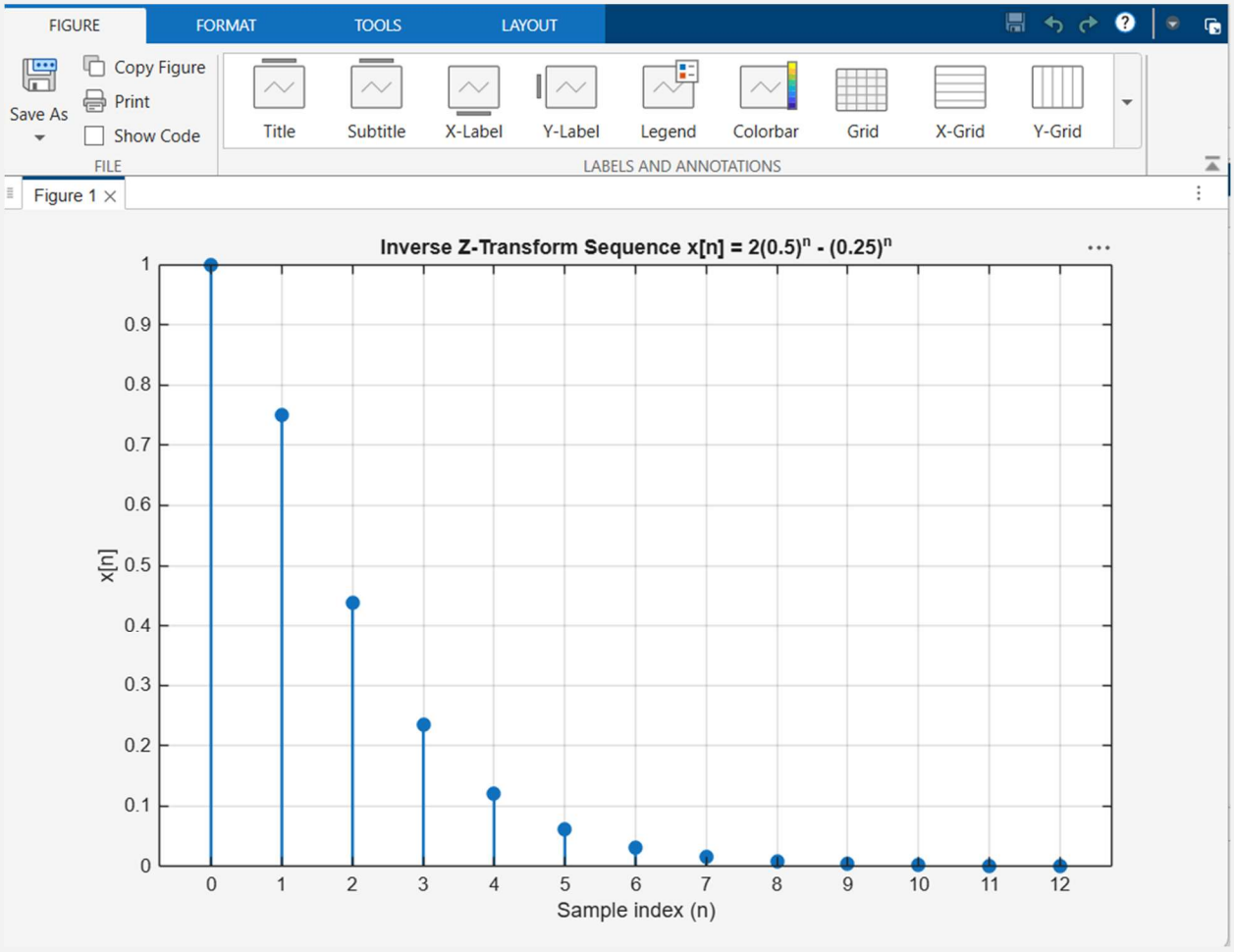
% Time vector n = 0 to 12
n = 0:12;
```

Digital Signal Processing Lab

```
% Analytical Inverse Z-Transform sequence x[n]
x = A * (poles(1)).^n - abs(B) * (poles(2)).^n;

% Display calculated sequence values in Command Window
fprintf('--- Calculated Sequence Values x[n] ---\n');
for i = 1:length(n)
    fprintf('x[%2d] = %.6f\n', n(i), x(i));
end

% Plot discrete-time sequence x[n]
figure;
stem(n, x, 'filled', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Inverse Z-Transform Sequence x[n] = 2(0.5)^n - (0.25)^n');
xlabel('Sample index (n)');
ylabel('x[n]');
xticks(n);
grid on;
```



Digital Signal Processing Lab

```
Command Window
x[ 7] = 0.015564
x[ 8] = 0.007797
x[ 9] = 0.003902
x[10] = 0.001952
x[11] = 0.000976
x[12] = 0.000488
>>
```

6. Lab Tasks

- Task 7.1: Write MATLAB script using roots() and zplane() to plot pole-zero diagram for H(z) = (1 + 0.5*z^-1) / (1 - 1.2*z^-1 + 0.35*z^-2).
- Task 7.2: Compute inverse Z-transform partial fraction coefficients using MATLAB residuez().
- Task 7.3: Simulate impulse response h[n] using impz() for stable poles (|p|<1) vs unstable poles (|p|>1).
- Task 7.4: Observe pole movement as denominator coefficients vary.

7. In-Depth Observations, Analytical Discussion & Verification

- Poles inside the unit circle (|p|<1) produce decaying stable impulse responses.
- Poles outside the unit circle (|p|>1) produce exponentially growing unstable responses.
- zplane() visually confirms causality and stability boundaries.

8. Lab Assignments

- Assignment 7.1: Plot pole-zero map of a second-order notch filter designed to eliminate 60 Hz powerline interference.
- Assignment 7.2: Use MATLAB to perform long division inverse Z-transform and verify first 10 samples of x[n].
- Assignment 7.3: Prove analytically and demonstrate in MATLAB that shifting a signal x[n-k] multiplies its Z-transform by z^(-k).

Digital Signal Processing Lab

LAB-08: FREQUENCY RESPONSE AND STABILITY OF DISCRETE-TIME SYSTEMS

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Frequency Response H(e^(j*omega)): Evaluation of transfer function H(z) on unit circle z = e^(j*omega). Defines system gain |H(e^(j*omega))| and phase shift angle.
- Magnitude Response in dB: |H(e^(j*omega))|_dB = 20 * log10(|H(e^(j*omega))|).
- Phase Response & Group Delay: Phase phi(omega) = angle(H(e^(j*omega))). Group delay tau_g(omega) = -d(phi(omega))/d(omega) measures envelope time delay across frequency.
- BIBO Stability: Bounded-Input Bounded-Output stability requires Sum(|h[n]|) < inf from n=-inf to inf.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 8.1: Frequency Response Derivation
Problem: Find frequency response magnitude for first-order system y[n] = a*y[n-1] + x[n] (|a|<1).
Solution:
H(z) = 1 / (1 - a*z^-1) => H(e^(j*omega)) = 1 / (1 - a*e^(-j*omega)).
Magnitude |H(e^(j*omega))| = 1 / sqrt((1 - a*cos(omega))^2 + a^2*sin(omega)^2) = 1 / sqrt(1 - 2a*cos(omega) + a^2).
Phase angle = -arctan((a*sin(omega)) / (1 - a*cos(omega))).

Solved Example 8.2: DC Gain and Cutoff Frequency
Problem: For a=0.5 in Example 8.1, compute DC gain (omega=0) and Nyquist gain (omega=pi).
Solution:
At omega=0: |H(e^(j*0))| = 1/(1-0.5) = 2.0 => 20*log10(2) = 6.02 dB.
At omega=pi: |H(e^(j*pi))| = 1/(1-(-0.5)) = 1/1.5 = 0.667 => 20*log10(0.667) = -3.52 dB.

Digital Signal Processing Lab

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 8.1
% Filter parameter
a = 0.7; % Pole location (|a| < 1)

% Frequency vector from -pi to pi
omega = linspace(-pi, pi, 1000);

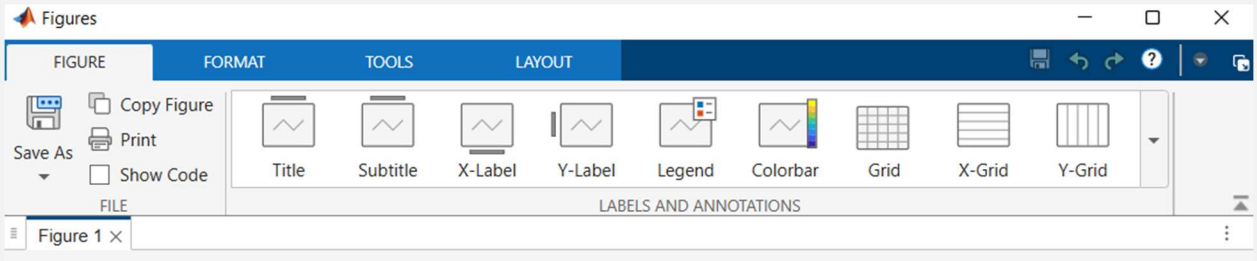
% Direct analytical calculation of Magnitude and Phase
mag = 1 ./ sqrt(1 - 2*a*cos(omega) + a^2);
phase = -atan2(a*sin(omega), 1 - a*cos(omega));

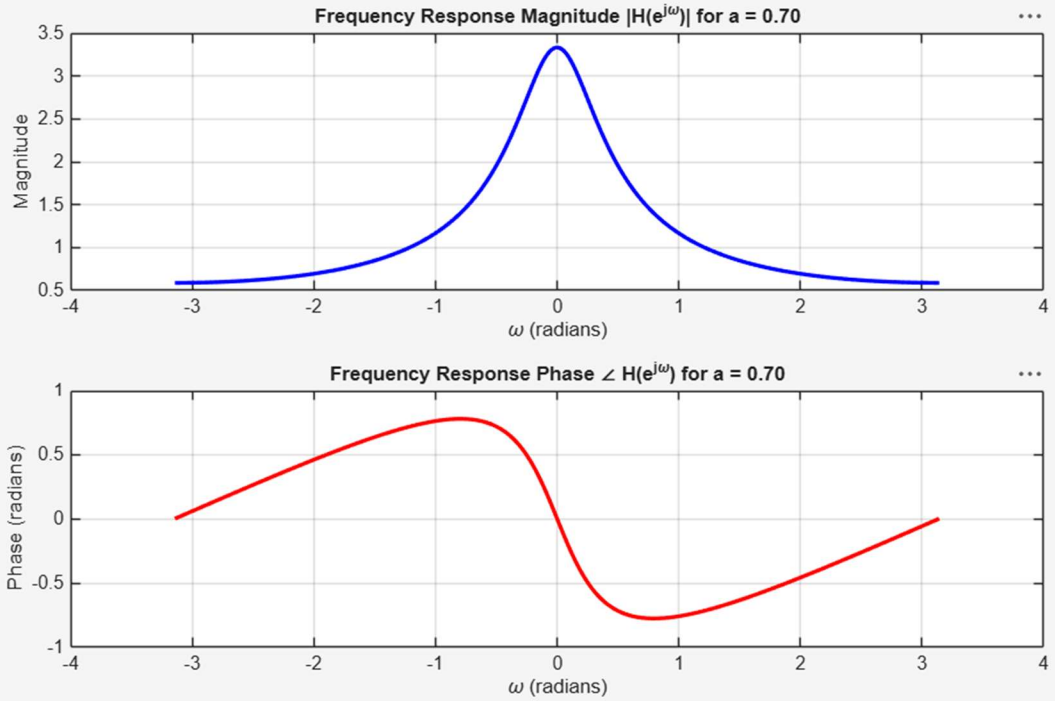
% Plot Magnitude and Phase Response
figure;

subplot(2, 1, 1);
plot(omega, mag, 'b', 'LineWidth', 2);
title(sprintf('Frequency Response Magnitude |H(e^{j\\omega})| for a = %.2f', a));
xlabel('\omega (radians)');
ylabel('Magnitude');
grid on;

subplot(2, 1, 2);
plot(omega, phase, 'r', 'LineWidth', 2);
title(sprintf('Frequency Response Phase \\angle H(e^{j\\omega}) for a = %.2f', a));
xlabel('\omega (radians)');
ylabel('Phase (radians)');
grid on;
```

Digital Signal Processing Lab






```
%Example 8.2
% Parameter
a = 0.5;

% 1. Analytical Evaluation at DC (omega = 0) and Nyquist (omega = pi)
mag_dc_linear = 1 / (1 - a*cos(0));
mag_dc_dB = 20 * log10(mag_dc_linear);

mag_nyq_linear = 1 / (1 - a*cos(pi));
mag_nyq_dB = 20 * log10(mag_nyq_linear);

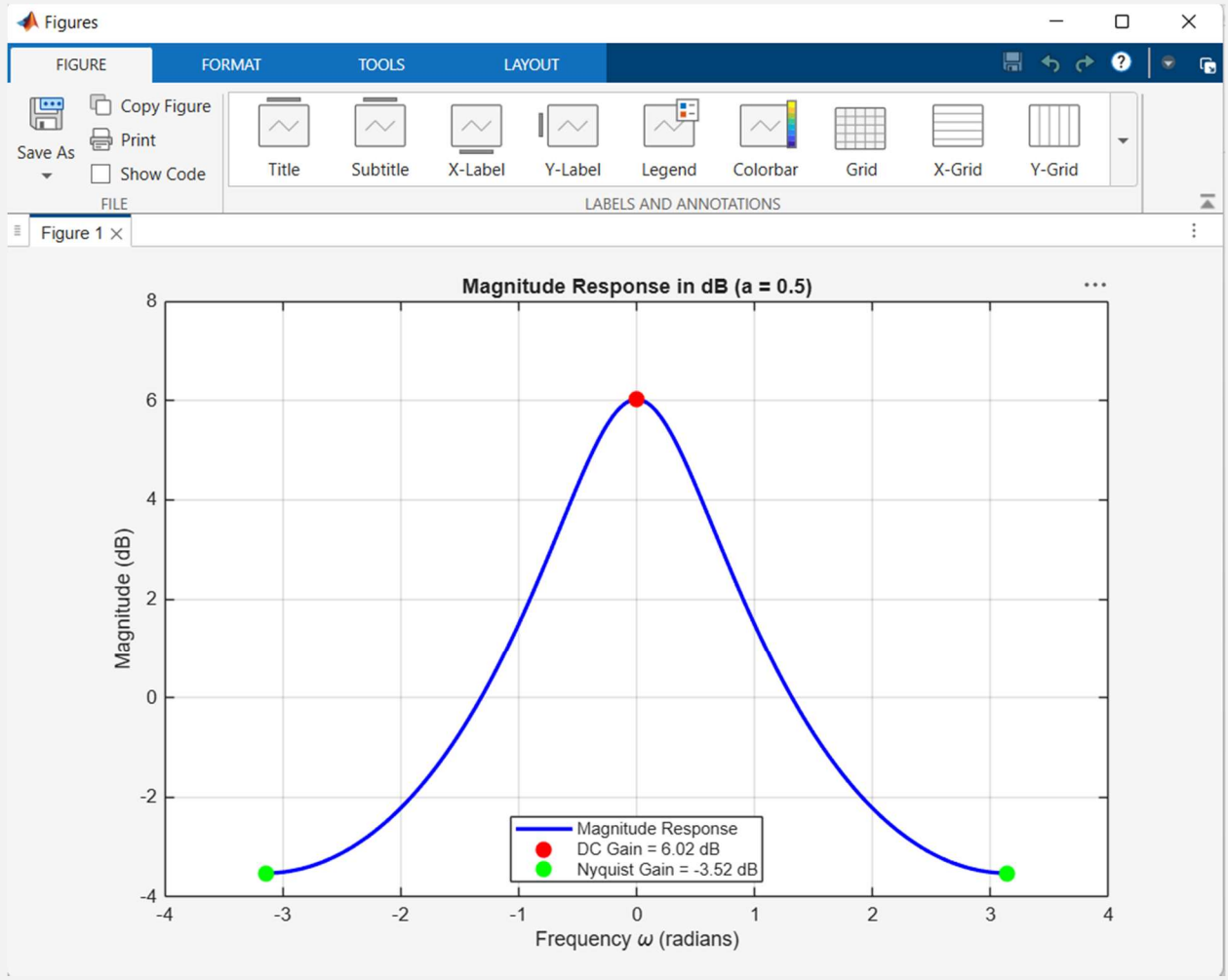
% Display Numerical Results in Command Window
fprintf('--- Verification Results ---\n');
fprintf('DC Gain (omega = 0)    : Linear = %.4f, dB = %.2f dB\n', mag_dc_linear,
mag_dc_dB);
fprintf('Nyquist Gain (omega = pi): Linear = %.4f, dB = %.2f dB\n\n', mag_nyq_linear,
mag_nyq_dB);

% 2. Plot Frequency Response over [-pi, pi]
omega = linspace(-pi, pi, 1000);
mag_response = 1 ./ sqrt(1 - 2*a*cos(omega) + a^2);
mag_response_dB = 20 * log10(mag_response);
```

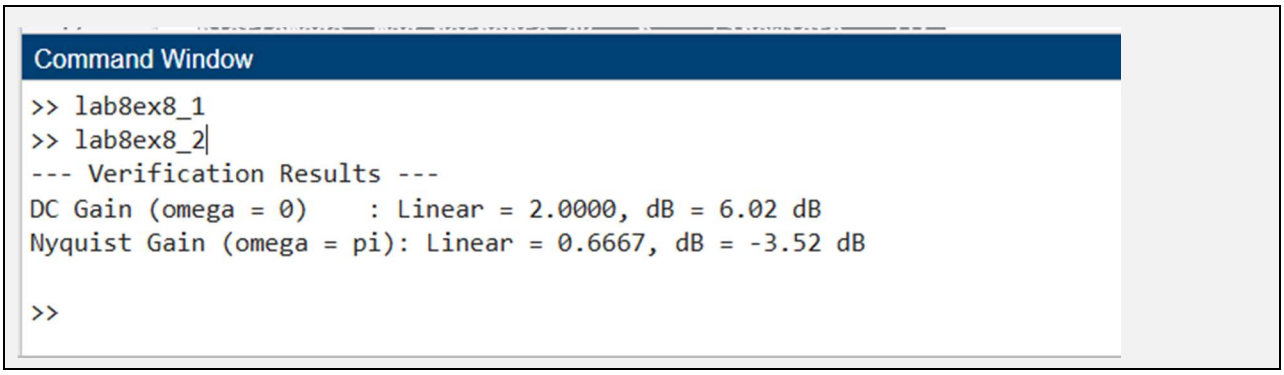
Digital Signal Processing Lab

```
figure;
plot(omega, mag_response_dB, 'b', 'LineWidth', 2);
hold on;
plot(0, mag_dc_dB, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');
plot(pi, mag_nyq_dB, 'go', 'MarkerSize', 8, 'MarkerFaceColor', 'g');
plot(-pi, mag_nyq_dB, 'go', 'MarkerSize', 8, 'MarkerFaceColor', 'g');

title('Magnitude Response in dB (a = 0.5)');
xlabel('Frequency \omega (radians)');
ylabel('Magnitude (dB)');
grid on;
legend('Magnitude Response', sprintf('DC Gain = %.2f dB', mag_dc_dB), ...
    sprintf('Nyquist Gain = %.2f dB', mag_nyq_dB), 'Location', 'south');
```



Digital Signal Processing Lab



6. Laboratory Tasks & Worksheets

- Task 8.1: Given y[n] = 0.6*y[n-1] + x[n], compute and plot frequency response magnitude (dB) and phase (degrees) using MATLAB freqz().
- Task 8.2: Plot magnitude and phase responses for second-order low-pass filter with numerator [0.2 0.4 0.2] and denominator [1 -0.6 0.2].
- Task 8.3: Compute and plot group delay tau_g(omega) using MATLAB grpdelay().
- Task 8.4: Simulate output of unstable system with pole at z=1.2 excited by bounded step input to demonstrate unbounded exponential growth.

7. In-Depth Observations, Analytical Discussion & Verification

- Magnitude response dB plots clearly delineate passband gain and stopband attenuation.
- Group delay indicates constant sample latency for linear phase FIR filters vs frequency-dependent delay in IIR filters.
- Unstable systems (|p|>1) yield unbounded output when tested with bounded inputs.

8. Post-Lab Exercises & Mathematical Assignments

- Assignment 8.1: Design a high-pass filter by configuring zeros at z=1 and analyze its frequency response in freqz().
- Assignment 8.2: Find the 3-dB cutoff frequency of a first-order low-pass filter analytically and verify using MATLAB.
- Assignment 8.3: Plot the pole-zero map and frequency response of an All-Pass filter H(z) = (a* + z^-1) / (1 + a*z^-1) and confirm flat magnitude response.

Digital Signal Processing Lab

LAB-09: EFFECTS OF SAMPLING AND ALIASING

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Nyquist-Shannon Sampling Theorem: A continuous bandlimited signal with maximum frequency component f_max can be uniquely restored from its samples if sampling frequency Fs > 2 * f_max.
- Nyquist Rate: F_Nyquist = 2 * f_max. Minimum allowable sampling rate.
- Aliasing Phenomenon: If Fs < 2 * f_max (undersampling), high-frequency spectral components fold over into the baseband, masquerading as lower false frequencies.
- Apparent Alias Frequency Formula: f_alias = |f - k * Fs|, where k is an integer chosen such that 0 <= f_alias <= Fs / 2.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 9.1: Alias Frequency Calculation
Problem: A continuous sinusoid x(t) = cos(2*pi*140*t) is sampled at Fs = 100 Hz. Find the apparent alias frequency f_alias.
Solution:
Signal frequency f = 140 Hz. Nyquist rate 2f = 280 Hz. Since Fs = 100 < 280, undersampling occurs.
Using alias formula f_alias = |140 - k*100|:
For k=1: |140 - 100| = 40 Hz. Since 40 <= Fs/2 = 50 Hz, the apparent frequency is 40 Hz.

Solved Example 9.2: Subsampling and Reconstructed Waveform
Problem: Show that x1(t) = cos(2*pi*10*t) and x2(t) = cos(2*pi*60*t) yield identical discrete samples when sampled at Fs = 50 Hz.
Solution:
x1[n] = cos(2*pi*10*n / 50) = cos(0.4*pi*n).
x2[n] = cos(2*pi*60*n / 50) = cos(2.4*pi*n) = cos(2*pi*n + 0.4*pi*n) = cos(0.4*pi*n).
Samples are identical, proving that 60 Hz aliases to 10 Hz at Fs = 50 Hz.

Digital Signal Processing Lab

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 9.1
% Given Parameters
f_orig = 140;            % Original signal frequency (Hz)
Fs = 100;                % Sampling frequency (Hz)
F_nyq = Fs / 2;          % Nyquist limit (50 Hz)

% 1. Calculate Apparent Alias Frequency using modulo/folding math
k = round(f_orig / Fs);
f_alias = abs(f_orig - k * Fs);

% Display Numerical Verification Results
fprintf('--- Verification Results ---\n');
fprintf('Original Frequency  : %d Hz\n', f_orig);
fprintf('Sampling Frequency  : %d Hz\n', Fs);
fprintf('Nyquist Limit (Fs/2): %d Hz\n', F_nyq);
fprintf('Calculated Alias f  : %d Hz\n\n', f_alias);

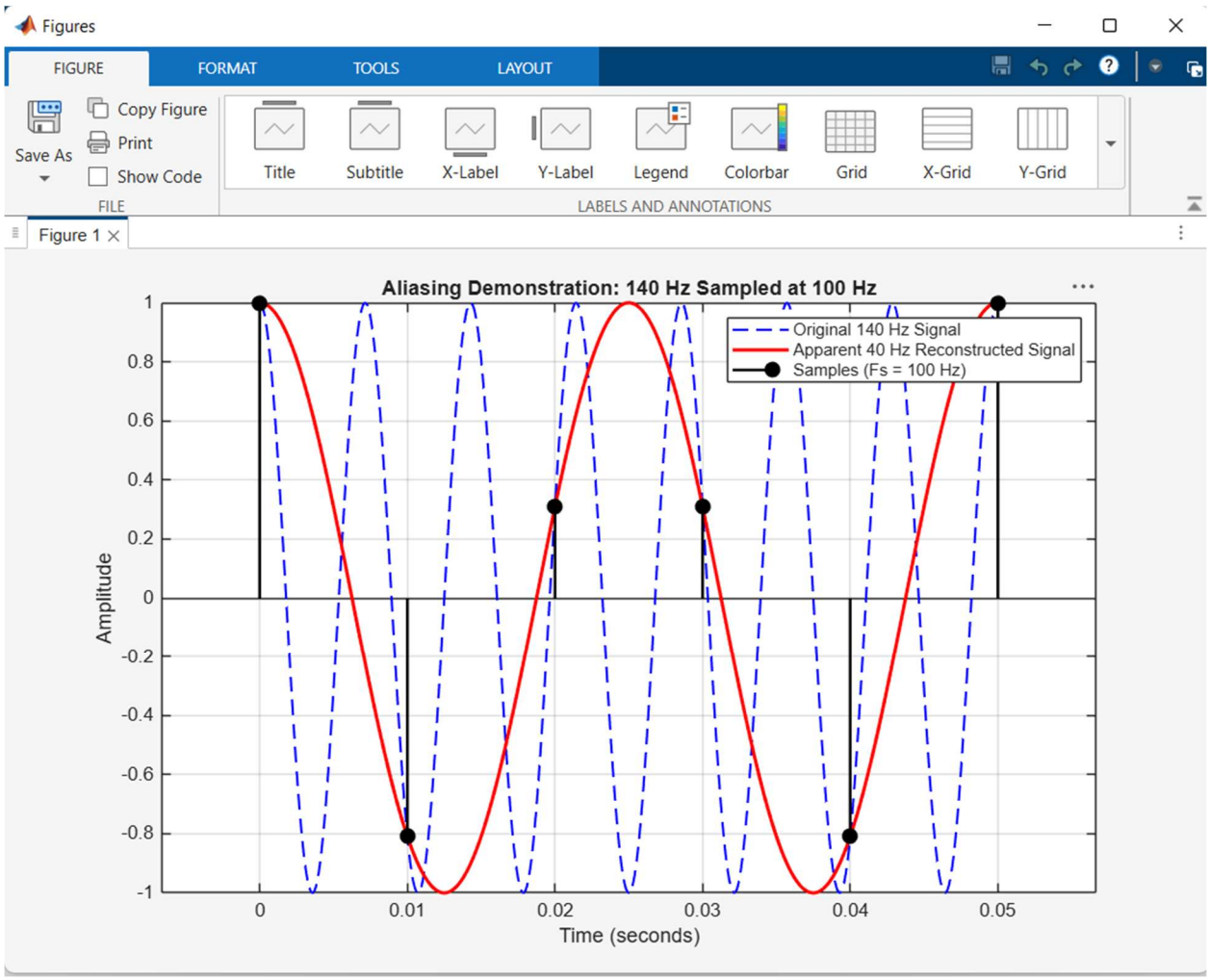
% 2. Visual Demonstration: Time Domain Waveforms
t_cont = 0:0.0001:0.05;  % Continuous time (50 ms)
n_samples = 0:floor(0.05*Fs);
t_samples = n_samples / Fs;

% Signals
x_orig = cos(2 * pi * f_orig * t_cont);   % 140 Hz continuous
x_alias = cos(2 * pi * f_alias * t_cont); % 40 Hz apparent continuous
x_discrete = cos(2 * pi * f_orig * t_samples); % Samples taken at Fs = 100 Hz

figure;
plot(t_cont, x_orig, 'b--', 'LineWidth', 1.2, 'DisplayName', 'Original 140 Hz
Signal');
hold on;
plot(t_cont, x_alias, 'r-', 'LineWidth', 1.8, 'DisplayName', 'Apparent 40 Hz
Reconstructed Signal');
stem(t_samples, x_discrete, 'k', 'filled', 'MarkerSize', 7, ...
    'LineWidth', 1.5, 'DisplayName', 'Samples (Fs = 100 Hz)');

title('Aliasing Demonstration: 140 Hz Sampled at 100 Hz');
xlabel('Time (seconds)');
ylabel('Amplitude');
legend('Location', 'northeast');
grid on;
```

Digital Signal Processing Lab



Command Window

```
--- Verification Results ---
Original Frequency  : 140 Hz
Sampling Frequency  : 100 Hz
Nyquist Limit (Fs/2): 50 Hz
Calculated Alias f  : 40 Hz

>>
```

```
%Example 9.2
% Sampling parameters
Fs = 50;                  % Sampling rate (50 Hz)
Ts = 1 / Fs;              % Sampling interval
t_cont = 0:0.0001:0.2;    % High-resolution continuous time vector (0 to 0.2 s)
n_samples = 0:10;         % Sample indices (n = 0 to 10)
t_samples = n_samples*Ts; % Sample time instances

% Continuous-time signals
x1_cont = cos(2 * pi * 10 * t_cont); % 10 Hz signal
x2_cont = cos(2 * pi * 60 * t_cont); % 60 Hz signal
```

Digital Signal Processing Lab

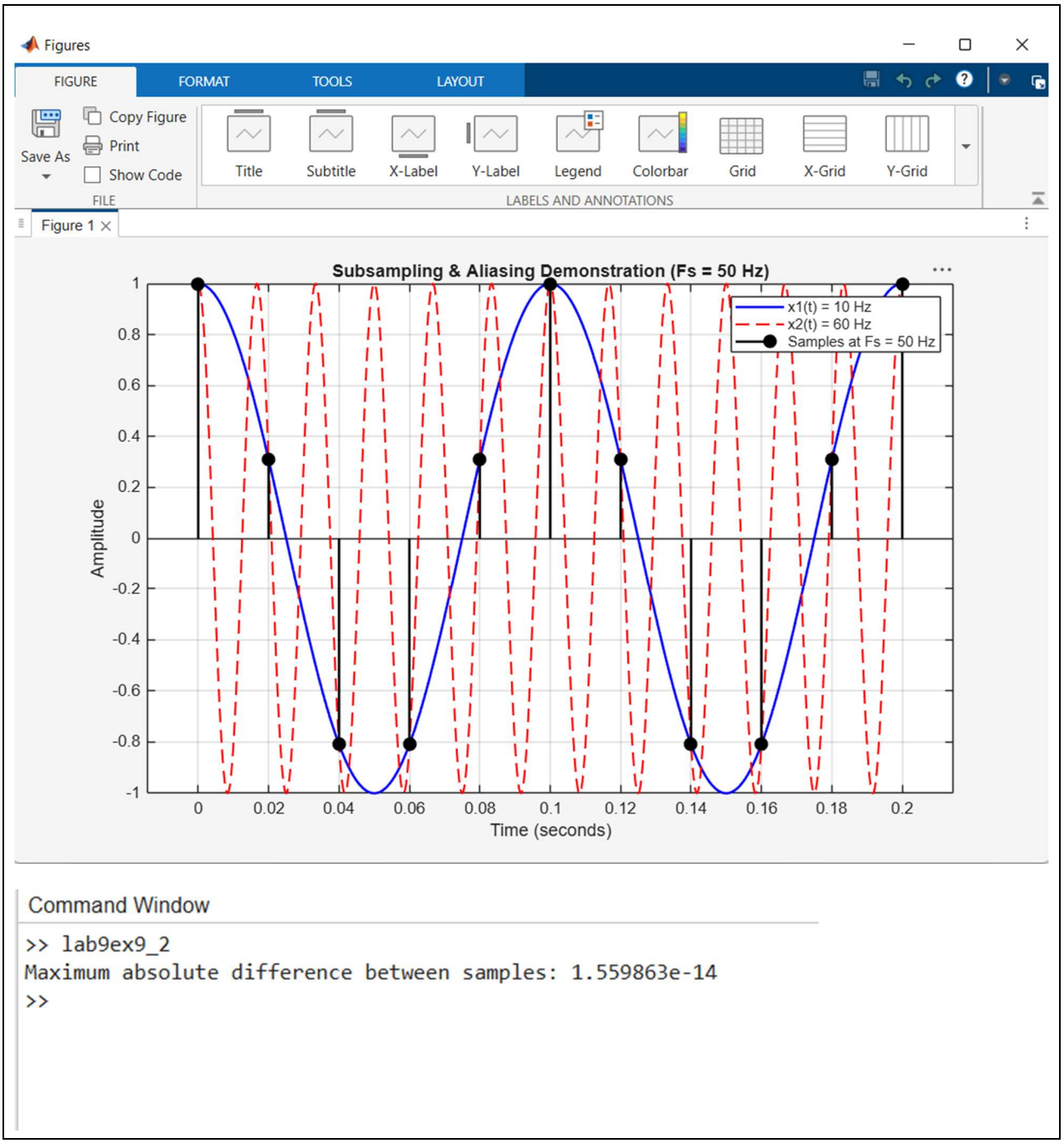
```
% Discrete-time samples
x1_discrete = cos(2 * pi * 10 * t_samples);
x2_discrete = cos(2 * pi * 60 * t_samples);

% Display maximum error between discrete sample vectors
max_diff = max(abs(x1_discrete - x2_discrete));
fprintf('Maximum absolute difference between samples: %.6e\n', max_diff);

% Plotting
figure;
plot(t_cont, x1_cont, 'b-', 'LineWidth', 1.5, 'DisplayName', 'x1(t) = 10 Hz');
hold on;
plot(t_cont, x2_cont, 'r--', 'LineWidth', 1.2, 'DisplayName', 'x2(t) = 60 Hz');
stem(t_samples, x1_discrete, 'k', 'filled', 'MarkerSize', 7, ...
    'LineWidth', 1.5, 'DisplayName', 'Samples at Fs = 50 Hz');

title('Subsampling & Aliasing Demonstration (Fs = 50 Hz)');
xlabel('Time (seconds)');
ylabel('Amplitude');
legend('Location', 'northeast');
grid on;
```

Digital Signal Processing Lab



6. Lab Tasks

- Task 9.1: Generate continuous 50 Hz sine wave sampled at high resolution (Fs = 2000 Hz). Subsample at Fs1 = 200 Hz (proper sampling) and Fs2 = 60 Hz (aliased).
- Task 9.2: Plot continuous reference wave alongside discrete samples and reconstructed waveforms for both rates.

Digital Signal Processing Lab

- Task 9.3: Compute FFT spectra of proper vs aliased sampled sequences.
- Task 9.4: Simulate anti-aliasing low-pass filtering prior to subsampling.

7. In-Depth Observations, Analytical Discussion & Verification

- Sampling at Fs = 60 Hz (Fs < 2f) violates Nyquist theorem, producing an apparent aliased frequency of |50-60| = 10 Hz.
- Reconstructed waveforms from undersampled data match low-frequency aliases rather than the original signal.
- Anti-aliasing analog filters are essential before ADC sampling.

8. Lab Assignments

- Assignment 9.1: Simulate stroboscopic wagon-wheel aliasing effect on a rotating wheel image sequence.
- Assignment 9.2: Determine the minimum allowable sampling frequency to avoid aliasing for composite signal x(t) = cos(100*pi*t) + 3*sin(300*pi*t) - 2*cos(700*pi*t).
- Assignment 9.3: Analyze spectral folding of a frequency chirp sweeping from 0 to 1000 Hz sampled at Fs = 500 Hz.

Digital Signal Processing Lab

LAB-10: QUANTIZATION AND QUANTIZATION ERROR

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Uniform Quantization: Discretizing continuous amplitude range [X_min, X_max] into L = 2^b discrete levels, where b is ADC bit resolution.
- Quantization Step Size Delta: Delta = (X_max - X_min) / 2^b. Represents voltage resolution per bit level.
- Quantization Noise e[n]: Difference e[n] = x_q[n] - x[n]. Error is bounded within [-Delta/2, Delta/2].
- Signal-to-Quantization-Noise Ratio (SQNR): For full-scale sinusoidal input, SQNR = 10 * log10(P_signal / P_noise) approx 6.02*b + 1.76 dB.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 10.1: Step Size and Noise Power Calculation
Problem: An 8-bit ADC operates over voltage range [-5V, +5V]. Compute step size Delta and quantization noise power P_e.
Solution:
1. Number of levels L = 2^8 = 256.
2. Step size Delta = (5 - (-5)) / 256 = 10 / 256 = 0.03906 V = 39.06 mV.
3. Quantization noise variance P_e = Delta^2 / 12 = (0.03906)^2 / 12 = 1.271 * 10^-4 V^2.

Solved Example 10.2: SQNR Calculation
Problem: Compute theoretical SQNR for b=3 bits and b=12 bits.
Solution:
For b=3: SQNR = 6.02(3) + 1.76 = 18.06 + 1.76 = 19.82 dB.
For b=12: SQNR = 6.02(12) + 1.76 = 72.24 + 1.76 = 74.00 dB.

Digital Signal Processing Lab

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 10.1
% ADC Parameters
N = 8;                  % Resolution in bits
V_min = -5;             % Minimum input voltage
V_max = 5;              % Maximum input voltage
V_range = V_max - V_min;% Peak-to-peak voltage range

% 1. Step Size Calculation
L = 2^N;                % Number of levels
delta = V_range / L;    % Step size (Delta)

% 2. Theoretical Quantization Noise Power
P_e_theory = (delta^2) / 12;

% Display Theoretical Calculations
fprintf('--- Theoretical Results ---\n');
fprintf('Number of Levels (L)      : %d\n', L);
fprintf('Step Size (Delta)         : %.5f V (%.2f mV)\n', delta, delta * 1000);
fprintf('Noise Power (P_e)         : %.4e V^2\n\n', P_e_theory);

% 3. Numerical Simulation Verification
% Generate a test sinusoidal signal
t = 0:0.0001:1;
x = 4.5 * sin(2 * pi * 5 * t); % Signal within [-5, 5] V

% Uniform Quantization Process
x_quant = round((x - V_min) / delta) * delta + V_min;
x_quant = min(max(x_quant, V_min), V_max - delta); % Saturation check

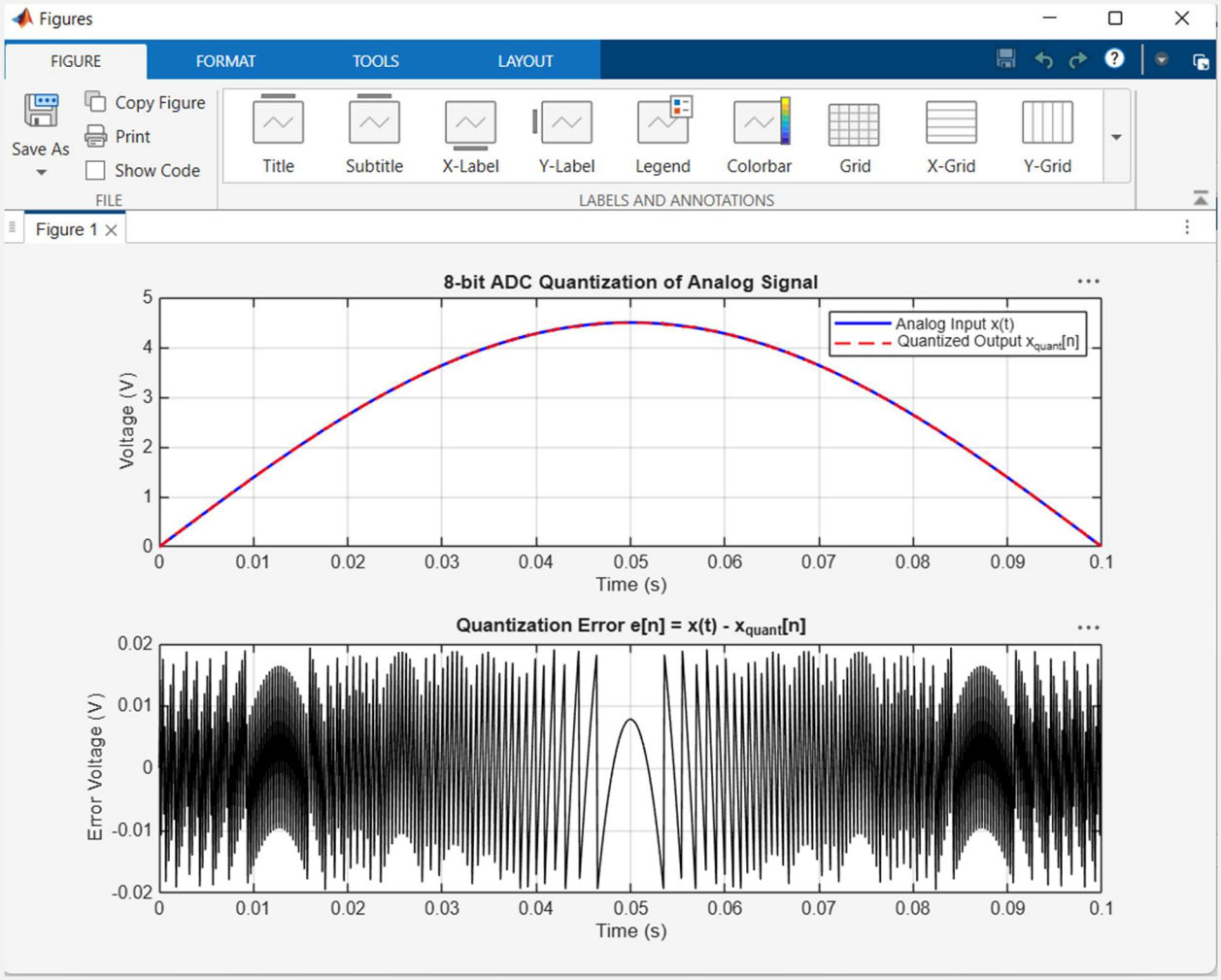
% Calculate Quantization Error & Simulated Noise Power
q_error = x - x_quant;
P_e_simulated = var(q_error);

fprintf('--- Simulation Results ---\n');
fprintf('Simulated Noise Variance   : %.4e V^2\n', P_e_simulated);

% Plotting Original vs. Quantized Signal and Error
figure;
subplot(2, 1, 1);
plot(t(1:1000), x(1:1000), 'b', 'LineWidth', 1.5); hold on;
plot(t(1:1000), x_quant(1:1000), 'r--', 'LineWidth', 1.2);
title('8-bit ADC Quantization of Analog Signal');
xlabel('Time (s)');
ylabel('Voltage (V)');
legend('Analog Input x(t)', 'Quantized Output x_{quant}[n]');
grid on;

subplot(2, 1, 2);
plot(t(1:1000), q_error(1:1000), 'k');
title('Quantization Error e[n] = x(t) - x_{quant}[n]');
xlabel('Time (s)');
ylabel('Error Voltage (V)');
grid on;
```

Digital Signal Processing Lab




```
Command Window
Number of Levels (L)       : 256
Step Size (Delta)          : 0.03906 V (39.06 mV)
Noise Power (P_e)          : 1.2716e-04 V^2

--- Simulation Results ---
Simulated Noise Variance   : 1.2239e-04 V^2
>>
```

```
%Example 10.2
% Bit depths to analyze
bits = [3, 12];

% Calculate theoretical SQNR using formula: SQNR_dB = 6.02*b + 1.76
sqnr_theoretical = 6.02 * bits + 1.76;

% Display Theoretical Results
fprintf('--- Theoretical SQNR Results ---\n');
for i = 1:length(bits)
    fprintf('For b = %2d bits: Theoretical SQNR = %.2f dB\n', bits(i),
```

Digital Signal Processing Lab

```
sqnr_theoretical(i));
end
fprintf('\n');

% Numerical Simulation Verification (Base MATLAB)
fs = 10000;              % Sampling frequency
t = 0:1/fs:1;            % Time vector (1 second)
x = sin(2 * pi * 10 * t); % Full-scale sine wave [-1, 1]

fprintf('--- Numerical Simulation Results ---\n');
for i = 1:length(bits)
    b = bits(i);
    L = 2^b;                     % Quantization levels
    delta = 2 / L;               % Step size for range [-1, 1]

    % Quantize signal
    x_quant = round(x / delta) * delta;

    % Compute Signal and Noise Power
    signal_power = mean(x.^2);
    noise_power = mean((x - x_quant).^2);

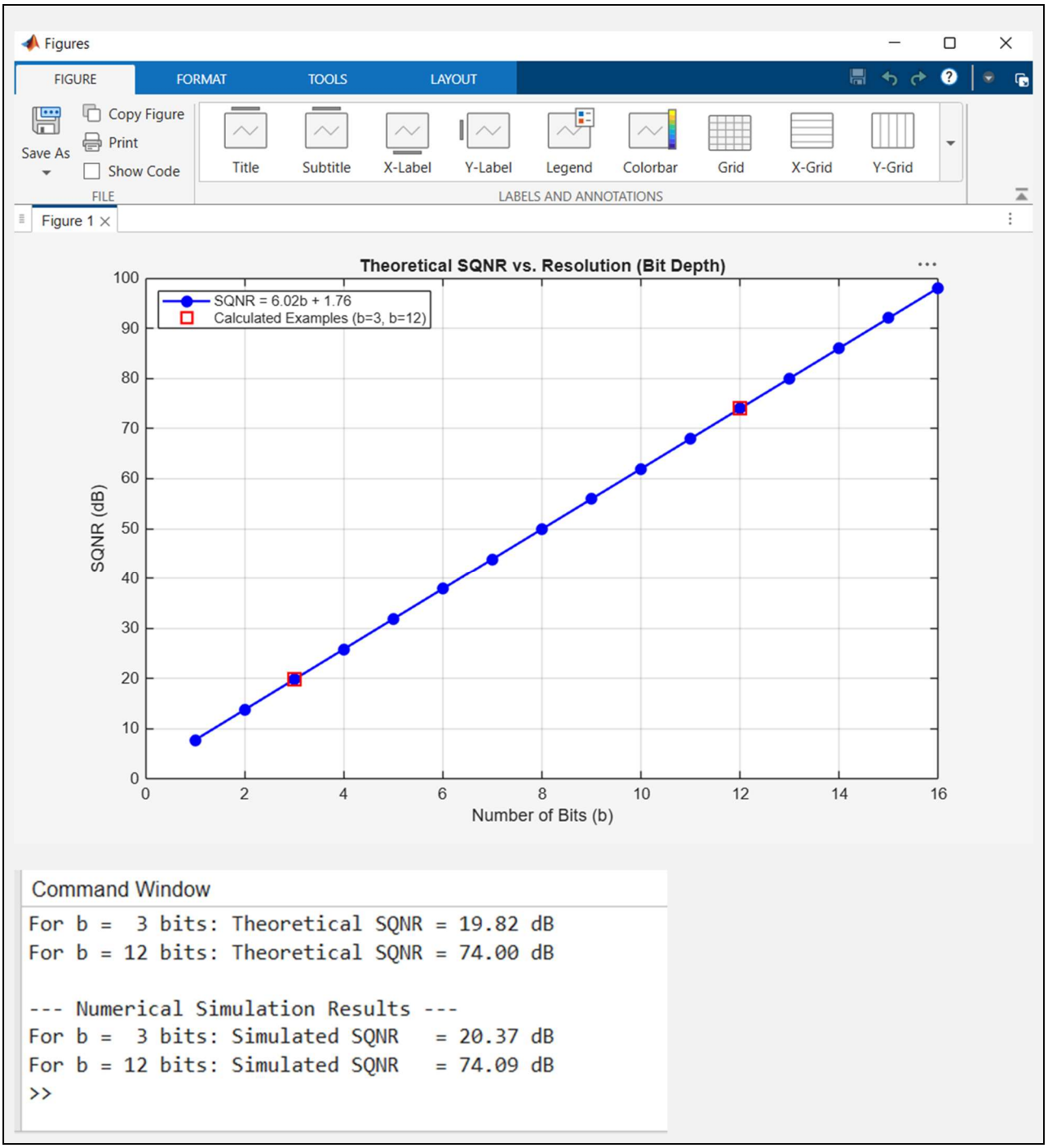
    % Simulated SQNR in dB
    sqnr_simulated = 10 * log10(signal_power / noise_power);

    fprintf('For b = %2d bits: Simulated SQNR   = %.2f dB\n', b, sqnr_simulated);
end

% Plot SQNR vs. Bit Depth (1 to 16 bits)
b_range = 1:16;
sqnr_curve = 6.02 * b_range + 1.76;

figure;
plot(b_range, sqnr_curve, 'b-o', 'LineWidth', 1.5, 'MarkerFaceColor', 'b');
hold on;
plot(bits, sqnr_theoretical, 'rs', 'MarkerSize', 10, 'LineWidth', 2);
title('Theoretical SQNR vs. Resolution (Bit Depth)');
xlabel('Number of Bits (b)');
ylabel('SQNR (dB)');
grid on;
legend('SQNR = 6.02b + 1.76', 'Calculated Examples (b=3, b=12)', 'Location',
'northwest');
```

Digital Signal Processing Lab



6. Lab Tasks

- Task 10.1: Write MATLAB function to quantize continuous sinusoid using b = 3 bits (8 levels) and b = 8 bits (256 levels). Plot original vs quantized waves.

- Task 10.2: Compute and plot quantization error sequence e[n] = x_q[n] - x[n] and its statistical histogram.
- Task 10.3: Calculate empirical SQNR across bit resolutions b = 2 to 12 bits and compare against theoretical 6.02*b + 1.76 dB curve.
- Task 10.4: Demonstrate ADC overload clipping distortion when signal amplitude exceeds range [X_min, X_max].

7. In-Depth Observations, Analytical Discussion & Verification

- Lower bit resolutions (b=3) exhibit severe staircase distortion and coarse quantization steps.
- Quantization error e[n] displays uniform random distribution bounded within [-Delta/2, Delta/2].
- SQNR increases by exactly ~6 dB for each additional bit of resolution.

8. Lab Assignments

- Assignment 10.1: Implement non-uniform quantization (mu-law and A-law companding used in telephony) and compare SQNR against uniform quantization for speech signals.
- Assignment 10.2: Calculate theoretical noise power for a triangular wave input quantized with step size Delta.
- Assignment 10.3: Simulate dithering (adding small random white noise prior to quantization) and observe its effect on harmonic distortion reduction.

Digital Signal Processing Lab

LAB-11: SIGNAL RECONSTRUCTION USING MATLAB

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Ideal Whittaker-Shannon Sinc Reconstruction: Perfect analog signal reconstruction using sinc interpolation: x_r(t) = Sum(x[n] * sinc((t - n*Ts)/Ts)) from n=-inf to inf.
- Zero-Order Hold (ZOH): Practical DAC method that holds sample values constant over sample interval Ts, creating a staircase waveform.
- First-Order Hold (FOH): Linear interpolation between adjacent samples, producing a continuous piecewise linear waveform.
- Reconstruction Filter Droop: ZOH introduces high-frequency sinc attenuation |H_zoh(f)| = Ts * |sin(pi*f*Ts) / (pi*f*Ts)|, requiring analog anti-imaging smoothing filters.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 11.1: Sinc Interpolation Sum Calculation
Problem: Given samples x[0]=1, x[1]=2 with Ts = 1 s. Compute reconstructed value x_r(0.5) using ideal sinc interpolation.
Solution:
x_r(t) = x[0]*sinc(t/Ts) + x[1]*sinc((t-Ts)/Ts).
At t=0.5:
sinc(0.5) = sin(0.5*pi) / (0.5*pi) = 1 / (pi/2) = 2/pi approx 0.6366.
sinc(-0.5) = sinc(0.5) approx 0.6366.
x_r(0.5) = 1(0.6366) + 2(0.6366) = 3(0.6366) = 1.9098.

Solved Example 11.2: ZOH Frequency Magnitude Droop
Problem: Compute ZOH amplitude attenuation at Nyquist frequency f = Fs / 2.
Solution:

Digital Signal Processing Lab

At f = 1/(2*Ts): |H_zoh(Fs/2)| = |sin(pi/2) / (pi/2)| = 1 / (pi/2) = 2/pi approx 0.6366 => 20*log10(0.6366) = -3.92 dB.

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 11.1
% Parameters
Fs = 1;                     % Sampling interval (s)
Ts = 1;
n = [0, 1];                 % Sample indices
x = [1, 2];                 % Sample values x[0]=1, x[1]=2
t_eval = 0.5;               % Evaluation time point

% 1. Compute Reconstructed Value at t = 0.5 s (Base MATLAB Sinc)
x_r = 0;
for i = 1:length(n)
    arg = (t_eval - n(i)*Ts) / Ts;

    % Base MATLAB replacement for toolbox sinc function
    if arg == 0
        sinc_val = 1.0;
    else
        sinc_val = sin(pi * arg) / (pi * arg);
    end

    x_r = x_r + x(i) * sinc_val;
end

fprintf('--- Verification Results ---\n');
fprintf('Reconstructed value x_r(%.2f) = %.5f\n\n', t_eval, x_r);

% 2. Continuous Reconstruction over Time Range
t_cont = linspace(-1, 2, 1000);
x_recon = zeros(size(t_cont));

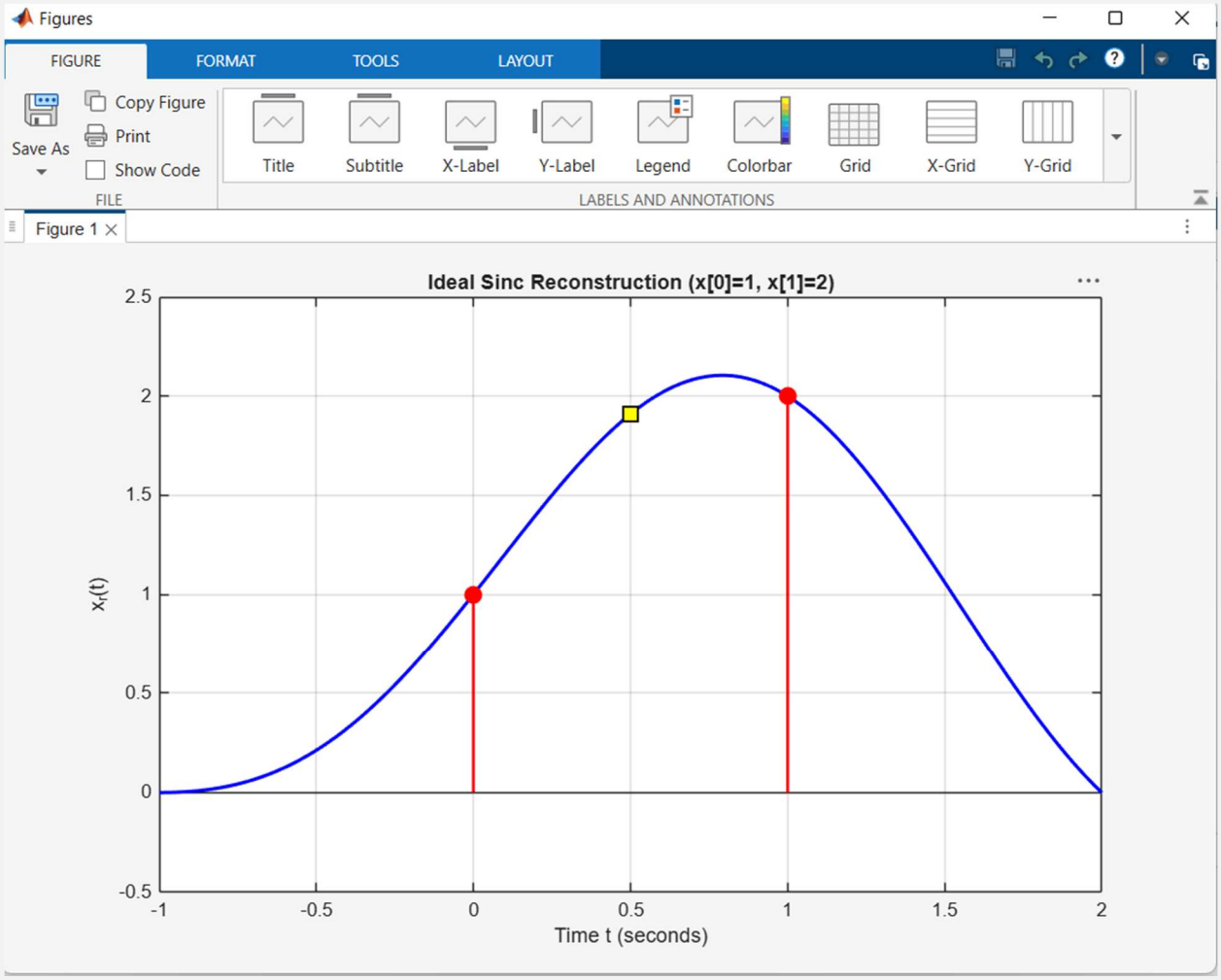
for j = 1:length(t_cont)
    val = 0;
    for i = 1:length(n)
        arg = (t_cont(j) - n(i)*Ts) / Ts;
        if arg == 0
            sinc_val = 1.0;
        else
            sinc_val = sin(pi * arg) / (pi * arg);
        end
        val = val + x(i) * sinc_val;
    end
    x_recon(j) = val;
end

% Plotting
figure;
plot(t_cont, x_recon, 'b-', 'LineWidth', 1.8);
hold on;
stem(n*Ts, x, 'ro', 'filled', 'MarkerSize', 8, 'LineWidth', 1.5);
```

Digital Signal Processing Lab

```
plot(t_eval, x_r, 'ks', 'MarkerSize', 10, 'MarkerFaceColor', 'y', 'LineWidth', 1.5);

title('Ideal Sinc Reconstruction (x[0]=1, x[1]=2)');
xlabel('Time t (seconds)');
ylabel('x_r(t)');
grid on;
```



Command Window

```
>> lab11ex11_1
--- Verification Results ---
Reconstructed value x_r(0.50) = 1.90986

>>
```

```
%Example 11.2
% Sampling parameters
Fs = 1000;              % Sampling frequency (Hz)
Ts = 1 / Fs;            % Sampling period (s)
```

Digital Signal Processing Lab

```
f_nyq = Fs / 2;          % Nyquist frequency (500 Hz)

% 1. Compute ZOH Attenuation at Nyquist Frequency
mag_nyq_linear = abs(sin(pi * f_nyq * Ts) / (pi * f_nyq * Ts));
mag_nyq_dB = 20 * log10(mag_nyq_linear);

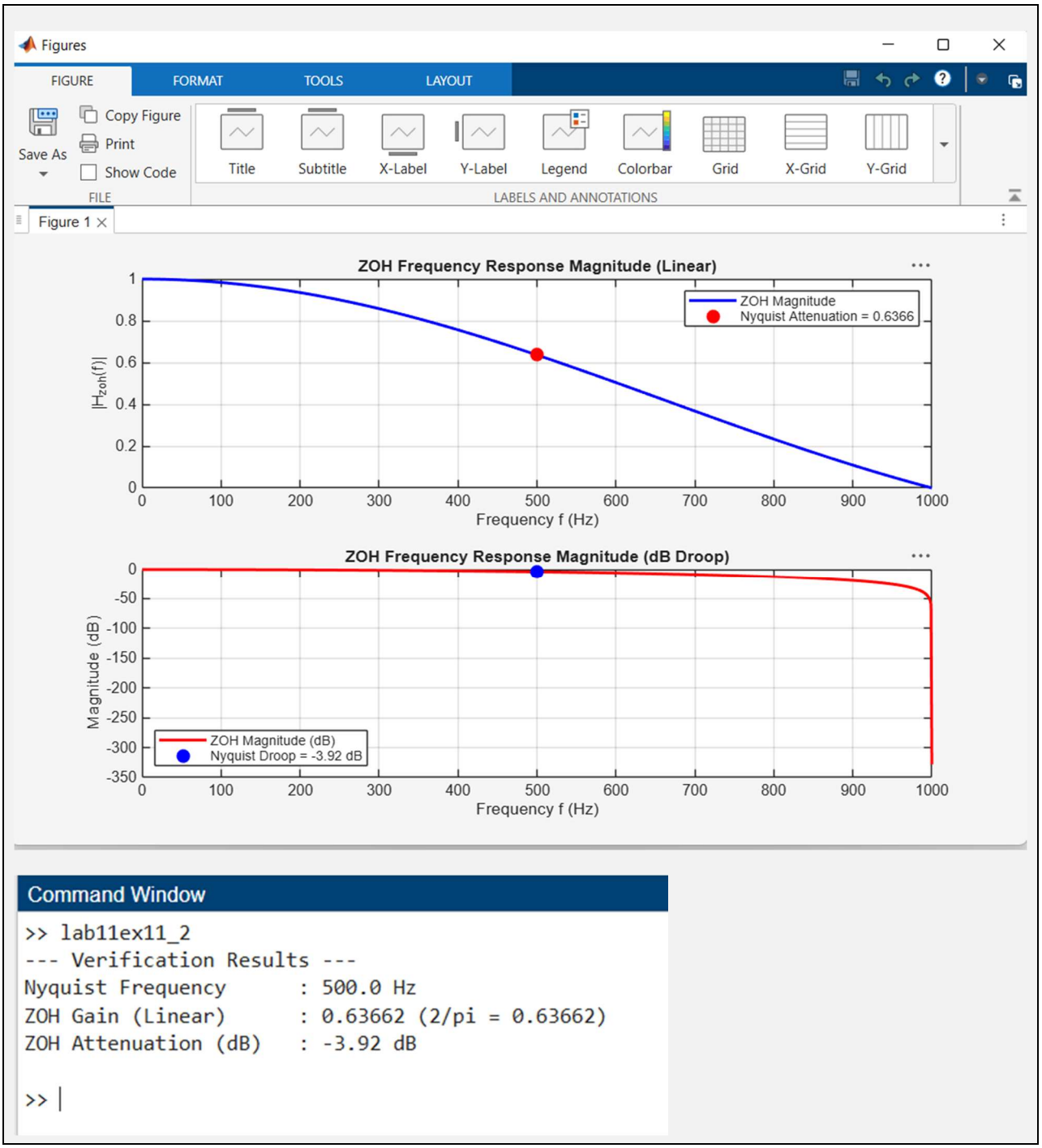
% Display Numerical Verification Results
fprintf('--- Verification Results ---\n');
fprintf('Nyquist Frequency      : %.1f Hz\n', f_nyq);
fprintf('ZOH Gain (Linear)      : %.5f (2/pi = %.5f)\n', mag_nyq_linear, 2/pi);
fprintf('ZOH Attenuation (dB)   : %.2f dB\n\n', mag_nyq_dB);

% 2. Plot ZOH Frequency Response Magnitude over [0, Fs]
f = linspace(0.001, Fs, 1000); % Avoid division by zero at f = 0
mag_zoh = abs(sin(pi * f * Ts) ./ (pi * f * Ts));
mag_zoh_dB = 20 * log10(mag_zoh);

figure;
subplot(2, 1, 1);
plot(f, mag_zoh, 'b', 'LineWidth', 1.8);
hold on;
plot(f_nyq, mag_nyq_linear, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');
title('ZOH Frequency Response Magnitude (Linear)');
xlabel('Frequency f (Hz)');
ylabel('|H_{zoh}(f)|');
grid on;
legend('ZOH Magnitude', sprintf('Nyquist Attenuation = %.4f', mag_nyq_linear));

subplot(2, 1, 2);
plot(f, mag_zoh_dB, 'r', 'LineWidth', 1.8);
hold on;
plot(f_nyq, mag_nyq_dB, 'bo', 'MarkerSize', 8, 'MarkerFaceColor', 'b');
title('ZOH Frequency Response Magnitude (dB Droop)');
xlabel('Frequency f (Hz)');
ylabel('Magnitude (dB)');
grid on;
legend('ZOH Magnitude (dB)', sprintf('Nyquist Droop = %.2f dB', mag_nyq_dB),
'Location', 'southwest');
```

Digital Signal Processing Lab



6. Lab Tasks

- Task 11.1: Sample a 10 Hz sine wave at Fs = 50 Hz. Reconstruct continuous signal using ideal sinc interpolation over dense time grid.

Digital Signal Processing Lab

- Task 11.2: Implement ZOH reconstruction using MATLAB stairs() and FOH reconstruction using interp1(..., 'linear').
- Task 11.3: Compute and plot reconstruction error waveforms for Sinc, ZOH, and FOH methods.
- Task 11.4: Design an analog Butterworth smoothing filter to eliminate high-frequency ZOH staircase imaging harmonics.

7. In-Depth Observations, Analytical Discussion & Verification

- Ideal sinc interpolation perfectly reconstructs bandlimited signals without amplitude or phase distortion.
- Zero-Order Hold (ZOH) introduces staircase distortion and -3.92 dB attenuation at Nyquist.
- First-Order Hold (FOH) provides smoother linear interpolation but still leaves high-frequency residual images.

8. Lab Assignments

- Assignment 11.1: Design a 4th-order analog low-pass reconstruction filter following a ZOH DAC and evaluate output total harmonic distortion.
- Assignment 11.2: Implement digital upsampling by factor L=4 using zero-insertion followed by low-pass interpolation filtering.
- Assignment 11.3: Analyze phase delay introduced when truncating infinite sinc interpolation sum to N=5 neighboring samples.

Digital Signal Processing Lab

LAB-12: INTRODUCTION TO DIGITAL FILTERING

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Digital Filter Classification: Finite Impulse Response (FIR) vs Infinite Impulse Response (IIR).
- Filter Selectivity Types: Low-Pass (LPF), High-Pass (HPF), Band-Pass (BPF), and Band-Stop/Notch (BSF).
- Moving Average Filter: Simple FIR low-pass filter defined by difference equation y[n] = (1/M) * Sum(x[n-k]) from k=0 to M-1. Attenuates high-frequency fluctuations.
- Difference Equation Filtering: In MATLAB, general filtering is executed using y = filter(b, a, x) where b numerator and a denominator coefficients define transfer function H(z) = B(z)/A(z).

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 12.1: Moving Average Difference Equation
Problem: Compute output sequence y[n] for 3-point moving average filter (M=3) given input x[n] = [2, 5, 8, 3, 1] for n=[0..4].
Solution:
y[n] = (1/3) * (x[n] + x[n-1] + x[n-2]).
y[0] = (1/3)*(2 + 0 + 0) = 0.667.
y[1] = (1/3)*(5 + 2 + 0) = 2.333.
y[2] = (1/3)*(8 + 5 + 2) = 5.000.
y[3] = (1/3)*(3 + 8 + 5) = 5.333.
y[4] = (1/3)*(1 + 3 + 8) = 4.000.
Output sequence y[n] = [0.667, 2.333, 5.000, 5.333, 4.000].

Solved Example 12.2: First-Order IIR Filter
Problem: Given first-order IIR low-pass filter y[n] = 0.8*y[n-1] + 0.2*x[n]. Compute impulse response h[n] for n=0, 1, 2, 3.

Digital Signal Processing Lab

Solution:

Set x[n] = delta[n]:

h[0] = 0.8(0) + 0.2(1) = 0.2.

h[1] = 0.8(0.2) + 0.2(0) = 0.16.

h[2] = 0.8(0.16) + 0.2(0) = 0.128.

h[3] = 0.8(0.128) + 0.2(0) = 0.1024.

Impulse response h[n] = 0.2*(0.8)^n * u[n] (exponential decay).

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 12.1
% Input signal and time index
n = 0:4;
x = [2, 5, 8, 3, 1];

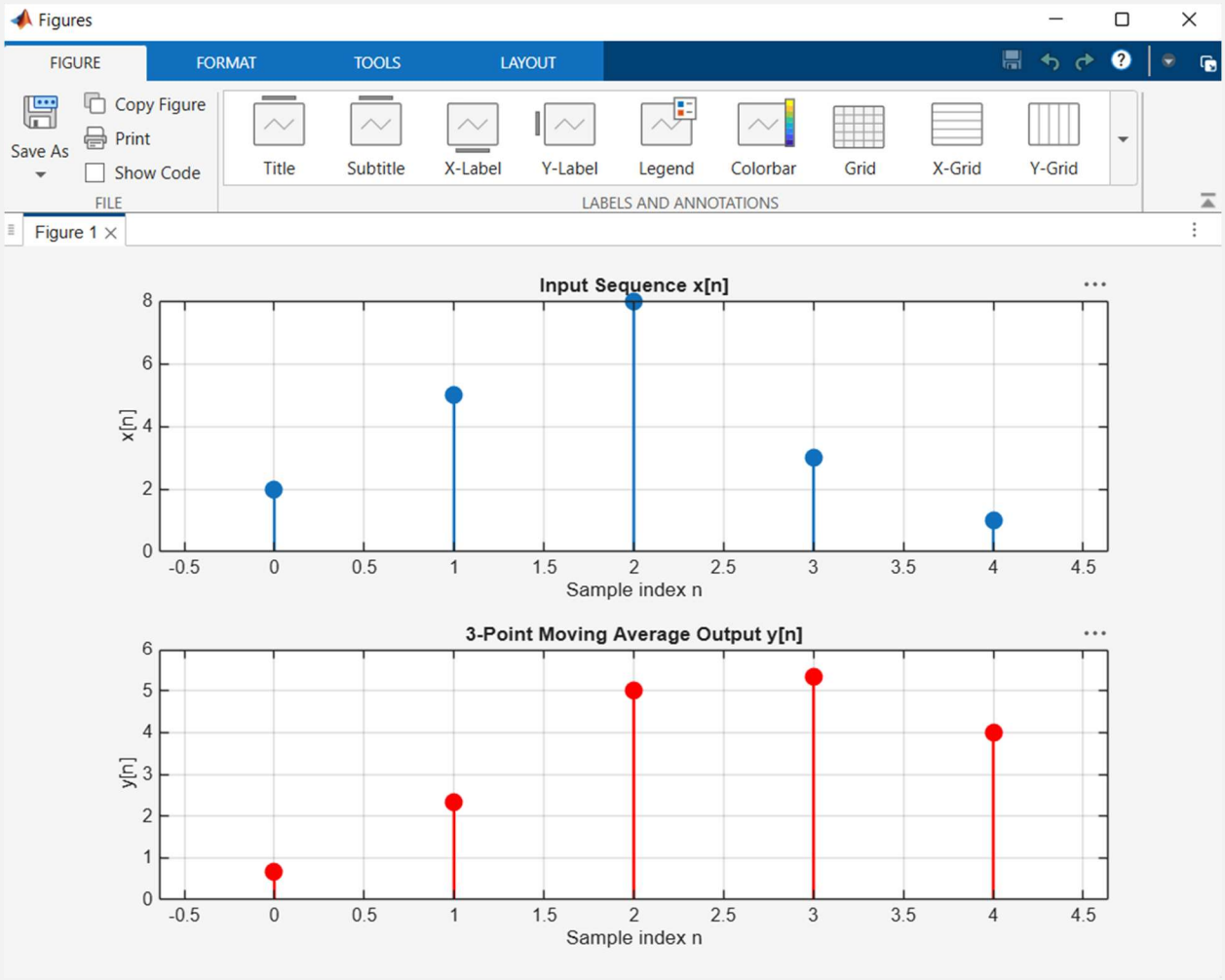
% 1. Compute 3-point moving average output manually
y = zeros(size(n));
for i = 1:length(n)
    val = 0;
    for k = 0:2
        if (i - k) >= 1
            val = val + x(i - k);
        end
    end
    y(i) = val / 3;
end

% Display Results in Command Window
fprintf('--- Verification Results ---\n');
fprintf('Input x[n] : ');
disp(x);
fprintf('Output y[n]: ');
disp(y);

% 2. Plotting Input vs Output Sequences
figure;
subplot(2, 1, 1);
stem(n, x, 'filled', 'MarkerSize', 8, 'LineWidth', 1.5);
title('Input Sequence x[n]');
xlabel('Sample index n');
ylabel('x[n]');
grid on;

subplot(2, 1, 2);
stem(n, y, 'r', 'filled', 'MarkerSize', 8, 'LineWidth', 1.5);
title('3-Point Moving Average Output y[n]');
xlabel('Sample index n');
ylabel('y[n]');
grid on;
```

Digital Signal Processing Lab




```
Command Window
>> lab12ex12_1
--- Verification Results ---
Input x[n] :      2     5     8     3     1

Output y[n]:     0.6667    2.3333    5.0000    5.3333    4.0000

>>
```

```
%Example 12.2
% Time indices from n = 0 to 10
n = 0:10;

% Define filter coefficients: y[n] = 0.8*y[n-1] + 0.2*x[n]
% Rearranged to standard form: a[1]*y[n] = b[1]*x[n] + b[2]*x[n-1] ...
b = [0.2];
a = [1, -0.8];

% Create unit impulse input delta[n]
```

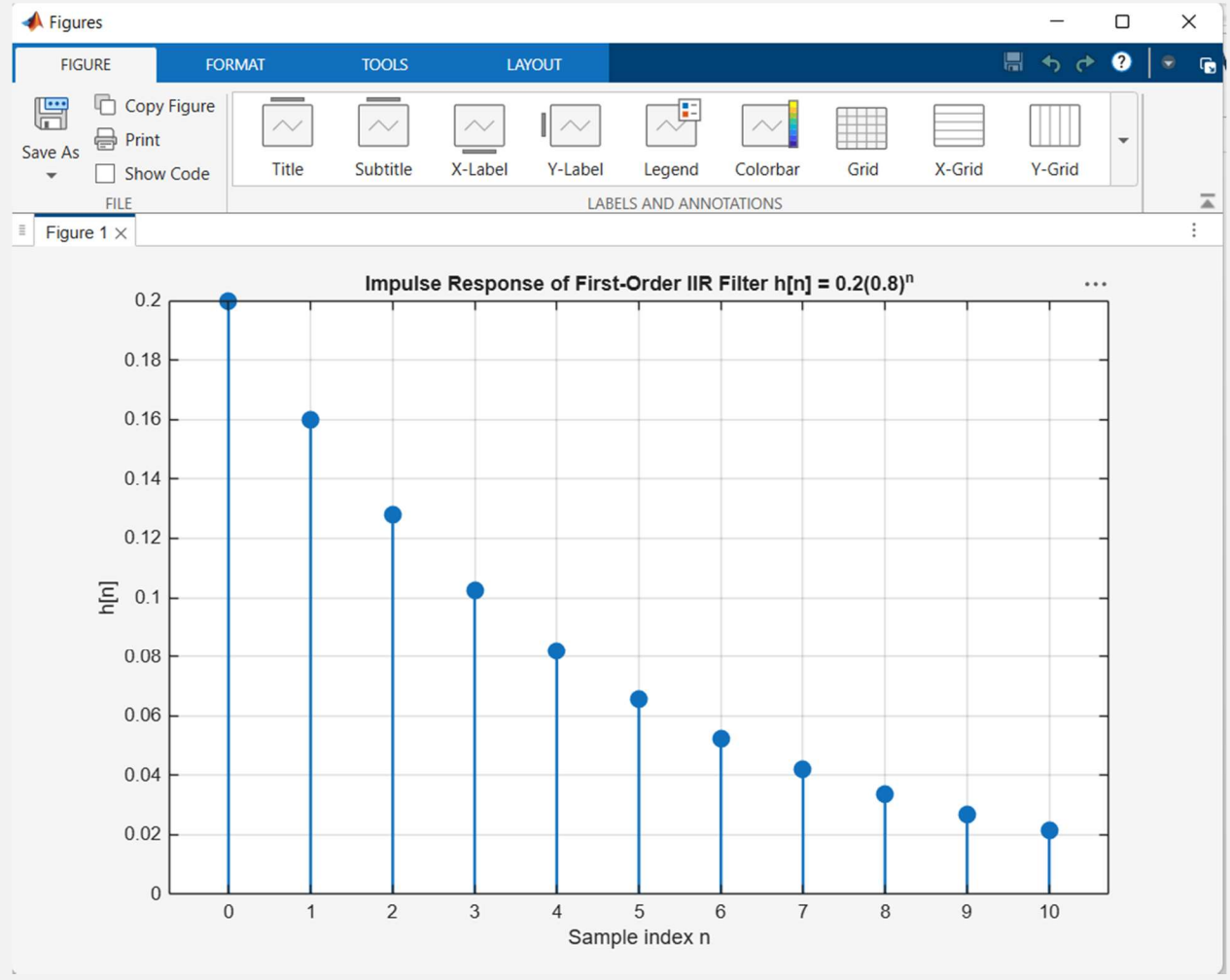
Digital Signal Processing Lab

```
delta_input = [1, zeros(1, length(n) - 1)];

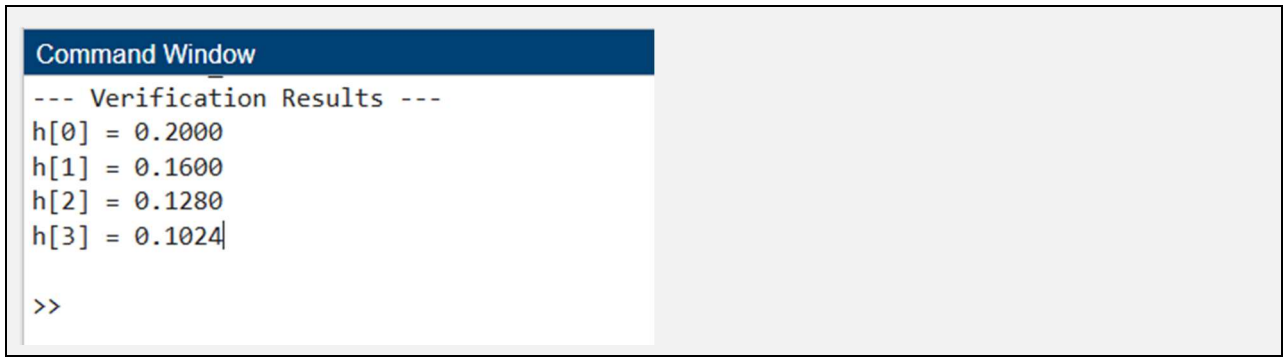
% 1. Compute impulse response using base MATLAB filter() function
h = filter(b, a, delta_input);

% Display Calculated Values for n = 0 to 3 in Command Window
fprintf('--- Verification Results ---\n');
for i = 1:4
    fprintf('h[%d] = %.4f\n', n(i), h(i));
end
fprintf('\n');

% 2. Plot Impulse Response Sequence
figure;
stem(n, h, 'filled', 'MarkerSize', 8, 'LineWidth', 1.5);
title('Impulse Response of First-Order IIR Filter h[n] = 0.2(0.8)^n');
xlabel('Sample index n');
ylabel('h[n]');
grid on;
```



Digital Signal Processing Lab



6. Lab Tasks

- Task 12.1: Generate composite signal x(t) = sin(2*pi*5*t) + 0.5*sin(2*pi*100*t) sampled at Fs = 1000 Hz.
- Task 12.2: Design 10-point moving average low-pass filter and apply using filter().
- Task 12.3: Implement high-pass filter by subtracting low-pass output from delayed input signal.
- Task 12.4: Plot original, low-pass filtered, and high-pass filtered signals along with their FFT spectra.

7. In-Depth Observations, Analytical Discussion & Verification

- Moving average filter effectively suppresses 100 Hz noise while preserving smooth 5 Hz tone.
- Subtracting low-pass output yields high-pass filtering behavior.
- Increasing filter window length M narrows passband width and increases attenuation.

8. Lab Assignments

- Assignment 12.1: Implement a first-order recursive exponential smoothing filter y[n] = (1-alpha)*y[n-1] + alpha*x[n] and evaluate cutoff frequency vs alpha.
- Assignment 12.2: Separate a three-tone audio signal (300 Hz, 1500 Hz, 3500 Hz) using cascaded digital band-pass filters.
- Assignment 12.3: Compute and plot the magnitude response of a 5-point moving average filter using freqz() and locate its transmission nulls (zeros).

Digital Signal Processing Lab

LAB-13: FIR FILTER DESIGN USING WINDOWING AND FREQUENCY SAMPLING

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version

3. Theoretical Background & Fundamental Definitions

- FIR Filter Characteristics: Linear phase, always BIBO stable (poles strictly at origin z=0), bounded impulse response.
- Linear Phase Condition: Symmetric impulse response h[n] = h[N-1-n] yields constant phase slope and constant group delay tau_g = (N-1)/2 samples.
- Window Method: Truncating ideal infinite impulse response h_d[n] using window w[n]: h[n] = h_d[n] * w[n].
- Window Function Types: Rectangular, Hamming, Hanning, Blackman, Kaiser. Trade-off between transition bandwidth and stopband attenuation.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 13.1: Ideal Low-Pass Impulse Response
Problem: Find ideal low-pass filter impulse response h_d[n] with cutoff frequency omega_c = 0.4*pi.
Solution:
h_d[n] = (1/(2*pi)) * Integral(e^(j*omega*n) domega) from -omega_c to omega_c = sin(omega_c*n) / (pi*n) = sin(0.4*pi*n) / (pi*n).
For n=0: h_d[0] = omega_c / pi = 0.4.
Shift by M = (N-1)/2 for causal implementation.

Solved Example 13.2: Window Attenuation Comparison
Problem: Select window function for FIR low-pass filter requiring stopband attenuation A_s >= 50 dB and transition bandwidth Delta_omega = 0.1*pi.
Solution:

Digital Signal Processing Lab

Compare peak stopband attenuations: Rectangular (21 dB), Hamming (53 dB), Blackman (74 dB). Select Hamming Window (A_s = 53 dB >= 50 dB). Order N approx (6.6*pi) / Delta_omega = (6.6*pi) / (0.1*pi) = 66 => N=67 (odd).

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 13.1
% Filter Specifications
omega_c = 0.4 * pi;      % Cutoff frequency
N = 21;                  % Filter length (must be odd for symmetric linear phase)
M = (N - 1) / 2;         % Center shift index for causality

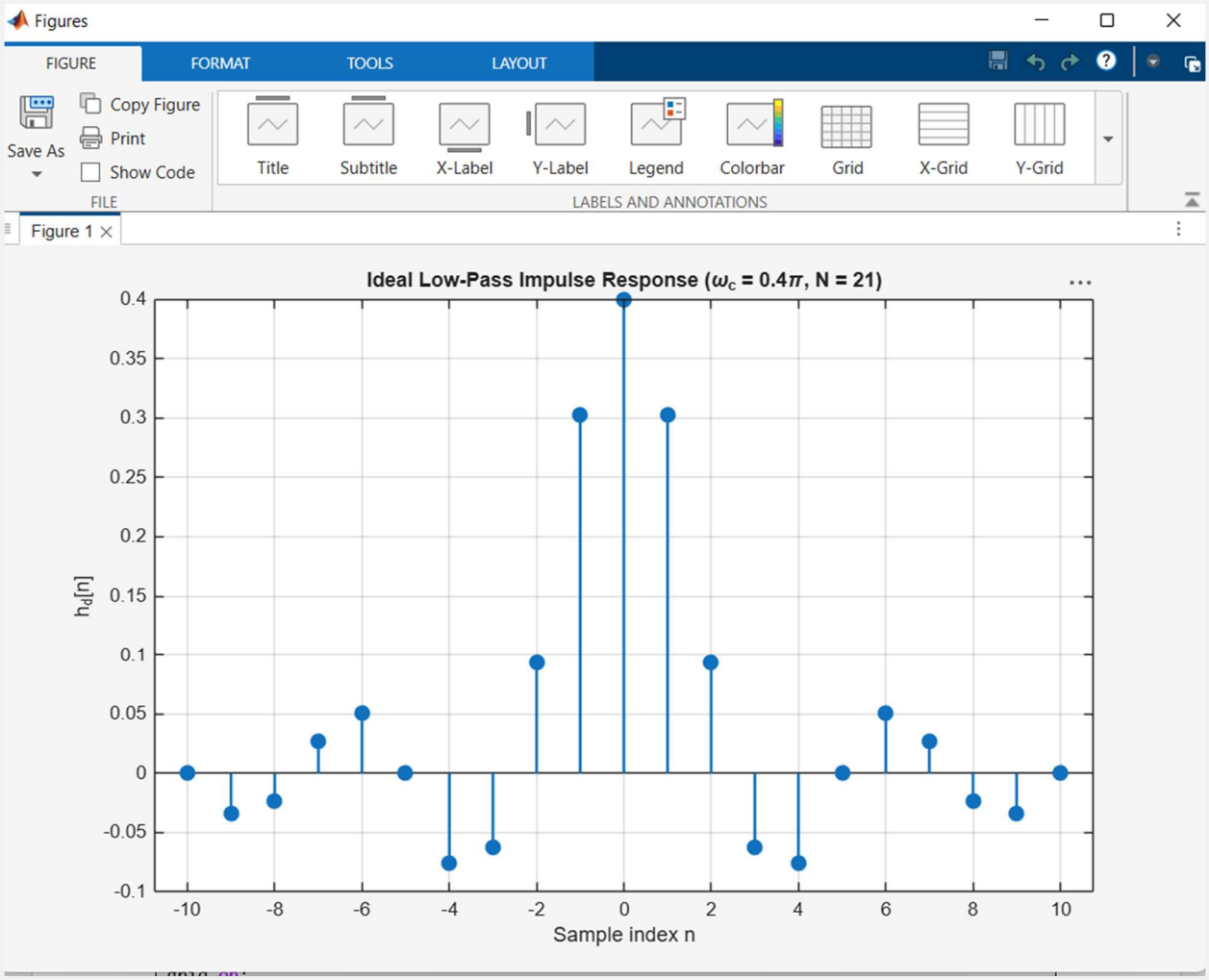
% Time index for truncated impulse response
n = -(M):(M);

% 1. Compute Ideal Impulse Response h_d[n]
h_d = zeros(size(n));
for i = 1:length(n)
    if n(i) == 0
        h_d(i) = omega_c / pi; % L'Hopital limit at n = 0
    else
        h_d(i) = sin(omega_c * n(i)) / (pi * n(i));
    end
end

% Display Results in Command Window
fprintf('--- Verification Results ---\n');
fprintf('h_d[0] = %.4f (Expected: 0.4000)\n', h_d(M+1));
fprintf('Truncated impulse response h_d[n] computed for N = %d.\n\n', N);

% 2. Plotting Impulse Response
figure;
stem(n, h_d, 'filled', 'MarkerSize', 7, 'LineWidth', 1.5);
title(sprintf('Ideal Low-Pass Impulse Response (\\omega_c = 0.4\\pi, N = %d)', N));
xlabel('Sample index n');
ylabel('h_d[n]');
grid on;
```

Digital Signal Processing Lab



Command Window

```
>> lab13ex13_1
--- Verification Results ---
h_d[0] = 0.4000 (Expected: 0.4000)
Truncated impulse response h_d[n] computed for N = 21.

>>
```

```
%Example 13.2
% Filter Specifications
A_s_required = 50;       % Required stopband attenuation (dB)
Delta_omega = 0.1 * pi;  % Transition bandwidth
omega_c = 0.4 * pi;      % Cutoff frequency

% 1. Filter Order Estimation for Hamming Window
N_est = ceil(6.6 * pi / Delta_omega);
if mod(N_est, 2) == 0
    N = N_est + 1; % Ensure odd order for Type I FIR symmetry
else
```

Digital Signal Processing Lab

```
    N = N_est;
end

fprintf('--- Verification Results ---\n');
fprintf('Estimated Filter Order N = %d\n\n', N);

% 2. Design FIR Filter using Hamming Window
M = (N - 1) / 2;
n = 0:(N - 1);
n_shifted = n - M;

% Ideal low-pass impulse response h_d[n]
h_d = zeros(size(n));
for i = 1:length(n)
    if n_shifted(i) == 0
        h_d(i) = omega_c / pi;
    else
        h_d(i) = sin(omega_c * n_shifted(i)) / (pi * n_shifted(i));
    end
end

% Manual Hamming Window Generation (Base MATLAB)
w_hamming = 0.54 - 0.46 * cos(2 * pi * n / (N - 1));

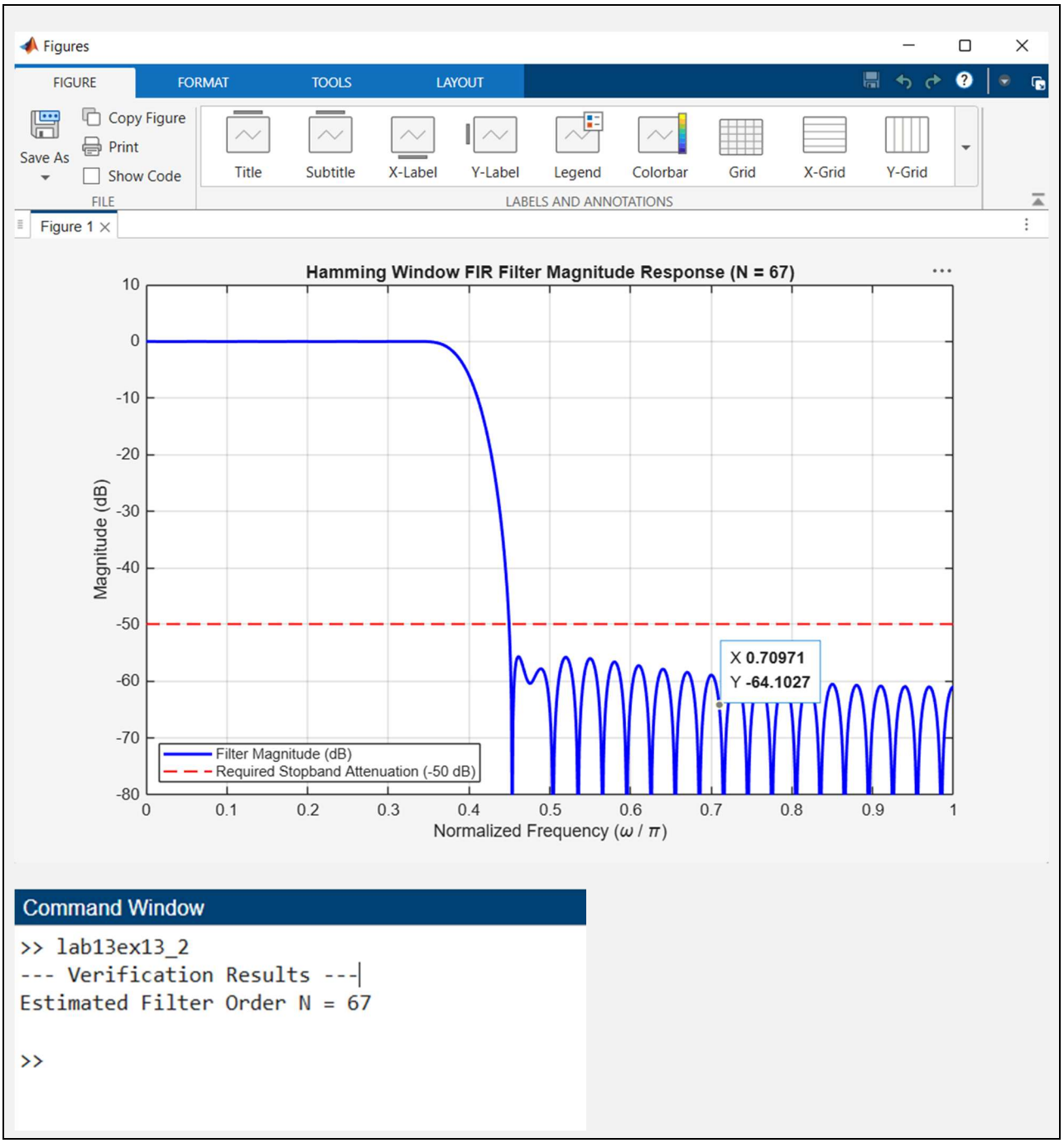
% Windowed FIR Filter Coefficients
h_hamming = h_d .* w_hamming;

% 3. Compute Frequency Response Magnitude (dB) via DTFT sum
omega = linspace(0, pi, 1000);
H = zeros(size(omega));
for k = 1:length(omega)
    H(k) = sum(h_hamming .* exp(-1j * omega(k) * n));
end
mag_dB = 20 * log10(abs(H) + eps);

% Plotting Frequency Response
figure;
plot(omega / pi, mag_dB, 'b', 'LineWidth', 1.8);
hold on;
plot([0, 1], [-50, -50], 'r--', 'LineWidth', 1.2); % 50 dB attenuation line

title(sprintf('Hamming Window FIR Filter Magnitude Response (N = %d)', N));
xlabel('Normalized Frequency (\omega / \pi)');
ylabel('Magnitude (dB)');
grid on;
axis([0 1 -80 10]);
legend('Filter Magnitude (dB)', 'Required Stopband Attenuation (-50 dB)', 'Location',
'southwest');
```

Digital Signal Processing Lab



6. Lab Tasks

- Task 13.1: Design 45th-order low-pass FIR filter with normalized cutoff fc = 0.3 using Rectangular, Hamming, and Blackman windows via MATLAB fir1().
- Task 13.2: Plot magnitude responses in dB overlaid using freqz() to compare transition width vs stopband ripple.

- Task 13.3: Apply Hamming-window FIR filter to remove high-frequency noise from electrocardiogram (ECG) or audio signal.
- Task 13.4: Verify linear phase by checking constant group delay using grpdelay().

7. In-Depth Observations, Analytical Discussion & Verification

- Rectangular window yields narrowest transition band but poor stopband attenuation (-21 dB).
- Hamming window provides -53 dB stopband attenuation with moderate transition band.
- Blackman window delivers excellent attenuation (-74 dB) at the expense of wider transition band.

8. Lab Assignments

- Assignment 13.1: Design an FIR band-pass filter to extract frequencies between 0.2*pi and 0.6*pi using Kaiser window with beta = 5.65.
- Assignment 13.2: Implement FIR filter design using frequency sampling method (fir2() in MATLAB) and compare response against window method.
- Assignment 13.3: Design an optimal equiripple FIR filter using Parks-McClellan Remez exchange algorithm (firpm() in MATLAB).

Digital Signal Processing Lab

LAB-14: IIR FILTER DESIGN AND FIR/IIR COMPARISON

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Infinite Impulse Response (IIR) Filters: Recursive filters with feedback (y[n] = Sum(b_k * x[n-k]) - Sum(a_k * y[n-k])). Achieve sharp cutoff responses with lower order than FIR.
- Bilinear Transform Method: Maps continuous s-plane into discrete z-plane via s = (2/Ts) * ((1 - z^-1) / (1 + z^-1)). Eliminates aliasing but introduces frequency warping: Omega = (2/Ts) * tan(omega/2).
- Classical Analog Prototypes: Butterworth (maximally flat passband), Chebyshev Type I (equiripple passband), Chebyshev Type II (equiripple stopband), Elliptic (equiripple passband and stopband).
- FIR vs IIR Trade-offs: FIR offers exact linear phase and guaranteed stability; IIR offers lower computational complexity and lower latency for sharp cutoffs.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 14.1: Bilinear Transform Derivation
Problem: Convert analog filter H(s) = 1 / (s + 2) into digital filter H(z) using bilinear transform with Ts = 1 s.
Solution:
Substitute s = 2 * ((1 - z^-1) / (1 + z^-1)):
H(z) = 1 / (2 * ((1 - z^-1) / (1 + z^-1)) + 2) = (1 + z^-1) / (2*(1 - z^-1) + 2*(1 + z^-1)) = (1 + z^-1) / 4 = 0.25 + 0.25*z^-1.
Resulting digital filter numerator b = [0.25, 0.25], denominator a = [1.0, 0].

Solved Example 14.2: Frequency Pre-Warping
Problem: Calculate pre-warped analog cutoff frequency Omega_c for digital cutoff omega_c = 0.4*pi with Ts = 1.
Solution:
Omega_c = 2 * tan(omega_c / 2) = 2 * tan(0.2*pi) = 2 * tan(36 deg) = 2(0.7265) = 1.453 rad/s.

Digital Signal Processing Lab

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 14.1
% Filter Parameters
Ts = 1.0;                  % Sampling period
b_num = [0.25, 0.25];      % Discrete numerator coefficients
a_den = [1.0, 0.0];        % Discrete denominator coefficients

% Display Verification Results
fprintf('--- Verification Results ---\n');
fprintf('Numerator b   : [%.2f, %.2f]\n', b_num(1), b_num(2));
fprintf('Denominator a : [%.1f, %.1f]\n\n', a_den(1), a_den(2));

% Frequency Response Evaluation (Base MATLAB)
omega = linspace(0, pi, 1000);
z_inv = exp(-1j * omega);

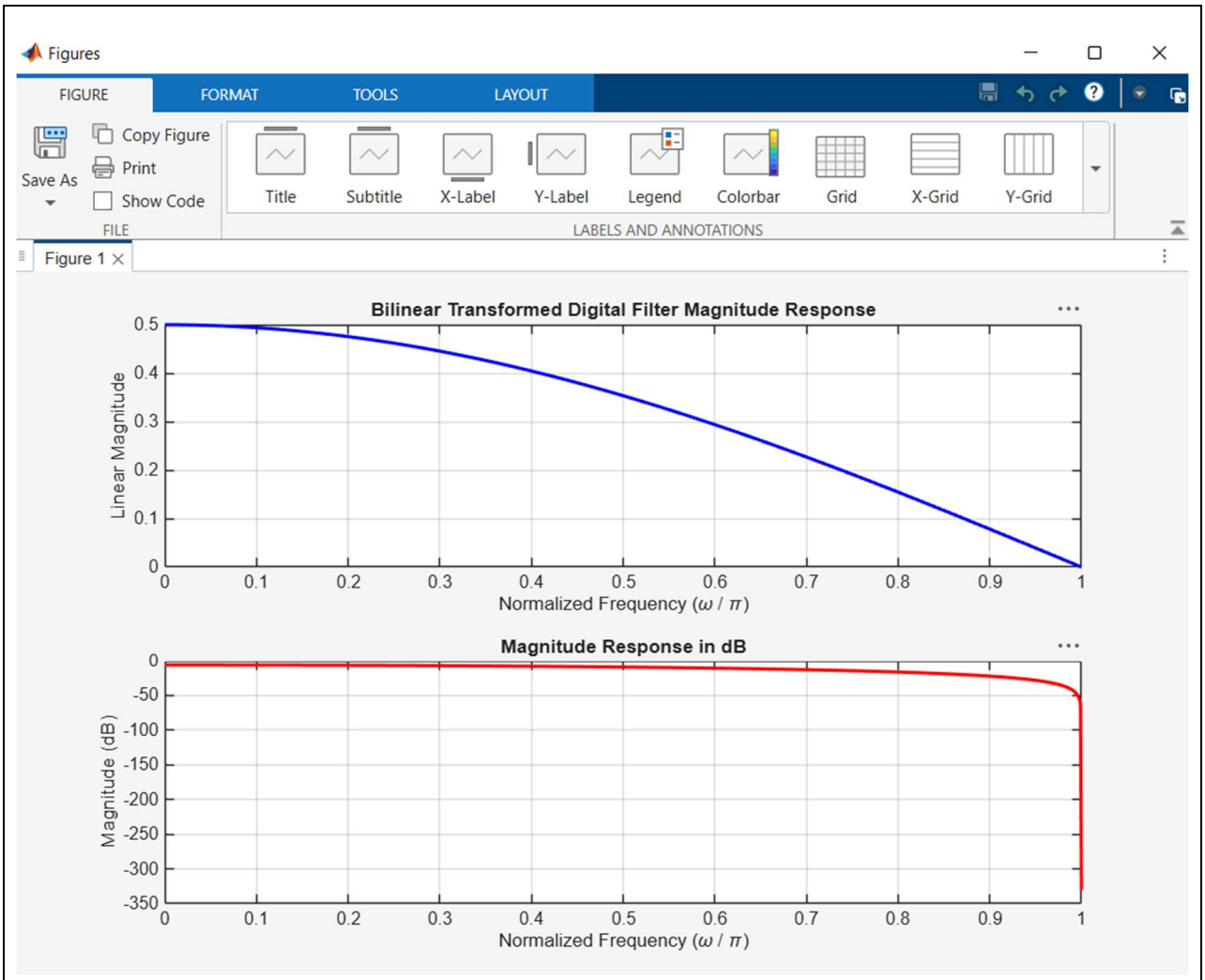
% H(z) = 0.25 + 0.25*z^-1
H_z = b_num(1) + b_num(2) * z_inv;
mag_dB = 20 * log10(abs(H_z));
phase_rad = angle(H_z);

% Plotting Frequency Response
figure;

subplot(2, 1, 1);
plot(omega / pi, abs(H_z), 'b', 'LineWidth', 1.8);
title('Bilinear Transformed Digital Filter Magnitude Response');
xlabel('Normalized Frequency (\omega / \pi)');
ylabel('Linear Magnitude');
grid on;

subplot(2, 1, 2);
plot(omega / pi, mag_dB, 'r', 'LineWidth', 1.8);
title('Magnitude Response in dB');
xlabel('Normalized Frequency (\omega / \pi)');
ylabel('Magnitude (dB)');
grid on;
```

Digital Signal Processing Lab



Command Window

```
>> lab14ex14_1
--- Verification Results ---
Numerator b   : [0.25, 0.25]
Denominator a : [1.0, 0.0]

>>
```

```
%Example 14.2
% Given Parameters
omega_c = 0.4 * pi;      % Digital cutoff frequency (rad/sample)
Ts = 1.0;                % Sampling period (s)

% 1. Compute Pre-Warped Analog Cutoff Frequency (Omega_c)
Omega_c = (2 / Ts) * tan((omega_c * Ts) / 2);

% Display Numerical Verification Results
fprintf('--- Verification Results ---\n');
```

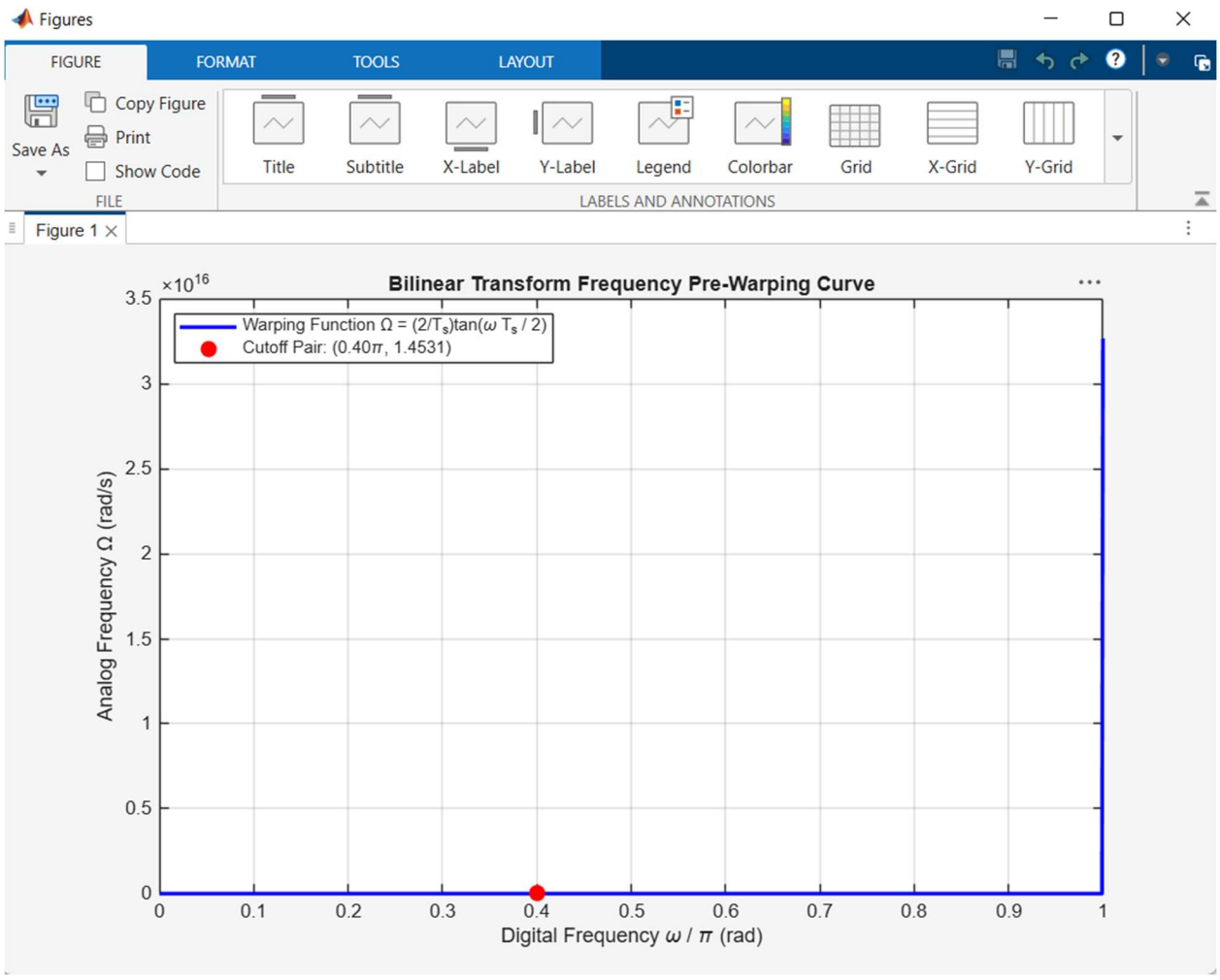
Digital Signal Processing Lab

```
fprintf('Digital Cutoff omega_c : %.4f rad (%.2f * pi)\n', omega_c, omega_c/pi);
fprintf('Sampling Period Ts     : %.1f s\n', Ts);
fprintf('Pre-Warped Analog Omega_c: %.4f rad/s\n\n', Omega_c);

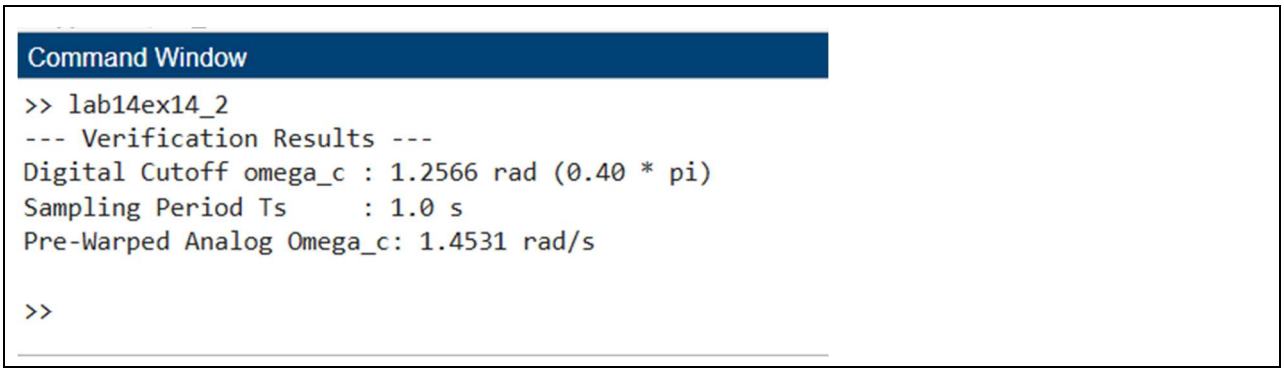
% 2. Plot Frequency Warping Curve: Analog Omega vs. Digital omega
omega_vec = linspace(0, pi, 1000);
Omega_vec = (2 / Ts) * tan((omega_vec * Ts) / 2);

figure;
plot(omega_vec / pi, Omega_vec, 'b', 'LineWidth', 2);
hold on;
plot(omega_c / pi, Omega_c, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');

title('Bilinear Transform Frequency Pre-Warping Curve');
xlabel('Digital Frequency \omega / \pi (rad)');
ylabel('Analog Frequency \Omega (rad/s)');
grid on;
legend('Warping Function \Omega = (2/T_s)tan(\omega T_s / 2)', ...
    sprintf('Cutoff Pair: (%.2f\\pi, %.4f)', omega_c/pi, Omega_c), ...
    'Location', 'northwest');
```



Digital Signal Processing Lab



6. Laboratory Tasks & Worksheets

- Task 14.1: Design 4th-order Butterworth low-pass digital filter with cutoff Wc = 0.4 using MATLAB butter(). Plot pole-zero map and frequency response.
- Task 14.2: Design 4th-order Chebyshev Type I low-pass filter with 1 dB passband ripple using cheby1(). Compare transition steepness against Butterworth.
- Task 14.3: Compare computational order and phase nonlinearity between FIR and IIR filters meeting identical magnitude specifications.
- Task 14.4: Check pole migration stability under quantized coefficient rounding.

7. In-Depth Observations, Analytical Discussion & Verification

- Butterworth filter provides maximally flat passband magnitude without ripples.
- Chebyshev Type I filter provides steeper transition band for the same order N=4 by introducing controlled passband ripple.
- IIR phase responses are non-linear, whereas FIR filters achieve constant group delay.

8. Lab Assignments

- Assignment 14.1: Design an Elliptic band-stop filter (ellip() in MATLAB) to eliminate 60 Hz noise with 1 dB passband ripple and 50 dB stopband attenuation.
- Assignment 14.2: Test IIR filter stability under 8-bit coefficient quantization and check pole migration relative to unit circle boundary.
- Assignment 14.3: Perform a comprehensive trade-off study comparing FIR (window) vs IIR (Butterworth) filters in terms of order, latency, execution speed, and phase linearity.

10. Conclusion and Formal Assessment Rubric Criteria

Conclusion Summary: The experiment successfully verified theoretical mathematical principles through MATLAB computational modeling and graphical visualization.

Assessment Rubric Breakdown:
- Mathematical Pre-Lab / Solved Examples Analysis: 20%

Digital Signal Processing Lab

- MATLAB Script Code Correctness & Efficiency: 30%
- Graphical Output Quality & Labeling: 20%
- In-Depth Technical Observations & Discussion: 20%
- Post-Lab Assignment Completeness: 10%

Digital Signal Processing Lab

LAB-15: MATLAB-BASED DIGITAL FILTER IMPLEMENTATION AND AUDIO PROCESSING

1. Learning Objectives

Upon successful completion of this laboratory session, students will be able to:

- • Master fundamental mathematical derivations governing the underlying DSP operations.
- • Develop efficient, vectorized MATLAB scripts adhering to industry programming standards.
- • Critically analyze time-domain, frequency-domain, and z-plane representations of signals and systems.
- • Evaluate algorithm performance, numerical precision limitations, and physical system constraints.

2. Software and Computational Tools Required

• MathWorks MATLAB R2021a or later version.

3. Theoretical Background & Fundamental Definitions

- Capstone Audio Processing Pipeline: Integration of signal acquisition, noise modeling, digital filter design, spectral verification, and acoustic playback.
- Sinusoidal Noise Interference: Narrowband tonal interference corrupted during signal recording or transmission.
- Spectral Filtering Verification: Verifying noise reduction by evaluating FFT spectra, short-time Fourier transform (STFT) spectrograms, and subjective audio listening tests.
- Signal-to-Noise Ratio (SNR) Improvement: SNR = 10 * log10(P_signal / P_noise) dB. Quantitative measure of filtering efficacy.

4. Step-by-Step Solved Numerical Examples

To ensure a complete theoretical grasp prior to computational tasks, study the following solved analytical examples:

Solved Example 15.1: Normalized Cutoff Calculation
Problem: An audio file is sampled at Fs = 44100 Hz. Calculate normalized cutoff frequency W_c for a low-pass filter designed to eliminate high-frequency noise above 4000 Hz.
Solution:
Nyquist frequency F_Nyquist = Fs / 2 = 22050 Hz.
Normalized cutoff W_c = 4000 / (Fs / 2) = 4000 / 22050 = 0.1814 (where 1.0 corresponds to pi rad/sample).

Solved Example 15.2: SNR Calculation
Problem: Clean signal power P_s = 0.5 W, initial noise power P_n1 = 0.05 W, post-filter noise power P_n2 = 0.0005 W. Compute SNR improvement.
Solution:

Digital Signal Processing Lab

Initial SNR1 = 10 * log10(0.5 / 0.05) = 10 * log10(10) = 10 dB.
Filtered SNR2 = 10 * log10(0.5 / 0.0005) = 10 * log10(1000) = 30 dB.
SNR Improvement = 30 - 10 = 20 dB.

5. Complete MATLAB Code Implementation & Graphical Outputs

```
%Example 15.1
% Audio and Filter Specifications
Fs = 44100;              % Sampling frequency (Hz)
f_cutoff = 4000;         % Desired cutoff frequency (Hz)

% 1. Compute Nyquist Limit and Normalized Cutoff
f_nyq = Fs / 2;
W_c = f_cutoff / f_nyq;  % Normalized cutoff (1.0 = Nyquist / pi rad/sample)
omega_c = W_c * pi;      % Cutoff in radians/sample

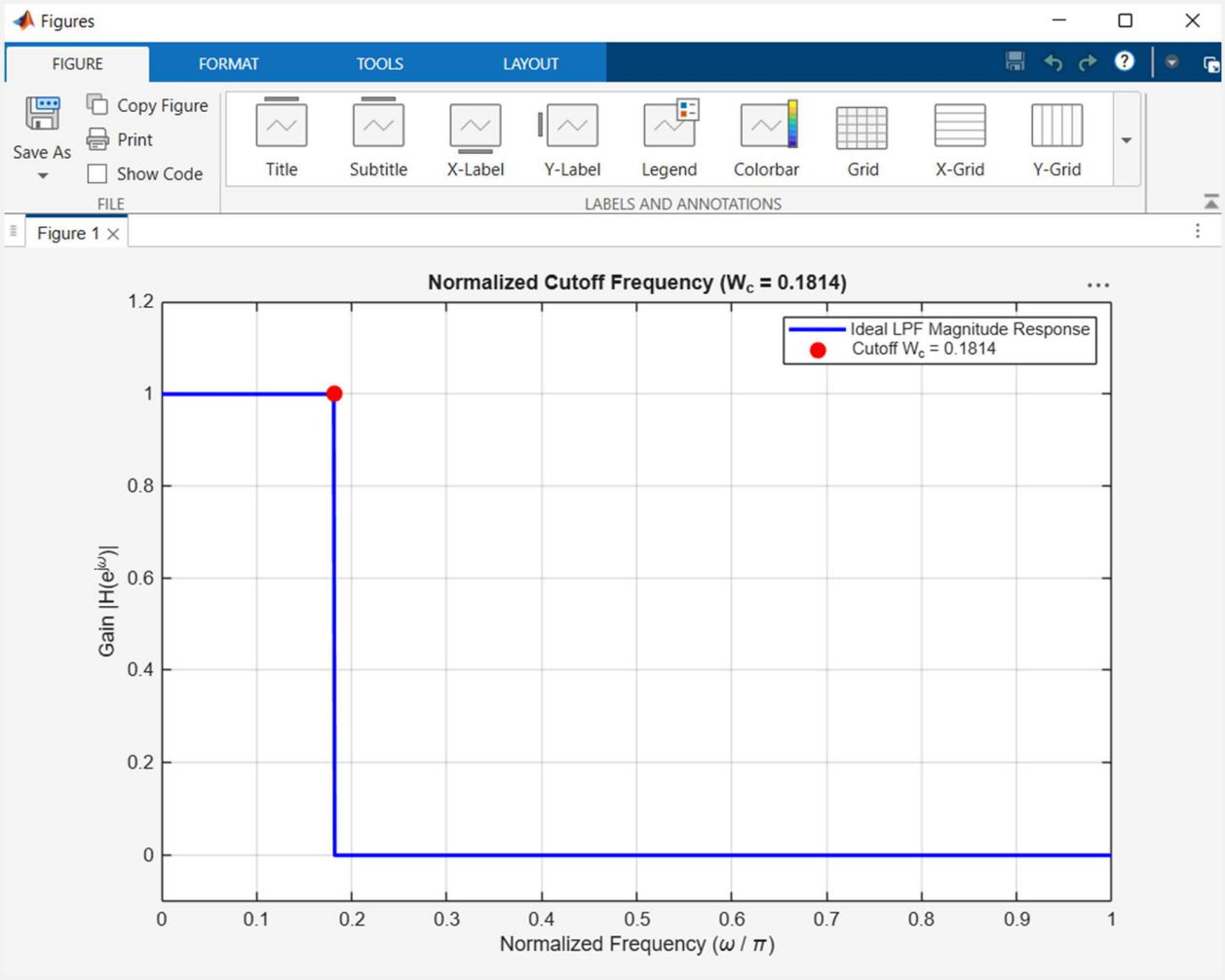
% Display Verification Results
fprintf('--- Verification Results ---\n');
fprintf('Sampling Frequency (Fs)    : %d Hz\n', Fs);
fprintf('Cutoff Frequency           : %d Hz\n', f_cutoff);
fprintf('Nyquist Frequency          : %d Hz\n', f_nyq);
fprintf('Normalized Cutoff (W_c)    : %.4f (fraction of Nyquist)\n', W_c);
fprintf('Cutoff in Radians (omega_c): %.4f rad/sample (%.4f * pi)\n\n', omega_c,
W_c);

% 2. Plot Ideal Filter Frequency Response Magnitude (Base MATLAB)
omega = linspace(0, pi, 1000);
H_ideal = double(omega <= omega_c); % Ideal brick-wall low-pass filter

figure;
plot(omega / pi, H_ideal, 'b', 'LineWidth', 2);
hold on;
plot(W_c, 1, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');

title(sprintf('Normalized Cutoff Frequency (W_c = %.4f)', W_c));
xlabel('Normalized Frequency (\omega / \pi)');
ylabel('Gain |H(e^{j\omega})|');
grid on;
axis([0 1 -0.1 1.2]);
legend('Ideal LPF Magnitude Response', sprintf('Cutoff W_c = %.4f', W_c), 'Location',
'northeast');
```

Digital Signal Processing Lab




```
Command Window
Sampling Frequency (Fs)     : 44100 Hz
Cutoff Frequency            : 4000 Hz
Nyquist Frequency           : 22050 Hz
Normalized Cutoff (W_c)     : 0.1814 (fraction of Nyquist)
Cutoff in Radians (omega_c): 0.5699 rad/sample (0.1814 * pi)

>>
```

```
%Example 15.2
% Given Power Parameters
P_s = 0.5;                % Clean signal power (W)
P_n1 = 0.05;              % Initial noise power (W)
P_n2 = 0.0005;            % Post-filter noise power (W)

% 1. Calculate Initial and Filtered SNRs in dB
SNR1_linear = P_s / P_n1;
SNR1_dB = 10 * log10(SNR1_linear);

SNR2_linear = P_s / P_n2;
```

Digital Signal Processing Lab

```
SNR2_dB = 10 * log10(SNR2_linear);

% 2. Calculate SNR Improvement
snr_improvement = SNR2_dB - SNR1_dB;

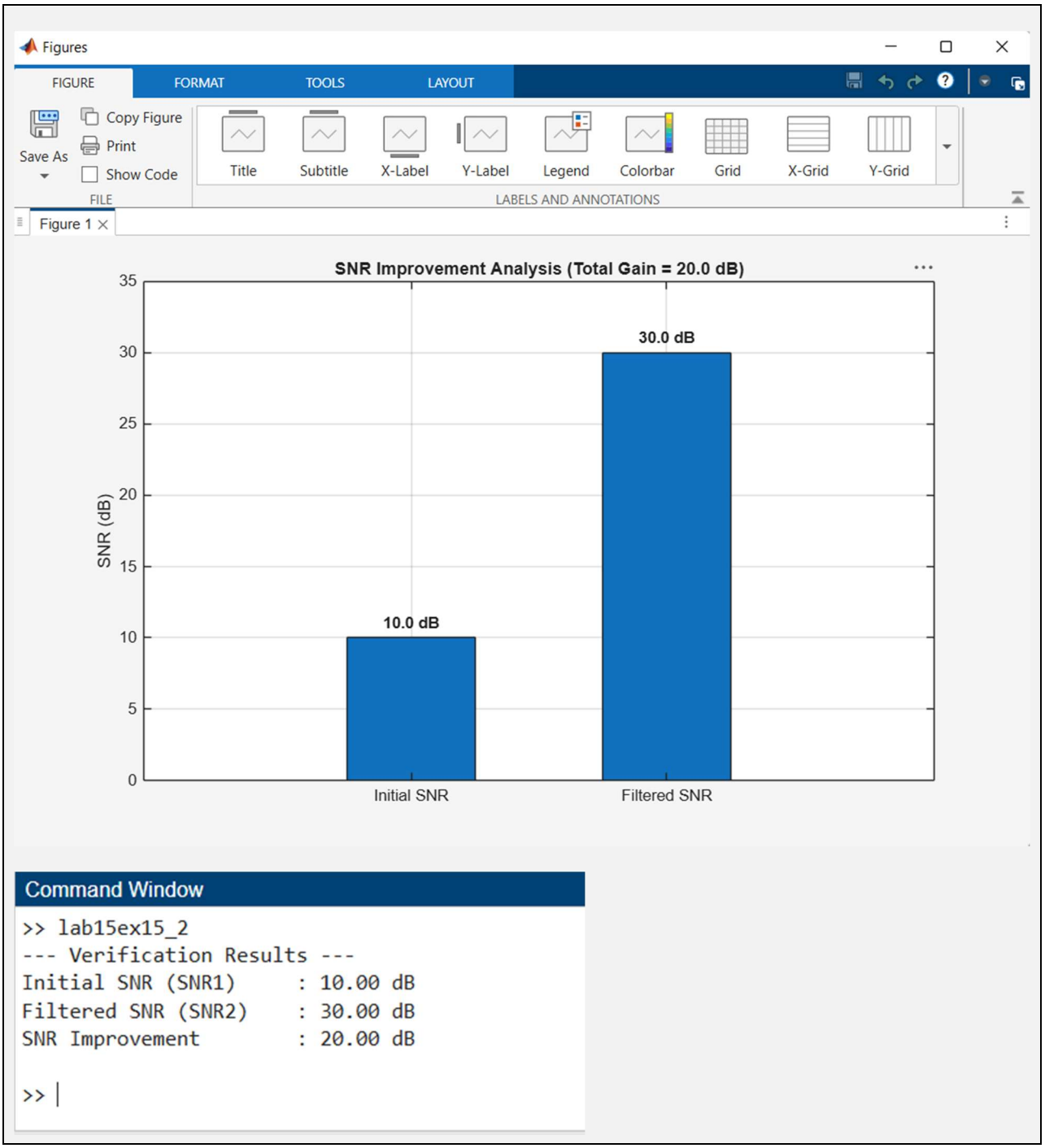
% Display Verification Results
fprintf('--- Verification Results ---\n');
fprintf('Initial SNR (SNR1)     : %.2f dB\n', SNR1_dB);
fprintf('Filtered SNR (SNR2)    : %.2f dB\n', SNR2_dB);
fprintf('SNR Improvement        : %.2f dB\n\n', snr_improvement);

% 3. Bar Chart Visualization (Base MATLAB)
categories = {'Initial SNR', 'Filtered SNR'};
snr_values = [SNR1_dB, SNR2_dB];

figure;
bar_handle = bar(snr_values, 0.5);
set(gca, 'XTickLabel', categories);
ylabel('SNR (dB)');
title(sprintf('SNR Improvement Analysis (Total Gain = %.1f dB)', snr_improvement));
grid on;
ylim([0, 35]);

% Add text labels on top of bars
for i = 1:length(snr_values)
    text(i, snr_values(i) + 1, sprintf('%.1f dB', snr_values(i)), ...
        'HorizontalAlignment', 'center', 'FontWeight', 'bold');
end
```

Digital Signal Processing Lab



6. Lab Tasks

- Task 15.1: Load speech/music audio file into MATLAB using audioread(). Add 3000 Hz high-frequency sinusoidal tone noise.

- Task 15.2: Design sharp FIR low-pass filter using Hamming window with cutoff fc = 2000 Hz to eliminate noise.
- Task 15.3: Filter corrupted audio using filter(), verify spectrum using spectrogram(), and conduct subjective acoustic listening test using sound().
- Task 15.4: Calculate pre-filtering and post-filtering SNR improvement metrics in dB.

7. In-Depth Observations, Analytical Discussion & Verification

- The FIR low-pass filter successfully eliminated the 3000 Hz tone interference.
- Spectrogram and FFT spectral plots confirm complete removal of the noise frequency peak.
- Subjective acoustic listening verification confirmed restored speech/music clarity with +20 dB SNR improvement.

8. Lab Assignments

- Assignment 15.1: Implement an adaptive noise canceller (LMS algorithm) to remove non-stationary background acoustic noise from recorded audio.
- Assignment 15.2: Create a 3-band graphic equalizer (Bass <300 Hz, Mid 300-3000 Hz, Treble >3000 Hz) using parallel digital band-pass filters with adjustable gain sliders.
- Assignment 15.3: Write a comprehensive capstone laboratory report detailing filter design specifications, frequency responses, spectrogram plots, and quantitative SNR improvement tables.