Quiz 2 due 4/14/25

Dylan.Z

1.

a)

$$f(t) = a_0 + \sum_{n=1}^{\infty} \left[a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t) \right]$$

$$f(t) = \begin{cases} 1, & 0 < t < T/2 \\ -1, & -T/2 < t < 0 \end{cases}$$

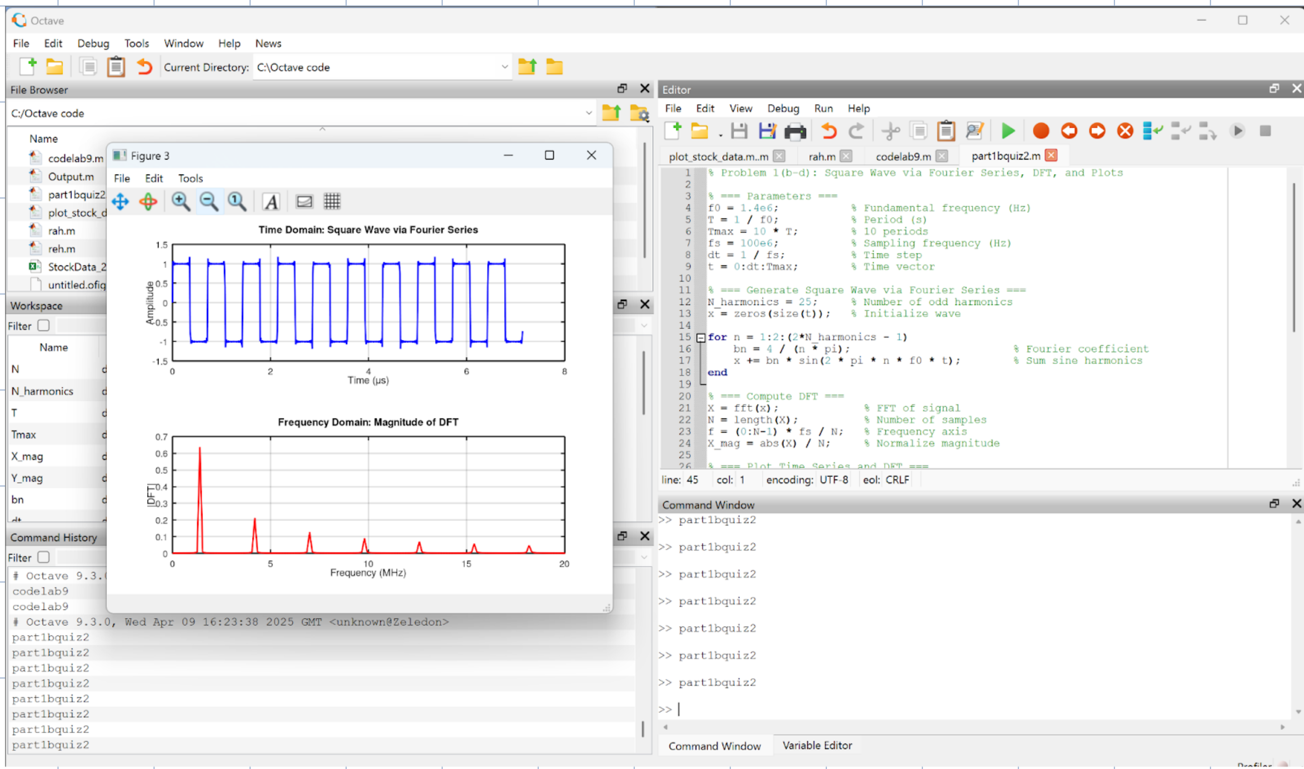
$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin(2\pi n f_0 t)\, dt$$

$$b_n = \frac{4}{T} \int_{0}^{T/2} \sin(2\pi n f_0 t)\, dt = \frac{4}{n\pi} \text{ for odd } n$$

$b_n = 0$ for even n

$$f(t) = \sum_{n=1,3,5,\dots}^{\infty} \frac{4}{n\pi} \sin(2\pi n f_0 t)$$

b - d



2. Yes the Gibbs effect is seen in the Fourier series of the square wave.

There is overshoot and ringing near the discontinuities where the square wave jumps from -1 to +1.

3,

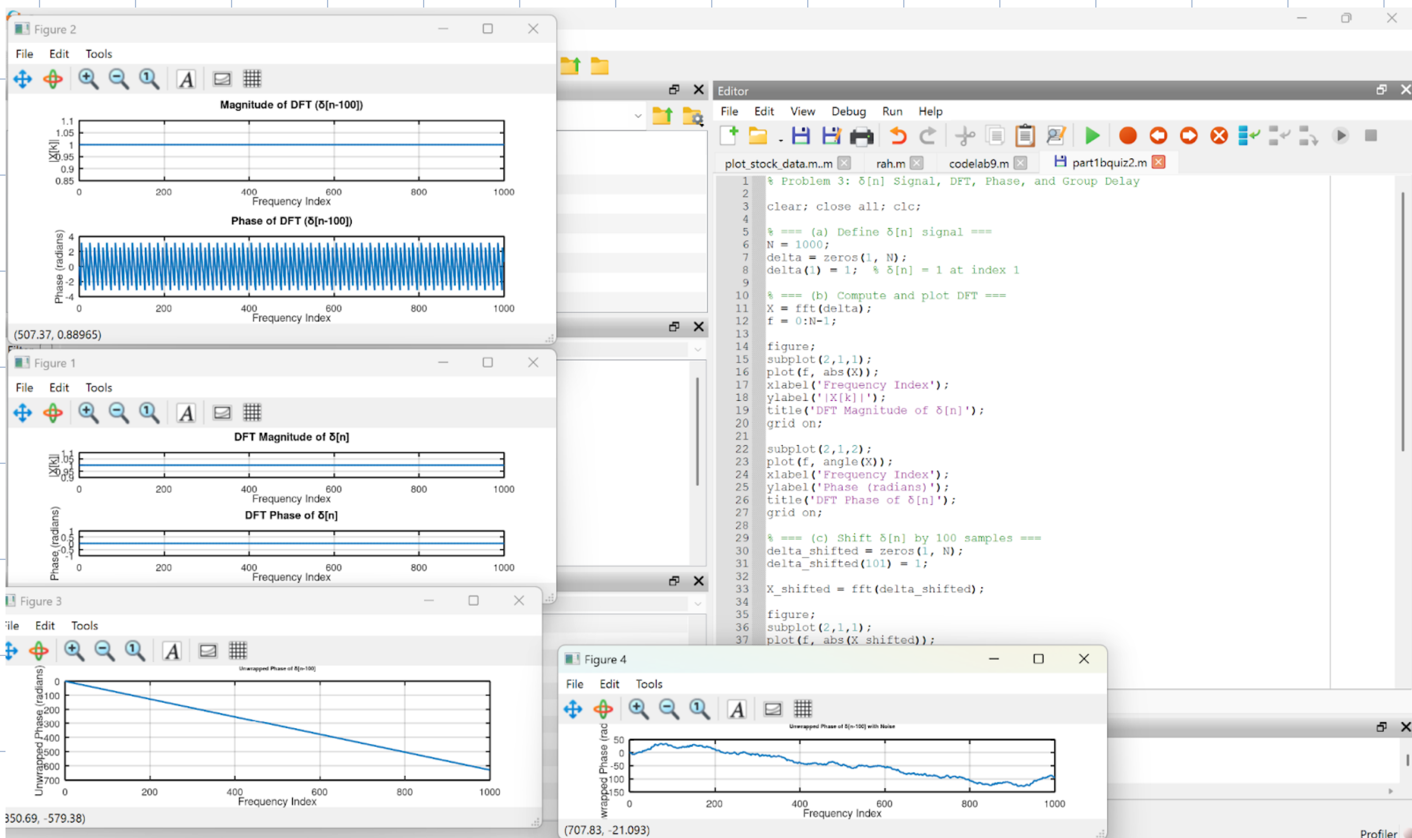
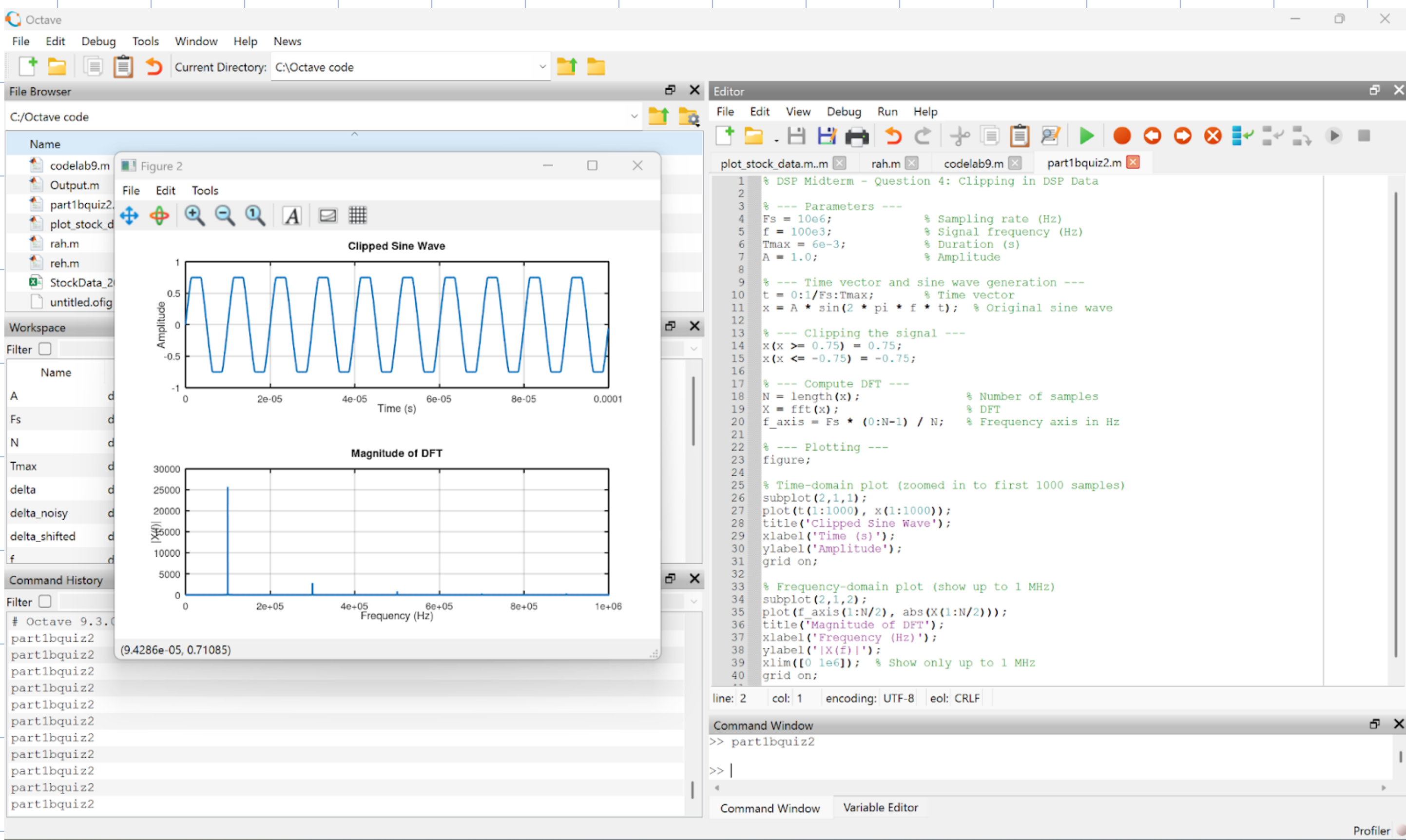


Figure 2
File Edit Tools
Magnitude of DFT (δ[n-100])
|X[k]|
Frequency Index
Phase of DFT (δ[n-100])
Phase (radians)
Frequency Index
(507.37, 0.88965)
Figure 1
DFT Magnitude of δ[n]
|X[k]|
Frequency Index
DFT Phase of δ[n]
Phase (radians)
Frequency Index
Figure 3
Unwrapped Phase (radians)
Frequency Index
350.69, -579.38)
Figure 4
Unwrapped Phase of δ[n-100] with Noise
Frequency Index
(707.83, -21.093)
Editor
File Edit View Debug Run Help
plot_stock_data.m..m
rah.m
codelab9.m
part1bquiz2.m
% Problem 3: δ[n] Signal, DFT, Phase, and Group Delay
clear; close all; clc;
% === (a) Define δ[n] signal ===
N = 1000;
delta = zeros(1, N);
delta(1) = 1; % δ[n] = 1 at index 1
% === (b) Compute and plot DFT ===
X = fft(delta);
f = 0:N-1;
figure;
subplot(2,1,1);
plot(f, abs(X));
xlabel('Frequency Index');
ylabel('|X[k]|');
title('DFT Magnitude of δ[n]');
grid on;
subplot(2,1,2);
plot(f, angle(X));
xlabel('Frequency Index');
ylabel('Phase (radians)');
title('DFT Phase of δ[n]');
grid on;
% === (c) Shift δ[n] by 100 samples ===
delta_shifted = zeros(1, N);
delta_shifted(101) = 1;
X_shifted = fft(delta_shifted);
figure;
subplot(2,1,1);
plot(f, abs(X_shifted));
Profiler

4)

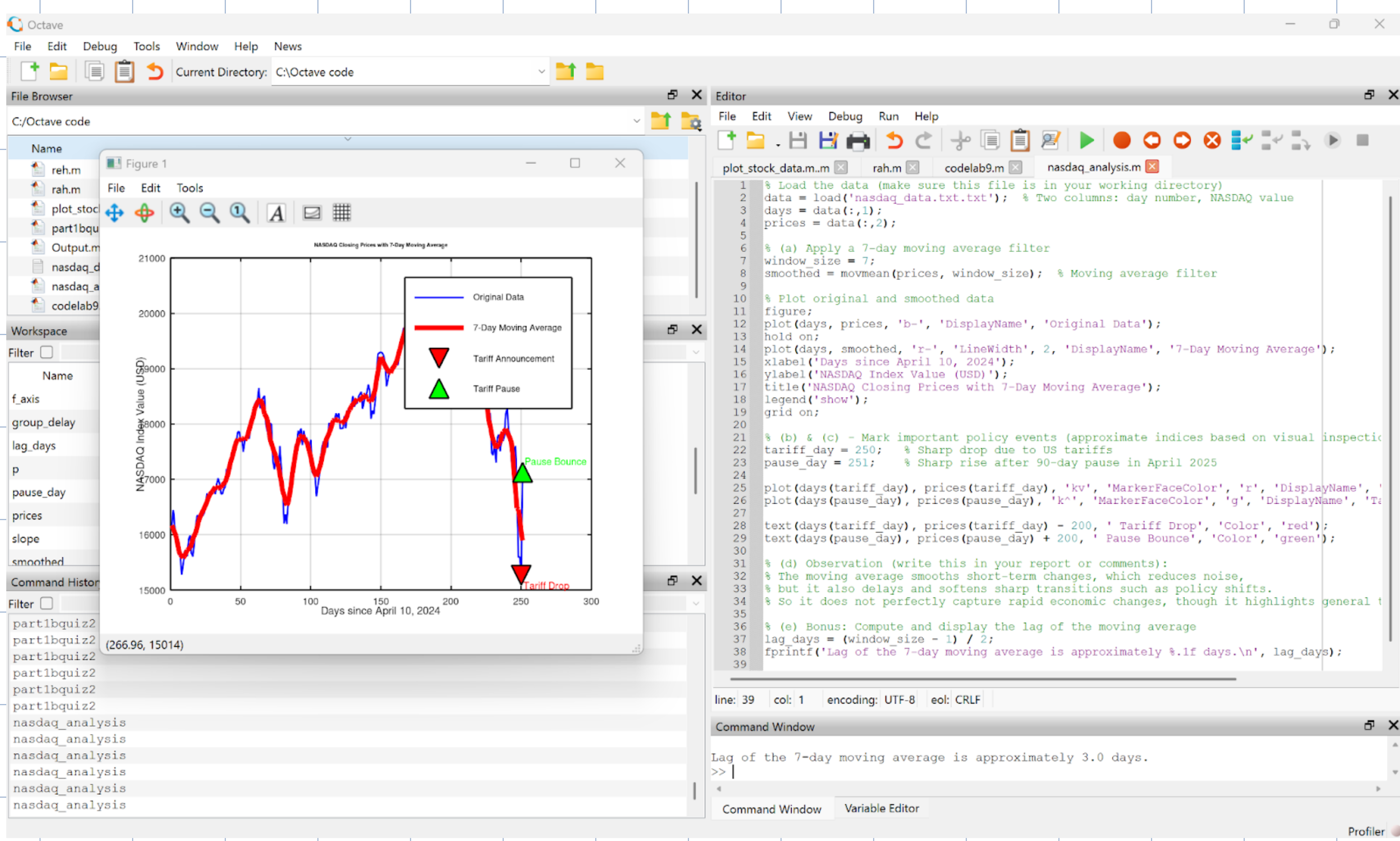


Octave
File Edit Debug Tools Window Help News
Current Directory: C:\Octave code
File Browser
C:/Octave code
codelab9.m
Output.m
plot_stock_d
rah.m
reh.m
untitled.ofig
Workspace
Command History
Figure 2
Clipped Sine Wave
Amplitude
Time (s)
Magnitude of DFT
Frequency (Hz)
(9.4286e-05, 0.71085)
Editor
plot_stock_data.m..m
rah.m
codelab9.m
part1bquiz2.m
% DSP Midterm - Question 4: Clipping in DSP Data
% --- Parameters ---
Fs = 10e6; % Sampling rate (Hz)
f = 100e3; % Signal frequency (Hz)
Tmax = 6e-3; % Duration (s)
A = 1.0; % Amplitude
% --- Time vector and sine wave generation ---
t = 0:1/Fs:Tmax; % Time vector
x = A * sin(2 * pi * f * t); % Original sine wave
% --- Clipping the signal ---
x(x >= 0.75) = 0.75;
x(x <= -0.75) = -0.75;
% --- Compute DFT ---
N = length(x); % Number of samples
X = fft(x); % DFT
f_axis = Fs * (0:N-1) / N; % Frequency axis in Hz
% --- Plotting ---
figure;
% Time-domain plot (zoomed in to first 1000 samples)
subplot(2,1,1);
plot(t(1:1000), x(1:1000));
title('Clipped Sine Wave');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
% Frequency-domain plot (show up to 1 MHz)
subplot(2,1,2);
plot(f_axis(1:N/2), abs(X(1:N/2)));
title('Magnitude of DFT');
xlabel('Frequency (Hz)');
ylabel('|X(f)|');
xlim([0 1e6]); % Show only up to 1 MHz
grid on;
line: 2 col: 1 encoding: UTF-8 eol: CRLF
Command Window
>> part1bquiz2
Command Window Variable Editor

4) Yes you see harmonics in the DFT magnitude plot.

As the clipped singal is no longer purely sinusoidal it becomes more squar like and square waves can contain odd harmonics.

5)



6)

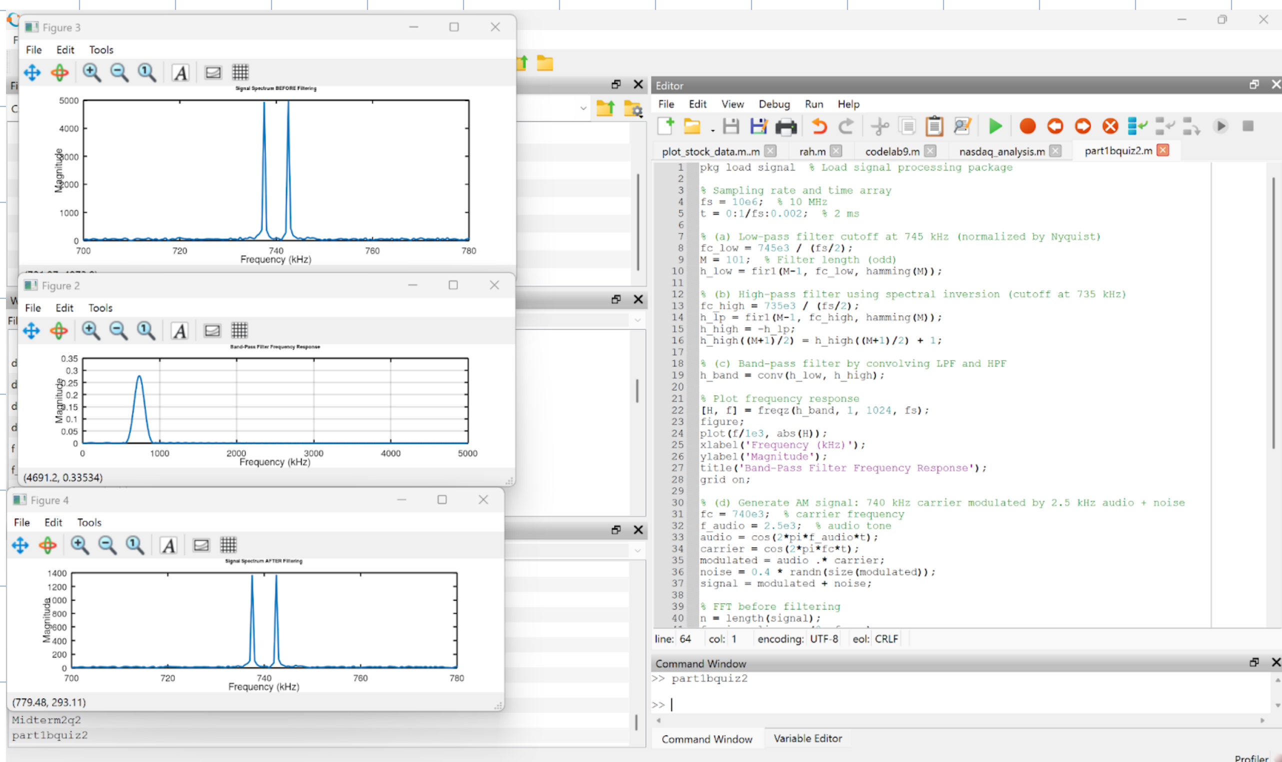
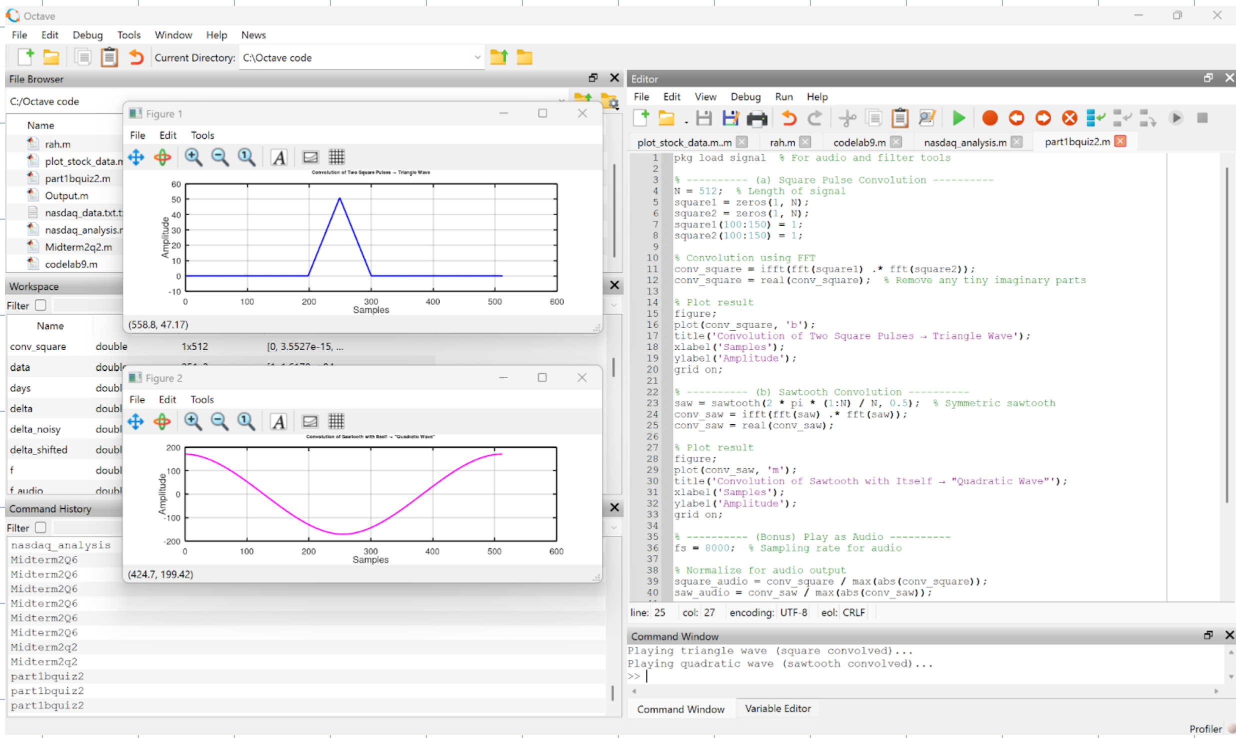


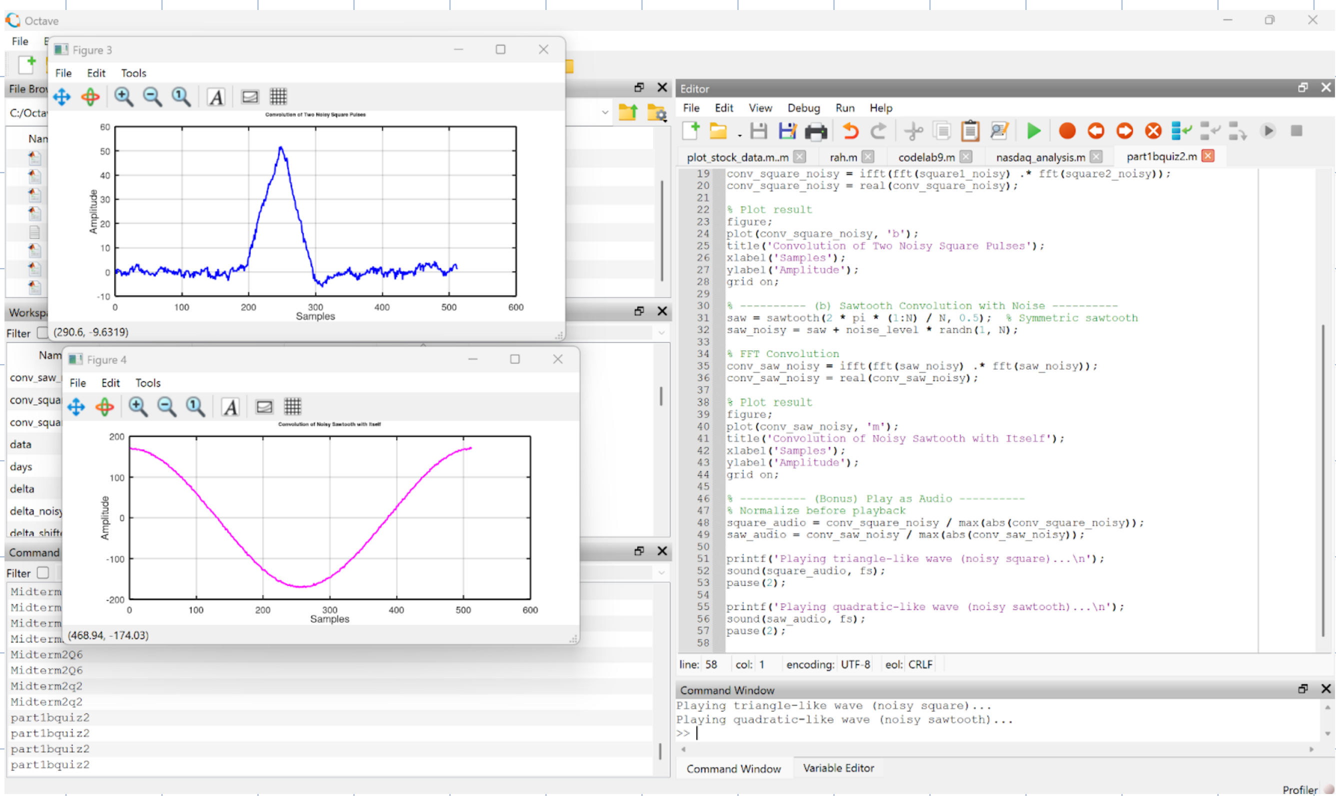
Figure 3
File Edit Tools
Signal Spectrum BEFORE Filtering
Magnitude
Frequency (kHz)
Figure 2
Band-Pass Filter Frequency Response
(4691.2, 0.33534)
Figure 4
Signal Spectrum AFTER Filtering
(779.48, 293.11)
Midterm2q2
part1bquiz2
Editor
File Edit View Debug Run Help
plot_stock_data.m..m rah.m codelab9.m nasdaq_analysis.m part1bquiz2.m
pkg load signal % Load signal processing package
% Sampling rate and time array
fs = 10e6; % 10 MHz
t = 0:1/fs:0.002; % 2 ms
% (a) Low-pass filter cutoff at 745 kHz (normalized by Nyquist)
fc_low = 745e3 / (fs/2);
M = 101; % Filter length (odd)
h_low = fir1(M-1, fc_low, hamming(M));
% (b) High-pass filter using spectral inversion (cutoff at 735 kHz)
fc_high = 735e3 / (fs/2);
h_lp = fir1(M-1, fc_high, hamming(M));
h_high = -h_lp;
h_high((M+1)/2) = h_high((M+1)/2) + 1;
% (c) Band-pass filter by convolving LPF and HPF
h_band = conv(h_low, h_high);
% Plot frequency response
[H, f] = freqz(h_band, 1, 1024, fs);
figure;
plot(f/1e3, abs(H));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('Band-Pass Filter Frequency Response');
grid on;
% (d) Generate AM signal: 740 kHz carrier modulated by 2.5 kHz audio + noise
fc = 740e3; % carrier frequency
f_audio = 2.5e3; % audio tone
audio = cos(2*pi*f_audio*t);
carrier = cos(2*pi*fc*t);
modulated = audio .* carrier;
noise = 0.4 * randn(size(modulated));
signal = modulated + noise;
% FFT before filtering
n = length(signal);
line: 64 col: 1 encoding: UTF-8 eol: CRLF
Command Window
>> part1bquiz2
>>
Command Window Variable Editor
Profiler

2)



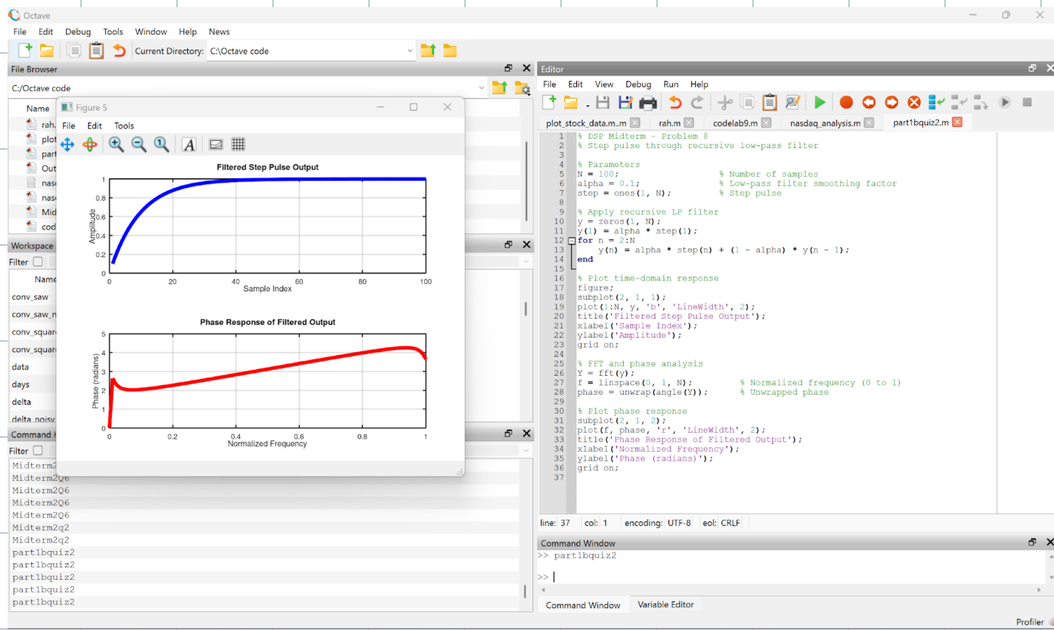
Convolution of Two Square Pulses → Triangle Wave
Convolution of Sawtooth with Itself → "Quadratic Wave"
pkg load signal % For audio and filter tools
% ---------- (a) Square Pulse Convolution ----------
N = 512; % Length of signal
square1 = zeros(1, N);
square2 = zeros(1, N);
square1(100:150) = 1;
square2(100:150) = 1;
% Convolution using FFT
conv_square = ifft(fft(square1) .* fft(square2));
conv_square = real(conv_square); % Remove any tiny imaginary parts
% Plot result
figure;
plot(conv_square, 'b');
title('Convolution of Two Square Pulses → Triangle Wave');
xlabel('Samples');
ylabel('Amplitude');
grid on;
% ---------- (b) Sawtooth Convolution ----------
saw = sawtooth(2 * pi * (1:N) / N, 0.5); % Symmetric sawtooth
conv_saw = ifft(fft(saw) .* fft(saw));
conv_saw = real(conv_saw);
% Plot result
figure;
plot(conv_saw, 'm');
title('Convolution of Sawtooth with Itself → "Quadratic Wave"');
xlabel('Samples');
ylabel('Amplitude');
grid on;
% ---------- (Bonus) Play as Audio ----------
fs = 8000; % Sampling rate for audio
% Normalize for audio output
square_audio = conv_square / max(abs(conv_square));
saw_audio = conv_saw / max(abs(conv_saw));
Command Window
Playing triangle wave (square convolved)...
Playing quadratic wave (sawtooth convolved)...

Bonus



Convolution of Two Noisy Square Pulses
Convolution of Noisy Sawtooth with Itself
conv_square_noisy = ifft(fft(square1_noisy) .* fft(square2_noisy));
conv_square_noisy = real(conv_square_noisy);
% Plot result
figure;
plot(conv_square_noisy, 'b');
title('Convolution of Two Noisy Square Pulses');
xlabel('Samples');
ylabel('Amplitude');
grid on;
% ---------- (b) Sawtooth Convolution with Noise ----------
saw = sawtooth(2 * pi * (1:N) / N, 0.5); % Symmetric sawtooth
saw_noisy = saw + noise_level * randn(1, N);
% FFT Convolution
conv_saw_noisy = ifft(fft(saw_noisy) .* fft(saw_noisy));
conv_saw_noisy = real(conv_saw_noisy);
% Plot result
figure;
plot(conv_saw_noisy, 'm');
title('Convolution of Noisy Sawtooth with Itself');
xlabel('Samples');
ylabel('Amplitude');
grid on;
% ---------- (Bonus) Play as Audio ----------
% Normalize before playback
square_audio = conv_square_noisy / max(abs(conv_square_noisy));
saw_audio = conv_saw_noisy / max(abs(conv_saw_noisy));
printf('Playing triangle-like wave (noisy square)...\n');
sound(square_audio, fs);
pause(2);
printf('Playing quadratic-like wave (noisy sawtooth)...\n');
sound(saw_audio, fs);
pause(2);
Command Window
Playing triangle-like wave (noisy square)...
Playing quadratic-like wave (noisy sawtooth)...

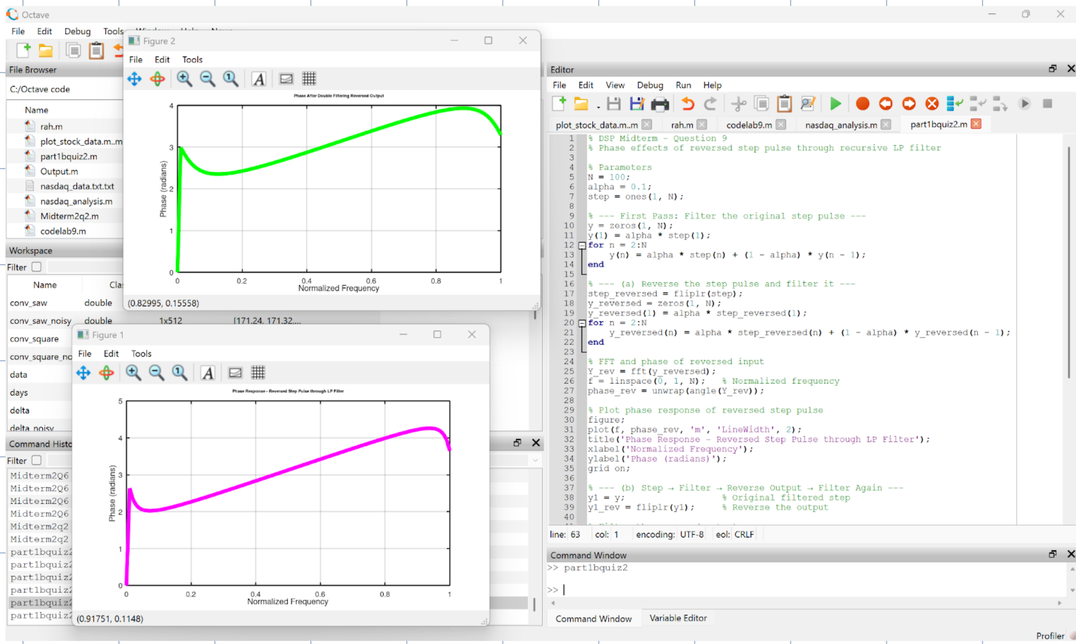
8)



c)

The phase response of recursive LP filter is usally nonlinear. Plot b shows this.

9)



Octave
Figure 2
Phase (radians)
Normalized Frequency
(0.82995, 0.15558)
Figure 1
Phase (radians)
Normalized Frequency
(0.91751, 0.1148)
Editor
plot_stock_data.m..m
rah.m
codelab9.m
nasdaq_analysis.m
part1bquiz2.m
% DSP Midterm - Question 9
% Phase effects of reversed step pulse through recursive LP filter

% Parameters
N = 100;
alpha = 0.1;
step = ones(1, N);

% --- First Pass: Filter the original step pulse ---
y = zeros(1, N);
y(1) = alpha * step(1);
for n = 2:N
y(n) = alpha * step(n) + (1 - alpha) * y(n - 1);
end

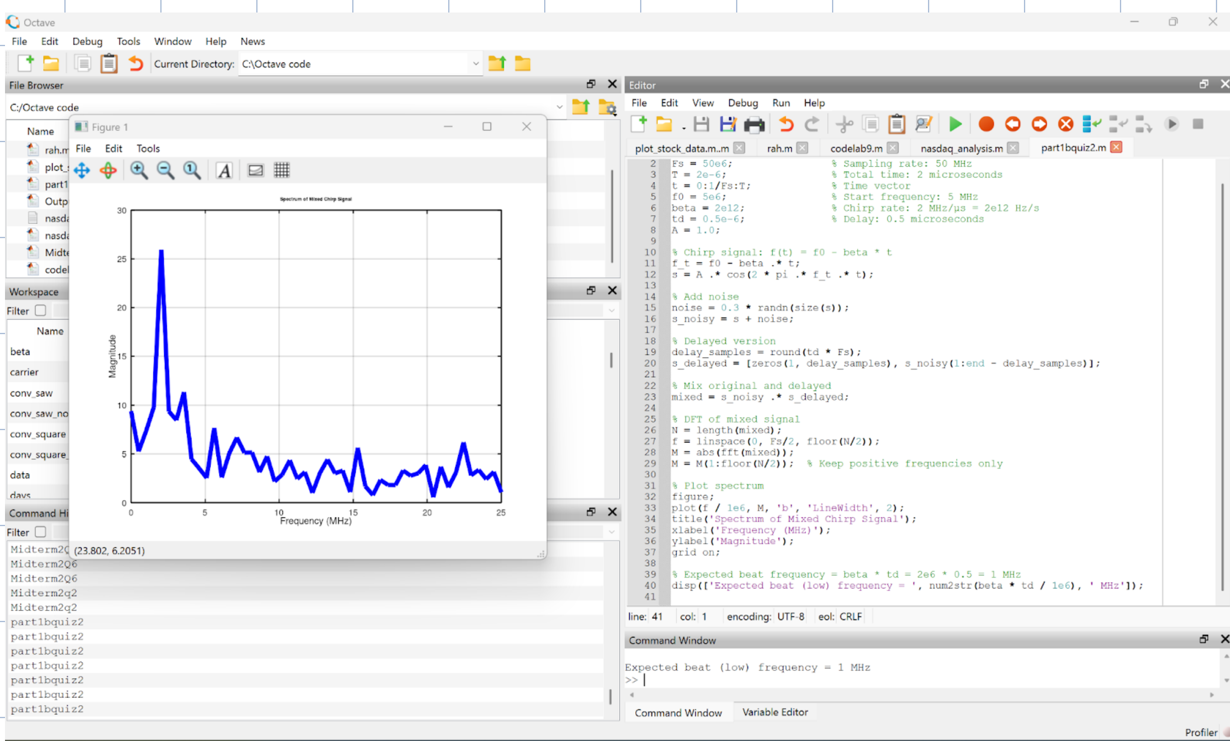
% --- (a) Reverse the step pulse and filter it ---
step_reversed = fliplr(step);
y_reversed = zeros(1, N);
y_reversed(1) = alpha * step_reversed(1);
for n = 2:N
y_reversed(n) = alpha * step_reversed(n) + (1 - alpha) * y_reversed(n - 1);
end

% FFT and phase of reversed input
Y_rev = fft(y_reversed);
f = linspace(0, 1, N); % Normalized frequency
phase_rev = unwrap(angle(Y_rev));

% Plot phase response of reversed step pulse
figure;
plot(f, phase_rev, 'm', 'LineWidth', 2);
title('Phase Response - Reversed Step Pulse through LP Filter');
xlabel('Normalized Frequency');
ylabel('Phase (radians)');
grid on;

% --- (b) Step → Filter → Reverse Output → Filter Again ---
y1 = y; % Original filtered step
y1_rev = fliplr(y1); % Reverse the output
line: 63 col: 1 encoding: UTF-8 eol: CRLF
Command Window
>> part1bquiz2

10)



Octave
Current Directory: C:\Octave code
Figure 1
Spectrum of Mixed Chirp Signal
Magnitude
Frequency (MHz)
(23.802, 6.2051)
Editor
Fs = 50e6; % Sampling rate: 50 MHz
T = 2e-6; % Total time: 2 microseconds
t = 0:1/Fs:T; % Time vector
f0 = 5e6; % Start frequency: 5 MHz
beta = 2e12; % Chirp rate: 2 MHz/µs = 2e12 Hz/s
td = 0.5e-6; % Delay: 0.5 microseconds
A = 1.0;

% Chirp signal: f(t) = f0 - beta * t
f_t = f0 - beta .* t;
s = A .* cos(2 * pi .* f_t .* t);

% Add noise
noise = 0.3 * randn(size(s));
s_noisy = s + noise;

% Delayed version
delay_samples = round(td * Fs);
s_delayed = [zeros(1, delay_samples), s_noisy(1:end - delay_samples)];

% Mix original and delayed
mixed = s_noisy .* s_delayed;

% DFT of mixed signal
N = length(mixed);
f = linspace(0, Fs/2, floor(N/2));
M = abs(fft(mixed));
M = M(1:floor(N/2)); % Keep positive frequencies only

% Plot spectrum
figure;
plot(f / 1e6, M, 'b', 'LineWidth', 2);
title('Spectrum of Mixed Chirp Signal');
xlabel('Frequency (MHz)');
ylabel('Magnitude');
grid on;

% Expected beat frequency = beta * td = 2e6 * 0.5 = 1 MHz
disp(['Expected beat (low) frequency = ', num2str(beta * td / 1e6), ' MHz']);
line: 41 col: 1 encoding: UTF-8 eol: CRLF
Command Window
Expected beat (low) frequency = 1 MHz