1. Develop a MATLAB program to synthesize a square wave using Fourier series, applicable in creating custom sounds and analyzing audio signals.

- **Define parameters** such as sampling rate, fundamental frequency, duration, and number of harmonics.
- **Generate a time vector** based on the sampling frequency and duration.
- **Synthesize the square wave** using the Fourier series by summing odd harmonics with decreasing amplitudes.
- **Plot the waveform** in the time domain to visualize how closely the approximation resembles a square wave.
- **Analyze and plot the frequency spectrum** using FFT to observe the harmonic content of the synthesized signal.

```
clc;
clear;
close all;
Fs = 44100;
f0 = 440;
T = 1;
N = 10;
t = 0:1/Fs:T;

x = zeros(size(t));
for k = 1:2:N
   x = x + (1/k) * sin(2 * pi * k * f0 * t);
end
x = (4/pi) * x;

figure;
plot(t, x, 'b');
xlabel('Time (s)');
ylabel('Amplitude');
title(['Fourier Series Approximation of Square Wave (', num2str(N), ' Harmonics)']);
grid on;
xlim([0, 3/f0]); % Show three periods
% Play the sound
sound(x, Fs);
% Frequency domain analysis
X = abs(fft(x));
f = linspace(0, Fs, length(X));
% Plot the spectrum
figure;
plot(f(1:length(X)/2), X(1:length(X)/2));
```

```
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Spectrum of Synthesized Square Wave');
grid on;
```

---

2 Develop a MATLAB program to analyze ECG signals using Fourier and Z-Transforms, helping to identify heart rate variability (HW).

- **Load and preprocess the ECG image** by converting it to grayscale, binarizing it, and detecting edges to extract waveform data.
- **Extract the ECG signal** by locating waveform pixel positions, sorting them, and converting them into a normalized 1D signal over time.
- **Analyze in frequency domain** using FFT to understand the spectral components of the extracted ECG signal.
- **Compute HRV (Heart Rate Variability)** by detecting peaks (heartbeats), calculating RR intervals, and evaluating their standard deviation.
- **Visualize and save results** with plots for the image, extracted signal, FFT, and saving HRV data for further use.

```
clc; clear; close all;
%% 1. Simulated ECG Signal (Predefined)
fs = 250;                  % Sampling rate (Hz)
t = 0:1/fs:10;              % 10 seconds duration
% Simulated ECG-like waveform: combination of sinusoids + noise
ec_signal = 1.5*sin(2*pi*1.2*t) + 0.5*sin(2*pi*3*t);
ec_signal = ec_signal + 0.1*randn(size(t));        % Add slight noise
%% 2. Fourier Transform Analysis
N = length(ec_signal);
f = (0:N-1) * (fs/N);        % Frequency vector
ecq_fft = abs(fft(ec_signal)); % FFT magnitude
%% 3. Z-Transform Demo (Symbolic for theory, not on long numeric signals)
disp('--- Z-Transform Educational Demo ---');
syms z n;
x = (1/2)^n * heaviside(n);   % Simple symbolic sequence
Xz = ztrans(x, z, n);        % Z-transform
disp('Z-transform of (1/2)^n:');
pretty(Xz);
%% 4. Heart Rate Variability (HRV) Analysis
% Peak detection simulates R-peaks in ECG
[pks, locs] = findpeaks(ec_signal, 'MinPeakHeight', 1, 'MinPeakDistance', 0.6*fs);
RR_intervals = diff(locs) / fs;     % RR intervals in seconds
```

```
HRV = std(RR_intervals);          % Standard deviation of RR intervals
%% 5. Visualization
figure('Name','ECG Signal Analysis','NumberTitle','off');
subplot(3,1,1);
plot(t, ec_signal);
title('Simulated ECG Signal');
xlabel('Time (s)'); ylabel('Amplitude'); grid on;
subplot(3,1,2);
plot(f(1:round(N/2)), ecq_fft(1:round(N/2)));
title('FFT of ECG Signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude'); grid on;
subplot(3,1,3);
stem(RR_intervals, 'filled');
title('RR Intervals');
xlabel('Interval #'); ylabel('Time (s)'); grid on;
%% 6. Display HRV
disp(['Estimated Heart Rate Variability (HRV): ', num2str(HRV), ' sec']);
%% 7. Save HRV Data
save('HRV_results.mat', 'RR_intervals', 'HRV');
```

—----------

3 Develop a MATLAB program to generate and visualize common continuous-time and discrete-time signals (step, impulse, exponential, ramp, sine), analyzing and comparing their properties in both domains to model and understand the behavior of a first-order RC circuit's response to various inputs.

- First, define the time vectors for both continuous (t) and discrete (n) signals to specify their time range.
- Next, generate basic continuous-time signals like step, impulse, exponential, ramp, and sine using the t vector.
- Then, generate the corresponding discrete-time signals using the n vector to simulate digital behavior.
- After that, plot all the continuous-time signals using the plot() function in separate subplots for better visualization.
- Finally, display the discrete-time signals using the stem() function to clearly show sampled values and label each plot.

```
clc;
clear;
% Time vectors
t = 0:0.01:5;        % Continuous time
n = 0:1:50;          % Discrete time
```

```
RC = 1;              % Time constant for RC circuit
% --- CONTINUOUS-TIME SIGNALS ---
% Step signal
u_ct = ones(size(t));
% Impulse signal (approximated)
imp_ct = zeros(size(t));
imp_ct(t==0) = 1;
% Exponential signal
exp_ct = exp(-t/RC);
% Ramp signal
ramp_ct = t;
% Sine wave
sine_ct = sin(2*pi*1*t);
% --- DISCRETE-TIME SIGNALS ---
% Step signal
u_dt = ones(size(n));
% Impulse signal
imp_dt = zeros(size(n));
imp_dt(n==0) = 1;
% Exponential signal
exp_dt = exp(-n/RC);
% Ramp signal
ramp_dt = n;
% Sine wave
sine_dt = sin(2*pi*1*n/length(n)*10);  % adjust frequency
% --- PLOTTING ---
figure('Name','Continuous-Time Signals','NumberTitle','off');
subplot(3,2,1); plot(t, u_ct); title('Step (CT)'); grid on;
subplot(3,2,2); plot(t, imp_ct); title('Impulse (CT)'); grid on;
subplot(3,2,3); plot(t, exp_ct); title('Exponential (CT)'); grid on;
subplot(3,2,4); plot(t, ramp_ct); title('Ramp (CT)'); grid on;
subplot(3,2,5); plot(t, sine_ct); title('Sine (CT)'); grid on;
figure('Name','Discrete-Time Signals','NumberTitle','off');
subplot(3,2,1); stem(n, u_dt); title('Step (DT)'); grid on;
subplot(3,2,2); stem(n, imp_dt); title('Impulse (DT)'); grid on;
subplot(3,2,3); stem(n, exp_dt); title('Exponential (DT)'); grid on;
subplot(3,2,4); stem(n, ramp_dt); title('Ramp (DT)'); grid on;
subplot(3,2,5); stem(n, sine_dt); title('Sine (DT)'); grid on;
```

---

4. Develop a MATLAB program to analyze the frequency response of a digital filter, visualizing its magnitude and phase characteristics to evaluate its performance and key parameters.

- Begin by designing a low-pass Butterworth filter using the butter function, specifying the order and normalized cutoff frequency.
- Compute the frequency response of the filter using the freqz function to get both magnitude and phase over 512 frequency points.
- Prepare to visualize the response by setting up a figure with two subplots—one for magnitude and one for phase.
- In the first subplot, plot the magnitude of the frequency response (|H(w)|) against the normalized frequency to observe filter behavior.
- In the second subplot, plot the phase response to understand how the filter alters the phase of input signals across frequencies.

```
clc; clear; close all;
% Define a simple low-pass digital filter (Butterworth)
[b, a] = butter(3, 0.4);  % 3rd order, cutoff at 0.4*pi rad/sample
% Frequency response
[H, w] = freqz(b, a, 512);  % 512 frequency points
% Plot magnitude and phase
subplot(2,1,1);
plot(w/pi, abs(H));
title('Magnitude Response');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('|H(w)|'); grid on;
subplot(2,1,2);
plot(w/pi, angle(H));
title('Phase Response');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Phase (radians)'); grid on;
```

—----------

5 Develop a MATLAB program to compute and visualize the Fourier and Z-transforms of a sampled audio signal, analyzing its frequency content and relating the pole-zero plot of the Z-transform to the presence of specific frequencies and the overall stability of the digital representation.

- Load a mono audio signal (first 1024 samples) from a .wav file using audioread.
- Compute the Fourier Transform (FFT) of the signal to analyze its frequency components.
- Plot the magnitude of the FFT to observe the frequency spectrum of the audio signal.
- Generate a pole-zero plot using zplane to visualize the Z-transform of the signal.

- Finally, plot the time-domain signal to visualize the original audio waveform over time.

Add audio file - [x, fs] = audioread('sample-3s.wav');

```
clc; clear; close all;
%% 1. Load an example audio signal
[x, fs] = audioread('sample-3s.wav'); % Built-in sample in MATLAB
x = x(:,1);                    % Use mono channel
x = x(1:1024);                   % Take first 1024 samples
%% 2. Fourier Transform (FFT)
X = fft(x);
N = length(x);
f = (0:N-1)*(fs/N);              % Frequency axis
% Plot FFT magnitude
figure;
subplot(3,1,1);
plot(f, abs(X));
title('Fourier Transform');
xlabel('Frequency (Hz)'); ylabel('|X(f)|'); grid on;
%% 3. Z-Transform (Pole-Zero Plot)
% Use filter representation: numerator = signal, denominator = 1
zplane(x, 1);
title('Pole-Zero Plot of Audio Signal');
%% 4. Plot time-domain signal
subplot(3,1,3);
plot((0:N-1)/fs, x);
title('Time-Domain Audio Signal');
xlabel('Time (s)'); ylabel('Amplitude'); grid on;
```

—----------

6 Develop a MATLAB program to compute and visualize the cross- correlation of two given sequences x(n) and y(n), and the autocorrelation of sequence x(n), demonstrating their properties and potential use in signal analysis applications like echo detection or signal alignment.

- Define two sequences x(n)x(n)x(n) and y(n)y(n)y(n) as example input signals.
- Compute the cross-correlation between x(n)x(n)x(n) and y(n)y(n)y(n) using xcorr.
- Calculate the autocorrelation of x(n)x(n)x(n) using xcorr with the sequence itself.
- Plot the cross-correlation and autocorrelation using stem to visualize the lag-based correlation.

- Display the results with appropriate labels and titles for both the cross-correlation and autocorrelation plots.

```
clc; clear; close all;
%% 1. Define two sequences x(n) and y(n)
x = [1, 2, 3, 4, 5];    % Example sequence x(n)
y = [5, 4, 3, 2, 1];    % Example sequence y(n)
%% 2. Compute Cross-Correlation of x(n) and y(n)
cross_corr = xcorr(x, y);  % Cross-correlation of x and y
%% 3. Compute Autocorrelation of x(n)
auto_corr = xcorr(x, x);  % Autocorrelation of x
%% 4. Visualize Results
figure;
% Plot Cross-Correlation
subplot(2,1,1);
stem(-length(x)+1:length(x)-1, cross_corr, 'filled');
title('Cross-Correlation of x(n) and y(n)');
xlabel('Lag (n)'); ylabel('Correlation'); grid on;
% Plot Autocorrelation
subplot(2,1,2);
stem(-length(x)+1:length(x)-1, auto_corr, 'filled');
title('Autocorrelation of x(n)');
xlabel('Lag (n)'); ylabel('Correlation'); grid on;
```

—----------

7 Develop a MATLAB program to implement and visualize linear convolution of discrete-time sequences, simulating a digital filter's effect on an input signal for applications in areas like audio or image processing.

- **Define the input signal** x(n) and the **impulse response** h(n) of the system.
- **Perform linear convolution** using the conv() function to get the output signal y(n).
- Create **time index vectors** for each signal (x(n), h(n), and y(n)) based on their lengths.
- Use **stem plots** to visualize each signal: input, impulse response, and output.
- **Label the axes and titles** for clarity and enable grid for better readability.

```
clc; clear; close all;
%% 1. Define input signal x(n) and impulse response h(n)
```

```
x = [1 2 3 4];        % Example input signal
h = [1 -1 2];         % Example filter (impulse response)
%% 2. Perform linear convolution
y = conv(x, h);       % Convolved output
%% 3. Plot all signals
n_x = 0:length(x)-1;
n_h = 0:length(h)-1;
n_y = 0:length(y)-1;
figure;
subplot(3,1,1);
stem(n_x, x, 'filled'); title('Input Signal x(n)');
xlabel('n'); ylabel('x(n)'); grid on;
subplot(3,1,2);
stem(n_h, h, 'filled'); title('Impulse Response h(n)');
xlabel('n'); ylabel('h(n)'); grid on;
subplot(3,1,3);
stem(n_y, y, 'filled'); title('Output y(n) = x(n) * h(n)');
xlabel('n'); ylabel('y(n)'); grid on;
```