Experiment No 04

BECS 32461

Paper C

IMPLEMENTATION OF N - POINT FFT AND IFFT ALGORITHM

Student Name: W. K. G. K. Jayawardana

Student No: EC/2021/006

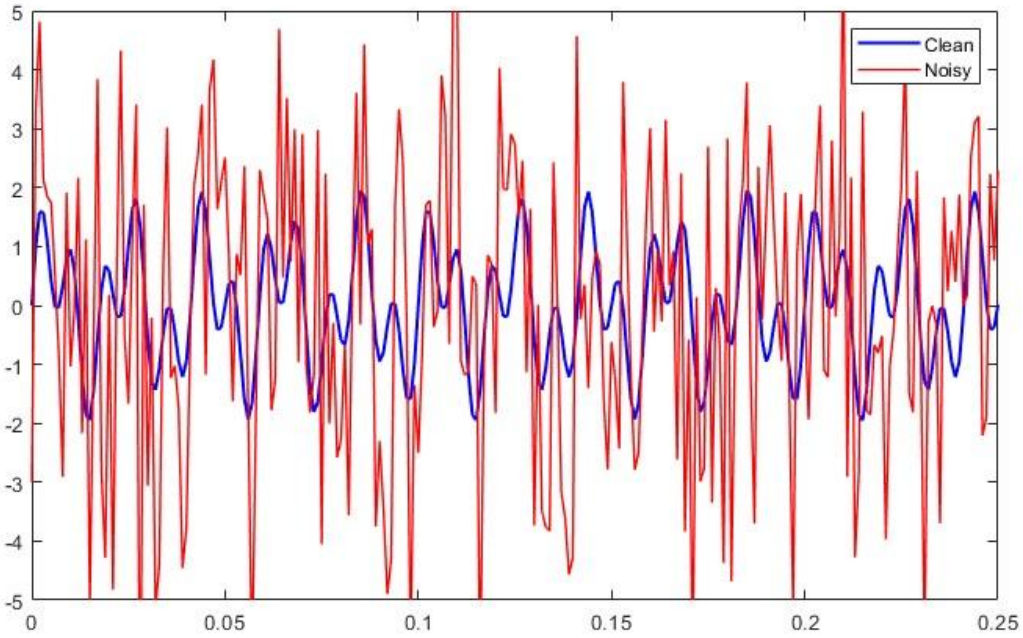
Date Performed: 2025/10/17

Date Submitted: 2025/10/17

PROCEDURE

FO1.

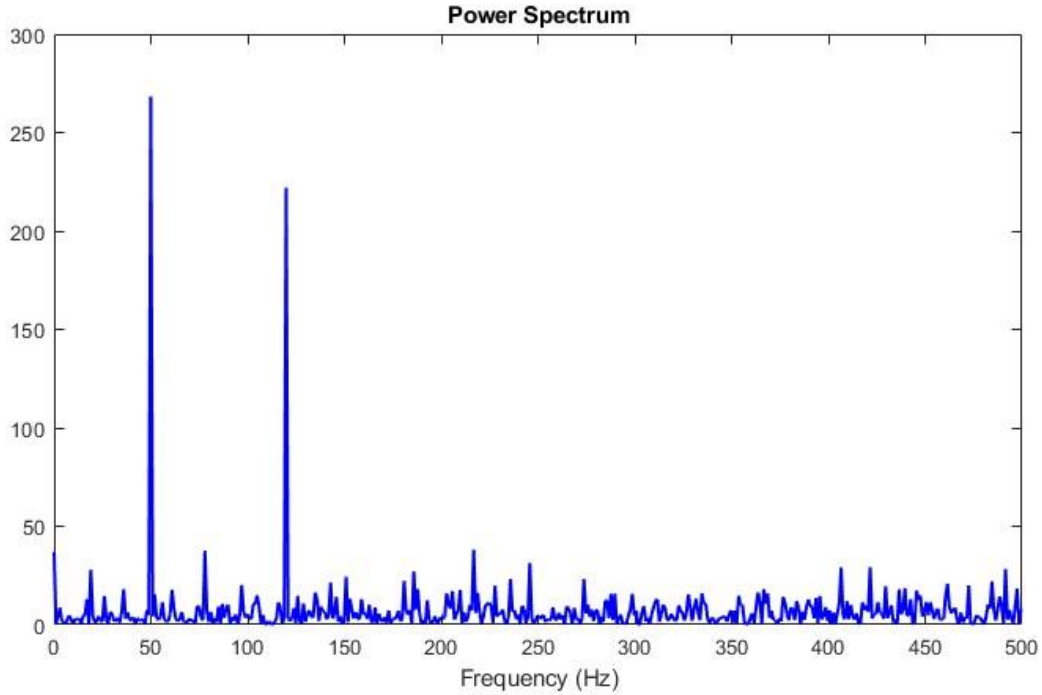
```
dt = 0.001;
t = 0:dt:1;

x_clean = sin(2*pi*50*t) + sin(2*pi*120*t);
y_noisy = x_clean + 2.5*randn(size(t));

figure;
plot(t, x_clean, 'b-', 'LineWidth', 1.5);
hold on;
plot(t, y_noisy, 'r-', 'LineWidth', 1);
legend('Clean', 'Noisy');
xlim([0, 0.25]);
ylim([-5, 5]);
```



FO2.

```
dt = 0.001;
t = 0:dt:1;

x_clean = sin(2*pi*50*t) + sin(2*pi*120*t);
y_noisy = x_clean + 2.5*randn(size(t));
N = length(t);
Y = fft(y_noisy, N);
PSD = Y .* conj(Y) / N;
freq = (0:N-1) / (dt*N);

figure;
```

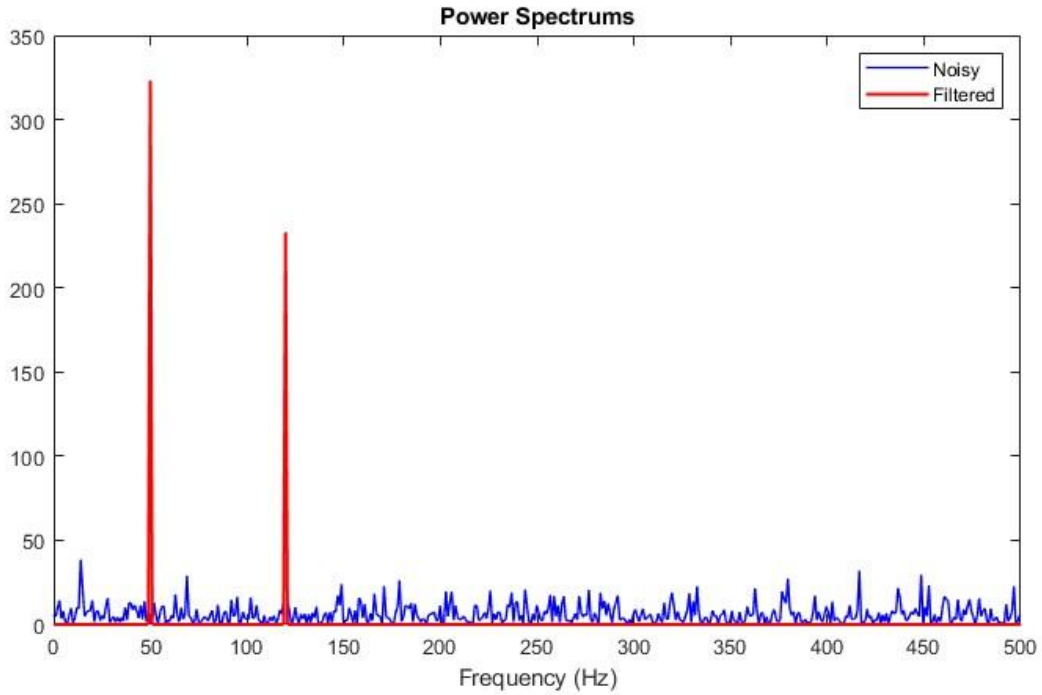
```
plot(freq, PSD, 'b-', 'LineWidth', 1.5);
xlabel('Frequency (Hz)');
title('Power Spectrum');
xlim([0, 500]);
ylim([0, 300]);
```



FO3.

```
dt = 0.001;
t = 0:dt:1;

x_clean = sin(2*pi*50*t) + sin(2*pi*120*t);
y_noisy = x_clean + 2.5*randn(size(t));

N = length(t);
Y = fft(y_noisy, N);
PSD = Y .* conj(Y) / N;
freq = (0:N-1) / (dt*N);

threshold = 60;
indices = PSD > threshold;
PSD_filtered = PSD .* indices;

figure(3);
plot(freq, PSD, 'b-', 'LineWidth', 1);
hold on;
plot(freq, PSD_filtered, 'r-', 'LineWidth', 1.5);

xlabel('Frequency (Hz)');
title('Power Spectrums');
legend('Noisy', 'Filtered');
grid on;
```

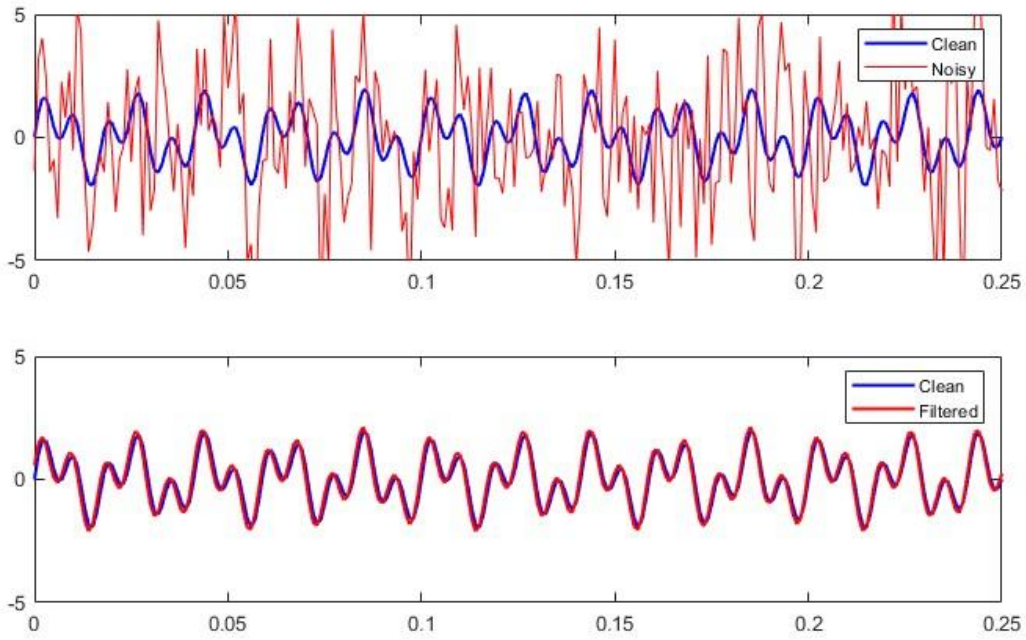
```
xlim([0, 500]);
ylim([0, 350]);
```



FO4.

```
dt = 0.001;
t = 0:dt:1;

x_clean = sin(2*pi*50*t) + sin(2*pi*120*t);
rng(42);
y_noisy = x_clean + 2.5*randn(size(t));

N = length(t);
Y = fft(y_noisy, N);
PSD = Y .* conj(Y) / N;

threshold = 60;
indices = PSD > threshold;
Y_filtered = Y .* indices;
y_filtered = real(ifft(Y_filtered));

figure;
subplot(2,1,1);
plot(t, x_clean, 'b-', 'LineWidth', 1.5);
hold on;
plot(t, y_noisy, 'r-', 'LineWidth', 0.5);
legend('Clean', 'Noisy');
xlim([0, 0.25]);
ylim([-5, 5]);

subplot(2,1,2);
plot(t, x_clean, 'b-', 'LineWidth', 1.5);
```

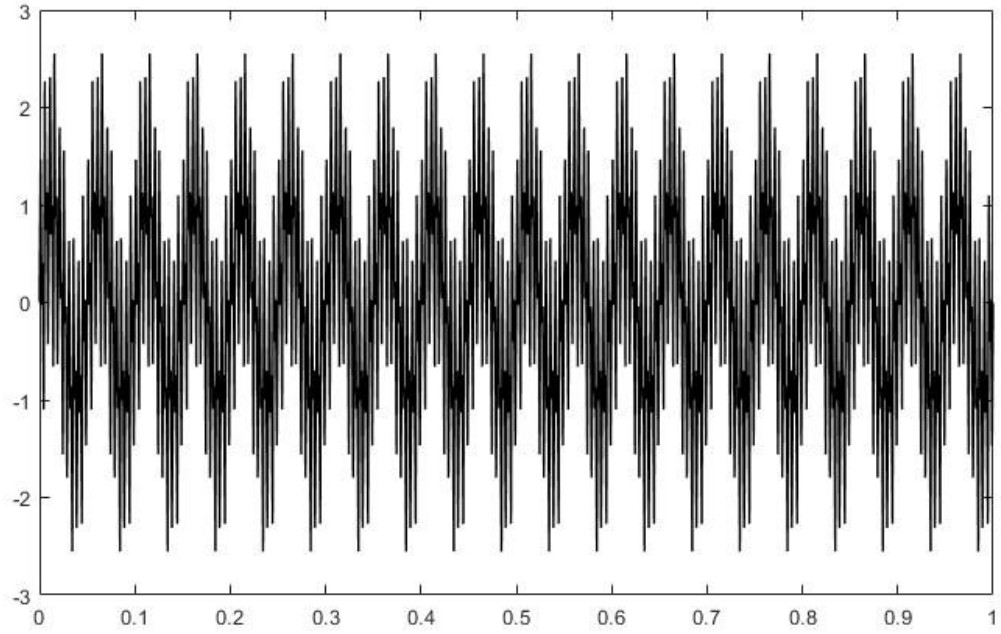
```
hold on;
plot(t, y_filtered, 'r-', 'LineWidth', 1.5);
legend('Clean', 'Filtered');
xlim([0, 0.25]);
ylim([-5, 5]);
```



EXERCISE

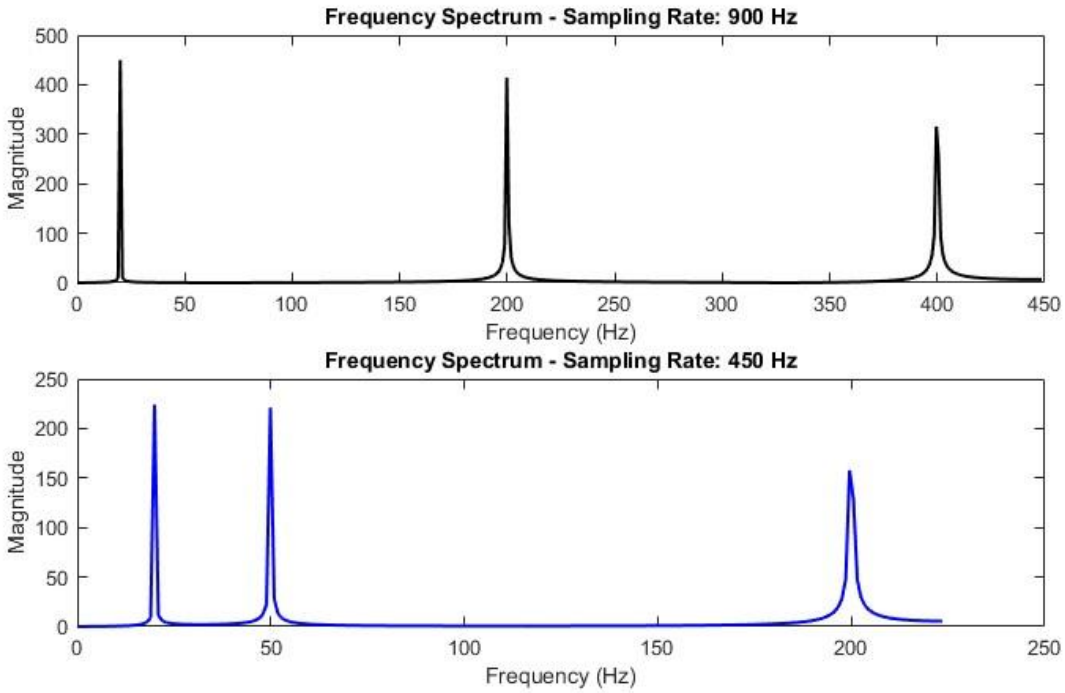
EO1.

```
f1 = 20; f2 = 200; f3 = 400;
A = 1;

fs1 = 900;
t1 = 0:1/fs1:1;
x1 = A*sin(2*pi*f1*t1) + A*sin(2*pi*f2*t1) + A*sin(2*pi*f3*t1);

figure;
plot(t1, x1, 'k-', 'LineWidth', 1);
```




```
f1 = 20; f2 = 200; f3 = 400;
A = 1;

fs1 = 900;
t1 = 0:1/fs1:1;
x1 = A*sin(2*pi*f1*t1) + A*sin(2*pi*f2*t1) + A*sin(2*pi*f3*t1);

fs2 = 450;
t2 = 0:1/fs2:1;
x2 = A*sin(2*pi*f1*t2) + A*sin(2*pi*f2*t2) + A*sin(2*pi*f3*t2);

N1 = length(x1);
N2 = length(x2);

X1 = fft(x1, N1);
freq1 = (0:N1-1) * fs1 / N1;

X2 = fft(x2, N2);
freq2 = (0:N2-1) * fs2 / N2;

figure;
subplot(2,1,1);
plot(freq1(1:N1/2), abs(X1(1:N1/2)), 'k-', 'LineWidth', 1.5);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Spectrum - Sampling Rate: 900 Hz');


subplot(2,1,2);
plot(freq2(1:N2/2), abs(X2(1:N2/2)), 'b-', 'LineWidth', 1.5);
xlabel('Frequency (Hz)');
```

```
ylabel('Magnitude');
title('Frequency Spectrum - Sampling Rate: 450 Hz');
```



Nyquist rate for highest frequency (400 Hz): 800 Hz

When sampling at 450 Hz (below Nyquist rate), aliasing occurs.

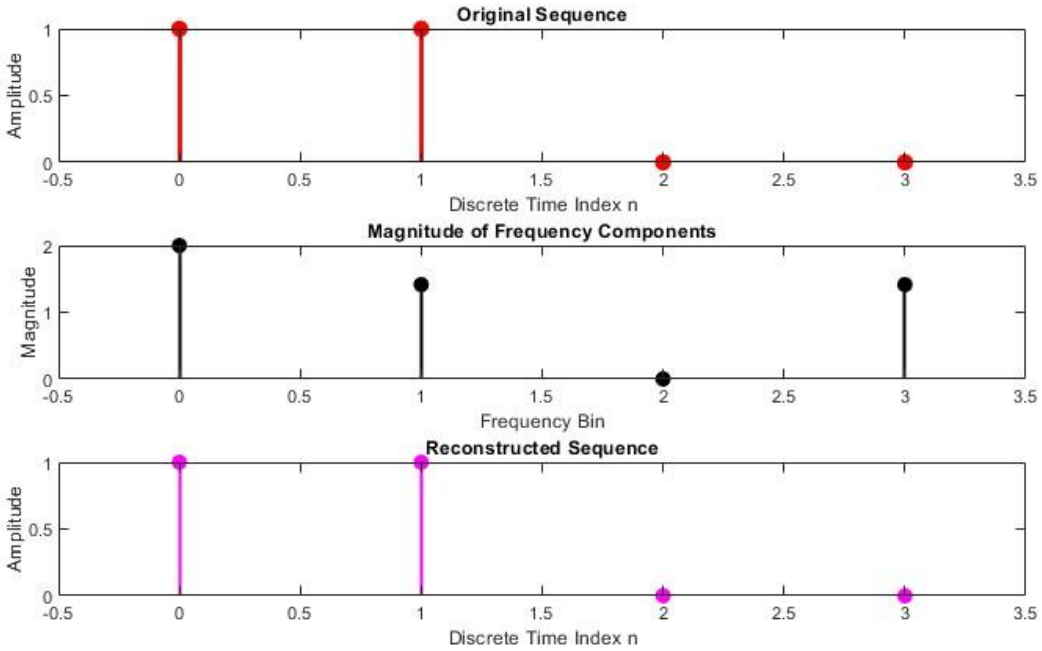
The 400 Hz component appears as a lower frequency due to folding.

General rule: Sampling rate must be at least twice the highest frequency.

E02.

```
x_original = [1, 1, 0, 0];
N = length(x_original);
X = fft(x_original);
x_reconstructed = ifft(X);

fprintf('Original sequence: [');
fprintf('%.1f ', x_original);
fprintf(']\n');

fprintf('Reconstructed sequence: [');
fprintf('%.6f ', real(x_reconstructed));
fprintf(']\n');
```

```
figure;

subplot(3,1,1);
stem(0:N-1, x_original, 'r', 'filled', 'LineWidth', 2);
title('Original Sequence');
xlabel('Discrete Time Index n');
ylabel('Amplitude');
xlim([-0.5, N-0.5]);

subplot(3,1,2);
stem(0:N-1, abs(X), 'k', 'filled', 'LineWidth', 1.5);
title('Magnitude of Frequency Components');
xlabel('Frequency Bin');
ylabel('Magnitude');
xlim([-0.5, N-0.5]);


subplot(3,1,3);
stem(0:N-1, real(x_reconstructed), 'm', 'filled', 'LineWidth', 1.5);
title('Reconstructed Sequence');
xlabel('Discrete Time Index n');
ylabel('Amplitude');
xlim([-0.5, N-0.5]);

tolerance = 1e-10;
if max(abs(real(x_reconstructed) - x_original)) < tolerance
    fprintf('The reconstructed sequence matches the original sequence\n');
else
    fprintf('The reconstructed sequence does not match the original sequence\n');
end
```

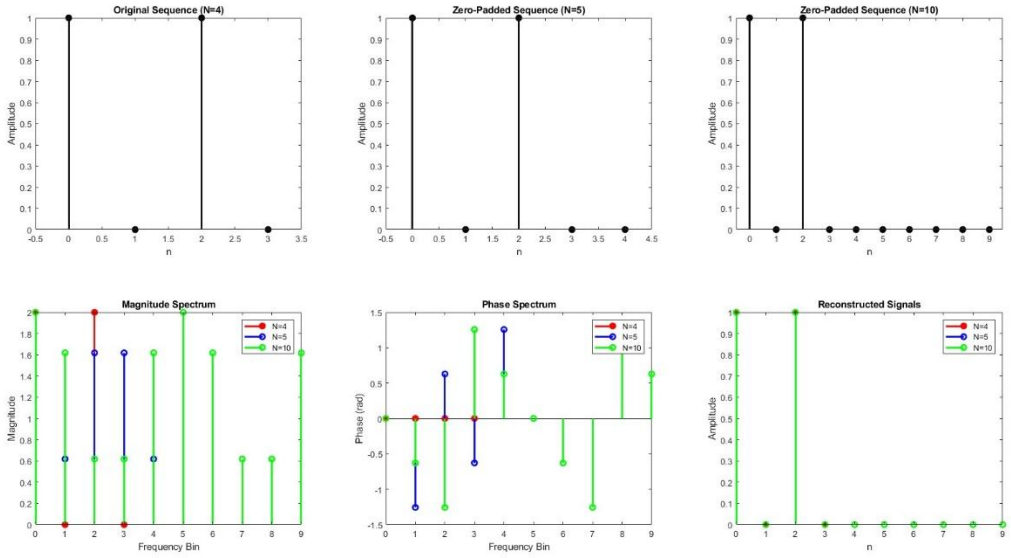


```
>> E02
Original sequence: [1.0 1.0 0.0 0.0 ]
Reconstructed sequence: [1.000000 1.000000 0.000000 0.000000 ]
The reconstructed sequence matches the original sequence
```

E03.

```
x_original = [1, 0, 1, 0];
N_original = length(x_original);

x_padded5 = [x_original, 0];
x_padded10 = [x_original, zeros(1, 6)];

X_original = fft(x_original);
X_padded5 = fft(x_padded5);
X_padded10 = fft(x_padded10);

x_recon_original = ifft(X_original);
x_recon_padded5 = ifft(X_padded5);
x_recon_padded10 = ifft(X_padded10);

figure;

subplot(2,3,1);
stem(0:N_original-1, x_original, 'k', 'filled', 'LineWidth', 2);
title('Original Sequence (N=4)');
xlabel('n'); ylabel('Amplitude');
xlim([-0.5, 3.5]);

subplot(2,3,2);
stem(0:4, x_padded5, 'k', 'filled', 'LineWidth', 2);
title('Zero-Padded Sequence (N=5)');
xlabel('n'); ylabel('Amplitude');
xlim([-0.5, 4.5]);

subplot(2,3,3);
stem(0:9, x_padded10, 'k', 'filled', 'LineWidth', 2);
title('Zero-Padded Sequence (N=10)');
xlabel('n'); ylabel('Amplitude');
xlim([-0.5, 9.5]);

subplot(2,3,4);
freq_orig = 0:N_original-1;
freq_pad5 = 0:4;
freq_pad10 = 0:9;

stem(freq_orig, abs(X_original), 'r', 'filled', 'LineWidth', 2);
hold on;
stem(freq_pad5, abs(X_padded5), 'b', 'LineWidth', 2);
stem(freq_pad10, abs(X_padded10), 'g', 'LineWidth', 2);
title('Magnitude Spectrum');
xlabel('Frequency Bin'); ylabel('Magnitude');
legend('N=4', 'N=5', 'N=10');
```

```
subplot(2,3,5);
stem(freq_orig, angle(X_original), 'r', 'filled', 'LineWidth', 2);
hold on;
stem(freq_pad5, angle(X_padded5), 'b', 'LineWidth', 2);
stem(freq_pad10, angle(X_padded10), 'g', 'LineWidth', 2);
title('Phase Spectrum');
xlabel('Frequency Bin'); ylabel('Phase (rad)');
legend('N=4', 'N=5', 'N=10');

subplot(2,3,6);
stem(0:N_original-1, real(x_recon_original), 'r', 'filled', 'LineWidth', 2);
hold on;
stem(0:4, real(x_recon_padded5), 'b', 'LineWidth', 2);
stem(0:9, real(x_recon_padded10), 'g', 'LineWidth', 2);
title('Reconstructed Signals');
xlabel('n'); ylabel('Amplitude');
legend('N=4', 'N=5', 'N=10');

error_original = max(abs(real(x_recon_original) - x_original));
error_padded5 = max(abs(real(x_recon_padded5(1:4)) - x_original));
error_padded10 = max(abs(real(x_recon_padded10(1:4)) - x_original));

fprintf('Maximum absolute differences:\n');
fprintf('Original: %.10f\n', error_original);
fprintf('Padded to 5: %.10f\n', error_padded5);
fprintf('Padded to 10: %.10f\n', error_padded10);
```




```
>> E03
Maximum absolute differences:
Original: 0.0000000000
Padded to 5: 0.0000000000
Padded to 10: 0.0000000000
>>
```

Zero-padding effects:

- Increases frequency resolution (more frequency bins)

- Does not add new frequency information

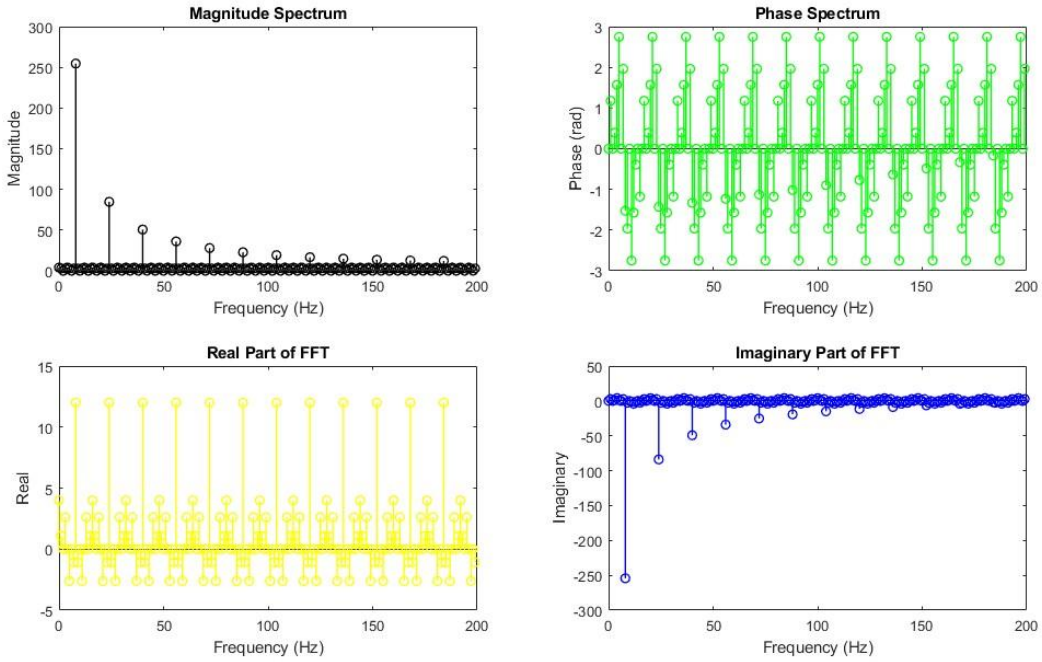
- Provides smoother frequency spectrum interpolation

- Actual frequency components remain unchanged

>>

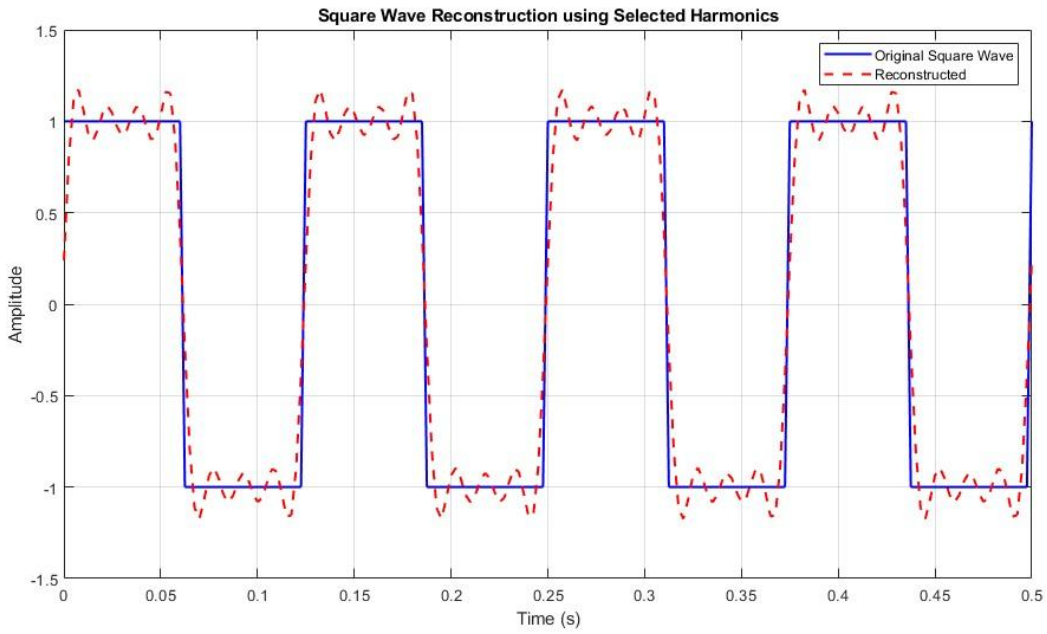
E04.

```
f0 = 8;
fs = 400;
duration = 1;

t = 0:1/fs:duration-1/fs;
N = length(t);

square_wave = square(2*pi*f0*t);

X = fft(square_wave, N);
freq = (0:N-1) * fs / N;

figure;
subplot(2,2,1);
stem(freq(1:N/2), abs(X(1:N/2)), 'k', 'LineWidth', 1);
title('Magnitude Spectrum');
xlabel('Frequency (Hz)'); ylabel('Magnitude');

subplot(2,2,2);
stem(freq(1:N/2), angle(X(1:N/2)), 'g', 'LineWidth', 1);
title('Phase Spectrum');
xlabel('Frequency (Hz)'); ylabel('Phase (rad)');

subplot(2,2,3);
stem(freq(1:N/2), real(X(1:N/2)), 'y', 'LineWidth', 1);
title('Real Part of FFT');
xlabel('Frequency (Hz)'); ylabel('Real');

subplot(2,2,4);
stem(freq(1:N/2), imag(X(1:N/2)), 'b', 'LineWidth', 1);
title('Imaginary Part of FFT');
xlabel('Frequency (Hz)'); ylabel('Imaginary');
```




```
f0 = 8;
fs = 400;
duration = 1;

t = 0:1/fs:duration-1/fs;
N = length(t);

square_wave = square(2*pi*f0*t);

X = fft(square_wave, N);
freq = (0:N-1) * fs / N;

X_filtered = zeros(1, N);
harmonic_indices = [1, 3, 5, 7]; % Harmonics to keep

for k = harmonic_indices
    freq_component = k * f0;
    idx = round(freq_component * N / fs) + 1;
    if idx <= N/2
        X_filtered(idx) = X(idx);
        X_filtered(N - idx + 2) = X(N - idx + 2); % Symmetric component
    end
end

square_reconstructed = real(ifft(X_filtered));

figure;
plot(t, square_wave, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Original Square Wave');
hold on;
plot(t, square_reconstructed, 'r--', 'LineWidth', 1.5, 'DisplayName',
'Reconstructed');
```

```
xlabel('Time (s)');
ylabel('Amplitude');
title('Square Wave Reconstruction using Selected Harmonics');
legend('show');
grid on;
xlim([0, 0.5]);

mse = mean((square_wave - square_reconstructed).^2);
fprintf('Mean Squared Error: %.6f\n', mse);
```



Effect of harmonics on reconstruction:

- More harmonics: Better approximation, sharper edges

- Fewer harmonics: Smoother reconstruction, rounded edges

- Gibbs phenomenon: Overshoot at discontinuities persists

- Square wave requires infinite harmonics for perfect reconstruction