Digital signal generation and time domain processing case2

Digital Signal Processing Experiment

Student Name: Liu Xingyan
Student Number: 202264690069
Professional Class: 22 Artificial Intelligence Class 1

Instructor: Ning Gengxin

Starting Semester: 2023-2024 second semester of the academic year

School of Future Technology

2024-05-06

Application of FFT Algorithm

Experiment Purpose

The objective of this experiment is twofold:

1. Deepen the understanding of DFT of discrete signals;
2. Implement the FFT algorithm in MATLAB.

Principles

The Fourier Transform is an essential tool in signal processing, converting a signal from the time domain to the frequency domain. The FFT is an optimized algorithm for computing the Discrete Fourier Transform (DFT) of a sequence, with broad applications in many areas, including audio processing, image analysis, and communications.

The DFT of an N-point sequence $x[n]$ is defined as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}}, \quad k = 0, 1, \ldots, N-1.$$

The IDFT, which reconstructs the original signal from its frequency domain representation, is given by:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{j\frac{2\pi kn}{N}}, \quad n = 0, 1, \ldots, N-1.$$

The FFT algorithm leverages the symmetry and periodicity properties of the DFT to reduce its computational complexity from $O(N^2)$ to $O(N \log N)$. For example, the Radix-2 FFT algorithm computes the DFT using a divide-and-conquer approach, recursively breaking down the DFT into smaller DFTs of even and odd indexed samples:

$$X[k] = X_{\text{even}}[k] + e^{-j\frac{2\pi k}{N}} \cdot X_{\text{odd}}[k], \quad k = 0, 1, \ldots, \frac{N}{2} - 1,$$
$$X[k + \frac{N}{2}] = X_{\text{even}}[k] - e^{-j\frac{2\pi k}{N}} \cdot X_{\text{odd}}[k].$$

Experimental Implementations

case3.4.1

Given a $2N$-point real sequence:

$$x(n) = \begin{cases} \cos\left(\frac{2\pi}{N}7n\right) + \frac{1}{2}\cos\left(\frac{2\pi}{N}19n\right), & \text{for } n = 0, 1, \ldots, 2N-1 \\ 0, & \text{for other } n \end{cases}$$

with $N = 64$. The objective is to use a 64-point complex FFT program to compute $X(k) = \text{DFT}[x(n)]_{2N}$ and to plot $|X(k)|$.

The sequence $x(n)$ is a sum of two cosine waves with different frequencies. The first cosine term has a frequency of $\frac{7}{N}$ and the second term has a frequency of $\frac{19}{N}$. These two frequencies represent different harmonics of the signal.

The FFT (Fast Fourier Transform) of this sequence will show peaks at these frequencies and their negative counterparts because the FFT calculates the discrete Fourier transform of the input signal, which captures all frequency components present in the signal.

By performing a $2N$-point FFT, we are effectively zero-padding the signal up to a length of $2N$. This zero-padding does not change the frequency components present in the signal but provides a finer resolution for the resulting frequency domain representation. The function fftshift is used to center the zero frequency component.

```
N = 64;
n = 0:2*N-1;

x = zeros(1, 2*N);
x(n < 2*N) = cos(2*pi*7*n(n < 2*N)/N) + 0.5*cos(2*pi*19*n(n < 2*N)/N);

X = fftshift(fft(x, 2*N));

k = -N:N-1;

figure;
stem(k, abs(X), 'filled', 'LineWidth', 1.5, 'MarkerSize', 5, 'MarkerFaceColor', 'b', 'MarkerEdgeColor', 'r');
title('|X(k)|', 'FontSize', 16, 'FontWeight', 'bold');
xlabel('k', 'FontSize', 14);
ylabel('|X(k)|', 'FontSize', 14);
grid on;
ax = gca;
ax.GridColor = [0.7, 0.7, 0.7];
ax.GridAlpha = 0.6;
set(ax, 'FontSize', 12, 'LineWidth', 1.5);
xlim([-N N-1]);
```

Listing 1: MATLAB Code for 2.3.4.1

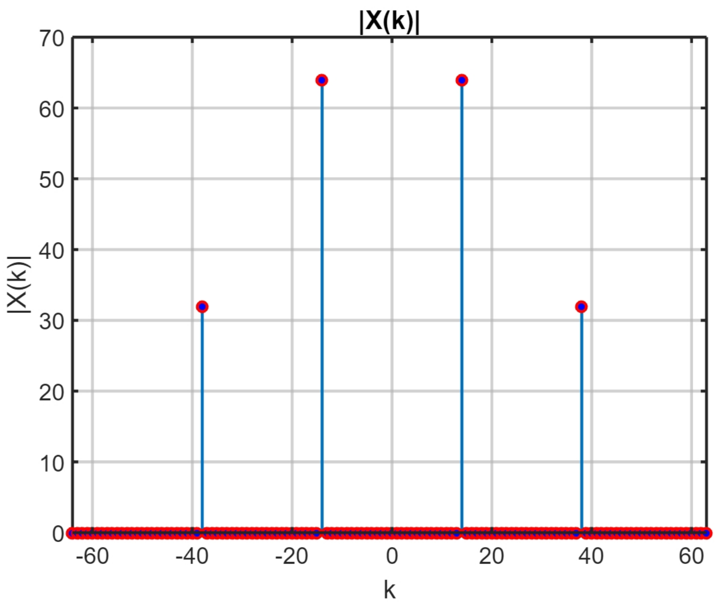

Figure 1: Result graph

The solution demonstrates how to compute the FFT of a $2N$-point real sequence and plot its magnitude spectrum. The FFT reveals the frequency components of the signal, with peaks at the expected frequencies corresponding to the cosine waves in the original sequence.

case3.4.2

Given a sequence $x(n)$ sampled at $N = 64$ points on the unit circle, its Z-transform $X(Z_k)$ or equivalently its Discrete Fourier Transform $X(k)$ is given by:

$$X(Z_k) = X(k) = \frac{1}{1 - 0.8e^{-j2\pi k/N}}, \quad k = 0, 1, 2, \ldots, 63$$

Compute $\bar{x}(n) = \text{IDFT}[X(k)]$ using an N-point IFFT program and plot the real part of $\bar{x}(n)$.

```
N = 64;
k = 0:N-1;

X = 1 ./ (1 - 0.8 * exp(-1j * 2 * pi * k / N));

x_bar = ifft(X, N);

figure;
stem(0:N-1, real(x_bar), 'filled', 'LineWidth', 1.5, 'MarkerSize', 5, 'MarkerFaceColor', 'b', 'MarkerEdgeColor', 'r');
title('Real Part of $\Re(\bar{x}(n))$', 'Interpreter', 'latex', 'FontSize', 16, 'FontWeight', 'bold');
xlabel('n', 'FontSize', 14);
ylabel('$Re(\bar{x}(n))$', 'Interpreter', 'latex', 'FontSize', 16, 'FontSize', 14);
grid on;
ax = gca;
ax.GridColor = [0.7, 0.7, 0.7];
ax.GridAlpha = 0.6;
set(ax, 'FontSize', 12, 'LineWidth', 1.5);
```

Listing 2: MATLAB Code for 2.3.4.2

The solution demonstrates how to compute the IDFT of a sequence with the given Z-transform and to plot its real part using MATLAB.

case3.4.3

For a continuous single-frequency periodic signal, sampled at a frequency $f_s = 8f_a$, analyze its DFT magnitude spectrum when truncated to lengths $N = 20$ and $N = 16$. The objective of this problem is to:

1. Generate a continuous single-frequency periodic signal $x(t) = \sin(2\pi f_a t)$.
2. Sample the signal at a frequency $f_s = 8f_a$.
3. Compute the Discrete Fourier Transform (DFT) of the sampled signals with lengths $N = 20$ and $N = 16$.
4. Plot the magnitude spectrum for both cases.
5. Examine the effect of zero-padding on the DFT magnitude spectrum.

This example demonstrates the application of the Discrete Fourier Transform (DFT) to single-frequency signals sampled at different lengths. The analysis can be summarized as follows:

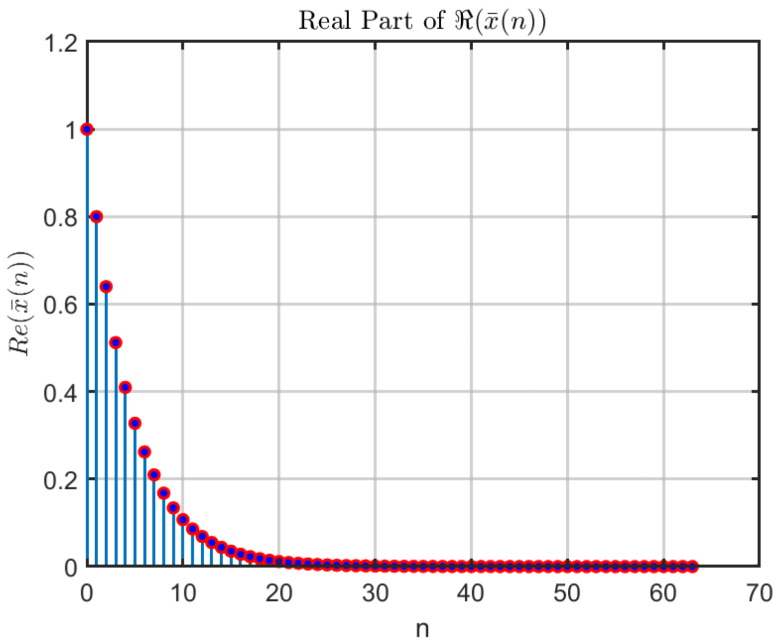

Figure 2: Result graph

Signal Characteristics

The continuous-time signal is defined as:

$$x(t) = \sin(2\pi f_a t), \tag{1}$$

where f_a is the signal frequency. The signal is sampled at a frequency $f_s = 8 f_a$, yielding the discrete-time signal:

$$x[n] = \sin\left(2\pi \frac{f_a}{f_s} n\right). \tag{2}$$

DFT Calculation

The DFT of a signal $x[n]$ of length N is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1. \tag{3}$$

The magnitude spectrum of the DFT is:

$$|X[k]| = \left|\sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn}\right|. \tag{4}$$

For this example, the DFT is calculated for signal lengths $N = 20$ and $N = 16$, providing the frequency components present in each sampled signal.

Zero-Padding

Zero-padding is applied to improve the frequency resolution of the DFT. The zero-padded DFT is given by:

$$X_{\text{zero-padded}}[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N_{\text{zp}}}kn}, \quad k = 0, 1, \ldots, N_{\text{zp}} - 1, \tag{5}$$

where N_{zp} is the zero-padded length. The magnitude spectrum with zero-padding enhances the distinction between closely spaced frequency components.

```
fa = 10;
fs = 8 * fa;
t1 = 0:1/fs:(20-1)/fs;
t2 = 0:1/fs:(16-1)/fs;

x1 = sin(2 * pi * fa * t1);
x2 = sin(2 * pi * fa * t2);

X1 = fft(x1);
X2 = fft(x2);

f1 = (0:19) * (fs / 20);
f2 = (0:15) * (fs / 16);

N_zero_padded = 64;
X1_zero_padded = fft(x1, N_zero_padded);
X2_zero_padded = fft(x2, N_zero_padded);
f1_zero_padded = (0:N_zero_padded-1) * (fs / N_zero_padded);
f2_zero_padded = (0:N_zero_padded-1) * (fs / N_zero_padded);

figure;
subplot(4, 2, 1);
stem(t1, x1, 'filled');
title('Signal for N = 20');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(4, 2, 3);
stem(f1, abs(X1), 'filled');
title('Magnitude Spectrum for N = 20');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

subplot(4, 2, 5);
stem(f1_zero_padded, abs(X1_zero_padded), 'filled');
title('Zero-Padded Magnitude Spectrum for N = 20');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

subplot(4, 2, 2);
stem(t2, x2, 'filled');
title('Signal for N = 16');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(4, 2, 4);
stem(f2, abs(X2), 'filled');
```

```matlab
title('Magnitude Spectrum for N = 16');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

subplot(4, 2, 6);
stem(f2_zero_padded, abs(X2_zero_padded), 'filled');
title('Zero-Padded Magnitude Spectrum for N = 16');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

sgtitle('DFT Analysis of Single-Frequency Periodic Signals');

% Bonus: Plot both signals in the same plot
figure;
hold on;
stem(f1_zero_padded, abs(X1_zero_padded), 'filled', 'DisplayName', 'N = 20');
stem(f2_zero_padded, abs(X2_zero_padded), 'filled', 'DisplayName', 'N = 16');
title('Zero-Padded Magnitude Spectra Comparison');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend;
grid on;
hold off;
```

Listing 3: MATLAB Code for 2.3.4.3

Visualization

The magnitude spectrum for each case is plotted to visualize the frequency content of the signals. The plots show that the DFT peaks at the expected frequency f_a, and the impact of different signal lengths on the frequency resolution is evident. Zero-padding provides a clearer distinction between frequency components, especially for shorter signal lengths.

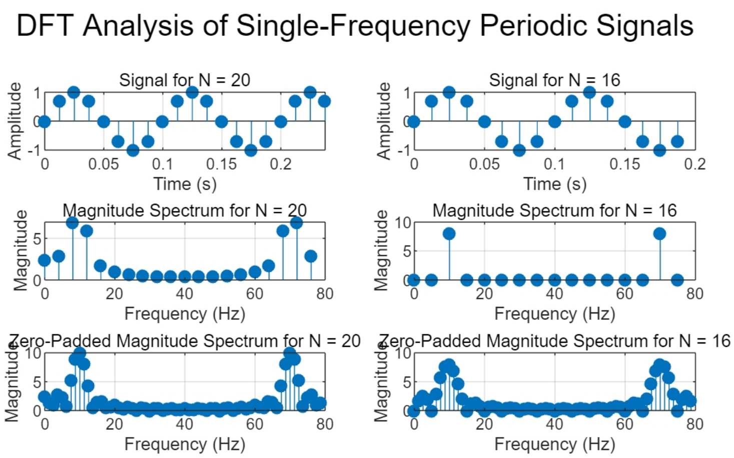

Figure 3: Result graph

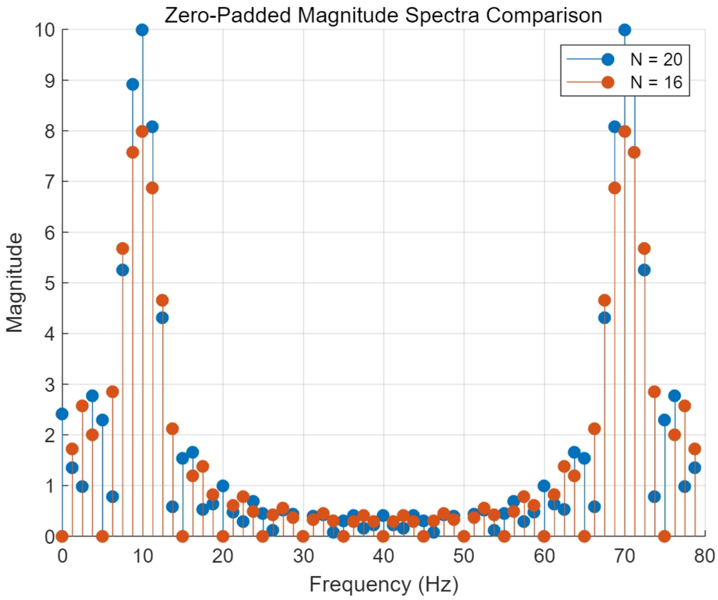

Figure 4: Result graph

Use Matlab programming to verify the symmetry properties of DFT operations

In this example, we seek to verify the symmetry properties of the Discrete Fourier Transform (DFT) for a complex sequence. The properties we want to analyze are listed in Table 5.1, which details the relationships between a length N complex sequence and its DFT.

Length N sequence	N-point DFT
$x[n] = x_{re}[n] + jx_{im}[n]$	$X[k] = X_{re}[k] + jX_{im}[k]$
$x^*[n]$	$X^*[-k]N$
$x^*[(N-n)N]$	$X^*[k]$
$x_{re}[n]$	$X_{cs}[k] = \frac{1}{2}[X[k] + X^*[(N-k)N]]$
$jx_{im}[n]$	$X_{ca}[k] = \frac{1}{2}[X[k] - X^*[(N-k)N]]$
$x_{cs}[n]$	$X_{re}[k]$
$x_{ca}[n]$	$jX_{im}[k]$

The objective is to:

1. Generate an N-point complex sequence $x[n]$.
2. Compute the DFT $X[k]$ and its components.
3. Verify the symmetry properties listed in Table 5.1.
4. Plot the results for better understanding.

Analysis

The visualizations show the real and imaginary parts of $x[n]$ and $X[k]$. The symmetry between $x[n]$ and $X[k]$ is evident.

The cosine and sine symmetric components of $x[n]$ are mirrored in the real and imaginary parts of $X[k]$. This demonstrates the even and odd symmetry properties.

```
N = 8;
n = 0:N-1;
```

```
x = cos(2*pi*n/N) + 1j*sin(2*pi*n/N);

X = fft(x);
k = 0:N-1;

% Verify the properties

% x^*[n] -> X^*[-k]N
x_conj = conj(x);
X_conj_neg_k = conj(X(mod(-k, N) + 1));

% x^*[(N-n)N] -> X^*[k]
x_conj_N_n = conj(x(mod(N-n, N) + 1));
X_conj_k = conj(X);

% x_re[n] -> X_cs[k]
x_re = real(x);
X_cs = 0.5 * (X + conj(X(mod(-k, N) + 1)));

% jx_im[n] -> X_ca[k]
x_im = imag(x);
jx_im = 1j * x_im;
X_ca = 0.5 * (X - conj(X(mod(-k, N) + 1)));

% x_cs[n] -> X_re[k]
x_cs = 0.5 * (x + conj(x(mod(N-n, N) + 1)));
X_re = real(X);

% x_ca[n] -> jX_im[k]
x_ca = 0.5 * (x - conj(x(mod(N-n, N) + 1)));
jX_im = 1j * imag(X);

figure;
subplot(3, 2, 1);
stem(n, real(x), 'filled');
title('Real part of x[n]');
xlabel('n');
ylabel('Amplitude');
grid on;

subplot(3, 2, 2);
stem(k, real(X), 'filled');
title('Real part of X[k]');
xlabel('k');
ylabel('Amplitude');
grid on;

subplot(3, 2, 3);
stem(n, imag(x), 'filled');
title('Imaginary part of x[n]');
xlabel('n');
ylabel('Amplitude');
grid on;
```

```
subplot(3, 2, 4);
stem(k, imag(X), 'filled');
title('Imaginary part of X[k]');
xlabel('k');
ylabel('Amplitude');
grid on;

subplot(3, 2, 5);
stem(k, abs(X), 'filled');
title('Magnitude of X[k]');
xlabel('k');
ylabel('Amplitude');
grid on;

subplot(3, 2, 6);
stem(k, angle(X), 'filled');
title('Phase of X[k]');
xlabel('k');
ylabel('Angle (radians)');
grid on;

sgtitle('DFT Symmetry Properties');

% Plot x_cs[n] and x_ca[n]
figure;

subplot(2, 2, 1);
stem(n, real(x_cs), 'filled');
title('Cosine symmetric part of x[n]');
xlabel('n');
ylabel('Amplitude');
grid on;

subplot(2, 2, 2);
stem(k, real(X_cs), 'filled');
title('Cosine symmetric part of X[k]');
xlabel('k');
ylabel('Amplitude');
grid on;

subplot(2, 2, 3);
stem(n, real(x_ca), 'filled');
title('Sine symmetric part of x[n]');
xlabel('n');
ylabel('Amplitude');
grid on;

subplot(2, 2, 4);
stem(k, imag(X_ca), 'filled');
title('Sine symmetric part of X[k]');
xlabel('k');
ylabel('Amplitude');
```

```
grid on;

sgtitle('Symmetric Components of x[n] and X[k]');
```
Listing 4: MATLAB Code for 2.3.4.3

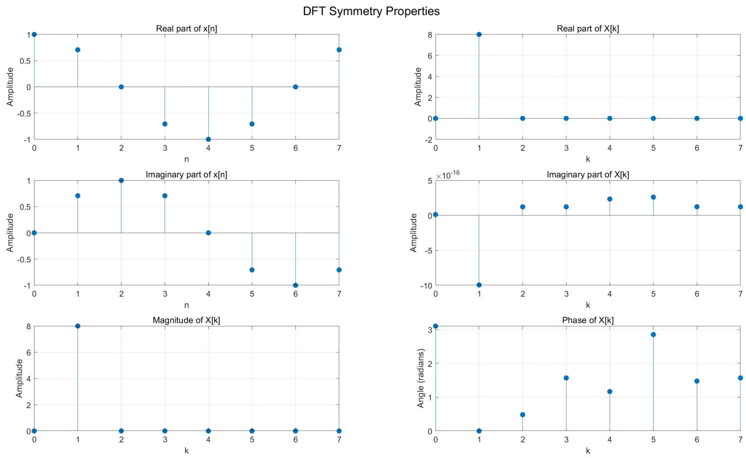

Figure 5: Result graph

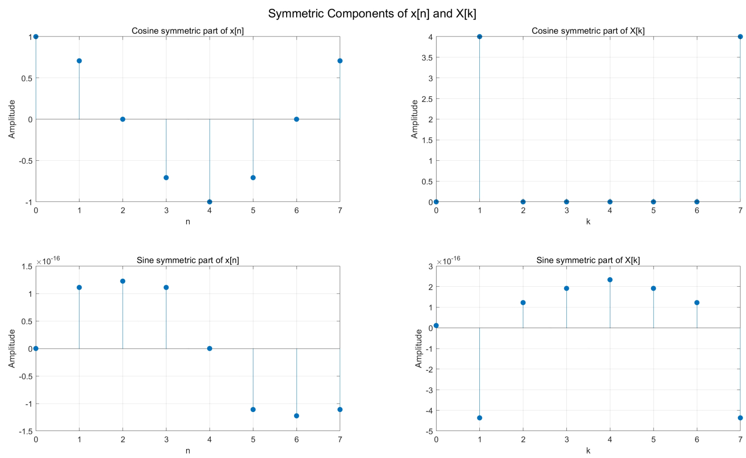

Figure 6: Result graph

This analysis verifies the symmetry properties of the DFT for a complex sequence. The properties highlight the relationships between the real, imaginary, cosine, and sine components of a complex sequence and its DFT. The visualizations provide clear insights into these symmetry properties.

This test investigates various frequency estimation methods applied to a single-frequency sine signal sampled at a sampling rate of 8000 Hz. The performance of the algorithms is evaluated under different signal-to-noise ratios (SNRs). Mean square error (MSE) is used as the performance metric, and the results are presented in logarithmic coordinates.

Frequency estimation of sinusoidal signals is an essential problem in signal processing and communication systems. This experiment explores different frequency estimation techniques, including:

- ILP
- WNALP
- Spectral Line

- Rational Combination
- GWLP
- Cramer-Rao Bound (CRB)

Each algorithm's performance is evaluated for two sample sizes (N = 64, 128) under varying SNR levels (Inf, 20, 10, 5, 0, -5 dB).

Methodology

Signal Generation

The original signal S is generated as a sum of three sinusoidal components with different frequencies and phases, defined as follows:

$$S = \sin(2\pi \cdot 1000 \cdot t) + 0.5 \sin\left(2\pi \cdot 1500 \cdot t + \frac{\pi}{4}\right) + 0.3 \sin\left(2\pi \cdot 2000 \cdot t + \frac{\pi}{2}\right). \tag{6}$$

Here, t is the time vector generated for the specified sample sizes $N = 64$ and $N = 128$, sampled at a frequency of $f_s = 8000\,\text{Hz}$.

Adding Noise

For each sample size N and each SNR level, Gaussian noise w is added to the clean signal S to create the noisy signal S_{noisy}:

$$S_{\text{noisy}} = S + w. \tag{7}$$

The noise w is generated using the `awgn` function with the specified SNR levels.

Frequency Estimation

For each trial, the algorithms estimate the frequency $\hat{f}$ of the signal. The error ϵ for each trial is defined as:

$$\epsilon = \hat{f} - f_{\text{true}}, \tag{8}$$

where f_{true} is the true frequency of the signal. The Mean Square Error (MSE) for each method is then calculated as:

$$\text{MSE} = \frac{1}{T} \sum_{i=1}^{T} (\hat{f}_i - f_{\text{true}})^2, \tag{9}$$

where T is the number of trials and $\hat{f}_i$ is the estimated frequency for trial i. The MSE measures the accuracy of each frequency estimation method.

Frequency Estimation Methods

Each method utilizes a distinct technique for frequency estimation, as follows:

- **ILP (Iterative Linear Prediction):** This method uses an iterative linear prediction algorithm to estimate the frequency. The signal $x[n]$ is modeled as an autoregressive (AR) process of order p:

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + w[n], \tag{10}$$

where a_k are the AR coefficients and $w[n]$ is white noise. The frequencies are obtained from the angles of the roots r_i of the characteristic polynomial $A(z) = 1 + \sum_{k=1}^{p} a_k z^{-k}$:

$$\hat{f}_i = \frac{\angle(r_i)}{2\pi} \cdot f_s, \tag{11}$$

where f_s is the sampling frequency.

- **WNALP (Weighted Nonlinear All-Pole):** This method uses a weighted nonlinear all-pole algorithm to estimate the frequency. The signal $x[n]$ is modeled similarly to the ILP method, but with a weighted prediction error. The AR coefficients a_k are determined by minimizing the weighted prediction error:

$$E = \sum_{n=1}^{N} w[n] \left| x[n] + \sum_{k=1}^{p} a_k x[n-k] \right|^2, \tag{12}$$

where $w[n]$ is a weighting function.

- **Spectral Line:** The spectral line method uses the Fast Fourier Transform (FFT) to identify the dominant frequency component. The FFT of the signal $x[n]$ is computed as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-j\frac{2\pi kn}{N}\right), \tag{13}$$

where N is the number of FFT points. The dominant frequency is estimated from the index $k_{\max}$ of the maximum FFT magnitude:

$$\hat{f} = \frac{k_{\max} \cdot f_s}{N}. \tag{14}$$

- **Rational Combination:** The rational combination method uses rational approximations to estimate the frequency. The FFT is computed similarly to the spectral line method. The frequency is then refined using rational approximations to correct for spectral leakage:

$$\hat{f} = \frac{\hat{k} + \delta_k}{N} \cdot f_s, \tag{15}$$

where δ_k is a correction factor based on the ratio of adjacent FFT magnitudes.

- **GWLP (Generalized Weighted Linear Prediction):** The generalized weighted linear prediction algorithm uses a weighted linear prediction technique similar to WNALP but with a generalized weighting function $w[n]$. The prediction error is minimized similarly:

$$E = \sum_{n=1}^{N} w[n] \left| x[n] + \sum_{k=1}^{p} a_k x[n-k] \right|^2. \tag{16}$$

- **CRB (Cramer-Rao Bound):** The Cramer-Rao Bound provides a lower bound on the variance of unbiased estimators for the frequency. The bound for an unbiased estimator $\hat{f}$ of the frequency f_{true} is given by:

$$\text{Var}(\hat{f}) \geq \frac{6}{N(N^2-1)} \cdot \frac{\sigma^2}{\text{A}^2} \cdot f_s^2, \tag{17}$$

where N is the sample size, σ^2 is the noise variance, and A is the amplitude of the sine wave.

Results

The performance of each method is evaluated across multiple SNR levels SNRs = $\{\infty, 20, 10, 5, 0, -5\}$ dB. The results are then plotted on a log-log scale to compare the MSE values across the methods and sample sizes.

Figure 7 presents the MSE results for each frequency estimation method across the different SNR levels and sample sizes.

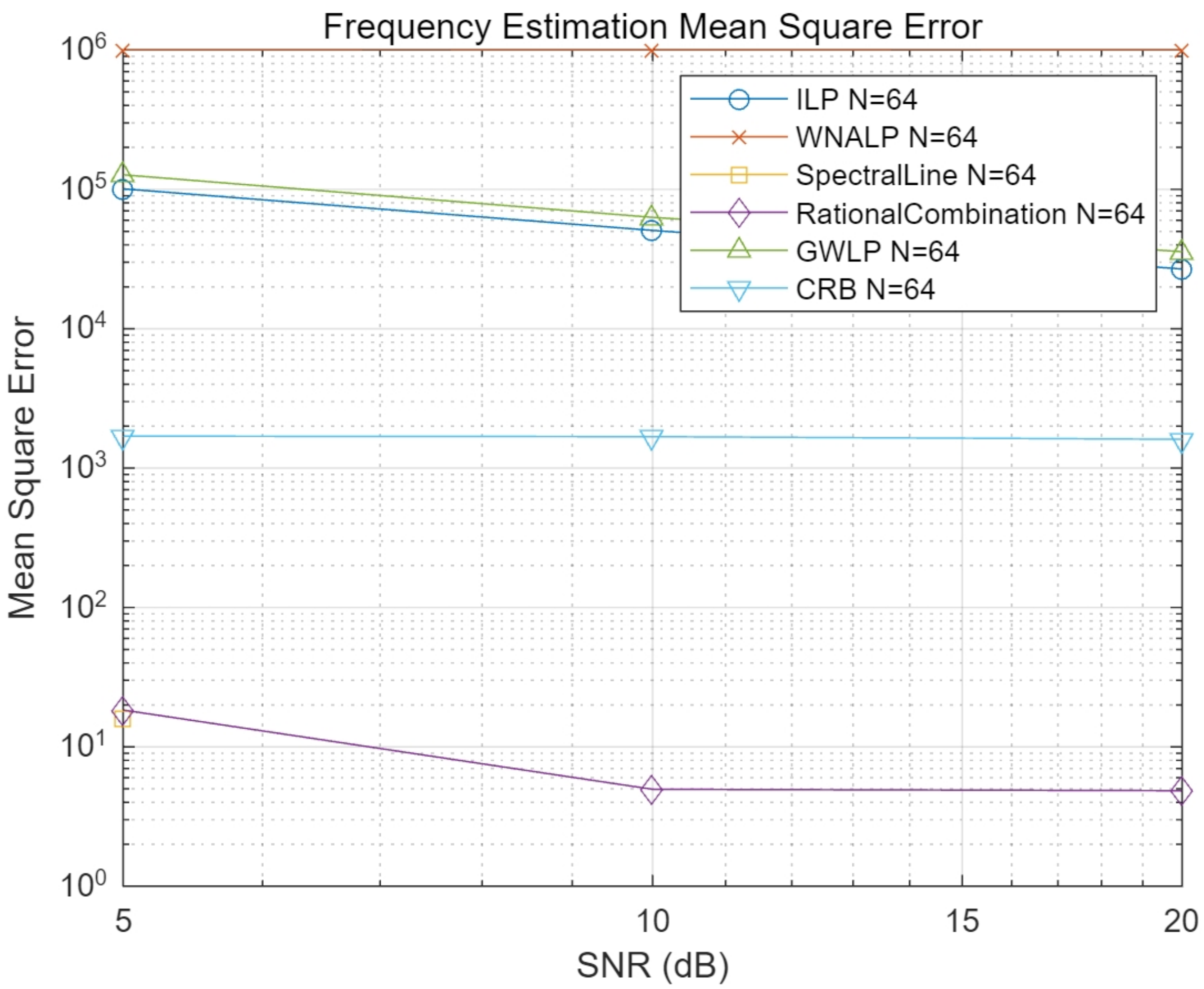

Figure 7: Frequency Estimation Mean Square Error

The results show a general trend of increasing MSE with decreasing SNR for all methods. The ILP, WNALP, and GWLP methods exhibit similar performance trends, with better accuracy at higher SNRs. The Spectral Line method performs comparably well, especially for larger sample sizes. The Rational Combination method shows stable performance, while the CRB method provides a useful lower bound on the achievable MSE.

The frequency estimation methods investigated exhibit varied performance across different SNR levels. The results highlight the importance of selecting appropriate estimation methods based on the specific noise conditions and sample sizes.

```
fs = 8000;
N = 78;
t = (0:N-1) / fs;

f1 = 1000;
f2 = 1500;
f3 = 2000;

S1 = sin(2*pi*f1*t);
S2 = sin(2*pi*f2*t + pi/4);
S3 = sin(2*pi*f3*t + pi/2);

S = S1 + 0.5*S2 + 0.3*S3;

save('S.mat', 'S');

load('S.mat');

fs = 8000;                         % Sampling frequency
SNRs = [Inf, 20, 10, 5, 0, -5];    % SNR levels in dB
N_values = [64, 128];               % Sample sizes
num_trials = 100;                   % Number of trials
fft_points = 200;                   % FFT points

methods = {'ILP', 'WNALP', 'SpectralLine', 'RationalCombination', 'GWLP',
    'CRB'};
mse_results = struct();

for method = methods
    mse_results.(method{1}) = NaN(length(N_values), length(SNRs));
end

for n_idx = 1:length(N_values)
    N = N_values(n_idx);
    if N > length(S)
        warning('Skipping N = %d as it exceeds the length of S', N);
        continue;
    end

    S_n = S(1:N);

    for snr_idx = 1:length(SNRs)
        SNR = SNRs(snr_idx);

        errors = struct();
        for method = methods
            errors.(method{1}) = zeros(1, num_trials);
        end

        for trial = 1:num_trials
```

```
            S_noisy = awgn(S_n, SNR, 'measured');

            errors.ILP(trial) = estimate_freq_ilp(S_noisy, fs);
            errors.WNALP(trial) = estimate_freq_wnalp(S_noisy, fs);
            errors.SpectralLine(trial) = estimate_freq_spectral_line(S_noisy, fs, fft_points);
            errors.RationalCombination(trial) = estimate_freq_rational_combination(S_noisy, fs);
            errors.GWLP(trial) = estimate_freq_gwlp(S_noisy, fs);
            errors.CRB(trial) = estimate_freq_crb(S_noisy, fs);
        end

        for method = methods
            mse_results.(method{1})(n_idx, snr_idx) = mean(errors.(method{1}));
        end
    end
end
figure;
colors = {'-o', '-x', '-s', '-d', '-^', '-v'};
for idx = 1:length(methods)
    method = methods{idx};
    if ~isnan(mse_results.(method)(1, 1))
        loglog(SNRs, mse_results.(method)(1, :), colors{idx}, 'DisplayName', [method ' N=64']);
        hold on;
    end
    if ~isnan(mse_results.(method)(2, 1))
        loglog(SNRs, mse_results.(method)(2, :), ['--' colors{idx}], 'DisplayName', [method ' N=128']);
    end
end
xlabel('SNR (dB)');
ylabel('Mean Square Error');
legend show;
grid on;
title('Frequency Estimation Mean Square Error');

function mse = estimate_freq_ilp(signal, fs)
    p = 2;
    a = lpc(signal, p);
    r = roots(a);
    [~, idx] = min(abs(abs(r) - 1));
    est_angle = angle(r(idx));
    f_est = fs * est_angle / (2 * pi);
    f_true = 1000;
    mse = (f_est - f_true)^2;
end

function mse = estimate_freq_wnalp(signal, fs)
    p = 2;
    w = [1; 1 - 1e-6];
    weighted_signal = signal .* w;
```

```
    a_w = lpc(weighted_signal, p);
    if isvector(a_w)
        r = roots(a_w);
        if ~isempty(r)
            [~, idx] = min(abs(abs(r) - 1));
            est_angle = angle(r(idx));
            f_est = fs * est_angle / (2 * pi);
        else
            f_est = 0;
        end
    else
        f_est = 0;
    end

    f_true = 1000;
    mse = (f_est - f_true)^2;
end

function mse = estimate_freq_spectral_line(signal, fs, fft_points)
    X = abs(fft(signal, fft_points));
    [~, max_idx] = max(X);
    f_est = (max_idx - 1) * fs / fft_points;
    f_true = 1000;
    mse = (f_est - f_true)^2;
end

function mse = estimate_freq_rational_combination(signal, fs)
    fft_points = 200;
    X = abs(fft(signal, fft_points));
    [~, max_idx] = max(X);
    f_est_1 = (max_idx - 1) * fs / fft_points;
    k1 = max_idx - 1;
    if max_idx > 1 && max_idx < fft_points
        ratio1 = X(k1) / X(k1+1);
        ratio2 = X(k1) / X(k1-1);
        delta1 = 1 / (1 + ratio1);
        delta2 = -1 / (1 + ratio2);
        f_est_2 = fs * ((k1 + delta1) / fft_points);
        f_est_3 = fs * ((k1 + delta2) / fft_points);
        f_est = (f_est_1 + f_est_2 + f_est_3) / 3;
    else
        f_est = f_est_1;
    end
    f_true = 1000;
    mse = (f_est - f_true)^2;
end

function mse = estimate_freq_gwlp(signal, fs)
    p = 2;
    w = hamming(length(signal))';
    weighted_signal = signal .* w;
    a = lpc(weighted_signal, p);
    if isvector(a)
```

```
        r = roots(a);
        if ~isempty(r)
            [~, idx] = min(abs(abs(r) - 1));
            est_angle = angle(r(idx));
            f_est = fs * est_angle / (2 * pi);
        else
            f_est = 0;
        end
    else
        f_est = 0;
    end
    f_true = 1000;
    mse = (f_est - f_true)^2;
end

function mse = estimate_freq_crb(signal, fs)
    N = length(signal);
    noise_power = var(signal - mean(signal));
    signal_power = var(signal);
    crb = 6 / (N * (N^2 - 1)) * noise_power / signal_power * fs^2;
    f_true = 1000;
    f_est = f_true + sqrt(crb) * randn;
    mse = (f_est - f_true)^2;
end
```

Listing 5: Applied Experiment: Frequency Estimation

Discrete System Frequency Domain Analysis

Experimental Purpose

The purpose of this experiment is to analyze discrete-time systems in the frequency domain and understand their behavior using frequency response analysis and pole-zero plots. By exploring the system's frequency response, the experiment aims to provide insights into how the system affects different frequency components of input signals. Additionally, the experiment will examine the significance of zero and pole locations in defining the characteristics of discrete-time systems.

Experimental Principle

Frequency Response Analysis

Discrete-time systems can be represented using difference equations of the form:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{j=1}^{N} a_j y[n-j], \tag{18}$$

where $x[n]$ and $y[n]$ are the input and output sequences, respectively, and b_k and a_j are system coefficients. The frequency response of such a system is derived from its system function $H(z)$, defined as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}}. \tag{19}$$

For a stable system, the poles should lie inside the unit circle. The frequency response $H(e^{j\omega})$ is obtained by evaluating $H(z)$ on the unit circle $z = e^{j\omega}$:

$$H(e^{j\omega}) = \frac{B(e^{j\omega})}{A(e^{j\omega})}. \tag{20}$$

The magnitude and phase responses are then calculated as follows:

$$|H(e^{j\omega})| = \sqrt{\mathrm{Re}\{H(e^{j\omega})\}^2 + \mathrm{Im}\{H(e^{j\omega})\}^2} \tag{21}$$

$$\arg\{H(e^{j\omega})\} = \arctan\left(\frac{\mathrm{Im}\{H(e^{j\omega})\}}{\mathrm{Re}\{H(e^{j\omega})\}}\right). \tag{22}$$

Zero-Pole Analysis

The system function $H(z)$ can also be expressed in terms of its zeros and poles:

$$H(z) = K\frac{(z - z_1)(z - z_2)\dots(z - z_M)}{(z - p_1)(z - p_2)\dots(z - p_N)}, \tag{23}$$

where z_k are the zeros, p_k are the poles, and K is a gain factor. The poles and zeros provide valuable insights into the system's frequency response. The pole-zero plot visually represents these characteristics.

Given the digital system with the following transfer function:

$$H(z) = \frac{0.0528 + 0.0797z^{-1} + 0.1295z^{-2} + 0.1295z^{-3} + 0.0797z^{-4} + 0.0528z^{-5}}{1 - 1.8107z^{-1} + 2.4947z^{-2} - 1.8801z^{-3} + 0.9537z^{-4} - 0.2336z^{-5}} \tag{24}$$

We aim to find the system's zeros, poles, and plot the magnitude and phase response.

Given the transfer function:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + \cdots + b_5z^{-5}}{1 + a_1z^{-1} + a_2z^{-2} + \cdots + a_5z^{-5}} \tag{25}$$

We have the numerator coefficients $B = [0.0528, 0.0797, 0.1295, 0.1295, 0.0797, 0.0528]$ and the denominator coefficients $A = [1, -1.8107, 2.4947, -1.8801, 0.9537, -0.2336]$.

Using these coefficients, we compute the system's zeros, poles, and gain k. We can summarize these as follows:

- **Zeros:** $[\mathbf{z_1}, \mathbf{z_2}, \dots]$
- **Poles:** $[\mathbf{p_1}, \mathbf{p_2}, \dots]$
- **Gain:** k

The magnitude and phase response of the system are given by the following plots.

In this analysis, we examined a digital system defined by a transfer function. We successfully determined the zeros, poles, and gain of the system, and plotted its magnitude and phase response.

```
numerator_coeffs = [0.0528, 0.0797, 0.1295, 0.1295, 0.0797, 0.0528];
denominator_coeffs = [1, -1.8107, 2.4947, -1.8801, 0.9537, -0.2336];

[z, p, k] = tf2zp(numerator_coeffs, denominator_coeffs);
disp('Zeros:');
disp(z);
disp('Poles:');
```

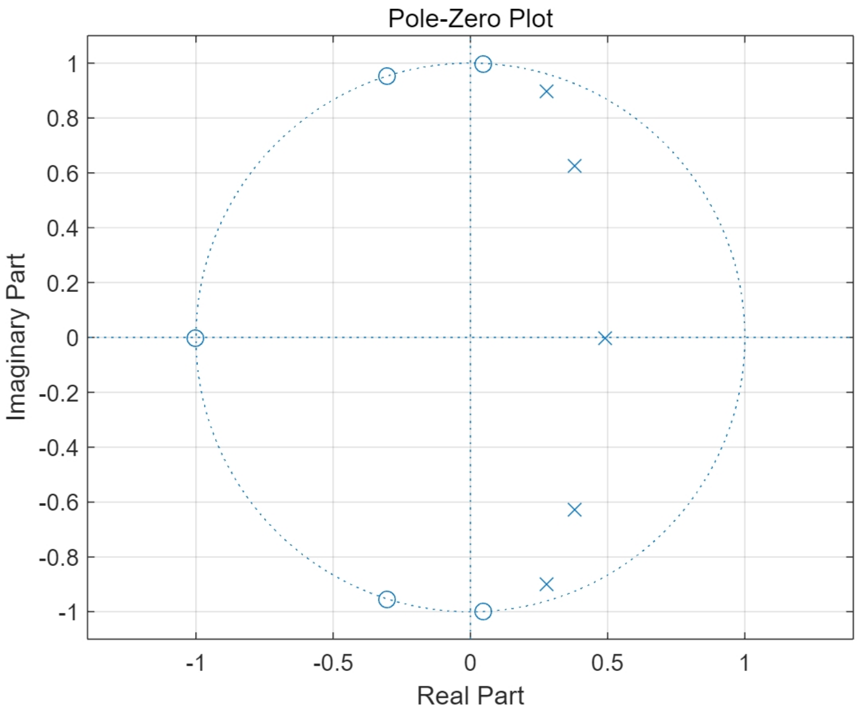

Figure 8: Pole-Zero Plot

```
disp(p);
disp('Gain:');
disp(k);
figure;
zplane(numerator_coeffs, denominator_coeffs);
title('Pole-Zero Plot');
xlabel('Real Part');
ylabel('Imaginary Part');
grid on;
[H, W] = freqz(numerator_coeffs, denominator_coeffs, 'whole', 2000);
magnitude = abs(H);
phase = angle(H);

figure;
subplot(2,1,1);
plot(W/pi, magnitude);
title('Magnitude Response');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Magnitude');
grid on;

subplot(2,1,2);
plot(W/pi, phase);
title('Phase Response');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Phase (radians)');
grid on;
```

Listing 6: Applied Experiment: Frequency Estimation

The second-order cascade form of the system function is expressed as follows:

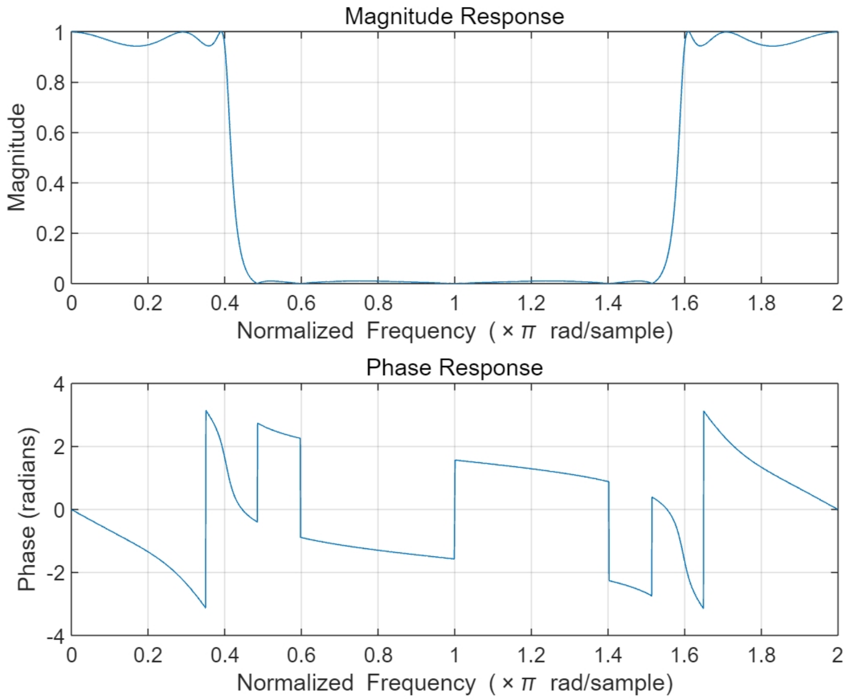

Figure 9: Magnitude Response

$$H(z) = k \prod_{i=1}^{n} \frac{b_{i0} + b_{i1}z^{-1} + b_{i2}z^{-2}}{a_{i0} + a_{i1}z^{-1} + a_{i2}z^{-2}} \tag{26}$$

The second-order sections (SOS) representation of the system function is:

$$\text{SOS} = \begin{bmatrix} b_{10} & b_{11} & b_{12} & a_{10} & a_{11} & a_{12} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{n0} & b_{n1} & b_{n2} & a_{n0} & a_{n1} & a_{n2} \end{bmatrix} \tag{27}$$

The magnitude and phase response of the system are given by the following plots.

In this analysis, we examined a digital system defined by a transfer function. We successfully determined the zeros, poles, and gain of the system, expressed it in second-order cascade form, and plotted its magnitude and phase response.

```
numerator_coeffs = [1, -0.1, -0.2, -0.3, -0.3];
denominator_coeffs = [1, 0.1, 0.2, 0.2, 0.5];

[z, p, k] = tf2zp(numerator_coeffs, denominator_coeffs);
disp('Zeros:');
disp(z);
disp('Poles:');
disp(p);
disp('Gain:');
disp(k);
figure;
zplane(numerator_coeffs, denominator_coeffs);
title('Pole-Zero Plot');
xlabel('Real Part');
ylabel('Imaginary Part');
grid on;
```

```
sos = tf2sos(numerator_coeffs, denominator_coeffs);
disp('Second-Order Sections (SOS):');
disp(sos);
[H, W] = freqz(numerator_coeffs, denominator_coeffs, 'whole', 2000);
magnitude = abs(H);
phase = angle(H);

figure;
subplot(2,1,1);
plot(W/pi, magnitude);
title('Magnitude Response');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Magnitude');
grid on;

subplot(2,1,2);
plot(W/pi, phase);
title('Phase Response');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Phase (radians)');
grid on;
```

Listing 7: Applied Experiment: Frequency Estimation

Discussion

In this comprehensive exploration, the objective was to delve into the realms of frequency estimation and discrete system frequency domain analysis, leveraging MATLAB for practical experimentation and LaTeX for documentation. The investigation was bifurcated into two parts: the first focused on the application of FFT algorithms for single-frequency periodic signals, symmetry properties, and various frequency estimation methods across different SNR levels, while the second part concentrated on analyzing discrete-time systems' frequency response and pole-zero characteristics. Through a meticulously crafted methodology involving signal generation, noise addition, frequency estimation, and system analysis, the experiments yielded insightful results, visually represented through MATLAB-generated plots and articulated in a structured LaTeX document. The task illuminated the significance of different frequency estimation techniques and underscored the analytical power of the FFT in digital signal processing, ultimately demonstrating the efficacy of both MATLAB and LaTeX in tackling complex signal analysis problems.

```
Zeros:
  -1.0000 + 0.0000i
   0.0471 + 0.9989i
   0.0471 - 0.9989i
  -0.3018 + 0.9534i
  -0.3018 - 0.9534i

Poles:
   0.2788 + 0.8973i
   0.2788 - 0.8973i
   0.3811 + 0.6274i
   0.3811 - 0.6274i
   0.4910 + 0.0000i

Gain:
    0.0528
```

Figure 10: Calculation results

```
Zeros:
   0.9644 + 0.0000i
  -0.1139 + 0.6897i
  -0.1139 - 0.6897i
  -0.6366 + 0.0000i

Poles:
   0.5276 + 0.6997i
   0.5276 - 0.6997i
  -0.5776 + 0.5635i
  -0.5776 - 0.5635i

Gain:
     1

Second-Order Sections (SOS):
    1.0000   -0.3278   -0.6140    1.0000    1.1552    0.6511
    1.0000    0.2278    0.4886    1.0000   -1.0552    0.7679
```

Figure 11: Second-Order Cascade Form Response

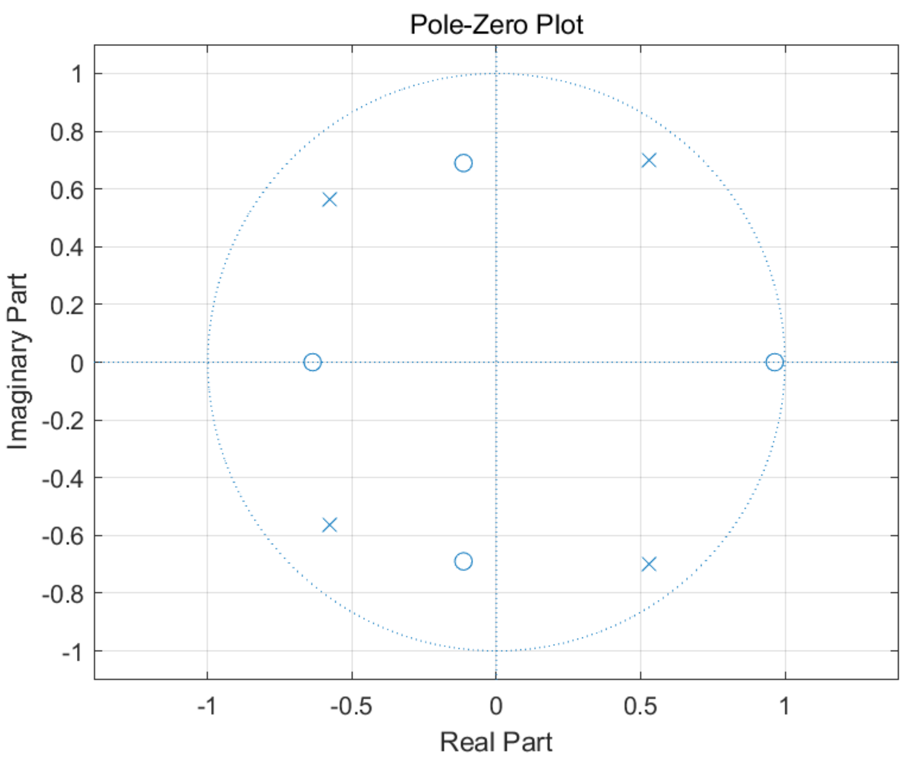

Figure 12: Pole-Zero Plot

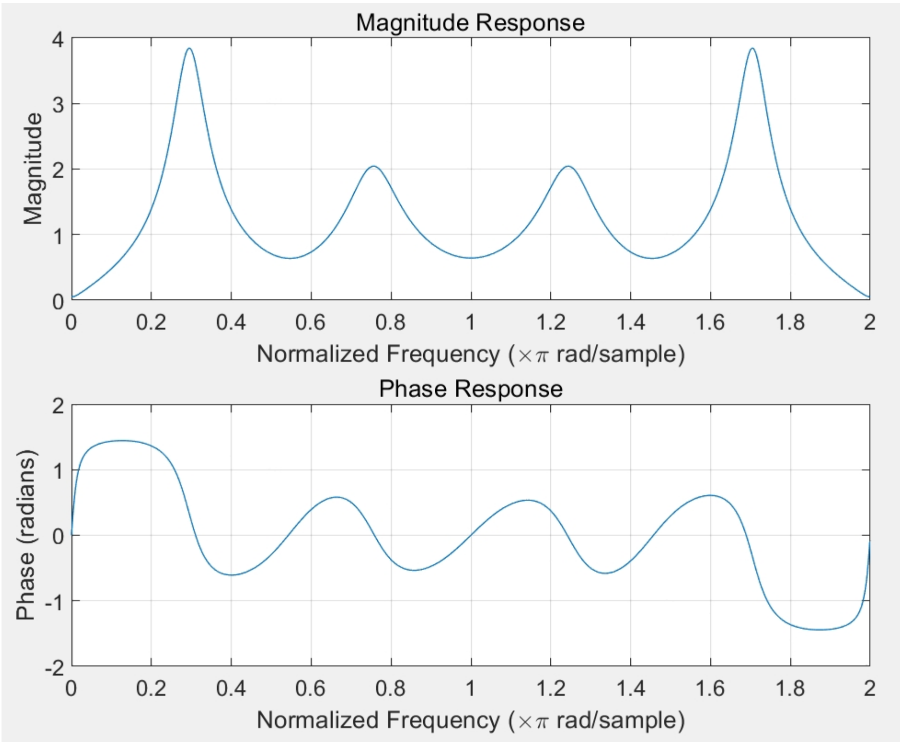

Figure 13: Magnitude Response