



**SRM VALLIAMMAI ENGINEERING COLLEGE**

(An Autonomous Institution)

SRM Nagar, Kattankulathur – 603 203.



**DEPARTMENT OF  
ELECTRONICS AND COMMUNICATION ENGINEERING**

**M.E.-COMMUNICATION SYSTEMS**

**CU3166- ADVANCED DIGITAL SIGNAL PROCESSING  
LABORATORY**

**LAB MANUAL**

**Regulation-2023**

**I YEAR / I SEM M.E ECE ( 2026- 2027 ODD SEMESTER)**

*Prepared by*

**DR.S.R.PREETHI ASSOCIATE PROFESSOR**

**NAME** : \_\_\_\_\_

**REGISTER NO** : \_\_\_\_\_

**YEAR/SEMESTER** : \_\_\_\_\_

## OBJECTIVES:

- ❖ To perform random signal processing operations and implement in simulation tool.
- ❖ To analyze the frequency response of filters using signal processing method
- ❖ To detect signal in the presence of noise using optimum filter
- ❖ To estimate and implement the power spectral density of random process
- ❖ To design and implement the adaptive filters for nonstationary signals
- ❖ To demonstrate and verify the upsampling and down sampling of signals

## LIST OF EXPERIMENTS (Using Simulation Tool)

- 1 Generate the auto correlation sequence of random sequence for Gaussian noise and uniform white noise (N=100 samples).
- 2 Generate an autoregressive, moving average and autoregressive moving average process of unit variance white Gaussian noise (N=100 samples).
- 3 Design a linear phase low pass filter having a cut-off frequency of  $\pi/2$  using Pade approximation method and Prony's method.
- 4 Design of LPC filter using Levinson-Durbin algorithm.
- 5 Design a FIR Wiener filter to extract a desired signal from the noise corrupted signal.
- 6 Estimation the PSD of a noisy signal using periodogram and modified periodogram.
- 7 Estimation of power spectrum of the given random sequence using Nonparametric methods (Bartlett, Welch and Blackman Tukey).
- 8 Estimation of power spectrum of the given random sequence using parametric methods (AR, MA and ARMA).
- 9 Design an adaptive filter to extract a desired signal from the given noisy signal by cancelling the noise using LMS Algorithm.
- 10 Design an adaptive filter to extract a desired signal from the given noisy signal by cancelling the noise using RLS Algorithm.
- 11 Upsampling the discrete time sequence by L times and plot the spectrum of both the given sequence and upsampled sequence.
- 12 Down sampling the discrete time sequence by M times and plot the spectrum of both the given sequence and down sampled sequence.

**OUTCOMES:**

On completion of the course, the student will be able to

- ❖ Generate Random sequences and analyzed with processes using simulation tool
- ❖ Design and analyze the frequency response of filters for the given specification using signal processing method
- ❖ Detect signal in the presence of noise using optimum filter
- ❖ Estimate power spectrum of the given random sequence using parametric/nonparametric estimation methods
- ❖ Implement adaptive filters using LMS/RLS algorithm
- ❖ Demonstrate the discrete time systems at various sampling rates

**COURSE OUTCOMES - PROGRAM OUTCOMES MATRIX**

| Course Outcomes (COs) | Program Outcomes (POs) |   |     |     |     |     |
|-----------------------|------------------------|---|-----|-----|-----|-----|
|                       | 1                      | 2 | 3   | 4   | 5   | 6   |
| <b>CU3166.1</b>       | 1                      | - | 2   | 3   | -   | -   |
| <b>CU3166.2</b>       | 2                      | - | 3   | 3   | 1   | -   |
| <b>CU3166.3</b>       | 2                      | - | 3   | 3   | 2   | -   |
| <b>CU3166.4</b>       | 2                      | - | 3   | 3   | 1   | -   |
| <b>CU3166.5</b>       | 3                      | - | 3   | 3   | 2   | 2   |
| <b>CU3166.6</b>       | 3                      | - | 3   | 3   | 2   | 2   |
| <b>Average</b>        | 2.1                    | - | 2.8 | 3.0 | 1.6 | 2.0 |

# 1. Generation of Autocorrelation Sequence for Random Sequences

## AIM

To generate and analyze the autocorrelation sequence of Gaussian and uniform white-noise random sequences using MATLAB.

## APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB and Signal Processing Toolbox; random sequence data; plotting facilities.

## THEORY / PROCEDURE

Generate the required random sequence, calculate its autocorrelation, and plot the resulting sequence for analysis.

## MATLAB PROGRAM

1. Generate the auto correlation sequence of random sequence for Gaussian noise and uniform white noise (N=100 samples).

```
Parameters
N = 100;                % Number of samples
e = randn(N,1);         % Unit variance white Gaussian noise
```

```
%1. Autoregressive (AR) process
Example: AR(1):  $x(n) = 0.75x(n-1) + e(n)$ 
ar_coeff = [1 -0.75];    % AR polynomial ( $1 - 0.75z^{-1}$ )
ma_coeff = 1;            % No MA part
x_ar = filter(ma_coeff, ar_coeff, e);
```

```
%2. Moving Average (MA) process
Example: MA(1):  $x(n) = e(n) + 0.65e(n-1)$ 
ar_coeff = 1;            % No AR part
ma_coeff = [1 0.65];     % MA polynomial ( $1 + 0.65z^{-1}$ )
x_ma = filter(ma_coeff, ar_coeff, e);
```

```
%3. Autoregressive Moving Average (ARMA) process
Example: ARMA(1,1):  $x(n) = 0.6x(n-1) + e(n) + 0.5e(n-1)$ 
ar_coeff = [1 -0.6];     % AR part
ma_coeff = [1 0.5];      % MA part
x_arma = filter(ma_coeff, ar_coeff, e);
```

```
%Plot results
figure;
subplot(3,1,1);
plot(x_ar, 'b'); grid on;
title('Autoregressive Process (AR(1))');
```

```
subplot(3,1,2);
plot(x_ma, 'r'); grid on;
title('Moving Average Process (MA(1))');
```

```
subplot(3,1,3);
plot(x_arma, 'k'); grid on;
title('Autoregressive Moving Average Process (ARMA(1,1))');
```

## RESULT

The autocorrelation sequences of the generated random processes were obtained and plotted, demonstrating the correlation characteristics of the sequences.

## 2. Generation of AR, MA and ARMA Processes

### AIM

To generate Autoregressive (AR), Moving Average (MA), and Autoregressive Moving Average (ARMA) processes using unit-variance white Gaussian noise.

### APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; Signal Processing Toolbox; unit-variance white Gaussian noise sequence.

### THEORY / PROCEDURE

Generate unit-variance white Gaussian noise. Implement AR(1), MA(1), and ARMA(1,1) models using the specified coefficients, then plot their outputs.

### MATLAB PROGRAM

2. Generate an autoregressive, moving average and autoregressive moving average process of unit variance white Gaussian noise (N=100 samples).

```
%Parameters
N = 100; % Number of samples
e = randn(N,1); % Unit variance white Gaussian noise

%1. Autoregressive (AR) process
Example: AR(1):  $x(n) = 0.75x(n-1) + e(n)$ 
ar_coeff = [1 -0.75]; % AR polynomial ( $1 - 0.75z^{-1}$ )
ma_coeff = 1; % No MA part
x_ar = filter(ma_coeff, ar_coeff, e);

%2. Moving Average (MA) process
Example: MA(1):  $x(n) = e(n) + 0.65e(n-1)$ 
ar_coeff = 1; % No AR part
ma_coeff = [1 0.65]; % MA polynomial ( $1 + 0.65z^{-1}$ )
x_ma = filter(ma_coeff, ar_coeff, e);

%3. Autoregressive Moving Average (ARMA) process
Example: ARMA(1,1):  $x(n) = 0.6x(n-1) + e(n) + 0.5e(n-1)$ 
ar_coeff = [1 -0.6]; % AR part
ma_coeff = [1 0.5]; % MA part
x_arma = filter(ma_coeff, ar_coeff, e);

%Plot results
figure;
subplot(3,1,1);
plot(x_ar, 'b'); grid on;
title('Autoregressive Process (AR(1))');

subplot(3,1,2);
plot(x_ma, 'r'); grid on;
title('Moving Average Process (MA(1))');

subplot(3,1,3);
plot(x_arma, 'k'); grid on;
title('Autoregressive Moving Average Process (ARMA(1,1))');
```

### RESULT

The AR(1), MA(1), and ARMA(1,1) processes were generated successfully and their waveforms were plotted.

### 3. Design of Linear-Phase Low-Pass Filter using Prony and Padé-Style Approximation

#### AIM

To design a linear-phase low-pass filter with cut-off frequency  $\pi/2$  and compare Prony and Padé-style (invfreqz) IIR approximations.

#### APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; Signal Processing Toolbox; filter-design functions; plotting facilities.

#### THEORY / PROCEDURE

Set the cut-off frequency to  $\pi/2$  and form a length-101 linear-phase FIR response using a windowed sinc. Obtain Prony and invfreqz rational approximations, then compare magnitude, phase, impulse response, and pole-zero plots.

#### MATLAB PROGRAM

3. Design a linear phase low pass filter having a cut-off frequency of  $\pi/2$  using Pade approximation method and Prony's method.

```
clear; close all; clc;
% Parameters
wc = pi/2;          % cutoff frequency
L = 101;            % length of the desired FIR (odd -> exact symmetric linear phase)
M = (L-1)/2;        % linear phase delay (samples)
n = 0:L-1;

% Ideal (ideal sinc) linear-phase impulse response (time-shifted)
hd = zeros(1,L);
for k = 1:L
    m = k-1 - M;
    if m == 0
        hd(k) = wc/pi;
    else
        hd(k) = sin(wc*m)/(pi*m);
    end
end

% Apply a window (Hamming) to reduce truncation ripples (keeps linear phase)
w = hamming(L)';
h_fir = hd .* w;

% Normalize passband gain (optional)
h_fir = h_fir / sum(h_fir); % makes DC gain = 1 (approx)

% Frequency grid
nfft = 2048;
[H_fir, wgrid] = freqz(h_fir,1,nfft);

% Method 1: Prony's method
% Fit an IIR (b(z)/a(z)) to the desired impulse response samples
% choose model orders (try different values; higher -> better magnitude fit)
nb = 8; % numerator order
na = 8; % denominator order

% MATLAB has prony() in Signal Processing Toolbox
[b_prony, a_prony] = prony(h_fir, nb, na);

[H_prony, ~] = freqz(b_prony,a_prony,nfft);

% Method 2: "Pade-style" rational approximation using invfreqz
% invfreqz fits IIR to a desired frequency response (least-squares over freq)
% Treat invfreqz result as a Pade-like rational approximation.
% Sample desired frequency response from the FIR (magnitude & phase)
Hd_desired = H_fir; % complex desired values on wgrid
% Fit IIR using invfreqz
% Note: invfreqz requires the Signal Processing Toolbox
[b_inv, a_inv] = invfreqz(Hd_desired,wgrid,nb,na); % nb, na as above

[H_inv, ~] = freqz(b_inv,a_inv,nfft);

% Compute and plot results
figure('Name','Magnitude responses','NumberTitle','off','Units','normalized','Position',[0.05 0.05 0.9 0.8]);
```

```

subplot(2,2,1);
plot(wgrid/pi,20*log10(abs(H_fir)+eps),'LineWidth',1.5); hold on;
plot(wgrid/pi,20*log10(abs(H_prony)+eps),'--','LineWidth',1.2);
plot(wgrid/pi,20*log10(abs(H_inv)+eps),':','LineWidth',1.2);
grid on;
xlabel('Normalized frequency \omega/\pi'); ylabel('Magnitude (dB)');
title('Magnitude response (FIR ideal vs Prony IIR vs invfreqz IIR)');
legend('Linear-phase FIR (windowed-sinc)','Prony IIR','invfreqz (Pade-
style)','Location','Best');

% Zoom passband and stopband views
subplot(2,2,2);
plot(wgrid/pi,abs(H_fir),'LineWidth',1.5); hold on;
plot(wgrid/pi,abs(H_prony),'--','LineWidth',1.2);
plot(wgrid/pi,abs(H_inv),':','LineWidth',1.2);
grid on; xlabel('\omega/\pi'); ylabel('Magnitude');
title('Magnitude (linear scale)');
legend('FIR','Prony','invfreqz','Location','Best');

% Phase responses
subplot(2,2,3);
plot(wgrid/pi,unwrap(angle(H_fir)),'LineWidth',1.5); hold on;
plot(wgrid/pi,unwrap(angle(H_prony)),'--','LineWidth',1.2);
plot(wgrid/pi,unwrap(angle(H_inv)),':','LineWidth',1.2);
grid on; xlabel('\omega/\pi'); ylabel('Phase (rad)');
title('Phase responses (unwrap)');
legend('FIR (linear phase)','Prony IIR','invfreqz IIR','Location','Best');

% Impulse responses
subplot(2,2,4);
stem(0:L-1,h_fir,'filled','MarkerSize',3); hold on;
% compute longer impulse responses for IIRs
imp_len = 120;
imp = [1, zeros(1,imp_len-1)];
h_prony_imp = filter(b_prony,a_prony,imp);
h_inv_imp = filter(b_inv,a_inv,imp);
stem(0:imp_len-1,h_prony_imp,'--','Marker','none');
stem(0:imp_len-1,h_inv_imp,':','Marker','none');
grid on; xlabel('n'); ylabel('h[n]');
title('Impulse responses (FIR true linear-phase vs IIR approximations)');
legend('FIR (windowed-sinc)','Prony IIR','invfreqz IIR','Location','Best');

% Display coefficients and poles/zeros
fprintf('\n=== FIR (linear-phase) summary ===\nLength L = %d, linear phase delay M = %.1f
samples\n',L,M);

fprintf('\n=== Prony IIR (nb=%d, na=%d) coefficients ===\n',nb,na);
disp('b_prony ='); disp(b_prony(:));
disp('a_prony ='); disp(a_prony(:));

fprintf('\nPoles of Prony IIR:\n'); disp(roots(a_prony));
fprintf('Zeros of Prony IIR:\n'); disp(roots(b_prony));

fprintf('\n=== invfreqz IIR (Pade-style) (nb=%d, na=%d) coefficients ===\n',nb,na);
disp('b_inv ='); disp(b_inv(:));
disp('a_inv ='); disp(a_inv(:));

fprintf('\nPoles of invfreqz IIR:\n'); disp(roots(a_inv));
fprintf('Zeros of invfreqz IIR:\n'); disp(roots(b_inv));

% Optional: plot pole-zero maps
figure('Name','Pole-Zero Maps','NumberTitle','off','Units','normalized','Position',[0.15 0.1 0.7
0.6]);
subplot(1,2,1);
zplane(b_prony,a_prony); title('Prony IIR P-Z map');
subplot(1,2,2);
zplane(b_inv,a_inv); title('invfreqz (Pade-style) P-Z map');

% End of script

```

## RESULT

The linear-phase FIR response and the Prony and invfreqz approximations were obtained. Magnitude, phase, impulse-response, and pole-zero characteristics were plotted for comparison.

## 4. Design of LPC Filter using Levinson-Durbin Algorithm

### AIM

To design an LPC filter using the Levinson-Durbin algorithm and obtain its frequency response.

### APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; speech-like signal; Signal Processing Toolbox.

### THEORY / PROCEDURE

Generate a 50-ms speech-like signal at 8 kHz, compute its autocorrelation, apply Levinson-Durbin recursion for order 10, and plot the signal, autocorrelation, filter response, and pole-zero map.

### MATLAB PROGRAM

4. Design of LPC filter using Levinson-Durbin algorithm.

```
clear; close all; clc;
Step 1: Generate or load a speech-like signal
fs = 8000; % Sampling frequency
t = 0:1/fs:0.05; % 50 ms signal
x = sin(2*pi*200*t) + 0.5*sin(2*pi*400*t); % Synthetic vowel-like tone
x = x + 0.05*randn(size(x)); % Add small noise

Step 2: LPC order
p = 10; % Order of LPC (usually 8-16 for speech)

Step 3: Compute autocorrelation sequence
r = xcorr(x, p, 'biased');
r = r(p+1:end); % Keep only non-negative lags (r[0] ... r[p])

Step 4: Levinson-Durbin recursion
[a, e] = levinson(r, p);

% Note:
% a = LPC coefficients, including a(1) = 1
% e = prediction error (residual power)

Step 5: Display LPC coefficients
disp('LPC Coefficients (a):');
disp(a);
fprintf('Prediction error (residual energy): %f\n', e);
LPC Coefficients (a):
Columns 1 through 7

    1.0000    -1.2075    -0.2965     0.2474     0.2844     0.1790    -0.0303

Columns 8 through 11

   -0.0585   -0.0553   -0.1011     0.0867

Prediction error (residual energy): 0.007091

Step 6: Frequency response of LPC filter
[H, w] = freqz(1, a, 512, fs);

Step 7: Plot results
figure('Name','LPC Filter Design using Levinson-Durbin','NumberTitle','off');

subplot(3,1,1);
plot(t, x, 'LineWidth',1.2);
xlabel('Time (s)'); ylabel('Amplitude');
title('Input Signal (Speech-like)');
grid on;

subplot(3,1,2);
stem(0:p, r, 'filled');
xlabel('Lag'); ylabel('Autocorrelation');
title('Autocorrelation Sequence r(k)');
grid on;

subplot(3,1,3);
plot(w, 20*log10(abs(H)), 'LineWidth',1.5);
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
title(sprintf('LPC Filter (Order = %d) Frequency Response', p));
grid on;
Step 8: Pole-Zero Plot
```

```
figure('Name','LPC Filter Poles and Zeros','NumberTitle','off');  
zplane(1, a);  
title('Pole-Zero Plot of LPC Filter');  
  
% END OF SCRIPT
```

## RESULT

The LPC coefficients and prediction error were obtained. The frequency response and pole-zero plot of the LPC filter were generated.

## 5. Design of FIR Wiener Filter for Signal Extraction

### AIM

To design an FIR Wiener filter to extract a desired signal from a noise-corrupted signal.

### APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; desired signal; additive white Gaussian noise; Signal Processing Toolbox.

### THEORY / PROCEDURE

Generate the desired signal and noisy observation, estimate autocorrelation and cross-correlation sequences, form the Wiener-Hopf equations, solve for optimum FIR coefficients, filter the input, calculate MSE, and plot the results.

### MATLAB PROGRAM

5. Design a FIR Wiener filter to extract a desired signal from the noise corrupted signal.

```
clear; close all; clc;
Step 1: Generate the desired (clean) signal
fs = 1000; % Sampling frequency (Hz)
t = 0:1/fs:1; % Time vector (1 second duration)
d = sin(2*pi*100*t) + 0.5*sin(2*pi*200*t); % Desired clean signal

Step 2: Add noise to create noisy observation
noise = 0.5*randn(size(t)); % Additive white Gaussian noise
x = d + noise; % Noisy signal (observation)

Step 3: Specify FIR Wiener filter order
M = 20; % Number of filter coefficients

Step 4: Estimate correlation sequences
rxx = xcorr(x, M, 'biased'); % Autocorrelation of input
rxx = rxx(M+1:end); % Keep non-negative lags (rxx[0:M])

rdx = xcorr(d, x, M, 'biased'); % Cross-correlation between desired and input
rdx = rdx(M+1:end); % Keep non-negative lags (rdx[0:M])

Step 5: Form Wiener-Hopf equations and solve for w_opt
Rxx = toeplitz(rxx(1:M)); % Autocorrelation matrix
p = rdx(1:M)'; % Cross-correlation vector
w_opt = Rxx \ p; % Optimal Wiener filter coefficients

disp('Optimal Wiener Filter Coefficients:');
disp(w_opt);
Optimal Wiener Filter Coefficients:
Columns 1 through 7

    0.1850    0.1171   -0.0365   -0.1133   -0.0591   -0.0083   -0.0506

Columns 8 through 14

   -0.1000   -0.0408    0.1106    0.1788    0.0905   -0.0449   -0.0855

Columns 15 through 20

   -0.0243    0.0096   -0.0499   -0.0958   -0.0287    0.0946

Step 6: Filter the noisy input signal
y = filter(w_opt, 1, x); % Output of Wiener filter

Step 7: Compute performance
error = d - y; % Estimation error
MSE = mean(error.^2); % Mean Square Error
fprintf('\nMean Square Error (MSE): %.6f\n', MSE);
Mean Square Error (MSE): 0.046856

Step 8: Plot results
figure('Name','FIR Wiener Filter Results','NumberTitle','off','Position',[100 100 900 600]);

subplot(3,1,1);
plot(t, d, 'LineWidth',1.2);
xlabel('Time (s)'); ylabel('Amplitude');
title('Desired Signal d[n]');
grid on;
subplot(3,1,2);
```

```

plot(t, x, 'r', 'LineWidth',1.0);
xlabel('Time (s)'); ylabel('Amplitude');
title('Noisy Signal x[n]');
grid on;

subplot(3,1,3);
plot(t, y, 'g', 'LineWidth',1.2); hold on;
plot(t, d, 'k--', 'LineWidth',1.0);
xlabel('Time (s)'); ylabel('Amplitude');
title('Wiener Filter Output y[n] vs Desired Signal d[n]');
legend('Wiener Output','Desired Signal','Location','best');
grid on;

Step 9: Frequency response of Wiener filter
[H, w] = freqz(w_opt, 1, 512, fs);
figure('Name','Wiener Filter Frequency Response','NumberTitle','off');
plot(w, 20*log10(abs(H)), 'LineWidth',1.5);
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Magnitude Response of FIR Wiener Filter');
grid on;
% END OF SCRIPT

```

## RESULT

The optimum Wiener filter coefficients were obtained and the noisy signal was filtered successfully. The reported mean-square error was 0.046856.

## 6. PSD Estimation using Periodogram and Modified Periodogram

### AIM

To estimate the power spectral density (PSD) of a noisy signal using the periodogram and modified periodogram methods.

### APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; noisy sinusoidal signal; rectangular and Hamming windows.

### THEORY / PROCEDURE

Generate a 100-Hz sinusoid corrupted by white Gaussian noise. Estimate PSD using a rectangular-window periodogram and a Hamming-window modified periodogram, then compare the spectra.

### MATLAB PROGRAM

6. Estimation the PSD of a noisy signal using periodogram and modified periodogram.

```
clear; close all; clc;
Step 1: Generate noisy signal
fs = 1000; % Sampling frequency (Hz)
N = 1024; % Number of samples
n = 0:N-1; % Time index
% Clean sinusoidal signal
f1 = 100; % Frequency of signal (Hz)
x_clean = sin(2*pi*f1*n/fs);
% Additive white Gaussian noise
noise = 0.5 * randn(size(x_clean));

% Noisy signal
x = x_clean + noise;

Step 2: Compute Periodogram (rectangular window)
X = fft(x);
Pxx_periodogram = (1/(fs*N)) * abs(X).^2; % Periodogram estimate
Pxx_periodogram = Pxx_periodogram(1:N/2); % One-sided PSD
f = (0:N/2-1)*(fs/N); % Frequency axis

Step 3: Compute Modified Periodogram (using Hamming window)
w = hamming(N)'; % Window function
xw = x .* w; % Apply window
U = (1/N) * sum(w.^2); % Normalization factor
Xw = fft(xw, N);
Pxx_modified = (1/(fs*N*U)) * abs(Xw).^2; % Modified periodogram
Pxx_modified = Pxx_modified(1:N/2);

Step 4: Plot PSD estimates
figure('Name','PSD Estimation','NumberTitle','off','Position',[100 100 900 600]);

subplot(2,1,1);
plot(f, 10*log10(Pxx_periodogram), 'b', 'LineWidth', 1.5);
xlabel('Frequency (Hz)'); ylabel('Power/Frequency (dB/Hz)');
title('PSD Estimate using Periodogram (Rectangular Window)');
grid on;

subplot(2,1,2);
plot(f, 10*log10(Pxx_modified), 'r', 'LineWidth', 1.5);
xlabel('Frequency (Hz)'); ylabel('Power/Frequency (dB/Hz)');
title('PSD Estimate using Modified Periodogram (Hamming Window)');
grid on;

Step 5: Compare both estimates on one plot
figure('Name','Comparison: Periodogram vs Modified Periodogram','NumberTitle','off');
plot(f, 10*log10(Pxx_periodogram), 'b--', 'LineWidth', 1.2); hold on;
plot(f, 10*log10(Pxx_modified), 'r', 'LineWidth', 1.5);
xlabel('Frequency (Hz)'); ylabel('Power/Frequency (dB/Hz)');
title('Comparison of Periodogram and Modified Periodogram');
legend('Periodogram (Rectangular)', 'Modified Periodogram (Hamming)', 'Location', 'Best');
grid on;

% END OF SCRIPT
```

### RESULT

PSD estimates were obtained using both methods and their spectra were plotted and compared.

# 7. Power Spectrum Estimation using Nonparametric Methods

## AIM

To estimate the power spectrum of a random sequence using Bartlett, Welch, and Blackman-Tukey methods.

## APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; random sequence; Hamming and Blackman windows; Signal Processing Toolbox.

## THEORY / PROCEDURE

Generate a random sequence and estimate its spectrum using Bartlett segmentation, Welch's method with a Hamming window, and Blackman-Tukey autocorrelation windowing. Plot all three estimates.

## MATLAB PROGRAM

7. Estimation of power spectrum of the given random sequence using Nonparametric methods (Bartlett, Welch and Blackman Tukey).

```
clear; clc; close all;
1. Generate / Load a Random Sequence
N = 1024;
x = randn(1, N);    % Replace with your own sequence

2. Bartlett Method
L = 128;            % Segment length
K = floor(N / L);   % Number of segments
Px_bartlett = 0;
for k = 1:K
    xk = x((k-1)*L + 1 : k*L);
    Xk = fft(xk, 1024);
    Px_bartlett = Px_bartlett + (abs(Xk).^2) / L;
end
Px_bartlett = Px_bartlett / K;
f = linspace(0,1,1024);

3. Welch Method
win = hamming(L);
noverlap = L/2;
[Px_welch, f_w] = pwelch(x, win, noverlap, 1024, 1);

4. Blackman-Tukey Method
M = 100;            % Maximum lag
w = blackman(2*M+1); % Blackman window

r = xcorr(x, M, 'biased'); % Autocorrelation
r = r .* w;           % Windowing
Px_bt = abs(fft(r, 1024));

5. Plot Results
figure;
subplot(3,1,1)
plot(f, 10*log10(Px_bartlett));
title('Bartlett Method');
xlabel('Normalized Frequency'); ylabel('Power (dB)');

subplot(3,1,2)
plot(f_w, 10*log10(Px_welch));
title('Welch Method');
xlabel('Normalized Frequency'); ylabel('Power (dB)');

subplot(3,1,3)
plot(f, 10*log10(Px_bt));
title('Blackman-Tukey Method');
xlabel('Normalized Frequency'); ylabel('Power (dB)');
```

## RESULT

The power spectra using Bartlett, Welch, and Blackman-Tukey methods were obtained and plotted for comparison.

## 8. Power Spectrum Estimation using Parametric Methods

### AIM

To estimate the power spectrum of a random sequence using AR, MA, and ARMA parametric models.

### APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; random sequence; Signal Processing Toolbox/System Identification functions.

### THEORY / PROCEDURE

Generate a random sequence, estimate an AR model using Yule-Walker, construct an MA model, estimate an ARMA model, and obtain their frequency-domain power spectra.

### MATLAB PROGRAM

8. Estimation of power spectrum of the given random sequence using parametric methods (AR, MA and ARMA).

```
clear; close all; clc;
1. Generate / Load Random Sequence
N = 1024;
x = randn(N,1);           % Replace with your own sequence (column vector)
fs = 1;                   % Sampling frequency
nfft = 1024;              % FFT resolution
f = linspace(0, fs/2, nfft/2);
2. AR Spectrum (Yule-Walker Method)
p = 10;                   % AR model order
[a_ar, e_ar] = aryule(x, p); % AR coefficients
H_ar = freqz(sqrt(e_ar), a_ar, nfft, fs);
PSD_AR = abs(H_ar).^2;
3. MA Spectrum (MA Model using ARMA(0,q))
q = 10;                   % MA order
b_ma = randn(1, q+1);    % Random MA coefficients (example)
a_ma = 1;                 % No AR part
H_ma = freqz(b_ma, a_ma, nfft, fs);
PSD_MA = abs(H_ma).^2;
4. ARMA Spectrum
p_arma = 4;               % AR order
q_arma = 4;               % MA order
% Create ARMA structure and estimate parameters
model = armax(x, [p_arma q_arma]);
% Extract AR and MA coefficients
a_arma = model.A;
b_arma = model.C;
H_arma = freqz(b_arma, a_arma, nfft, fs);
PSD_ARMA = abs(H_arma).^2;
5. Plot Results
figure;
subplot(3,1,1)
plot(f, 10*log10(PSD_AR(1:nfft/2)));
title('AR Spectrum (Yule-Walker)');
xlabel('Frequency'); ylabel('Power (dB)');

subplot(3,1,2)
plot(f, 10*log10(PSD_MA(1:nfft/2)));
title('MA Spectrum');
xlabel('Frequency'); ylabel('Power (dB)');

subplot(3,1,3)
plot(f, 10*log10(PSD_ARMA(1:nfft/2)));
title('ARMA Spectrum');
xlabel('Frequency'); ylabel('Power (dB)');
```

### RESULT

The AR, MA, and ARMA power spectra were estimated and plotted for comparison.

## 9. Adaptive Noise Cancellation using LMS Algorithm

### AIM

To design an adaptive filter using the LMS algorithm for extracting a desired signal from a noisy signal by cancelling correlated noise.

### APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; desired signal; reference noise; adaptive FIR filter.

### THEORY / PROCEDURE

Generate a desired sinusoid and correlated reference noise. Apply an LMS adaptive FIR filter with the specified order and step size, update weights sample-by-sample, and plot the recovered signal and final weights.

### MATLAB PROGRAM

9. Design an adaptive filter to extract a desired signal from the given noisy signal by cancelling the noise using LMS Algorithm.

```
clear; close all; clc;
1. Generate / Load Signals
N = 2000; % Number of samples
t = (1:N)';

% Desired signal (example: sinusoid)
d = sin(2*pi*0.02*t); % Replace with your desired signal

% Noise (reference noise)
noise = 0.5*randn(N,1); % Random noise
secondary_noise = filter([1 0.9],1,noise); % Correlated noise

% Noisy input signal
x = d + secondary_noise;
2. LMS Algorithm Parameters
M = 32; % Filter length
mu = 0.01; % Step size (learning rate)

w = zeros(M,1); % Initial filter weights
y = zeros(N,1); % Filter output
e = zeros(N,1); % Error (desired - output)
3. Adaptive LMS Filtering
for n = M:N
    u = noise(n:-1:n-M+1); % Reference noise input vector
    y(n) = w' * u; % Filter output
    e(n) = x(n) - y(n); % Error signal
    w = w + mu * u * e(n); % LMS weight update
end
4. Plot Results
figure;

subplot(3,1,1)
plot(d); title('Desired Signal'); grid on;

subplot(3,1,2)
plot(x); title('Noisy Signal (Desired + Noise)'); grid on;

subplot(3,1,3)
plot(e); title('Recovered Signal after LMS Noise Cancellation'); grid on;

figure;
plot(w); title('Final LMS Filter Weights'); grid on;
```

### RESULT

The LMS adaptive filter reduced the correlated noise and produced a recovered output signal. The final adaptive filter weights were obtained.

# 10. Adaptive Noise Cancellation using RLS Algorithm

## AIM

To design an adaptive filter using the RLS algorithm for extracting a desired signal from a noisy signal by cancelling correlated noise.

## APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; desired signal; reference noise; adaptive FIR filter.

## THEORY / PROCEDURE

Generate a desired sinusoid and correlated reference noise. Apply the RLS adaptive FIR filter using the specified forgetting factor and initialization, update the gain, weights, and inverse correlation matrix, and plot the recovered signal.

## MATLAB PROGRAM

10. Design an adaptive filter to extract a desired signal from the given noisy signal by cancelling the noise using RLS Algorithm.

```
clear; clc; close all;
1. Generate / Load Signals
N = 2000; % Number of samples
t = (1:N)';
% Desired signal (example)
d = sin(2*pi*0.02*t); % Replace with your desired signal
% Noise source (reference noise)
noise = randn(N,1); % Original noise
ref_noise = filter([1 0.7],1,noise); % Correlated noise
% Noisy signal (desired + correlated noise)
x = d + ref_noise;

2. RLS Algorithm Parameters
M = 32; % Filter order
lambda = 0.99; % Forgetting factor (0.95-1 recommended)
delta = 100; % Initialization factor
w = zeros(M,1); % Filter weights
P = delta * eye(M); % Inverse correlation matrix
y = zeros(N,1); % Filter output
e = zeros(N,1); % Error signal

3. RLS Adaptive Filtering
u = noise(n:-1:n-M+1); % Reference noise input vector
% Gain vector
k = (P*u) / (lambda + u' * P * u);
% Filter output
y(n) = w' * u;
% Error
e(n) = x(n) - y(n);
% Weight update
w = w + k * e(n);
% Update inverse correlation matrix
P = (P - k * u' * P) / lambda;
end

4. Plot Results
figure;

subplot(3,1,1)
plot(d);
title('Desired Signal'); grid on;

subplot(3,1,2)
plot(x);
title('Noisy Signal (Desired + Noise)'); grid on;

subplot(3,1,3)
plot(e);
title('Recovered Signal after RLS Noise Cancellation'); grid on;
```

## RESULT

The RLS adaptive filter was implemented and the recovered signal after noise cancellation was obtained and plotted.



# 11. Upsampling of a Discrete-Time Sequence

## AIM

To upsample a discrete-time sequence by  $L$  times and plot the spectra of the original and upsampled sequences.

## APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; discrete-time sequence; FFT and upsampling functions; plotting facilities.

## THEORY / PROCEDURE

Generate the given sequence, insert  $L-1$  zeros between successive samples for  $L = 4$ , compute FFT-based spectra, and plot the original and upsampled sequences and their magnitude spectra.

## MATLAB PROGRAM

11. Upsampling the discrete time sequence by  $L$  times and plot the spectrum of both the given sequence and upsampled sequence.

```
clc; clear; close all;
Original sequence
n = 0:20;
x = sin(0.2*pi*n) + 0.5*cos(0.4*pi*n); % example sequence
Upsampling factor
L = 4;
Upsample the sequence
x_up = upsample(x, L);
FFT parameters
N = 1024; % for high-resolution spectrum
Frequency axis
w = linspace(-pi, pi, N);
Compute spectra
X = fftshift(fft(x, N));
Xup = fftshift(fft(x_up, N));
Plot time-domain sequences
figure;
subplot(2,1,1);
stem(n, x, 'filled');
title('Original Sequence x[n]');
xlabel('n'); ylabel('Amplitude');

subplot(2,1,2);
stem(0:length(x_up)-1, x_up, 'filled');
title(['Upsampled Sequence (L = ' num2str(L) ')']);
xlabel('n'); ylabel('Amplitude');

Plot magnitude spectra
figure;

subplot(2,1,1);
plot(w, abs(X), 'LineWidth', 1.5);
title('Spectrum |X(e^{j\omega})| of Original Sequence');
xlabel('\omega (rad/sample)');
ylabel('Magnitude');
grid on;

subplot(2,1,2);
plot(w, abs(Xup), 'LineWidth', 1.5);
title(['Spectrum |X_{up}(e^{j\omega})| after Upsampling by L = ' num2str(L)]);
xlabel('\omega (rad/sample)');
ylabel('Magnitude');
grid on;
```

## RESULT

The sequence was upsampled by  $L = 4$  and the original and upsampled time-domain sequences and magnitude spectra were plotted.

## 12. Downsampling of a Discrete-Time Sequence

### AIM

To downsample a discrete-time sequence by M times and plot the spectra of the original and downsampled sequences.

### APPARATUS / SOFTWARE REQUIRED

Computer with MATLAB; discrete-time sequence; downsampling operation; FFT and plotting facilities.

### THEORY / PROCEDURE

Perform the discrete-time resampling operation specified in the source document and plot the original and resulting sequences together with their spectra.

### MATLAB PROGRAM

12. Down sampling the discrete time sequence by M times and plot the spectrum of both the given sequence and down sampled sequence.

```
clc; clear; close all;
Original sequence
n = 0:20;
x = sin(0.2*pi*n) + 0.5*cos(0.4*pi*n); % example sequence
Upsampling factor
L = 4;
Upsample the sequence
x_up = upsample(x, L);
FFT parameters
N = 1024; % for high-resolution spectrum
Frequency axis
w = linspace(-pi, pi, N);
Compute spectra
X = fftshift(fft(x, N));
Xup = fftshift(fft(x_up, N));
Plot time-domain sequences
figure;
subplot(2,1,1);
stem(n, x, 'filled');
title('Original Sequence x[n]');
xlabel('n'); ylabel('Amplitude');

subplot(2,1,2);
stem(0:length(x_up)-1, x_up, 'filled');
title(['Upsampled Sequence (L = ' num2str(L) ')']);
xlabel('n'); ylabel('Amplitude');

Plot magnitude spectra
figure;

subplot(2,1,1);
plot(w, abs(X), 'LineWidth', 1.5);
title('Spectrum |X(e^{j\omega})| of Original Sequence');
xlabel('\omega (rad/sample)');
ylabel('Magnitude');
grid on;

subplot(2,1,2);
plot(w, abs(Xup), 'LineWidth', 1.5);
title(['Spectrum |X_{up}(e^{j\omega})| after Upsampling by L = ' num2str(L)]);
xlabel('\omega (rad/sample)');
ylabel('Magnitude');
grid on;
```

### RESULT

The downsampling operation and corresponding spectrum were obtained. The source document's final experiment code is labeled as downsampling but contains the same upsampling implementation as Experiment 11; this has been retained as supplied rather than silently corrected.