# ADSI Exam 2019

**Author:** Omar Ali Sheikh, au597954, 201800078

**Date:** 12th June 2019

**Table of Contents**



## Problem 1

```
clear variables;
```

**Problem 1**

For a 4th order all-zero lattice filter the transfer function from the input to the third stage is given by

$$A_3(z) = 1 - \frac{1}{2}z^{-1} - \frac{1}{16}z^{-2} + \frac{1}{2}z^{-3}$$

### 1) Determine the reflection coefficients

1. Determine the numerical values of $k_1$, $k_2$ and $k_3$.

The recursive algorithm for determining the reflection coefficients works as follows:

1. Compute $A_M(z) = \frac{H(z)}{h[0]}$
2. Compute $k_M = a_M$ where $a_M$ is the last coefficients of $A_M(z)$. For example, if $A_M(z) = 1 + 0.06z^{-1} - 0.42z^{-2} + 0.5z^{-3}$ then $k_M = 0.5$
3. Compute $B_M(z)$ by flipping the coefficients of $A_M(z)$
4. Set $m = M$
5. Compute $A_{m-1}(z) = \frac{1}{1-k_m^2}[A_m(z) - k_m B_m(z)]$. The algorithm fails if $k_m = 1 \rightarrow k_m^2 = 1$ because of division by zero.
6. Compute $k_{m-1} = a_{m-1}$ where $a_{m-1}$ is the last coefficients of $A_{m-1}(z)$
7. Compute $B_{m-1}(z)$ by flipping the coefficients of $A_{m-1}(z)$. Alternatively, compute $B_{m-1}(z) = z^{-m-1}A_{m-1}\left(\frac{1}{z}\right)$
8. Set $m = m - 1$
9. Go to step 5 if $m \neq 0$
10. We know that $A_0(z) = B_0(z) = 1$

Step 1:

$$A_3(z) = 1 - \frac{1}{2}z^{-1} - \frac{1}{16}z^2 + \frac{1}{2}z^{-3}$$

Step 2:

$$k_3 = \frac{1}{2}$$

Step 3:

$$B_3(z) = \frac{1}{2} - \frac{1}{16}z^{-1} - \frac{1}{2}z^{-2} + z^{-3}$$

Step 5:

$$A_2(z) = \frac{1}{1-\left(\frac{1}{2}\right)^2}\left[1 - \frac{1}{2}z^{-1} - \frac{1}{16}z^2 + \frac{1}{2}z^{-3} - \frac{1}{2}\left(\frac{1}{2} - \frac{1}{16}z^{-1} - \frac{1}{2}z^{-2} + z^{-3}\right)\right]$$

$$A_2(z) = \frac{1}{0.75}[0.75 - 0.4688z^{-1} + 0.1875z^2]$$

$$A_2(z) = 1 - 0.6251z^{-1} + 0.25z^2$$

Step 6:

$$k_2 = 0.25$$

Step 7:

$$B_2(z) = 0.25 - 0.6251z^{-1} + z^{-2}$$

Step 5:

$$A_{m-1}(z) = \frac{1}{1-k_m^2}[A_m(z) - k_m B_m(z)]$$

$$A_1(z) = \frac{1}{1-(0.25)^2}[1 - 0.6251z^{-1} + 0.25z^2 - 0.25(0.25 - 0.6251z^{-1} + z^{-2})]$$

$$= 1.0667[1 - 0.6251z^{-1} + 0.25z^2 - 0.0625 + 0.1563z^{-1} - 0.25z^{-2}]$$

$$= 1 - 0.5001z^{-1}$$

Step 6:

$k_1 = -0.5001$ (probably due to some rounding error)

Let us check the results in MATLAB:

```
b = [1, -1/2, -1/16, 1/2];
tf2latc(b)
```

```
ans = 3×1
   -0.5000
    0.2500
    0.5000
```

```
% Recheck the results
k = [-0.5, 0.25, 0.5];
lat2fir(k, 1)
```

```
ans = 1×4
    1.0000   -0.5000   -0.0625    0.5000
```

The filter coefficients are: $k_1 = -0.5, k_2 = 0.25, k_3 = 0.5$

## 2) Compute a symbolic expression for A4(z)

Assume that one more stage with reflection coefficient $k_4$ is added to the all-zero lattic filter

2. Compute the symbolic expression for $A_4(z)$.

If $m = 4$ then we have

$$A_3(z) = \frac{1}{1-k_4^2}[A_4(z) - k_4 B_4(z)]$$

We want to isolate $A_4$:

$$A_3(z) = \frac{A_4(z)}{1-k_4^2} - \frac{k_4 B_4(z)}{1-k_4^2}$$

$$A_3 + \frac{k_4 B_4(z)}{1-k_4^2} = \frac{A_4(z)}{1-k_4^2}$$

$$A_4(z) = (1-k_4^2)\left(A_3 + \frac{k_4 B_4(z)}{1-k_4^2}\right)$$

$$A_4(z) = \left((1-k_4^2)A_3 + k_4 B_4(z)\right)$$

We know that $A_3(z) = 1 - \frac{1}{2}z^{-1} - \frac{1}{16}z^2 + \frac{1}{2}z^{-3}$, our final expression is:

$$A_4(z) = \left((1-k_4^2)\left(1 - \frac{1}{2}z^{-1} - \frac{1}{16}z^2 + \frac{1}{2}z^{-3}\right) + k_4 B_4(z)\right)$$

## Problem 2

```
clear variables;
```

One often encountered building block in signal processing systems is the comb filter. In the FIR implementation, the comb filter is given by the following difference equation:

$$y[n] = x[n] + \alpha x[n - D],$$

where $|\alpha|<1$ and $D$ is a constant delay.

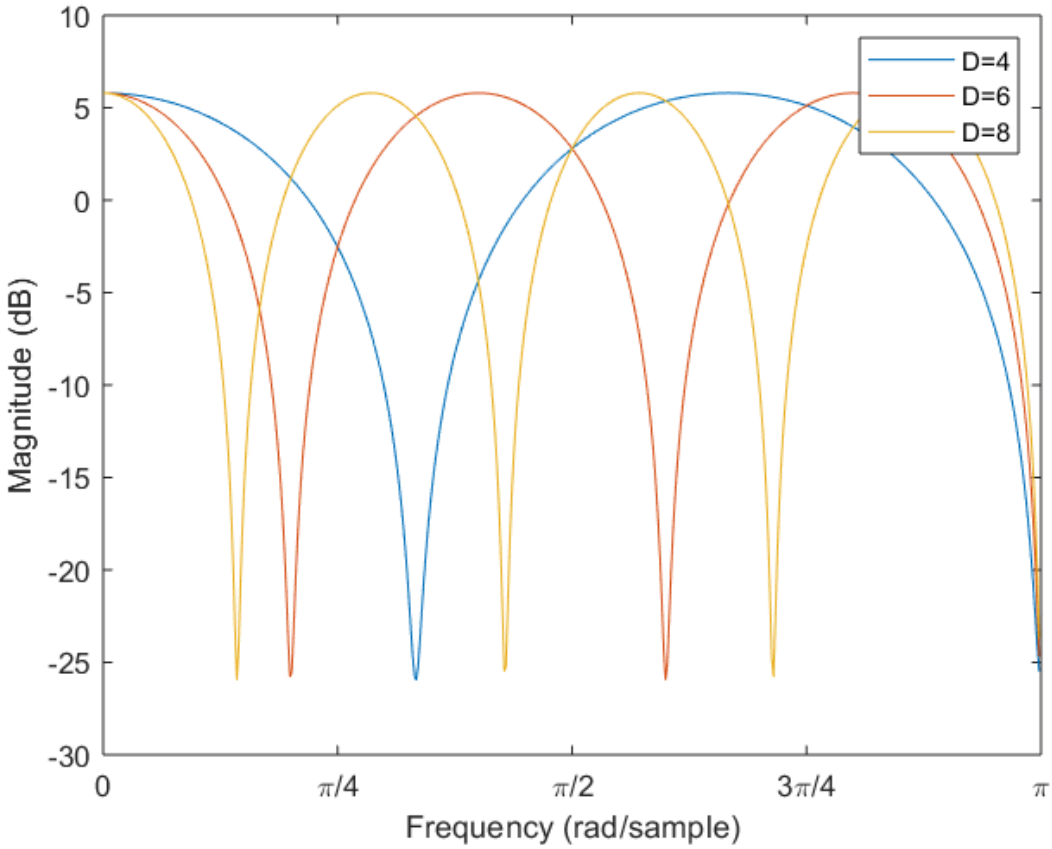
1. Compute the transfer function of the comb filter. Sketch the magnitude response of the comb filter for $\alpha = 0.95$ and $D = 4$, $D = 6$ and $D = 8$. Comment on the results.

The general transfer function for the comb filter is:

$$H(z) = \frac{Y(z)}{X(z)} = 1 + \alpha z^{-D}$$

The magnitude response for the comb filter:

```
alpha = 0.95;
b4 = [1, 0, 0, alpha];
b6 = [1, 0, 0, 0, 0, alpha];
b8 = [1, 0, 0, 0, 0, 0, 0, alpha];

[H, w] = freqz(b4, 1);
H4 = pow2db(H.*conj(H));
[H, w] = freqz(b6, 1);
H6 = pow2db(H.*conj(H));
[H, w] = freqz(b8, 1);
H8 = pow2db(H.*conj(H));

plot(w, H4, w, H6, w, H8);
legend('D=4', 'D=6', 'D=8')
set(gca,'XTick', 0:pi/4:pi)
set(gca,'XTickLabel', {'0','\pi/4','\pi/2','3\pi/4','\pi'})
xlabel('Frequency (rad/sample)')
ylabel('Magnitude (dB)')
xlim([0, pi]);
```




The comb filter attenuates certain frequency components of the input signal. It has multiple passbands and stopbands. The comb filter with D=4 attenuates 2 frequency components of the signal. The D=6, attenuates 3 components and finally the D=8 filter attenuates 4 frequency components.

## 2) Compute the autocorrelation function for the comb filter output

Assume that the comb filter, $y[n] = x[n] + \alpha x[n - D]$, is driven by white noise with unit variance.

2. Compute an analytic expression for the autocorrelation function of the comb filter output.

Given $x(n) \sim \mathrm{WN}(0, 1)$ we have $r_{\mathrm{xx}}(\ell) = \sigma_x^2 \delta(\ell) = 1 \cdot \delta(\ell)$

Also, the output of the filter is $y(n) = x(n) + \alpha x(n - D)$

$$
\begin{aligned}
r_{\mathrm{yy}}(\ell) &= E[y(n)y(n-\ell)] \\
&= E[(x(n) + \alpha x(n-D))(x(n-\ell) + \alpha x(n-\ell-D))] \\
&= E[x(n)x(n-\ell)] + E[x(n)\alpha x(n-\ell-D)] + E[\alpha x(n-D)x(n-\ell)] + E[\alpha x(n-D)\alpha x(n-\ell-D)] \\
&= r_{\mathrm{xx}}(\ell) + \alpha r_{\mathrm{xx}}(\ell - D) + \alpha r_{\mathrm{xx}}(\ell + D) + \alpha^2 r_{\mathrm{xx}}(\ell) \\
&= \delta(\ell) + \alpha\delta(\ell - D) + \alpha\delta(\ell + D) + \alpha^2\delta(\ell)
\end{aligned}
$$

## 3) Is the comb filter invertible?

3. Is the comb filter invertible?

A filter $H(z) = \dfrac{B(z)}{A(z)}$ is said to be stable and causal if all the poles of $H(z)$ are inside the unit circle.

If the inverse filter $H_{\mathrm{inv}}(z) = \dfrac{A(z)}{B(z)}$ has to be stable and causal, then all the poles of $H_{\mathrm{inv}}(z)$ must be inside the unit circle or equivalently all zeros of $H(z)$ must be inside the unit circle.

In practice, we say that a system is only invertible if its zeros and poles are inside the unit circle (minimum-phase system).

Finding zeros of the system functions, we observe that the zeros are inside the unit circle for $\alpha = 0.98$ but very close to being on the unit circle. However, if $\alpha = 1$ then the zeros will be on the unit circle which means that the filter will remove certain frequency components. The problem is that the inverse filter cannot reconstruct the original signal because of deleted information from the signal i.e., the missing frequency component cannot be recovered.

```
abs(roots(b4))
```

```
ans = 3×1
    0.9830
    0.9830
```

```
    0.9830
```

```
abs(roots(b6))
```

```
ans = 5×1
    0.9898
    0.9898
    0.9898
    0.9898
    0.9898
```

```
abs(roots(b8))
```

```
ans = 7×1
    0.9927
    0.9927
    0.9927
    0.9927
    0.9927
    0.9927
    0.9927
```

## Problem 3

```
clear variables;
```

Let the autocorrelation function of random signal be given by

$$r[l] = (-1)^{|l|} e^{-0.25|l|}.$$

```
p = 10;
ell = 0:p-1;
r_xx = ((-1).^abs(ell)).*exp(-0.25.*abs(ell))
```

```
r_xx = 1×10
    1.0000   -0.7788    0.6065   -0.4724    0.3679   -0.2865    0.2231   -0.1738 ···
```

### 1) Compute and plot the spectrum assuming an MA(2) model

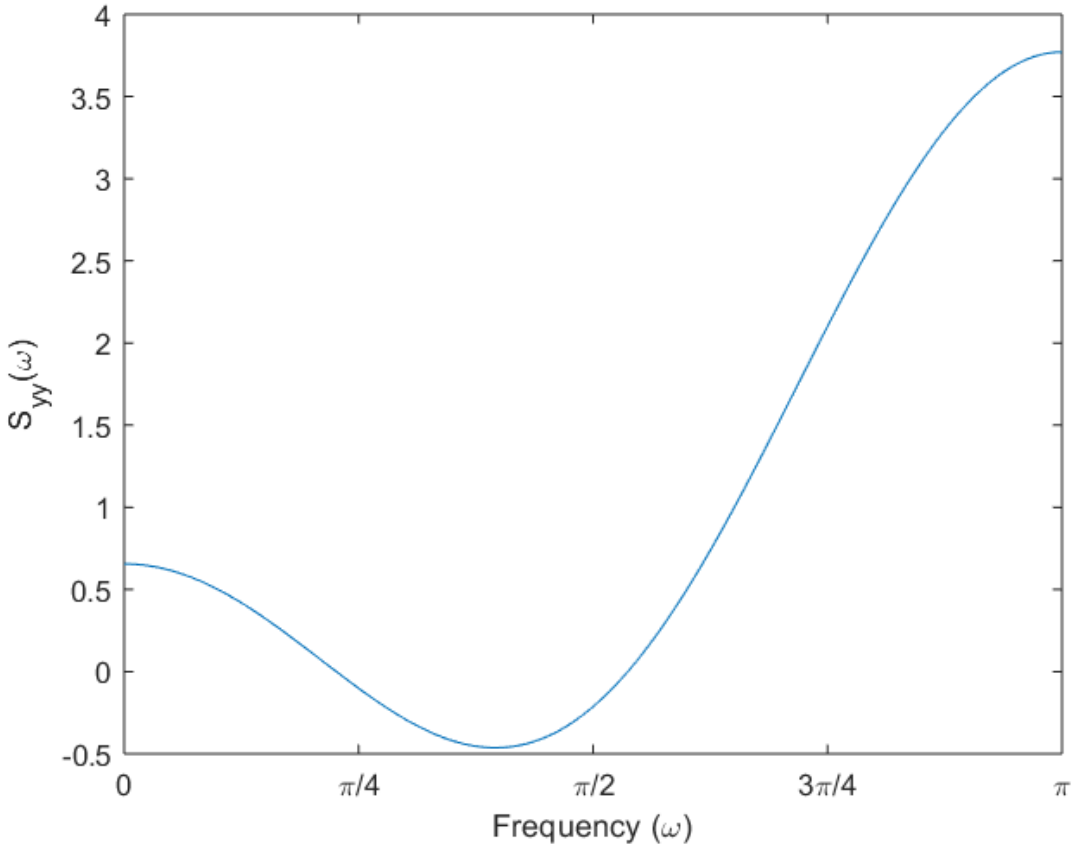
1. Compute and plot the spectrum assuming an MA(2) model.

The PSD for an MA(q) process can be computed using Eq. (13.119):

$$S_{xx}(\omega) = r_{xx}[0] + 2\sum_{\ell=1}^{\infty} r_{xx}[\ell]\cos\omega\ell, \qquad (13.119)$$

Compute and plot the PSD in MATLAB:

```
w=0:0.001:pi;

S = r_xx(1);
for l = 2:3
    S = S + 2 * r_xx(l)*cos(w*(l-1));
end

plot(w, S);
xlabel('Frequency (\omega)');
ylabel('S_{yy}(\omega)');
set(gca,'XTick',0:pi/4:pi)
set(gca,'XTickLabel',{'0', '\pi/4', '\pi/2', '3\pi/4', '\pi'})
xlim([0, pi])
```




## 2) Compute and plot the spectrum assuming Pisarenko model

2. Compute and plot the spectrum assuming the signal consists of a sinusoid embedded in white noise i.e. a Pisarenko model.

The Pisarenko method can be used to recover the sinusoidal frequencies of a corrupted signal $x(n)$ given two assumptions:

- The signal $x(n)$ consists of $p$ sinusoids that has been corrupted by white noise.
- The autocorrelation matrix of size $(p+1)\times(p+1)$ is known or can be estimated from data

The Pisarenko method consists of the following steps:

***Step 1***: Compute the autocorrelation matrix $\mathbf{R}_{xx}$

***Step 2:*** Find the eigenvector corresponding to the smallest minimum eigenvalue. The elements of this eigenvector is the parameters of the $\mathrm{ARMA}(2p, 2p)$ model

***Step 3:*** Find the frequencies $\{f_i\}$ of the sinusoids. This can be done by computing the roots of the polynomial $A(z)$ in Eq. (14.5.4) in the book. This polynomial has $2p$ poles on the unit circle which correspond to the frequencies of the system.

$$A(z) = 1 + \sum_{m=1}^{2p} a_m z^{-m} \qquad (14.5.4)$$

***Step 4***: Solve Eq. (14.5.11) for the signal powers $\{P_i\}$

$$\begin{bmatrix} \cos 2\pi f_1 & \cos 2\pi f_2 & \cdots & \cos 2\pi f_p \\ \cos 4\pi f_1 & \cos 4\pi f_2 & \cdots & \cos 4\pi f_p \\ \vdots & \vdots & & \vdots \\ \cos 2\pi p f_1 & \cos 2\pi p f_2 & \cdots & \cos 2\pi p f_p \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_p \end{bmatrix} = \begin{bmatrix} \gamma_{yy}(1) \\ \gamma_{yy}(2) \\ \vdots \\ \gamma_{yy}(p) \end{bmatrix} \qquad (14.5.11)$$

where

- $\gamma_{yy}(1), \gamma_{yy}(2), \cdots, \gamma_{yy}(p)$ are the estimated autocorrelation values
- $P_i = \dfrac{A_i^2}{2}$ is the average power of the $i$th sinusoid and $A_i$ is the corresponding amplitude

***Step 5***: Estimate the amplitude $A_i = \sqrt{2P_i}$

These steps are coded up in the `pisarenko()` function (see at end of this document).

```
[F, A, P, lambda_min] = pisarenko(r_xx, 1)
```

```
F = 0.4393
A = 1.2955
P = 0.8391
lambda_min = 0.1609
```

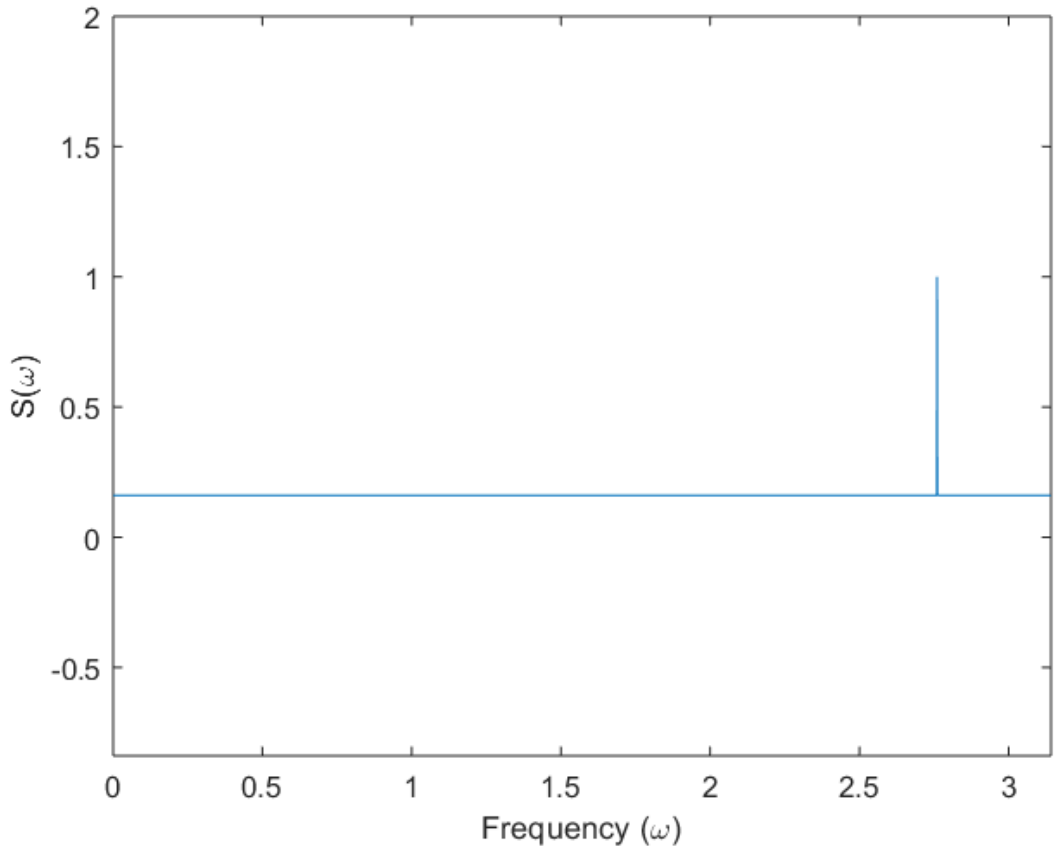
We can describe the signal as:

$$x(n) = 1.2955\cos(2\pi \cdot 0.4393n + \phi) + w(n)$$

where $w(n)$ is white noise with variance $\sigma_w^2 = 0.1609$

A sketch of the PSD in MATLAB:

```
D = 3;
w = 0:10^-D:pi;
delta = @(n) round(n, D) == 0;
S = lambda_min*ones(1, numel(w)) + P*delta(w - 2*pi*F);
plot(w, S)
xlim([0, pi])
ylim([lambda_min-1, lambda_min+P+1])
xlabel('Frequency (\omega)')
ylabel('S(\omega)')
```



Since we have a formal description of the sinusoid, we can compute the power spectral density by finding the autocorrelation function of the sinusoid.

In ADSI Problem 4.4, we found that the autocorrelation of a real sinusoid given by $y(n) = A\cos(\omega n + \phi)$ where $A$ and $\omega$ are real constants and $\phi$ is a random variable with $\phi \sim U(0, 2\pi)$ is:

$$r_{yy}(\ell) = \frac{A^2}{2}\cos(\omega\ell)$$

The autocorrelation function of white noise with variance $\sigma_w^2$ is given by:

$$r_{ww}(\ell) = \sigma_w^2\delta(\ell)$$

Combining these results, the general autocorrelation function of our signal is:

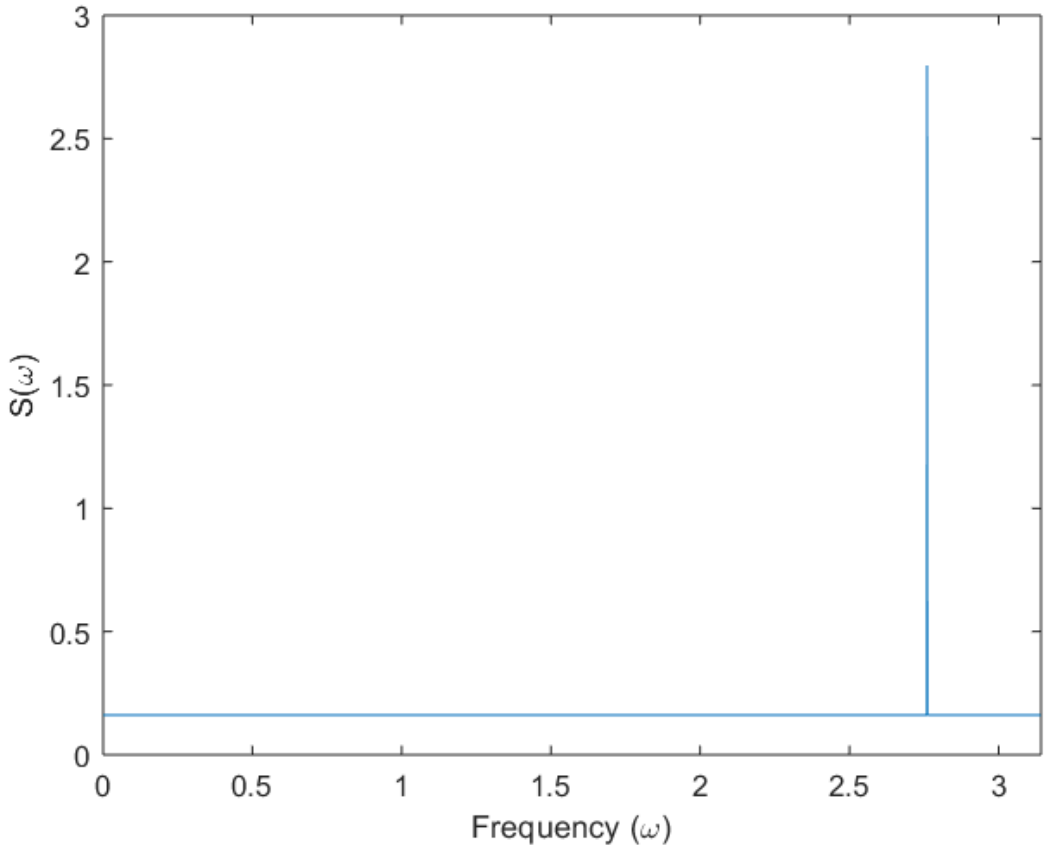
$$r_{xx}(\ell) = \frac{A^2}{2}\cos(\omega\ell) + \sigma_w^2\delta(\ell)$$

The power spectral density is the Fourier transform the autocorrelation function which is given by:

$$S(\omega) = \frac{A^2}{2}\pi[\delta(\omega - \omega_0) + \delta(\omega + \omega_0)] + \sigma_w^2$$

We can plot the PSD:

```
D = 3;
w = 0:10^(-D):pi;
delta = @(n) round(n, D) == 0;
S = (A^2/2)*pi * (delta(w-(F*2*pi)) + delta(w+(F*2*pi))) + lambda_min;
plot(w, S);
xlabel('Frequency (\omega)');
ylabel('S(\omega)');
xlim([0, pi])
```




### 3) Comment on the appropriateness of the two models

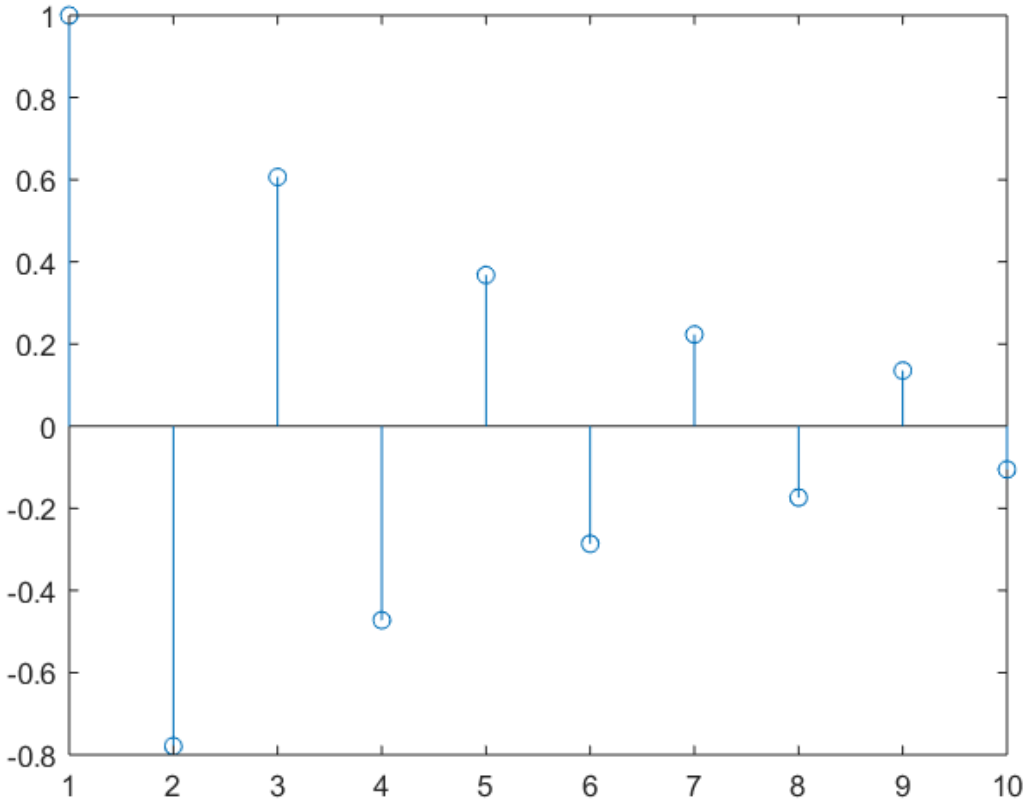
3. Comment on the appropriateness of the above two models.

Plotting the ACRS, we observe that it oscillates which indicates a high frequency signal.

The MA(2) model is basically a FIR filter with two coefficents and white noise as input where the Pisarenko model assumes that the signal is sinusoidal.

Due to the oscillating nature of the ACRS, it is highly unlikely that the random signal is generated by a FIR filter. Therefore, the Pisarenko model is more appropriate in this model.

```
stem(r_xx)
```




## Problem 4

```
clear variables;
```

```
p = 3;
ell = 0:p-1;
r_vv = [3, 1, 0];
```

### 1) Calculate the signal to noise ratio in x(n)

The signal to noise ratio of the input signal is given by

$$\mathrm{SNR}_i = \frac{\text{power of signal}}{\text{power of noise}} = \frac{r_s(0)}{r_v(0)}$$

First, we need to find an analytical expression for the autocorrelation of $s(n)$.

In ADSI Problem 4.4.1, we know that the autocorrelation of the complex sinusoid $z(n) = A\,e^{j(\omega n+\phi)}$ where $A$ and $\omega$ are real constants and $\phi \sim U(0, 2\pi)$ is given by:

$$r_{zz}(\ell) = A^2\,e^{j\omega\ell}$$

Since the result from 4.4.1) uses Euler, we need to convert the signal to complex exponential.

We use the relation $\cos(\theta) = \frac{1}{2}(e^{j\theta} + e^{-j\theta})$

$$x(n) = A\cos(\omega\, n + \phi)$$

$$x(n) = \frac{A}{2}\,(e^{j(\omega n+\phi)} + e^{-j(\omega n+\phi)})$$

$$x(n) = \frac{A}{2}e^{j(\omega n+\phi)} + \frac{A}{2}e^{-j(\omega n+\phi)}$$

Therefore, the autocorrelation of the real signal is:

$$r_{xx}(\ell) = \left(\frac{A}{2}\right)^2 e^{j\omega\ell} + \left(\frac{A}{2}\right)^2 e^{-j\omega\ell}$$

$$r_{xx}(\ell) = \frac{A^2}{4}e^{j\omega\ell} + \frac{A^2}{4}e^{-j\omega\ell}$$

$$r_{xx}(\ell) = \frac{A^2}{4}(e^{j\omega\ell} + e^{-j\omega\ell})$$

$$r_{xx}(\ell) = \frac{A^2}{2}\frac{1}{2}(e^{j\omega\ell} + e^{-j\omega\ell})$$

Using the relation $\cos(\theta) = \frac{1}{2}(e^{j\theta} + e^{-j\theta})$ we can rewrite the autocorrelation to:

$$r_{xx}(\ell) = \frac{A^2}{2}\cos(\omega\ell)$$

Thus, the autocorrelation function of a real signal $d(n) = A\cos(\omega n + \phi)$ where $A$ and $\omega$ are real constants and $\phi \sim U(0, 2\pi)$ is:

$$r_{dd}(\ell) = \frac{A^2}{2}\cos(\omega\ell)$$

In this problem, the autocorrelation of $s(n)$ is:

$$r_{ss}(\ell) = 2\cos\left(\frac{1}{4}\ell\right)$$

```
r_ss = 2.*cos(1/4.*ell)
```

```
r_ss = 1×3
    2.0000    1.9378    1.7552
```

Now, we can calculate the SNR:

```
SNR_i = r_ss(1) / r_vv(1)
```

```
SNR_i = 0.6667
```

## 2) Solve the Wiener-Hopf equation

The $p$-order Wiener filter for estimating the signal $s(n)$ is given by:

$$\hat{y}[n] = \sum_{k=1}^{p} h_k x[n+1-k] \qquad (14.112)$$

The optimum Wiener filter coefficients is given by the Wiener-Hopf equation:

$$\boldsymbol{h}_{\mathrm{o}} = \boldsymbol{R}_x^{-1}\boldsymbol{g}, \qquad (14.109)$$

where $\boldsymbol{R}_x$ is the autocorrelation matrix of the corrupted signal $s(n)$ and $\boldsymbol{g}$ is the cross-correlation between the desired signal $s(n)$ and the corrupted signal $x(n)$.

Designing a Wiener filter to recover a corrupted signal involves 3 steps:

1. Compute the autocorrelation sequence $r_x(\ell)$ and matrix $R_x$
2. Compute the cross-correlation $r_{\mathrm{sx}}(\ell)$
3. Solve the Wiener-Hopf equation to find the optimum Wiener filter coefficients

**Step 1:** Compute $r_{\mathrm{xx}}(\ell)$

Since the signal $s(n)$ and the noise $v(n)$ are uncorrelated the autocorrelation function of $s(n)$ is just the sum of the individual autocorrelation functions:

$$\begin{aligned} r_{\mathrm{xx}}(\ell) &= E[x(n)x(n-\ell)] \\ &= E[(s(n)+v(n))(s(n-\ell)+v(n-\ell))] \\ &= E[s(n)s(n-\ell)+s(n)v(n-\ell)+v(n)s(n-\ell)+v(n)v(n-\ell)] \\ &= r_{\mathrm{ss}}(\ell)+r_{\mathrm{sv}}(\ell)+r_{\mathrm{sv}}(\ell)+r_{\mathrm{vv}}(\ell) \\ &= r_{\mathrm{ss}}(\ell)+r_{\mathrm{vv}}(\ell) \end{aligned}$$

```
r_xx = r_ss + r_vv
```

```
r_xx = 1×3
    5.0000    2.9378    1.7552
```

```
R_xx = toeplitz(r_xx)
```

```
R_xx = 3×3
    5.0000    2.9378    1.7552
    2.9378    5.0000    2.9378
    1.7552    2.9378    5.0000
```

**Step 2:** Compute the cross-correlation $r_{sx}(\ell)$:

Since the desired signal is $y(n) = s(n)$ is uncorrelated with the noise $v(n)$, the cross-correlation between $s(n)$ and $v(n)$ simplifies to the autocorrelation of the signal $r_s(\ell)$.

$$
\begin{aligned}
r_{yx}(\ell) &= E[y(n)x(n-\ell)] \\
&= E[s(n)(s(n-\ell)+v(n-\ell))] \\
&= E[s(n)s(n-\ell)] + E[s(n)v(n-\ell)] \\
&= r_{ss}(\ell) + r_{sv}(\ell) \\
&= r_{ss}(\ell) + 0 \quad \text{(since } s(n) \text{ and } v(n) \text{ are uncorrelated } r_{sv}(\ell) = 0
\end{aligned}
$$

```
g = r_ss'
```

```
g = 3×1
    2.0000
    1.9378
    1.7552
```

**Step 3:** Compute the optimum filter coefficients:

```
h_opt = R_xx\g
```

```
h_opt = 3×1
    0.2615
    0.1246
    0.1860
```

## 3) Compute the increase in MSE using another filter

The minimum value of the mean square error $E[e^2(n)] = E\left[(y(n) - \hat{y}(n))^2\right]$ is given by

$$J_o = r_y[0] - \boldsymbol{h}_o^T \boldsymbol{g} = r_y[0] - \sum_{k=0}^{p-1} h_o[k] r_{yx}[k]. \qquad (14.115)$$

Let us compute it for the optimum 3-tap Wiener filter:

```
mse_opt = r_ss(1) - h_opt'*g
```

```
mse_opt = 0.9090
```

And the for other filter:

```
h_nonopt = [2, -1, 1]';
mse_nonopt = r_ss(1) - h_nonopt'*g
```

```
mse_nonopt = -1.8173
```

The difference is:

```
mse_opt - mse_nonopt
```

```
ans = 2.7264
```

## Problem 5

```
clear variables;
```

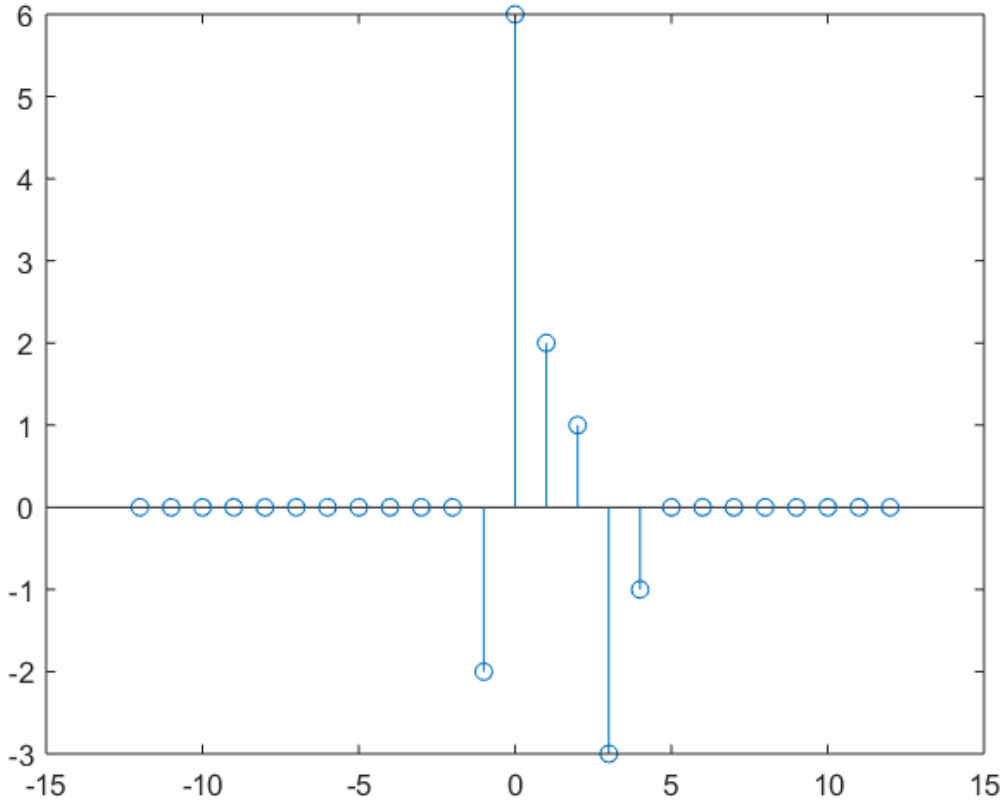
```
N = 3;
n = -N:N;
x = zeros(numel(n), 1);
g = zeros(numel(n), 1);
zero_point = ceil(numel(x)/2);
x(zero_point) = 2;
x(zero_point+1) = -1;

g(zero_point-1) = -1;
g(zero_point) = 3;
g(zero_point+1) = 1;
```

### 1) Upsample the signal x(n) by a factor of 3 and sketch the result

We can use MATLAB function to upsample the signal $x(n)$ by a factor of 3 using the given interpolation kernel $g(n)$:

```
y = upfirdn(x, g, 3);
mid_y = ceil(numel(y)/2)-1;
n_y = -mid_y:mid_y;
```

```
stem(n_y, y)
```




## 2) Upsample the signal x(n) by a factor of 3 and compute the interpolated values

We use Eq. (12.33) to compute $x_I[1]$ and $x_I[2]$ where $I = 3$ is upsampling factor.

$$x_{\mathrm{I}}[n] = \sum_{m=-\infty}^{\infty} x[m]g_{\mathrm{r}}[n - mI], \qquad (12.33)$$

$$x_I[1] = \sum_{m=0}^{1} x[m]g[n - 3m] = x[0]g[1] + x[1]g[-2] = x[0]g[1] = 2 \cdot 1 = 2$$

$$x_I[2] = \sum_{m=0}^{1} x[m]g[n - 3m] = x[0]g[2] + x[1]g[-1] = 0 + (-1)(-1) = 1$$

### 3) Discuss the performance of the interpolation kernel

The interpolation kernel seems to be poor because the interpolated signal does not look like the original signal. When interpolating, it can be expected that the kernel inserts $I-1$ samples between consecutive samples of $x(n)$. The given kernel adds modifies the signal.

## MATLAB Functions

```
function res = test()
    res = 1
end

function [k,G] = fir2lat(h)
% Converts FIR filter coefficients to lattice coefficients.
% From Figure 9.24 (p. 514)
    G = h(1);
    a = h/G;
    M = length(h)-1;
    k(M) = a(M+1);
    for m = M:-1:2
        b = fliplr(a);
        a = (a-k(m)*b)/(1-k(m)^2); a = a(1:m);
        k(m-1) = a(m);
    end
end


function [h] = lat2fir(k, G)
% Converts lattice coefficients to FIR filter coefficients.
% From Figure 9.25 (p. 515)
    a = 1;
    b = 1;
    M = length(k);
    for m = 1:1:M
        a = [a,0]+k(m)*[0,b];
        b = fliplr(a);
    end
    h = G*a;
end

function [F, A, P, lambda_min]=pisarenko(r_xx, p)
    % Estimates the frequencies and amplitudes using the Pisarenko method
    %  r_xx: autocorrelation sequence starting from zero.
    %        The length of ACRS must be at least 2p+1.
```

```
%  p: assumed number of sinusoids in the signal
%  F: normalised frequencies of the sinuoids
%  A: amplitudes of the sinuoids
if numel(r_xx) < 2*p+1
    error(strcat('The length of ACRS must be at least ', int2str(2*p+1)));
end

% Compute the autocorrelation matrix
R_xx = toeplitz(r_xx(1:2*p+1));

% Perform the eigendecomposition
[eigvecs, eigvals] = eigs(R_xx, size(R_xx, 1), 'smallestreal');

eigvals = diag(eigvals);
lambda_min = eigvals(1);

% Find the eigenvector 'a' corresponding to the smallest eigenvalue.
% The function eigs() sorts eigenvectors, so just pick the first column.
a = eigvecs(:, 1);

% Ensure that a_0 = 1 (this is by definition)
a = a / a(1);

% The elements of this eigenvector corresponds to the parameters
% of an ARMA(2p, 2p) model: a_0, a_1, ..., a_2p where p: number of sinusoids
% The polynomial A(z) in (14.5.4) has 2p poles on the unit circle.
% Obtain the poles by finding the roots of the system.
z = roots(a);

% Estimate frequencies
F = zeros(p, 1);
for i = 1:p
    % The poles come in pairs. Each pair is complex conjugate
    % of one another. Only use one of them and find the absolute value.
    z_i = z(2*i);
    F(i) = abs(angle(z_i)) / (2*pi);
end

% Build the matrix of cosines
C = zeros(p);
for i = 1:p
    for j = 1:p
        C(i, j) = cos(i * 2*pi * F(j));
    end
end

% Start the ACRS from the second element according to equation (14.5.11)
gamma = r_xx(2:p+1);

% Solve equation (14.5.11) for P
P = C\gamma; % Same as `inv(C)*gamma` but faster and more accurate

% Since P=A^2/2, we can compute the amplitude A=sqrt(2*P)
```

```
    A = sqrt(2 * P);
end
```