Homework 7

Table of Contents

[✓] Problem 14.42: Compare periodogram, modified periodogram and Blackman-Tukey methods in terms of signal resolution

Consider the harmonic process signal model

$$x[n] = \cos(0.44\pi n + \phi_1) + \cos(0.46\pi + \phi_2) + v[n], \quad 0 \le n \le 256$$

where ϕ_1 and ϕ_2 are IID random variables uniformly distributed over $[-\pi, \pi]$ and $v[n] \sim \text{WGN}(0, 2)$.

So from the signal model, we know from that there are two peaks and constant noise floor at 2. How do we know this? We can compute it:

The autocorrelation of real sinusoid $z(n) = A\cos(\omega_0 n + \phi)$ is $r_{zz}(\ell) = \frac{A^2}{2}\cos(\omega_0 \ell)$.

Since the PSD is the Fourier transform of the autocorrelation, we get:

$$S_{zz}(\omega) = \frac{A^2}{4}\delta(\omega + \omega_0) + \frac{A^2}{4}\delta(\omega - \omega_0)$$

This means that power is concentrated at $\pm\omega_0$. In the PSD, we will therefore see two peaks $\pm\omega_0$

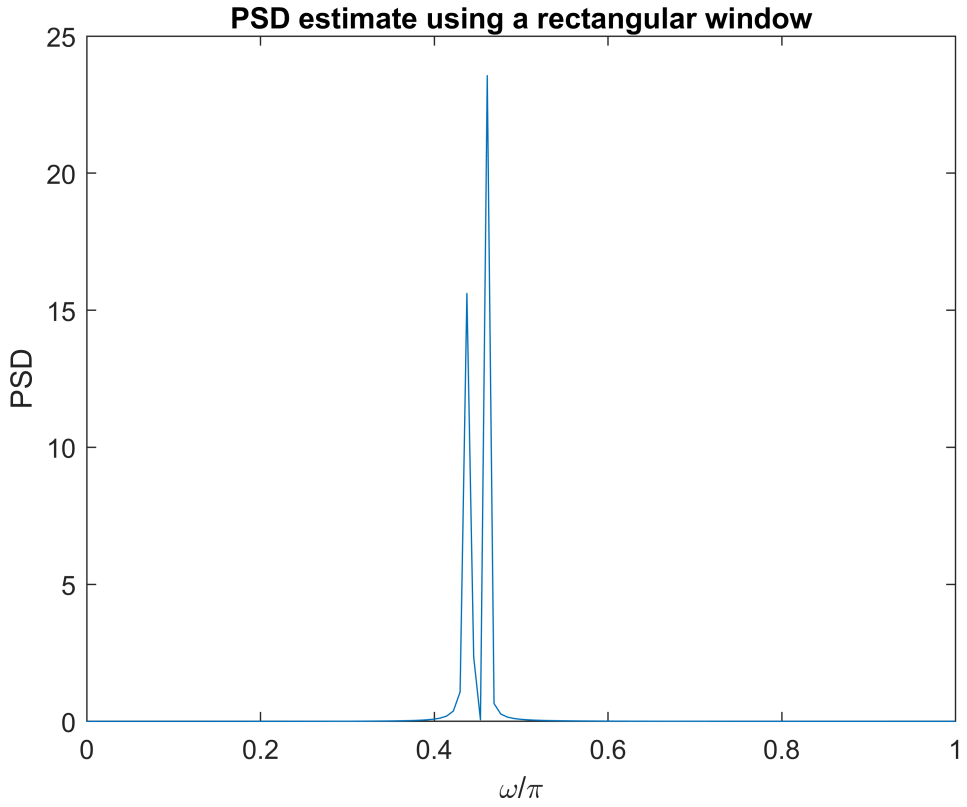
The autocorrelation of white Gaussian noise is $r_{vv}(\ell) = \sigma_v^2\delta(\ell) = 2\delta(\ell)$. The PSD of Gaussian noise is constant around σ_v^2.

So the true PSD should be something like:

$$S_{xx}(\omega) = \frac{1}{4}\delta(\omega - 0.44\pi) + \frac{1}{4}\delta(\omega - 0.46\pi) + 2$$

1) Estimate the PSD using the periodogram and plot the spectrum.

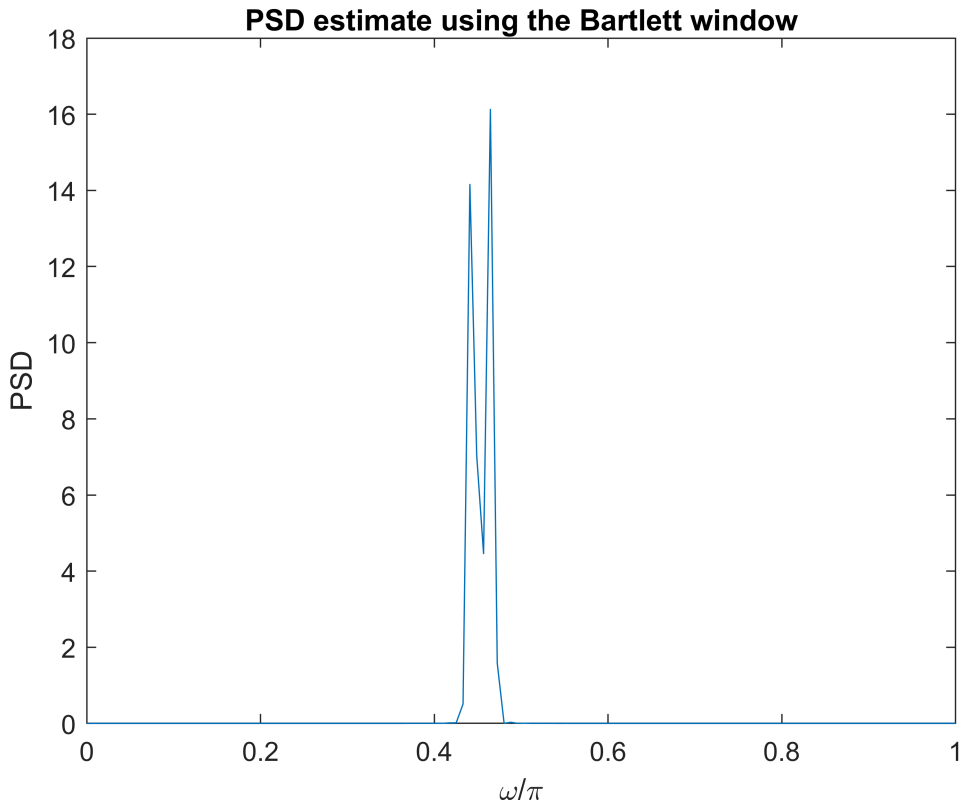
```
N = 256;
n = 0:N-1;
v = randn(1,N) * sqrt(2);
phi = 2*pi*(rand(1,2)-0.5);
x = cos(0.44*pi*n + phi(1)) + cos(0.46*pi*n + phi(2)) + v;
%plot(n, x); xlim([0, N]);

[I, w] = periodogram(x, [], N);
IdB = pow2db(I);
plot(w/pi, I);
xlabel('\omega/\pi')
ylabel('PSD')
title('PSD estimate using a rectangular window')
```



2) Estimate the PSD using the modified periodogram with Bartlett window

Estimate the PSD using the modified periodogram with Bartlett window and plot the spectrum.

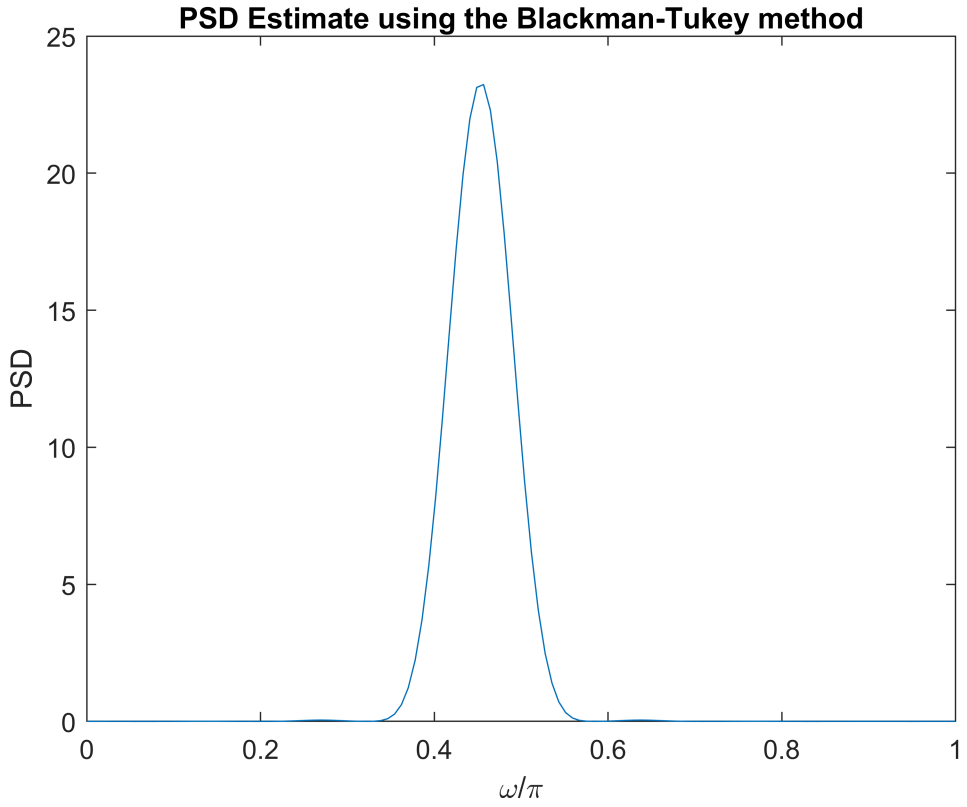
```
L = N;
I = periodogram(x, bartlett(L));
w = linspace(0,1,N/2)*pi;
plot(w/pi, I(1:N/2));
xlabel('\omega/\pi')
ylabel('PSD')
title('PSD estimate using the Bartlett window')
```



3) Estimate the PSD using the Blackman–Tukey method

Estimate the PSD using the Blackman–Tukey method with Parzen window and *L*=32 and plot the spectrum.

```
L = 32;
K = N;
I = psdbt(x,L,K);
plot(w/pi, I(1:N/2));
xlabel('\omega/\pi')
ylabel('PSD')
title('PSD Estimate using the Blackman-Tukey method')
```



4) Which method performs best in terms of signal resolution?

The PSD estimate using the periodogram (with rectangular window) shows two distinct peaks at the appropriate frequencies. However, the picture is not completely clear because the variance is high. Using the Bartlett (triangular) window yields similar characteristics. It is difficult to say whether the modified periodogram is better or worse. Theory tells us that the rectangular window yields the smallest main lope-width.

Blackman-Tukey performs some smoothing in order to minimise the variance of the spectrum. (The BT method attempts to supress the high frequencies.) The price is that the BT method **smears out** the two frequency peaks so it looks like one peak. Since in cases like this problem where we have two sinusoids, it will be impossible to distinguish between them.

In terms of signal resolution, the periodogram is clearly better to separate the two frequencies from each other. In general, when the frequencies are close to each other then we are forced to use the periodogram.

LESSON: if you know something about the system under study then we can get better solution.

[✓] Problem 14.57:

This problem is basically the same as 14.42. The only difference is that we will work with data from the real world.

This problem uses the signal file f16.mat that contains noise recorded at the copilot's seat of an F-16 airplane using a 16 bit A/D converter with $F_S = 19.98$ kHz.

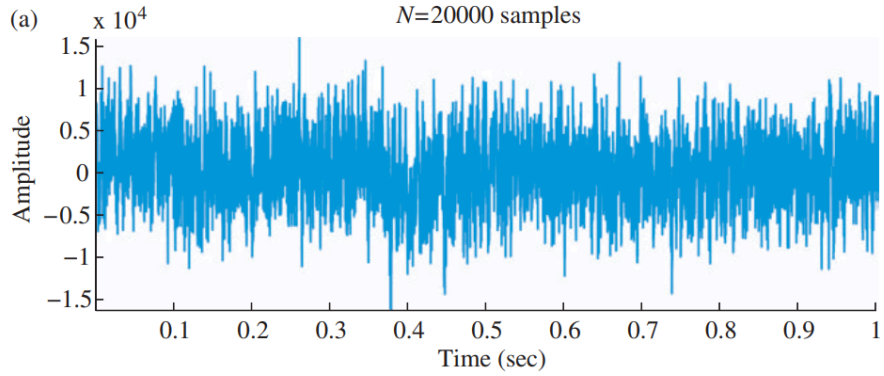


Figure (a) shows a waveform of F-16 noise recorded at the co-pilot's seat with a sampling rate of 19.98 kHz using a 16 bit ADC.

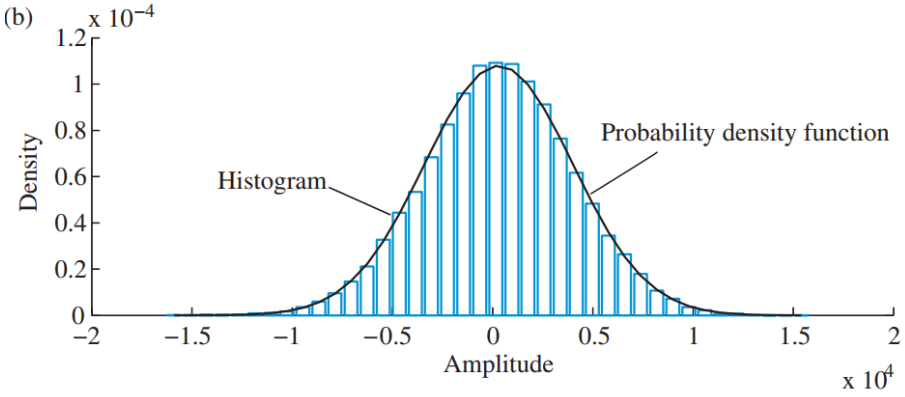
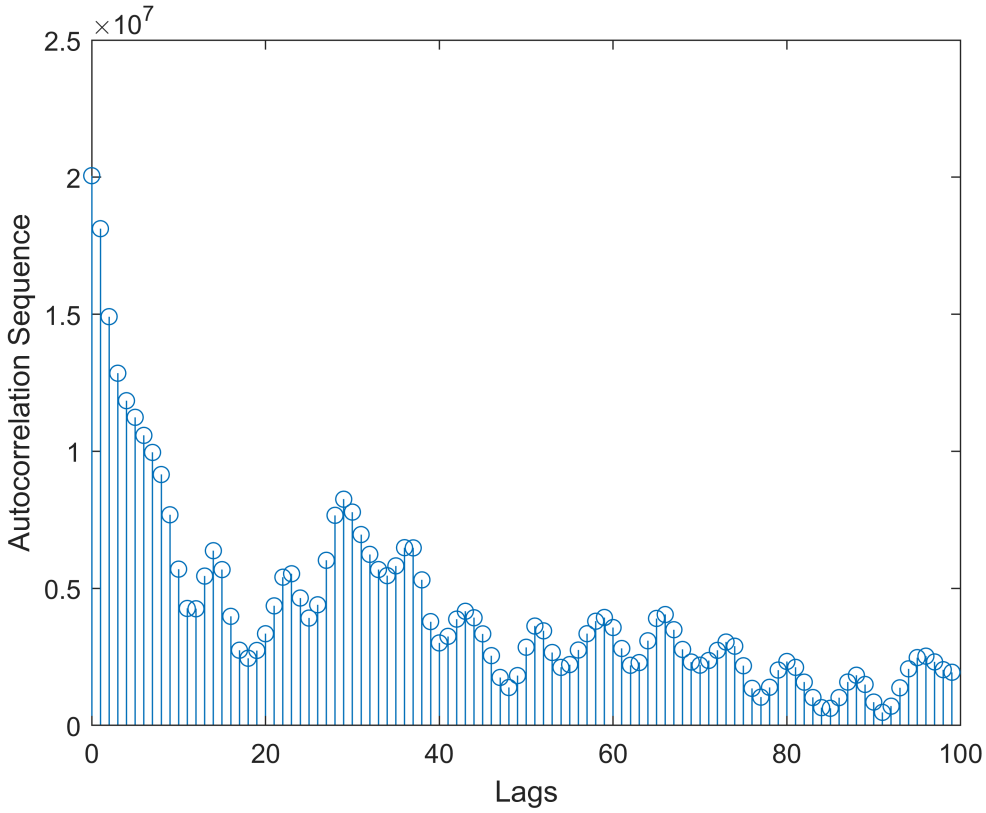


Figure (b) shows the histogram and theoretical probability density function of the F-16 noise.

We want to analyze this signal in terms of its ACRS and its spectral characteristics.

1) Compute and plot the ACRS estimate of the noise process.

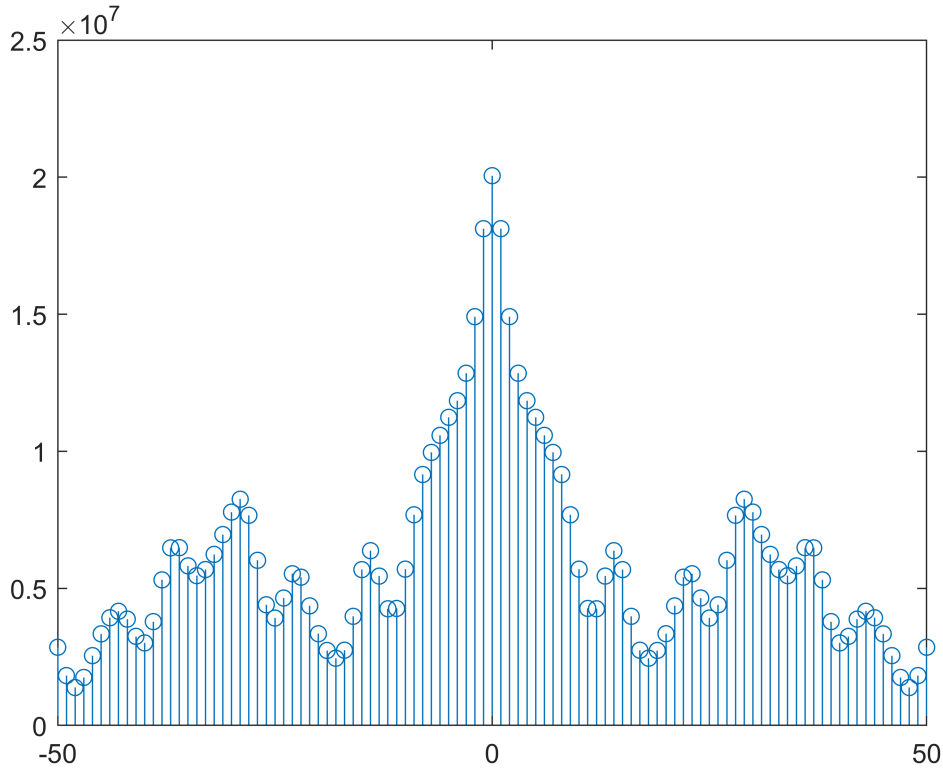
```
load('f16.mat')
y = f16;
L = 100;
r_yy = acrsfft(y, L); % MATLAB function from the book
stem(0:L-1, r_yy)
xlabel('Lags')
ylabel('Autocorrelation Sequence')
```



How should we interpret the ACRS?

- From the autocorrelation sequence, we see that there is some structure in the signal. The sequence is not decaying smoothly from lag zero to 100.
- The ACRS does not change sign so it seems to be some kind is low-frequency signal
- In order for the ACRS to go from lags zero to 100, it must be some higher-order process AR(p) model. It is definitely not a MA(q) process model.

```
% Let us verify it with MATLAB's xcorr function
[lags, r_yy] = xcorr(y, 50, 'biased');
stem(r_yy, lags)
```



2) Estimate model parameters for an *AR(2) and AR(4)* models.

An AR(p) model is given by:

$$y(n) = -\sum_{k=1}^{p} [a_k\, y(n-k)] + b_0 x(n)$$

The autocorrelation of AR(q) model was derived in Eq. 13.141 as:

$$r_{yy}(\ell) = -\sum_{k=1}^{p} a_k r_{yy}(\ell - k), \quad \ell > 0$$

This is useful because we can use it to find the coefficients $\{a_k\}$ of an AR(q) model using the autocorrelation $r_{yy}(\ell)$ computed numerically in MATLAB. This means that the expression in Eq. 13.141 becomes a set of p linear equations.

An AR(2) model is given by:

$$y(n) = -(a_1\, y(n-1) + a_2\, y(n-2)) + b_0 x(n)$$

We can estimate the model parameters of a second-order AR model $p = 2$ by creating two equations with two unknowns:

$$r_{yy}(1) = -a_1 r_{yy}(0) - a_2 r_{yy}(-1)$$

$$r_{yy}(2) = -a_1 r_{yy}(1) - a_2 r_{yy}(0)$$

We can write it into matrix form using the Toeplitz matrix:

$$\begin{bmatrix} r_{yy}(0) & r_{yy}(-1) \\ r_{yy}(1) & r_{yy}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = - \begin{bmatrix} r_{yy}(1) \\ r_{yy}(2) \end{bmatrix}$$

In the general case, it becomes:

$$\begin{bmatrix} r_{yy}[0] & r_{yy}[1] & \dots & r_{yy}[p-1] \\ r_{yy}[1] & r_{yy}[0] & \dots & r_{yy}[p-2] \\ \vdots & \vdots & \ddots & \vdots \\ r_{yy}[p-1] & r_{yy}[p-2] & \dots & r_{yy}[0] \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = - \begin{bmatrix} r_{yy}[1] \\ r_{yy}[2] \\ \vdots \\ r_{yy}[p] \end{bmatrix},$$

More compactly written as:

$$\boldsymbol{R}_y \boldsymbol{a} = -\boldsymbol{r}_y$$

This is systems of linear equations which can be solved in MATLAB.

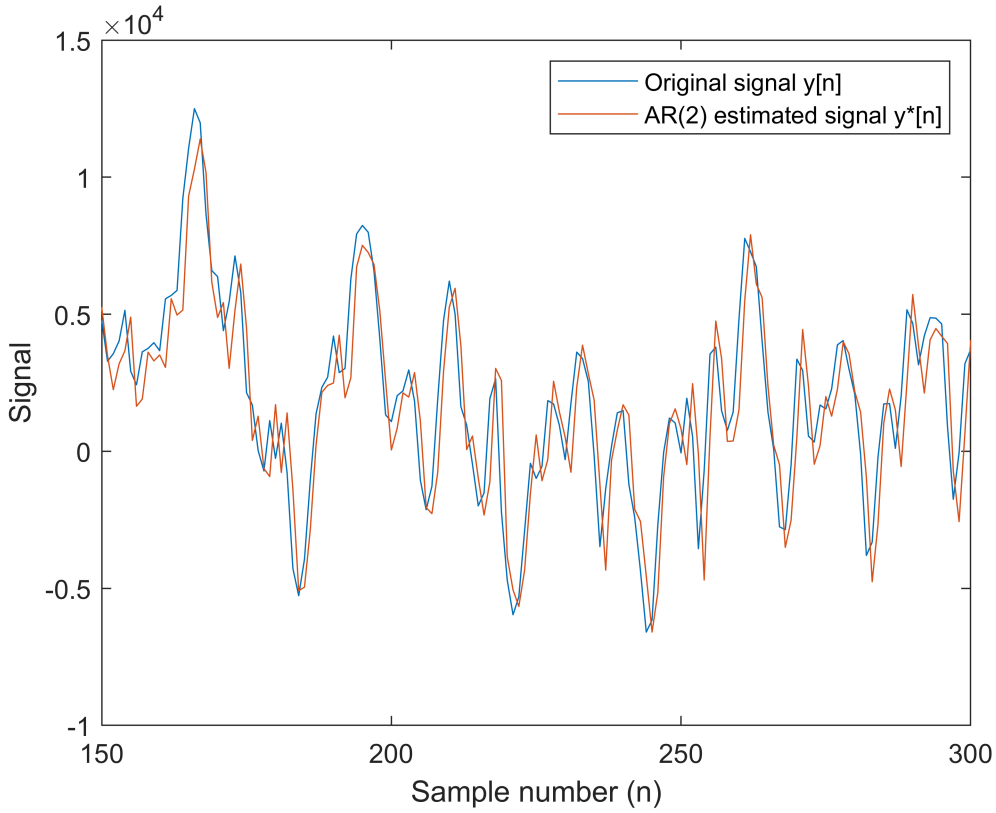
```
p = 2; % Model order
[r_yy, lags] = xcorr(y, p, 'biased');

% Select elements r_yy[0] to r_yy[p-1]
R_elems = r_yy(p+1:2*p);

% Create the Toeplitz matrix
R = toeplitz(R_elems);

% Select elements r_yy[1] to r_yy[p]
r = r_yy(p+2:2*p+1);

% Solve systems of linear equations using mldivide function
a = mldivide(R, -r)
```

```
a = 2×1
   -1.2628
    0.3975
```

Once we have found the coefficients, we can predict the signal using the previous two samples:

$$y(n) = -(-1.2628\, y(n-1) + 0.3975 y(n-2)) + b_0 x(n)$$

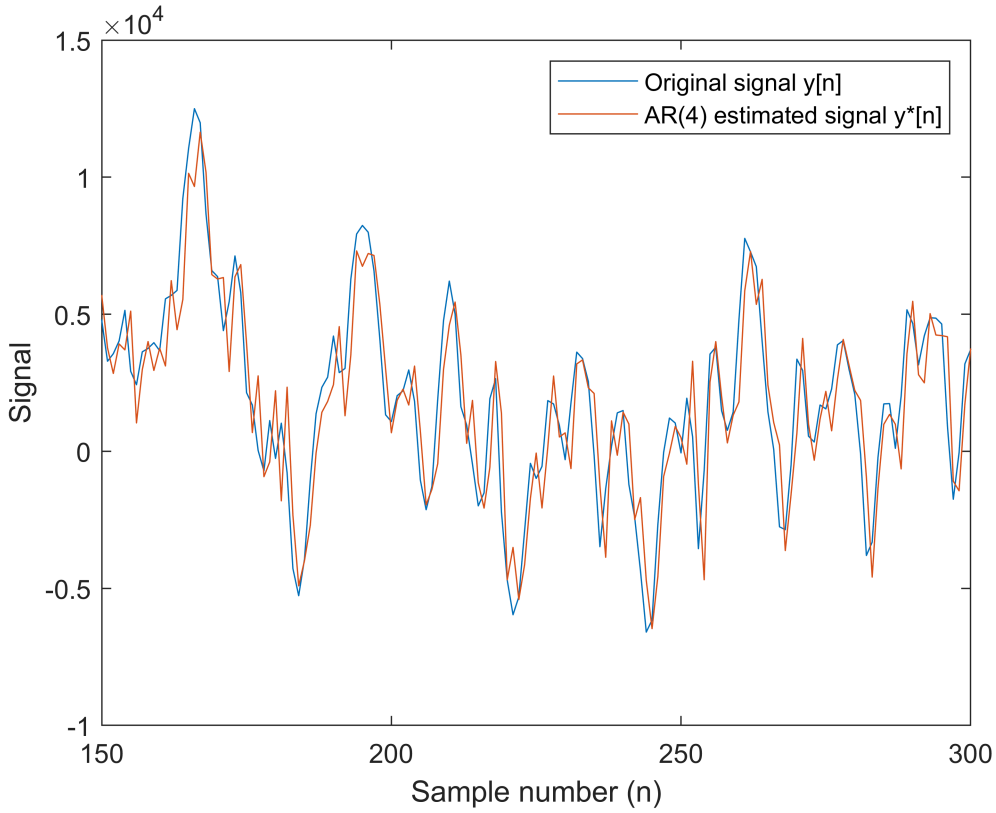
```
N = 1000;
y_hat = zeros(N);
n = 3:N; % Trick to make it work in MATLAB
y_hat(n) = -(a(1)*y(n-1) + a(2)*y(n-2));

% Plotting code
n = 1:N;
plot(n, y(n), n, y_hat(n))
legend('Original signal y[n]', 'AR(2) estimated signal y*[n]');
xlabel('Sample number (n)')
ylabel('Signal')
xlim([150, 300]) % Zoom
```




```
[a, v] = arfit(y, 4)
```

```
a = 4×1
   -1.4473
    0.9584
   -0.4942
    0.0705
v = 2.6038e+06
```

```
N = 1000;
y_hat = zeros(N);
n = 5:N; % Trick to make it work in MATLAB
y_hat(n) = -(a(1)*y(n-1) + a(2)*y(n-2) + a(3)*y(n-3) + a(4)*y(n-4));


% Plotting code
n = 1:N;
plot(n, y(n), n, y_hat(n))
legend('Original signal y[n]', 'AR(4) estimated signal y*[n]');
xlabel('Sample number (n)')
ylabel('Signal')
xlim([150, 300]) % Zoom
```



3) Compute and plot the PSD estimate using the AR(2) and AR(4) models.

The power spectrum of an ARMA(p, q) process is given by:

$$S_{yy}(\omega) = \sigma_x^2 \left| H(\mathrm{e}^{\mathrm{j}\omega}) \right|^2 = \sigma_x^2 \left| \frac{\sum_{k=0}^{q} b_k \mathrm{e}^{-\mathrm{j}\omega k}}{1 + \sum_{k=1}^{p} a_k \mathrm{e}^{-\mathrm{j}\omega k}} \right|^2 . \qquad (13.133)$$

The power spectrum of an AR(p) process is given by:

$$S_{\mathrm{yy}}(\omega) = \sigma_x^2 \left| \frac{1}{1 + \sum_{k=1}^{p} a_k e^{-j\omega k}} \right|^2$$

The algorithm is as follows:

1. Find the coefficents $\{a_1, a_2, \cdots, a_p\}$ for the $\mathrm{AR}(p)$ model
2. Compute the transfer for the $\mathrm{AR}(p)$ by computing the sum and finding its reciprocal
3. Compute the conjugate of the transfer function: $|H(e^{j\omega})|^2$
4. Multiply it with the variance σ_x^2

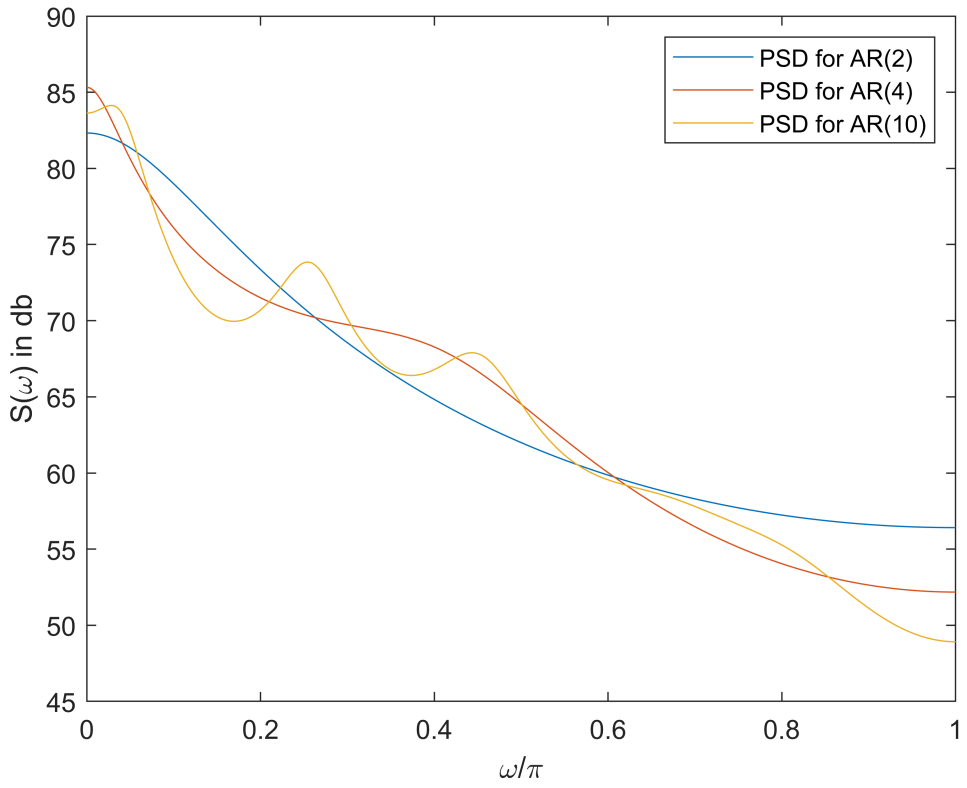
The algorithm is implemented in the functions `arfit()` and `ar2psd()` functions:

```
N = 256;
```

```
[a2, v2] = arfit(y, 2);
[S2, w] = ar2psd(a2, v2, N);

[a4, v4] = arfit(y, 4);
[S4, w] = ar2psd(a4, v4, N);

[a10, v10] = arfit(y, 10);
[S10, w] = ar2psd(a10, v10, N);

w_over_pi = w/pi;
S2db = pow2db(S2); % Use real(S2) to get rid of warning
S4db = pow2db(S4);
S10db = pow2db(S10);

plot(w_over_pi, S2db, w_over_pi, S4db, w_over_pi, S10db)
```

```
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

```
legend('PSD for AR(2)','PSD for AR(4)', 'PSD for AR(10)')
xlabel('\omega/\pi')
ylabel('S(\omega) in db')
```

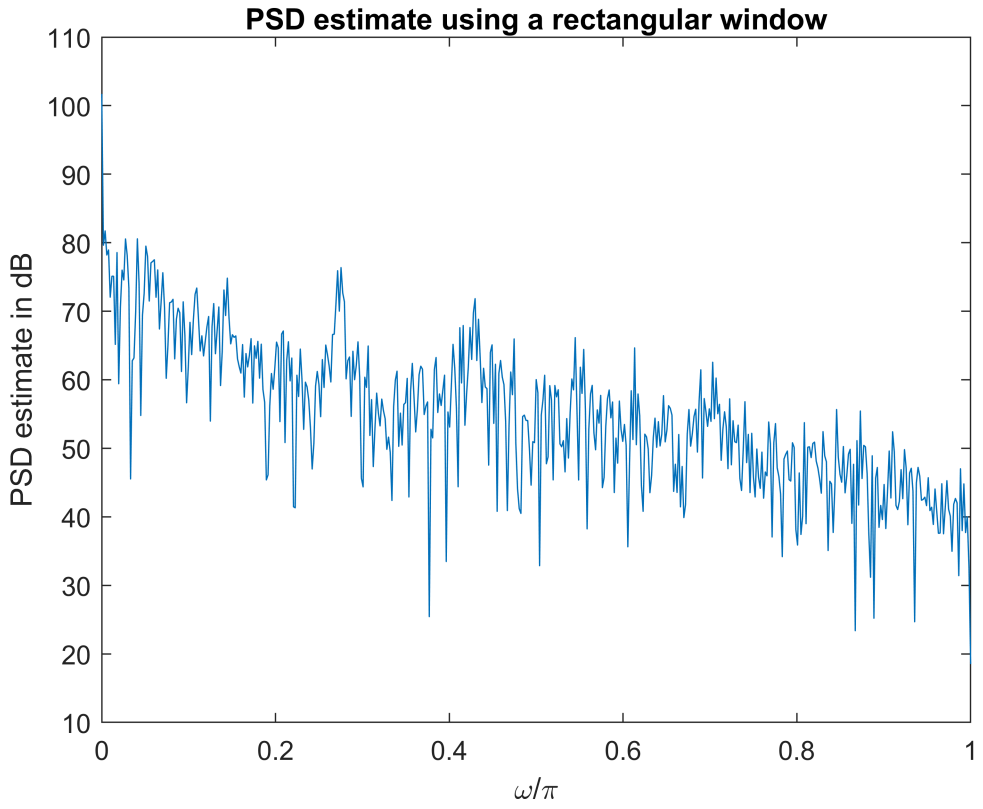


From this plot, we see that the majority of the energy is located at low frequencies. Since an AR(2) has only two model parameters, it has the smoothest spectrum. When we increase the number of model parameters, we see more structure and finer details of the spectrum. Notice that we get some peaks at frequencies $0.25\frac{\omega}{\pi}$ and $0.45\frac{\omega}{\pi}$.

4) Compute and plot the periodogram PSD estimate of the noise process.

We can compute a PSD estimate using the built-in `periodogram()` function in MATLAB:

```
N = 1024;
[I, w] = periodogram(y, [], N);
plot(w/pi, pow2db(I));
xlabel('\omega/\pi')
ylabel('PSD estimate in dB')
title('PSD estimate using a rectangular window')
```



Again, we observe the similar picture than we fitted an AR(p) models and computed the corresponding PSD estimates.

Now, we get some peaks at frequencies $0.27\frac{\omega}{\pi}$ and $0.42\frac{\omega}{\pi}$. Note how the periodogram has a large variance.

5) Compute the Bartlett PSD estimate

Compute the Bartlett PSD estimate using L = 32, 64, and 128. Plot your result of the averaged estimate.

To Bartlett PSD estimate, we use the modified periodogram with the triangular window. In this problem, we are asked to segment the signal in smaller parts of sizes $L = 32, L = 64$ and $L = 128$, compute the estimates for each segment and average them together to get the PSD estimate. This is basically the Welch method without any overlap.

How do we compute the Bartlett PSD estimate? The periodogram() function in MATLAB complains

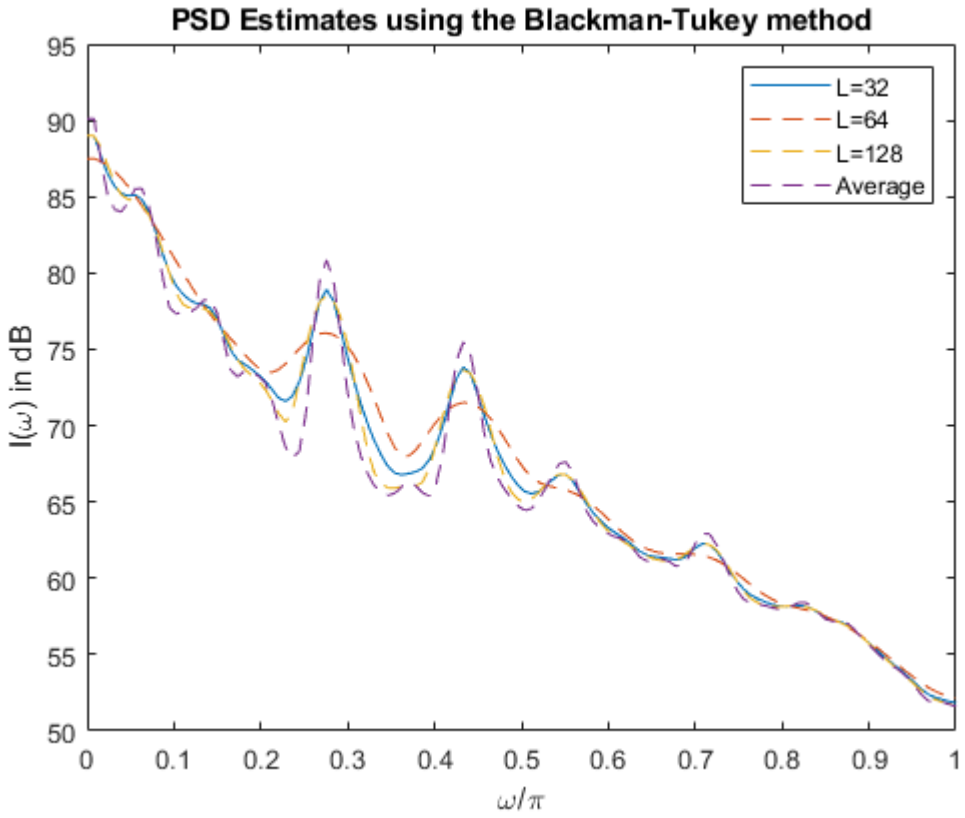
```
N = 1024;
% This results in an error:
% "The WINDOW must be a vector of the same length as the signal"
% Can we just pad the window signal with zeros?
%[I_L32, w] = periodogram(y, bartlett(32), N);
%[I_L64, w] = periodogram(y, bartlett(64), N);
%[I_L128, w] = periodogram(y, bartlett(128), N);
%plot(w/pi, pow2db(I_L32));
```

LESSON: This is just Welch method without overlapping.

6) Compute the Blackman–Tukey PSD estimate

Compute the Blackman-Tukey PSD estimate using L = 32, 64, and 128 using the Bartlett lag window. Plot your result of the averaged estimate.

```
load('f16.mat'); y = f16;

N = 256;
w = linspace(0, 1, N/2);
I_L32 = psdbt(y, 32, N);
I_L64 = psdbt(y, 64, N);
I_L128 = psdbt(y, 128, N);

% Concatenate along the second dimension
I_con = cat(2, I_L32, I_L64, I_L128);

% Compute the mean along the second dimension
I_mean = mean(I_con, 2);

plot(w, pow2db(I_mean), w, pow2db(I_L32), '--', w, pow2db(I_L64), '--', w, pow2db(I_L128), '--');
legend('L=32', 'L=64', 'L=128', 'Average')
title('PSD Estimates using the Blackman-Tukey method')
xlabel('\omega/\pi')
ylabel('I(\omega) in dB')
```

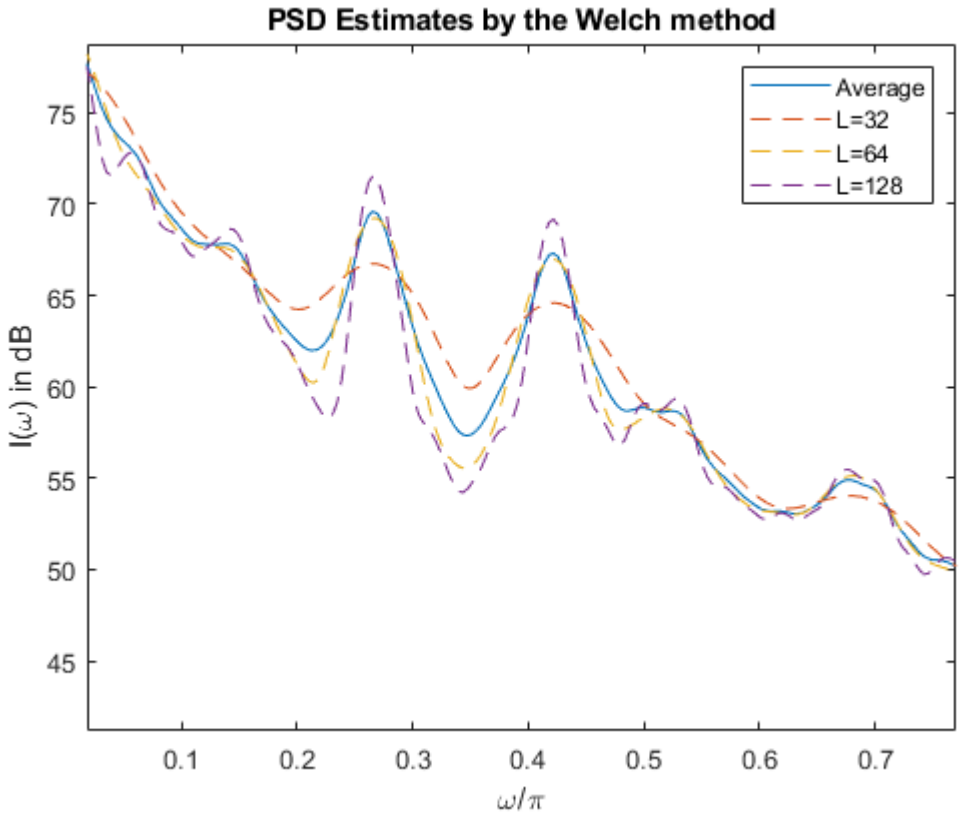


7) Compute the Welch PSD estimate

Compute the Welch PSD estimate using 50% overlap, Hamming window, and L = 32, 64, and 128. Plot your result of the averaged estimate.

```
N = 256;
N_fft = 512;
overlap_pct = 0.5;
w = linspace(0, 1, N+1);

% Speed up computation by using a subset of the data
y_reduced = y(1:N*10);

I_L32 = pwelch(y_reduced, hamming(32), overlap_pct*32, N_fft);
I_L64 = pwelch(y_reduced, hamming(64), overlap_pct*64, N_fft);
I_L128 = pwelch(y_reduced, hamming(128), overlap_pct*128, N_fft);

% Concatenate along the second dimension
I_concat = cat(2, I_L32, I_L64, I_L128);

% Compute the mean along the second dimension
I_mean = mean(I_concat, 2);

plot(w, pow2db(I_mean), w, pow2db(I_L32), '--', w, pow2db(I_L64), '--', w, pow2db(I_L128), '--');

legend('Average', 'L=32', 'L=64', 'L=128')
```

```
title('PSD Estimates by the Welch method')
xlabel('\omega/\pi')
ylabel('I(\omega) in dB')
```



8) Compare the plots in the above four parts and comment on your observation.

[✓] ADSI Problem 4.21: Minimum variance spectral estimation

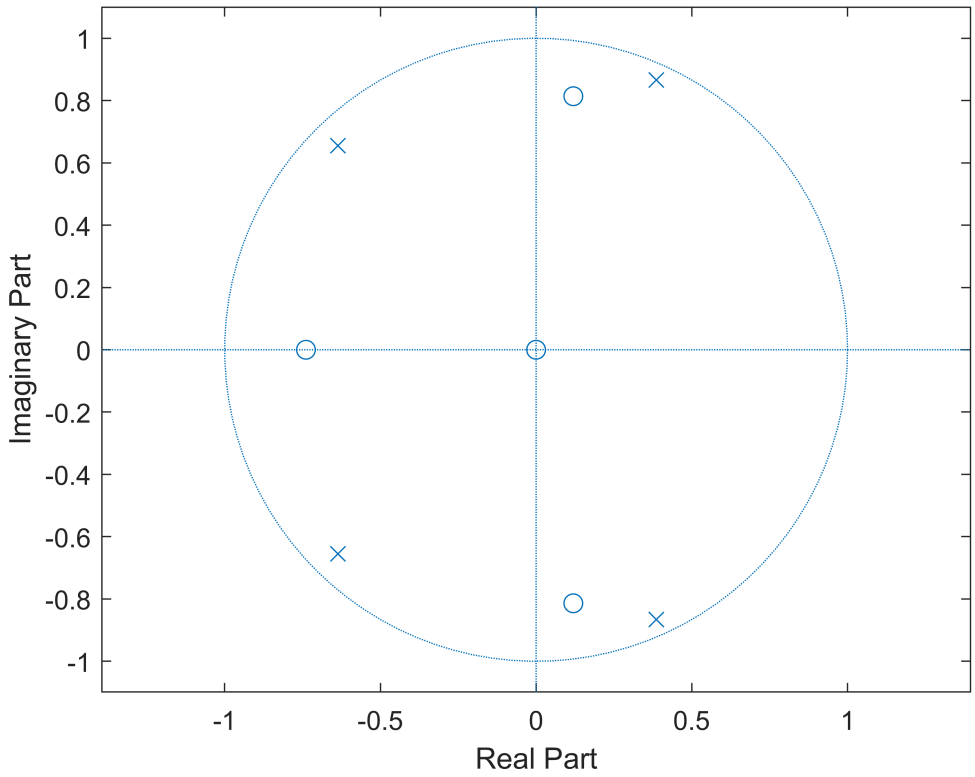
The discussion of the improved filter bank method taught us to choose filters, that for a given center frequency reduced the influence of spectral leakage by selectively moving the sidelobes around to attenuate specific frequency regions of high spectral density. This problem takes a closer look at this interpretation.

1) Create a stable signal model

Create a signal model with a suitable behaviour i.e. a couple of peaks in the spectrum. It is unimportant whether the model is AR, MA or ARMA as long as the model is stable.

Once we have created a signal, we can check whether it is stable by ensuring that all the poles and zeros are within the unit circle.

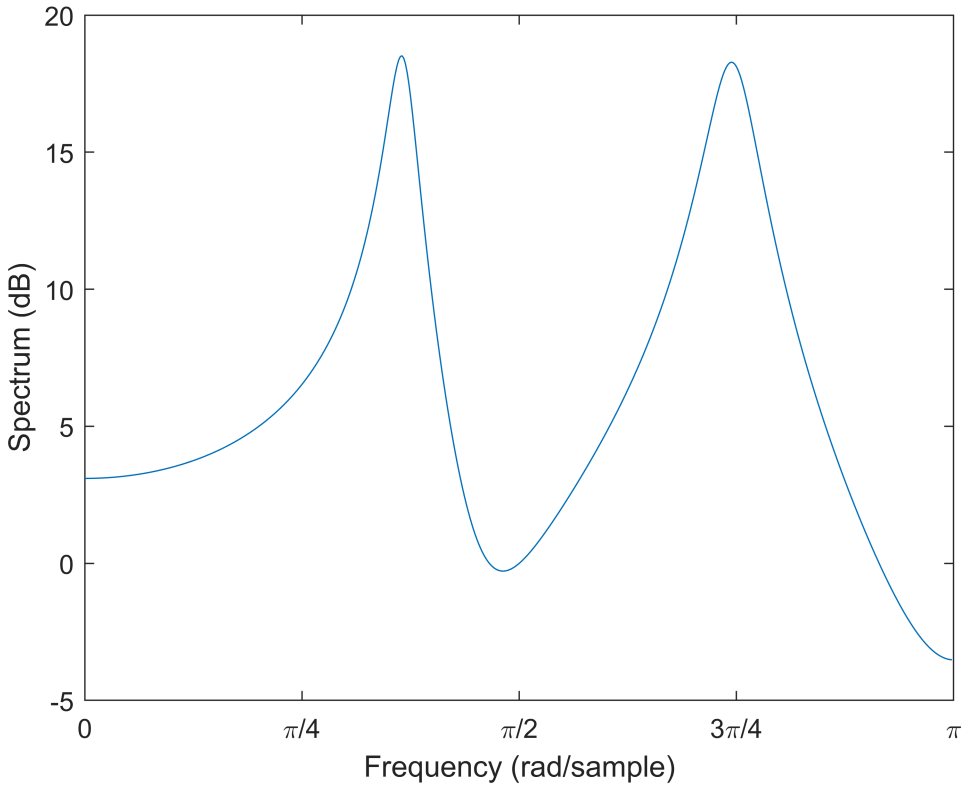
```
b = [2, 1, 1, 1];
a = [1, 0.5, 0.75, 0.5, 0.75];
zplane(b,a)
```



2) Use the signal model to plot the true spectrum.

We can plot the true spectrum by plotting the magnitude response of the transfer function or simply use the freqz function in MATLAB:

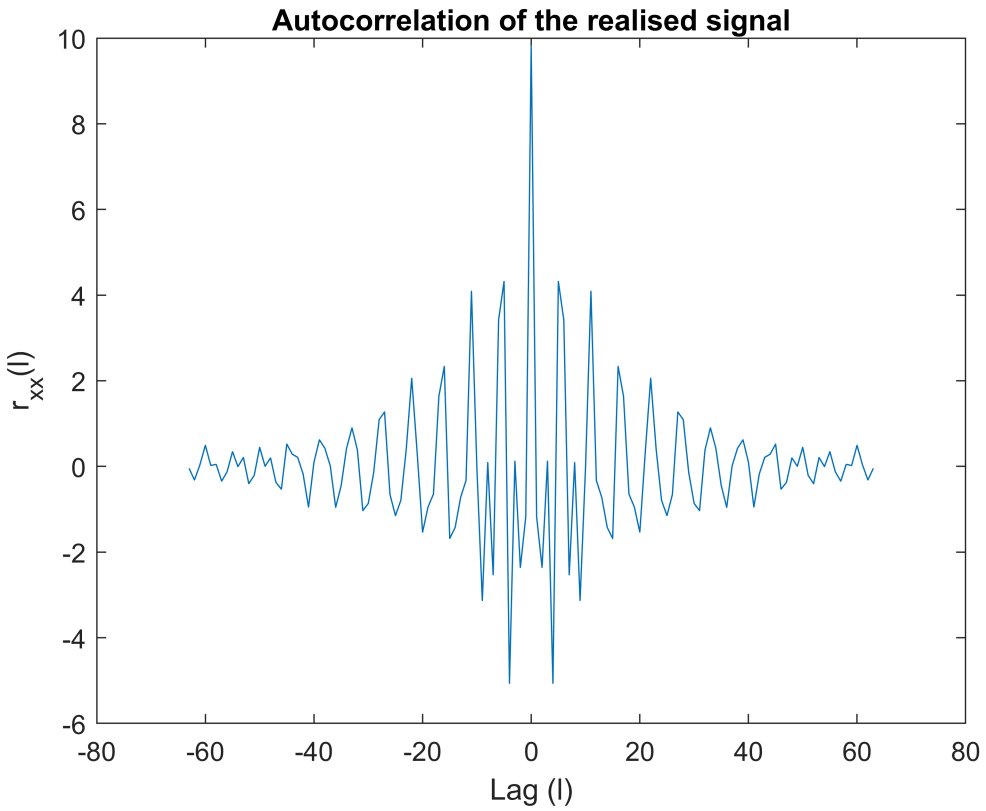
```
[H,w] = freqz(b, a);
plot(w, pow2db(H.*conj(H)));
set(gca,'XTick', 0:pi/4:pi)
set(gca,'XTickLabel', {'0','\pi/4','\pi/2','3\pi/4','\pi'})
xlabel('Frequency (rad/sample)')
ylabel('Spectrum (dB)')
xlim([0, pi]);
```



3) Create a realization of the signal and determine the autocorrelation matrix *Rx* in an appropriate size.

We can create a realisation of a signal using the filter function in MATLAB:

```
N = 4096;
n = randn(N,1);
x = filter(b, a, n);
N_Rx = 64; % size of autocorrelation matrix
[r_xx, lags] = xcorr(x, N_Rx-1, 'biased');

% Plot the autocorrelation for fun
plot(lags, r_xx);
xlabel('Lag (l)')
ylabel('r_{xx}(l)')
title('Autocorrelation of the realised signal')
```




```
% Create the autocorrelation matrix
R_xx = toeplitz(r_xx(N_Rx:end))
```

```
R_xx = 64×64
    9.8277   -1.1763   -2.3584    0.1223   -5.0645    4.3196    3.4248   -2.5309 ···
   -1.1763    9.8277   -1.1763   -2.3584    0.1223   -5.0645    4.3196    3.4248
   -2.3584   -1.1763    9.8277   -1.1763   -2.3584    0.1223   -5.0645    4.3196
    0.1223   -2.3584   -1.1763    9.8277   -1.1763   -2.3584    0.1223   -5.0645
   -5.0645    0.1223   -2.3584   -1.1763    9.8277   -1.1763   -2.3584    0.1223
    4.3196   -5.0645    0.1223   -2.3584   -1.1763    9.8277   -1.1763   -2.3584
    3.4248    4.3196   -5.0645    0.1223   -2.3584   -1.1763    9.8277   -1.1763
   -2.5309    3.4248    4.3196   -5.0645    0.1223   -2.3584   -1.1763    9.8277
    0.0918   -2.5309    3.4248    4.3196   -5.0645    0.1223   -2.3584   -1.1763
   -3.1301    0.0918   -2.5309    3.4248    4.3196   -5.0645    0.1223   -2.3584
      ⋮
```

```
% Compute the inverse of the autocorrelation matrix
R_xx_inv = inv(R_xx)
```

```
R_xx_inv = 64×64
    0.2589   -0.0083    0.0643   -0.0381    0.1869   -0.1054   -0.0149   -0.0227 ···
   -0.0083    0.2591   -0.0103    0.0655   -0.0441    0.1903   -0.1050   -0.0142
    0.0643   -0.0103    0.2751   -0.0198    0.1120   -0.0703    0.1866   -0.1107
   -0.0381    0.0655   -0.0198    0.2804   -0.0475    0.1272   -0.0678    0.1894
    0.1869   -0.0441    0.1120   -0.0475    0.4153   -0.1237    0.1166   -0.0845
   -0.1054    0.1903   -0.0703    0.1272   -0.1237    0.4580   -0.1174    0.1253
   -0.0149   -0.1050    0.1866   -0.0678    0.1166   -0.1174    0.4587   -0.1156
   -0.0227   -0.0142   -0.1107    0.1894   -0.0845    0.1253   -0.1156    0.4598
    0.0865   -0.0255    0.0074   -0.1230    0.2520   -0.1193    0.1200   -0.1225
   -0.0108    0.0868   -0.0282    0.0087   -0.1309    0.2562   -0.1185    0.1205
```

$$\vdots$$

4) Calculate the spectrum using Eq. (14.4.12) from the note.

We can compute minimum variance power spectrum estimate as follows

$$P_{xx}^{\mathrm{MV}}(f) = \frac{1}{\mathbf{E}^H(f)\mathbf{R}_{xx}^{-1}\mathbf{E}(f)} \qquad (14.4.12)$$

where

- f is the frequency,
- $\boldsymbol{R}_{\mathrm{xx}}$ is the autocorrelation matrix
- $\boldsymbol{E}(f)$ is the constraint vector:

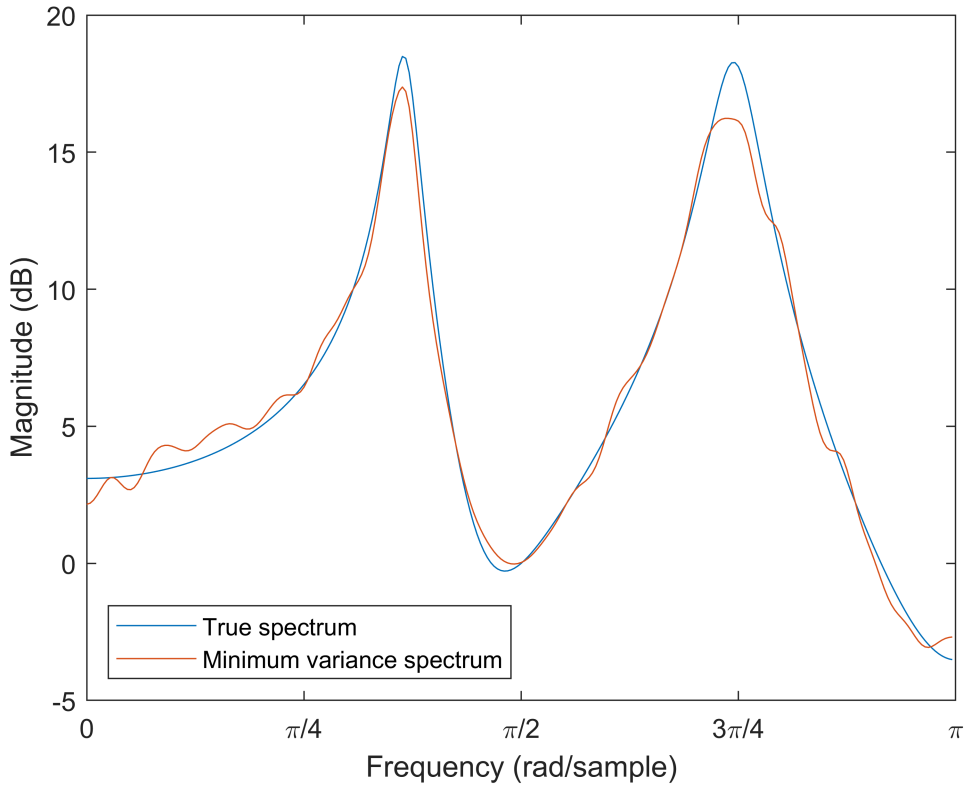
$$\mathbf{E}^t(f_l) = [1 \quad e^{j2\pi f_l} \quad \dots \quad e^{j2\pi p f_l}]$$

This is implemented in the function psdmv() function (see at the end of the document):

```
M = 64; % Size of the autocorrelation matrix
kF = 4; % Frequency resolution multiplier
[I, w] = psdmv(x, M, kF);
[H,w2] = freqz(b, a, M*kF);
plot(w, pow2db(H.*conj(H)), w, pow2db(I));
```

```
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

```
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)')
legend({'True spectrum', 'Minimum variance spectrum'}, 'Location', 'southwest')

xlim([0, pi]);
```



5) Calculate the optimum filters

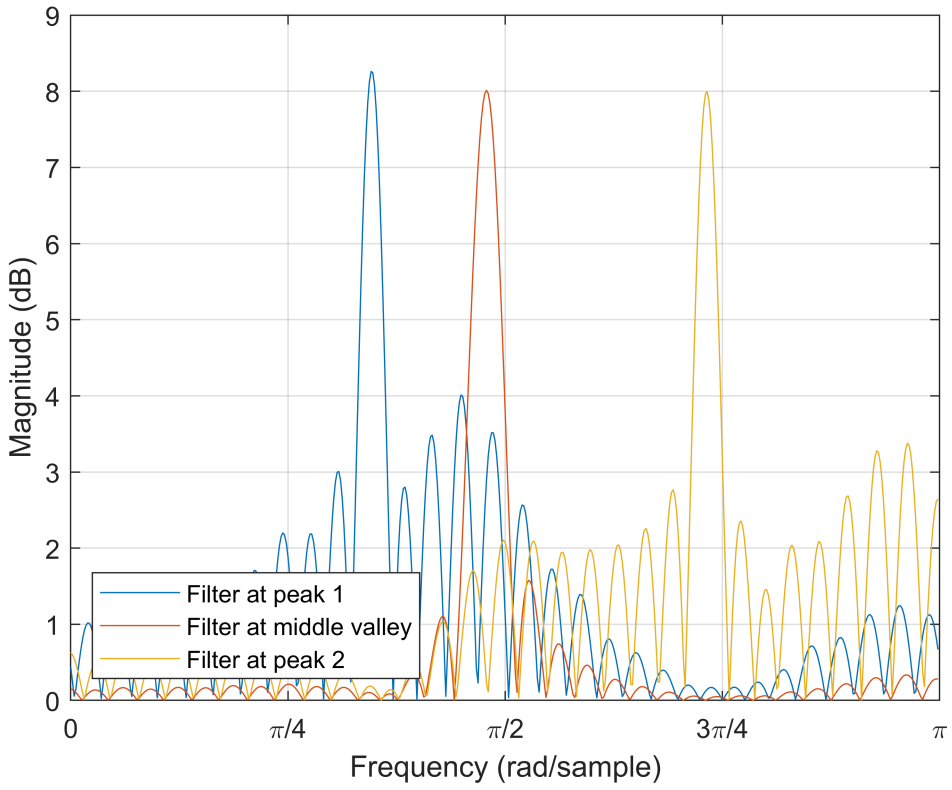
Choose a few representative frequencies in your spectrum. Calculate the optimum filters for estimation of these frequencies and plot the magnitude response of these. Are the filter magnitude responses in concordance with your intuition?

The optimum filters are given by

$$\hat{\mathbf{h}}_{\text{opt}} = \mathbf{\Gamma}_{xx}^{-1}\mathbf{E}(f_l)/\mathbf{E}^H(f_l)\mathbf{\Gamma}_{xx}^{-1}\mathbf{E}(f_l) \qquad (14.4.10)$$

```
M = 64;
f1 = 1.1/(2*pi); % Correspond to the first peak
f2 = 1.5/(2*pi); % Middle valley
f3 = 2.3/(2*pi); % Second peak

% Compute the optimal filters
h1 = optfilter(x, M, f1);
h2 = optfilter(x, M, f2);
h3 = optfilter(x, M, f3);

% Compute the transfer function
[H1, w] = freqz(h1, 1);
[H2, ~] = freqz(h2, 1);
[H3, ~] = freqz(h3, 1);
```

```
plot(w, abs(H1), w, abs(H2), w, abs(H3));
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)')
legend({'Filter at peak 1', 'Filter at middle valley', 'Filter at peak 2'}, 'Location', 'southwest')
grid on;
xlim([0, pi]);
```




```
%ylim([-50, 10])
```

Note how the blue and red peak filters goes to zero at the other peak.

ADSI Problem 4.22: Minimum variance and spectral resolution

On page 472 of the book "Statistical and adaptive signal processing - Spectral estimation, signal modeling, adaptive filtering and array processing" by Manolakis, Ingle and Kogon it is stated that *In addition, minimum-variance spectral estimation provides improved resolution - better than the the* $\delta f \approx 1/N$ *associated with the DFT methods.* In this problem this claim is investigated. Let an $AR(2)$ process be given by

$$x(n) = 0.75x(n-1) - 0.5x(n-2) + w(n)$$

Where $w(n)$ is Gaussian zero-mean unit-variance white noise. Two sinusoidal signals are added to the $AR(2)$ process. They are given by

$$s_1(n) = 5\cos(2\pi\frac{1}{7}n + \phi_1) \quad \text{and} \quad s_2(n) = 4\cos(2\pi\left(\frac{1}{7} + \delta f\right)n + \phi_2)$$

Where δf is a small frequency offset and ϕ_1 and ϕ_2 are random phases uniformly distributed in the range from 0 to 2π.

1) Create a realisation of the signal

Create a 4096 samples long realization of the signal where $\delta f = \frac{1}{50}$.

2) Plot the spectrum of the process using minimum variance and the periodogram

Eventhough the periodogram is generally a poor spectral estimator we will use it in this exercise for simplicity.

2. Plot the spectrum of the process using the minimum variance method and the periodogram. Can the two sinusoids be resolved with both methods?

3) Verify if the book's claim is justified

Adjust δf and find the minimum value where the two sinusoids can be resolved with the two techniques. Is the claim from the book justified?

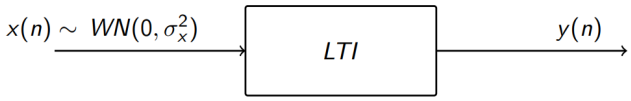
[✓] Exam 2012, Problem 2: Construct an AR(2) model using a signal from a random process

A sequence of data from a random process is given by

$$\{1,\ -1,\ 0,\ 2,\ 3,\ 2,\ -4,\ 1\}$$

Use the data to construct an $AR(2)$ model of the random process and account for any assumptions you make.

We assume that the given sequence is the output of a process whose input was white Gaussian noise with zero mean and unit variance:



Given the input is white noise with zero mean $x(n) \sim \mathrm{WN}\left(0, \sigma_x^2\right)$, the autocorrelation of an AR(q) model is given by Eq. 13.141:

$$r_{yy}(\ell) = -\sum_{k=1}^{p} a_k r_{yy}(\ell - k), \quad \ell > 0$$

This is useful because we can use it to find the coefficients $\{a_k\}$ of an AR(q) model. Given an output signal $y(n)$, we can compute the autocorrelation $r_{yy}(\ell)$ numerically in MATLAB. This means that the expression in Eq. 13.141 becomes a set of p linear equations.

For a second-order $p = 2$ AR model, we get two equations with two unknowns:

$$r_{yy}(1) = -a_1 r_{yy}(0) - a_2 r_{yy}(-1)$$

$$r_{yy}(2) = -a_1 r_{yy}(1) - a_2 r_{yy}(0)$$

We can write it into matrix form:

$$\begin{bmatrix} r_{yy}(0) & r_{yy}(1) \\ r_{yy}(1) & r_{yy}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = -\begin{bmatrix} r_{yy}(1) \\ r_{yy}(2) \end{bmatrix}$$

Solving it will give us the AR(2) model coefficients $\{a_1, a_2\}$.

This is implemented in the arfit() function:

```
y = [1, -1, 0, 2, 3, 2, -4, 1];
[a, ~] = arfit(y, 2)
```

```
a = 2×1
    0.0340
    0.2232
```

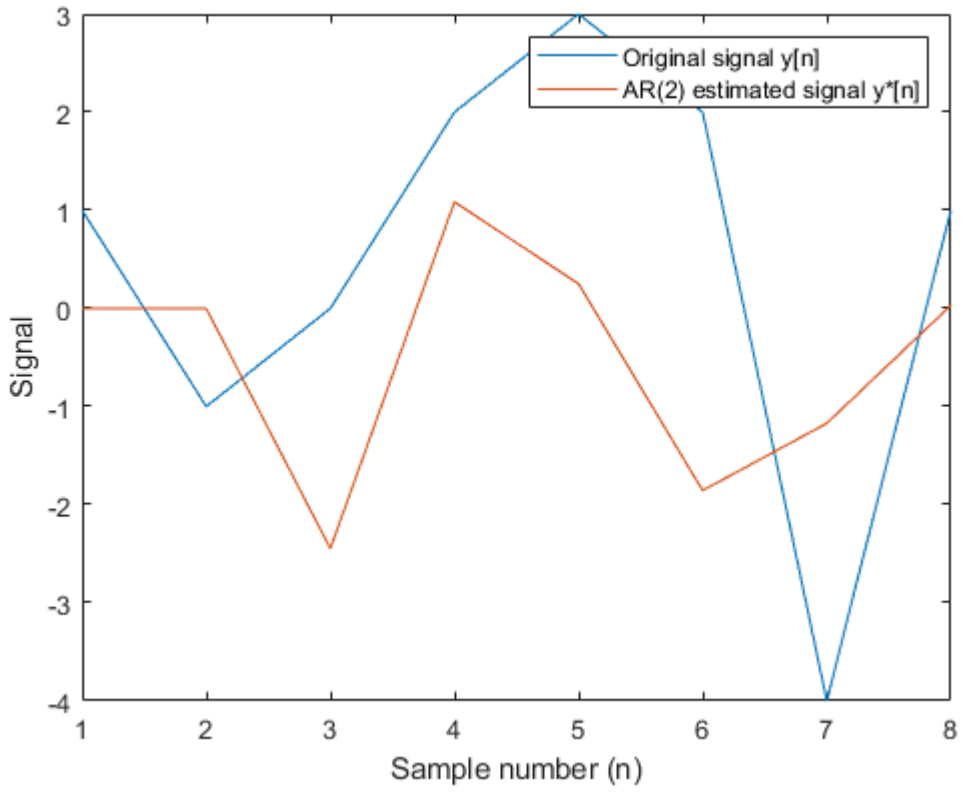
The coefficients for the AR(2) model is; $a_1 = 0.0340$ and $a_2 = 0.2232$.

$$y(n) = -[0.0340 y(n-1) + 0.2232 y(n-2)] + b_0 x(n)$$

We can plot the signal and the estimated AR(2) signal:

```
N = 8;
```

```
% Generate the input
x = randn(N, 1)'; % Random noise with zero mean and unit variance.
b0 = 1;

% Estimate the output signal using the AR(2) model
y_hat = zeros(N, 1);
n = 3:N; % Trick to make it work in MATLAB
y_hat(n) = -(a(1)*y(n-1) + a(2)*y(n-2)) + b0*x(n);

% Plot the two
n = 1:N;
plot(n, y(n), n, y_hat(n))
legend('Original signal y[n]', 'AR(2) estimated signal y*[n]');
xlabel('Sample number (n)')
ylabel('Signal')
```



Functions

```
function r=acrs(x, L)
    % Computes the ACRS r[m] for 0<= m <= L
```

```
    % r=acrs(x-mean(x),L) yields the ACVS
    N=length(x);
    x1=zeros(N+L-1,1);
    x2=x1;
    x1(1:N,1)=x;
    for m=1:L
        x2=zeros(N+L-1,1);
        x2(m:N+m-1,1)=x;
        r(m)=x1'*x2;
    end
    r=r(:)/N;
end

function I=psdper(x, K)
    % Compute periodogram I(ω) using the FFT.
    % K-point FFT >= N
    N=length(x);
    X=fft(x,K);
    I=X.*conj(X)/N;
    I(1)=I(2); % Avoid DC bias
    I=I(:);
end

function r=acrsfft(x, L)
    % Computate the autocorrelation sequence using the FFT.
    % r=acrsfft(x-mean(x),L) yields the ACVS
    N=length(x);
    Q=nextpow2(N+L);
    X=fft(x,2^Q);
    r0=real(ifft(X.*conj(X)));
    r=r0(1:L)/N;
end

function I=psdmodper(x, K)
    % Compute the modified periodogram PSD estimate.
    % K-point FFT >= N
    N=length(x);
    w=hann(N); % choose window
    w=w/(norm(w)/sqrt(N)); % sum w^2[k]=N
    X=fft(x(:).*w(:),K);
    I=X.*conj(X)/N;
    I(1)=I(2); % Avoid DC bias
    I=I(:);
end

function S=psdbt(x, L, K)
    % Computate the Blackman-Tukey PSD estimate.
    % Blackman-Tukey PSD estimator of S(2*pi*k/K)
    if size(x,1) < size(x,2)
        x = x';
    end
    N=length(x);
    w=hann(N); % Data window
    w=w/(norm(w)/sqrt(N)); % sum w^2[k]=N
    x=x.*w; % Data windowing
    r=acrsfft(x,L);
    wc=parzenwin(2*L-1); % Lag window
    rw=r.*wc(L:2*L-1); % Lag windowing
```

```
    g=zeros(K,1);
    g(1:L)=rw;
    g(K-L+2:K)=flipud(rw(2:L));
    G=fft(g,K); % f even => F real
    S=2*real(G(1:K/2));
    S(1)=S(2);
end

function S=psdwelch(x, L, K)
    % Compute the Welch PSD estimate.
    % Welch PSD estimator of S(2*pi*k/K)
    M=fix((length(x)-L/2)/(L/2)) % 50% overlap
    time=(1:L)';
    I=zeros(K,1);
    w=hanning(L); % Choose window
    w=w/(norm(w)/sqrt(L)); % sum w^2[k]=L
    for m=1:M
    %xw=w.*detrend(x(time)); % detrenting
    xw=w.*x(time); % data windowing
    X=fft(xw,K);
    I=I+X.*conj(X);
    time=time+L/2;
    end
    I=I/(M*L); S=2*I(1:K/2); S(1)=S(2);
end

function [a,v] = arfit(x,p)
    % fit AR(p) model from data
    % x: data
    % p: model order
    % a: a coefficients
    % v: variance
    if isrow(x)
        x = x'; % Convert to a column vector
    end

    % Compute the autocorrelation
    [r_xx, lags] = xcorr(x, p, 'biased');

    % Select elements r_xx[0] to r_xx[p-1]
    R_elems = r_xx(p+1:2*p);

    % Create the Toeplitz matrix
    R = toeplitz(R_elems);

    % Select elements r_xx[1] to r_xx[p]
    r = r_xx(p+2:2*p+1);

    % Solve systems of linear equations using mldivide function
    a = mldivide(R, -r);

    % Compute the variance
    v = r_xx(p+1) + a'*r;
end

function [S, w] = ar2psd(a, v, N)
% AR2PSD Compute the Power Spectral Density from AR(p) coefficients
%   [S, w] = ar2psd(a, v, N)
```

```
% a: AR(p) coefficients
% v: the variance
% N: number of points in the range [1, pi]
% S: the estimated power spectrum
% w: frequencies
    w = linspace(0, 1, N) * pi;

    % Compute the transfer function
    % Used Eq. (13.133) in the book
    H = ones(N, 1);
    for k=1:numel(a)
        H = H + a(k)*exp(-1j * w' * k);
    end
    H = 1./H;

    % Finally compute the PSD
    S = v * H.*conj(H);
end

function [psd, f] = psdmv(x, M, kF)
% Estimate PSD using Capon's Minimum Variance method
% x: input signal
% M: autocorrelation matrix size
% kF:frequency resolution multiplier
    if nargin < 3
        kF = 4;
    end
    if nargin < 2
        M = 64;
    end
    [rx, ~] = xcorr(x, M-1, 'biased');
    Rx=toeplitz(rx(M:end));% Autocorrelation matrix
    RxInv=inv(Rx);
    Nf = kF*M; % frequency resolution, Nf >= M
    f=0.5/Nf*(0:Nf-1); % freq spectrum
    psd=zeros(M,1); % pre-alloc
    t=(0:M-1)';
    for q=1:Nf
        Ef=exp(1i*2*pi*f(q)*t);
        psd(q)=M/(Ef'*RxInv*Ef);
    end
    f = 2*pi*f;
end

function h = optfilter(x, M, f)
% Compute the optimum filter for estimating a given frequency
% Computes Eq. (14.4.10) in the Minimum Variance material
% x: input signal
% M: autocorrelation matrix size (default: 64)
% f: the given frequency
    if nargin < 2
        M = 64;
    end
    [rx,~]=xcorr(x, M-1, 'biased');
    Rx=toeplitz(rx(M:end));
    invRx=inv(Rx);
    Ef = zeros(M, 1);
    for t=1:M
```

```
        % Compute E(f) vector
        Ef(t)=exp(1i*2*pi*f*(t-1));
    end
    h = (sqrt(M)*invRx*Ef) / (Ef'*invRx*Ef);
end
```