Power Spectral Density

Table of Contents

Power Spectral Density

Definition

Power spectral density is used to characterise stationary random processes in the frequency domain. Power Spectral Density is defined as follows:

$$S_{xx}(\omega) = R_{xx}(e^{j\omega}) = \sum_{\ell=-\infty}^{\infty} r_{xx}(\ell) e^{-j\ell\omega}$$

Essentially, Power Spectral Density is the discrete-time Fourier Transform of the auto-correlation sequence $r_{xx}(\ell)$.

Notice that phases disappear when we compute the spectrum.

INSIGHT: If we have a signal with some oscillation, the oscilation is observed in the autocorrelation. Taking the Fourier Transform of the same autocorrelation, we will observe that there is energy at that particular frequency.

We can go the opposite direction. Given the PSD, we can compute the auto-correlation sequence by the inverse Fourier Transform as follows:

$$r_{xx}(\ell) = \frac{1}{2\pi} \int_{-\pi}^{\pi} S_{xx}(\omega) e^{\,j\omega\ell}\, \mathrm{d}\omega$$

What is the average power of a stable LTI system?

Suppose we want to know the average power of the filter i.e., $E[y^2(n)]$

The average power of a filter is actually just the value of the autocorrelation at lag 0. How? We can see this if look at the expression from a different perspective:

$$E[y^2(n)] = E[y(n)y(n-0)] = r_y(0)$$

Using the inverse definition of the PSD, we can compute $r_y(0)$

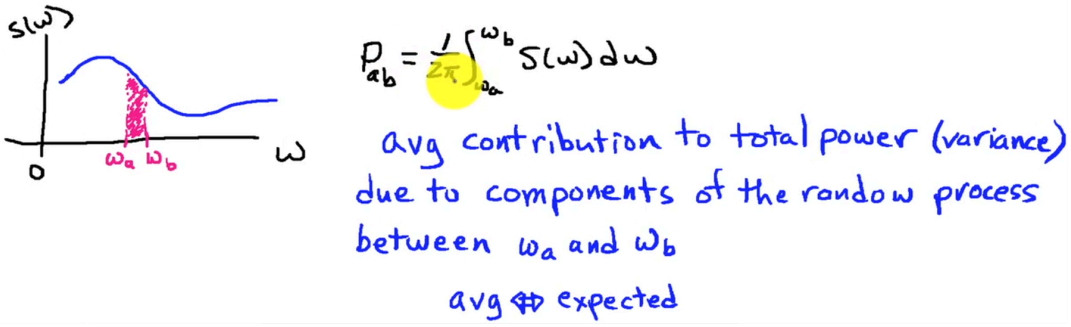
$$r_y(0) = \frac{1}{2\pi}\int_{-\pi}^{\pi} S_{xx}(\omega)e^{j\omega\cdot 0}\,\mathrm{d}\omega$$

$$= \frac{1}{2\pi}\int_{-\pi}^{\pi} S_{xx}(\omega)\,\mathrm{d}\omega$$

If we integrate the PSD from $-\pi$ to π, the resulting quantity is equal to the power in the time-domain. This means that **energy in the time-domain is equal to energy in the frequency domain**.

Another perspective:

Let P_{ab} be a normalised integration of the PSD from the interval ω_a to ω_b. The quantity P_{ab} is the expected (or average) contribution to the total power (or variance) that is due to the components of the random process between ω_a and ω_b. In other words, the area under the curve between ω_a and ω_b is the power that that portion of the spectrum is expected to contribute to the random process. It tells us how power is distrubuted in a frequency spectrum.



Example: Power Spectrum Analyser

Suppose we want to know the power spectrum of some input signal $x[n]$. We can pass that signal to an ideal bandpass filter with very narrow band and compute the power at the output. We define the bandpass filter to have the unit response in the vecinity of ω_c and the band $\Delta\omega$ is very small. Our bandpass filter rejects frequencies outside $\Delta\omega$ and pass any frequency inside this band.

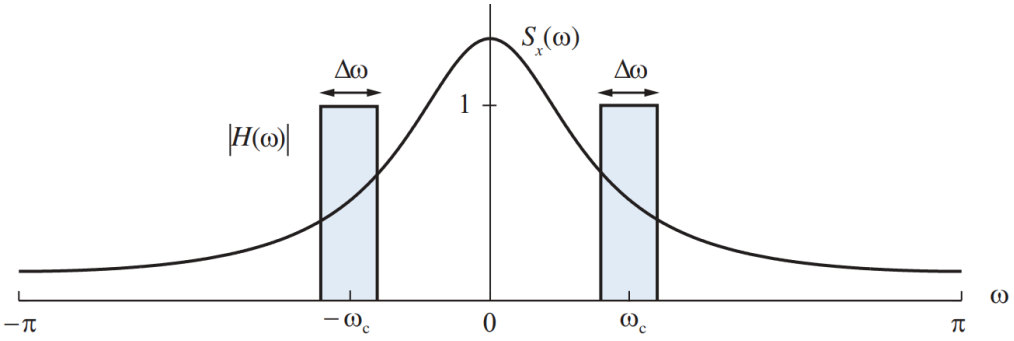


Figure 13.13 Physical interpretation of power spectrum density as power at the output of a narrowband LTI system.

Now, the question is how to do we compute the power spectrum of our ideal narrow bandpass filter when the signal is random?

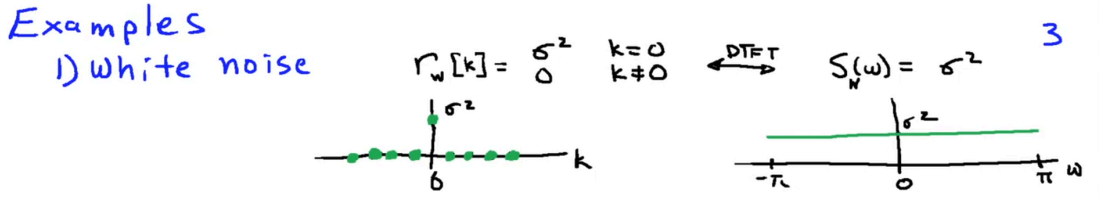
The answer is to compute the the average output of the filter.

$$E[y^2(n)] = 2\frac{1}{2\pi}\int_{\omega_c-\Delta\omega/2}^{\omega_c+\Delta\omega/2} S_{xx}(\omega)d\omega \approx \frac{1}{\pi}S_{xx}(\omega_c)\Delta\omega$$

Since the spectrum is symmetic around zero, we get power contribution from both sides of zero. This is called a double-sided spectrum. Therefore, we have $2\frac{1}{2\pi}$ in the equation above.

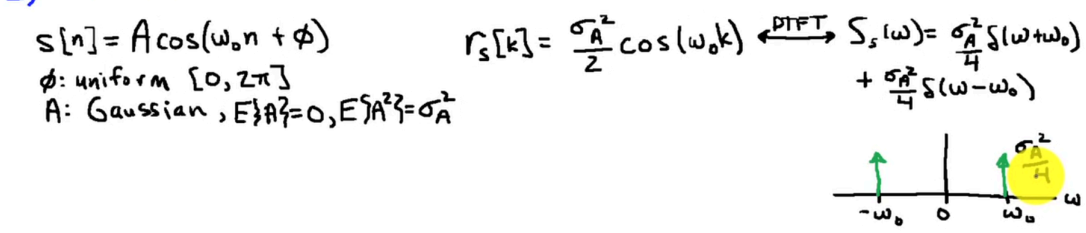
PSDs of Some Random Processes

White Noise



The power spectral density of white noise is constant which means that we have the same power across all frequencies. This signal is called white noise because the power is equally distributed across the entire spectrum.

Random Sinusoid

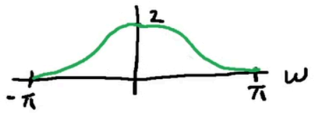


The power spectral density shows that the power is concentrated at $\pm\omega_0$. The area under these concentrations is $\frac{1}{4}\sigma A^2$.

Coloured Noise

3) Colored Noise

$$r_c[k] = \begin{cases} 1 & k=0 \\ 1/2 & k=\pm 1 \\ 0 & \text{otherwise} \end{cases} \quad \overset{\text{DTFT}}{\longleftrightarrow} \quad S_c(\omega) = 1 + \cos(\omega)$$



PSD Estimation

Since we cannot have a mathematical model of random signals instead we use a statistical model. The Power Spectrum Density describes the variance (or power) of a random signal as a function of frequency. PSD can tell us where the energy is distributed.

Typically, we can estimate the power spectrum given a set of data. There is two main approaches for spectrum estimation:

a) Non-parametric Methods: do not assume that a particular model generated the data

b) Parameteric Methods: assume that the data is generated in a certain way e.g. by an AR(q) process

INSIGHT: There are various ways to compute PSD estimates. Now the question is, which PSD estimate should we believe in? Doing PSD estimation is a little bit like black art. It is really difficult to estimate PSD without knowing something about the process that generated the signal. Are some of the bumps in the PSD caused by the engine or the turbulence? In practice, we would record the sound from multiple places. This allows us to compare PSD estimates and compare energy.

Non-parametric Methods

a) Non-parametric Methods: do not assume that a particular model generated the data. In this case, we just use the FFT. Given a finite set of N observations, the general approach for the non-parametric methods for power spectrum estimation of a random signal is:

1. Window the observed signal $y(n)$

$$y_w(n) = w(n)y(n),\ 0 \le n \le N-1$$

2. Take the Fourier Transform:

$$y_w(n) \leftrightarrow X_w(e^{j\omega})$$

3. Estimate the power spectrum

$\hat{S_{xx}}(\omega) = \frac{1}{N \cdot F} |X_w(e^{j\omega})|^2$ where F is constant that depends on the window

There are several techniques under the umbrella of non-parametric PSD estimation:

- Periodogram
- Bartlett's method
- Welch's method

Periodogram

A periodogram is a method for estimating the Power Spectrum Density of a random signal from a finite set of N observations given by:

$$I(\omega) = \frac{1}{N} \left| \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\mathrm{j}\omega n} \right|^2 \qquad (14.34)$$

The periodogram uses a rectangular window $w(n) = 1$.

Mean

Mean of the periodogram Taking the mathematical expectation of (14.33) and using (14.26) (recall that $\hat{c}[\ell] = \hat{r}[\ell]$ for $m = 0$), we obtain the relation

$$\mathrm{E}[I(\omega)] = \sum_{\ell=-(N-1)}^{N-1} \mathrm{E}(\hat{r}[\ell]) \mathrm{e}^{-\mathrm{j}\omega\ell} = \sum_{\ell=-(N-1)}^{N-1} \left(1 - \frac{|\ell|}{N}\right) r[\ell] \mathrm{e}^{-\mathrm{j}\omega\ell}. \qquad (14.40)$$

Since $\mathrm{E}[I(\omega)] \neq S(\omega)$, the periodogram is a *biased* estimator of $S(\omega)$. However, for each $|\ell|$, the factor $(1 - |\ell|/N) \to 1$ as $N \to \infty$. Hence, we have

$$\lim_{N\to\infty} \mathrm{E}[I(\omega)] = S(\omega), \qquad (14.41)$$

Eq. (14.41) says that as the window length N increases towards infinity, the mean of the periodogram converges to the true power spectrum. Asymptotically (as N increases) the periodogram is an unbiased estimator of $S(\omega)$.

For finite N, it has some bias which is reflected by the window function.

Variance

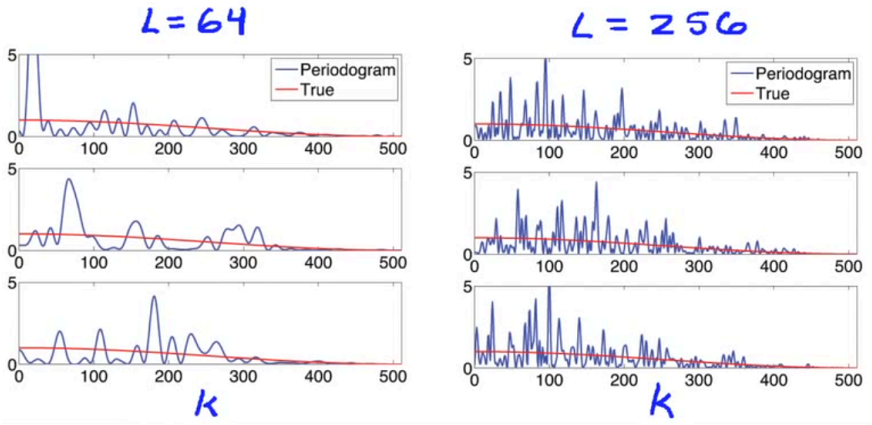
Under a variety of assumptions which holds for most random processes of practical interest, the variance of the periodogram estimator behaves like the square of the true PSD:

$$\text{var}[I(\omega)] \approx S^2(\omega), \quad 0 < \omega < \pi \tag{14.50}$$

The main implication of (14.50) is that the **periodogram is not a good estimator of PSD** because, independently of N, the standard deviation of the estimator is as large as the quantity to be estimated.

In other words, the periodogram is not a consistent estimator; that is, its distribution does not tend to cluster more closely around the true PSD as N increases.

Thus the definition of the PSD by $S(\omega) = \lim_{N\to\infty} I(\omega)$ is not valid because even if $\lim_{N\to\infty} \text{E}[(\omega)] = S(\omega)$, the variance of $I(\omega)$ does not tend to zero as $N \to \infty$.



The variance is high at lower frequencies where the true power spectrum is larger. As the true spectrum goes to zero, the variance gets smaller. This is in agreement with (14.50). If we get better resolution i.e.,L goes from 64 to 256, the behaviour is still present.

Covariance

The covariance between values of the periodogram at harmonically spaced frequencies is given by

$$\text{cov}\left[I\left(2\pi k_1/N\right), I\left(2\pi k_2/N\right)\right] \approx 0, \; k_1 \neq k_2 \tag{14.51}$$

which states that closely spaced values of the periodogram are uncorrelated.

Modified Periodogram

The term is modified periodogram is used to denote a periodogram where the window $w(n)$ is explictly given:

$$\tilde{I}(\omega) \triangleq \frac{1}{N}\left|\sum_{n=0}^{N-1} w[n]x[n]\mathrm{e}^{-\mathrm{j}\omega n}\right|^2 \qquad (14.52)$$

were $w[n]$ is a data window of size N.

The purpose of using a data window is to **reduce the spectral leakage** caused by strong narrowband components by lowering the level of sidelobes.

The data window must be normalised to ensure that the periodogram of *x*[*n*] is asymptotically unbiased :

$$\sum_{n=0}^{N-1} w^2(n) = N. \qquad (14.60)$$

```
N = 64;
w = bartlett(N);
w = w / (norm(w)/sqrt(N));
sum(w.^2)
```

```
ans = 64.0000
```

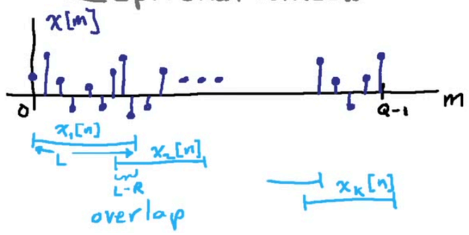
Blackman–Tukey Method

Welch's method

- average K independent random variables: reduce variance by factor of K
- Divide $x[m]$, $m=0,1,\ldots Q-1$ into (possibly) overlapping segments of length L. r^{th} segment:

$$x_r[n] = x[rR+n]\,w[n], \quad n=0,1,\ldots L-1, \quad r=1,2,\ldots,K$$

optional window



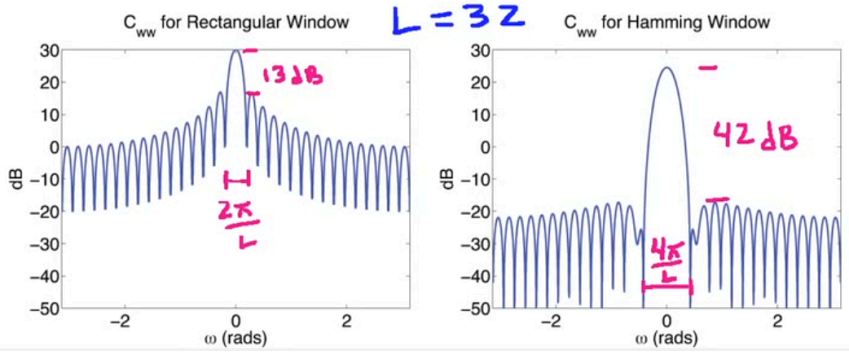
- Average periodograms from each segment

$$I_r(\omega) = \frac{1}{2\pi \sum_{n=0}^{L-1} w^2[n]} \left| \sum_{n=0}^{L-1} x_r[n] e^{-j\omega n} \right|^2$$

$$\hat{S}_{xx}^{av}(\omega) = \frac{1}{K} \sum_{k=1}^{K} I_r(\omega)$$

Variance: $Var\{\hat{S}_{xx}^{per}(\omega)\} \geq Var\{\hat{S}_{xx}^{av}(\omega)\} \geq \frac{1}{K} Var\{\hat{S}_{xx}^{per}(\omega)\}$

1/K reduction if $I_r(\omega)$ are independent



The nature of bias depends on the chosen window. There is a trade-off between main lobe width and side-lobe height.

- Rectangular window gives you the narrowest main lobe but a fairly large side-lobe height.
- Hamming window gives you twice the main lobe width but the side-lobes are much lower.

Hamming window sacrifies some resolution for increased dynamic range.

Dynamic range is the ratio of the largest signal amplitute and the smallest signal amplitude. How well can we realible distintinguish the largest from the smallest.

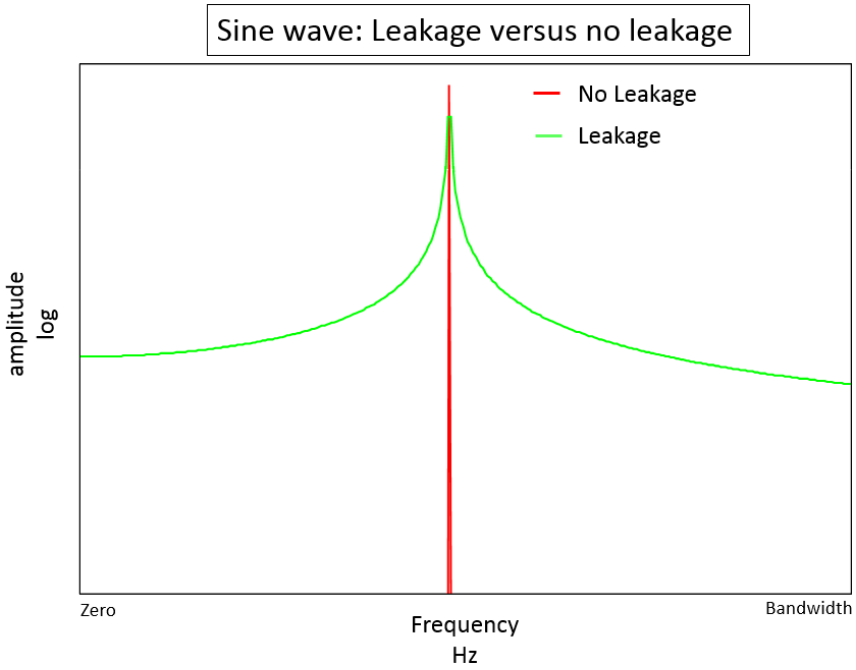
Windows are used to manage bias.

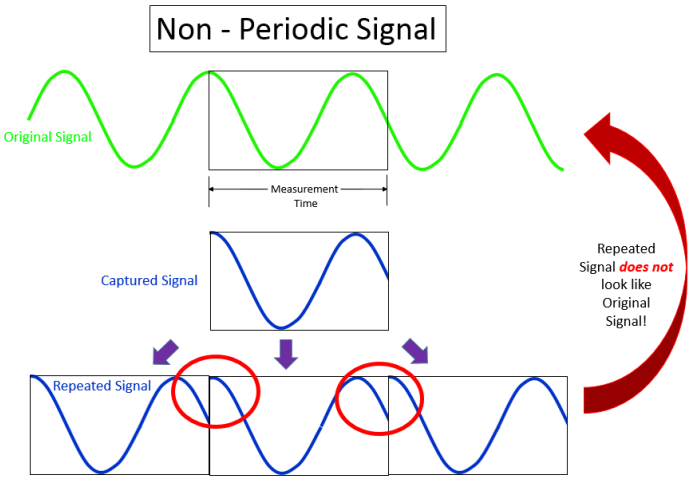
Typically, we will trade resolution (main-lobe width) for dynamic range (side-lobe height)

Spectral Leakage

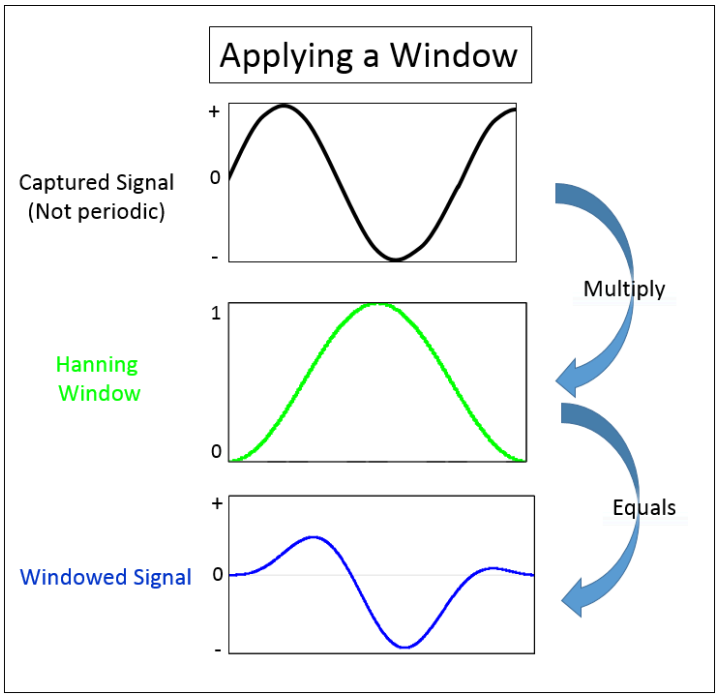
Leakage is an artefact of an FFT applied to a non-periodic data. Even if a signal is periodic like a sinousoid, when the signal is measured with some interval it can be non-periodic. The issue is that FFT assumes that the signal is periodic and repeats itself after the measured interval. This is not the case when the data is non-periodic, which contains sharp transitions at the end of each measured interval. These sharp changes have a broad frequency response which lead to spectral leakage. Windows can help ***minimize*** the effects of leakage by smoothing the time domain signal, but cannot eliminate leakage.

A signal with leakage (green in *Figure 2*) has lower amplitude and a broader frequency response than a signal with no leakage (red in *Figure 2*). This makes it difficult to quantify the signal properly in the frequency domain.

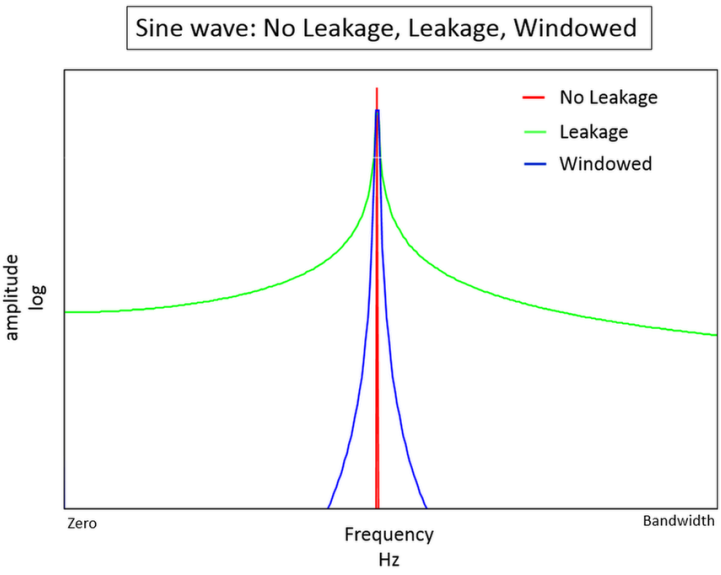




To reduce leakage, a mathematical function called a window is applied to the data. Windows are designed to reduce the sharp transient in the re-created signal as much as possible.



The main benefit of windowing is that the leakage is now confined over a smaller frequency range, instead of affecting the entire frequency bandwidth of the measurement.



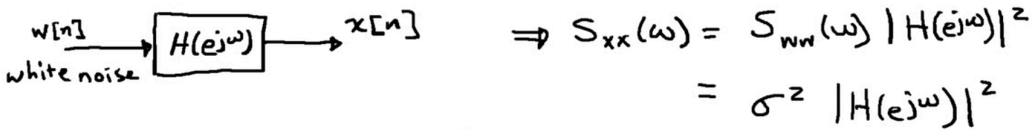
Parametric Methods

b) Parameteric Methods: assume that the data is generated in a certain way. The general approach is:

1. Measure and get some data $x(n)$
2. Use the data to estimate the model parameters e.g. ARMA coefficients
3. Express the estimated PSD $\hat{S}_{xx}(\omega)$ as a function pof the model parameters.

Parametric Model: ARMA(p, q)

For example, we might assume that the data is the output of the LTI system with frequency response $H(e^{j\omega})$ given white noise signal as input:



There are several different models that can be used to estimate the PSD.

- AR(q) model, autoregressive model, all-pole model, only poles in its frequency response.
- MA(p) model, moving average, only zeros in its frequency response.
- ARMA(q, p) model

Whether a model has poles, zeros or pole-zeros results in different characteristic of the PSD estimate.

Parametric Model: Sinusoids with Additive Noise

Another example of a parametric method is Sinusoid in Noise model. Here, the model assumes that the data that we have measured $x[n]$ consists of combination of multiple sinusoids with additive noise $w(n)$. With this model, the PSD is:

Sinusoids in noise model

$$x[n] = \sum_{i=1}^{L} A_i e^{j\omega_i n} + w[n] \Rightarrow S_{xx}(\omega) = \sum_{i=1}^{L} \sigma_i^2 \delta(\omega - \omega_i) + \sigma_w^2$$

unknown: $\sigma_i^2, \omega_i, \sigma^2$

If we know the parameters σ_i^2, ω_i and σ_w^2 then we our PSD.

Non-parametric Methods vs Parameteric Methods

Comparison

If model is correct -
- High quality spectrum estimates
- Significantly less data required

If model is wrong -
- Parametric can give wrong/misleading estimates

Coherence Function

Cross Power Spectrum Density

Cross power spectrum density $S_{yx}(\omega)$ is the Fourier Transform of the cross-correlation:

$$S_{yx}(\omega) = \sum_{\ell=-\infty}^{\infty} r_{yx}(\ell) e^{-j\omega\ell}$$

The cross power spectrum compares two signals and computes how much the **two signals oscillate in phase**. If we get a large value at a given frequency ω_0, then this means the two signals are oscilliting together at ω_0. If $S_{yx}(\omega_0)$ is small then the two signal are not in phase at ω_0.

Why do we care? The cross power spectrum density allows us to define a coherence function.

Definition of the Coherence Function

The (magnitude squared) coherence function allows us to determine in an elegant way when a system is linear.

Formally, the (magnitude squared) coherence function is defined:

$$|\gamma_{yx}(\omega)|^2 = \frac{|S_{yx}(\omega)|^2}{S_{yy}(\omega)S_{xx}(\omega)}$$

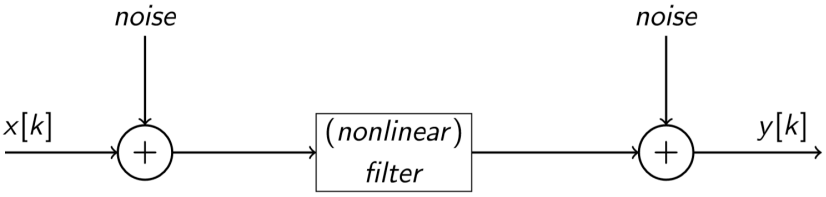
where $0 \leq |\gamma_{yx}(\omega)|^2 \leq 1$ i.e., the function yields a number between zero and one.

For a linear system, the coherence function should be 1 or very close to it:

$$|\gamma_{yx}(\omega)|^2 = 1$$

The coherence function of a perfect linear system is equal to 1.

However, in the real-world we don't always get a coherence of 1. Suppose we want to determine whether a system or a filter is linear:



There are three primal factors that causes the coherence function to fall below 1.

- Factor 1: After we measure $x(n)$, some unaccounted noise creeps into the signal before going into the system.
- Factor 2: Some unaccounted noise creeps into the output signal before we measure it.
- Factor 3: The filter contains some nonlinear components.

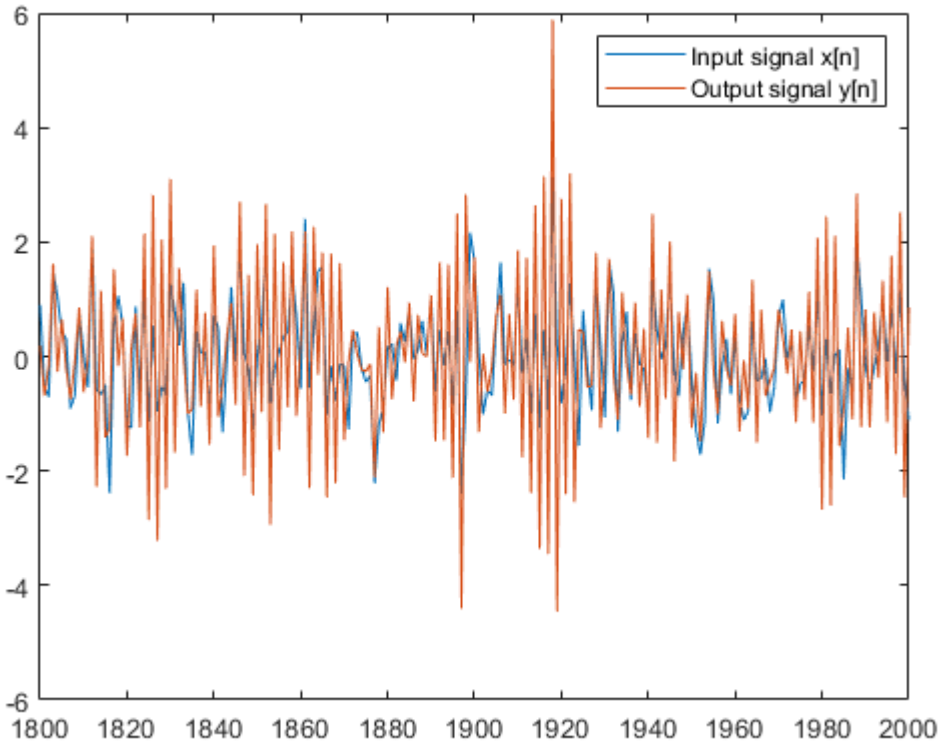
Important lesson: The coherence function is a tool that we can use to measure whether any given system is actually linear. Since the coherence function measures how close a system is linear, we don't know whether a low coherence is due to noise or a non-linear filter.

Coherence Estimates in MATLAB

```
clear variables;
```

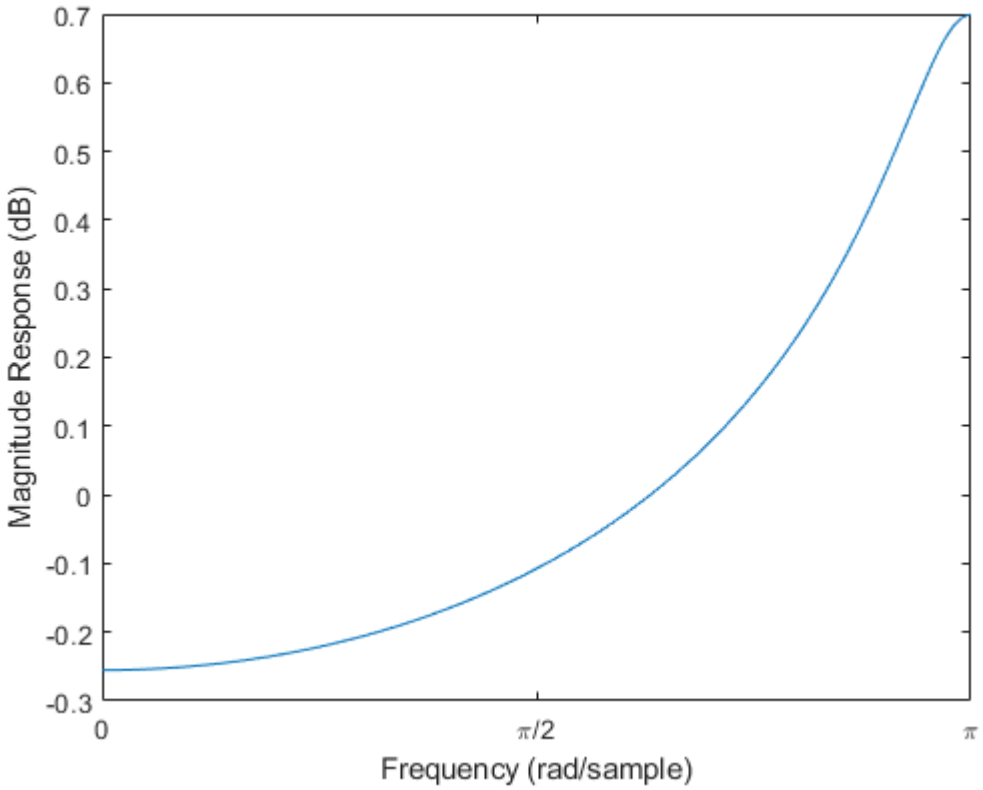
High-Pass IIR filter

```
N = 2000;
n = 1800:N;

% Create an input signal of random numbers
x = randn(N, 1);

% Define a high pass IIR filter that attenuates low
% frequencies and amplifies high frequencies
b = 1;
a = [1, 0.8];

y = filter(b, a, x);
plot(n, x(n), n, y(n));
legend('Input signal x[n]', 'Output signal y[n]')
```

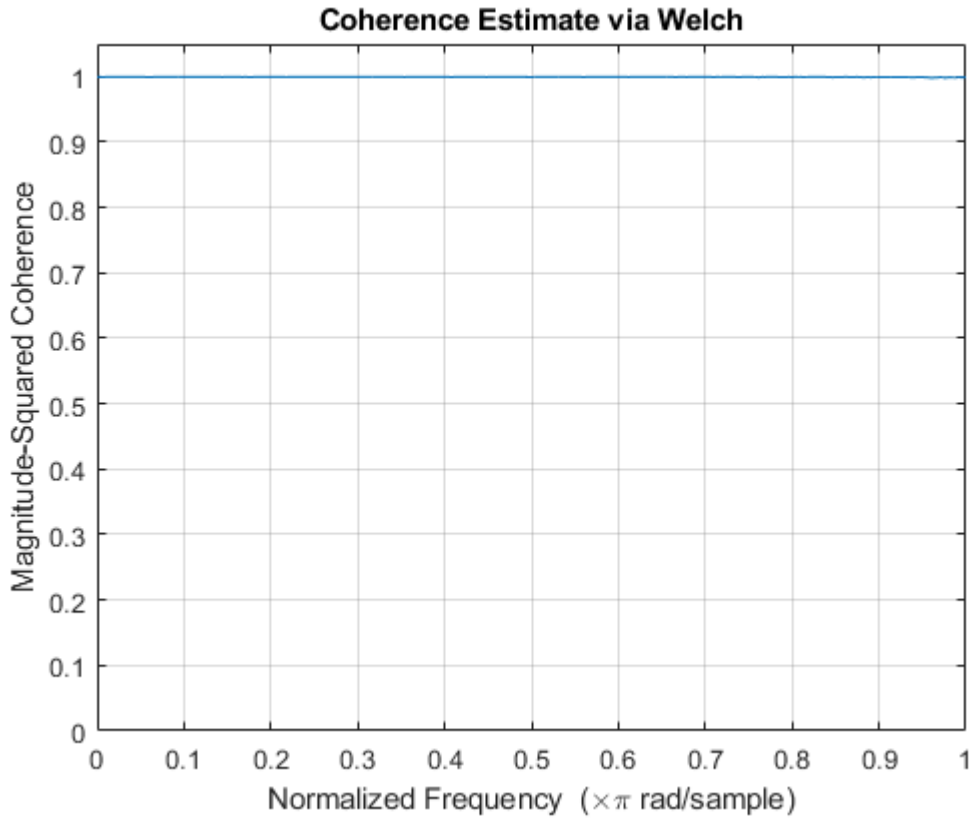


We can plot the magnitude response to see which frequencies the filter is attenuating and which frequency components it is amplifying:

```
[H, w] = freqz(b, a, 'whole');
plot(w, log10(abs(H)));
set(gca,'XTick',0:pi/2:2*pi)
set(gca,'XTickLabel',{'0','\pi/2','\pi','3\pi/2','2\pi'})
xlabel('Frequency (rad/sample)')
ylabel('Magnitude Response (dB)')
xlim([0, pi]);
```



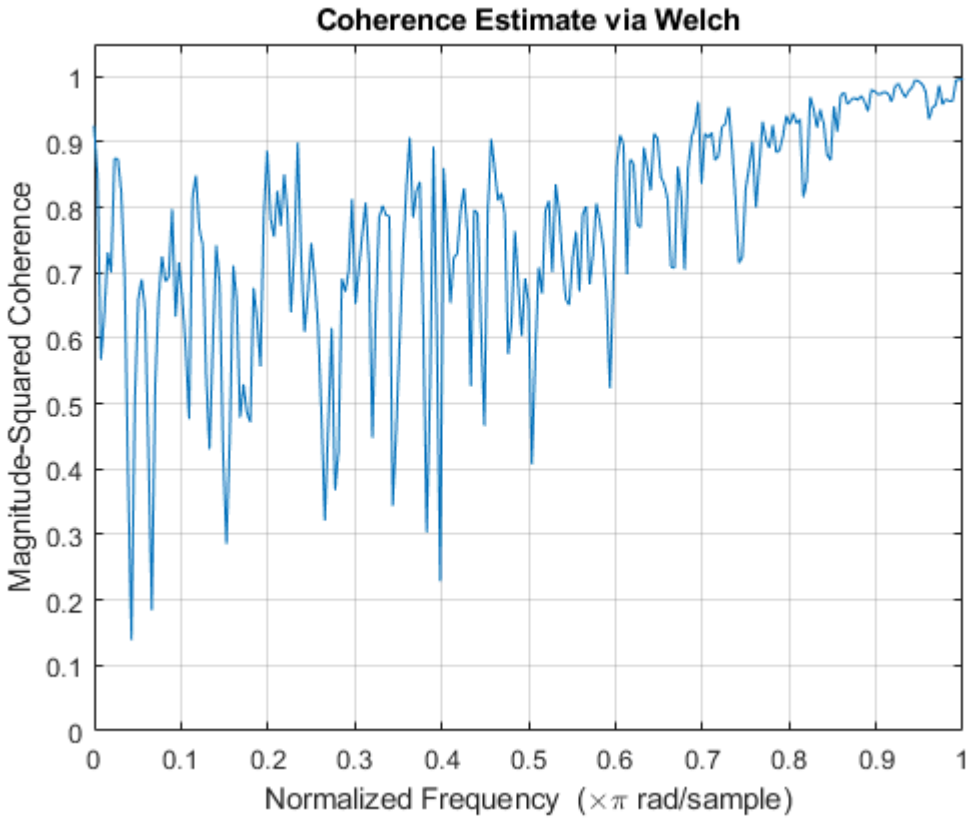

```
% Compute and plot the coherence estimate
mscohere(y, x);
ylim([0, 1.05]);
```



Notice that the estimate coherence function is 1. This is not a surprise because we are sending the input signal through a LTI system.

High-pass filter with noise at the output

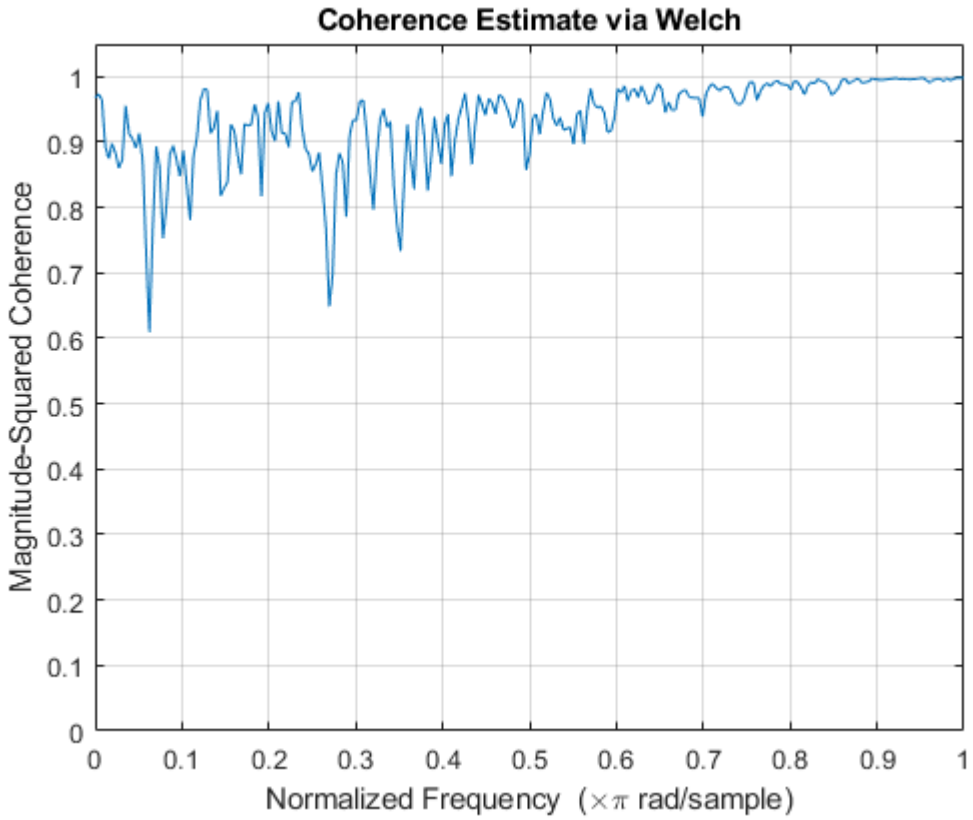
```
% Define a high pass IIR filter that attenuates low
% frequencies and amplifies high frequencies
noise = 0.5;
y = filter(b, a, x);
y = y + noise*randn(N, 1);

% Compute and plot the coherence estimate
mscohere(y, x);
ylim([0, 1.05]);
```



We observe that the coherence falls below 1 although we have a LTI filter because we have added a lot of noise to the output signal. Notice that there are low coherence at the low frequencies and high coherence at the high frequencies. This is because the high-pass filter attenuates low frequencies, and amplifies high frequencies.

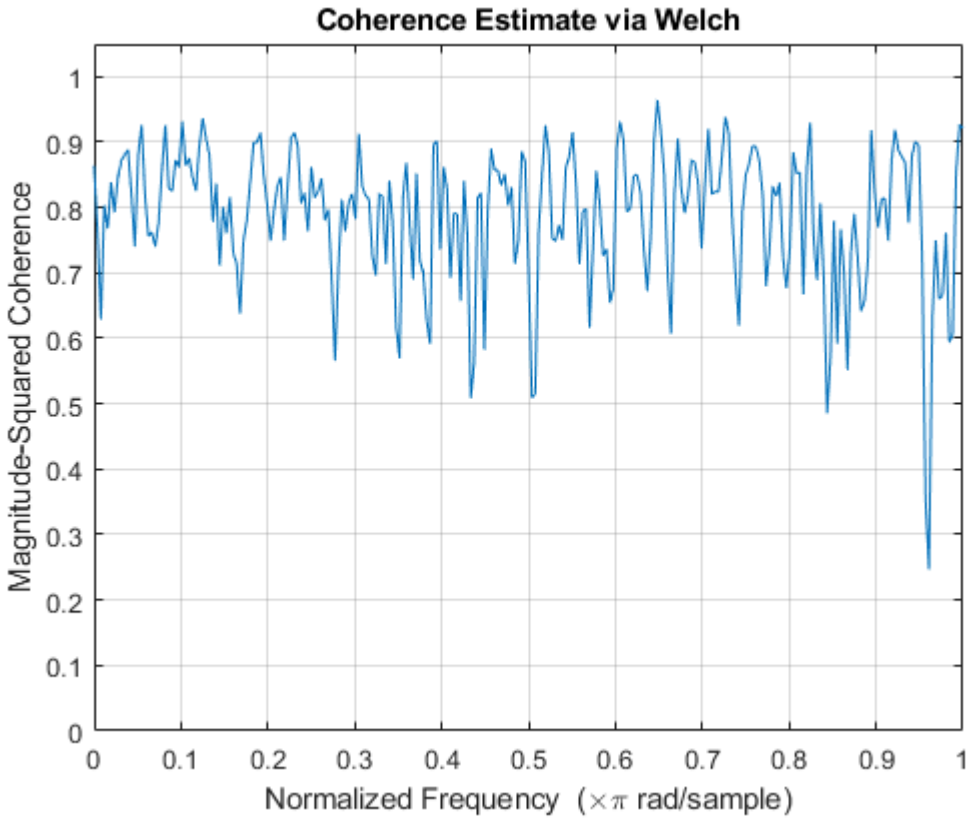
If the noise is reduced, we get close to 1:

```
noise = 0.2;
y = filter(b, a, x);
y = y + noise*randn(N, 1);

% Compute and plot the coherence estimate
mscohere(y, x);
ylim([0, 1.05]);
```



High-pass filter with noise at the input

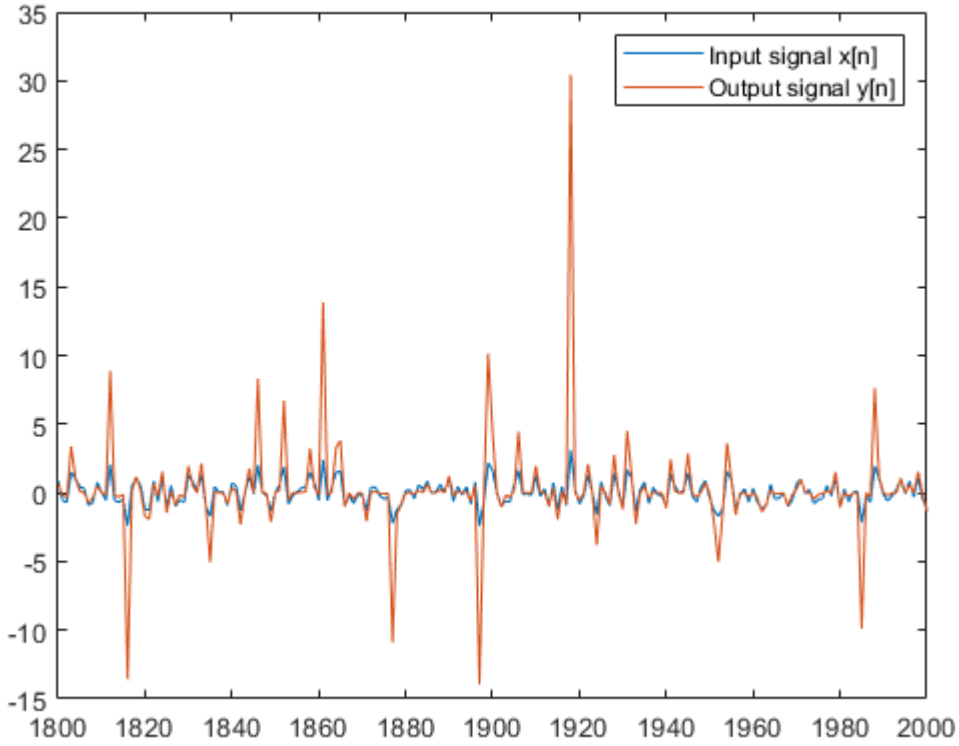
```
noise = 0.5;
y = filter(b, a, x + noise*randn(N, 1));
mscohere(y, x);
ylim([0, 1.05]);
```



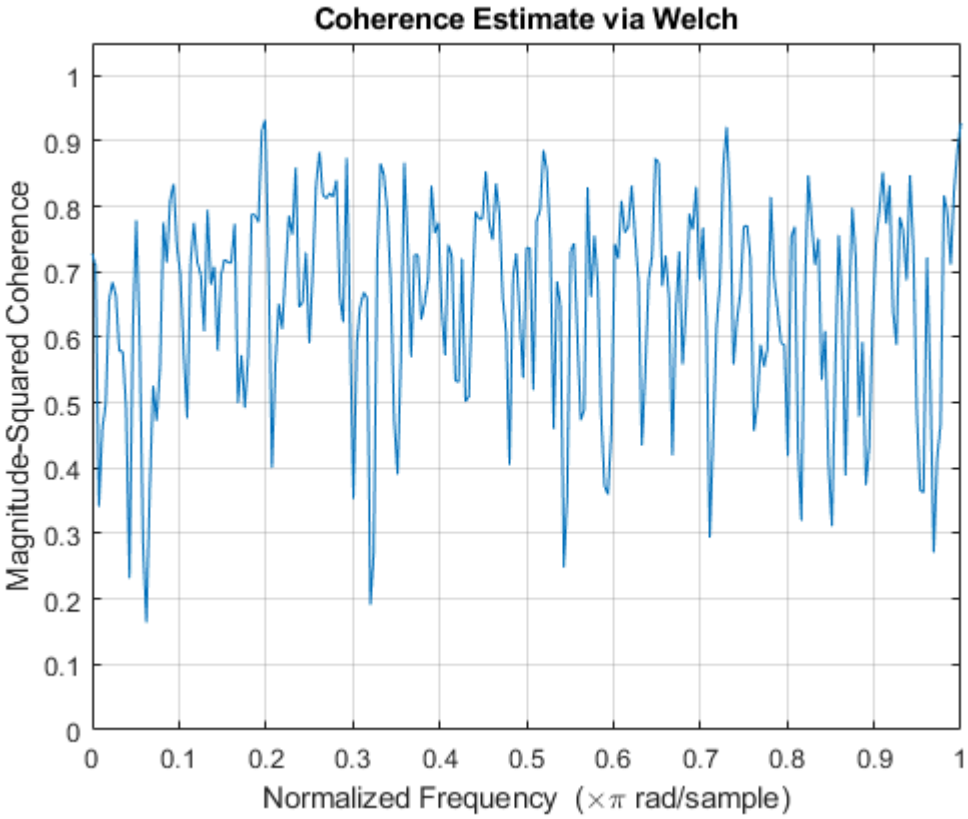
The additional noise is also being filtered so we have equal amount.

Nonlinear filter

```
% Take each sample and cube it
y = x.^3;
plot(n, x(n), n, y(n));
legend('Input signal x[n]', 'Output signal y[n]')
```




```
mscohere(y, x);
ylim([0, 1.05]);
```



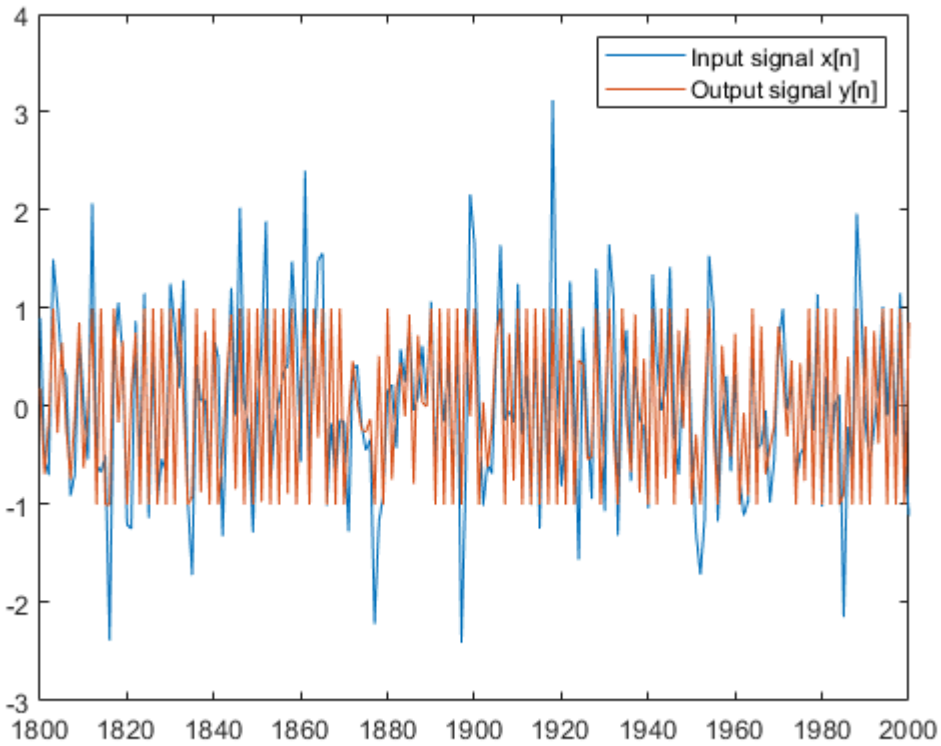
The coherence is all over the place.

Lesson: Since the coherence function measures how close a system is linear, we don't know whether a low coherence is due to noise or a non-linear filter.

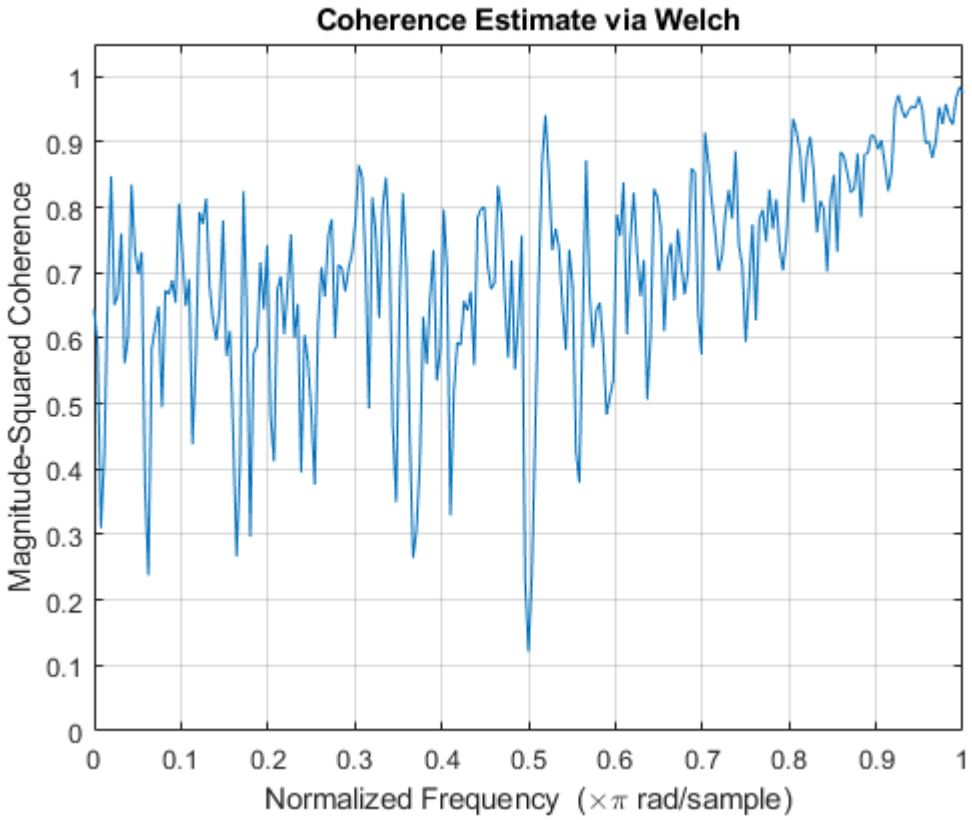
In practice, the coherence function is close to 1. The difference $1 - \mathrm{mscohere}(y, x)$ tells us how well we can remove noise.

Nonlinear filter via clipping

```
% Non-linear filter via clipping
y = filter(b, a, x);
for k=1:N
    if y(k)>1
        y(k)=1;
    end
    if y(k)<-1
        y(k)=-1;
    end
end
plot(n, x(n), n, y(n));
legend('Input signal x[n]', 'Output signal y[n]')
```




```
mscohere(y, x);
ylim([0, 1.05]);
```



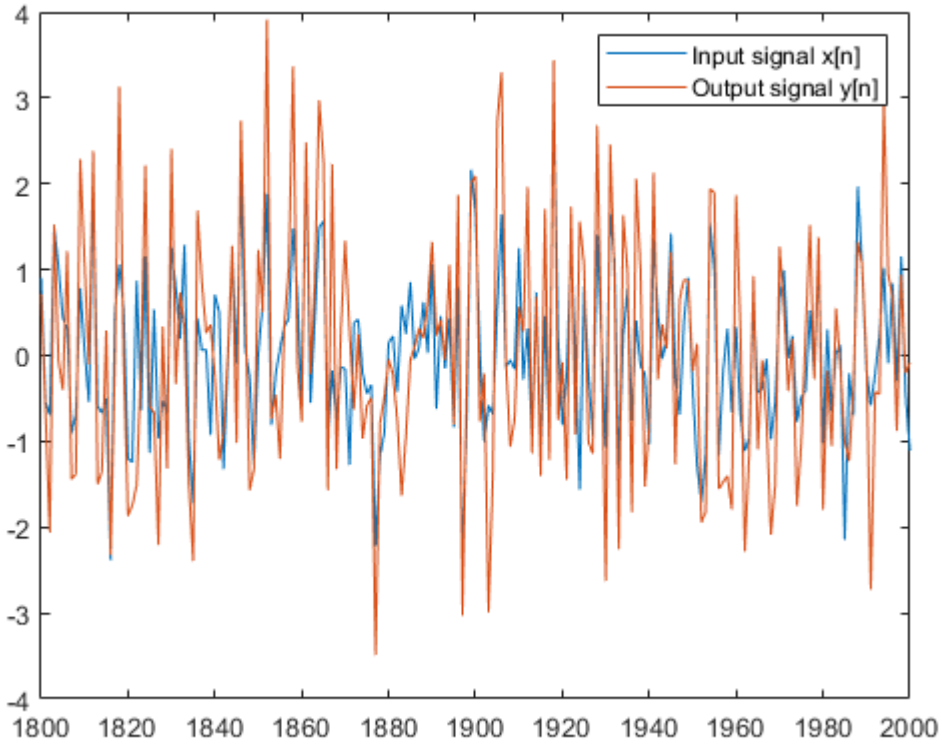
We see more coherence in the high frequencies than the low frequencies. Research why?

Comb Filter

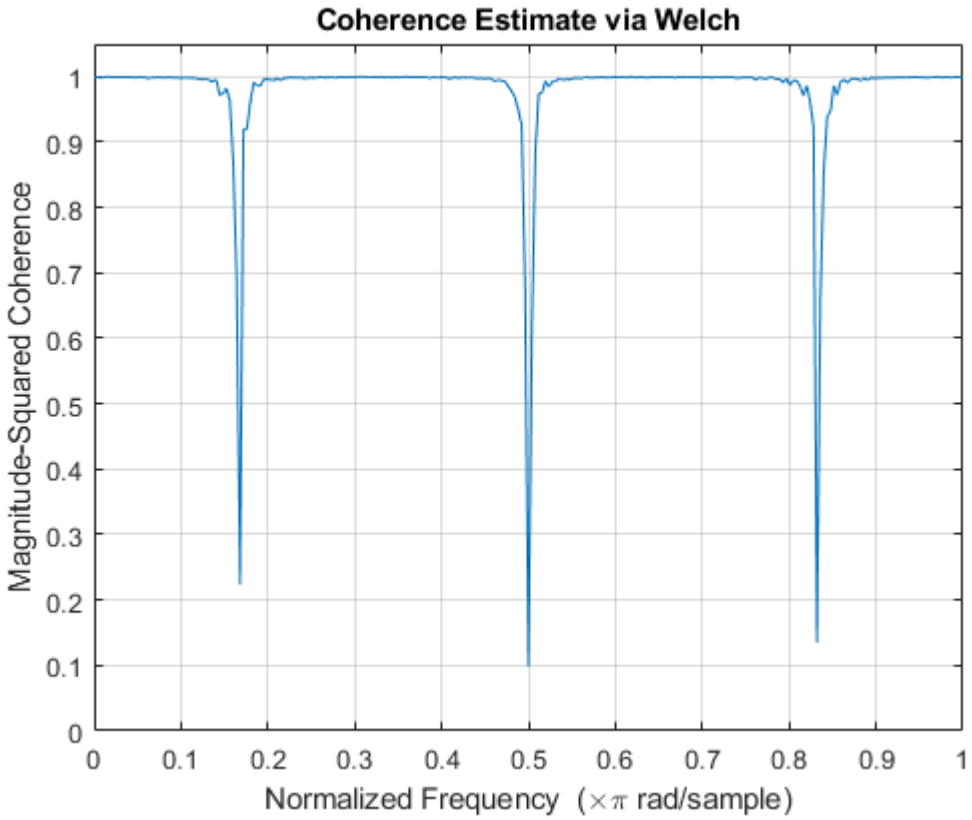
Let us defined a comb filter which has the difference equation:

$$y[n] = b_0 x[n] + b_M x[n - M]$$

```
% Define a comb filter
b = [1 0 0 0 0 0 1];
a = 1;
y = filter(b, a, x);
plot(n, x(n), n, y(n));
legend('Input signal x[n]', 'Output signal y[n]')
```



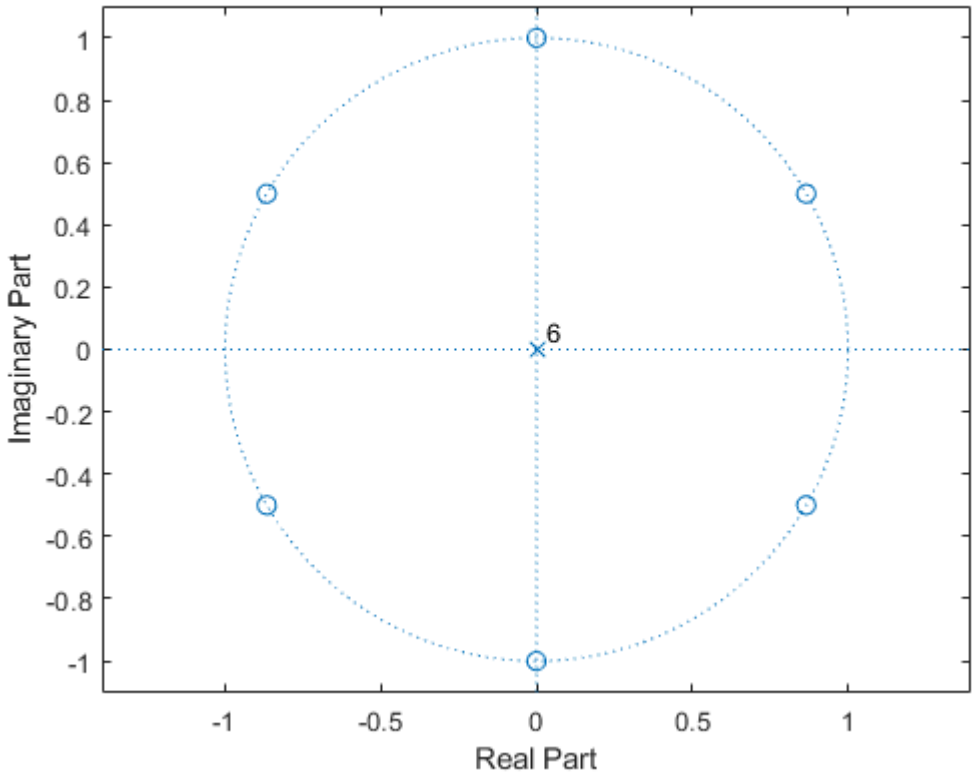

```
% Compute and plot the coherence estimate
mscohere(y, x);
ylim([0, 1.05]);
```



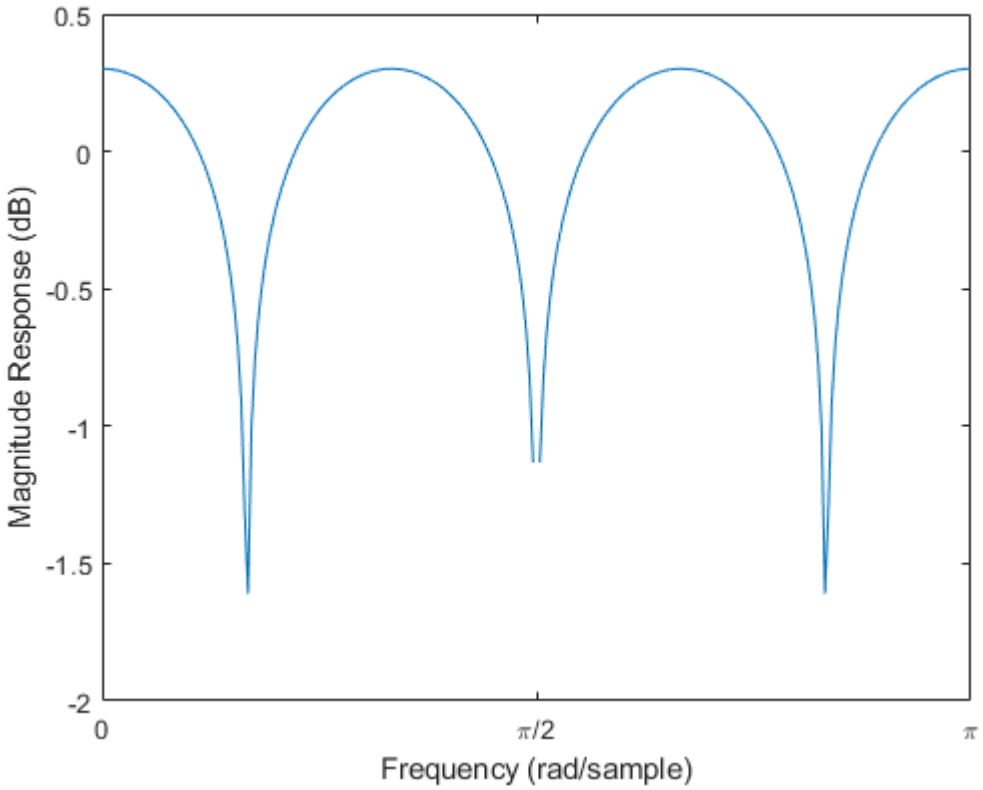
Notice that the coherence function dips at certain frequencies. This is because the comb filter removes certain frequency components of the input signal. Thefore, there is no coherence between the input signal and the output signal at those specific frequencies since the output signal does not contain these filtered frequencies.

The filter has 6 zeros on the unit circle, which means that it is removing 6/2=3 frequency components:

```
zplane(b, a)
```




```
% Plot the magnitude response to see which frequency
% components are being attenuated
[H, w] = freqz(b, a, 'whole');
plot(w, log10(abs(H)));
set(gca,'XTick',0:pi/2:2*pi)
set(gca,'XTickLabel',{'0','\pi/2','\pi','3\pi/2','2\pi'})
xlabel('Frequency (rad/sample)')
ylabel('Magnitude Response (dB)')
xlim([0, pi]);
```



Problems

Can Leakage be reduced in estimates of PSD by applying a window function to the data?

3. Leakage can be reduced in estimates of power spectral density by applying a window function to the data.

Answer: TRUE.

Leakage is an artefact of an FFT applied to a non-periodic data. Even if a signal is periodic like a sinousoid, when the signal is measured with some interval it can be non-periodic. The issue is that FFT assumes that the signal is periodic and repeats itself after the measured interval. This is not the case when the data is non-periodic, which contains sharp transitions at the end of each measured interval. These sharp changes have a broad frequency response which lead to spectral leakage. Windows can help *minimize* the effects of leakage by smoothing the time domain signal, but cannot eliminate leakage.

Estimate PSD from an ACRS assuming MA(1) process

The autocorrelation function for a MA(1) process has been found to be

$\|l\|$	$r_{yy}(l)$
0	2
1	1
$\|l\| \geq 2$	0

```
clear variables;
```

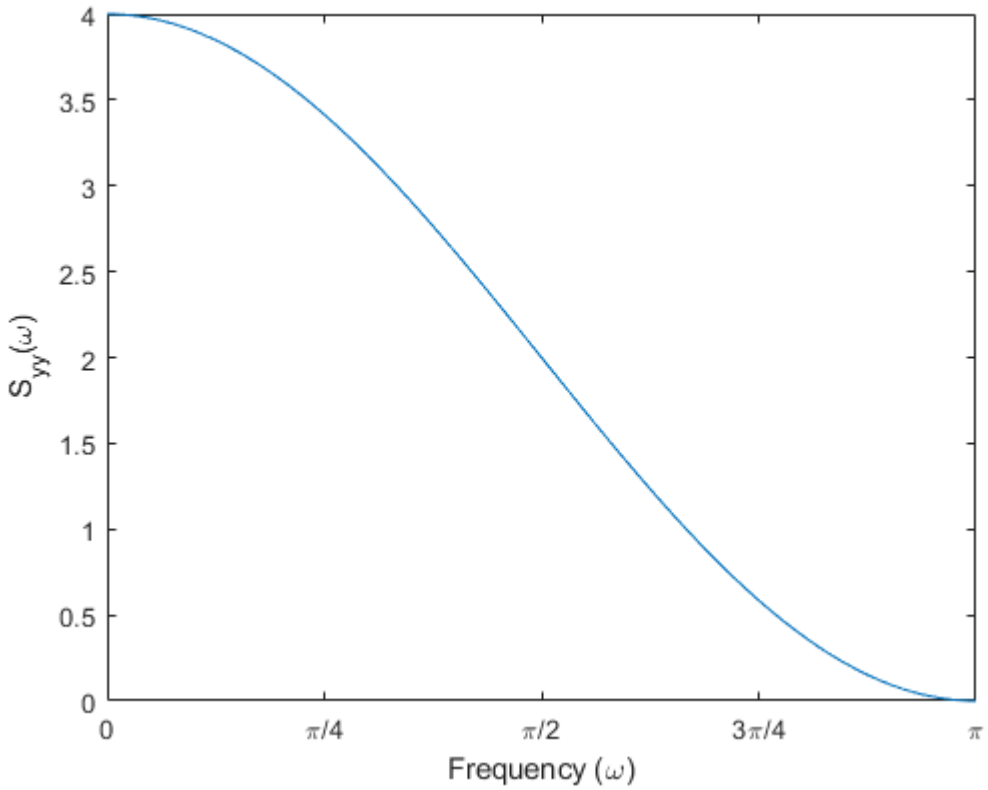
The PSD for an MA(q) process can be computed using Eq. (13.119):

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

Plugging in our values, we get:

$$S_{yy}(\omega) = 2 + 2(1\cos(1\omega)) = 2 + 2\cos(\omega)$$

```
w = 0:0.001:2*pi;
r_xx = [2, 1, 0];
S = r_xx(1);
for l = 2:numel(r_xx)
    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])
```



Estimate PSD from an ACRS assuming MA(2) process

For a given random process {*x*(*n*)} the autocorrelation has been estimated and is given by:

$\lvert m\rvert$	$r_x(m)$
0	4
1	2
2	-1

Estimate the power density spectrum under the assumption that {*x*(*n*)} can be described as a *MA*(2) process.

```
clear variables;
```

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

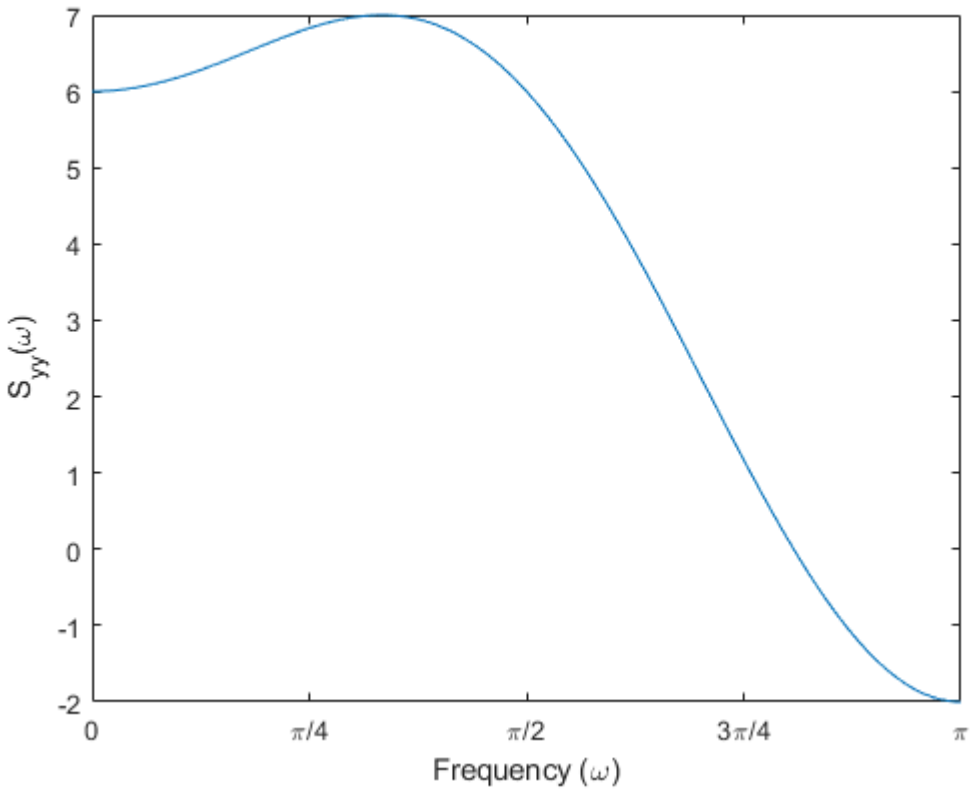
$$S_{xx}(\omega) = r_{xx}[0] + 2\sum_{\ell=1}^{\infty} r_{xx}[\ell]\cos\omega\ell, \qquad (13.119)$$

This gives following expression:

$$S_{xx}(\omega) = 4 + 2(2\cos(w) + (-1)\cos(2\omega))$$

$$= 4 + 4\cos(w) - 2\cos(2\omega)$$

```
w=0:0.001:pi;
r_xx = [4, 2, -1];

S = r_xx(1);
for l = 2:numel(r_xx)
    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])
```



Quiz: Determine if two sinusoids have the same PSD?

Consider the following two signals

$$x_1(k) = A_1 \cos(\omega k + \phi) + A_2 \cos(2\omega k + \phi)$$
$$x_2(k) = A_1 \cos(\omega k + \phi) + A_2 \cos(2\omega k + \phi + \Delta\theta)$$

Where ϕ is uniformly distributed on $[0 : 2\pi]$ and $\Delta\theta$ is a fixed number.

Do the two signals have the same power density spectrum?

A: Yes
B: No
C: Not enough information given

The answer is A; the two signals have the power density spectrum.

In this quiz, we need to compare the PSD of the two signals; $x_1(k)$ and $x_2(k)$

The only difference between the two signals is that $x_2(k)$ is a fixed phase shift of $\Delta\theta$

In order to compare compute the PSD, we need to compute the autocorrelation of the signal.

Last week (see problem ADSI Problem 4.4.3) we computed the autocorrelation of a generic real sinusoid $x(n) = \mathrm{A}\cos(\omega n + \phi)$, which was:

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

From this result, we see that the phase does not matter. Therefore, the autocorrelation of $x_1(k)$ and $x_2(k)$ are the same.

This means that the corresponding PSDs are also the same.

Lesson: when you compute the PSD, we only know the amount of energy at each given frequency. But we have no clue about the phase, which is essentially lost.

Quiz: How do we measure if a system is LTI?

In this class, we always make the assumptions that our systems are LTI system, but how do we actually know if a real-world system is LTI.

How do we measure if a system is linear and time-invariant?

We can measure if a system is linear by checking the two linearity properties:

1) $T(u+v) = T(u) + T(v)$

- Feed the system with the sum of two signals $x_1(n) + x_2(n)$ and observe the output $y(n)$.
- Feed the system with $x_1(n)$ to record the output $y_1(n)$.
- Feed the system with $x_2(n)$ to record the output $y_2(n)$.
- Now, the output $y(n)$ should be the same as $y_1(n) + y_2(n)$

2) $T(A\,v) = A\,T(v)$

- Feed the system with a signal with certain amplitude A e.g. $A = 42$. Record the output $y(n)$
- Feed the system with the same signal but where the amplitude is 1. Record the output $z(n)$

- The two outputs should have following relations: $y(n) = 42 \cdot z(n)$

We can measure if a system is time-invariant as follows:

- Feed the system with a known signal $x(n)$ and measure the output $y(n)$
- Feed the system with the same signal but shifted $x(n-k)$ the output should also be shifted $y(n-k)$

A more elegant approach is to compute the coherence function:

$$|\gamma_{yx}(\omega)|^2 = \frac{|S_{yx}(\omega)|^2}{S_{yy}(\omega)S_{xx}(\omega)} \qquad \text{with } 0 \leq |\gamma_{yx}(\omega)|^2 \leq 1$$

For a linear system, the coherence function should be 1 or very close to it:

$$|\gamma_{yx}(\omega)|^2 = 1$$

[✓] Problem 14.57: Compare different PSD estimates using noise recorded on F-16

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.

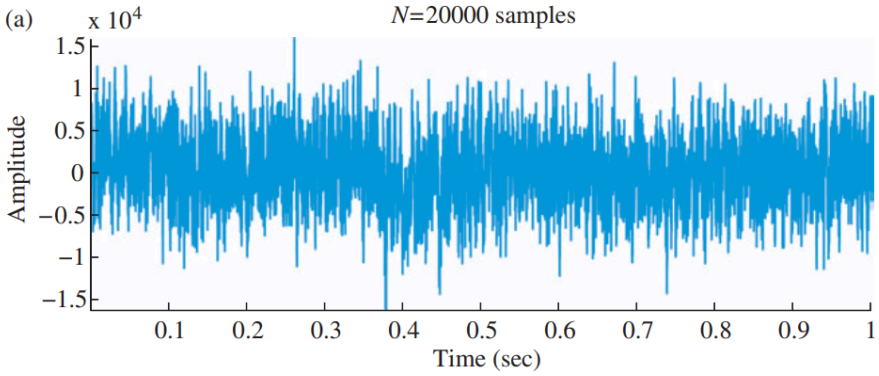


Figure (a) shows a waveform of F-16 noise recorded at the co-pilot's seat with a sampling rate of 19.98 kHz using a 16 bit ADC.

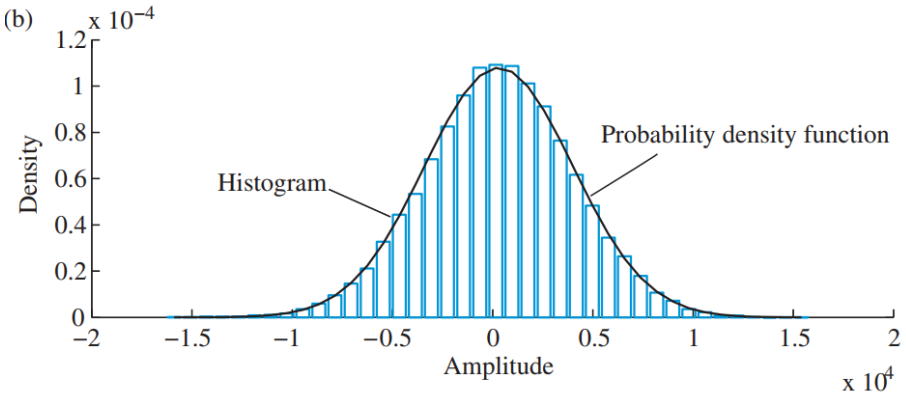
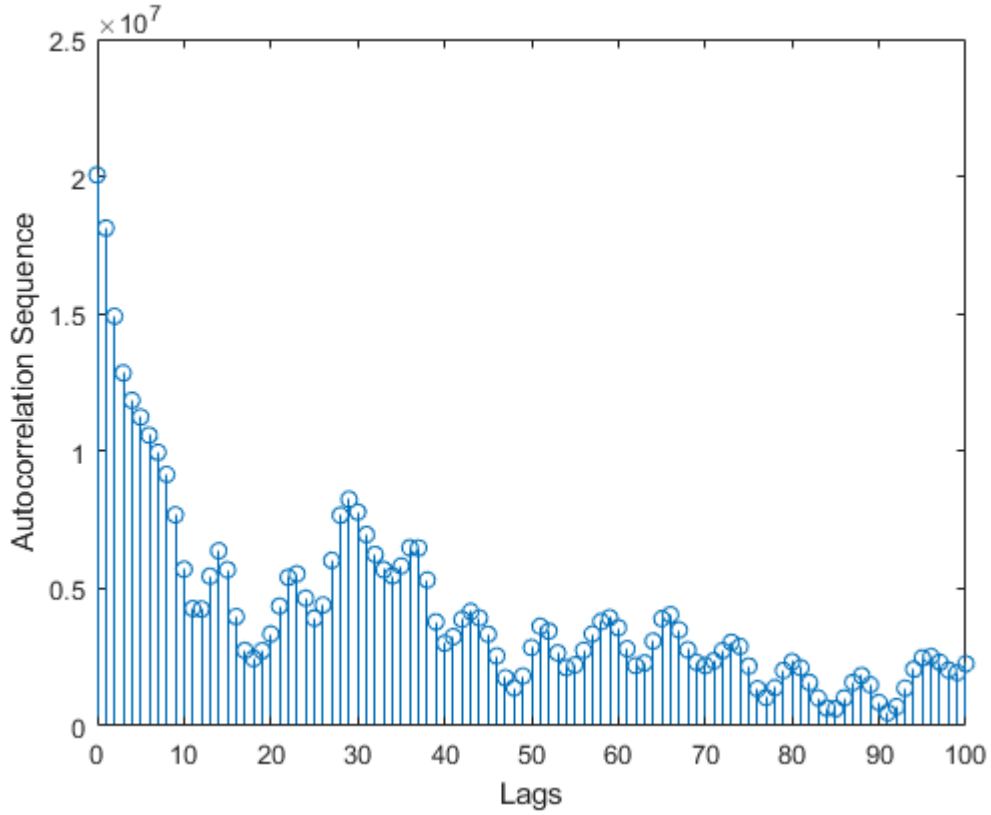


Figure (b) shows the histogram and theoretical probability density function of the F-16 noise.

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;
[lags, r_yy] = xcorr(y, L, 'biased');
mid = ceil(numel(r_yy)/2);
stem(r_yy(mid:end), lags(mid:end))
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.

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)
```

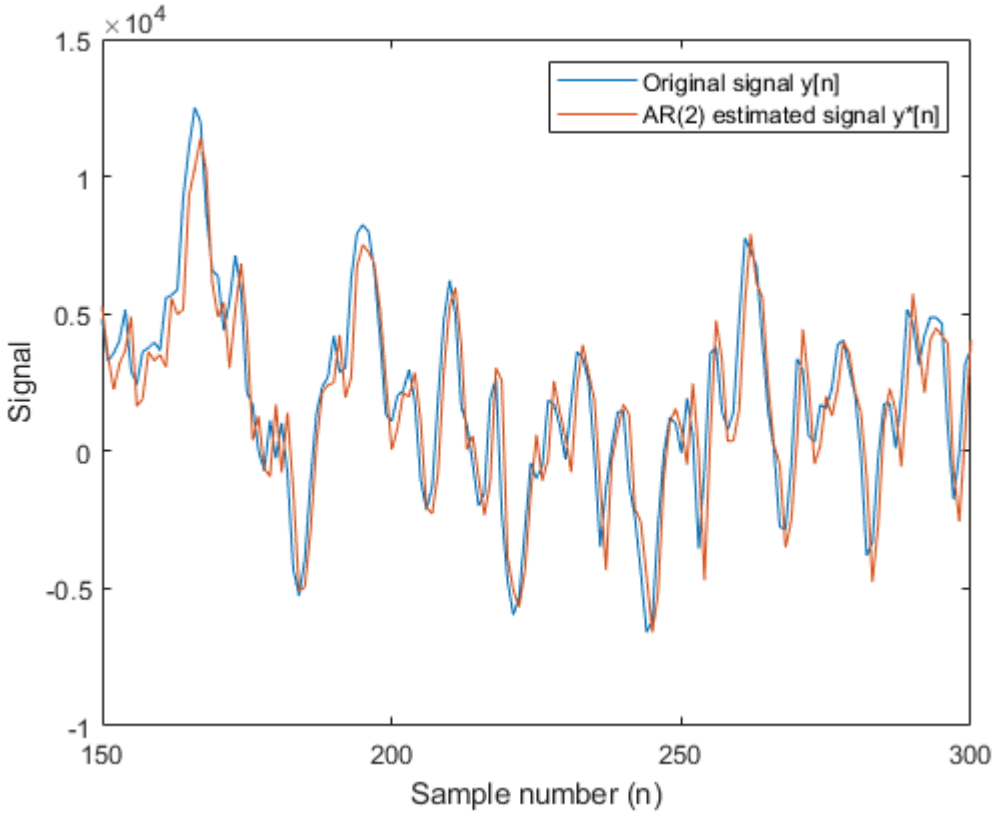
```
a = 2×1
   -1.2628
    0.3975
```

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;

% 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));
y_hat = y_hat + b0*x;


% 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
```

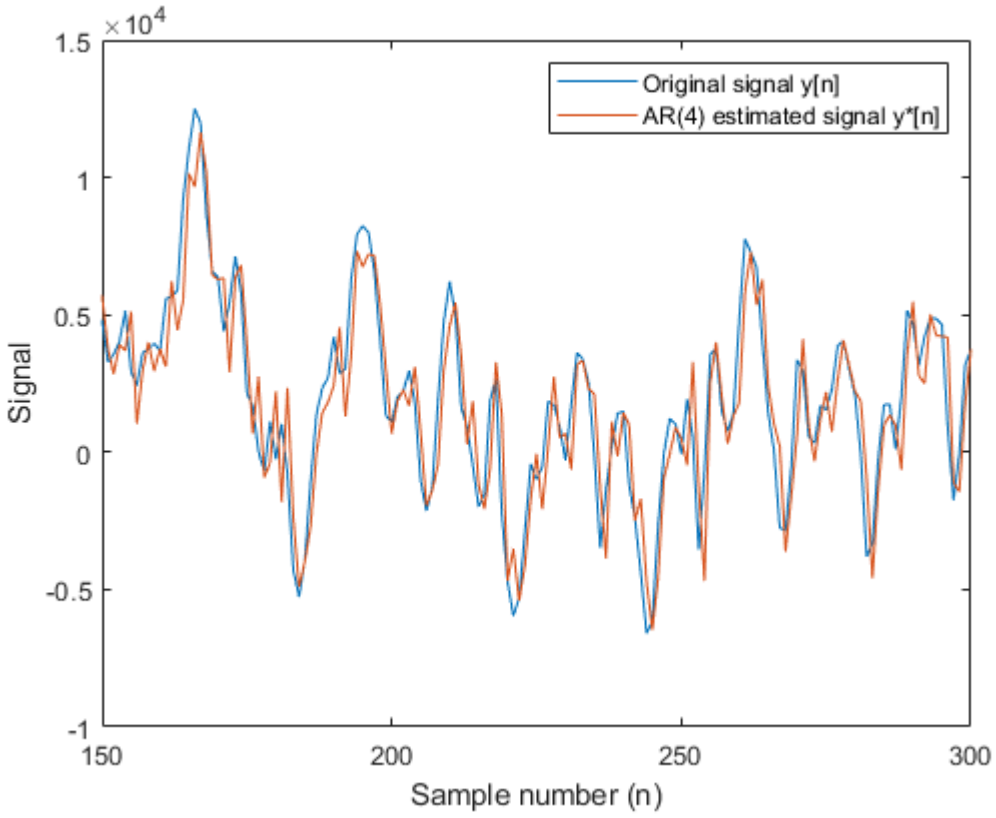



```
[a, v] = arfit(y, 4)
```

```
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:

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

The algorithm is as follows:

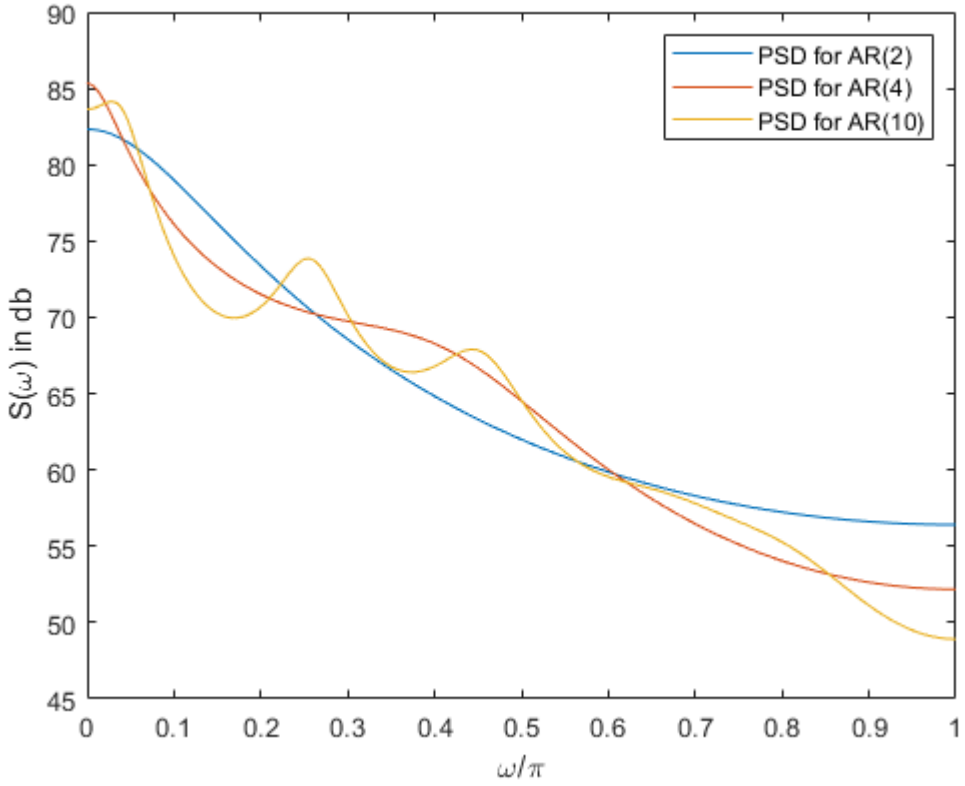
1. Find the coefficents $\{a_1, a_2, \cdots, a_p\}$ for the $\mathrm{AR}(p)$ model
2. Compute the transfer for the $\mathrm{AR}(p)$ by computing the sum and finding its reciprocal
3. Compute the conjugate of the transfer function: $|H(e^{j\omega})|^2$
4. Multiply it with the variance σ_x^2

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 $0.45\frac{\omega}{\pi}$.

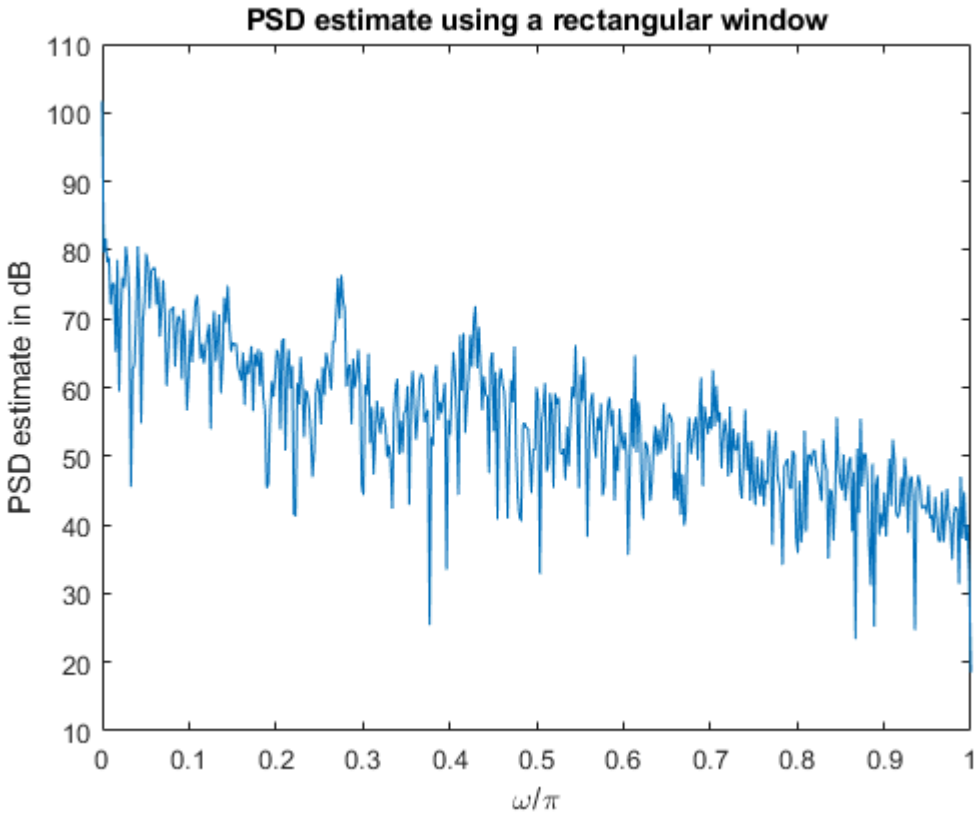
Why use AR(p) models to estimate the PSD? Because it is simple method if we assume that the given signal $y(n)$ is created by sending a white noise signal at an IIR filter. If this assumption is true, then we can model the signal $y(n)$ using the AR(p) model. The model parameters can be found by solving a linear problem which is easily solved. In comparison to the MA(q) models which yields nonlinear equations.

PSD estimate using AR(p) models does not reduce the uncertainty of whether to believe the estimate. For example, recomputing the PSD estimate on the next segment of the data, the autocorrelation will probably be slightly different. This means that the model parameters will be different which results in a different spectrum.

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.

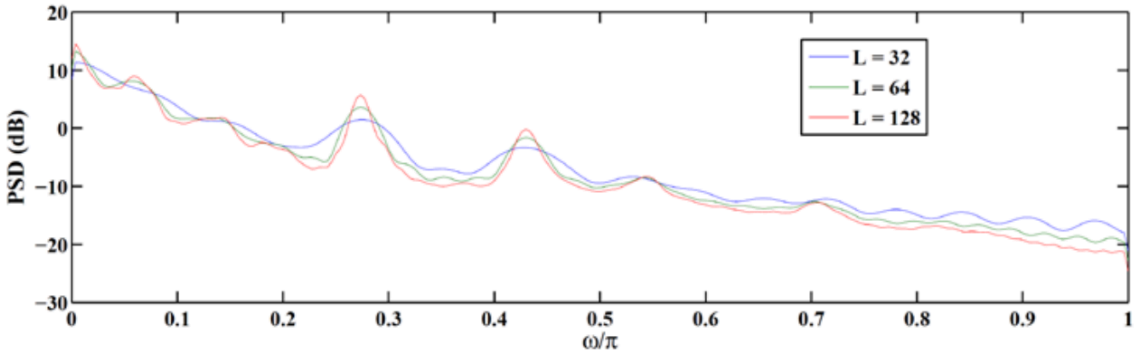
Notice some peaks at frequencies $0.27\frac{\omega}{\pi}$ and $0.42\frac{\omega}{\pi}$. Note how the periodogram has a large variance.

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.

```
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));
```



As we increase from 32 to 64, we observe a large change in the peaks. We increase the resolution by using longer windows. The window size determines the number of samples that we take the Fourier Transform on.

Going from 64 to 128, the change is less pronounced. It appears that it is saturated so thus a window size of 128 seems to be right size i.e., we have enough number of samples to estimate frequency peaks.

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);

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

As we increase the window size, we see peaks at the roughly the same frequencies as before. Whether these frequencies are real is difficult to say.

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);

plot(w, pow2db(I_L32), w, pow2db(I_L64), w, pow2db(I_L128));

legend('L=32', 'L=64', 'L=128')
title('PSD Estimates by the Welch method')
xlabel('\omega/\pi')
ylabel('I(\omega) in dB')
```

Again, we see that when the window size is 32, we get wider "peaks". As we increase the window size, these peaks become narrower. At window size 128, we cannot really say whether we have saturated.

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

LESSON: Now the question is, which PSD estimate should we believe in? Doing PSD estimation is a little bit like black art. It is really difficult to estimate PSD without knowing something about the process that generated the signal. Are some of the bumps in the PSD caused by the engine or the turbulence? In practice, we would record the sound from multiple places. This allows us to compare PSD estimates and compare energy.

[✓] 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 $v[n] \sim \text{WGN}(0, 2)$.

```
clear variables;
```

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 $\pm\omega_0$

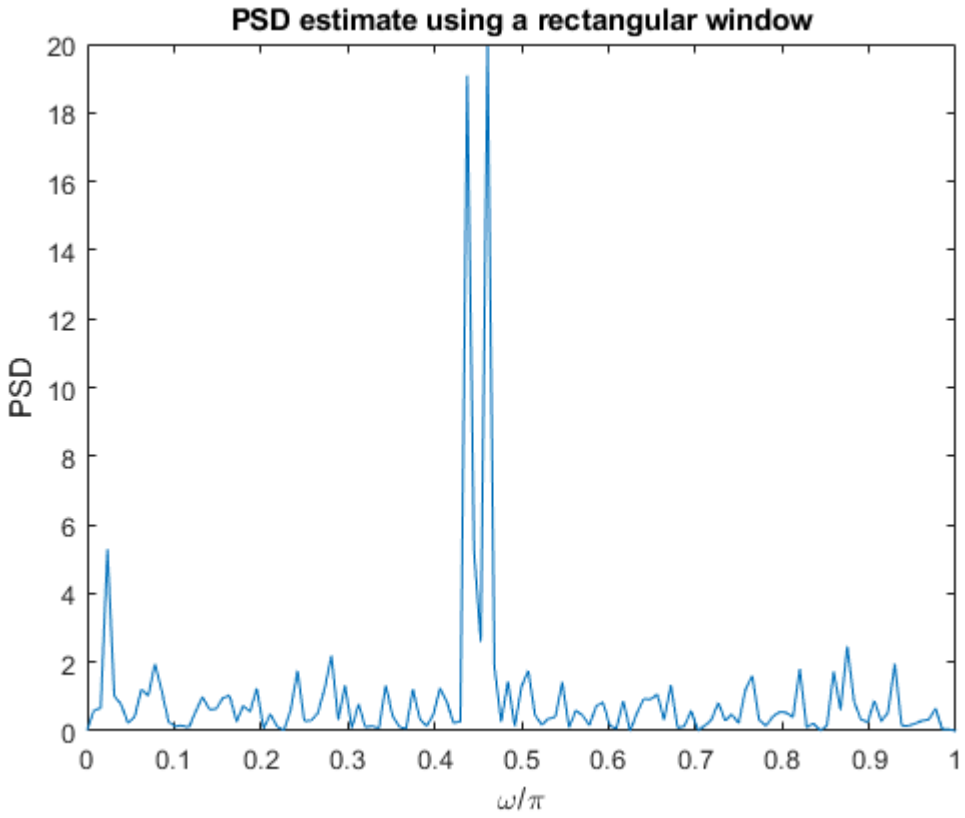
The autocorrelation of white Gaussian noise is $r_{vv}(\ell) = \sigma_v^2\delta(\ell) = 2\delta(\ell)$. The PSD of Gaussian noise is constant around σ_v^2.

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.

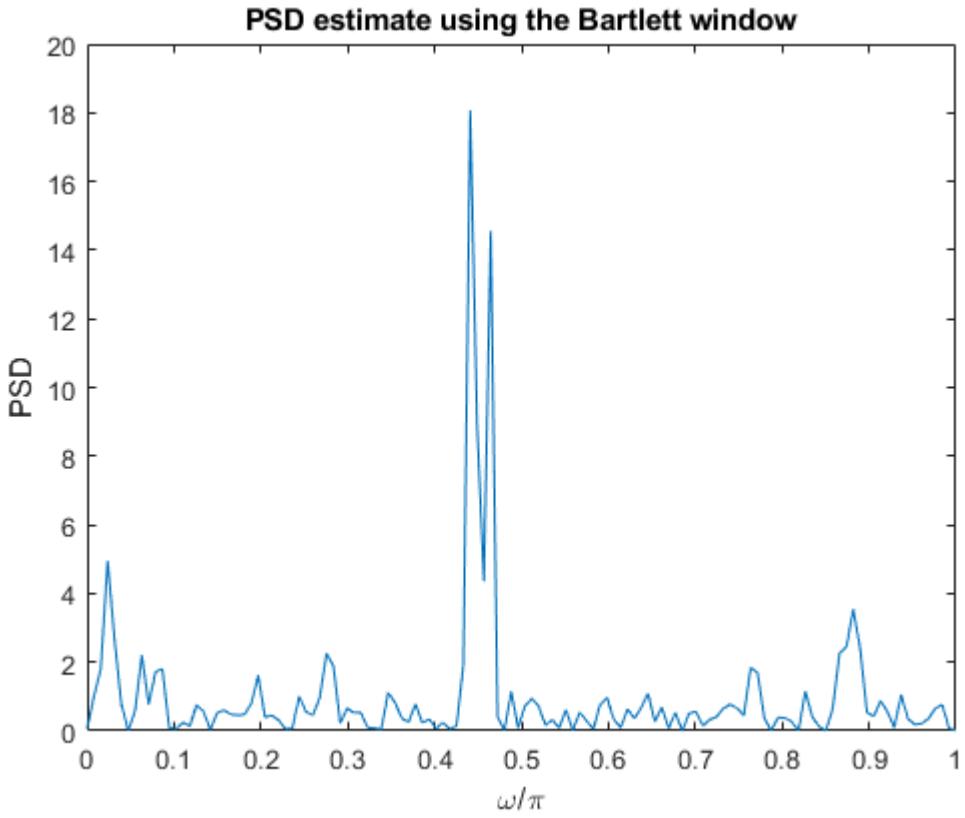
```
N = 256;
n = 0:N-1;
v = randn(1,N) * sqrt(2);
phi = 2*pi*(rand(1,2)-0.5);
x = cos(0.44*pi*n + phi(1)) + cos(0.46*pi*n + phi(2)) + v;
%plot(n, x); xlim([0, N]);

[I1, w] = periodogram(x, [], N);
plot(w/pi, I1);
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.

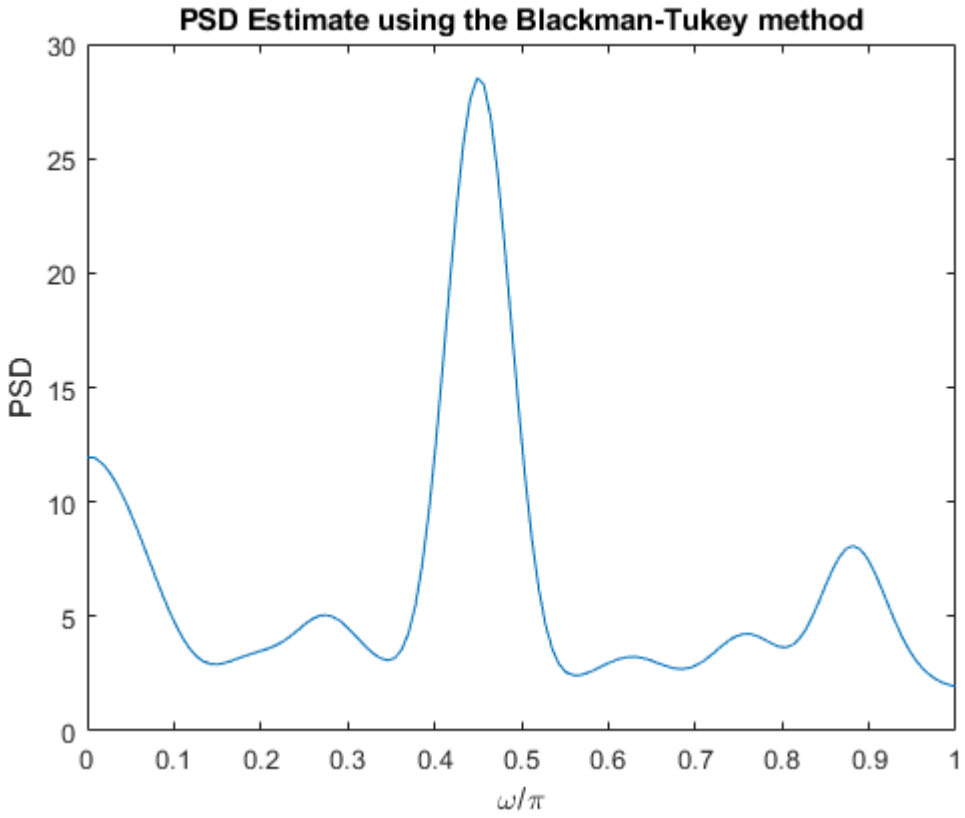
```
L = N;
I = periodogram(x, bartlett(L));
w = linspace(0,1,N/2)*pi;
plot(w/pi, I(1:N/2));
xlabel('\omega/\pi')
ylabel('PSD')
title('PSD estimate using the Bartlett window')
```



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.

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

ADSI Problem 4.7: Coherence function

The magnitude square coherence function of two signals $x[n]$ and $y[n]$ is given by

$$\gamma_{xy}^2(\omega) = \frac{|S_{xy}(\omega)|^2}{S_{xx}(\omega)S_{yy}(\omega)}.$$

Assume that $x[n]$ is the input to a given system, $y[n]$ is the output from the system, and the coherence function has been measured. What happens to the coherence function if

1) What happens to the coherence function if gain is increased from 1 to 2:

The gain setting of the amplifier in front of the ADC measuring $y[n]$ is increased from 1 to 2, i.e. $y[n] \rightarrow 2y[n]$?

To answer this question, we need to compute the new coherence function. Power Spectral Density $S_x(\omega)$ and Cross-Power Spectral Density $S_{yx}(\omega)$ are essentially the discrete-time Fourier Transform of the corresponding auto-correlation and cross-correlation.

So we need to compute those first. The autocorrelation for the input signal is the same because the input signal has not changed. We need to compute the autocorrelation for the new output signal.

Let $z(n) = 2y(n)$ denote the new output signal.

The autocorrelation for the new output signal $r_z(\ell)$ can be computed as follows:

$$r_z(\ell) = E[z(n)z(n-\ell)]$$

$$r_z(\ell) = E[2y(n)2y(n-\ell)]$$

$$r_z(\ell) = 4E[y(n)y(n-\ell)]$$

$$r_z(\ell) = 4r_y(\ell)$$

To find the PSD, we take the Fourier Transform:

$$S_z(\omega) = R_z(e^{j\omega}) = 4R_y(e^{j\omega}) = 4S_y(\omega)$$

This means that if the output gain is increased by 2, the PSD increases by a factor of 4.

Next, we need compute the cross-correlation $r_{zx}(\ell)$:

$$r_{zx}(\ell) = E[z(n)x(n-\ell)]$$

$$r_{zx}(\ell) = E[2y(n)x(n-\ell)]$$

$$r_{zx}(\ell) = 2E[y(n)x(n-\ell)]$$

$$r_{zx}(\ell) = 2r_{yx}(\ell)$$

To find the PSD, we take the Fourier Transform:

$$S_{zx}(\omega) = R_{zx}(e^{j\omega}) = 2R_{yx}(e^{j\omega}) = 2S_{yx}(\omega)$$

Finally, we can compute the coherence function:

$$|\gamma_{zx}(\omega)|^2 = \frac{|S_{zx}(\omega)|^2}{S_z(\omega)S_x(\omega)} = \frac{|2S_{yx}(\omega)|^2}{4S_y(\omega)S_x(\omega)} = \frac{|2|^2|S_{yx}(\omega)|^2}{4S_y(\omega)S_x(\omega)} = \frac{|S_{yx}(\omega)|^2}{S_y(\omega)S_x(\omega)}$$

The coherence function with the new output signal is the same the original coherence function.

This means that increasing the output gain by 2, nothing happens to the coherence function.

2) What happens to the coherence function if a delay happens in the measurement?

A delay happens in the measurement of $y[n]$ i.e. $y[n] \rightarrow y[n-D]$?

Let $z(n) = y(n-D)$

First, compute the autocorrelation for the new output signal $r_z(\ell)$:

$$r_z(\ell) = E[y(n-D)y(n-D-\ell)]$$

Let $m = n - D$ then we can write the above equation as:

$$r_z(\ell) = E[y(m)y(m-\ell)]$$

This expression is the same as $r_y(\ell)$ so the delayed output signal does not change the autocorrelation:

$$r_z(\ell) = r_y(\ell)$$

Next, we compute the cross-correlation:

$$r_{zx}(\ell) = E[z(n)x(n-\ell)]$$

$$r_{zx}(\ell) = E[y(n-D)x(n-\ell)]$$

$$r_{zx}(\ell) = E[y(n)x(n-\ell+D)]$$

$$r_{zx}(\ell) = r_{yx}(\ell+D)$$

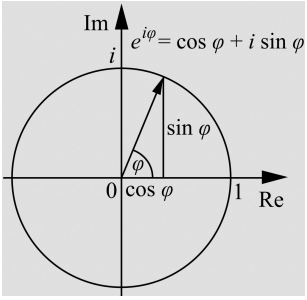
To find the Fourier Transform, we can use the Fourier pair: $f(t-t_0) \leftrightarrow F(e^{j\omega})e^{-j\omega t_0}$

$$R_{zx}(e^{j\omega}) = R_{yx}(e^{j\omega})e^{j\omega D} = S_{yx}(\omega)e^{j\omega D}$$

Finally, we can compute the coherence function:

$$|\gamma_{zx}(\omega)|^2 = \frac{|S_{zx}(\omega)|^2}{S_z(\omega)S_x(\omega)} = \frac{|e^{j\omega D}S_{yx}(\omega)|^2}{S_y(\omega)S_x(\omega)} = \frac{|e^{j\omega D}|^2\,|S_{yx}(\omega)|^2}{S_y(\omega)S_x(\omega)}$$

The norm of the $e^{j\omega D}$ is 1 because this quantity describes a point on the unit circle:



Since the square of 1 is 1, the coherence function is the same:

$$|\gamma_{zx}(\omega)|^2 = \frac{|S_{yx}(\omega)|^2}{S_y(\omega)S_x(\omega)}$$

This means that a delay in the measurement does not effect the coherence function!

[✓] 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.

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.

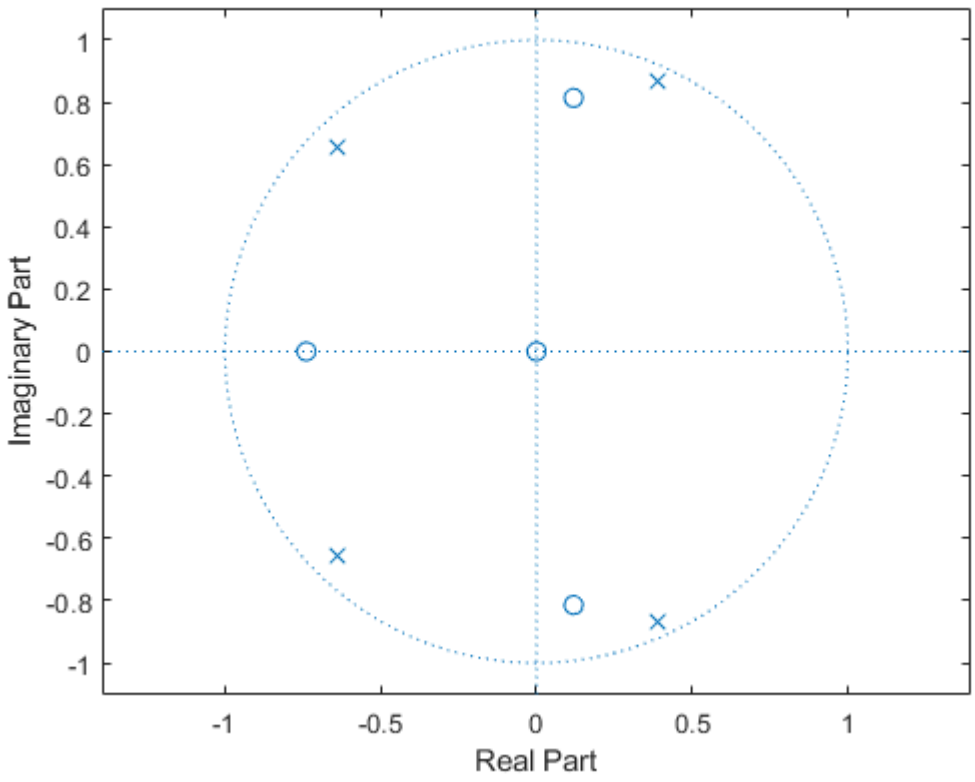
```
clear variables;
```

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.

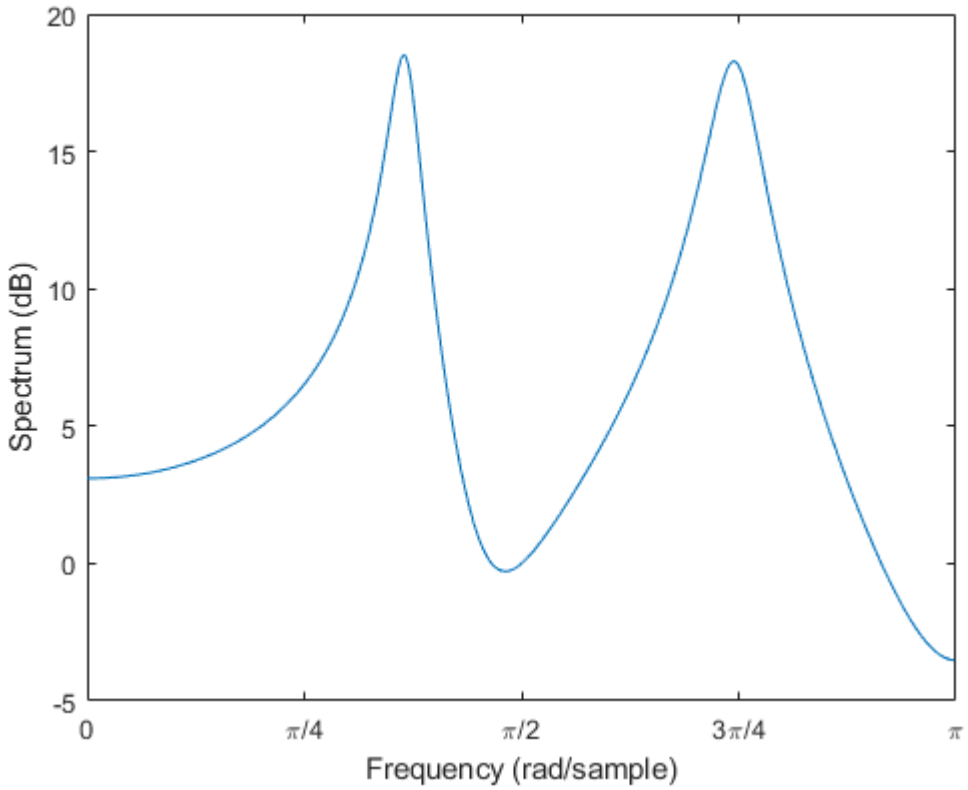
```
b = [2, 1, 1, 1];
a = [1, 0.5, 0.75, 0.5, 0.75];
zplane(b,a)
```



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:

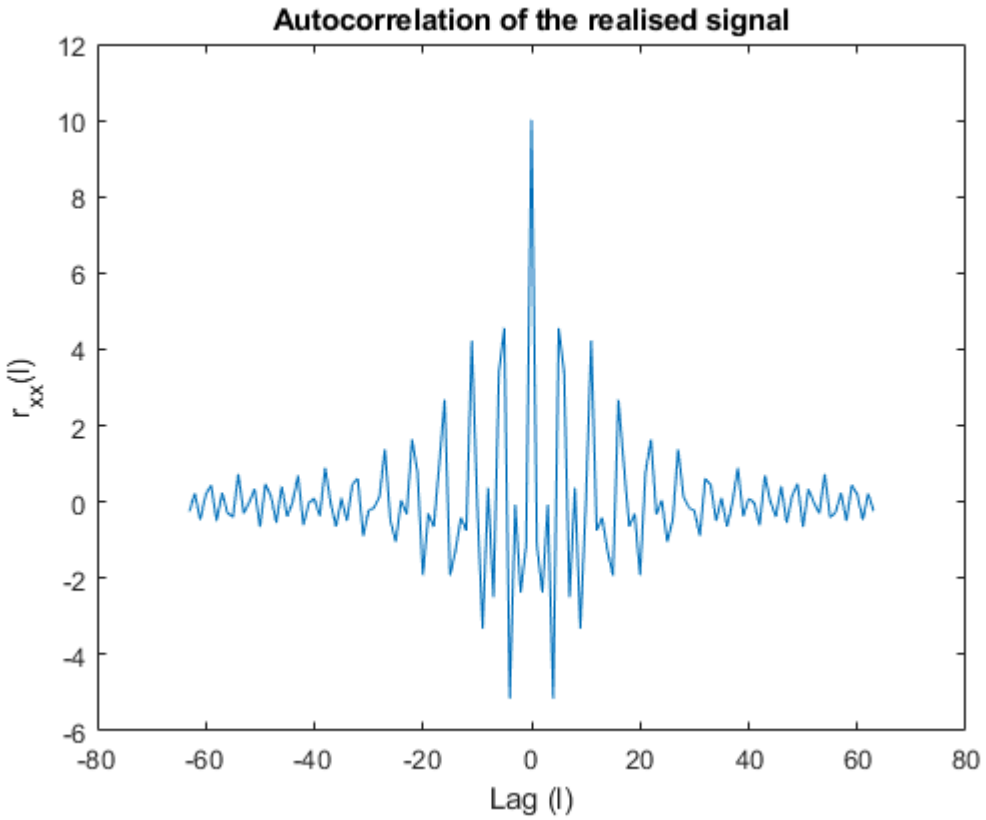
```
[H,w] = freqz(b, a);
plot(w, pow2db(H.*conj(H)));
set(gca,'XTick', 0:pi/4:pi)
set(gca,'XTickLabel', {'0','\pi/4','\pi/2','3\pi/4','\pi'})
xlabel('Frequency (rad/sample)')
ylabel('Spectrum (dB)')
xlim([0, pi]);
```



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
   10.0120   -1.1419   -2.3719   -0.0722   -5.1564    4.5521    3.4078   -2.4966 ···
   -1.1419   10.0120   -1.1419   -2.3719   -0.0722   -5.1564    4.5521    3.4078
   -2.3719   -1.1419   10.0120   -1.1419   -2.3719   -0.0722   -5.1564    4.5521
   -0.0722   -2.3719   -1.1419   10.0120   -1.1419   -2.3719   -0.0722   -5.1564
   -5.1564   -0.0722   -2.3719   -1.1419   10.0120   -1.1419   -2.3719   -0.0722
    4.5521   -5.1564   -0.0722   -2.3719   -1.1419   10.0120   -1.1419   -2.3719
    3.4078    4.5521   -5.1564   -0.0722   -2.3719   -1.1419   10.0120   -1.1419
   -2.4966    3.4078    4.5521   -5.1564   -0.0722   -2.3719   -1.1419   10.0120
    0.3697   -2.4966    3.4078    4.5521   -5.1564   -0.0722   -2.3719   -1.1419
   -3.3200    0.3697   -2.4966    3.4078    4.5521   -5.1564   -0.0722   -2.3719
     ⋮
```

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

```
R_xx_inv = 64×64
    0.2475   -0.0043    0.0598   -0.0225    0.1700   -0.1011   -0.0219   -0.0212 ···
   -0.0043    0.2475   -0.0053    0.0602   -0.0254    0.1718   -0.1007   -0.0215
    0.0598   -0.0053    0.2620   -0.0108    0.1013   -0.0499    0.1664   -0.1058
   -0.0225    0.0602   -0.0108    0.2640   -0.0263    0.1105   -0.0479    0.1683
    0.1700   -0.0254    0.1013   -0.0263    0.3808   -0.0957    0.0953   -0.0625
   -0.1011    0.1718   -0.0499    0.1105   -0.0957    0.4221   -0.0868    0.1039
   -0.0219   -0.1007    0.1664   -0.0479    0.0953   -0.0868    0.4238   -0.0851
   -0.0212   -0.0215   -0.1058    0.1683   -0.0625    0.1039   -0.0851    0.4255
    0.0654   -0.0222   -0.0058   -0.1118    0.2132   -0.0893    0.0979   -0.0908
   -0.0148    0.0656   -0.0258   -0.0044   -0.1220    0.2192   -0.0879    0.0992
```

$$\vdots$$

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:

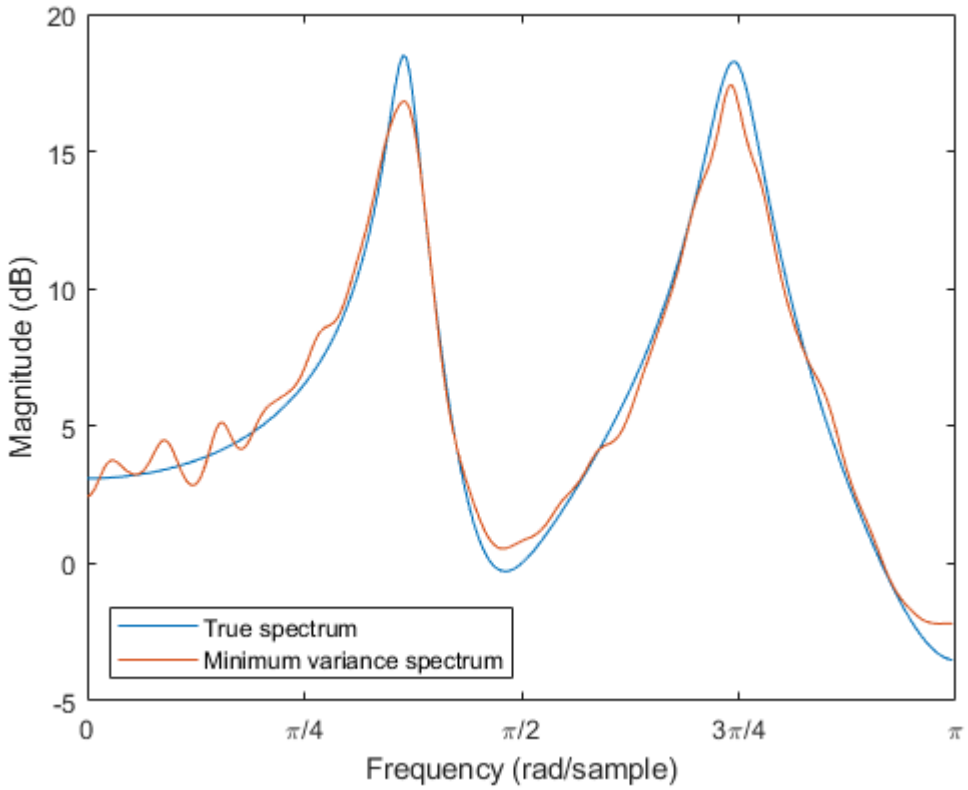
$$\mathbf{E}^t(f_l) = [1 \quad e^{j2\pi f_l} \quad \dots \quad e^{j2\pi p f_l}]$$

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

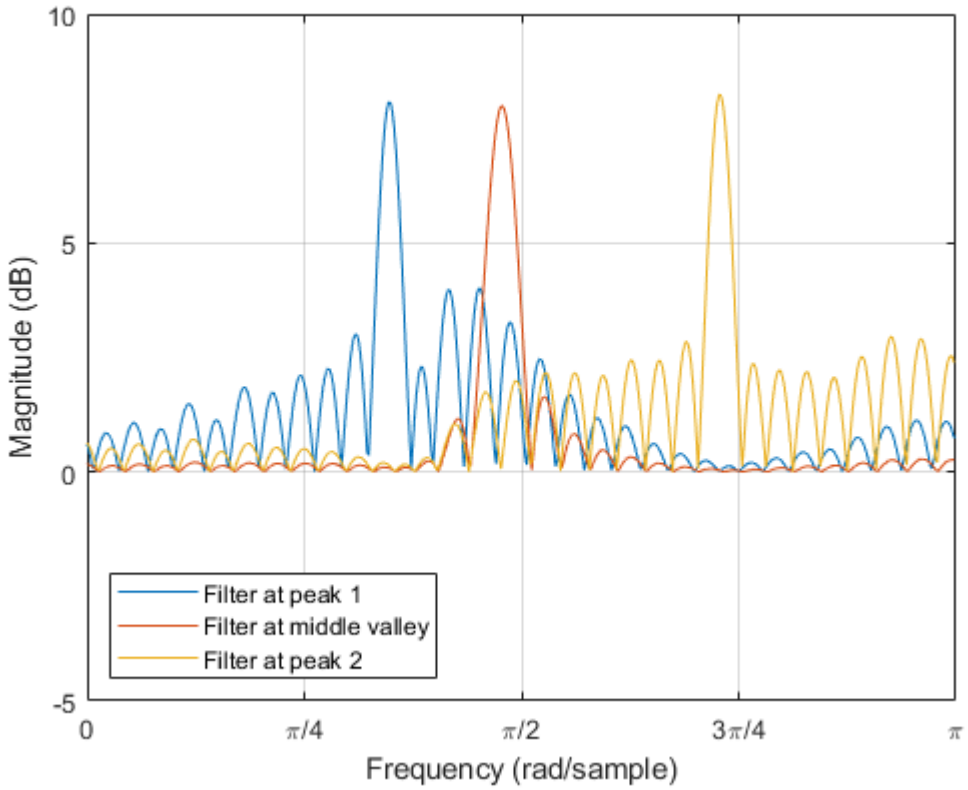
Choose a few representative frequencies in your spectrum. Calculate the optimum filters for estimation of these frequencies and plot the magnitude response of these. Are the filter magnitude responses in concordance with your intuition?

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;
ylim([-5, 10]);
xlim([0, pi]);
```



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

Problem 1: PSD from ACRS assuming AR(2) process

The first few values of the autocorrelation function of a random process has been determined as

$\|l\|$	$r_x(l)$
0	11.26
1	9.70
2	6.00
3	1.49
4	-2.53

Initially, it can be assumed that the random process can be modeled as an AR(2) process.

```
clear variables;
```

1) Compute AR(2) model coefficient given autocorrelation sequence

1. Calculate the model parameters and plot the power spectral density.

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 *Yule–Walker equations*:

$$\boldsymbol{R}_y \boldsymbol{a} = -\boldsymbol{r}_y. \qquad (13.143)$$

The input noise variance can be compute as follows:

$$\sigma_x^2 = \sigma_y^2 + \boldsymbol{a}^T \boldsymbol{r}_y = \sigma_y^2 - \boldsymbol{r}_y^T \boldsymbol{R}_y^{-1} \boldsymbol{r}_y \leq \sigma_y^2. \qquad (13.144)$$

This is coded in MATLAB function (see end of document):

```
p = 2;
r_xx = [11.26, 9.70, 6.00, 1.49, -2.53]';
[a, v] = ar_from_acrs(r_xx, p)
```

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:

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

To solve this problem, the method is as follows:

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

The method above is implemented in the function `ar2psd` (see at the end of document):

```
[S, w] = ar2psd(a, v, 256);
plot(w/pi, real(pow2db(S)))
legend('PSD for AR(2)')
xlabel('\omega/\pi')
ylabel('S(\omega) in db')
```

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
```