EE434
Biomedical Signal Processing
Lecture # 5

I would like to thank

Professor Robi Polikar for using his lecture notes.

Random Signal Analysis
&
Spectral Estimation

Outline

- Definition & characterization of **random signals**
  - Autocorrelation & Auto covariance functions
  - Stationarity
  - Ergodicity
- Transform domain representation of **random signals**
  - **Power density spectrum**
- Power spectrum density estimation:
  - Non-parametric techniques:
    - Periodogram
    - Welch's method
  - Parametric techniques:
    - ARMA, AR and MA models
    - Levinson – Durbin Recursion

Random Signals

- Most signals we have worked with so far are **deterministic signals**, because their value can be determined for any time instance n. They are given as functions of time, using an analytical expressions, e.g. $\sin(2\pi f_0 n)$.
- Many signals in real world cannot be expressed as an analytical function of time, and hence their values cannot be predicted ahead of time. Such signals are known as **random signals**, and the process that generate them are called **random processes**.
  - Speech signals, radar signals, biological signals, video signals, audio signals, etc. are all examples of **random signals**.

Random Signals

- **Random signals** (or **processes**) are usually modeled as an ensemble of signals $\{X[n]\}$.
  - One particular sequence $\{x[n]\}$ in this collection is called a **realization** of the **random process**.
  - At any given time index n, the observed value $x[n]$ is the value taken by the random variable, $X[n]$. Thus a **random process** is a family of random variables, $\{X[n]\}$.

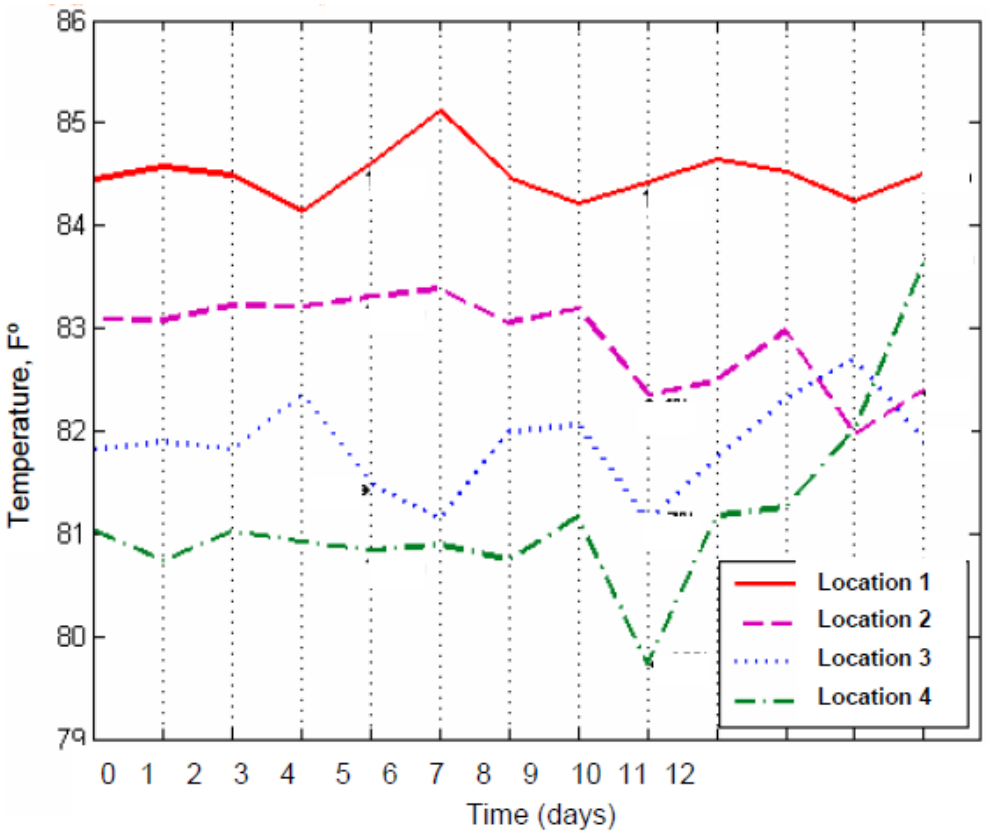
Random Process

Consider four thermometers placed at four close-by locations, monitoring the temperature in a plant. Each is a **random signal** (**realization**) of the **random process**.

Let's describe these sequences as $T(n,l)$, where T: Temperature, n: number of days (time), l:location.

We have the following interpretations: $T(n,l)$ is a

- **random variable** (r.v.) if n is fixed, and l is variable
- **sample sequence**, or **realization**, if n is variable, but l is fixed
- just **a number**, if both n and l are fixed
- **stochastic process**, if both are variables!



- **Random signals** are described by their statistical properties. For example, the **mean** or **expected value** of $\{X[n]\}$ is

$$\mu_X[n] = E(X[n]) = \int_{-\infty}^{\infty} x[n] \cdot p(x[n]) dx[n] = \int_{-\infty}^{\infty} x \cdot p(x) dx$$

  - Note that the **mean value** itself is a sequence: $\mu[n]$, $\mu[n\text{-}1]$, $\mu[n\text{-}2]$, …

- When we don't know the **pdf** – and we often don't, we estimate by:

$$\hat{\mu}_X[n] = \lim_{N\to\infty} \left\{ \frac{1}{N} \sum_{i=1}^{N} x_i[n] \right\}$$

i^{th} realization of the random process

N: number of realizations

- The **mean square value** of $\{X[n]\}$ at time n is

$$E\left(X[n]^2\right) = \int_{-\infty}^{\infty} x^2 \cdot p(x) dx$$

- And its **variance** is

$$\sigma_X^2[n] = E\left[(X-\mu_X)^2\right] = E\left[\{X[n]-\mu_X[n]\}^2\right] = \int_{-\infty}^{\infty} (x-\mu_X)^2 \cdot p(x)\, dx = E\left(X[n]^2\right) - [\mu_X[n]]^2$$

Autocorrelation Function

- In analysis of **random sequence**, an important entity is the two-dimensional **autocorrelation sequence**, which tells us how much two time instances of the signal are correlated with each other:

$$\phi_{XX}[m,n] = E\left(X[m]X^*[n]\right) \quad \text{or} \quad \phi_{XX}[n,n-k] = E\left(X[n]X^*[n-k]\right)$$

- To understand this sequence, recall that the random process is really an ensemble of sequences, $x_1[n]$, $x_2[n]$, …etc. Let's fix the value of n to say n_0, and obtain

$$\begin{aligned} &x_1[n_0]x_1^*[n_0-k], \quad k=0,\pm1,\pm2,\ldots \\ &x_2[n_0]x_2^*[n_0-k], \quad k=0,\pm1,\pm2,\ldots \\ &\vdots \end{aligned}$$

Autocorrelation Function

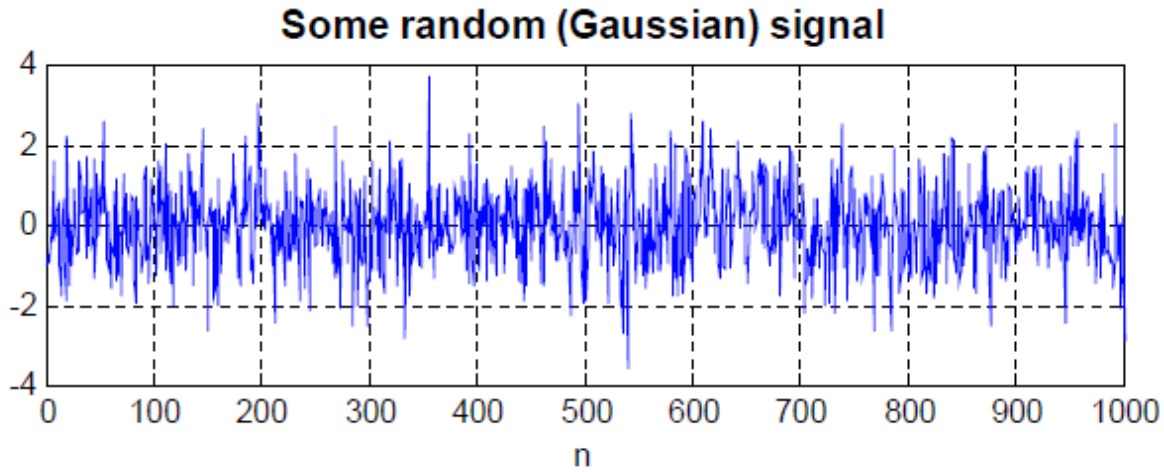
- The **mean** of these sequences give us $\phi_{XX}[n_0, n_0\text{-}k]$, for a particular value of n_0. By repeating it for all values of n, we obtain the **autocorrelation function**

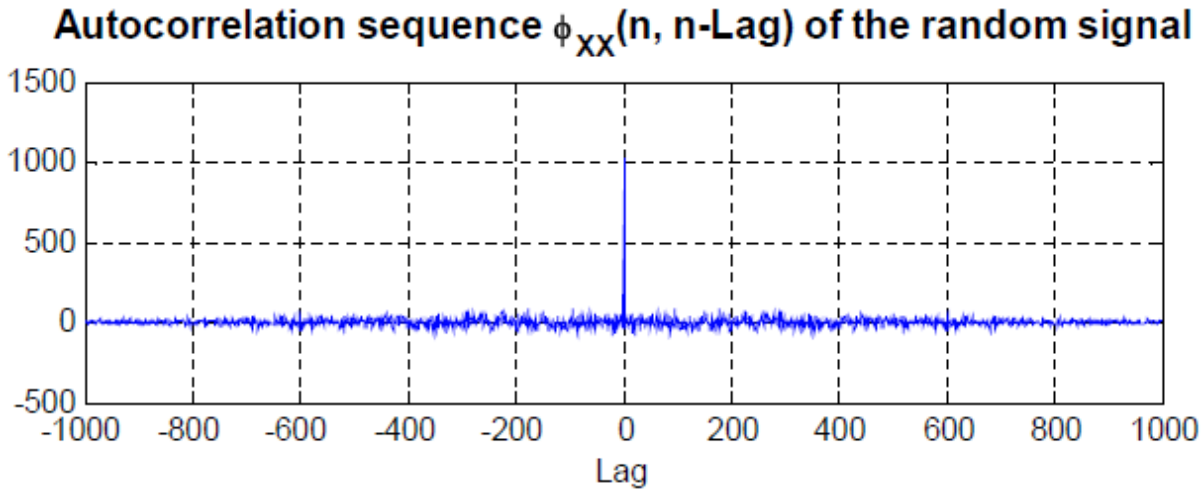
- $\phi_{XX}[n_0, n_0\text{-}k]$ tells us how much the random signal at time n_0 is correlated with itself after k delayed time steps (also called lags), or more generally, $\phi_{XX}[m,n]$ tells us how much the signal's m^{th} time step and n^{th} step are correlated with each other.

- When plotted as a function of lags, the **autocorrelation function** becomes 1-D

Autocorrelation Function

```
n=1:1000;
ww = randn(1000,1);
[c_ww,lags] = xcorr(ww);

subplot(211)
plot(n, ww);
grid
title('Some random (Gaussian) signal')
xlabel('n')

subplot(212)
plot(lags,c_ww)
grid
title('Autocorrelation sequence
R_{XX}(n, n-Lag) of the random signal')
xlabel('Lag')
```





What does this mean?

This **random signal** is so random that, it is only correlated with each other when k = 0, or at zero lag, or only when the signal overlaid on itself!

Autocovariance

- A related entity is the **autocovariance function**. Just like the autocorrelation function determines how much the sequence is correlated with itself (at different lags), the **autocovariance function** determines how much the sequence varies with itself.

- Technically, it is the mean-removed autocorrelation

$$\gamma_{XX}[m,n] = E\left(\left(X[m]-\mu_X[m]\right)\left(X[n]-\mu_X[n]\right)^*\right)$$
$$= \phi_{XX}[m,n]-\mu_X[m]\mu_X[n]$$

or

$$\gamma_{XX}[n,n-k] = E\left(\left(X[n]-\mu_X[n]\right)\left(X[n-k]-\mu_X[n-k]\right)^*\right)$$

- If $m = n$, then we get the **variance** of the function

$$\gamma_{XX}[n,n] = E\left[\left(X[m]-\mu_X[n]\right)^2\right] = E\left[X^2[n]\right]-\mu_X^2[n]$$

- If the mean is zero, then the autocorrelation and **variance** are the same!

Cross-Correlation & Cross-Covariance

- Instead of calculating the correlation and covariance of the sequence with shifted versions of itself, we can calculate correlation and covariance of two different sequences. These entities are called, **cross-correlation** and **cross-covariance**, respectively.

$$\phi_{XY}[m,n] = E\left(X[m]Y^*[n]\right) \quad \text{or} \quad \phi_{XY}[n,n-k] = E\left(X[n]Y^*[n-k]\right)$$

and

$$\gamma_{XY}[m,n] = E\left(\left(X[m]-\mu_X[m]\right)\left(Y[n]-\mu_Y[n]\right)^*\right) = \phi_{XY}[m,n]-\mu_X[m]\mu_Y[n] \quad \text{or}$$

$$\gamma_{XY}[n,n-k] = E\left(\left(X[n]-\mu_X[n]\right)\left(Y[n-k]-\mu_Y[n-k]\right)^*\right)$$

We will soon see that these functions become useful in analyzing the random signals in the frequency domain

Stationarity

- In general, all of the previously calculated sequences are functions of time, i.e., the quantities computed (mean, variance, auto-covariance, etc.) vary in time.
- If the PDF of a random process does not change in time, such a process is called **stationary**, however, strict stationarity is practically impossible for all real world signals.
- The class of signals often encountered in DSP are so-called **wide-sense-stationary** (**WSS**) processes, for which these quantities are either independent of time, or at least, the time origin.

Stationarity

- For a **WSS** process:
  - $X[n]$, the mean $\mu_X[n]$ is independent of time, and hence it is constant, μ_X
  - The autocorrelation and autocovariance functions are also independent of time, m and n, but only dependent on the difference of time-indices, $l = m\text{-}n$. Hence they are 1-D functions of the lag between time m and time n.

$$\phi_{XX}[l] = \phi_{XX}[n \pm l, l] = E\left(X[n+l]X^*[n]\right)$$

$$\gamma_{XX}[l] = \gamma_{XX}[n \pm l, l] = E\left(\left(X[n+l]-\mu_X\right)\left(X[n]-\mu_X\right)^*\right)$$

$$= \phi_{XX}[l] - |\mu_X|^2$$

Special Cases

- For a **WSS** process, the following holds:

$$\phi_{XX}[0] = E\left(X[n]X^*[n]\right) = E\left(\left|X[n]\right|^2\right)$$

$$\sigma_X^2 = \gamma_{XX}[0] = \phi_{XX}[0] - \left|\mu_X\right|^2$$

- The mean-square value of a **WSS** process is its autocorrelation function evaluated at zero lag!
- The variance of a **WSS** process is its auto-covariance evaluated at zero lag!

- We can show that the following symmetry properties are also valid

$$\phi_{XX}[-l] = \phi_{XX}^*[l]$$
$$\gamma_{XX}[-l] = \gamma_{XX}^*[l]$$
$$\phi_{XY}[-l] = \phi_{XY}^*[l]$$
$$\gamma_{XY}[-l] = \gamma_{XY}^*[l]$$

These functions, the autocorrelation, auto-covariance, cross correlation and cross covariance, are all two-sided sequences with conjugate symmetry.
If the original process is purely real, then the conjugation can also be dropped.

$$\phi_{XX}[0]\phi_{YY}[0] \geq \left|\phi_{XY}[l]\right|^2$$
$$\gamma_{XX}[0]\gamma_{YY}[0] \geq \left|\gamma_{XY}[l]\right|^2$$
$$\phi_{XX}[0] \geq \left|\phi_{XX}[l]\right|$$
$$\gamma_{XX}[0] \geq \left|\gamma_{XX}[l]\right|$$

Both the autocorrelation and auto-covariance functions assume their maximum values at zero lag. This should make sense, why?

Ergodicity

- First recall: A random signal $x[n]$ is really one of an ensemble of signals that are generated by a random process. Every time we generate $x[n]$, we in fact get a different realization.

- If the statistical properties of these different realizations are not different, then such a random processes is said to be **stationary**. In such a case, the probability distribution function of the random process is independent of time.

  - This is a very restrictive property, not generally satisfied.

  - Leaving the mean independent of time, if we require that the correlation and covariance functions be dependent only on the time-difference (lag), then we have **wide-sense stationarity**.

Ergodicity

- Now, notice that we normally do not have access to all realizations, but rather just one.
  - If each realization behaves exactly as the ensemble, in other words, if we can obtain the statistical behaviors of the ensemble from just one single realization, such processes are called **ergodic processes**. Notice that only (wide sense) stationary signals can be **ergodic**.
- More formally, a (WS) stationary random signal is said to be **ergodic**, if all of its statistical properties can be estimated from a single realization of a sufficiently large finite length.

Ergodicity

- For ergodic signals, then

$$\mu_X[n] = \mu_X = \lim_{M\to\infty} \frac{1}{2M+1} \sum_{n=-M}^{M} x[n]$$

$$\sigma_X^2[n] = \sigma_X^2 = \lim_{M\to\infty} \frac{1}{2M+1} \sum_{n=-M}^{M} \left(x[n]-\mu_X\right)^2$$

$$\phi_{XX}[l] = \lim_{M\to\infty} \frac{1}{2M+1} \sum_{n=-M}^{M} \left(x[n]x[n-l]\right)$$

$$\gamma_{XX}[l] = \lim_{M\to\infty} \frac{1}{2M+1} \sum_{n=-M}^{M} \left(x[n]-\mu_X\right)\left(x[n-l]-\mu_X\right)$$

- Of course, since we also have access to a finite observation only, we would have to get rid of the limits, and calculate the above quantities for the duration of the signal available. In such a case, we only calculate their **unbiased estimates**.

$$\hat{\mu}_X[n] = \hat{\mu}_X = \frac{1}{M+1}\sum_{n=0}^{M} x[n] \qquad \hat{\phi}_{XX}[l] = \frac{1}{M+1}\sum_{n=0}^{M}\left(x[n]x[n-l]\right)$$

$$\hat{\sigma}_X^2[n] = \hat{\sigma}_X^2 = \frac{1}{M+1}\sum_{n=0}^{M}\left(x[n]-\mu_X\right)^2 \qquad \hat{\gamma}_{XX}[l] = \frac{1}{M+1}\sum_{n=0}^{M}\left(x[n]-\mu_X\right)\left(x[n-l]-\mu_X\right)$$

xcorr() Cross-correlation function estimates.

C = xcorr(X,Y), where X and Y are length M vectors (M > 1), returns the length 2*M-1 cross-correlation sequence C. If X and Y are of different length, the shortest one is zero-padded. C will be a row vector if X is a row vector, and a column vector if X is a column vector.

xcorr() produces an estimate of the correlation between two random (jointly stationary) sequences:

$$R_{XY}(m) = E\left[x_{n+m}y_n^*\right] = E\left[x_n y_{n-m}^*\right]$$

It is also the deterministic correlation between two deterministic signals.

xcorr(X), when X is a vector, is the auto-correlation sequence. xcorr(X), when X is an M-by-N matrix, is a large matrix with 2*M -1 rows whose N^2 columns contain the cross-correlation sequences for all combinations of the columns of X. The zeroth lag of the output correlation is in the middle of the sequence, at element or row M.

xcorr(...,MAXLAG) computes the (auto/cross) correlation over the range of lags: -MAXLAG to MAXLAG, i.e., 2*MAXLAG+1 lags. If missing, default is MAXLAG = M-1.

[C,LAGS] = xcorr (...) returns a vector of lag indices (LAGS).

xcorr(...,SCALEOPT), normalizes the correlation according to SCALEOPT:

'biased' - scales the raw cross-correlation by 1/M.

'unbiased' - scales the raw correlation by 1/(M-abs(lags)).

'coeff' - normalizes the sequence so that the auto-correlations at zero lag are identically 1.0.

'none' - no scaling (this is the default).

- The actual function calculated by Matlab is in fact

$$\hat{R}_{XY}(m) = \begin{cases} \sum_{n=0}^{N-m-1} x_{n+m} \cdot y_n^* & m \geq 0 \\ \hat{R}_{YX}^*(-m) & m < 0 \end{cases}$$

  - For the “biased” option,

$$R_{XY,biased} = \frac{1}{N} R_{XY}(m)$$

  - For the “unbiased” option

$$R_{XY,biased} = \frac{1}{N - |m|} R_{XY}(m)$$

xcov() Cross-covariance function estimates.

xcov(X,Y) where X and Y are length M vectors, returns the length 2*M -1 cross-covariance sequence in a column vector.

xcov(X) when X is a vector, is the auto-covariance sequence. xcov(X) when X is an M-by-N matrix, is a large matrix with 2*M -1 rows whose N^2 columns contain the cross-covariance sequences for all combinations of the columns of X. The zeroth lag of the output covariance is in the middle of the sequence, at element or row M.

The cross-covariance is the cross-correlation function of two sequences with their means removed:

$$\gamma_{XY}(m) = E\left[(x_{n+m} - \mu_X)(y_n - \mu_Y)^*\right]$$

where μ_X and μ_Y are the means of X and Y respectively.

xcov(...,MAXLAG) computes the (auto/cross) covariance over the range of lags: -MAXLAG to MAXLAG, i.e., 2*MAXLAG+1 lags. If missing, default is MAXLAG = M-1.

[C,LAGS] = xcov(...) returns a vector of lag indices (LAGS).

By default, xcov() returns the raw (unnormalized) covariances as follows:

$$\gamma_{XY}(m)=\begin{cases}\sum_{n-0}^{N-|m|-1}\left(x(n+m)-\frac{1}{N}\sum_{i=0}^{N-1}x_i\right)\left(y_n^*-\frac{1}{N}\sum_{i=0}^{N-1}y_i^*\right), & m>0\\ \gamma_{YX}^*(-m), & m<0\end{cases}$$

The output vector is then $c(m) = \gamma_{XY}(m\text{-}N)$, $m = 1,\ldots 2N\text{-}1$

xcov(...,SCALEOPT), normalizes the covariance according to SCALEOPT:

biased - scales the raw cross-covariance by $1/M$.

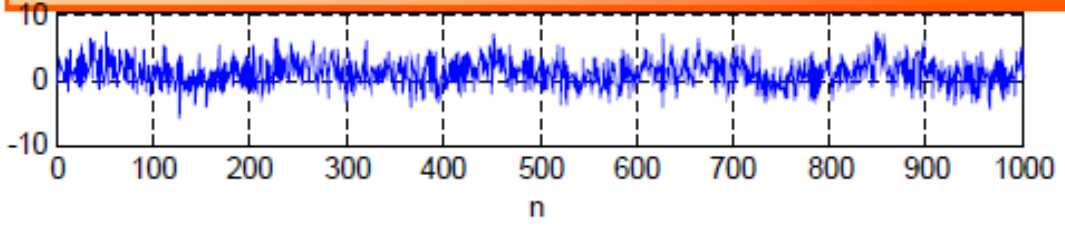
unbiased - scales the raw covariance by $1/(M\text{-abs}(k))$, where k is the index into the result.

coeff - normalizes the sequence so that the covariances at zero lag are identically 1.0.

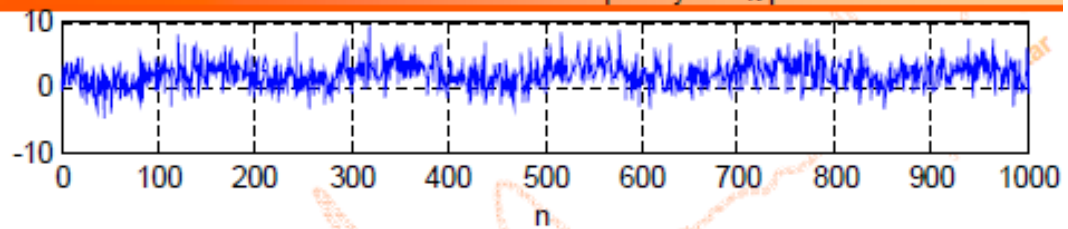
none - no scaling (this is the default).

Matlab Demo

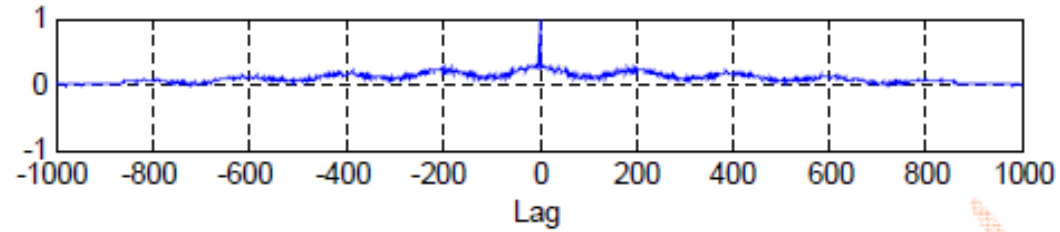
A sinusoid with AWGN



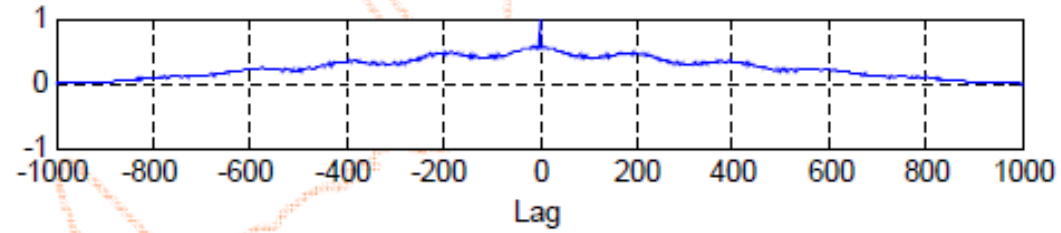
Another sinusoid of the same frequency but π phase shift



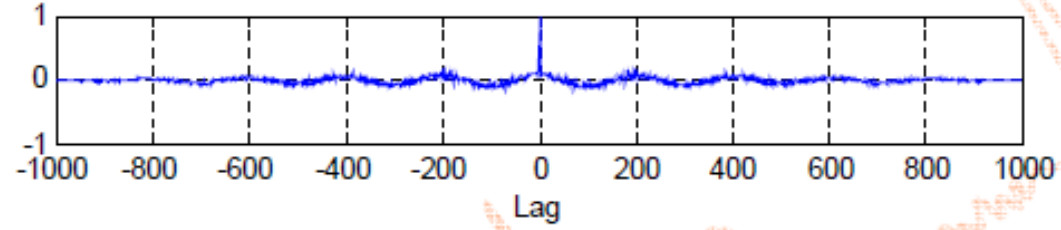
Autocorrelation sequence ϕ_{XX}(n, n-Lag) of this signal



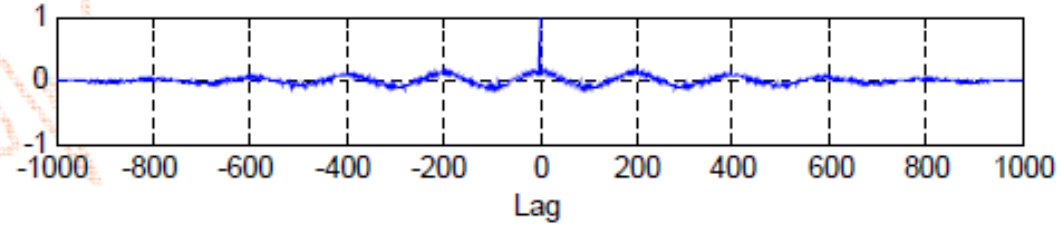
Autocorrelation sequence ϕ_{YY}(n, n-Lag) of this signal



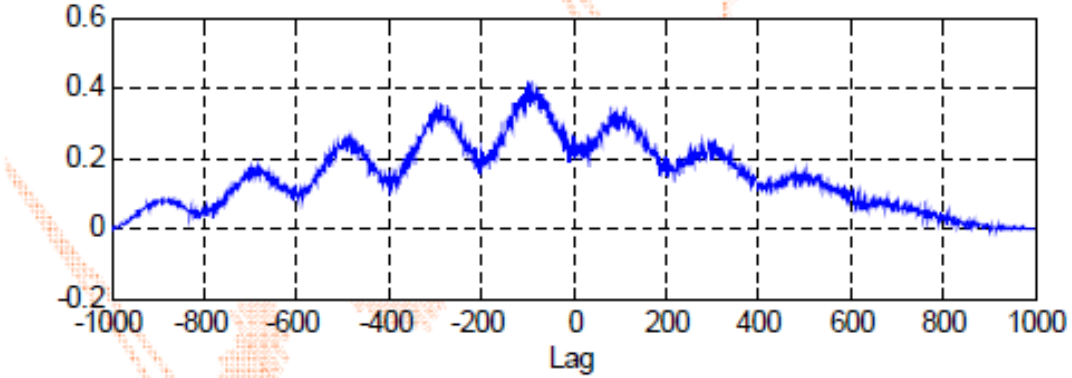
Autocovariance sequence γ_{XX}(n, n-Lag) of the random signal



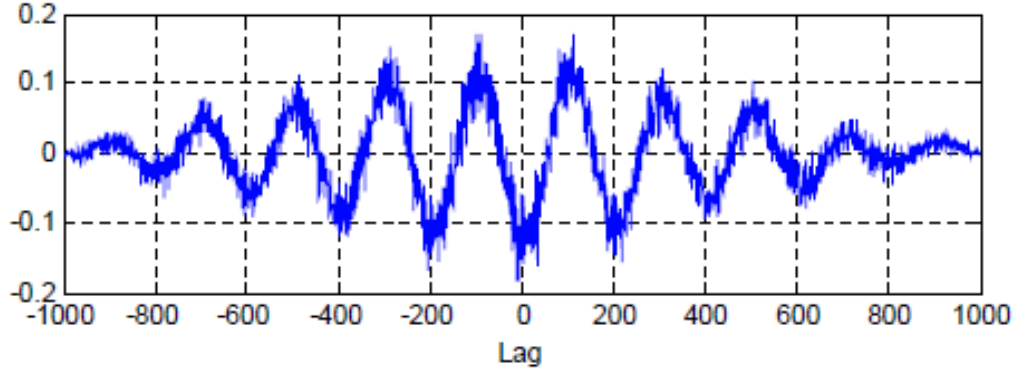
Autocovariance sequence γ_{YY}(n, n-Lag) of this signal



Crosscorrelation sequence ϕ_{XY}(n, n-Lag) of X and Y



Crosscovariance sequence γ_{XY}(n, n-Lag) of X and Y



- Now, suppose we would like to obtain the **frequency domain representation** of a random process
  - For example, we have a noise source, and we would like to find out the **frequency spectrum** of this noise generating process.
  - If $x[n]$ is the noise signal generated by this process, we might compute its **DTFT** to obtain $X(\omega)$, the spectrum of $x[n]$. Is it enough to achieve our goal?

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n}$$

- The answer is **No**! There are two reasons:
  - First, $X(\omega)$ is really the **DTFT** of a specific realization of X, which is $x[n]$. For each different realization of $x[n]$ we would get a different spectrum – yet we are interested in the spectrum of the overall process.
  - Second, **random signals** are by definition infinitely long in time. Therefore, they generally do not satisfy the Dirichlet conditions, and their **DTFT** is not defined in the first place.

So What to Do?

- While the random signals themselves usually do not have a DTFT, their autocorrelation and autocovariance sequences do!

- This is – inpart – because they are usually monotonically decreasing from their maximum value at lag=0, and hence the infinite Fourier sum is then absolutely summable.

- Specifically, the DTFT of the autocorrelation function of a WSS sequence $X[n]$ can be computed as:

$$S_{XX}(\omega) = S_X(\omega) = \Phi_{XX}(\omega) = \sum_{l=-\infty}^{\infty} \phi_{XX}[l] e^{-j\omega l}$$

which turns out to be the **power spectrum** of the original random process, which is known as the **Wiener – Khintchine theorem**. If $S_{XX}(\omega)$ is normalized by 2π, the quantity obtained is known as the **power density spectrum** or **power spectral density** (**PSD**), $P_{XX}(\omega)$, although $S_{XX}(\omega)$ is also routinely referred to as the **PSD** as well. Both functions – due to symmetry of ϕ_{XX} – are guaranteed to be real.

- Furthermore, the average power of a WSS random signal is computed as

$$P_X = E\left(\left|X[n]^2\right|\right) = \phi_{XX}[0] = \left|\mu_X\right|^2 + \sigma_X^2$$

Understanding Power Spectral Density

- Now let's look at this expression again: $P_X = E\left(\left|X[n]^2\right|\right) = \phi_{XX}[0] = |\mu_X|^2 + \sigma_X^2$
- Considering that the autocorrelation function is the inverse DTFT of power spectrum, $S_{XX}(\omega)$

$$\phi_{XX}[l] = \frac{1}{2\pi}\int_{-\pi}^{\pi} S_{XX}(\omega)e^{j\omega l}d\omega \qquad P_{XX}(\omega) = S_{XX}(\omega)/2\pi$$

- Then,

$$P_X = E\left(\left|X[n]^2\right|\right) = \phi_{XX}[0] = \frac{1}{2\pi}\int_{-\pi}^{\pi} S_{XX}(\omega)e^{j\omega 0}d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} S_{XX}(\omega)d\omega$$

That is, the average power is the PSD integrated (averaged) over all frequencies. If we wanted to find the average power in a particular frequency band, say $[\omega_1\ \omega_2]$, then

$$P_{[\omega_1\ \omega_2]} = \int_{\omega_1}^{\omega_2} P_{XX}(\omega)d\omega + \int_{-\omega_2}^{-\omega_1} P_{XX}(\omega)d\omega$$

Hence, $P_{XX}(\omega)$ is not the power at frequency ω, but rather the power density, in an infinitesimal frequency band around ω, or in other words, it is the power per unit frequency band. To obtain the actual average power in any frequency band, we simply integrate the power density in that interval.

A Very Special PSD: White Noise

- Similar to white light, which is defined as light that includes all frequencies (hence colors) of the visible light, a random signal that includes all frequencies is typically referred to as **white noise**.

- Theoretically speaking, a stationary random process is said to be **white**, if its PSD is constant across all frequencies, that is

$$S(\omega) = \sigma^2, \quad -\pi \le \omega \le \pi$$

  - Use of the letter sigma-squared for the constant value, is not by accident, as the variance of white noise is its power!

- The uniform noise we have seen so far whose PSD is (near) constant for all frequencies is therefore called **uniform white noise**.

- The autocorrelation function of **white noise** is, of course

$$\phi_{XX}(n) = \sigma_X^2 \delta[n]$$

Filtering Random Signals

- Since random signals change from one realization to the next, what can we say about their spectral properties when they are filtered?
  - Well, we can only talk about the statistical properties of their spectrum.
- We can easily show that if the random signal $x[n]$ (with autocorrelation function ϕ_{XX}) is the input to a filter whose impulse response is $h[n]$, then the autocorrelation function ϕ_{YY} of the output $y[n]$ is

$$\phi_{YY}[n] = \phi_{XX}[n] * h[n] * h[-n]$$

- And their PSD are related as

$$S_Y(e^{j\omega}) = S_X(e^{j\omega})\left|H(e^{j\omega})\right|^2$$

Estimating PSD

- There is still one more problem, however:
  - While the definition of the PSD as the DTFT of the autocorrelation function allows us to formally define PSD, it is still incomputable using a computer. This is because – not only because DTFT is continuous (that we can handle, by computing the DFT instead) - but also because the autocorrelation is still infinitely long.
- So how do we calculate the PSD?
  - The easiest approach is to
    - Truncate the autocorrelation function
    - Compute the DFT
  - This is really an approximation to the PSD, known as the **periodogram method** for PSD estimation, since the autocorrelation function is truncated,
    - Note that the truncation can be considered as being multiplied by a rectangular window function, which we know introduces the Gibbs's effect.
    - Over the years, many PSD estimation methods have been developed to reduce some of the negative effects of the PSD estimation through the **periodogram**.
    - For most practical purposes, and at least for the purposes of this class, **periodogram method** is sufficient.

The Periodogram

- The periodogram estimate of the PSD of a finite length-L sequence $x[n]$ is

$$\hat{P}_{XX}(\omega) = \frac{S_{XX}(\omega)}{2\pi} = \frac{1}{N}\left|\mathrm{DFT}_N(x[n])\right|^2$$

$$\hat{P}_{XX}(f) = \frac{1}{f_s N}\left|\mathrm{DFT}_N(x[n])\right|^2$$

where N is typically chosen as the next power of 2 value greater than L

- In Matlab, the function periodogram() implements the above expressions

Periodogram in Matlab

periodogram Power Spectral Density (PSD) estimate via periodogram method.

Pxx = PERIODOGRAM(X) returns the PSD estimate of the signal specified by vector X in the vector P_{XX}. By default, the signal X is windowed with a BOXCAR window of the same length as X. The PSD estimate is computed using an FFT of length given by the larger of 256 and the next power of 2 greater than the length of X. P_{XX} is the distribution of power per unit frequency. For real signals, PERIODOGRAM returns the one sided PSD by default; for complex signals, it returns the two-sided PSD. Note that a one-sided PSD contains the total power of the input signal.

Pxx = PERIODOGRAM(X,WINDOW) specifies a window to be applied to X. WINDOW must be a vector of the same length as X. If WINDOW is a window other than a boxcar (rectangular), the resulting estimate is a modified periodogram. If WINDOW is specified as empty, the default window is used.

[Pxx,W] = PERIODOGRAM(X,WINDOW,NFFT) specifies the number of FFT points used to calculate the PSD estimate. For real X, P_{XX} has length (NFFT/2+1) if NFFT is even, and (NFFT+1)/2 if NFFT is odd. Note that if NFFT is greater than the segment the data is zero-padded. If NFFT is less than the segment, the segment is "wrapped" (using DATAWRAP) to make the length equal to NFFT. This produces the correct FFT when NFFT < L, L being signal or segment length. W is the vector of normalized frequencies at which the PSD is estimated. W has units of rad/sample.

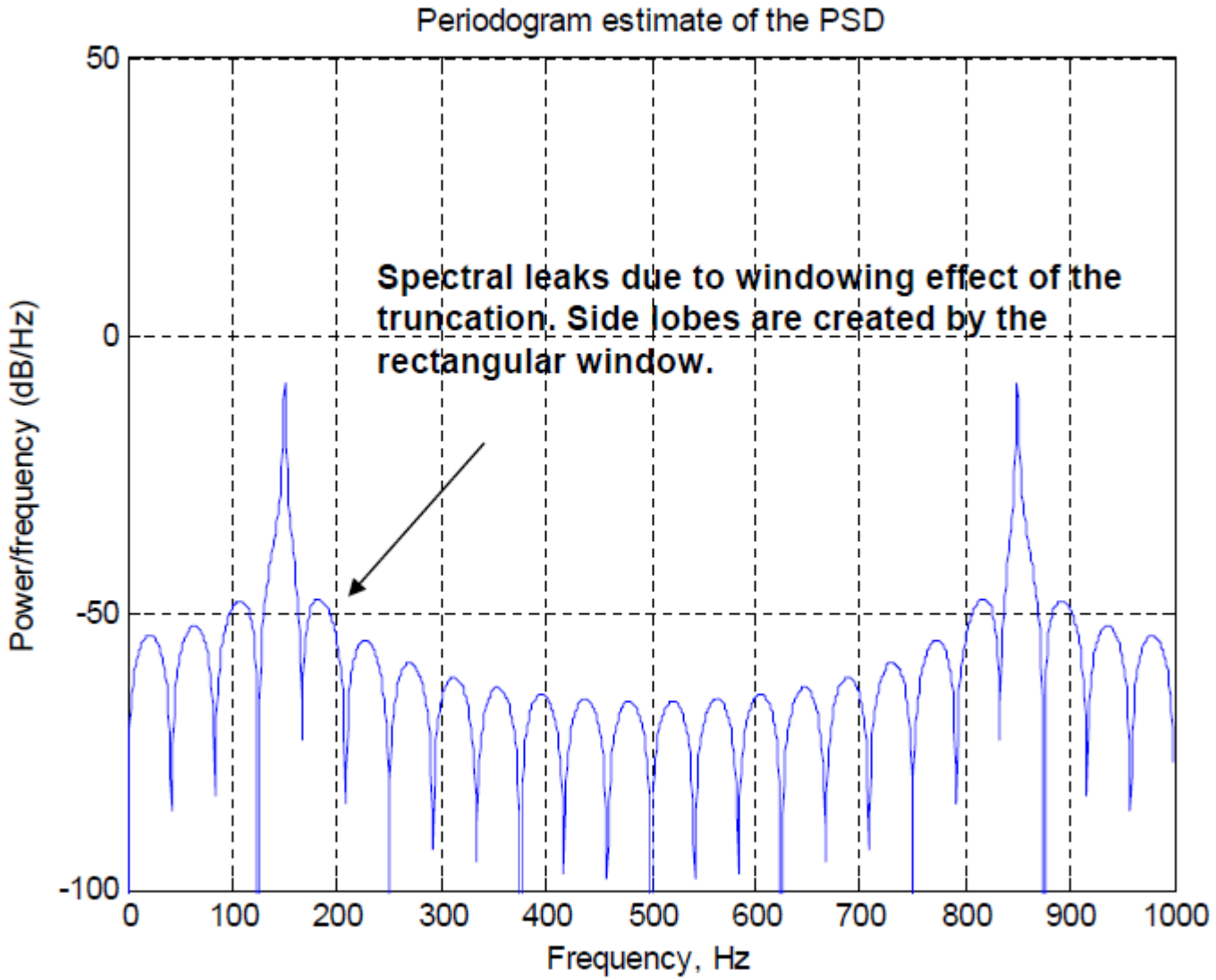
[Pxx,F] = PERIODOGRAM(X,WINDOW,NFFT,Fs) returns a PSD computed as a function of physical frequency (Hz). F_S is the sampling frequency specified in Hz. If F_S is empty, it defaults to 1 Hz. F is the vector of frequencies at which the PSD is estimated and has units of Hz.

[...] = PERIODOGRAM(...,'twosided') returns a two-sided PSD of a real signal X. In this case, P_{XX} will have length NFFT and will be computed over the interval [0,2*Pi) if F_S is not specified and over the interval [0, F_S) if F_S is specified.

PERIODOGRAM(...) with no output arguments by default plots the PSD estimate in dB per unit frequency in the current figure window.

An Example

```
fs=1000;
t=(0:fs)/fs;

ff=[150];
x=sin(2*pi*ff*t);

[Pxx f]=periodogram(x, [], 'twosided', 1024, fs);

plot(f, 10*log10(Pxx)); %Gain in power is calculated
with 10*log10(P)
grid
title('Periodogram estimate of the PSD')
xlabel('Frequency, Hz')
ylabel('Power/frequency (dB/Hz)')
```



Spectral Leaks & Resolution

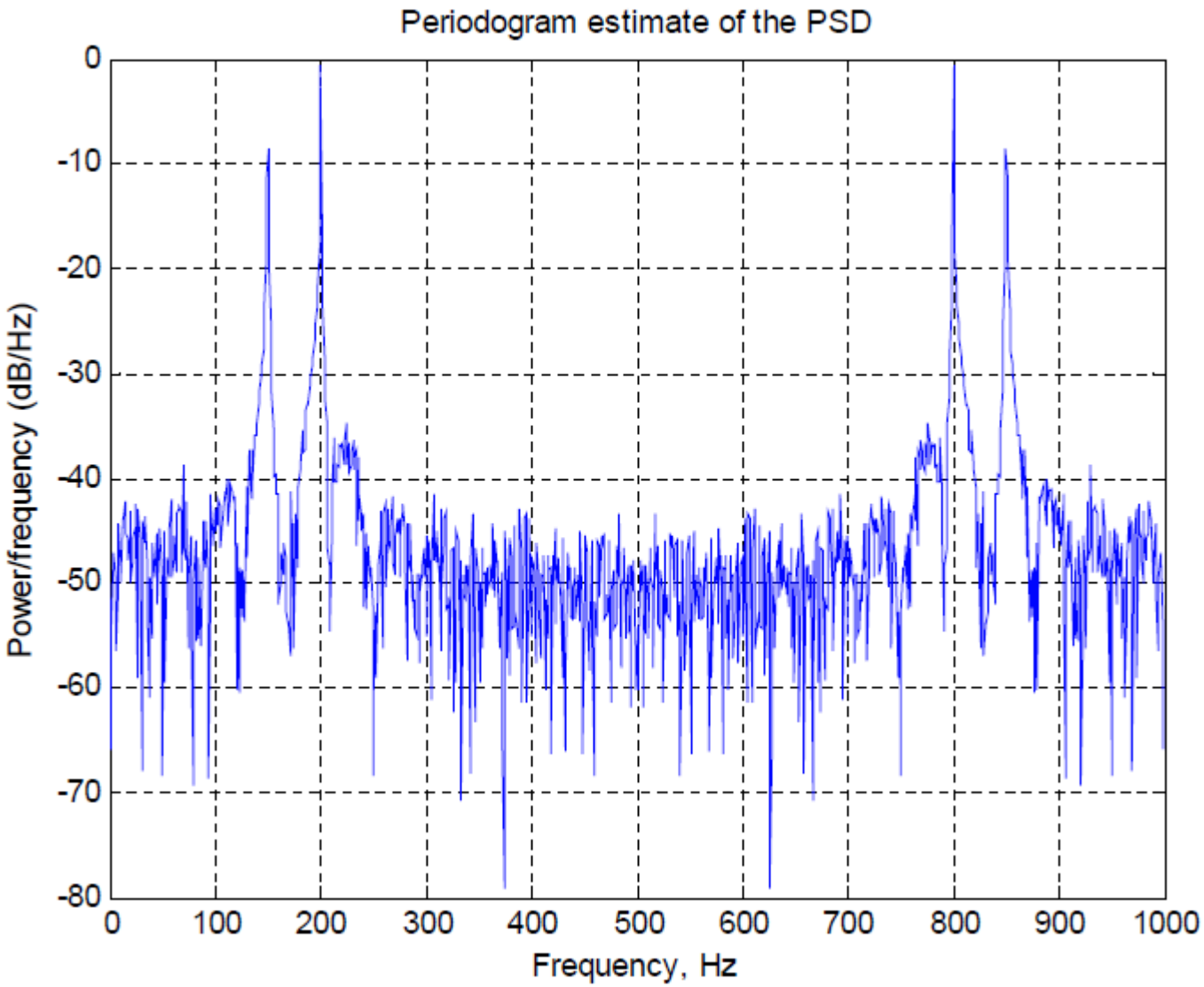
- The **spectral leaks** are the sidelobes created in the PSD due to the rectangular function used in truncating the signal
  - Recall that multiplying with a rectangular function in the time domain is equivalent to the convolution with a sinc in the frequency domain, which creates the sidelobes.
- A side effect of the **spectral leaks** is the resolution – the ability to discriminate spectral components that are located close to each other.
  - In order to resolve to frequencies that are close to each other, those frequencies must be further apart from each other than the width of the mainlobe of the leaked spectra for either of the spectra.
  - The main lobe width is defined as the width where the power is half the peak power of the main lobe (3dB), which is approximately f_S/N
  - Hence, in order to resolve two frequencies f_1 and f_2, they must satisfy

$$\Delta f = f_1 - f_2 > \frac{f_s}{N}$$

e.g., two resolve two frequencies 10 Hz apart, we need at least 100 samples at 1KHz.

Another Example

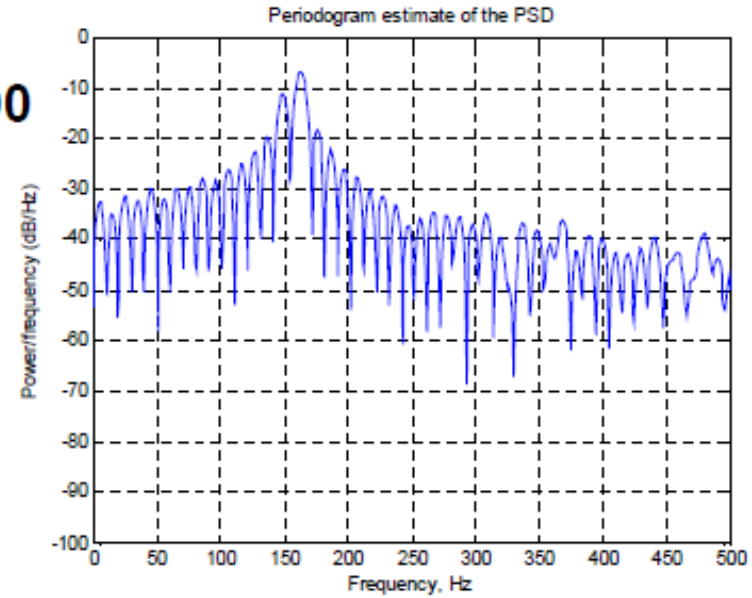
```
clear
close all

fs=1000;
t=(0:fs)/fs;
A=[1 2];
ff=[150; 200];
xn=A*sin(2*pi*ff*t)+0.1*randn(size(t));

%Equivalent to
% xn=sin(2*pi*150*t)+2*sin(2*pi*140*t)
% + 0.1*randn(size(t));

[Pxx f]=periodogram(xn, [], 'twosided', 1024, fs);
plot(f, 10*log10(Pxx));
%Gain in power is calculated with 10*log10(P)
grid
title('Periodogram estimate of the PSD')
xlabel('Frequency, Hz')
ylabel('Power/frequency (dB/Hz)')
```



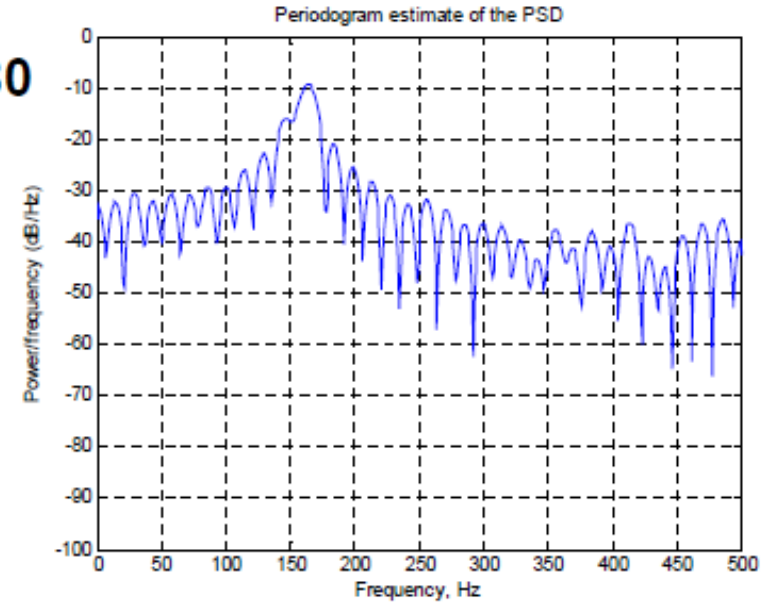
Another Example

```
fs=1000;
%t=(0:fs)/fs;
L=input('Enter the signal length:');
t=linspace(0, L/fs, L);
A=[1 2];
ff=[150; 160];
xn=A*sin(2*pi*ff*t)+0.1*randn(size(t));

%Equivalent to
% xn=sin(2*pi*150*t)+2*sin(2*pi*140*t) + .1*randn(size(t));

[Pxx f]=periodogram(xn, [], 1024, fs);

plot(f, 10*log10(Pxx)); %Gain in power is calculated with 10*log10(P)
axis([0 500 -100 0])
grid
title('Periodogram estimate of the PSD')
xlabel('Frequency, Hz')
ylabel('Power/frequency (dB/Hz)')

Ave_pow=fs/length(Pxx)*sum(Pxx)
```

L=100



L=80

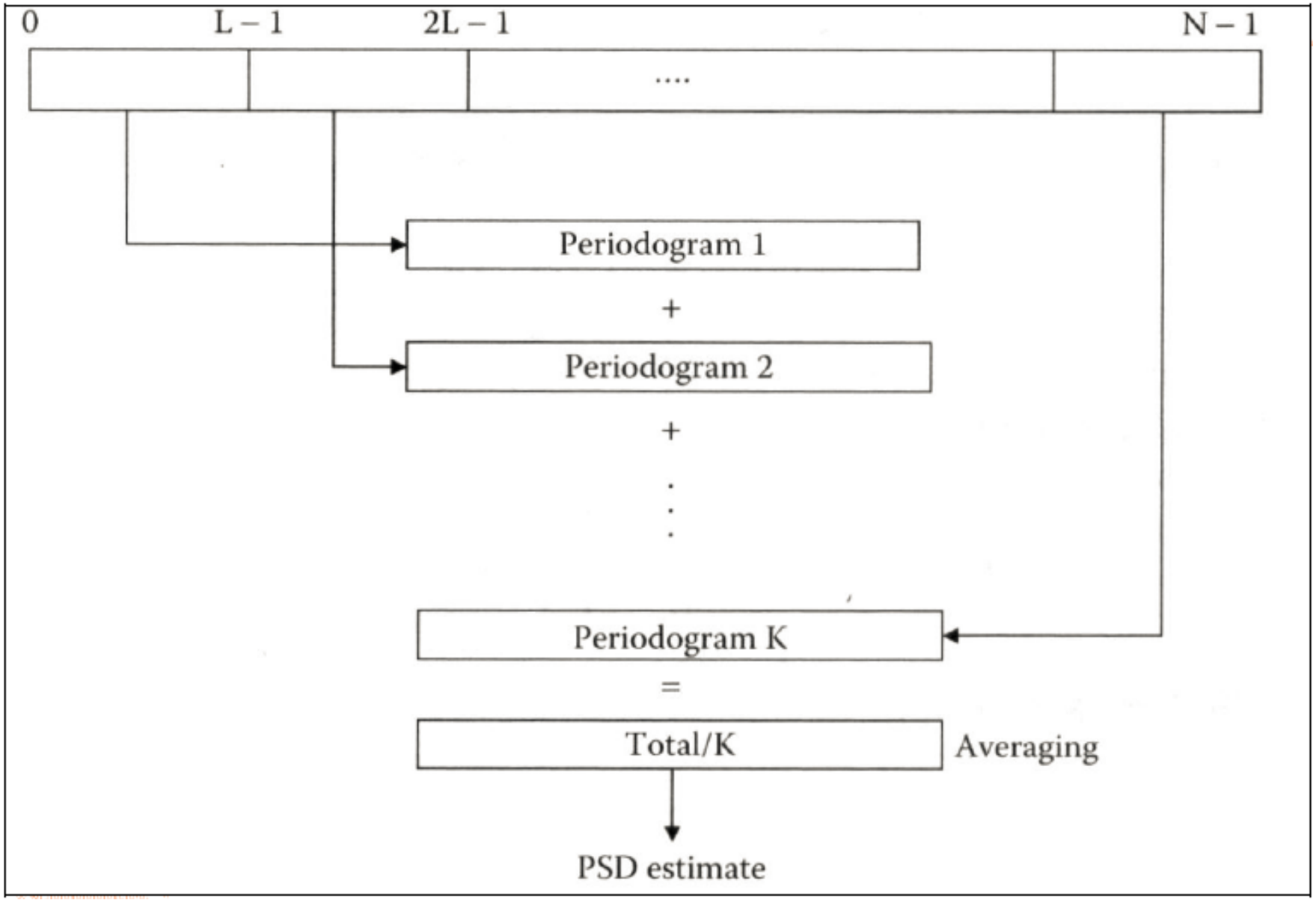


See what happens if you increase noise!

Modified Periodogram

- We can, of course, use a different windowing function than the rectangular window (boxcar), to smooth the edges of the truncation. This reduces the spurious frequencies introduced through leakage.
  - On the other hand, all other windows have a wider mainlobe, which means that the resolution is then reduced.
    - For Hamming window, for example, the 3dB width is twice that of the boxcar function, and hence only half the resolution can be obtained with the same signal length compared to boxcar function.
    - Try implementing periodogram(xn, hamming(length(xn)), 1024, fs);
  - Using different window functions is known as the **modified periodogram method**.

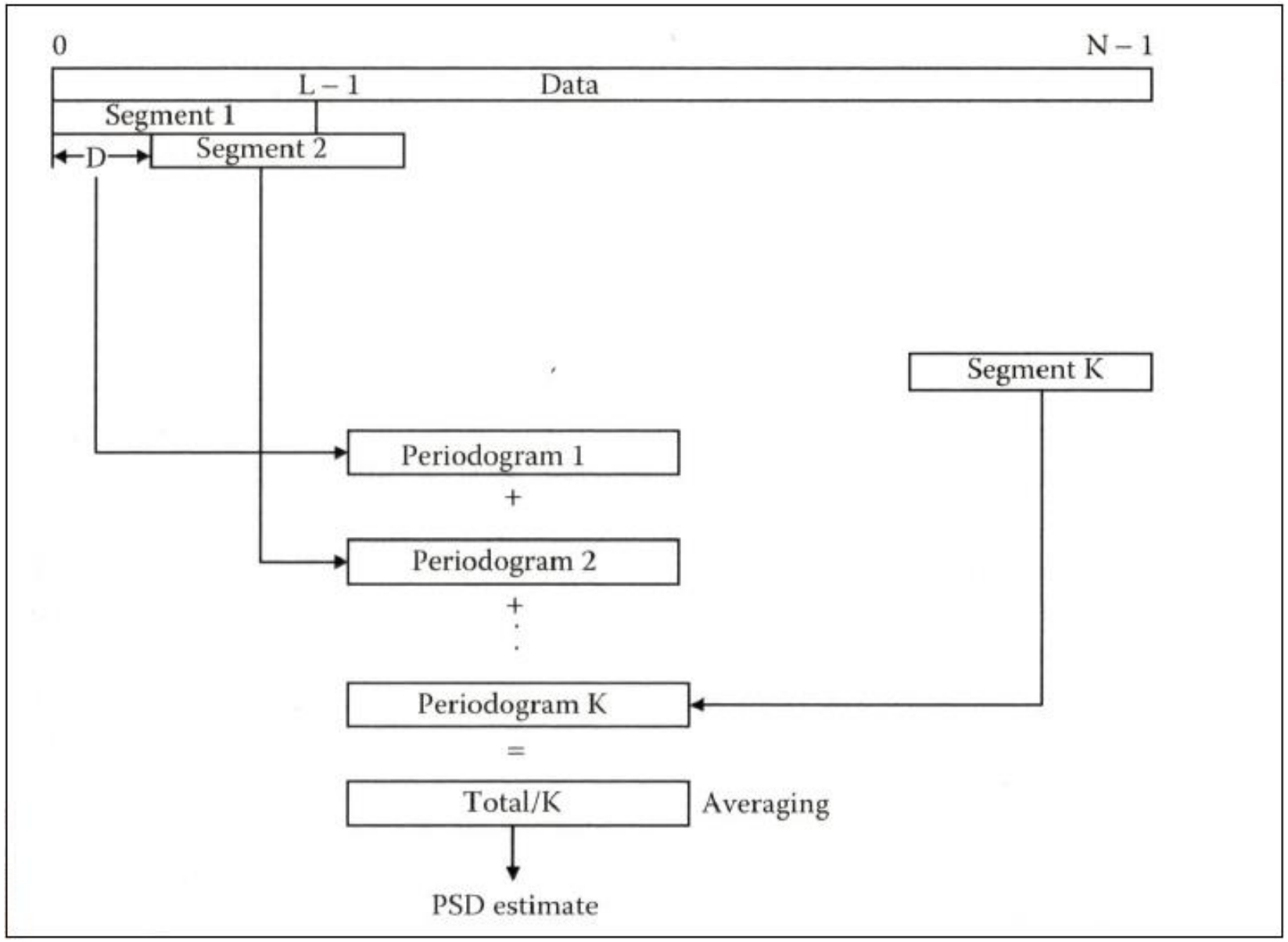
Bartlett's Method



Welch's Method

- A better estimate of the true PSD can be obtained using the following approach:
  - Divide the time-series data into (possibly overlapping) segments
  - Compute the modified periodogram of each segment
  - Average the PSD estimates obtained by each periodogram
- This is known as the **Welch's method**
  - The effect is decreased variance (from one realization to the next), hence a better estimate
  - However, the reduced data length also reduces the resolution
- In Matlab, the **Welch's method** is implemented by the pwelch() function.

Welch's Method



Welch's Method

pwelch Power Spectral Density estimate via Welch's method.

Pxx = PWELCH(X) returns the Power Spectral Density (PSD) estimate, Pxx, of a discrete-time signal vector X using Welch's averaged, modified periodogram method. By default, X is divided into eight sections with 50% overlap, each section is windowed with a Hamming window and eight modified periodograms are computed and averaged. If the length of X is such that it cannot be divided exactly into eight sections with 50% overlap, X will be truncated accordingly.

Pxx = PWELCH(X,WINDOW), when WINDOW is a vector, divides X into overlapping sections of length equal to the length of WINDOW, and then windows each section with the vector specified in WINDOW. If WINDOW is an integer, X is divided into sections of length equal to that integer value, and a Hamming window of equal length is used. If WINDOW is omitted or specified as empty, a default window is used to obtain eight sections of X.

Pxx = PWELCH(X,WINDOW,NOVERLAP) uses NOVERLAP samples of overlap from section to section. NOVERLAP must be an integer smaller than the WINDOW if WINDOW is an integer. NOVERLAP must be an integer smaller than the length of WINDOW if WINDOW is a vector. If NOVERLAP is omitted or specified as empty, the default value is used to obtain a 50% overlap.

Welch's Method

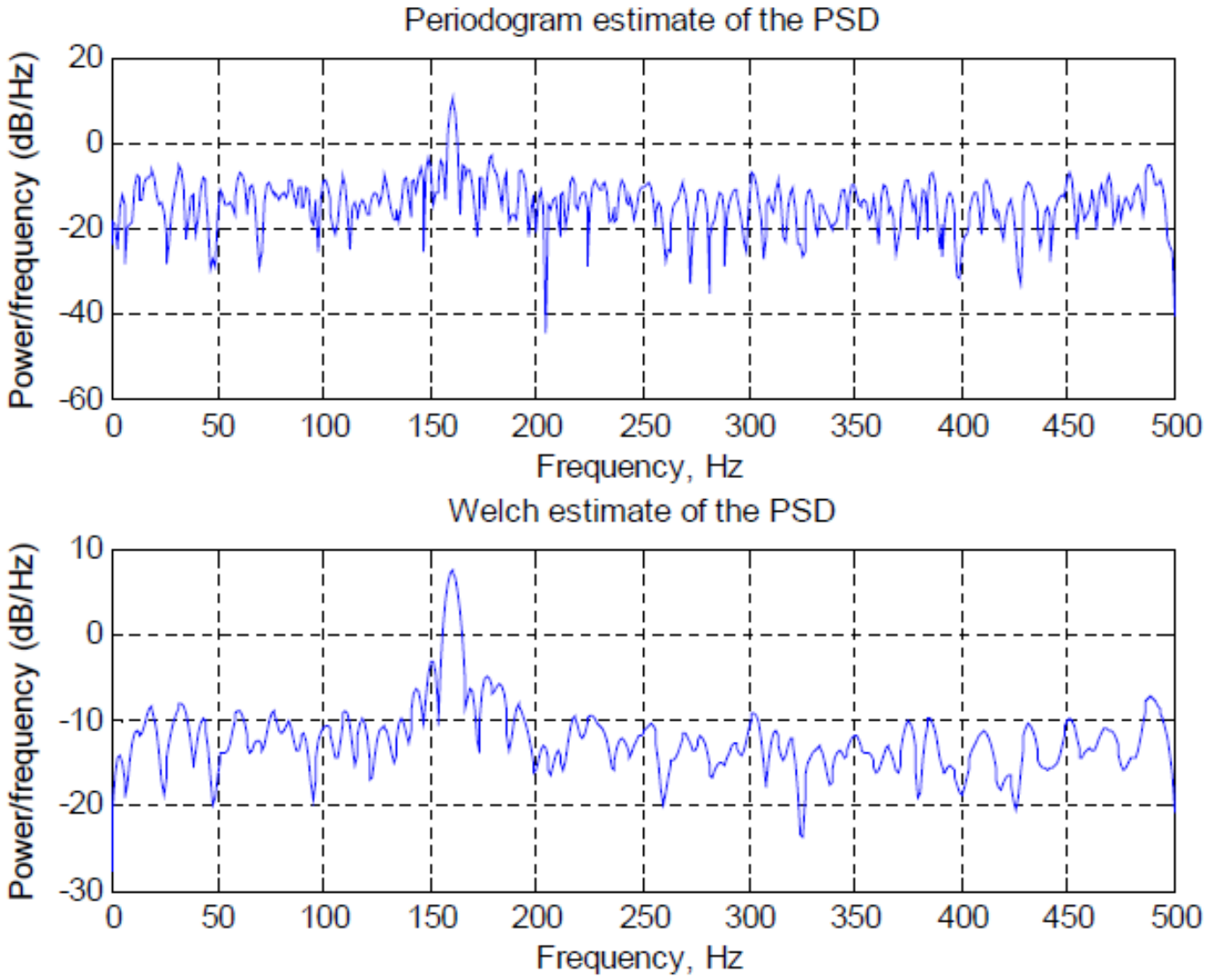
[Pxx,W] = PWELCH(X,WINDOW,NOVERLAP,NFFT) specifies the number of FFT points used to calculate the PSD estimate. If NFFT is specified as empty, the default NFFT -the maximum of 256 or the next power of two greater than the length of each section of X- is used. Note that if NFFT is greater than the segment the data is zero-padded. If NFFT is less than the segment, the segment is "wrapped" (using DATAWRAP) to make the length equal to NFFT. This produces the correct FFT when NFFT < L, L being signal or segment length. W is the vector of normalized frequencies at which the PSD is estimated. W has units of rad/sample.

[Pxx,F] = PWELCH(X,WINDOW,NOVERLAP,NFFT,Fs) returns a PSD computed as a function of physical frequency (Hz). Fs is the sampling frequency specified in Hz. If Fs is empty, it defaults to 1 Hz. F is the vector of frequencies at which the PSD is estimated and has units of Hz. For real signals, F spans the interval [0,Fs/2] when NFFT is even and [0,Fs/2) when NFFT is odd. For complex signals, F always spans the interval [0,Fs).

[...] = PWELCH(...,'twosided') returns a two-sided PSD of a real signal X. In this case, Pxx will have length NFFT and will be computed over the interval [0,2*Pi) if Fs is not specified and over the interval [0,Fs) if Fs is specified. Alternatively, the string 'twosided' can be replaced with the string 'onesided' for a real signal X. This would result in the default behavior. The string 'twosided' or 'onesided' may be placed in any position in the input argument list after NOVERLAP.

Example

```
fs=1000;
%t=(0:fs)/fs;
L=input('Enter the signal length:');
t=linspace(0, L/fs, L);
A=[2 8];
ff=[150; 160];
xn=A*sin(2*pi*ff*t)+5*randn(size(t));

[Pxx1 f]=periodogram(xn, rectwin(length(xn)), 1024, fs);
[Pxx2 f]=pwelch(xn, rectwin(150),75, 1024, fs); %For
300 points, this is 3 segments of 50% overlap

subplot(211)
plot(f, 10*log10(Pxx1)); grid
title('Periodogram estimate of the PSD')
xlabel('Frequency, Hz')
ylabel('Power/frequency (dB/Hz)')

subplot(212)
plot(f, 10*log10(Pxx2)); grid
title('Welch estimate of the PSD')
xlabel('Frequency, Hz')
ylabel('Power/frequency (dB/Hz)')
```

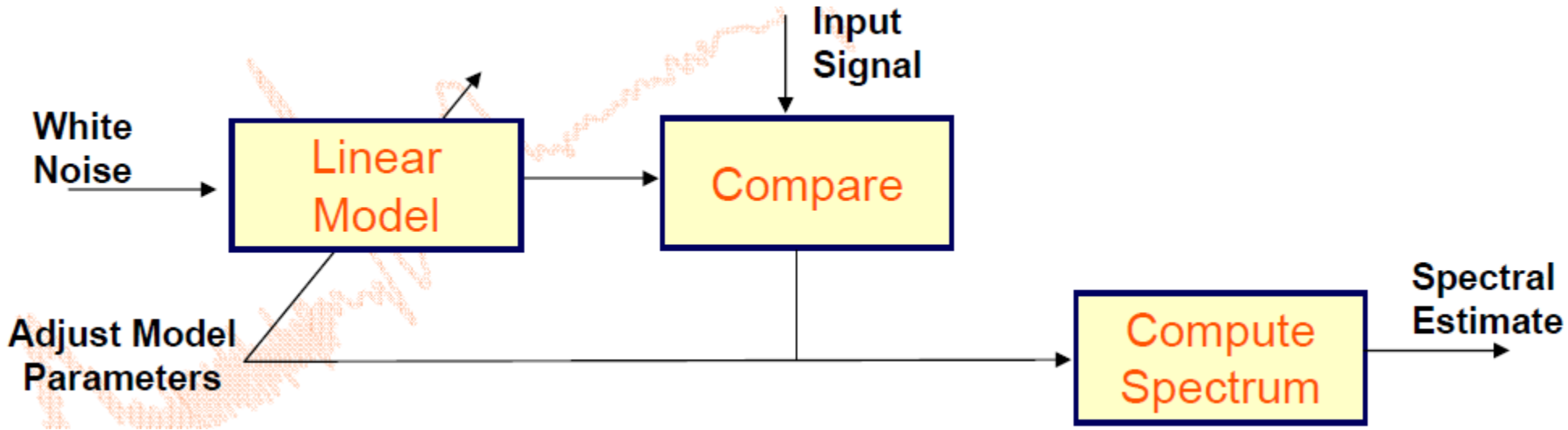


Welch's estimate provides, in general, smoother spectra. Is this good…?

- Recap the properties of the spectral estimation methods:
  - **Periodogram** → Good frequency resolution (due to assumed signal-long rectangular window), however there are spectral leaks due to sharp transitions (Gibb's phenomenon) of the rectangular window
    - For better time resolution, we need to have a longer time-record
  - The spectral leaks can be minimized by using a smoothing window, however, we then lose some of the frequency resolution (sharp peaks cannot be observed)
  - Spectral leaks can also be avoided using **Welch's method**. The larger the number of records are averaged, the smoother the spectrum. This technique provides good estimation (particularly if data length is long), because it provides smoothing to avoid spurious frequencies, but still provide decent resolution due to rectangular window it uses.
- The above problems stem from the fact that we assume the signal to be zero outside of the windowing function, which of course is not true!
  - **Model based approaches** are designed to overcome these problems, but only at the cost of additional computation and some prior knowledge requirement.

Parametric Model Based Techniques

- If prior information is available regarding the properties of the process that generated the random signal in the first place, then we can eliminate the need to window.
  - This would be particularly useful for short data, where FT based approaches do not work very well.
  - Fact: any WSS process can be realized as the output of a causal and stable filter (linear model) that is driven by white noise – after all, the white noise includes all frequencies, and by appropriately blocking some of those frequencies with a filter, any spectrum can be generated.
  - In this approach, we start with some parametric model filter and compare its output to the input signal – the model parameters are then adjusted until the filter output matches the input. The model's frequency spectrum is then the best estimate of the input signal's spectrum.



Parametric Models

- What models are available?
  - Many, but the following three are most common:
- **Autoregressive moving average process – ARMA model**
  - This model has a polynomial structure with both numerator and denominator coefficients → the model represents a spectrum with both poles and zeros
    - Appropriate, if the spectrum to be estimated is known to contain both sharp peaks (poles) and deep valleys (zeros)
- **Autoregressive process – AR model**
  - This is a special case of the ARMA model, where only the denominator coefficients are present → the model represents a spectrum with poles only
    - Appropriate if the spectrum to be estimated contains sharp peaks, but not valleys
- **Moving average process – MA model**
  - Also special case of the ARMA model, where only numerator coefficients are present → the model represents a spectrum with zeroes only
    - Appropriate if the spectrum to be estimated contains deep valleys, but no sharp peaks.

ARMA & MA Models

- Represents a stable, shift-invariant system with p poles and q zeros

$$H(z)=\frac{B(z)}{A(z)}=\frac{b_0+b_1z^{-1}+\cdots+b_qz^{-q}}{1+a_1z^{-1}+\cdots+a_pz^{-p}}=\frac{\sum_{k=0}^{q}b[k]z^{-k}}{1+\sum_{k=1}^{p}a[k]z^{-k}}$$

In frequency domain

$$y[n]=-\sum_{k=1}^{p}a[k]y[n-k]+\sum_{k=1}^{q}b[k]x[n-k]$$

In time domain

  - The problem is then to estimate the $b[k]$ and $a[k]$ coefficients. The primary method to do so is to use the so-called Yule-Walker equations and/or using an iterative technique, such as **Durbin algorithm**.

- In **MA model**, we have a system with q zeros and no poles

$$H(z)=B(z)=b_0+b_1z^{-1}+\cdots+b_qz^{-q}=\sum_{k=0}^{q}b[k]z^{-k}$$

In frequency domain

$$y[n]=\sum_{k=1}^{q}b[k]x[n-k]$$

In time domain

  - It turns out, the FT based methods fit best to **MA type system**, and since they are much simpler than estimating the $b[k]$ coefficients, they are often preferred.

AR Model

- In **AR model**, we assume a system with p-poles but no zeros:

$$H(z)=\frac{b_0}{A(z)}=\frac{b_0}{1+a_1z^{-1}+\cdots+a_pz^{-p}}=\frac{b_0}{1+\sum_{k=1}^{p}a[k]z^{-k}}$$

In frequency domain

$$y[n]=-\sum_{k=1}^{p}a[k]y[n-k]+b_0x[n]$$

In time domain

- If a white noise process $w[n]$ with variance σ^2 is the input to this filter, the output process $y[n]$ will be a WSS process with a power spectrum

$$S_y(\omega)=\sigma^2\frac{b_0^2}{|A(\omega)|^2}$$

- It can be shown that the autocorrelation of the output also satisfies

$$\phi_Y[k]+\sum_{n=1}^{p}a[n]\phi_Y[k-n]=\sigma^2b_0^2\delta[k]\quad k\geq 0$$

In (Toeplitz) matrix form

$$\begin{bmatrix}\phi(0) & \phi(-1) & \cdots & \phi(-p)\\ \phi(1) & \phi(0) & \cdots & \phi(-p+1)\\ \vdots & \vdots & & \vdots\\ \phi(p) & \phi(p-1) & \cdots & \phi(0)\end{bmatrix}\begin{bmatrix}1\\ a(1)\\ \vdots\\ a(p)\end{bmatrix}=\sigma^2b_0^2\begin{bmatrix}1\\ 0\\ \vdots\\ 0\end{bmatrix}$$

- These (**Yule-Walker**) **equations** can also be solved using **L-D algorithm**

Levinson-Durbin Recursion

- **LD Recursion** is an iterative technique for solving the coefficients of an all-pole (AR) model
  - Given the autocorrelation sequence ϕ , compute the $a[k]$ and b_0 coefficients

$$\begin{bmatrix} \phi(0) & \phi(1) & \cdots & \phi(p) \\ \phi(1) & \phi(0) & \cdots & \phi(p-1) \\ \vdots & \vdots & & \vdots \\ \phi(p) & \phi(p-1) & \cdots & \phi(0) \end{bmatrix} \begin{bmatrix} 1 \\ a(1) \\ \vdots \\ a(p) \end{bmatrix} = \sigma^2 b_0^2 \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \varepsilon_p \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Modeling error

$$\phi(k) = \phi^*(-k)$$

$$\mathbf{R}_p \mathbf{a}_p = \varepsilon_p \mathbf{u}_1$$

  - The algorithm's iterations are in the model order, p: the coefficients of the $(j+1)^{st}$ order all-pole model a_{j+1} are found from the coefficients of the j-pole model a_j

$$\mathbf{R}_{j+1} \mathbf{a}_{j+1} = \varepsilon_{j+1} \mathbf{u}_1$$

$$\begin{bmatrix} \phi(0) & \phi(1) & \cdots & \phi(j) \\ \phi(1) & \phi(0) & \cdots & \phi(j-1) \\ \vdots & \vdots & & \vdots \\ \phi(p) & \phi(p-1) & \cdots & \phi(0) \end{bmatrix} \begin{bmatrix} 1 \\ a(1) \\ \vdots \\ a(j) \end{bmatrix} = \begin{bmatrix} \varepsilon_j \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$\mathbf{R}_j$

Levinson-Durbin Recursion

1. Initialize the recursion
   - (a) $a_0(0) = 1$
   - (b) $\epsilon_0 = r_x(0)$
2. For $j = 0, 1, \ldots, p-1$
   - (a) $\gamma_j = r_x(j+1) + \sum_{i=1}^{j} a_j(i) r_x(j-i+1)$
   - (b) $\Gamma_{j+1} = -\gamma_j / \epsilon_j$
   - (c) For $i = 1, 2, \ldots, j$

     $$a_{j+1}(i) = a_j(i) + \Gamma_{j+1} a_j^*(j-i+1)$$
   - (d) $a_{j+1}(j+1) = \Gamma_{j+1}$
   - (e) $\epsilon_{j+1} = \epsilon_j \left[1 - |\Gamma_{j+1}|^2\right]$
3. $b(0) = \sqrt{\epsilon_p}$

$(j+1)^{st}$ reflection coefficient

levinson The Levinson-Durbin Recursion is an algorithm for finding an all-pole IIR filter with a prescribed deterministic autocorrelation sequence. It has applications in filter design, coding, and spectral estimation. The filter that levinson produces is minimum phase. The filter coefficients are ordered in descending powers of z.

A = levinson(R,N) solves the Hermitian Toeplitz system of equations

$$H(z) = \frac{1}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}}$$

```
[ R(1) R(2)* ... R(N)* ] [ A(2) ] = [ -R(2 ) ]
[ R(2) R(1) ... R(N-1)*] [ A(3) ] = [ -R(3) ]
[ . . ... . ] [ . ] = [ . ]
[ R(N-1) R(N-2) ...R(2)* ] [ A(N) ] = [ -R(N) ]
[ R(N) R(N-1) ... R(1) ] [ A(N+1) ] = [ -R(N+1) ]
```

Note that Matlab does not compute the b_0 coefficient

(also known as the Yule-Walker AR equations) using the Levinson- Durbin recursion. Input R is typically a vector of autocorrelation coefficients with lag 0 as the first element.

N is the order of the recursion; if omitted, N = length(R)-1. A will be a row vector of length N+1, with A(1) = 1.0.

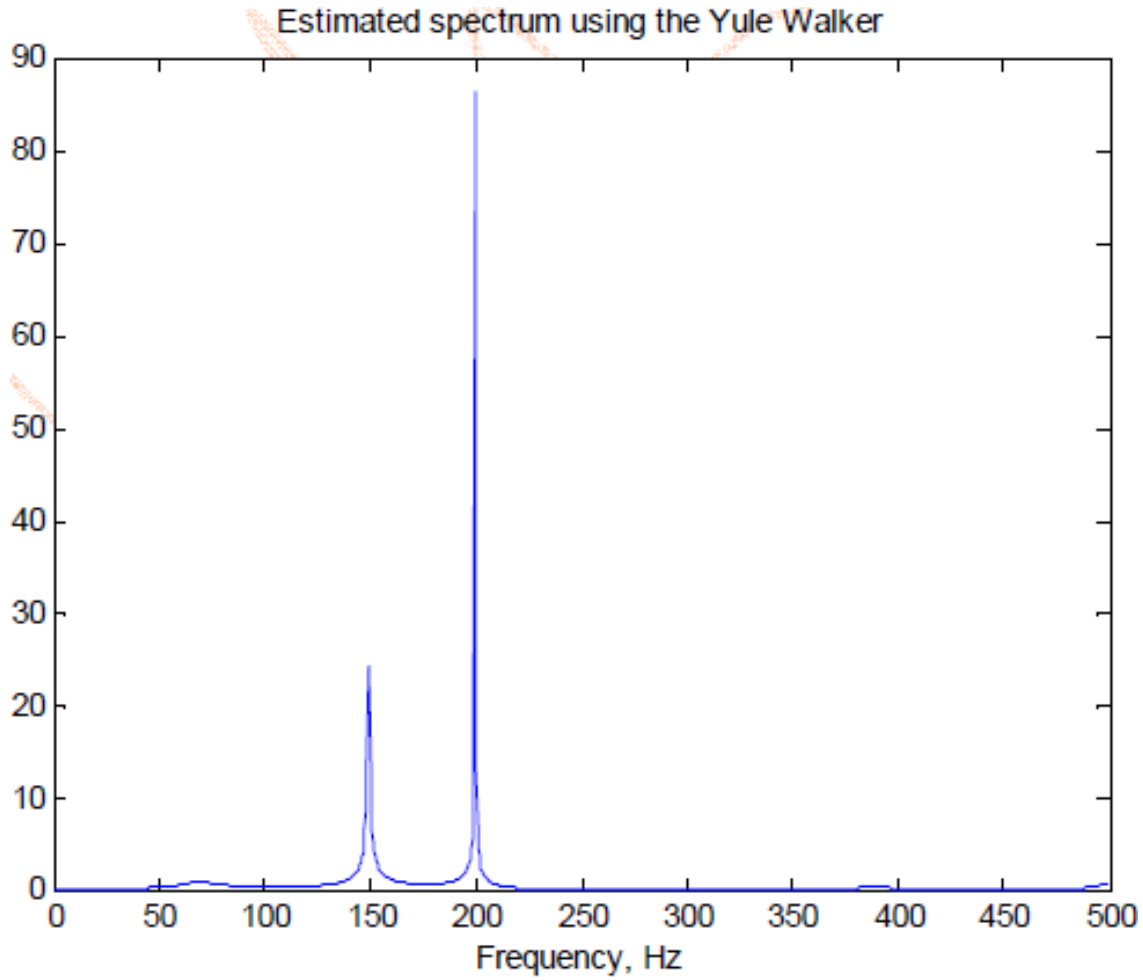
[A,E] = levinson(...) returns the prediction error, E, of order N.

[A,E,K] = levinson(...) returns the reflection coefficients K as a column vector of length N.

If R is a matrix, levinson finds coefficients for each column of R, and returns them in the rows of A

Example

```
fs=1000;
t=(0:fs)/fs;
A=[1 2];
ff=[150; 200];
xn=A*sin(2*pi*ff*t)+0.5*randn(size(t));

[r lags]=xcorr(xn,100);
subplot(211)
plot(t, xn); grid
title('Noisy signal')
subplot(212)
plot(lags, r); grid
title('Autocorrelation sequence')
a=levinson(r, 10);
figure
[H f]=freqz(1, a, 1024, 'half', 1000);
plot(f, abs(H))
title('Estimated spectrum using the Yule Walker')
xlabel('Frequency, Hz')
```



Try this using different noise levels, frequency resolutions and filter orders!