Biosignals

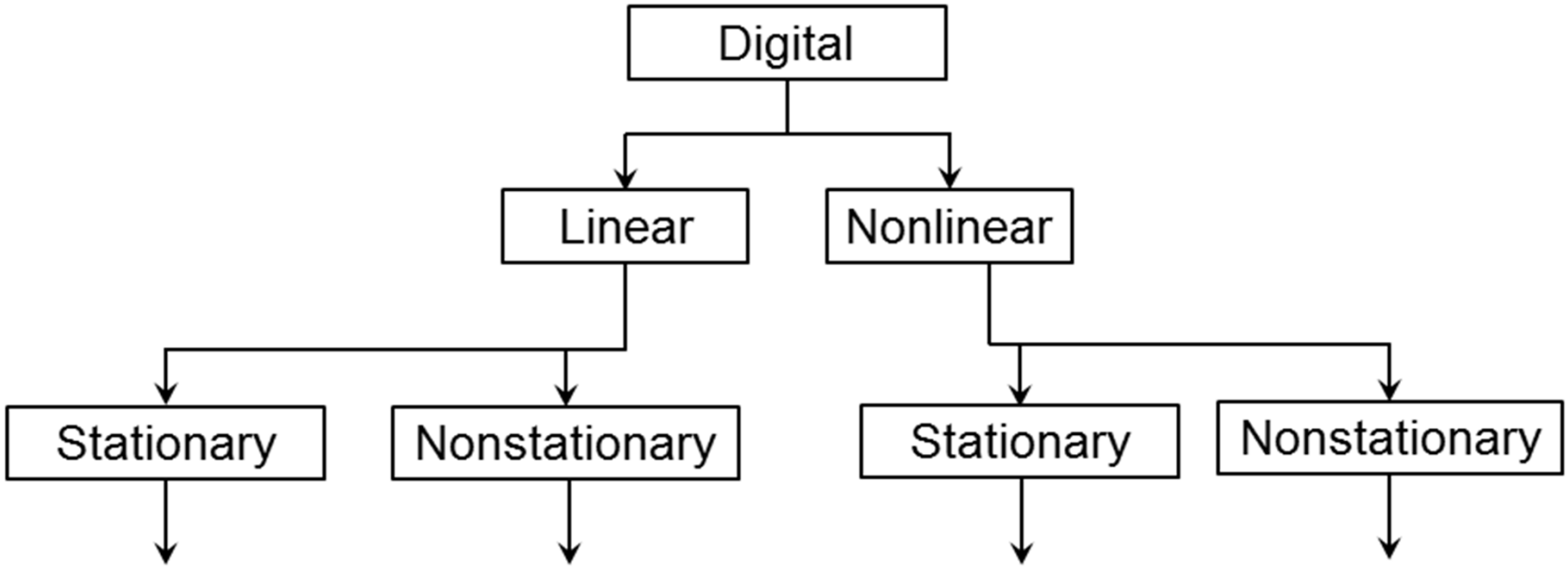
Types of Signals

- Digital signals can be classified into major subgroups: *linear* and *nonlinear*. Linear signals derive from linear processes while nonlinear signals arise from nonlinear processes such as chaotic or turbulent systems.

- These two signal classes can be further divided into *stationary* and *nonstationary* signals. Stationary signals have consistent statistical properties over the data set being analyzed.

- Applicable analytical tools depend on the signal that is being used.

Digital Signal Types



+ Two-dimensional signals

* Section 8.2

Signal Encoding

- All signals involve some type of encoding scheme.
- Most encoding strategies can be divided into two broad categories or domains: continuous and discrete.
- Continuous signals usually encode information in terms of signal amplitude (the intensity of the signal, voltage, or current values) as a function of time.
- For example, the temperature in a room can be encoded so that 0 volts represents 0.0 ^{o}C, 5 volts represents 10 ^{o}C, 10 volts represents 20 ^{o}C, and so on. If *linear*, the encoding equation would be:

$$voltage\ amplitude = temperature/2 \text{ volts}$$

Linear Signals

- The equation in the last slide relates the input (temperature) to the output (voltage) following the classic linear relationship:

  $y = mx + b$

  where *m* is the slope of the input-output relationship and *b* is the offset which in this case is 0.0.

- The temperature can be found from the voltage output of the transducer as:

  temperature = 2* *voltage* ^{o}C

- When the information is encoded in terms of signal amplitude, it is known as an analog signal.

Linearity

- The concept of linearity has a rigorous definition, but the basic concept is one of proportionality. If you double the input into a linear system, you will double the output.

- One way of stating this proportionality property mathematically is: if the independent variables of linear function are multiplied by a constant, *k*, the output of the function is simply multiplied by *k.*

- If $y = f(x)$ where f is a linear function:

$$ky = f(kx)$$

Properties of Linear Signals

- If *f* is a linear function:

$$f\left(x_1(t)\right)+\ f\left(x_2(t)\right)=\ f\left(x_1(t)+x_2(t)\right)$$

- If $z=\frac{df(x)}{dx}$ then $\frac{df(kx)}{dx}=k\left(\frac{df(x)}{df}\right)=kz$

- If $z=\int f dx$ then $\int f(kx)dx=k\int f(x)dx=kz$

- Derivation and integration are linear operations.

- Systems that contain derivative and integral operators and other linear operators produce linear signals.

Analog Signal

- Analog encoding was common in consumer electronics, but most of these applications now use digital encoding. (The strange resurgence of vinyl records is a notable exception.)

- Analog encoding is important to the biomedical engineer, because most biotransducers generate analog encoded signals.

- This book uses assumes signals are digitally encoded, often from an analog source.

- A discussion of analog signal processing and analog-to-digital conversion is found in Chapter 1.

Digital Signals

- A continuous analog signal after conversion to the digital domain is represented by a series of discrete samples (numbers) at specific points in time (see Section 1.6.2):

  $X[n] = x[1], x[2], x[3], \ldots x[n]$

- Usually this series of numbers would be stored in sequential memory locations: $x[1]$ followed by $x[2]$, etc.

- In this case, the memory index number, n, relates to the time associated with a given sample given by Eq. 1.5:

$$t = nT_s = \frac{n}{f_s}$$

where f_s is the sample frequency.

Time Invariance

- If a system's response characteristics do not change over time, that is, its statistical properties are constant, it is said to be *time-invariant*.

- Time invariance is a stricter version of stationarity since a time-invariant system would also be stationary.

- Mathematically: if f is a linear function, then for time invariance:

$$y(t-T) = f\left(x(t-T)\right)$$

Note that time-invariant signals should not be confused with time-varying, that is, signals that fluctuate in amplitude. Time-invariant signals can still be time-varying.

LTI Systems

- A system that is both linear and time-invariant is referred to as a *linear time-invariant* (*LTI*) system.

- The LTI assumptions allow us to apply a powerful array of mathematical tools known collectively as linear systems analysis or linear signal analysis.

- Most living systems change over time, they are adaptive, and they are often nonlinear, but the power of linear systems analysis is sufficiently seductive that simplifying assumptions or approximations are made so that these tools can be used.

Causality

- A system that responds only to current and past inputs is termed *causal*.

- Systems that exist in the real-world (e.g., analog electronic filters) must be causal.

- Computer programs can operate on stored in the computer using values that appear to be in the future with respect to a given operation.

- Such systems are *noncausal.*

(Some digital filters in Chapter 4 are noncausal.)

Superposition

- Linearity is required for the application of an important concept known as *superposition*.

- Superposition states that if there are two (or more) inputs acting on a linear system, the system responds to each as if it were the only input (i.e., the other input was not there). The influence of multiple inputs is the summation of each stimulus acting alone.

- This allows a “divide and conquer” approach, in which complex stimuli can be broken down into simpler components and an input-output analysis performed on each sub-stimulus as if the others did not exist.

Basic Signal Measurements

Mean or average value (discrete):

$$x_{avg} = \bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

Mean or average value (continuous):

$$\bar{x}(t) = \frac{1}{T}\int_0^T x(t)\,dt$$

RMS (discrete):

$$x_{rms} = \left[\frac{1}{N}\sum_{n=1}^{N} x_n^2\right]^{1/2}$$

RMS (continuous):

$$x(t)_{rms} = \left[\frac{1}{T}\int_0^T x(t)^2\,dt\right]^{1/2}$$

Example 2.1. Find the RMS value of the sinusoidal signal using both analytical and digital approaches.

Analytical:

$$\bar{x}(t)_{rms} = \left[\frac{1}{T}\int_0^T x(t)^2\,dt\right]^{1/2} = \left[\frac{1}{T}\int_0^T \left(A\sin\left(\frac{2\pi t}{T_p}\right)\right)^2 dt\right]^{1/2}$$

$$= \left[\frac{1}{T}\frac{A^2}{2\pi}\left(-\cos\left(\frac{2\pi t}{T}\right)\sin\left(\frac{2\pi t}{T}\right)+\frac{\pi t}{T}\right)\Bigg|_0^T\right]^{1/2}$$

$$= \left[\frac{A^2}{2\pi}\left(-\cos(2\pi)\sin(2\pi)+\pi+\cos 0\sin 0\right)\right]^{1/2}$$

$$= \left[\frac{A^2\pi}{2\pi}\right]^{1/2} = \left[\frac{A^2}{2}\right]^{1/2} = \frac{A}{\sqrt{2}} = 0.707A$$

Solution, digital: Generate a 1-cycle sine wave (T_s = 0.005 sec; N = 500; so T_T, = NT_s = 0.005(500) = 2.5 sec. To generate a single cycle given these parameters, the frequency of the sine wave should be $f = 1/T_T = 1/2.5$ = 0.4 Hz. Set the amplitude of the sine wave, A = 1.0.

```
N = 500;                    % Number of points for waveform
Ts = .005;                  % Sample interval = 5 msec
t  = (1:N)*Ts;              % Generate time vector (t = N Ts)
f = 1/(Ts*N);               % Sine wave freq. for 1 cycle
A = 1;                      % Sine wave amplitude
x = A*sin(2*pi*f*t);        % Generate sine wave
RMS = sqrt(mean(x.^2))      % Take the RMS value and output.
```

Results: The value produced by this program is **`0.7071`**, which is very close to the theoretical value of $\sqrt{2}$ times A.

Basic Signal Statistics

Variance, σ^2, is a measure of signal variability similar to RMS:

$$\sigma^2 = \frac{1}{N\text{-}1}\sum_{n=1}^{N}(x_n - \bar{x})^2 \qquad \sigma^2 = \frac{1}{T}\int_0^T \left(x(t) - \bar{x}\right)^2 dt$$

Standard deviation, σ, is the square root of variance: :

$$\sigma = \left[\frac{1}{N\text{-}1}\sum_{n=1}^{N}(x_n - \bar{x})^2\right]^{1/2} \qquad \sigma = \left[\frac{1}{T}\int_0^T \left(x(t) - \bar{x}\right)^2 dt\right]^{1/2}$$

MATLAB Implementation

```
xm = mean(x);        % Mean of x

xvar = var(x);       % Variance of x normalizing by N-1

xstd = std(x);       % Standard deviation of x
```

For matrices, **mean(X)** is a row vector containing the mean value of each column.

The same is true for **xstd** and **xvar.**

Example 2.2 Evaluate a signal to determine if it is stationary. If not, attempt to modify the signal to remove the nonstationarity, if possible.
The file **data_c1.mat** contains the signal in variable **x**. (N = 1000 and T_s = 0.001 sec.)

Solution, Part 1: In Part 1 we want to determine if the signal is stationary. If it is not, we will modify the signal in Part 2 to make it approximately stationary.

One straightforward method is to segment the data and evaluate the mean and variance of the segments. If they are the same for all segments, the signal is probably stationary.

If these measures change segment-to-segment, the signal is clearly nonstationary.

```
% Example 2.2, Part 1. Evaluate a signal for stationarity.
%
load data_c1;                    % Load the data. x
for k = 1:4                      % Divide into 4 segments
   m = 250*(k-1) + 1;            % Index of first segment sample
   segment = x(m:m+249);                % Extract segment
   avg(k) = mean(segment);              % Evaluate segment mean
   variance(k) = var(segment);          %   and segment variance
End
%
% Output means and variance with header comments
disp('Mean   Segment 1 Segment 2 Segment 3 Segment 4')
disp(avg)                                          % Output means
disp('Variance Segment 1 Segment 2 Segment 3 Segment 4')
disp(variance)                                      % Output variance
```

Results are shown in the next slide.

Example 2.2 Results, Part 1, show that the signal is nonstationary.

	Segment 1	Segment 2	Segment 3	Segment 4
Mean	0.0640	0.4594	0.6047	0.8095
Var	0.1324	0.1089	0.0978	0.1021

Solution, Part 2: One trick is to transform the signal by taking the derivative: the difference in value between each sample. Since only the mean is changing, another approach is to *detrend* the data, that is, subtract out and estimate the changing mean. (MATLAB's **detrend** operator can be used.)

The first approach is used here.

```
% Example 2.8, Part 2. Modify a nonstationary signal to become
stationary
%
y = [diff(x);0];                    % Take difference between points.
for k = 1:4                         % Segment signal into 4 segments
    m = 250*(k-1) + 1;              % Index of first segment sample
    segment = y(m:m+249);           % Extract segment
    avg(k) = mean(segment);         % Evaluate segment mean
    variance(k) = var(segment); %   and segment variance
end
    ..... Output avg and variance as above .....
```

Results, Part 2, show the modify signals is stationary with means near 0.0

	Segment 1	Segment 2	Segment 3	Segment 4
Mean	0.0020	-0.0008	0.0004	0.0004
Variance	0.0061	0.0057	0.0071	0.0066

Decibels

Decibels (dB) are units that compare the intensity of two signals using log ratios, V_{Sig1}/V_{Sig2}. Decibels are not really units, but a logarithmic scaling of a dimensionless ratio.

Advantages of decibels include:

1. The log operation compresses the range of values (e.g., a range of 1 to 1000 becomes a range of 1 to 3 in log units);

2. When numbers or ratios are to be multiplied, they are simply added if they are in log units;

3. The logarithmic characteristic is similar to human perception.

Decibels (dB) Definition

- The logarithmic unit called the Bel turned out to be inconveniently large, so it has been replaced by the decibel: dB = 1/10 Bel.

- While originally defined only in terms of a ratio, dB units are also used to express the power or intensity of a single signal.

- When applied to a power measurement, the decibel is defined as 10 times the log of the power ratio:

$$P_{dB} = 10 \log\left(\frac{P_2}{P_1}\right) \text{ dB}$$

Decibels (dB) Other Formulations

Power is usually proportional to the square of voltage. So when a signal voltage, or voltage ratio, is being converted to dB, Eq. 2.19 has a constant multiplier of 20 instead of 10 since log x^2 = 2 log x:

$$V_{dB} = 10\log\left(\frac{V_{Sig1}}{V_{Sig2}}\right)^2 = 20\log\left(\frac{V_{Sig1}}{V_{Sig2}}\right)$$

For a ratio of signals; or for a single signal:

$$V_{dB} = 10\log\left(V_{Sig}\right)^2 = 20\log\left(V_{Sig}\right)$$

Signal-to-Noise Ratio

- The Signal-to-noise ratio or SNR is simply the ratio of signal to noise, both measured in RMS (root-mean-squared) amplitude. The SNR is often expressed in dB where:

$$\mathrm{SNR} = 20 \log \left(\frac{\mathrm{signal}}{\mathrm{noise}} \right)$$

- To convert from dB scale to a linear scale:

$$\mathrm{SNR}_{\mathrm{Linear}} = 10^{\mathrm{dB}/20}$$

SNR Example

For example and SNR of:

+20 dB means that the signal RMS is 10 times the noise RMS (10(20/20) = 10);

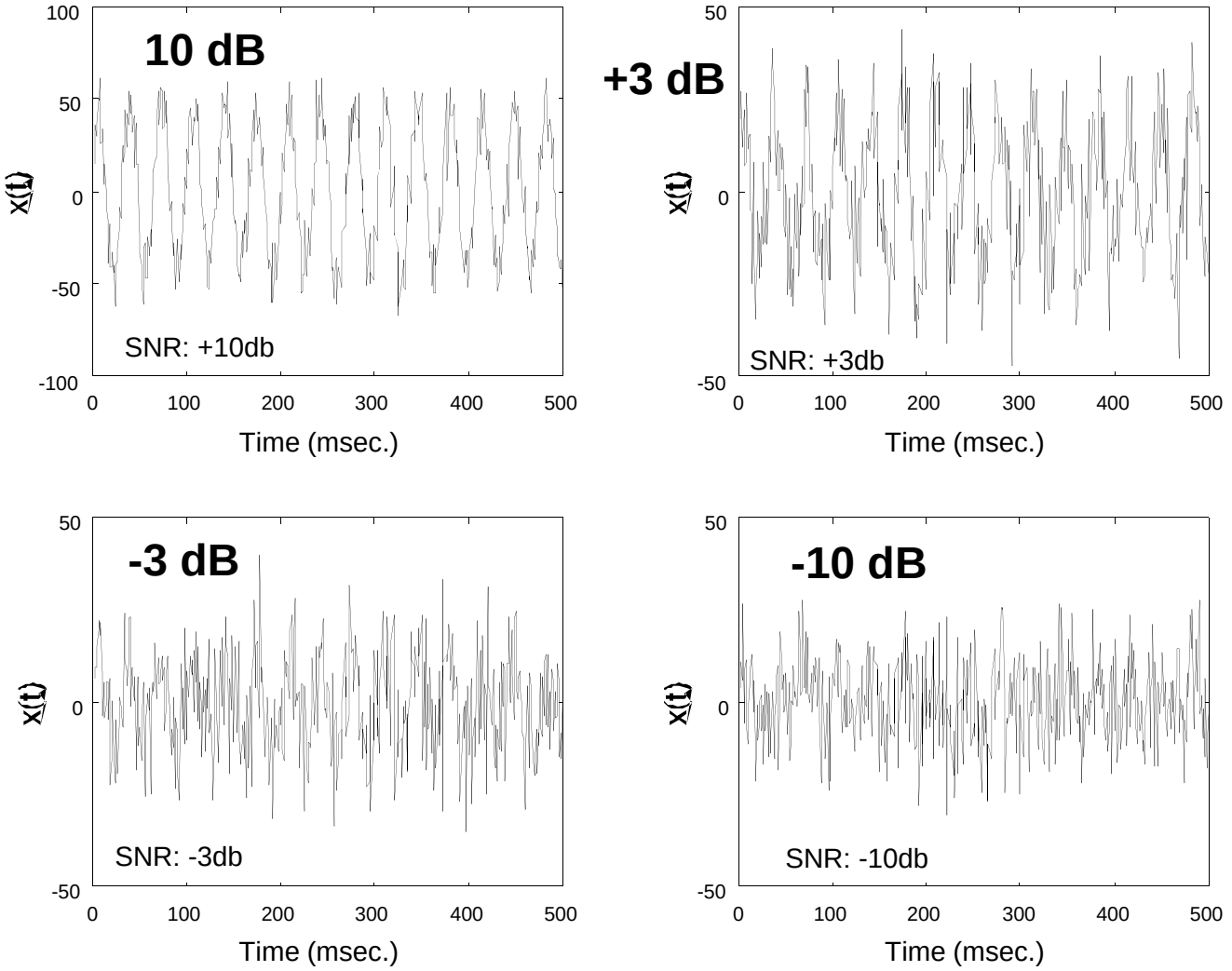
+3 dB indicates a ratio of 1.414 (10(3/20) = 1.414);

0 dB means the signal and noise have the same RMS values;

-3 dB means that the ratio is 1/1.414;

-20 db means the signal is 1/10 of the noise in RMS units.

Example of Different SNR Levels



A sinusoid with 4 levels of added noise. SNR's in dB.

NOISE AND VARIABILITY

What is noise? Noise is what you do not want and **signal** is what you do want (noise is unwanted variability). Noise often limits the usefulness or information content of a signal.

Where does noise come from?

Table 1-3 Sources of Variability

	Potential Remedy
Measurement only indirectly related to	Modify overall approach
Other sources of similar energy form	Noise cancellation Transducer design
Transducer responds to other energy	

Signal processing is often used to reduce the influence of noise.

Noise and Variability

What is noise? Noise is what you do not want and **signal** is what you do want (noise is unwanted variability). Noise often limits the usefulness or information content of a signal

Sources of Variability

Source	Cause	Potential Remedy
Physiological variability	Measurement only indirectly related to variable of interest	Modify overall approach
Environmental (internal or external)	Other sources of similar energy form	Noise cancellation, transducer design
Artifact	Transducer responds to other energy sources	Transducer design
Electronic	Thermal or shot noise	Transducer or electronic design

Noise and Variability

Variability really is noise, but the word is often used to indicate fluctuation between, as opposed to within, measurements.

A variety of signal processing tools exist to reduce noise.

The more that is known about the characteristics of the noise, the more powerful are the signal processing methods that can be applied.

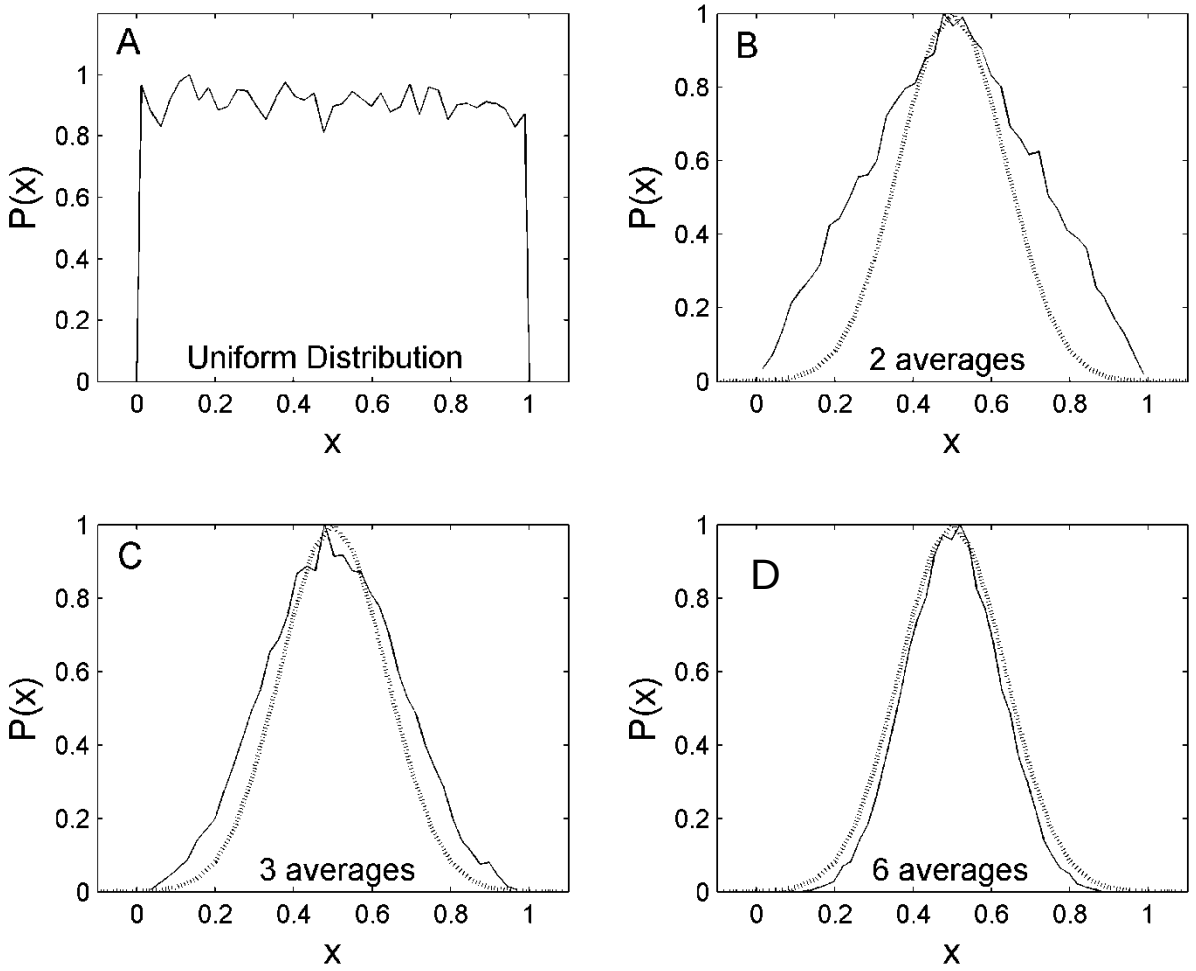
Noise Properties: Distribution Functions

- Since the noise is a random variable, describing it as a function of time is not useful. Common properties used to discuss noise include its probability distribution, variance, and spectrum.

- While noise can take on a variety of different probability distributions, the Central Limit Theorem implies that most noise will have a *Gaussian* or *normal* distribution.

- The Central Limit Theorem states that when noise is generated by a large number of independent sources, it will have a Gaussian probability distribution regardless of the probability distribution of the individual sources.

Gaussian Distribution: The Central Limit Theorem



The distribution of uniformly distributed random numbers before and after averaging.

A) No averaging.
B) 2 averages
C) 3 averages
D) 6 averages.

After 6 averages, the distribution of the average is close to Gaussian.

Gaussian distribution: $p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2}$

Example 2.3 Use a large data set generated by MATLAB'S **`randn`** function (Gaussian distribution) and determine the probability distribution. Also estimate the probability distribution of the data produced by **`rand`** (uniform distribution).

Solution: A histogram is a tabulation over the data set of the number of occurrences of a given range of values.

Counts of values that fall within a given range are stored in *bins* associated with that range.

The user usually decides how many bins to use.

Histograms are then plotted as counts against the range with value on the horizontal axis and counts on the vertical.

Bar-type plots are commonly used for plotting histograms.

Example 2.3 Solution (cont)

The MATLAB graphics routine **`hist`** evaluates the histogram. It has a number of options. The most useful calling structure for this example is:

```
[ht,xout] = hist(x,nu_bins);
```

where the inputs are data vector, **`x,`** and **`nu_bins`** is the number of bins desired.

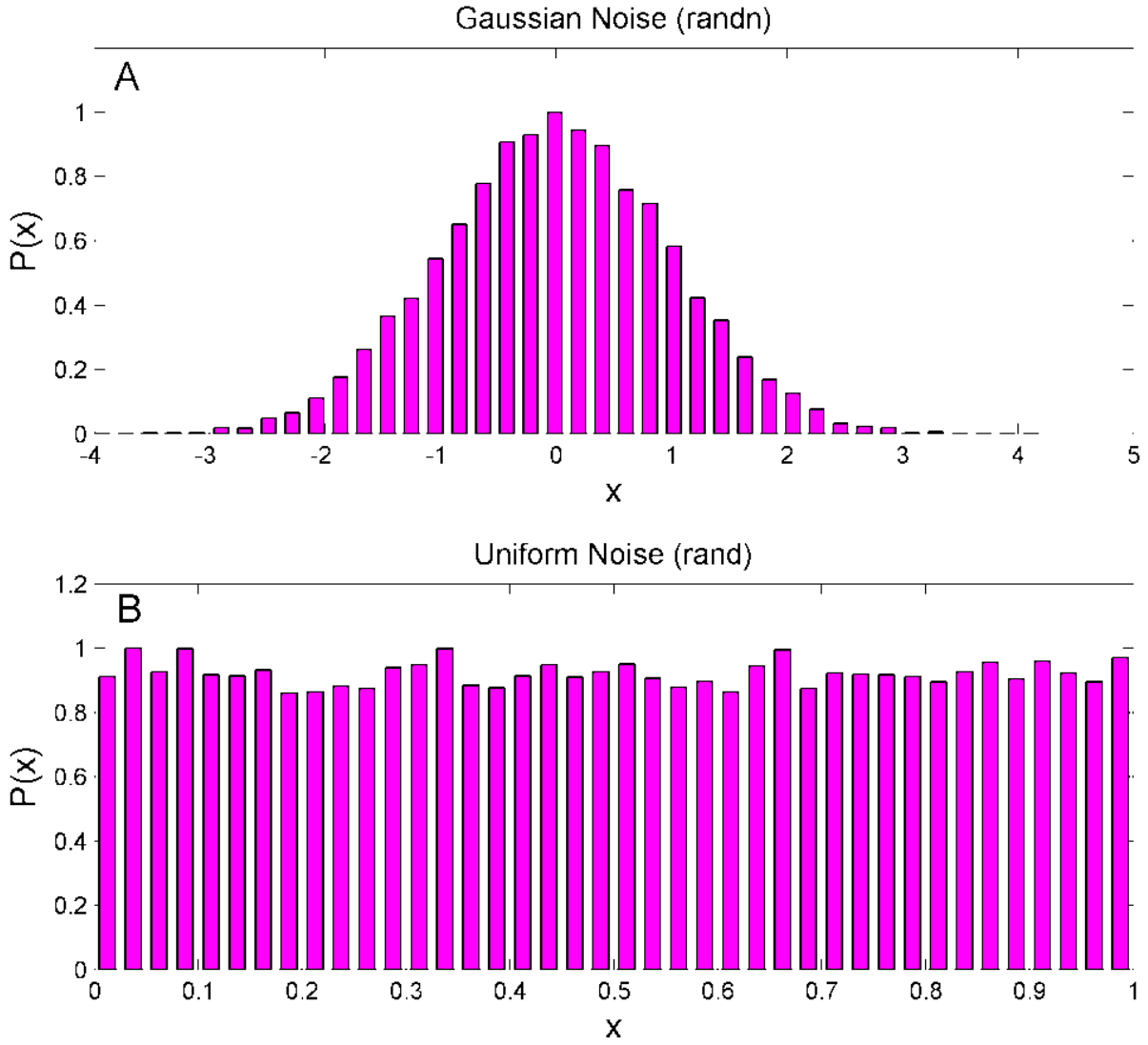
The outputs are the histogram vector, **`ht,`** and a vector, **`xout,`** which gives the mean of the ranges for the bins used. This vector is useful in scaling the horizontal axis when plotting.

Example 2.3 Solution (cont)

This example first constructs a large (20000-point) data set of Gaussainly distributed random numbers using **randn**, then uses **hist** to calculate the histogram and plot the results. This procedure will be repeated using **rand** to generate the data set.

```
% Example 2.3  Evaluation of the distribution of data produced
% by MATLAB's rand and randn functions.
%
N = 20000;                          % Number of data points
nu_bins = 40;                       % Number of bins
y = randn(1,N);                     % Generate random Gaussian noise
[ht,xout] = hist(y,nu_bins);        % Calculate histogram
ht = ht/max(ht);                    % Normalize histogram to 1.0
bar(xout, ht);                      % Plot as bar graph
      ..... Label axes and title .....
      .....Repeat for rand ........
```

Example 2.3 Results: The bar graphs produced by this example are shown to be very close to the Gaussian distribution for the **randn** function and close to flat for the **rand** function.



The vertical axis is often labeled "Counts," "Frequency," or "Frequency of Occurrence."

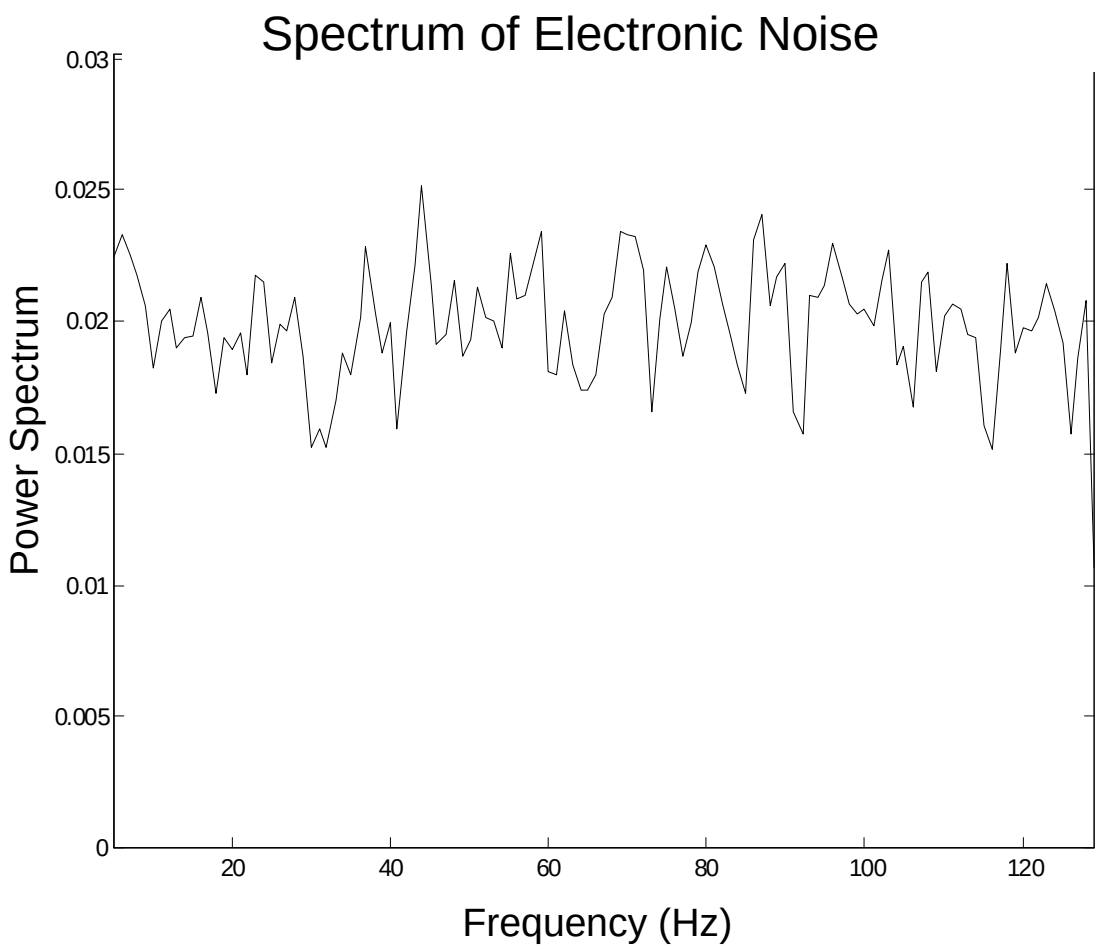
Electronic Noise

Electronic noise is the only noise that can be definitively described.

Electronic noise has energy at all frequencies.

To give a number to the amount of noise present, a range of frequencies must be specified.

A ‘range of frequencies’ is also known as ‘bandwidth.’



Electronic noise (cont)

- Johnson or thermal noise is produced by resistance sources and the amount of noise generated is related to the resistance and to the temperature (as well as the bandwidth).

$$V_J = \sqrt{4kT\,R\,BW} \text{ volts}$$

where R is the resistance in ohms, T the temperature in degrees Kelvin, BW the range of frequencies in Hz, and k is Boltzman's constant (k = 1.38 x 10-23 J/^{O}K).

$$I_J = \sqrt{4kT\ BW/R} \text{ amps}$$

- If noise current is of interest, the equation for Johnson noise current can be obtained from the above equation in conjunction with Ohm's law.

Electronic Noise (cont)

- **Shot noise** is defined as current noise and is proportional to the baseline current through a semiconductor junction:

$$I_s = \sqrt{2q\, I_d\, BW} \quad \text{amps}$$

- When **multiple noise sources** are present, their voltage or current contributions add as the square root of the sum of the squares (assuming independent sources): For voltages:

$$V_T = \sqrt{V_1^{\,2} + V_2^{\,2} + V_3^{\,2} + \ldots V_N^{\,2}}$$

Assuming the voltages are all uncorrelated.

Example (not in text) A 20-ma current flows through both a diode (i.e. a semiconductor) and a 200-Ω resistor. What is the total current noise? Assume a bandwidth of 1 MHz (1 x10^6 Hz) and room temperature. (A temperature of 310 ^{O}K is often used as room temperature, in which case $4kT$ = 1.7 x 10^{-20} J.)

Solution. Find the shot noise contributed by the diode and the Johnson noise contributed by the resistor, then combine them.

$$i_{nd} = \sqrt{2qI_dBW} = \sqrt{2(1.66x10^{-19})(20x10^{-3})10^6} = 8.15x10^{-8}\ amps$$

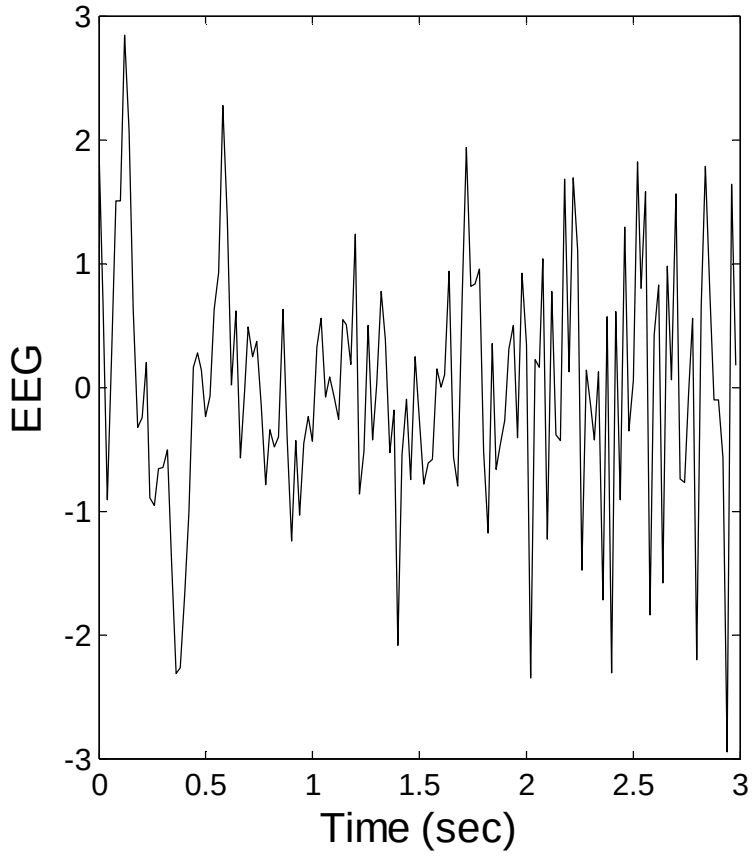
$$i_{nR} = \sqrt{4kT\ BW/R} = \sqrt{1.7x10^{-20}(10^6/200)} = 9.2x10^{-9}\ amps$$

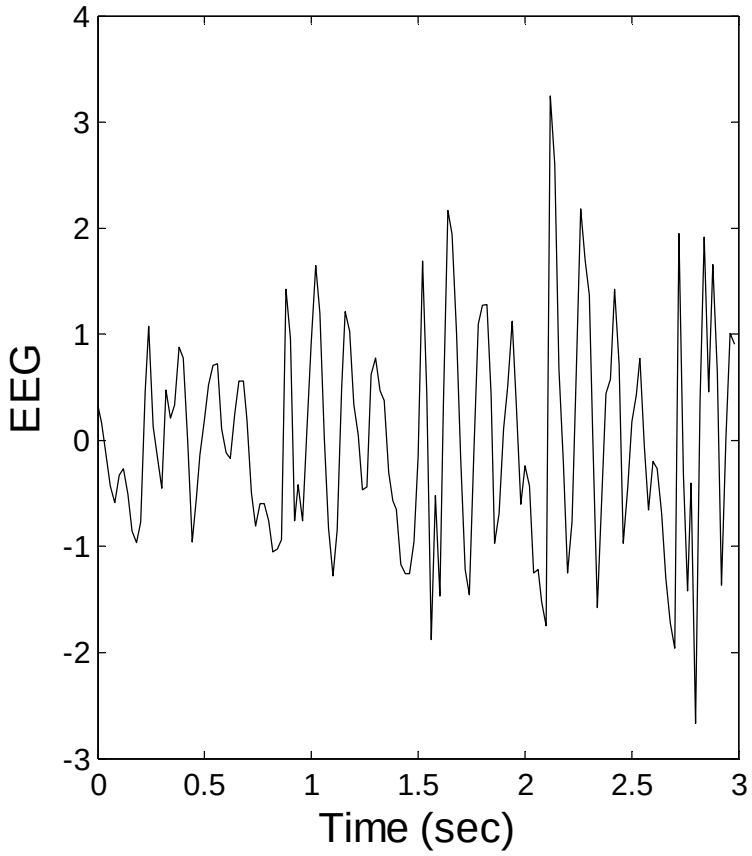
$$i_{nT} = \sqrt{i_{nd}^2 + i_{nR}^2} = \sqrt{6.64x10^{-15} + 8.46x10^{-17}} = 8.20x10^{-8}\ amps$$

Data Functions and Transforms

Basic measurements do not definitively describe signals.

For example, these two EEG segments have the same mean, RMS, and variance, but are clearly different.





Describing Signals

We would like some method to capture the differences between these two (and other) signals, and preferably to be able to quantify these differences.

Other functions (or waveforms) can be used to describe signals and their differences.

In signal processing, functions fall into two categories:

1) Data, including waveforms and images;

2) Functions that operate on data.

Transformations: Functions that Operate on Signals

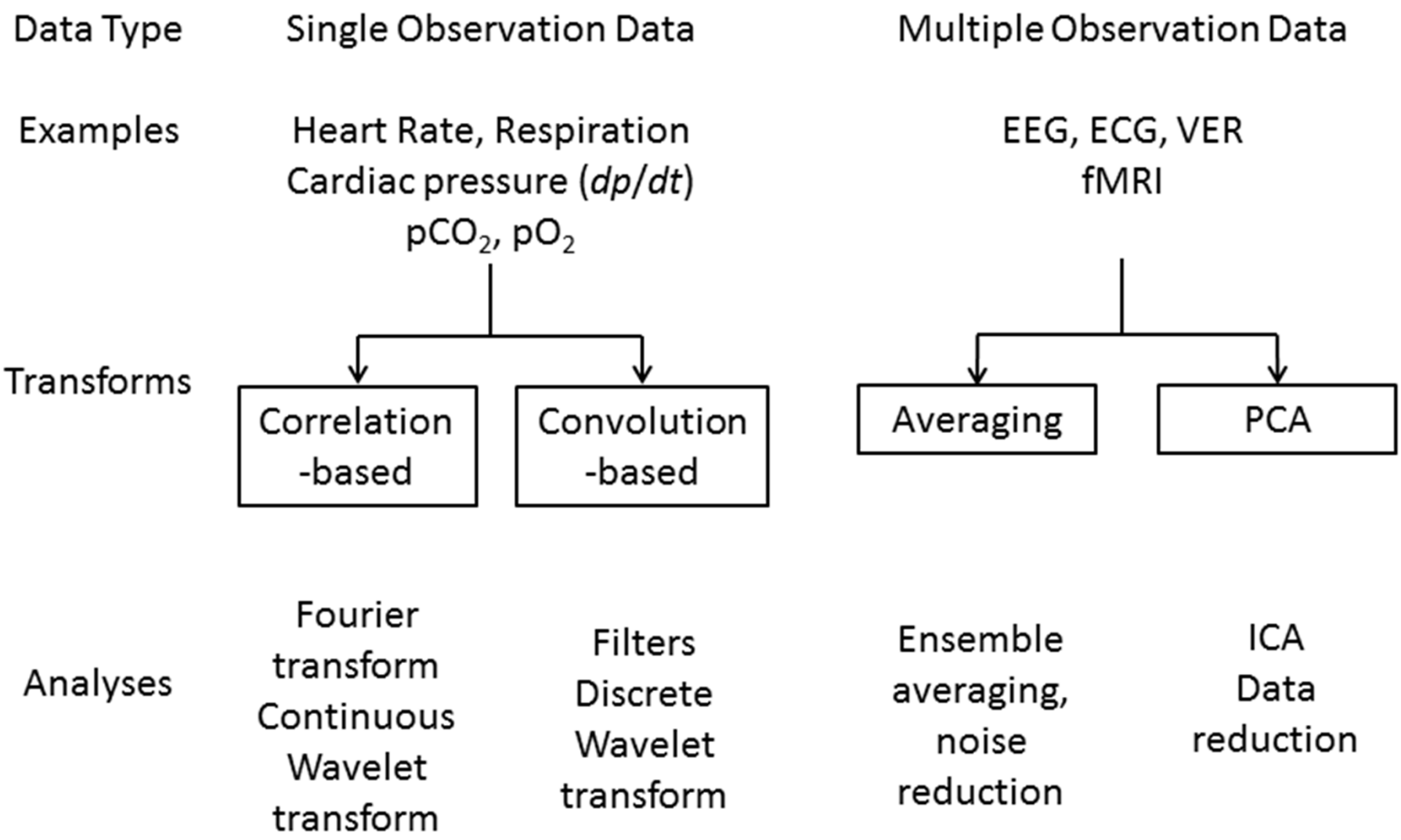
Transformations are operators that modify data.

Transformations are used to:

1.Improve data quality by removing noise as in filtering (Chapter 4);
2.Make the data easier to interpret as with the Fourier transform (Chapter 3); or
3.Reduce the size of the data by removing unnecessary components as with the Wavelet transform (Chapter 7) or principal component analysis (Chapter 9).

Transformations used here

depend on data type



Comparing Waveforms: Correlation

Correlation seeks to quantify how much one function (i.e., signal) is like another. (Mathematical correlation does a pretty good job of describing similarity, but once in a while it breaks down so that some functions that are conceptually similar, such as sine and cosine waves, have a mathematical correlation of zero.)

All correlation-based approaches involve taking the sum of the sample-by-sample product of the two functions:

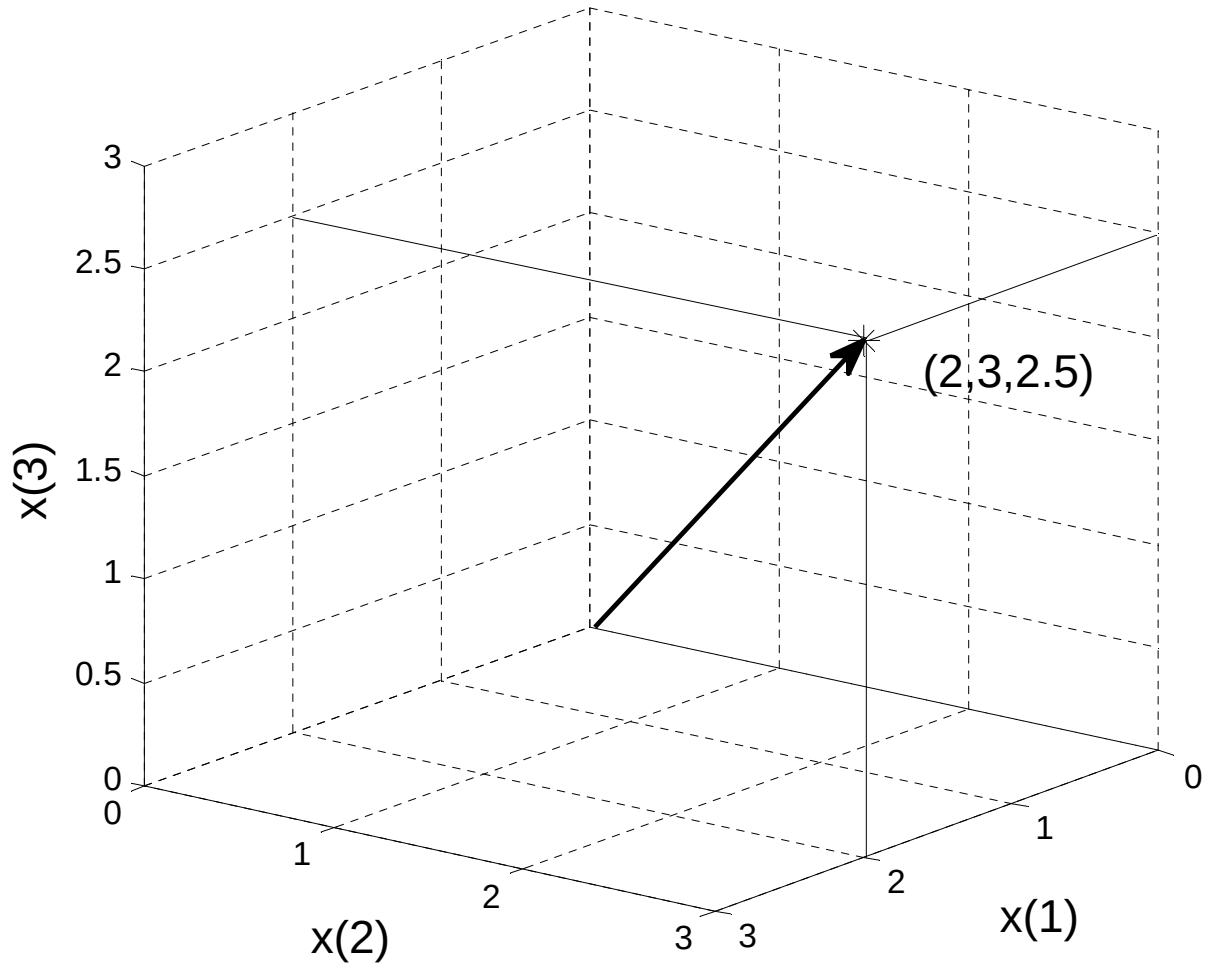
$$r_{xy} = x[1]y[1] + x[2]y[2] + x[3]y[3] + ... + x[N]y[N]$$

$$== \sum_{n=1}^{N} x_n y_n$$

where r_{xy} is used to indicate correlation and the subscripts x and y indicate what is being correlated. Different normalizations ($\frac{1}{N}\sum_{n=1}^{N} x_n y_n$) can be used as described later.

Vector Representation

A string of numbers can be thought of as a vector in N-dimensional space: $x[n] = [x_1, x_2, x_3, \ldots x_n]$



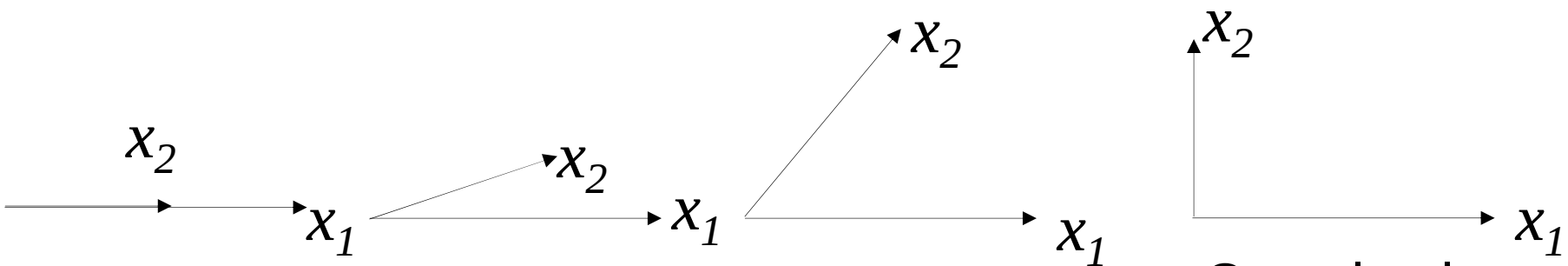
Data sequence: $x[n] = 2, 3, 2.5$ represented as a vector in 3-dimensional space.

This curious way of thinking about a data string does have its uses.

Signal Comparison using Vector Representation

The concept of thinking of a string of numbers (or signal) as a vector is useful when comparing two signals.

If two strings are mathematically similar, their vector representations will project on one another:



Completely similar
$\theta = 0$ deg.

Highly similar
$\theta =$ small

Moderately similar
$\theta =$ larger

Completely different
$\theta = 90$ deg

Shown here for only two dimensions, but generalizes to *N* dimensions.

Correlation and the Scalar Product

The projection of one vector on another is found by taking the *scalar product* of the two vectors.* This shows the relationship between vector projection and correlation. The scalar product is defined as:

.

$$Scalar\ product\ of\ x\ \&\ y \equiv \langle x, y \rangle = \begin{bmatrix} x_1 \\ x_2 \\ \mathrm{M} \\ x_N \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \mathrm{M} \\ y_N \end{bmatrix}$$

$$= x_1 y_1 + x_2 y_2 + \ldots + x_N y_N$$

$$= \sum_{n=1}^{N} x_n y_n$$

*The scalar product is also termed: the *inner product*, *the standard inner product*, or the *dot product.*

Scalar Product – Correlation (cont)

Note that the scalar product results in a single number (i.e., a scalar), not a vector.

The scalar product can also be defined in terms of the magnitude of the two vectors and the angle between them:

$$Scalar\ product\ of\ \ x\ \&\ y = \langle x, y \rangle = |x||y|\cos\theta$$

where θ is the angle between the two vectors.

Projection (correlation) is an excellent way to compare two signals or to compare a signal with a 'probing' or 'test' waveform.

In MATLAB, the scalar product is just: **`sum(x.*y)`**

Example 2.5 Find the angle between two short signals represented as vectors. Give the angle in deg. The signals are:

$$x = [1.7, 3, 2.2] \text{ and } y = [2.6, 1.6, 3.2].$$

Since these signals are short, plot the two vector representations in three dimensions.

Solution. Construct the two vectors and find the scalar product using the last equation (the "angle" equation). When multiplying the two vectors, be sure to use MATLAB's **.*** operator to implement a point-by-point multiplication.

Example 2.5 Solution (cont)

Recall that the magnitude of a vector can be found using:

$$|x| = \sqrt{x[1]^2 + x[2]^2 + x[3]^2}$$

Solve the "angle" dot product equation for *θ:*

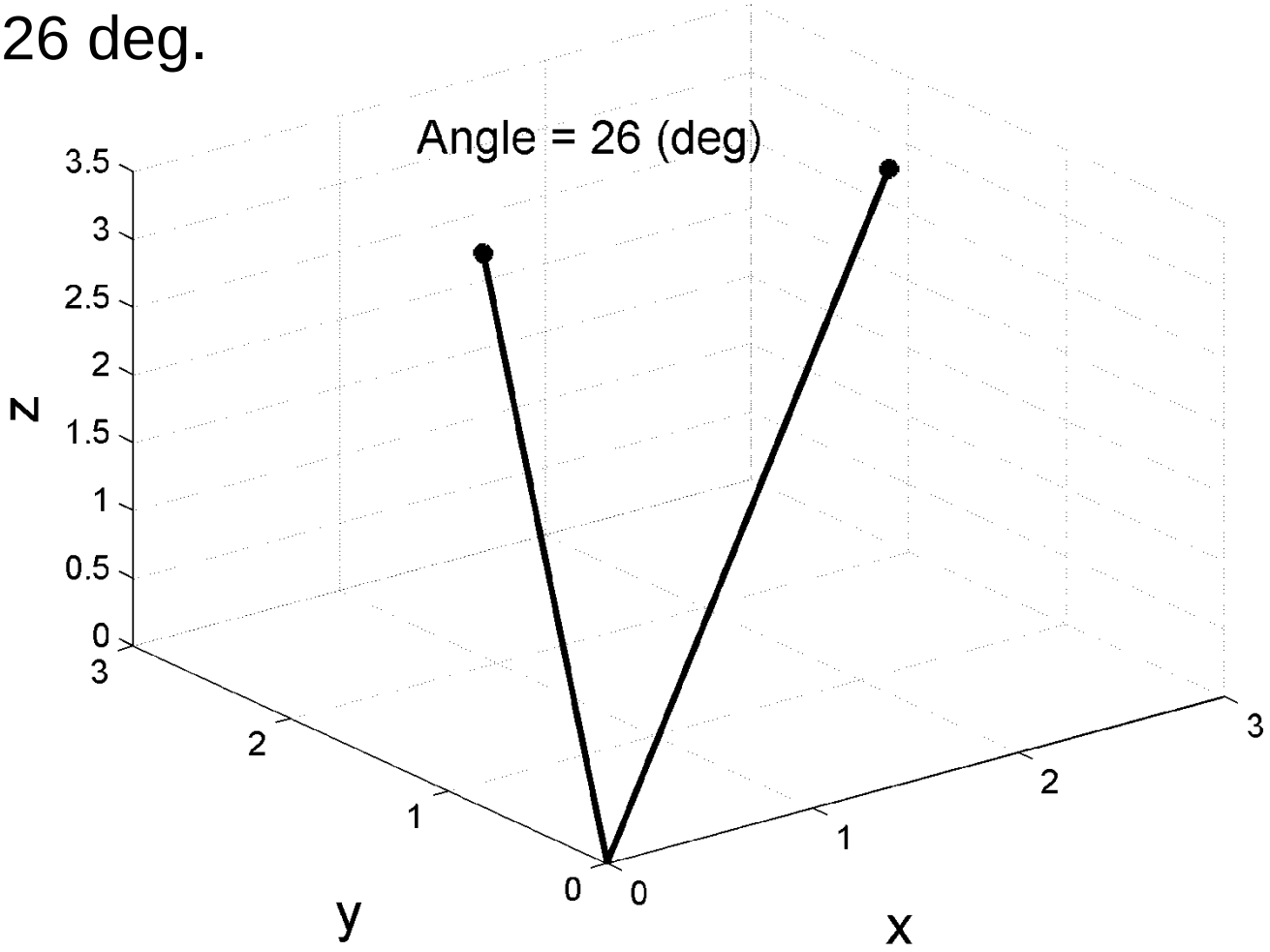
$$\langle x, y \rangle = |x||y|\cos\theta; \quad \cos\theta = \frac{\langle x, y \rangle}{|x||y|} \qquad \theta = \cos^{-1}\left(\frac{\langle x, y \rangle}{|x||y|} \right)$$

This equation is used in the MATLAB program in the next slide.

```
% Example 2.5 Find the angle between two vectors
%
x = [1.7, 3, 2.2];                    % Generate the vectors
y = [2.6, 1.6, 3.2];
sp = sum(x.*y);                       % Take the scalar (dot) product
mag_x = sqrt(sum(x.^2));              % Calculate mag. x vector
mag_y = sqrt(sum(y.^2));              % Calculate mag. y vector
cos_theta = sp/(mag_x*mag_y); % Calculate cosine of theta
angle = acos(cos_theta);              % Take the arc cosine and
angle = angle*360/(2*pi);             %    convert to degrees
%
% Plot in 3-D
hold on;
plot3(x(1),x(2),x(3),'k*');           % Plot x vector end point
plot3([0 x(1)],[0 x(2)],[0 x(3)]);    % Plot x vector line
plot3(y(1),y(2),y(3),'k*');           % Plot y vector end point
plot3([0 y(1)],[0 y(2)],[0 y(3)]);    % Plot vector y line
title(['Angle = ',num2str(angle,2),' (deg)']);    % Output angle
grid on;
```

Example 2.5 Result: The plot representing the two vectors is shown below.

The angle between the two vectors is calculated to be 26 deg.



If these were signals, the fairly small angle would indicate some correlation between them.

Orthogonality

- Orthogonal signals and functions are very useful in a variety of signal processing tools.
- In common usage, “orthogonal” means perpendicular: if two lines are orthogonal they are perpendicular.
- In vector representation, orthogonal signals would have orthogonal vectors.
- The formal definition for orthogonal signals is that their correlation (or scalar product) is zero:

$$\sum_{n=1}^{N} x[n]y[n] = 0$$

Orthogonality (cont)

- An important characteristic of signals that are orthogonal (i.e., uncorrelated) is that when they are combined or added together they do not interact with one another.

- Orthogonality simplifies many calculations and some analyses could not be done, at least not practically, without orthogonal signals.

- Orthogonality is not limited to two signals. Whole families of signals can be orthogonal (or orthonormal*) and are called *orthogonal* or *orthonormal sets*.

***** Orthonormal vectors are orthogonal, but also have unit length.

Example 2.6 Generate a 500-point, 2-Hz sine wave and a 4-Hz sine wave of the same length. Make T_T = 1 sec. Are these two waveforms orthogonal?

Solution. Generate the waveforms. Since N = 500 and T_T = 1 sec, $T_s = T_T/N$ = 0.002 sec.

Find the scalar product of these waveforms.

If the result is near zero the waveforms are orthogonal.

The MATLAB code is shown in the next slide.

```
% Example 2.6 Evaluate 2 waveform for Orthogonality.
%
Ts = 0.002;              % Sample interval
N = 500;                 % Number of points
t = (0:N-1)*Ts;          % Time vector
f1 = 2;                  % Frequency of sine wave 1
f2 = 4;                  % Frequency of sine wave 2
%
x = sin(2*pi*f1*t);      % Generate sine wave 1
y = sin(2*pi*f2*t);      % Generate sine wave 2
Corr = sum(x.*y);        % Scalar product
disp(Corr)
```

Result: The correlation produced by this program is **1.9657e-014**, very close to zero showing that the waveforms are orthogonal.

This is expected, since harmonically related sines (or cosines) are known to be orthogonal.

Basis Functions

- A transform can be thought of as a re-mapping of the original data into something that provides more information.

- Many transforms described here are achieved by comparing the signal of interest with some sort of probing function or a whole family of probing functions termed a *basis*.

- Usually the basis is simpler than the signal, for example, a sine wave or series of sine waves. (As shown in Chapter 3, a sine wave is about as simple as a waveform gets.)

Discrete Domain Comparisons

A quantitative comparison can tell you how much your complicated signal is like a simpler basis or reference family.

To compare a waveform with a number of functions that form the basis requires a simple modification of the basic correlation equation so that one of the functions becomes a family of functions, $f_m[n]$.

$$X[m] = \sum_{n=1}^{N} x[n] f_m[n]$$

When the comparison is made with a family of functions, a series of correlation values is produced, $X[m]$, one for each family member, m.

Comparisons in the Continuous Domain.

The same comparison can be made in the continuous domain, where the signal and basis are continuous time-domain functions and summation becomes integration.

$$X(m) = \int_{-\infty}^{\infty} x(t)\, f_m(t) dt$$

where $x(t)$ is a continuous signal and $f_m(t)$ is the set of continuous basis functions.

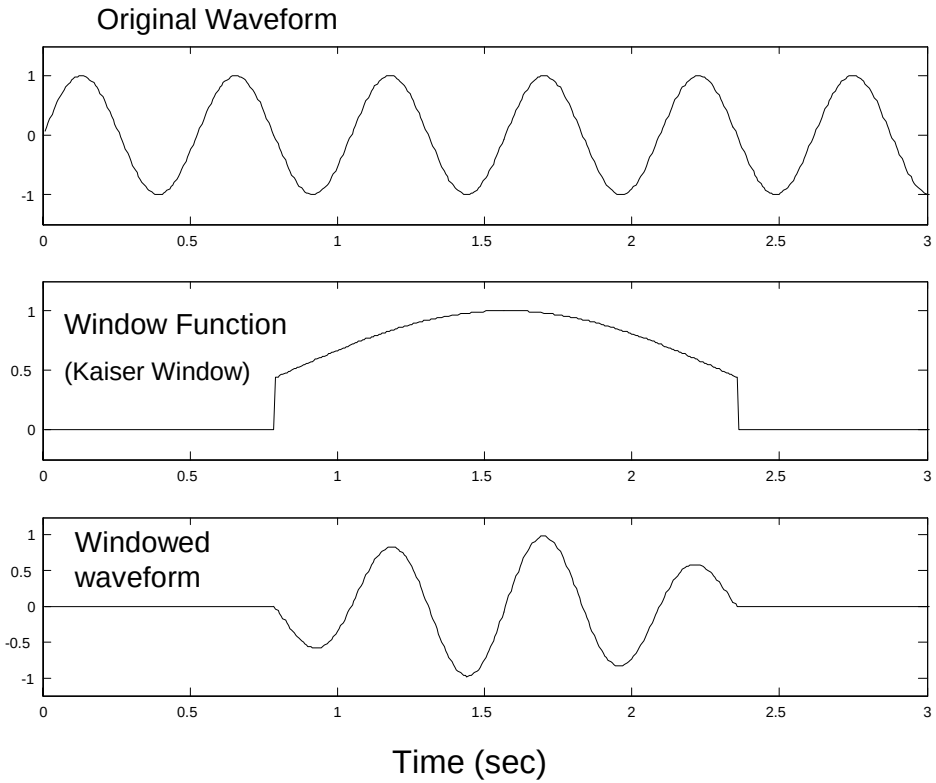
Mismatch Between Signal and Basis Length

- If the length of the basis, $f_m[n]$, is shorter than the waveform, then the comparison can only be carried out on a portion of $x[n]$.

- The signal can be segmented by truncation, cutting out the desired portion, or by multiplying the signal by yet another function that is zero outside the desired portion.

- A function used to segment a waveform is termed a *window* function and its application is illustrated in the next slide.

Window Functions



Window functions can reduce the signal length.

Simple truncation is the same as multiplying the signal by a rectangular window function.

When a window function is used, the correlation (scalar product) becomes:

$$X[m] = \sum_{n=1}^{N} x[n]\, f_m[n]\, W[n]$$

where $W[n]$ is the window function.

Sliding Projections (Correlations)

It is possible to do a sliding comparison, where the shorter function slides along the longer function and a correlation is made at every position.

This is called *crosscorrelation.*

$$X[m,k] = \sum_{n=1}^{N} x[n] \;\; f_m[n+k]$$

where *k* is the integer that 'slides' waveform f_m

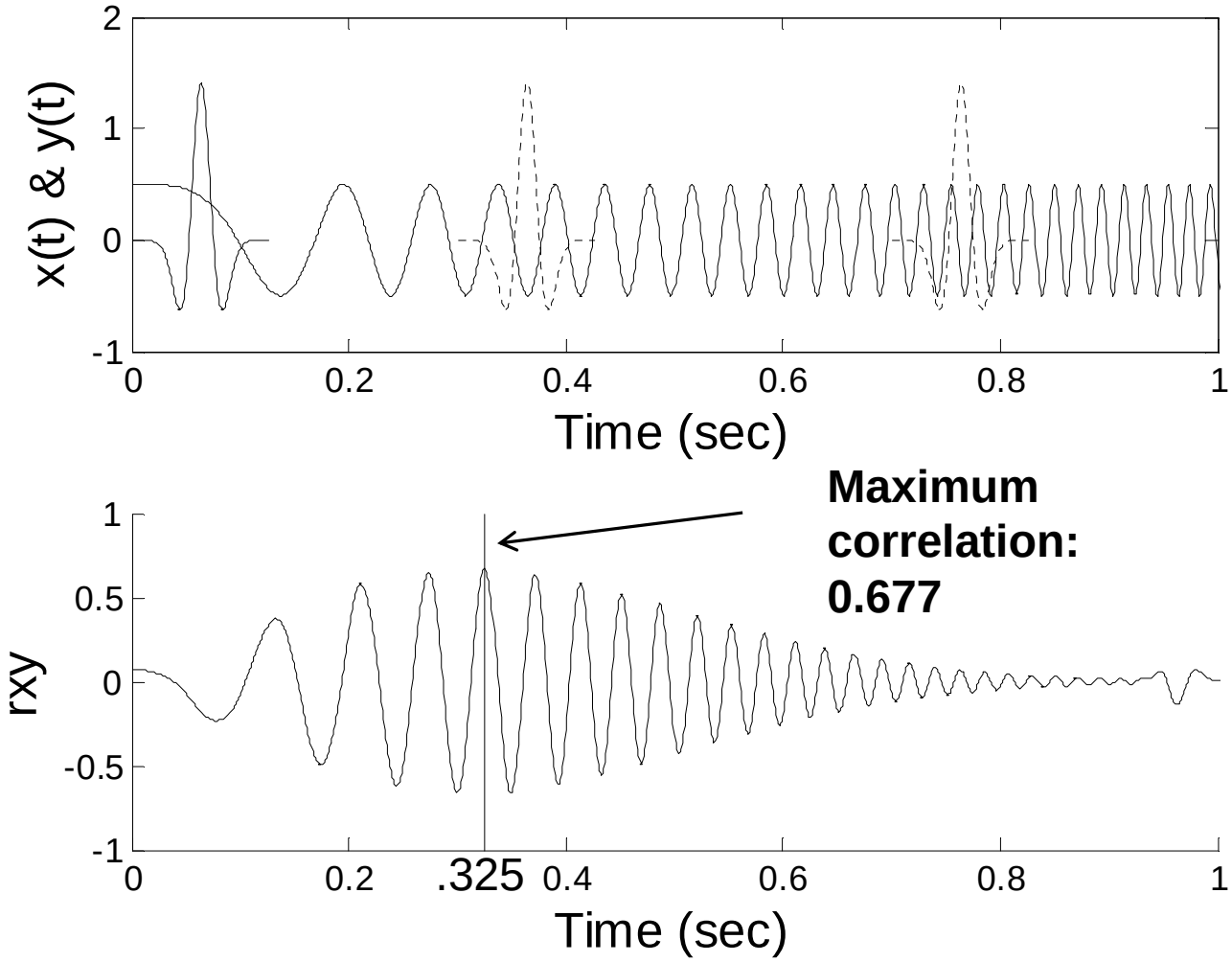
It is possible to both window and slide in one operation. This is used in the Short-Term Fourier Transform.

$$X[m,k] = \sum_{n=1}^{N} x[n]\left(W[n+k] \;\; f_m[n]\right)$$

Crosscorrelation (cont)

Unlike correlation, crosscorrelation can be done using signals that have different lengths (upper graph).

Peak correlation values indicate where the original signal is most similar to the reference function.



The probing function one member of a basis) slides along the signal of interest: a sinusoid that continuously increases in frequency.

Other Correlation Analyses: Correlation and Covariance

Covariance computes the variance that is shared between two (or more) waveforms.

The equation is similar to that of basic correlation except that the means are removed from the two waveforms and the correlation sum is normalized by dividing by $N - 1$:

$$\sigma_{xy} = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \bar{x})(y_n - \bar{y})$$

where σ_{xy} is the covariance and $\bar{x}$ and $\bar{y}$ are the means of x and y.

Pearson Correlation Coefficient

A popular variation of the basic correlation equation normalizes the correlation sum so that the outcome lies between ± 1.

This correlation value, known as the Pearson correlation coefficient, is obtained using a modification of the covariance equation:

$$r_{xy\ Pearson} = \frac{1}{(N-1)\sqrt{\sigma_x^2 \sigma_y^2}} \sum_{n=1}^{N} (x - \bar{x})(y - \bar{y})$$

where σ_x and σ_y are the variances and $\bar{x}$ and $\bar{y}$ are the means of *x* and *y*.

Pearson Correlation Coefficient (cont)

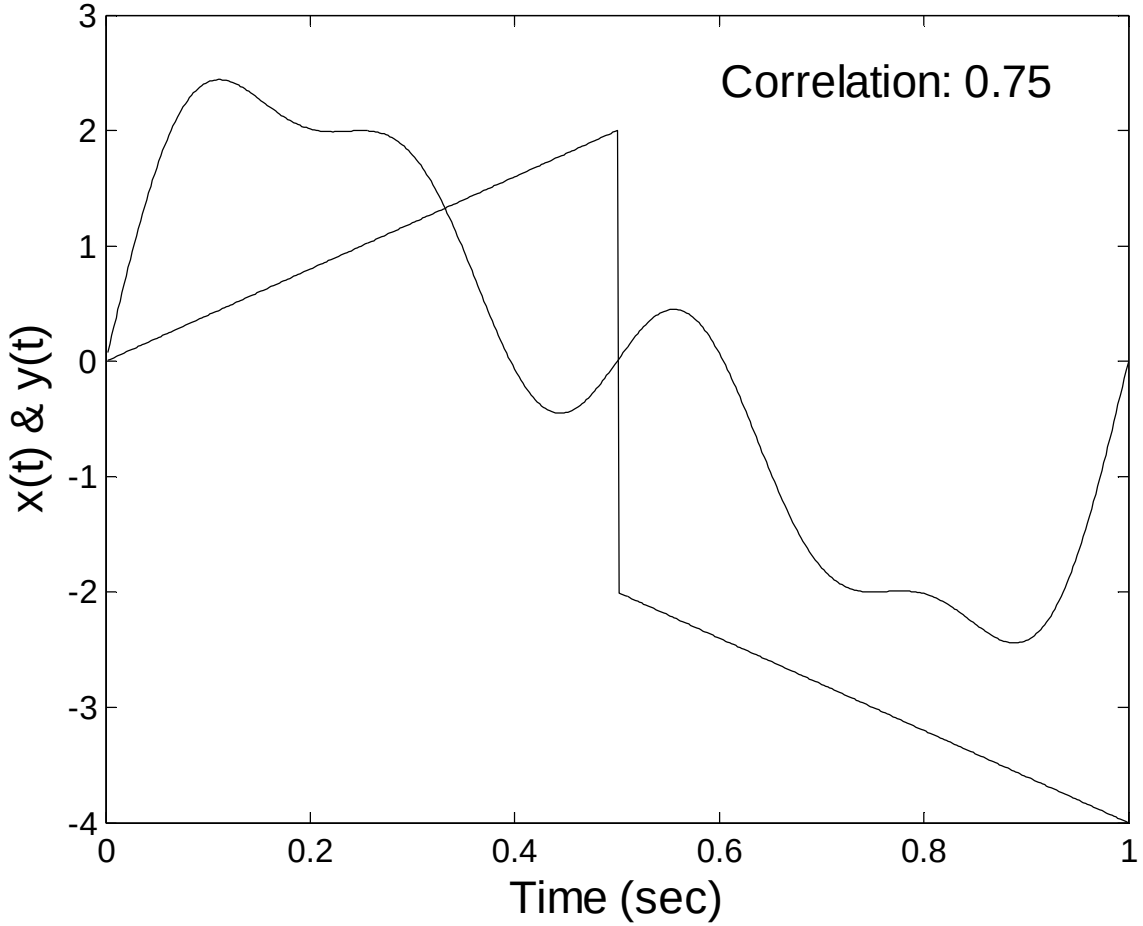
If the means of the waveforms are removed, the Pearson correlation coefficient can be obtained directly from the basic correlation equation using:

$$r_{xy\,Pearson} = \frac{r_{xy}}{(N-1)\sqrt{\sigma_1{}^2\sigma_2{}^2}}$$

This will make the correlation value equal to +1 when the two signals are identical and -1 if they are exact opposites.

In this text, the term “Pearson correlation coefficient” or just “correlation coefficient” implies this normalization, while the term “correlation” is used more generally to mean normalized or un-normalized correlation.

Example 2.7 Find the Pearson correlation coefficient between the two waveforms shown in Figure 2.14. These waveforms are stored as variables **x** and **y** in file **Ex2_7.mat**.



Solution: Load the file and find the un-normalized correlation using the approach in Example 2.5 and apply the last equation. Subtract the means of each signal before correlation.

```
% Example 2.7  Pearson correlation coefficient between two
%   waveforms.
%
load Ex2_7                                  % Load the signals
N = length(x);                              % Find N
% Subtract means and apply basic correlation
rxy = sum((x-mean(x)).*(y-mean(y)));
rxy = rxy/((N-1)*sqrt(var(x)*var(y)));      % Apply last eq.
title(['Correlation: ',num2str(rxy)]);      % Output correlation
```

Results: The Pearson correlation coefficient was found to be 0.75 indicating similarity between the waveforms.

Covariance Matrix: Covariances Between Multiple Signals

When more than two signals are involved, the covariance matrix shows the variances of each signal on the diagonals and the covariances on the off-diagonals.

$$S = \begin{bmatrix} \sigma_{1,1}^2 & \sigma_{1,2}^2 & \Lambda & \sigma_{1,N}^2 \\ \sigma_{2,1}^2 & \sigma_{2,2}^2 & \Lambda & \sigma_{2,N}^2 \\ M & M & O & M \\ \sigma_{N,1}^2 & \sigma_{N,2}^2 & \Lambda & \sigma_{N,N}^2 \end{bmatrix}$$

$\sigma^2_{i,i}$ is the variance of waveform *i*

$\sigma^2_{i,j}$ shows the way that waveform *i* and waveform *j* vary together, i.e., the covariance

MATLAB computes the covariance matrix using:

```
S = cov(X);          % Signal covariances
```

where **X** is a matrix that contains the various signals to be compared in columns

Matrix of Correlations: Correlations Between Multiple Signals

The matrix of correlations is similar to the covariance matrix, except the values are normalized as in the Pearson Correlation Coefficient. Hence the diagonals in the correlation matrix would all be one.

$$R_{xx} = \begin{bmatrix} r_{1,1} & r_{1,2} & \Lambda & r_{1,\,N} \\ r_{2,1} & r_{2,2} & \Lambda & r_{2,\,N} \\ M & M & O & M \\ r_{N,1} & r_{N,2} & \Lambda & r_{N,\,N} \end{bmatrix}$$

MATLAB Computes the matrix of correlations using:

```
S = cov(X);              % Signal covariances
```

where **x** is a matrix that contains the various signals to be compared in columns.

Example 2.8 Determine if a sine wave and a cosine wave at the same frequency are orthogonal and if sine waves at harmonically related frequencies are orthogonal. The term “harmonically related” means that sinusoids are related by frequencies that are multiples. Thus the signals sin(2*t*), sin(4*t*), and sin(6*t*) are harmonically related. Also determine if sawtooth and sine waves at the same frequency are orthogonal.

Solution: Generate a 500-point, 1.0-sec time vector. Use this time vector to generate a data matrix in which the columns represent 1.0-, 2- and 2.5-Hz cosine and sine waves. Apply the covariance and correlation MATLAB routines (i.e., **`cov`** and **`corrcoef`**) and display results.

```
% Example 2.8 Application of the covariance matrix to
%  sinusoids that are orthogonal and a sawtooth
%
N = 1000;                 % Number of points
Tt = 2;                   % desired total time
fs = N/Tt;                % Calculate sampling frequency
t = (0:N-1)/fs;           % Time vector
X(:,1) = cos(2*pi*t)';    % Generate a 1 Hz cosine
X(:,2) = sin(2*pi*t)';    % Generate a 1 Hz sine
X(:,3) = cos(4*pi*t)';    % Generate a 2 Hz cosine
X(:,4) = sin(4*pi*t)';    % Generate a 1 Hz sine
%
S = cov(X)                % Solve for covariance matrix
Rxx = corrcoef(X)         %      and matrix of correlations
```

Example 2.8 Results Covariance matrix, S, and correlation matrix, $R_{xx.}$ Note that sines and cosines are not correlated, nor are sines or cosines at multiple frequencies. Hence, they are orthogonal.

S =

0.5005	**-0.0000**	**-0.0000**	**0.0000**
-0.0000	**0.5005**	**-0.0000**	**0.0000**
-0.0000	**-0.0000**	**0.5005**	**0.0000**
0.0000	**0.0000**	**0.0000**	**0.5005**

Rxx =

1.0000	**-0.0000**	**-0.0000**	**0.0000**
-0.0000	**1.0000**	**-0.0000**	**0.0000**
-0.0000	**-0.0000**	**1.0000**	**0.0000**
0.0000	**0.0000**	**0.0000**	**1.0000**

Crosscorrelation and Autocorrelation

- The mathematical dissimilarity between sine and cosine is a problem when trying to determine general patterns such as the oscillatory characteristic of a sinusoid.

- Correlating a waveform with a sine wave might lead you to believe it does not exhibit sinusoidal behavior even if it is a perfect cosine wave.

- One way to circumvent this miss-identification would be to use crosscorrelation mentioned previously to asses the correlation between the two waveforms at many different time (or phase) shifts.

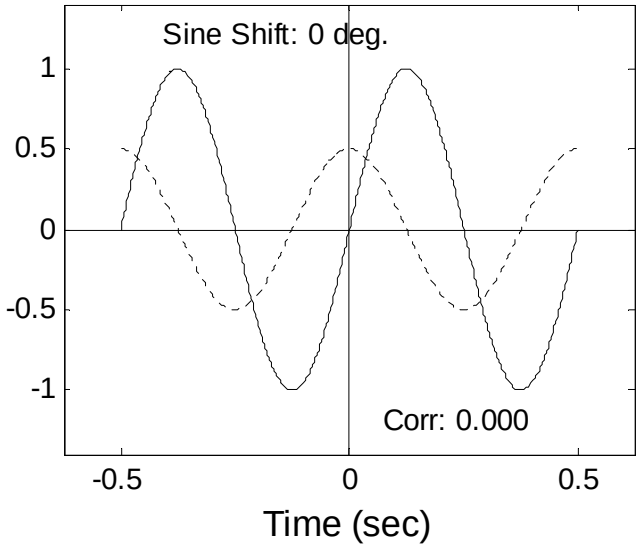
- Crosscorrelation is guaranteed to find the best match between any two signals.

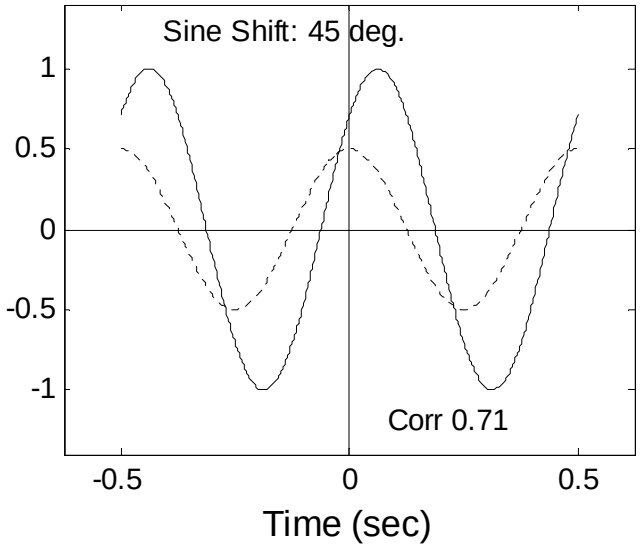
Effect of Shifting on Correlation

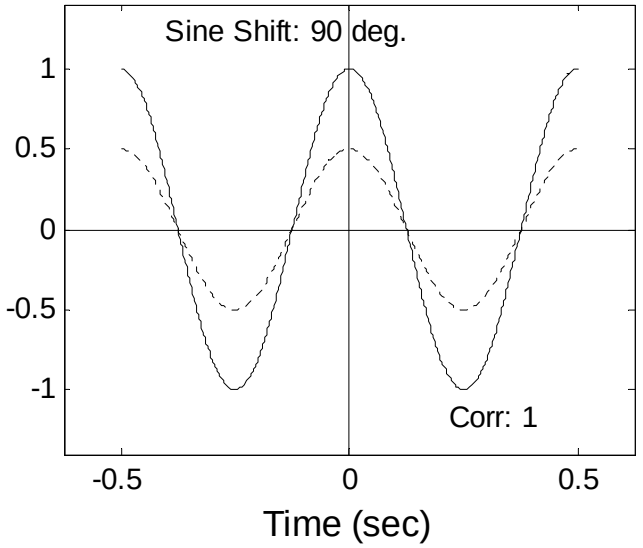
Shifting one sinusoid with respect to another probes all the possible relative positions.

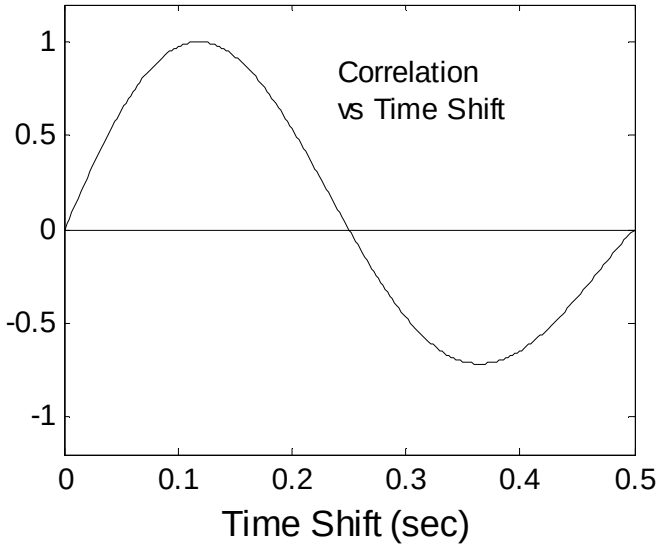
The maximum correlation occurs when the two are in phase and is 1.0.

When the two are out of phase the correlation is –1.









Crosscorrelation (revisited)

The equation for crosscorrelation has been given previously and is repeated here:

$$r_{xy}[k] = \frac{1}{N}\sum_{n=1}^{N} y[n]x[n+k]$$

where *k* specifies the shift in samples and $r_{xy}[k]$ is a series of correlations as a function of shift.

The shift is often called *lags* and will be ≤ ± *N*. If the signals were originally time functions, the lags may be converted to time shifts in secs.

It does not matter which function is shifted with respect to the other, the results will be the same.

Crosscorrelation (cont)

The two waveforms need not be the same length; however, even if one waveform is much shorter, there will be points missing in the unshifted waveform when the shift becomes large enough.

Missing points at the end can be added using padding to extend a waveform's length. This so-called "zero-padding" is commonly used.

If correlations are done at all possible shift positions, positive and negative, the maximum shift is the combined length of the two data sets minus one.

Crosscorrelation: MATLAB Implementation

In MATLAB, crosscorrelation could be implemented using the routine **`xcorr.`** The most common calling structure for this routine is:

```
[rxy,lags] = xcorr(x,y,maxlags);    % Crosscorrelation
```

where x and y are the waveforms to be crosscorrelated and **`maxlags`** specifies the shift range as ± **`maxlags`**. If that argument is omitted, the default maximum shift is: **`length(x) + length(y) – 1`**.

The **`xcorr`** function is part of the Signal Processing toolbox.

Crosscorrelation: MATLAB Implementation

A similar routine, **`axcor`**, can be found in the accessory material. The routine has the same calling structure except there is no option for modifying the maximum shift: it is always **`length(x) + length(y) – 1`**.

```
[rxy,lags] = axcor(x,y);                % Crosscorrelation
```

Although crosscorrelation operations usually scale by 1/*N*, this routine uses scaling that gives the output as Pearson's correlation coefficients.

Thus **`rxy`** ranges between ± 1. (Dividing the output of **`xcorr`** by the square root of the variances of each signal will give similar results.)

Example 2.9 File **`neural_data.mat`** contains two waveforms, **`x`** and **`y`**, that were recorded from two different neurons in the brain with a sampling interval of 0.2 msec.

They are believed to be separated by one or more neuronal junctions that impart a delay to the signal. Plot the original data, determine if they are related and, if so, what the time delay is between them.

Solution: Take the crosscorrelation between the two signals using **`axcor`**. Find the maximum correlation and the time shift at which that maximum occurs.

The former will tell us if they are related and the latter the time delay between the two nerve signals.

Example 2.9 Crosscorrelation between nerve signals

```
%
load neural_data.mat;              % Load data
fs = 1/0.0002;                     % Sampling freq. (1/Ts)
t = (1:length(x))/fs;              % Time vector
subplot(2,1,1);
plot(t,y,'k',t,x,':');             % Plot original data
.....labels.....
[rxy,lags] = axcor(x,y);           % Compute crosscorrelation
subplot (2,1,2);
plot(lags/fs,rxy,'k');             % Plot crosscorrelation
[max_corr, max_shift] = max(rxy);      % Find max correlation
and shift
```

Finding the maximum correlation is straightforward using MATLAB's max operator.

Finding the time at which the maximum value occurs is a bit more complicated as shown next.

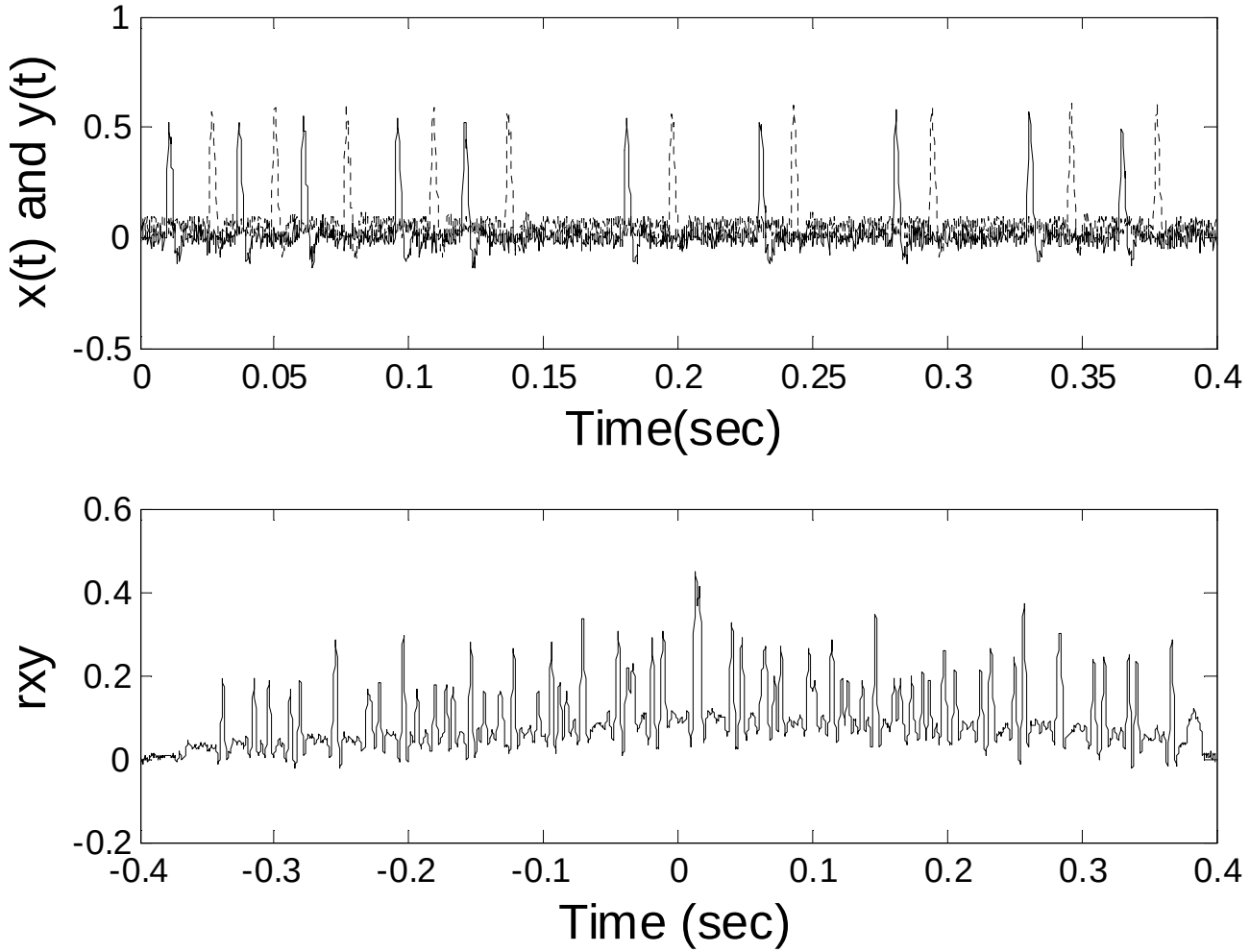
Example 2.9 (cont)
The max operator provides the index of the maximum value, labeled here as **`max_shift`**.

To find the shift corresponding to this index, we need to find the lag value at this shift: **`lags(max_shift)`**.

This lag value then needs to be converted to a corresponding time shift by dividing it by the sampling frequency, **`fs`**.

```
max_shift = lags(max_shift)/fs;      % Convert max shift to sec
plot(max_shift,max_corr,'*k');       % Plot max correlation
 disp([max_corr max_shift])          % Output delay in sec
      ..... labels, title, scaling.....
```

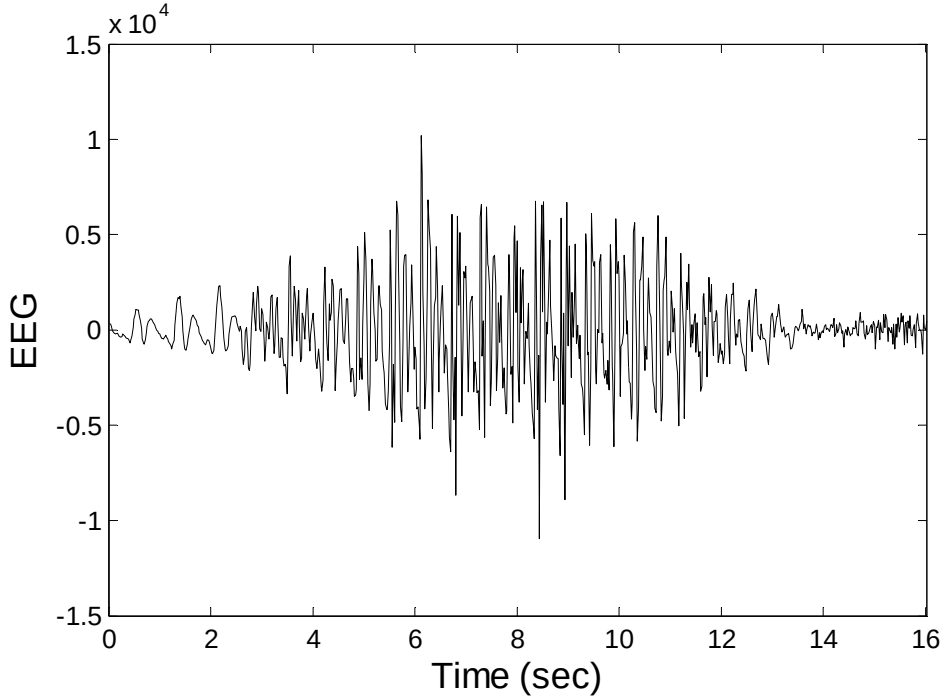
Example 2.9 Results: The time at which the peak occurs is indicated and is found to be 0.013 sec. The maximum correlation is 0.45, suggesting the two signals are related.



Example 2.10 Take the EEG signal shown below and compare it to a sinusoid at a given frequency.

We will do this comparison over a range of frequencies.

Since we want to compare it to a general sinusoid, not only a sine wave, we use crosscorrelation to compare it to shifted sine waves and take the maximum correlation.



EEG signal used in Example 2.10.

Example 2.10 Compare the EEG signal found in file **`eeg_data.mat`** with sinusoids ranging in frequencies between 1.0 and 25 Hz. The sinusoidal frequencies should be in 0.25-Hz increments.

Solution: Load the ECG signal. Use a loop to generate a series of sine waves from 0.25 to 25 Hz. (Cosine waves would work just as well since the crosscorrelation covers all possible phase shifts.)

Crosscorrelate these sine waves with the EEG signal and find the maximum crosscorrelation.

Plot this maximum correlation as a function of the sine wave frequency.

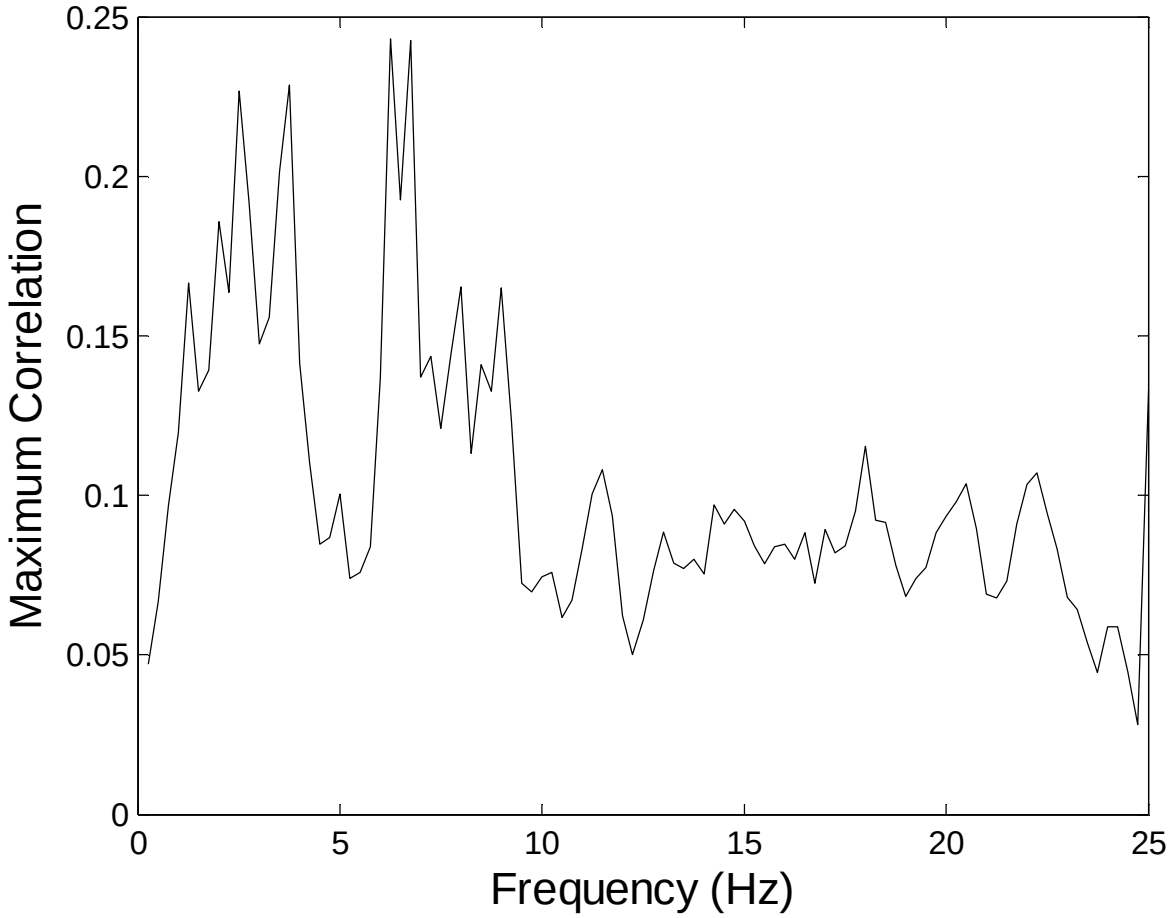
```
% Example 2.10 Camparison of an EEG signal with sinusoids
%
load eeg_data;                  % Get EEG data
fs = 50;                        % Sampling frequency
t = (1:length(eeg))/fs;         % Time vector
for i = 1:25
   f(i) = 0.25*i;               % Frequency range: 0.25-25 Hz
   x = sin(2*pi*f(i)*t);        % Generate sine
   rxy = axcor(eeg,x);          % Perform crosscorrelation
   rmax(i) = max(rxy);          % Store max value
end
plot(f,rmax,'k');               % Plot max values as function of freq.
```

Result:

The result of the multiple crosscorrelations is shown in the next slide.

An interesting structure emerges.

Result Example 2.10.



Some frequencies show much higher correlation between the sinusoid and the EEG.

A particularly strong peak is seen in the region of 7 to 9 Hz, indicating the presence of an oscillatory pattern known as the *alpha wave*.

The Fourier transform is a more efficient method for obtaining the same information as shown in Chapter 3.

Autocorrelation

- It is also possible to correlate a signal with other segments of itself.
- This can be done by performing crosscorrelation on two identical signals, a process called *autocorrelation*.
- Autocorrelation is easy to implement in MATLAB (e.g. **`axcor(x,x)`**), but it is harder to understand what it signifies.
- Autocorrelation describes how long (over what time period) a signal remains correlated with itself.

Autocorrelation (cont)

- As shift increases, the signal is compared with more distant neighbors.
- A signal that remains correlated with itself over long time periods must have been influenced by a system having “memory” (it must remember past values of the signal and use this information to shape the signal’s current values.)
- The longer the memory, the more the signal will remain partially correlated with shifted versions of itself.
- Just as memory tends to fade over time, the autocorrelation function usually goes to zero for large time shifts.

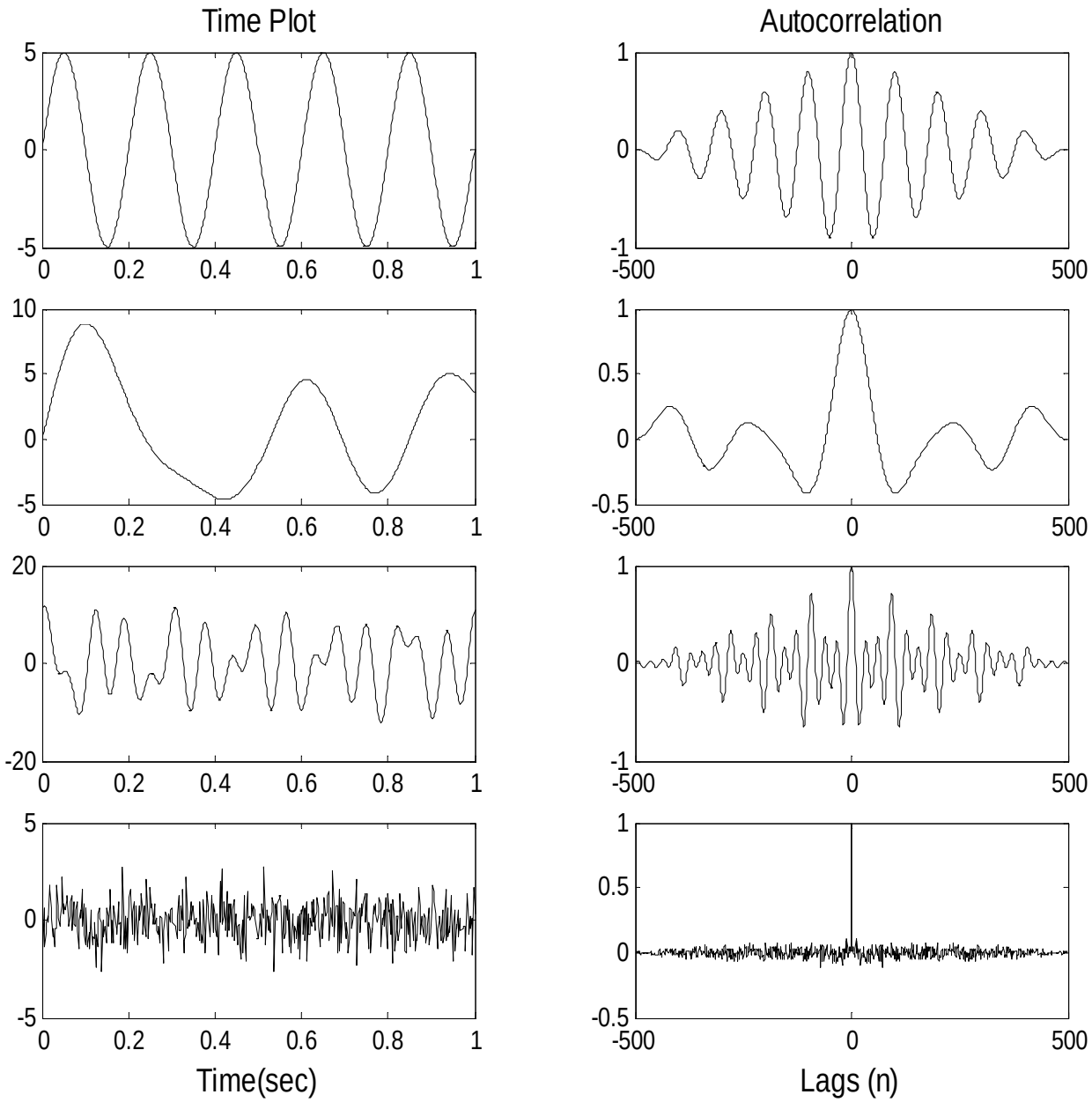
Autocorrelation (cont)

To derive the autocorrelation equation, simply substitute the same variable for *x* and *y* in the crosscorrelation equation:

$$r_{xx}[k] = \frac{1}{N}\sum_{n=1}^{N} x[n]\,x[n+k]$$

where $r_{xx}[k]$ is the autocorrelation function.

The next slide shows four different time functions and the corresponding autocorrelation functions.



Truncated sine wave

Slow time varying signal (Narrowband signal)

Fast time varying signal (Broadband signal)

Gaussian noise

Autocorrelation (cont)

- The autocorrelation of a sine wave is another sinusoid, since the correlation varies sinusoidally with the lag, or phase shift. (Theoretically the autocorrelation should be a pure cosine, but because the correlation routine adds zeros to the end of the signal in order to compute the autocorrelation at all possible time shifts, the cosine function decays as the shift increases.)

- A rapidly varying signal *decorrelates* quickly: the correlation of neighbors falls off rapidly for even small shifts of the signal with respect to itself. (A rapidly varying signal has a poor memory of its past values and is probably the product of a process with a short memory.)

Autocorrelation (cont)

- For slowly varying signals, the correlation falls slowly. Nonetheless, as for all signals, there is some time shift for which the signal becomes completely decorrelated with itself.

- For a Gaussian noise, the correlation falls to zero instantly for all positive and negative lags. This indicates that each signal sample is has no correlation with neighboring (or any other) samples.

Autocorrelation (cont)

- Since shifting the waveform with respect to itself produces the same results no matter which way the waveform is shifted, the autocorrelation function will be symmetrical about lag zero.

- Mathematically, the autocorrelation function is an even function: $r_{xx}(-\tau) = r_{xx}(\tau)$

- The maximum value of r_{xx} occurs at zero lag, where the waveform is correlated with itself.

- If the autocorrelation is normalized by the variance (common), the value at zero lag is 1.0.

Autocorrelation: MATLAB Implementation

Autocorrelation in MATLAB is just a special case of crosscorrelation.

When only a single input vector is provided, both **xcorr** and **axcor** assume the autocorrelation function is desired.

```
[rxx,lags] = axcor(x);                    % Autocorrelation

[rxx,lags] = xcorr(x,maxlags,'coeff');    % Autocorrelation
```

The **`'coeff'`** option is used with **`xcorr`** routine to indicate that the output should be normalized to 1.0 at zero shift (i.e., lags = 0). The routine **`axcor`** does this automatically when only a single input is given.

Autocovariance and Crosscovariance

These two operations are closely related to autocorrelation and crosscorrelation except that the signal means have been removed:

$$c_{xx}[k] = \frac{1}{N}\sum_{n=1}^{N}\left(x[n]-\bar{x}\right)\left(x[n+k]-\bar{x}\right)$$ Autocovariance

$$c_{xy}[k] = \frac{1}{N}\sum_{n=1}^{N}\left(y[n]-\bar{y}\right)\left(x[n+k]-\bar{x}\right)$$ Crosscovariance

Autocovariance and cross-covariance can be thought of as measuring the memory or self-similarity of the deviation of a signal(s) about their mean level(s).

Autocovariance and Crosscovariance MATLAB Implementation

Auto and cross-covariance can be implemented using **`axcor`** or **`xcorr`** with the data means subtracted out; e.g.:

```
[cxy,lags] = axcor(x-mean(x), y-mean(x));  % Cross-covariance
```

- Many physiological processes are repetitive, such as respiration and heart rate, yet vary somewhat cycle-to-cycle.
- Autocorrelation and autocovariance can be used to explore this variation.

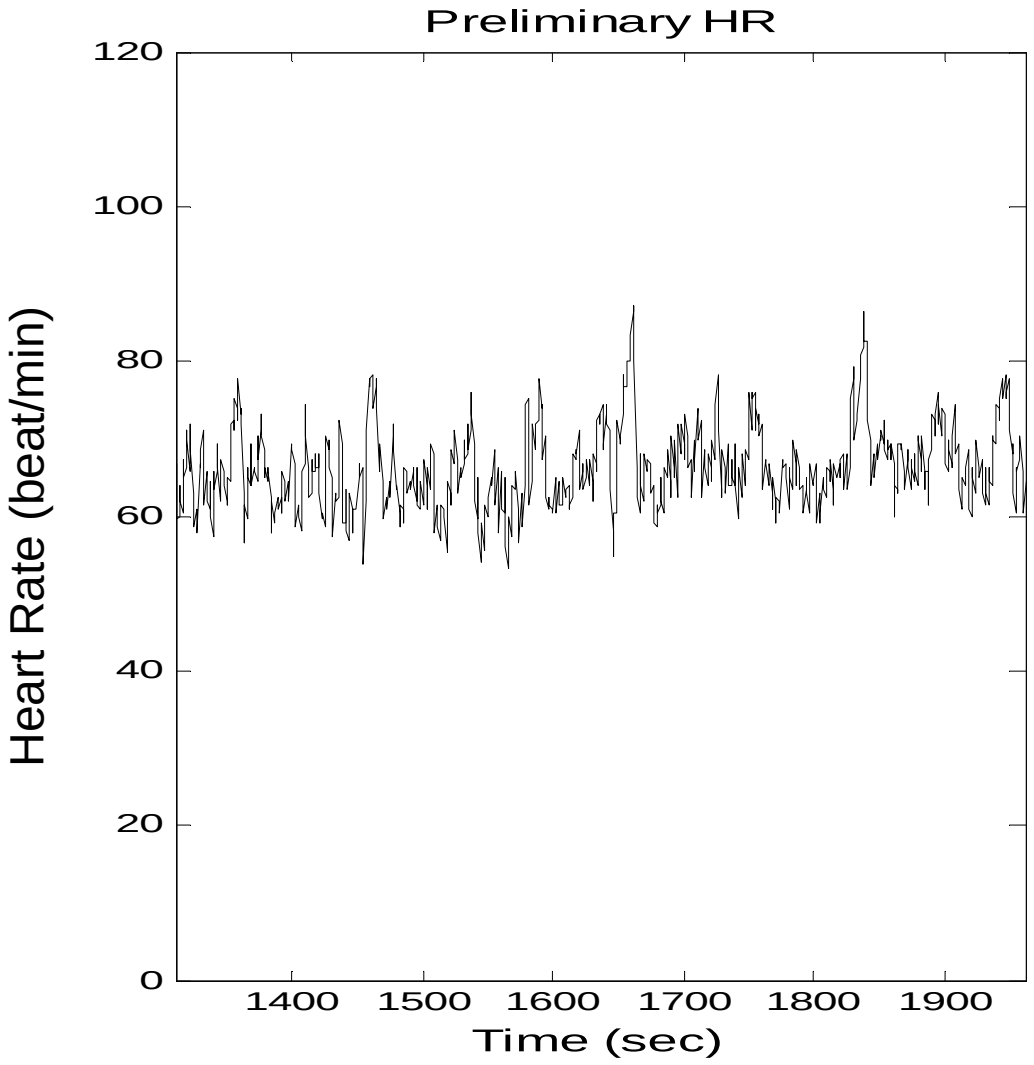
Autocovariance Application

Considerable interest revolves around the heart rate and its beat-to-beat variations. (Remember that autocovariance will subtract the mean value of the heart rate from the data and analyze only the variation.)

Example 2.11 uses autocovariance to examine the variation in successive beats.

In this example, we use autocovariance, not autocorrelation, since we are interested in the correlation of heart rate variability, not the correlation of heart rates per se.

Example 2.11 Determine if there is any correlation in the variation between the timing of successive heartbeats under normal resting conditions.



Normal heart rate in beats/min as a function of time.

The variability in heart rate is apparent.

Example 2.11 Solution: Load the heart rate data taken during normal conditions.

The file **`Hr_pre.mat`** contains the variable **`hr_pre`**, the instantaneous heart rate. However, the heart rate is determined each time a heartbeat occurs, so it is not evenly time-sampled. A second variable **`t_pre`** contains the time at which each beat is sampled.

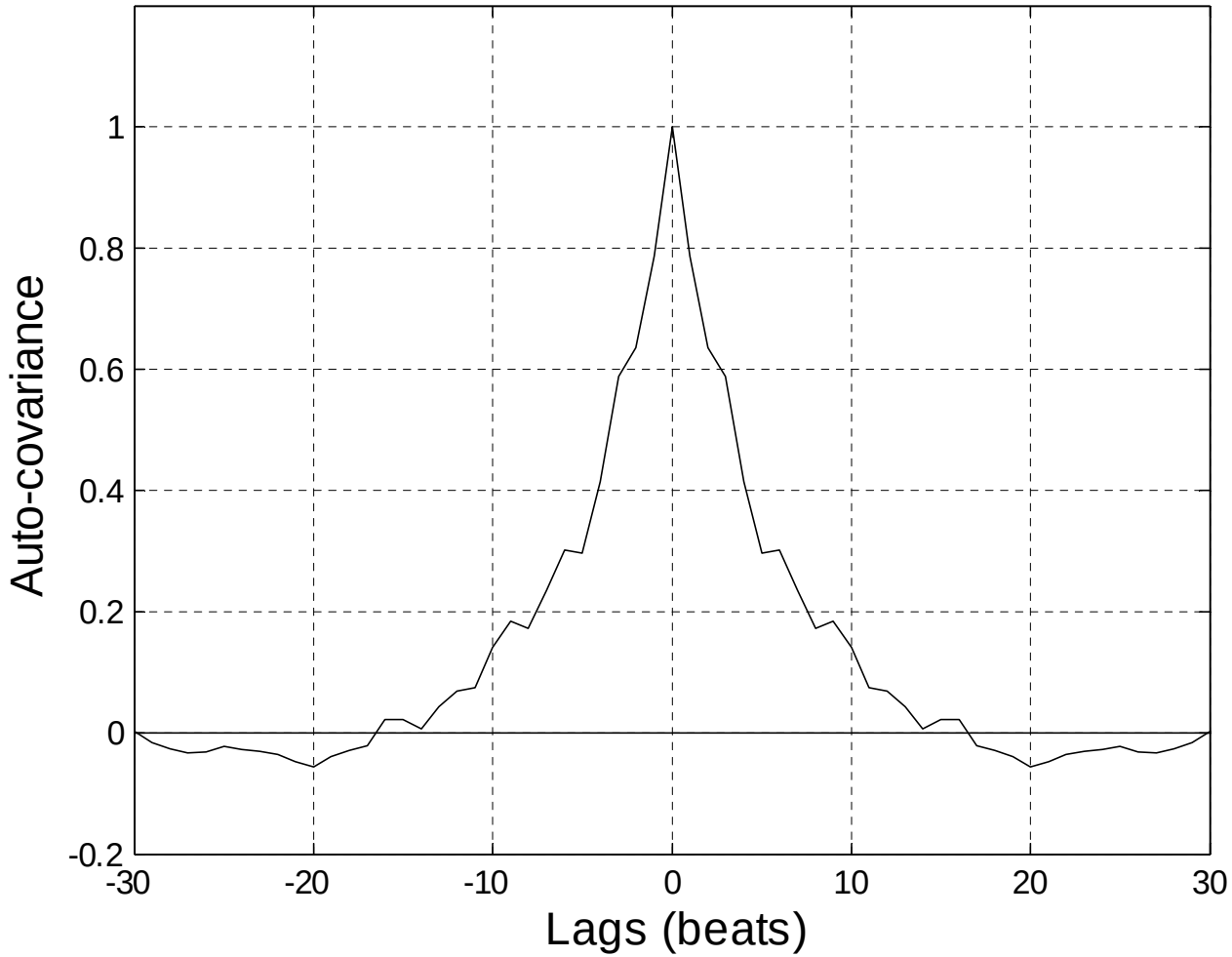
For this problem, we will determine the autocovariance as a function of heart beat and we will not need the time variable.

We can determine the autocovariance function using **`axcor`** by first subtracting the mean heart rate.

```
% Example 2.11 Use of autocovariance to determine the
%   correlation of heart rate variation between heart beats
%
load Hr_pre;                          % Load normal HR data
% Calculate auto-covariance
[cov_pre,lags_pre] = axcor(hr_pre - mean(hr_pre));
%
plot(lags_pre,cov_pre,'k'); hold on;          % Plot results
plot([lags_pre(1) lags_pre(end)], [0 0],'k'); % Plot a zero line
axis([-30 30 -0.2 1.2]);                      % Limit x-axis
```

We plot the resulting autocovariance function and limit the *x* axis to ± 30 successive beats to better evaluate the decrease in covariance with successive beats.

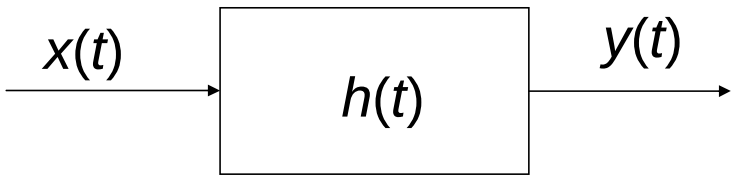
Example 2.11 Results: The results below show that there is correlation between adjacent heartbeats all the way out to 10 beats.



Convolution and the Impulse Response

Convolution can be thought of as a linear process acting on a signal, $x(t)$, to produce a modified signal, $y(t)$.

We will show it is also an example of sliding correlation were one function is projected as sliding on another.



$y(t) = \text{conv}\ (x(t)\ h(t))$

The operation of convolution is used in linear systems theory to calculate the output of an LTI system to any input signal.

The Impulse Response

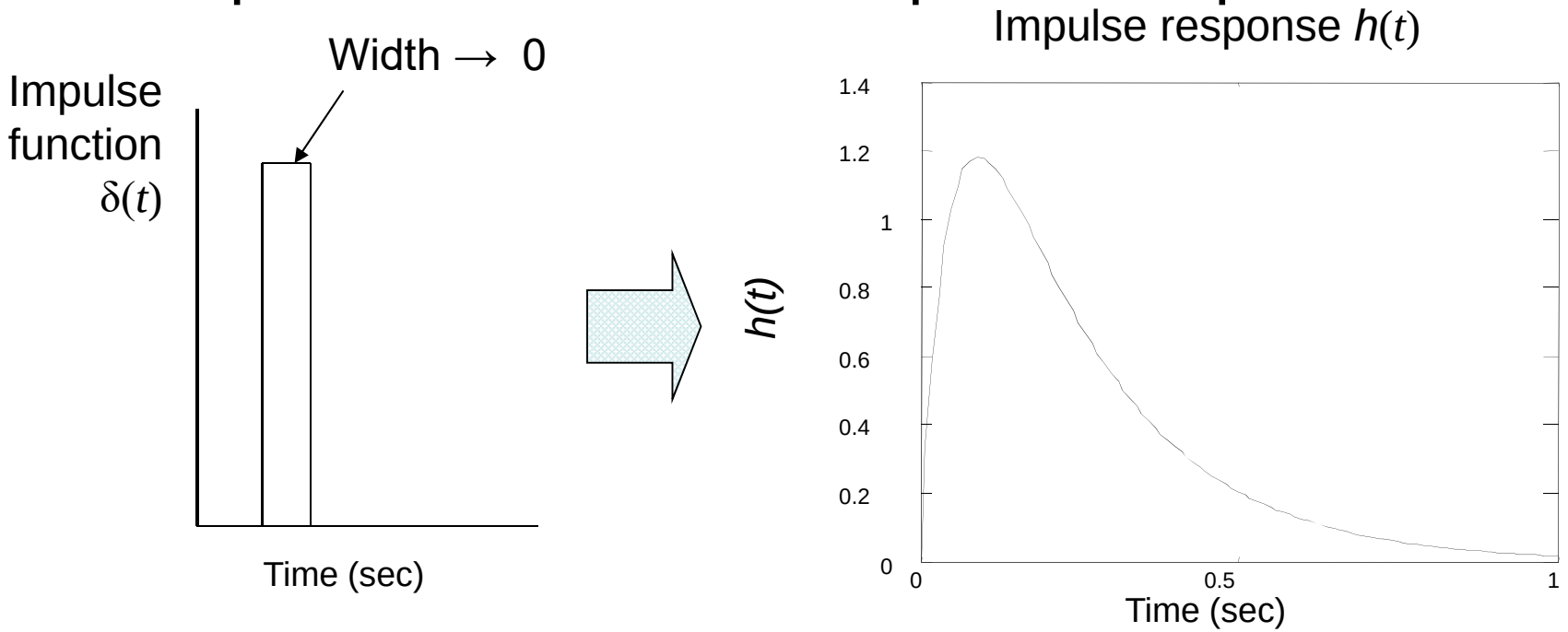
- The function, $h(t)$, is known as the *impulse response*. As the name implies, it is a system's response to an impulse input.

- An impulse input (also termed a *delta* function and commonly denoted $\delta(t)$) is a very short pulse with an area of 1.0 (in whatever units you are using).

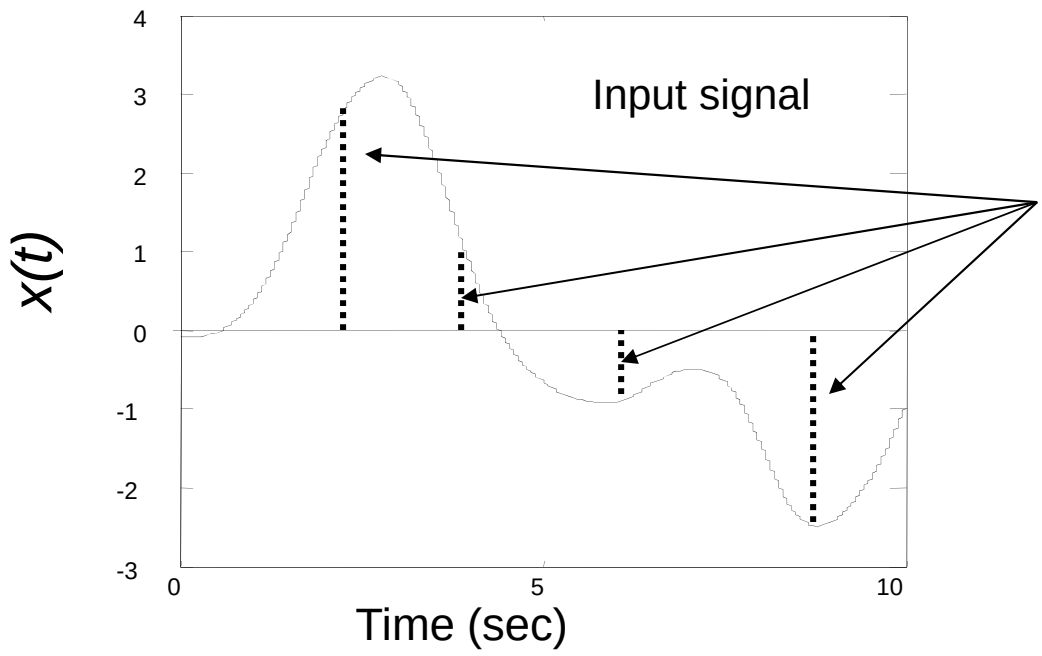
- In theory it is infinitely short but also of infinite amplitude, compensating in such a manner as to keep the area 1.0. (An infinitely short pulse is impossible, so in practice the pulse is short enough so further shortening does not change the basic shape of the response.)

The Impulse Function and Impulse Response

A



B



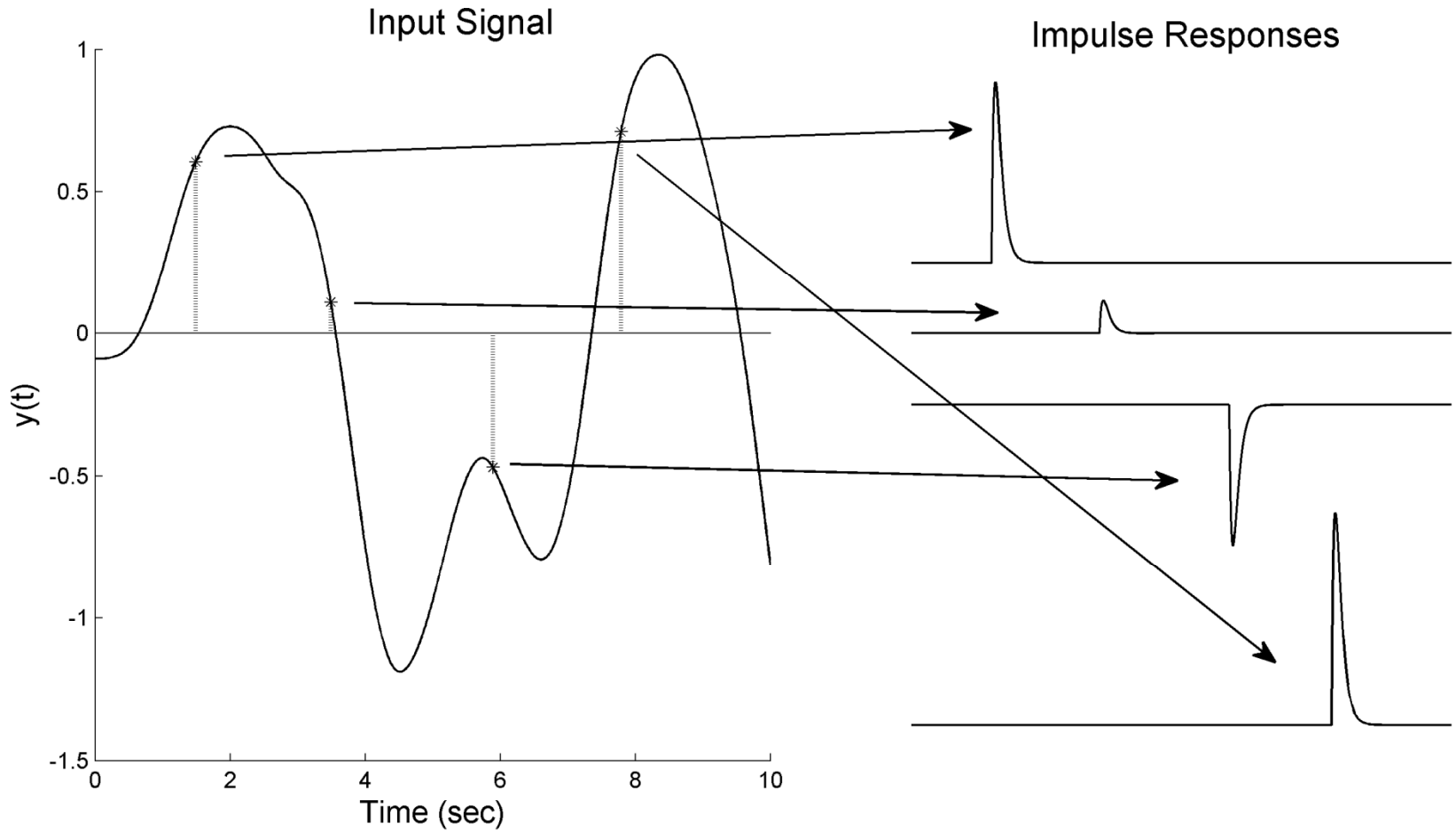
Any signal $x(t)$ can be considered to be made up of an infinite number of impulse functions.

Impulse Function: Application

- If you know the system's response to an impulse, you can determine its response to any input simply by dividing the input into a sequence of impulses.
- Each time slice will generate its own little impulse response.
- The amplitude and position of this impulse response is determined by the amplitude and position of the associate input signal segment.
- If superposition and time invariance hold, then the output can be determined by summing (or integrating) the impulse responses from all input signal segments.

Each signal segment contributes its own little impulse response to the output, scaled and shifted appropriately.

Since LTI systems are time-invariant, the time shifting does not alter the impulse response.



Convolution Sum

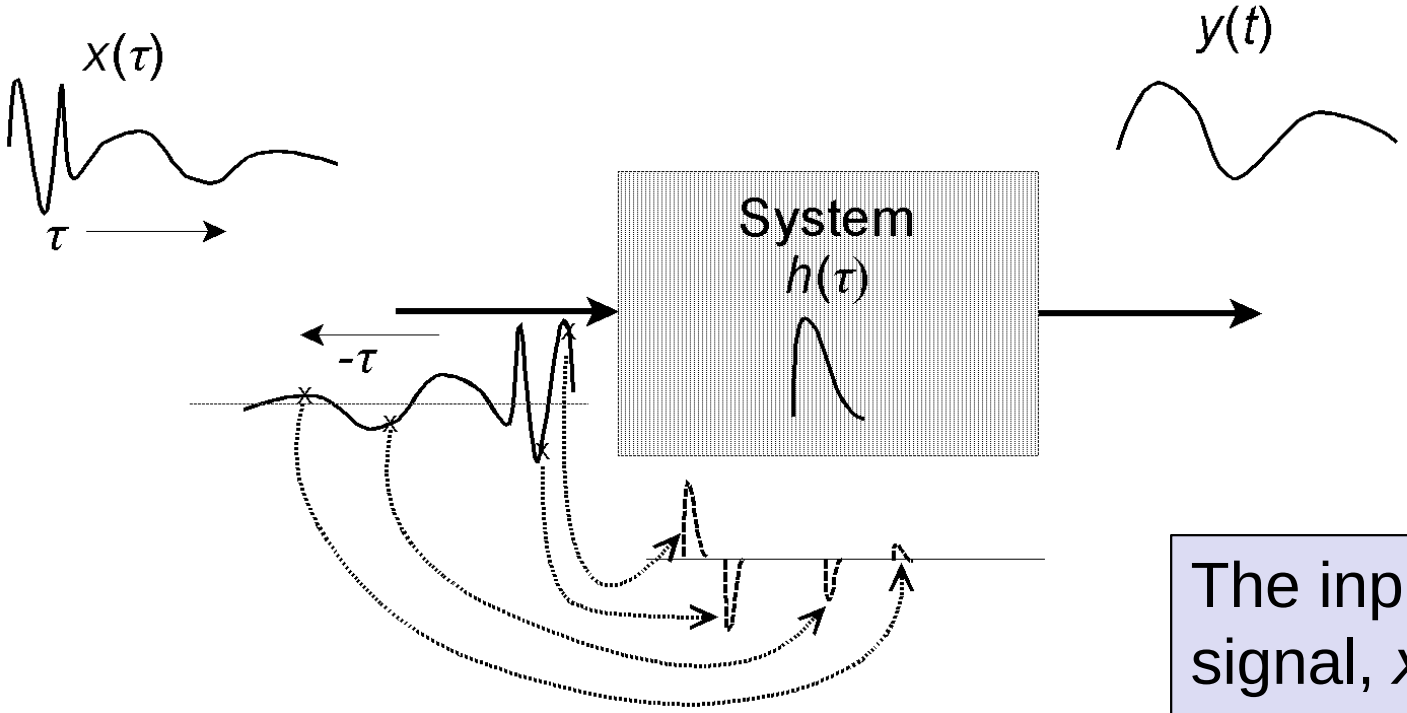
- Convolution can be used to perform a summation of the individual impulse responses.
- Convolving the input signal with the impulse response results in the output signal from the process that generated the impulse response.
- The convolution sum is similar to the crosscorrelation except that one of the functions is reversed:

$$y[n] = \sum_{k=0}^{K-1} h[k]\, x[n-k]$$

- Convolution is based on superposition and therefore only applies to LTI systems.

Reversing the Signal

When implementing convolution, we reverse the input signal because the low-time side of the signal (the left side) is first to enter a system.



The input signal, $x[k]$, becomes $x[-k]$

The Convolution Sum (cont)

Despite this similarity between the convolution sum and crosscorrelation, the intent of the two equations is completely different:

1. Crosscorrelation compares two functions.
2. Convolution provides the output of an LTI system given its impulse response.

It does not matter which function is shifted, the convolution sum can also be written as:

$$y[n] = \sum_{k=0}^{K-1} x[k]h[n-k] \equiv x[k] * h[k]$$

The '*****' symbol is sometimes used as shorthand for convolution which is confusing as it usually represents multiplication.

The Convolution Integral

In the continuous domain, the summation becomes integration leading to the *convolution integral*:

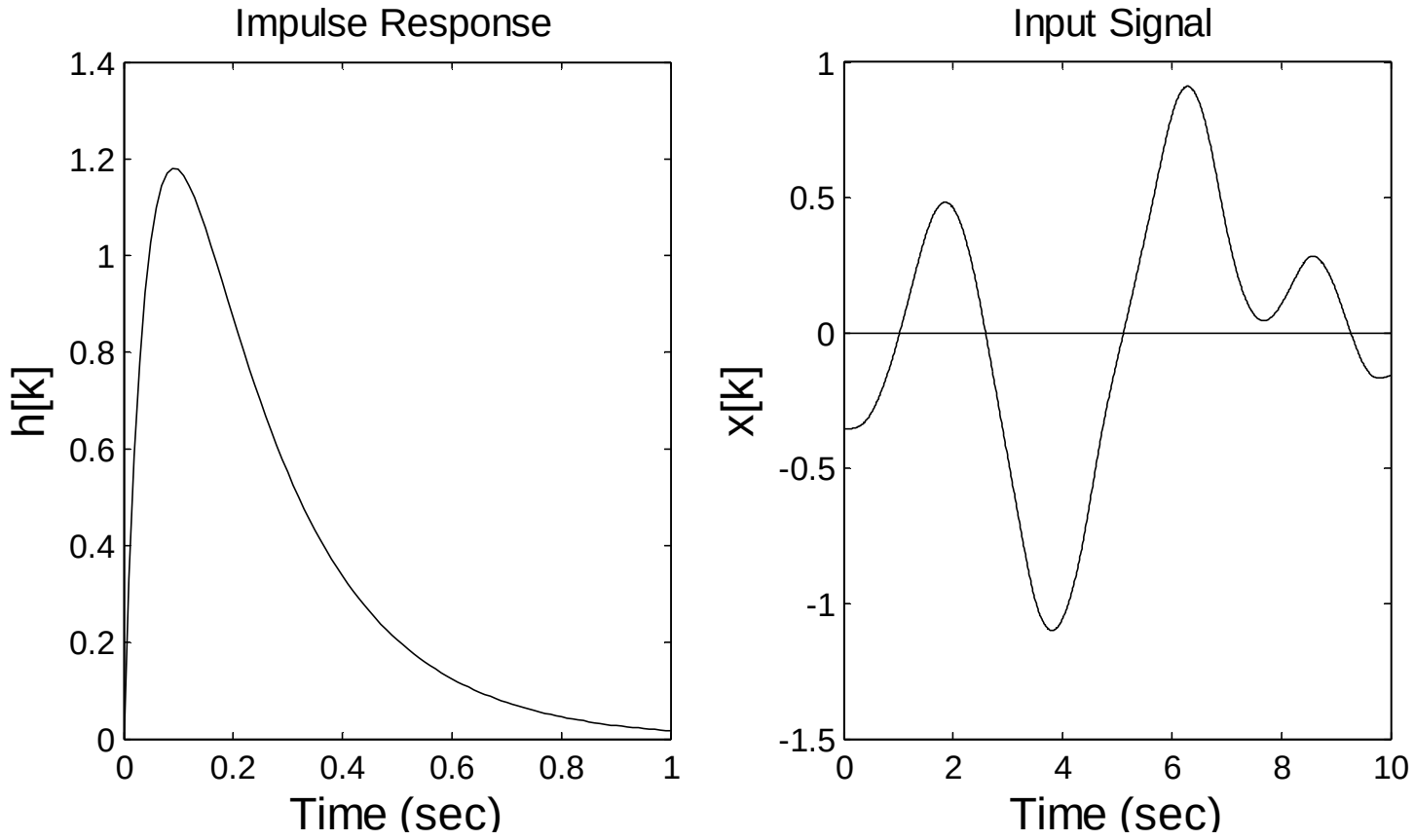
$$y(t) = \int_{-\infty}^{\infty} h(t-\tau)x(\tau)d\tau = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau$$

The next few slides provide an example of convolution using several different sampling intervals to show the effect.

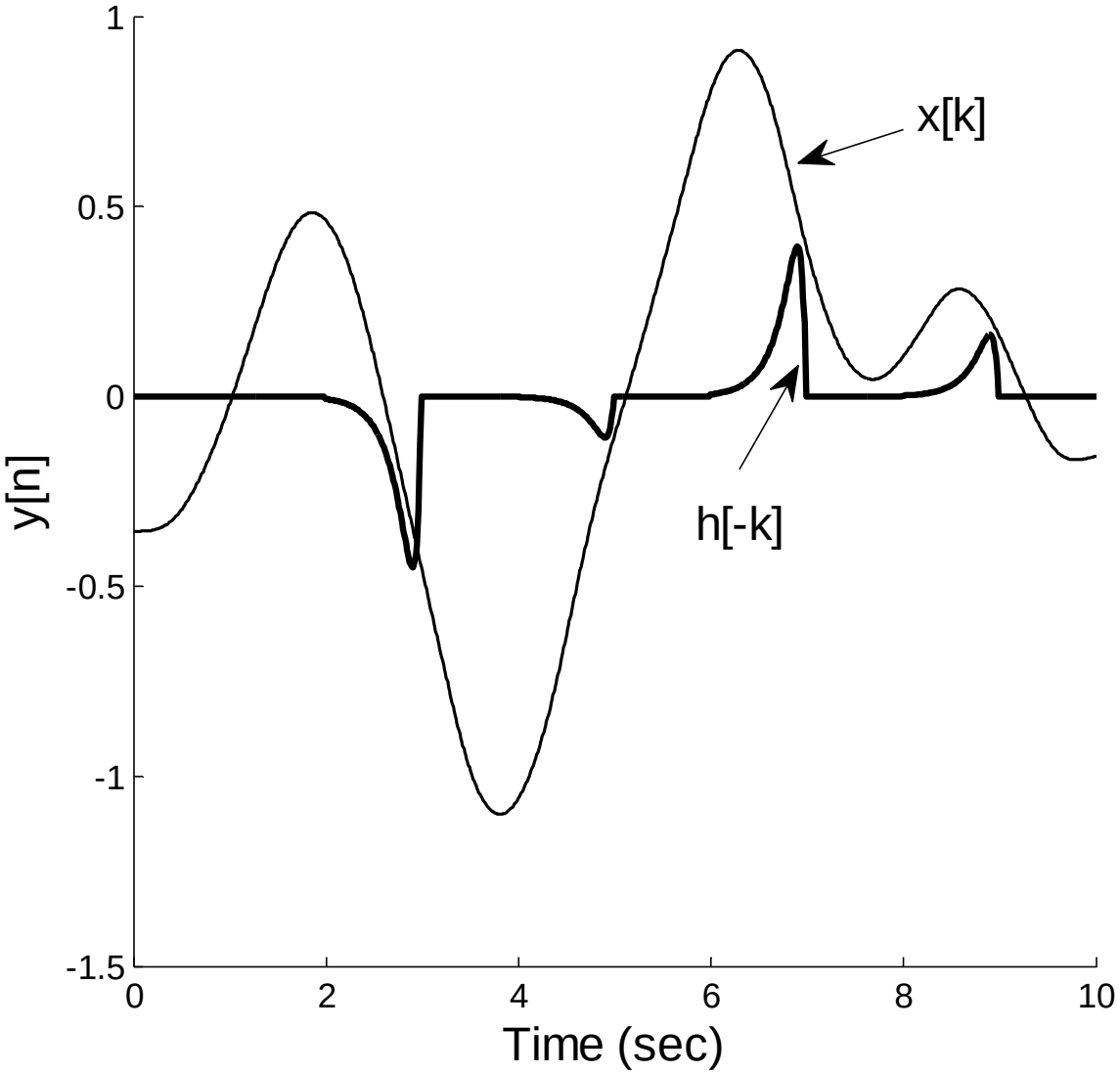
Convolution Illustration

Assume the signal on the right is the input to a system having the impulse response shown on the left.

The impulse or signal will be reversed as described previously. (Here we will reverse the impulse response.)



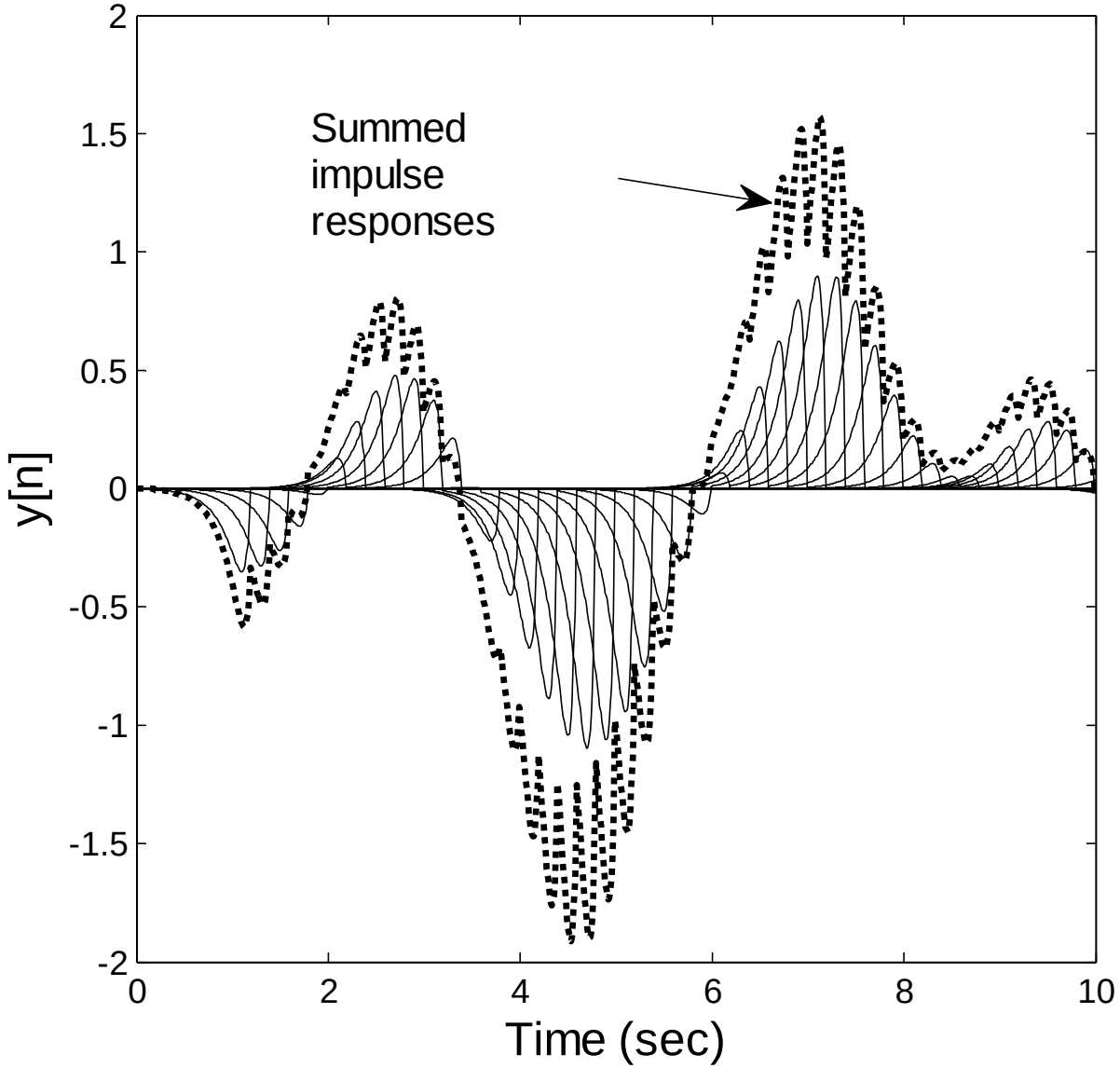
Convolution Illustration (cont)



Four (reversed) impulse responses: $h[-k]$;

delayed and scaled by the input signal, $x[k]$.

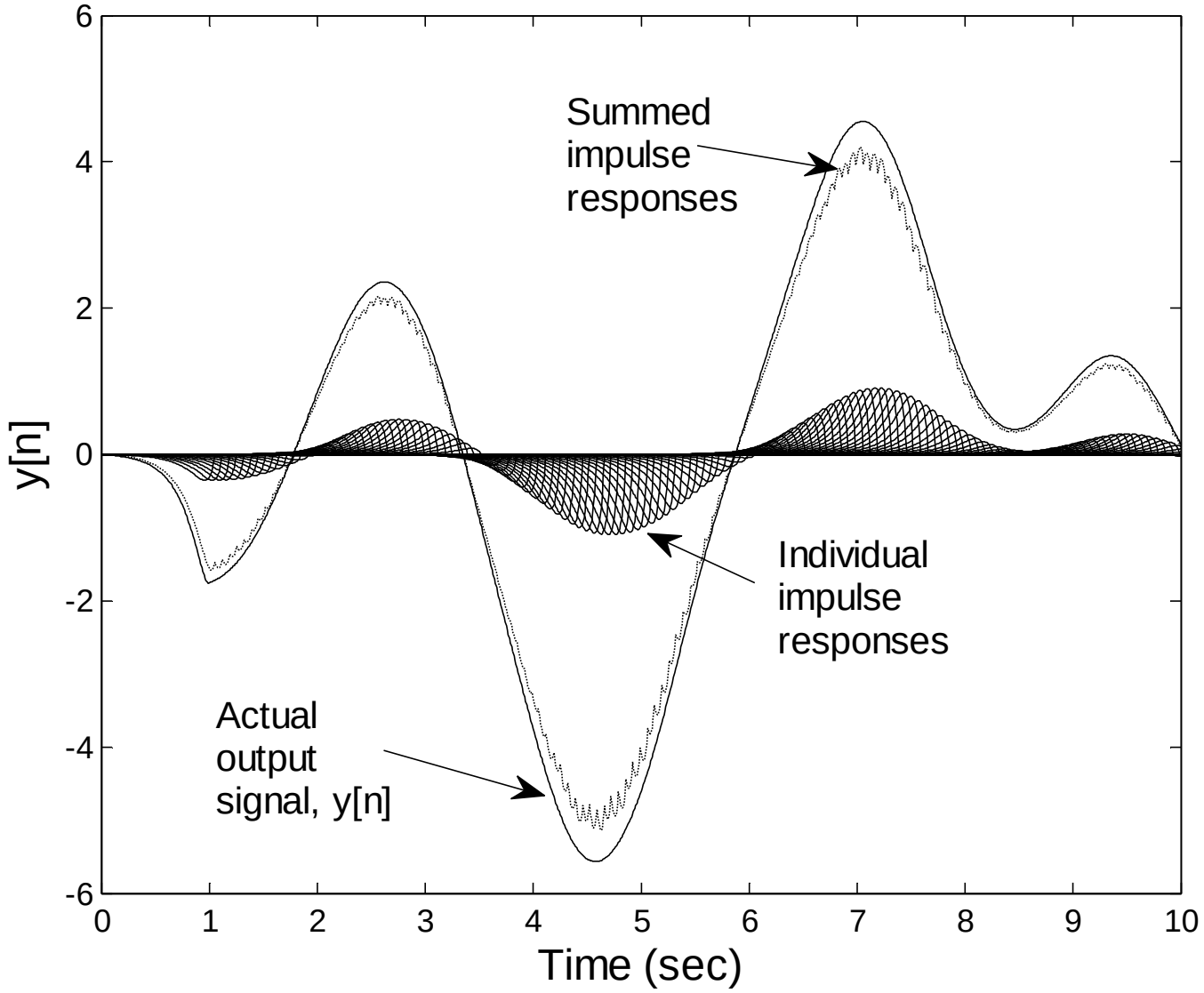
Convolution Illustration (cont)



Fifty reversed, delayed, and scaled impulse responses (solid line).

The summation (dashed line) begins to look like the output response.

Convolution Illustration (cont)



150 reversed, delayed, and scaled impulse responses.

The summation, dashed line, closely approximate the actual output signal, $y[n]$.

Convolution: MATLAB Implementation

Convolution is easy to implement in MATLAB:

```
y = conv(x,h);     %Convolution sum
```

There are some options to control how many data points are generated by the convolution, but the default produces **`length(h)+length(x)-1`** data points.

Usually only the first **`length(x)`** points are used and the additional points are ignored. (There is an alternative command for implementing convolution presented in Chapter 4 that does not generate additional points.)

Example 2.12 Construct an array containing the impulse response of a first-order process.

The impulse response of a first-order process is given by the equation: $h(t) = e^{-t/\tau}$ (scaled for unit amplitude). Assume a sampling frequency of 500 Hz and a time constant, τ, of 1 sec.

Use convolution to find the response of this system to a unit step input. (A unit step input jumps from 0.0 to 1.0 at $t = 0$.)

Plot both the impulse response and the output to the step input signal.

Repeat this analysis for an impulse response with a time constant of .2 sec (i.e., $\tau = 0.2$ sec.).

Example 2.12 Solution: The most difficult part of this problem is constructing the first-order impulse response, the discrete function $h[k]$.

In order to adequately represent the impulse response, a decaying exponential, we make the function at least 5 time constants long, the equivalent of 5 sec.

Thus we need an array that is $N = T_T/T_s = T_T\,(f_s) = 5(500) = 2500$ points.

The step function is just an array of ones 2500 points long.

Convolving this input with the impulse response and plotting is straightforward.

```
% Example 2.12  Convolution of a first-order system with a
%   step
%
fs = 500;                   % Sample frequency
N = 2500;                   % Construct 5 seconds worth of data
t = (0:N-1)/fs;             % Time vector 0 to 5
tau = 1;                    % Time constant
h = exp(-t./tau);           % Construct impulse response
x = ones(1,N);              % Construct step stimulus
y = conv(x,h);              % Get output (convolution)
subplot(1,2,1);
plot(t,h);                  % Plot impulse response
      .........label and title.......
subplot(1,2,2);
plot(t,y(1:N));             % Plot the step response
      .........label and title.......
```

Example 2.12 Results: The impulse and step responses of a first-order system is shown below.

Note that MATLAB's **conv** function generates more additional samples (in fact, 4999 points) so only the first 2500 points are used in plotting the response.