Basics of signal processing (3)

Prof. Febo CINCOTTI, febo.cincotti@uniroma1.it

Dept. of Computer, Control and Management Engineering

(DIAG, Via Ariosto)

Material for this section of the course

- Matlab notebooks available here:
  - https://drive.matlab.com/sharing/d5ad1819-5e50-442a-81fc-6017505d91f3
    - NEng_1920_02_Stats.mlx (cont'd)
    - NEng_1920_03_Spectr.mlx

- Not a textbook, but readings for those who want to have some context:
  - Steven W. Smith
    The Scientist and Engineer's Guide to Digital Signal Processing
    https://www.dspguide.com/pdfbook.htm

Neuroengineering - Statistics, Probability and Noise

Signal can be entirely deterministic, or they can be known only by means of their statistical properties. We will introduce a number measurements to characterize both types of signals.

Probability theory deals with the mathematical modeling of random variables (numbers) and processes (signals). Statistics deals with the description of empirical observations, and with the estimation of the (usually unknown) parameters of the mathematical models of the variables and processes. (stochastic signals)

Fundamental to several analysis algorithms, the Central Limit Theorem states the relevance of Normal (Gaussian) distributions in empirical sciences

See also https://www.dspguide.com/pdfbook.htm, Chapter 2

Basic measures for signal characterization

Mean:

$$\bar{X} = \frac{1}{N}\sum_{i=0}^{N-1} x_i \to \mu_X$$

Average Rectified Value (ARV):

$$ARV_X = \frac{1}{N}\sum_{i=0}^{N-1} |x_i|$$

Root Mean Square (RMS):

$$RMS_X = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} x_i^2}$$

Average deviation:

$$AD_X = \frac{1}{N}\sum_{i=0}^{N-1} |x_i - \bar{X}|$$

Variance σ^2 and Standard deviation (SD, σ):

$$s_X^2 = \frac{1}{N-1}\sum_{i=0}^{N-1} (x_i - \bar{X})^2 \to \sigma_X^2$$

$$s_X = \sqrt{\sigma_X^2} = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1} (x_i - \bar{X})^2} \stackrel{\sim}{\to} \sigma_X$$

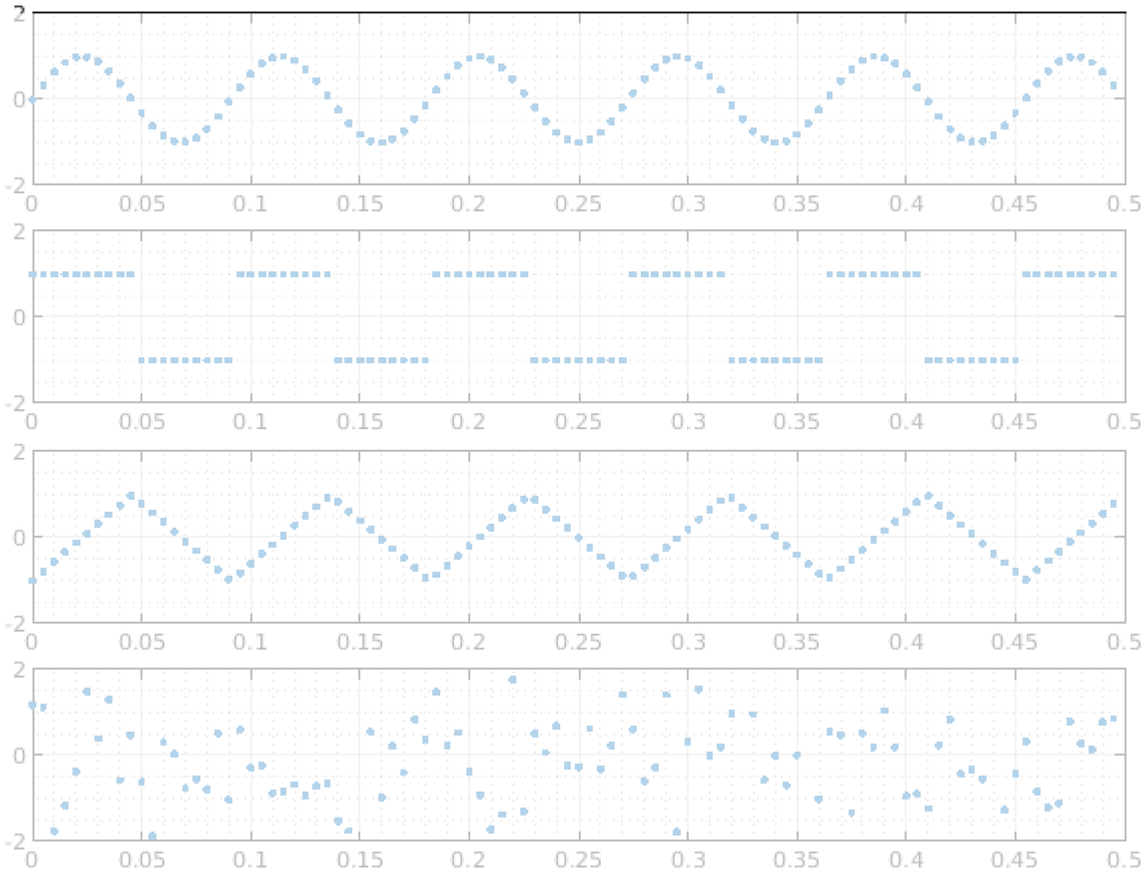
Note that when a signal has zero mean, $RMS_X \cong \sigma_X^2$, and $ARV_X \cong AD_X$

We introduce here four signals that we will analize in the following:

1. sinewave
2. square wave
3. triangular wave
4. gaussian white noise

The first three are deterministic waveforms, oscillating at fundamental frequency F_0. The last one is a stochastic signal, characterized by having incorrelated samples (whiteness, i.e. no statistical prediction can be made on the value of a specific sample by knowing the value of the others) and gaussian distribution of the sample values (see below).

```
sinewave = @(t, f0) sin(2*pi*f0*t);             % sinewave
squarewave = @(t, f0) square(2*pi*f0*t);        % square wave
triangwave = @(t, f0) sawtooth(2*pi*f0*t,1/2); % triangular wave
gwnoise = @(t, dummy) randn(size(t));           % gaussian white noise

F_0 = 11;  %Hz, fundamental frequency of waveforms
NUM_POINTS = 100;
t = 0.005 *(0:NUM_POINTS-1)';  % s, time axis

wavenames = ["sine" "square" "triangle" "noise"];
waves = {sinewave(t,F_0), squarewave(t,F_0), triangwave(t,F_0), gwnoise(t,NaN)};

linetype = ".";
figure(1)
clf
tiledlayout(length(waves),1, "TileSpacing","none", "Padding","none");
for ii = 1:length(waves)
    nexttile
    plot(t, waves{ii}, linetype)
    ylim([-2 2])
    grid on
    grid minor
end
```



Intermezzo -- Audio representation

```
F_0_audio = 400;  % Hz
F_samp_audio = 8000; % Samples/second, S/s
t_audio = (0:F_samp_audio-1)' / F_samp_audio;  % s
fun = gwnoise;
audio_wave = fun(t_audio,F_0_audio);
% audiowrite("wave.wav", audio_wave, F_samp_audio);
filename = regexprep(string(func2str(fun)),"@?\([^\)]*\)","_");
audiowrite(filename+".wav", audio_wave, F_samp_audio);
```

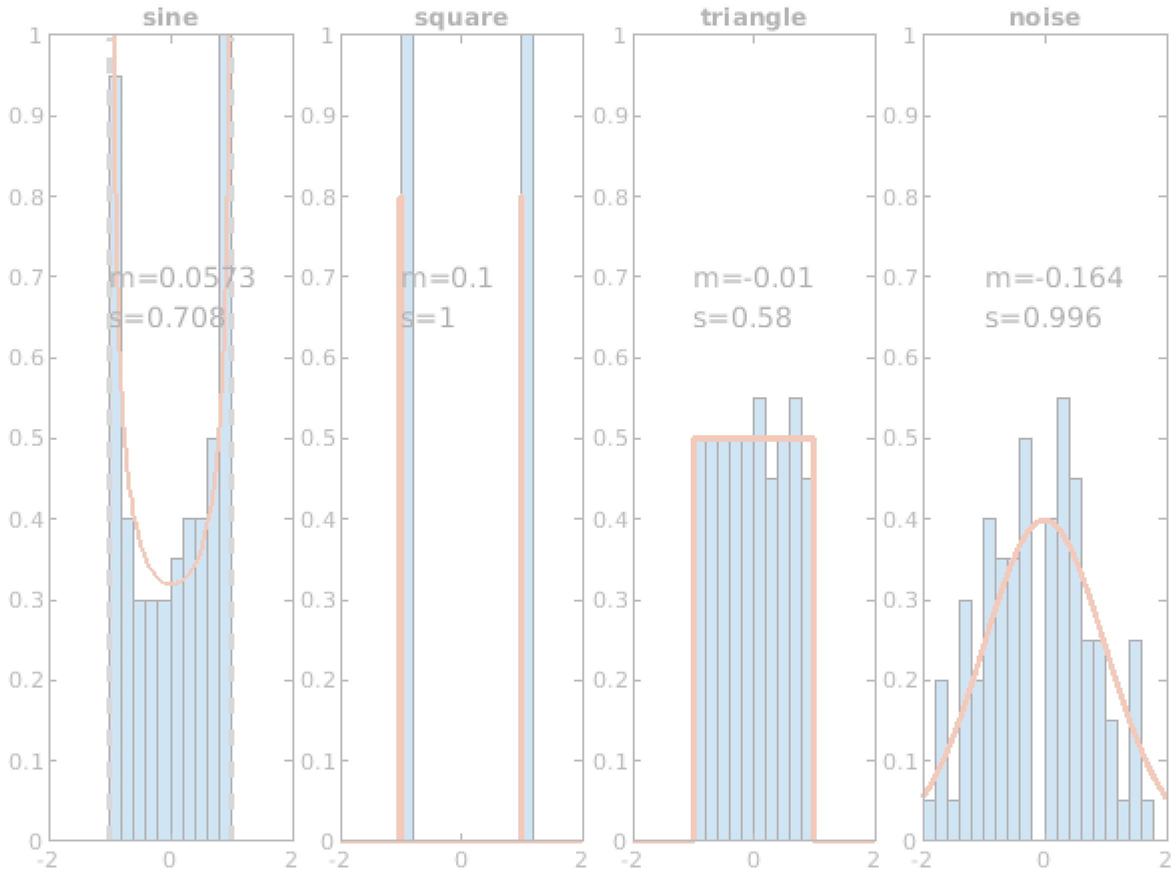
```
Warning: Data clipped when writing file.
```

We want to characterize here how the samples are distributed on the vertical axis: the central tendency (mean), the dispersion (standard deviation), the shape of the distribution (probability density function, pdf).

Note that we have perfect knowledge of the mathematical model we used to generate the determinist and stochastic signals. Nevertheless, even for deterministic signals there are sources of non-deterministic outcome (e.g. finite number and uncontrolled position of time samples) which make the empiric and ideal results to overlap only in part.

As for the stochastic signal, each time we repeat this simulation, we obtain different values of mean, standard deviation, and histogram. We can only assess the compatibility of empirical observations with the mathematical model in a statistical sense.

```
INFTY = .8; EPS = .01;
normal_pdf = @(x,m,s) (2*pi*s^2)^(-1/2) .* exp(-(x-m).^2./(2*s^2));
% these are the pdf's of the mathematical model we used to generate the
% waveforms.
pdfs = {
    @(a) 1./(pi*(1-a.^2).^(1/2))
    @(a) INFTY * double(abs(abs(a)-1) < EPS)  % should be a Dirac's delta
    @(a) 1/2 * double(abs(a)<=1)  % uniform
    @(a) normal_pdf(a,0,1)        % normal
    };

figure(2)
clf
tiledlayout(1, length(waves), "TileSpacing","none", "Padding","none");
for ii = 1:length(waves)
    nexttile
    histogram(waves{ii}, linspace(-2,2,21), ...
        "Normalization","pdf" ...
        )
    hold on
    fplot(pdfs{ii}, [-2 2], "LineWidth",2)
    hold off
    xlim([-2 2])
    ylim([0 1])
    title(wavenames{ii})
    m_x = mean(waves{ii});
    s_x = std(waves{ii});
    text(-1, .70, "m="+sprintf("%.3g",m_x))
    text(-1, .65, "s="+sprintf("%.3g",s_x))
end
```



True standard deviation of sinewave

```
disp("1/sqrt(2)  = "+1/sqrt(2))
```

```
1/sqrt(2)  = 0.70711
```

True standard deviation of triangle wave

```
disp("2/sqrt(12) = "+2/sqrt(12))
```

```
2/sqrt(12) = 0.57735
```

---- 29/04/2020 ----

^^^^^^^^^^^^^^^^^^^^^^

Central limit theorem (CLT)

When *N* independent and identically distributed random variables X_i are averaged, the resulting variable $Z = \frac{1}{N}\sum_{i=1}^{N} X_i$ tends toward a **normal distribution** even if the original variables themselves are not normally distributed. (*Normal distributions are sometimes called Gaussian distributions.*)

The mean of *Z* equals the mean of X_i, the variance decreases by a factor N, the standard deviation by a factor $\sqrt{N}$:

$$\mu_Z = \mu_X$$

$$\sigma_Z^2 = \frac{1}{N}\sigma_X^2$$

$$\sigma_Z = \frac{1}{\sqrt{N}}\sigma_X$$

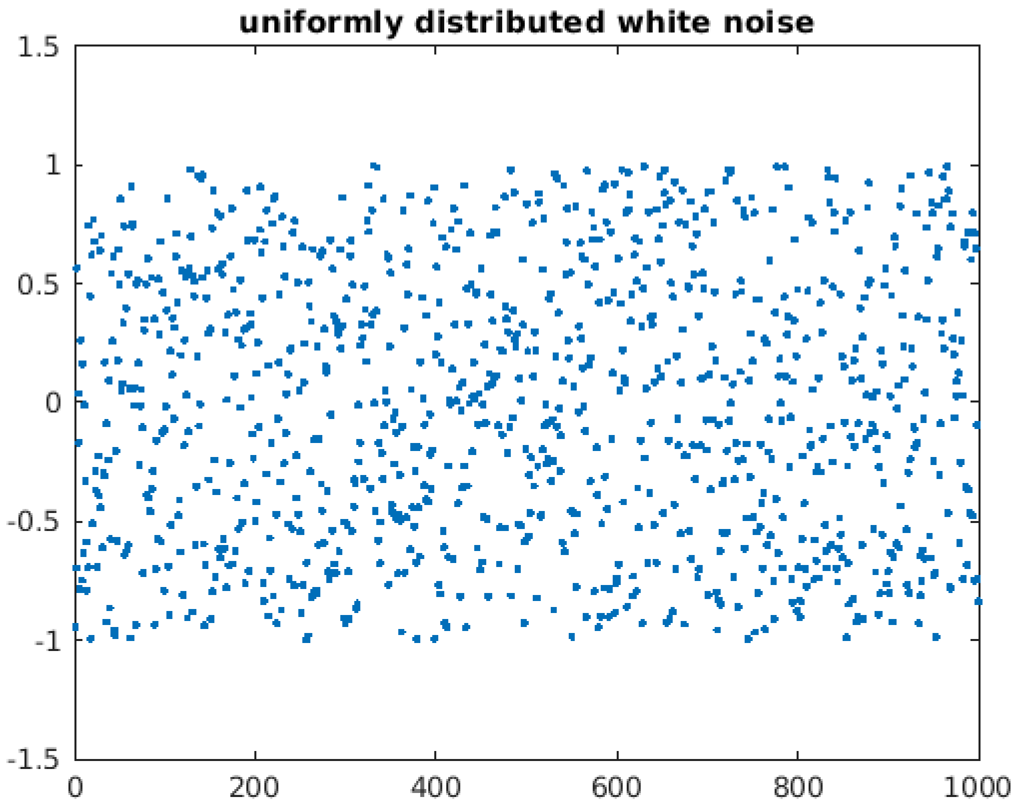
To visualize the consequences of the CLT, we first show the amplitude distribution of a (uniformly distributed) white noise, and its standard deviation. By design, the expected value of this noise's mean and standard deviation are respectively:

$\mu_X = 0$ (We will later manipulate the expected mean, by adding a small "useful" signal to this noise.)

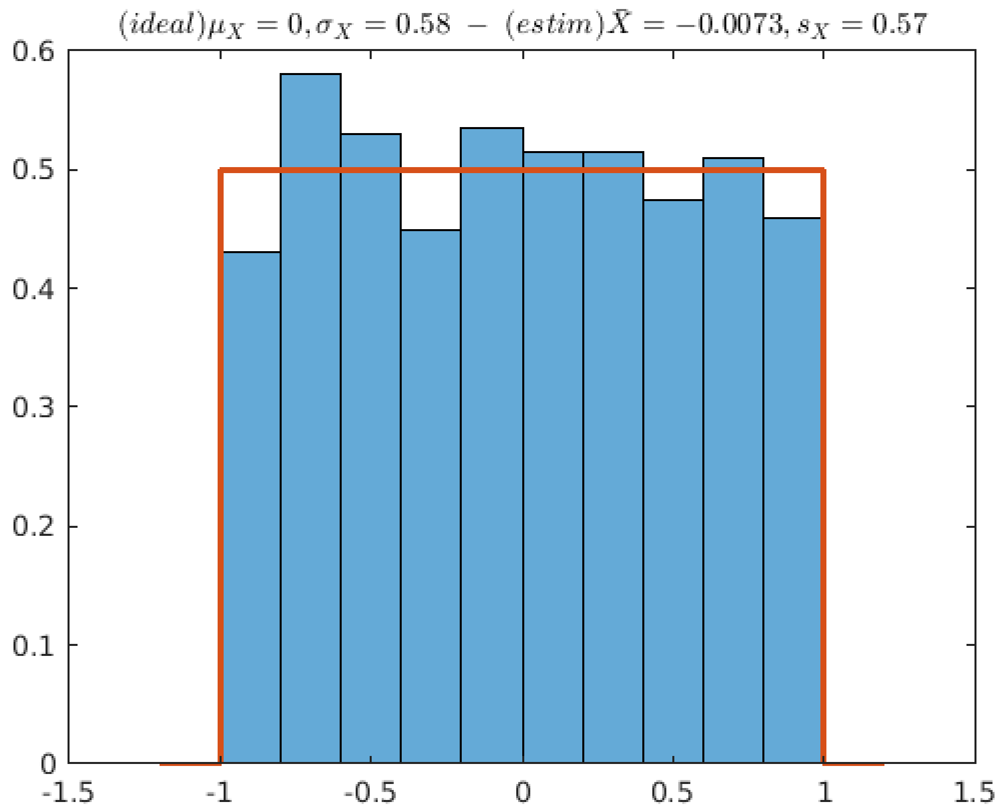
$$\sigma_X = \frac{2}{\sqrt{12}} \cong 0.57735$$

N.B. when we estimate these parameters from empirical data we can only approach the modelled values.

```
SIGNAL_LEN = 1000;
SIGNAL_AMPLITUDE = 0;  % amplitude of the "useful" signal
signal = SIGNAL_AMPLITUDE * sin(2*pi*(1:SIGNAL_LEN)'/SIGNAL_LEN);
uwnoise = -1 + 2*rand(SIGNAL_LEN,1);  % uniform white noise (N.B. uniform, non gaussian)
var_x = signal + uwnoise;

figure(3)
clf
plot(var_x, '.')
ylim([-1.5 1.5])
title ("uniformly distributed white noise")
```

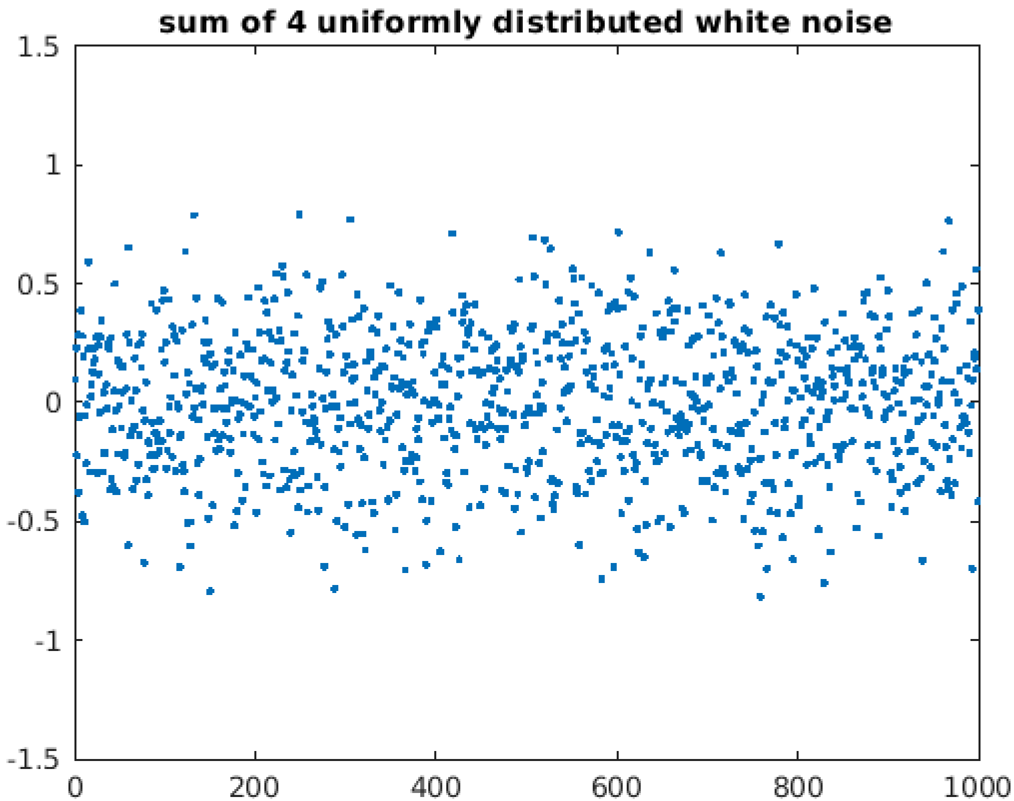



```
figure(4)
clf
histogram(uwnoise, "Normalization","pdf")
hold on
fplot(@(x) 1/2 * double(abs(x)<1), [-1.2 1.2], "LineWidth",2)
hold off
m_x = mean(uwnoise);
s_x = std(uwnoise);
titlestr = "$"+...
    "(ideal) \mu_X=0, \sigma_X="+sprintf("%.2g", 2/sqrt(12)) + "\;-\;" + ...
    "(estim) \bar{X}="+sprintf("%.2g", m_x)+", s_X="+sprintf("%.2g", s_x) + ...
    "$";
title(titlestr, "Interpreter","latex")
```

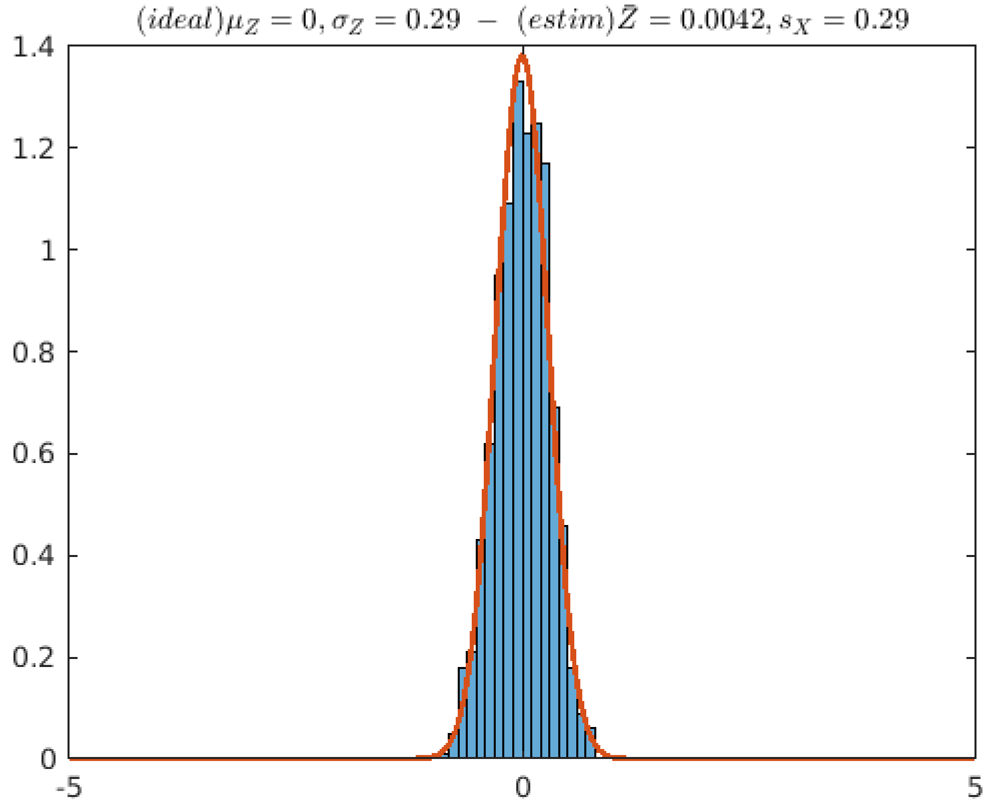


generate N independent copies of the sample, and take their average

```
N=4;
noise_copies = -1 + 2*rand(SIGNAL_LEN,N);
vars_xi = signal + noise_copies;
var_z = mean(vars_xi, 2);

figure(5)
clf
plot(var_z, '.')
ylim([-1.5 1.5])
title ("sum of "+ N +" uniformly distributed white noise")
```




```
figure(6)
clf
histogram(var_z-signal, "Normalization","pdf")
hold on
fplot(@(x) normal_pdf(x,0,(1/sqrt(N) * 2/sqrt(12))), "LineWidth",2)
hold off
m_z = mean(var_z-signal);
s_z = std(var_z-signal);
title("\sigma_Z="+sprintf("%.2g", s_z))

titlestr = "$"+...
    "(ideal) \mu_Z=0, \sigma_Z="+sprintf("%.2g", (1/sqrt(N) * 2/sqrt(12))) + "\;-\;" + ...
    "(estim) \bar{Z}="+sprintf("%.2g", m_z)+", s_X="+sprintf("%.2g", s_z) + ...
    "$";
title(titlestr, "Interpreter","latex")
```


$(ideal)\mu_Z = 0, \sigma_Z = 0.29$ – $(estim)\bar{Z} = 0.0042, s_X = 0.29$
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-5
0
5

,

FROM HERE 06 MAGGIO

Neuroengineering - Spectral analysis

or integral

Fourier Analysis, i.e. decomposition of signals into sum of sinewaves. Since each sinewave carries power at exactly one frequency, the decomposition can be used to analyze the signal in the frequency domain. Specifically Discrete Fourier Transform can be used to transform the (time-limited and sampled) time-domain representation of a signal into its (bandwidth limited an sampled) representation in the frequency domain.

DISCRETE FOURIER TRANSFORM

Using the DFT to analyze the spectral content of a (stochastic) signal may limit the ability to interpret the results. Specific techniques are commonly in use:

- Zeropadding and windowing are techniques aimed at compensating the effect on the spectrum of a limited number of samples in the time domain (low resolution, sidelobes)
- Power Spetral Density (PSD) can be estimated techniques such as the *averaged periodogram* which limit the variability of the power estimate at the expense of a loss of spectral resolution.

See also:

- Semmlow, Biosignal and medical image processing, Chapter 3
- (https://www.dspguide.com/pdfbook.htm, Chapters 6 and 31)

A sinewave is a function of time, with parameters: frequency f, amplitude A, and initial phase ϕ

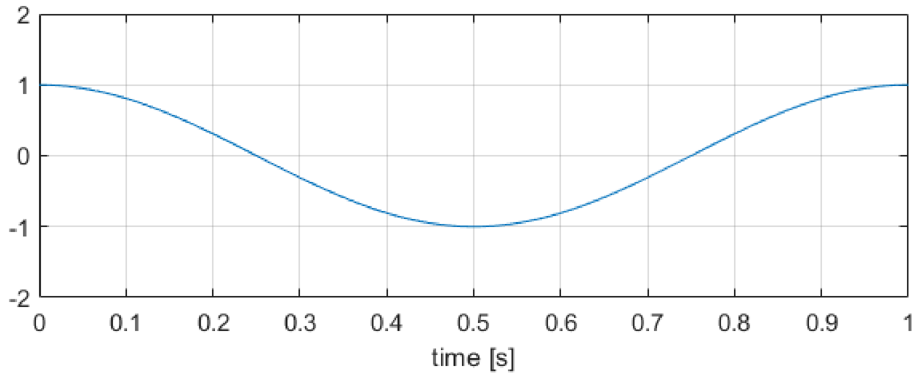
amplitude

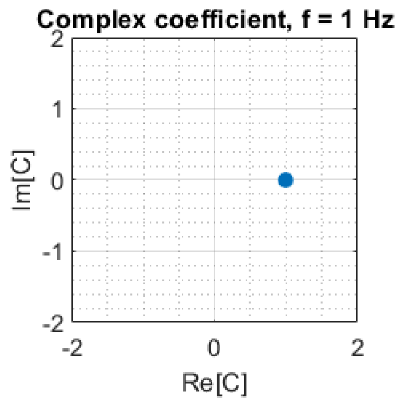
```
sinewave = @(t, f,A,phi) A * cos(2*pi*f*t + phi);
t = (0:.005:1)';  % seconds

f = 1;  % Hz
A = 1;
phi = 0 * pi;  % radians

figure(1)
clf
subplot 211
plot(t, sinewave(t, f,A,phi))
xlim([0 1])
ylim([-2 2])
grid on
xlabel("time [s]")
```

$C = Ae^{j\phi} \Leftrightarrow A = |C|, \phi = \angle C$

```
C = A*exp(1j*phi);  % Complex amplitude
subplot 212
plot(real(C), imag(C), '.', 'MarkerSize', 20)

axis square equal
xlim([-2 2])
ylim([-2 2])
grid on; grid minor
title("Complex coefficient, f = " + f + " Hz")
```

```
xlabel("Re[C]")
ylabel("Im[C]")
```



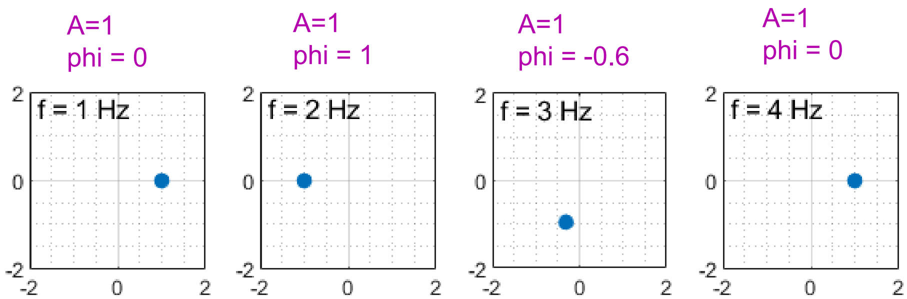


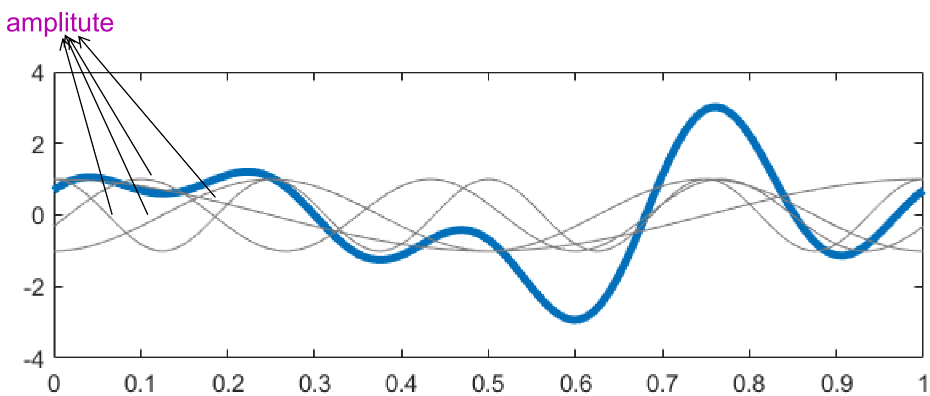
I can represent amplitude and phase at the same time by using a complex number

Sinewaves can be composed to create more complex waveforms:

```
f1 = 1;  % Hz
ff = f1 .* [1 2 3 4];
CC(1) = 1 * exp(1j * 0 * pi);
CC(2) = 1 * exp(1j * 1 * pi);
CC(3) = 1 * exp(1j * -0.6 * pi);
CC(4) = 1 * exp(1j * 0 * pi);

waveform = @(t) ...
    sinewave(t, ff(1), abs(CC(1)), angle(CC(1)) ) + ...
    sinewave(t, ff(2), abs(CC(2)), angle(CC(2)) ) + ...
    sinewave(t, ff(3), abs(CC(3)), angle(CC(3)) ) + ...
    sinewave(t, ff(4), abs(CC(4)), angle(CC(4)) );

figure(2)
clf
for k=1:4
    subplot(2,4,k)
    plot_complex_coefficient(CC(k), ff(k));
end
```

```
subplot(2,1,2)
plot(t, waveform(t), "Linewidth", 3)
hold on
for k=1:4
    plot(t, sinewave(t, ff(k), abs(CC(k)), angle(CC(k))), "Color", "#808080" );
end
hold off
```





Decompose and reconstruct arbitrary waveforms

complex number — sequence of sample x

Discrete Fourier Transform: $X[m] = DTF(x[n])$ the square brackets indicate that x is not a continuous signal but sampled

lenght of frequency domain signal — x[n] is the value of each sample

Transformation from time (N real samples, taken every $1/f_s$ seconds) to frequency domain (N complex samples, spanning $[0, f_s)$ Hz):

FFT - Fast Fourier Transform - is a numerical optimization of the DFT

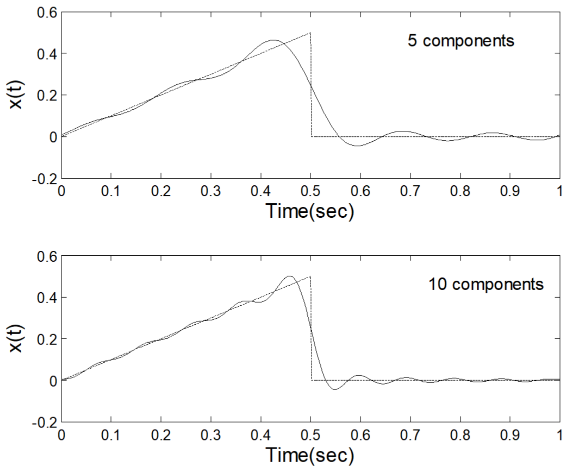
$$X[m] = \sum_{n=1}^{N} x[n] e^{-j2\pi mn/N}$$

Antitransformation from frequency to time domain:

n

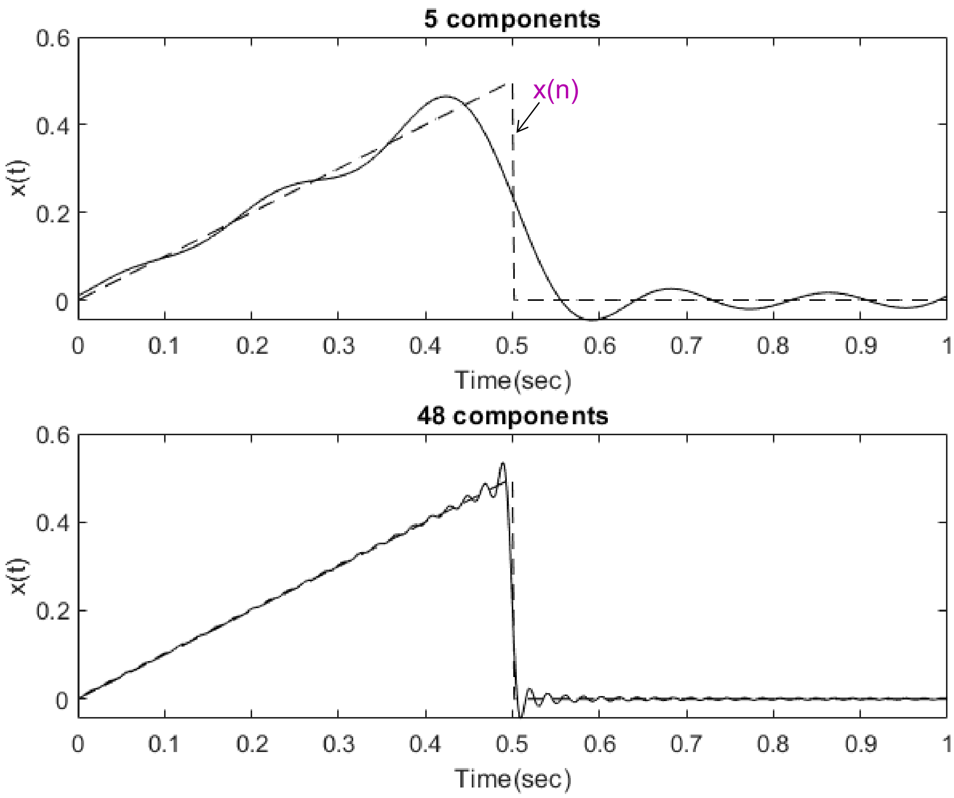
$$x[\cancel{m}] = \frac{1}{N} \sum_{m=1}^{N} X[m] e^{+j2\pi mn/N}$$

Demonstration -- decompose and reconstruct a ramp.



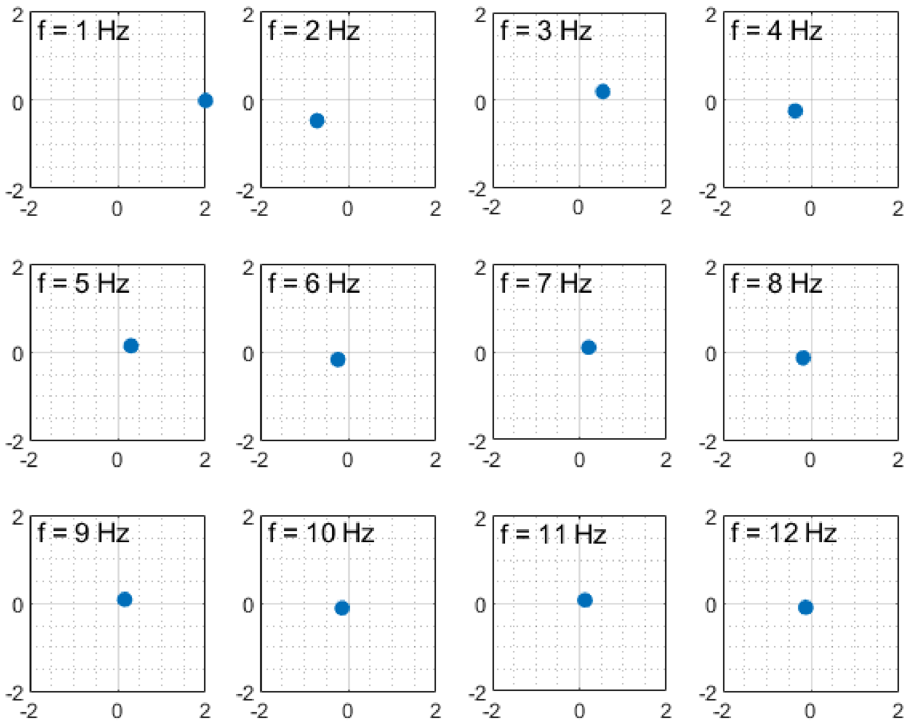

```
% Example 3.2 Perform a discrete Fourier series analysis on
%  the triangular waveform defined by the equation given
%  (modified to use complex coefficients)
% clear all; close all;
fs = 500;             % Sampling frequency
Tt = 1;               % Total time
N = Tt*fs;            % Determine N
f1 = 1/Tt;            % Fundamental frequency
t = (1:N)/fs;         % Time vector
% Construct waveform
x = zeros(1,N);       % Constuct waveform
x(1:N/2) = t(1:N/2);

% Fourier decomposition
% a0 = 2*mean(x);       % Calculate a(0).
X = fft(x);    this is the real transformation        here we compute the DFT

fft_magnitude = abs(X);
X_mag0 = fft_magnitude(1);  % remove DC, and pick only the first 10 components
X_mag = fft_magnitude(2:N/2);  % remove DC, and pick only the first 10 components
fft_angle = angle(X);           we compute the magnitude and the angle of the complex signal X
X_phase = fft_angle(2:N/2);
X_phase = unwrap(X_phase);      % Compenstes for shifts > 2 pi

% Reconstructions
x1 = zeros(1,N);
x1 = x1 + X_mag0/N;          % Add in DC term
for m = 1:5
    f(m) = m*f1;         % Sinusoidal frequencies
    x1 = x1 + 2/N * X_mag(m)*cos(2*pi*f(m)*t + X_phase(m)); % Eq. 3.15
end
subplot(2,1,1);
plot(t,x1,'k'); hold on;
plot(t,x,'--k');     % Plot reconstructed and original waveform
xlabel('Time(sec)','FontSize',14);
ylabel('x(t)','FontSize',14);
title('5 components','FontSize',12);
%
```

```
x2 = zeros(1,N);
NUM_COMPONENTS = 48;
x2 = x2 + X_mag0/N;          % Add in DC term
for m = 1:NUM_COMPONENTS
    f(m) = m*f1;          % Sinusoidal frequencies
    x2 = x2 + 2/N * X_mag(m)*cos(2*pi*f(m)*t + X_phase(m));
end
% x2 = x2 + a0/2;          % Add in DC term
subplot(2,1,2);
plot(t,x2,'k'); hold on
plot(t,x,'--k');     % Plot reconstructed and original waveform
xlabel('Time(sec)','FontSize',14);
ylabel('x(t)','FontSize',14);
title(sprintf('%d components', NUM_COMPONENTS),'FontSize',12);
```



Display ther first 12 complex coefficients (note the arbitrary scale factor)

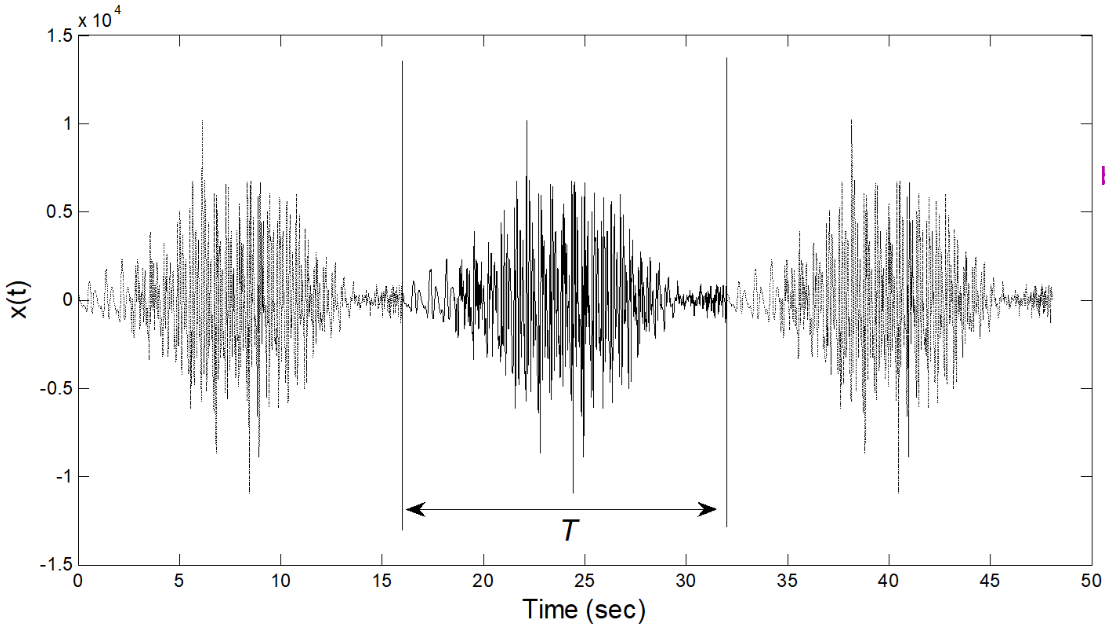
```
figure(3)
clf
for k = 1:12
    subplot(3,4, k)
    SCALE_FACTOR = 2/max(X(2:13)); % arbitrary scaling, for display purposes only
    plot_complex_coefficient(X(k+1)*SCALE_FACTOR, f(k));
end
```



Magnitude and phase plots

Representation in the frequency domain is mathematically equivalent to the time-domain representation.

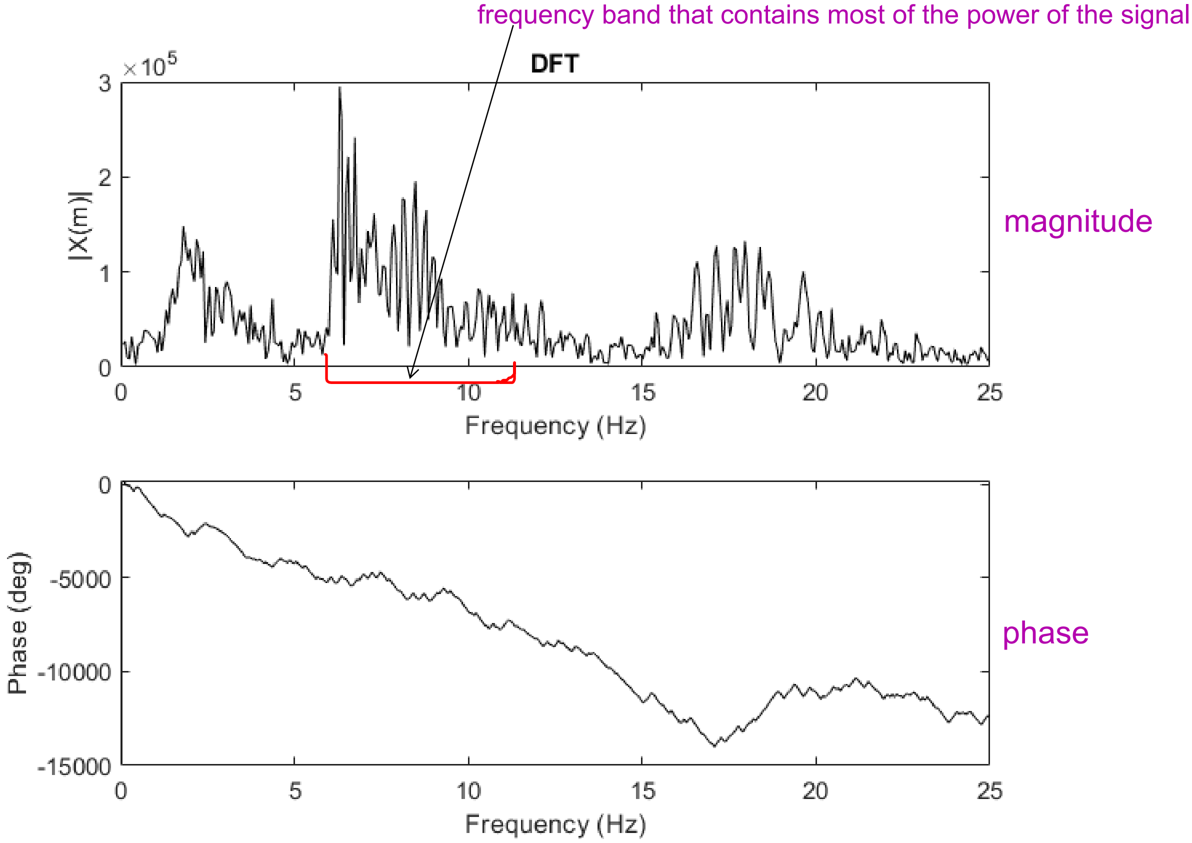
We still have to find a way to represent complex numbers. The most useful choice is to use two plots, one for the magnitude and one for the (unwrapped) phase.



biological signal

```
% Example 3.1 Use Fourier series analysis to generate the magnitude and
%  phase plot of the ECG signal originally shown in Figure 2.16 and repeated below.
%
% clear all; close all;
fs = 50;                        % Sample frequency
load eeg_data
N = length(eeg);               % Get N
% ...
figure(4);
N2 = round(N/2);
f = (1:N2)*fs/(2*N2);
X = fft(eeg);
X_mag = abs(X);
X_phase = angle(X);
X_phase = unwrap(X_phase);      % Compenstes for shifts > 2 pi
X_phase = X_phase*360/(2*pi);   % Convert phase to deg.
subplot(2,1,1);
plot(f,X_mag(1:N2),'k');                    % Plot magnitude spectrum
xlabel('Frequency (Hz)','FontSize',14);
ylabel('|X(m)|','FontSize',14); title('DFT')
subplot(2,1,2);
plot(f,X_phase(1:N2),'k');                  % Plot phase spectrum
xlabel('Frequency (Hz)','FontSize',14);
ylabel('Phase (deg)','FontSize',14);
```

if we use "k." we obtain another type of graph with points



Detecting narrowband signal in wideband noise

(sinewaves are the most narrow -band signal, white noise is the most wide-band signal)

The function `sig_noise()` generates data consisting of sinusoids and noise useful in evaluating spectral analysis algorithms.

The calling structure `for sig_noise` used in this example is:

`[x,t] = sig_noise([f],[SNR],N);` this generates N samples of white noise on top of which there is a sinewave at frequency n and the amplitude is such that it achieves a specified SNR

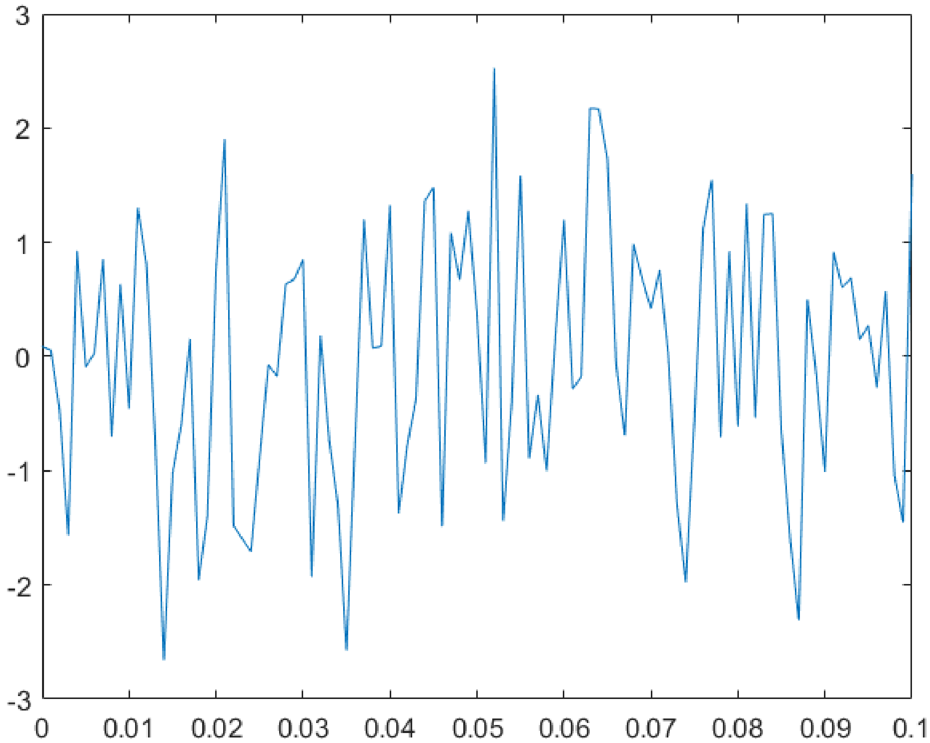
where `f`s pecifies the frequency of the sinusoid(s) in Hz, `N` is the number of points, and `SNR` specifies the desired noise in dB associated with the sinusoid(s). If `f` is a vector, multiple sinusoids are generated.

```
% Example 3.3  Determine the magnitude and phase spectrum of a noisy
%       waveform
% First generates a waveform consisting of a single sine in noise,
%     then calculates the magnitude and phase spectum from the FFT and plot
% clear all; close all;
N = 1024;                 % Data length
N2 = 511;                 % Valid spectral points
fs = 1000;                % Sample frequency (assumed by sig_noise)
[x,t] = sig_noise(250,-7,N); % Get data (250 Hz sin plus white noise)
figure(5)
clf
plot(t,x)
xlim([0, 0.1])
```

the fact that SNR is negative, it means that the amplitude of the sinewave is almost 10 times time lower than the amplitude of the noise

(SNR is the ratio of the amplitude/power of signal over noise, expressed in dB)

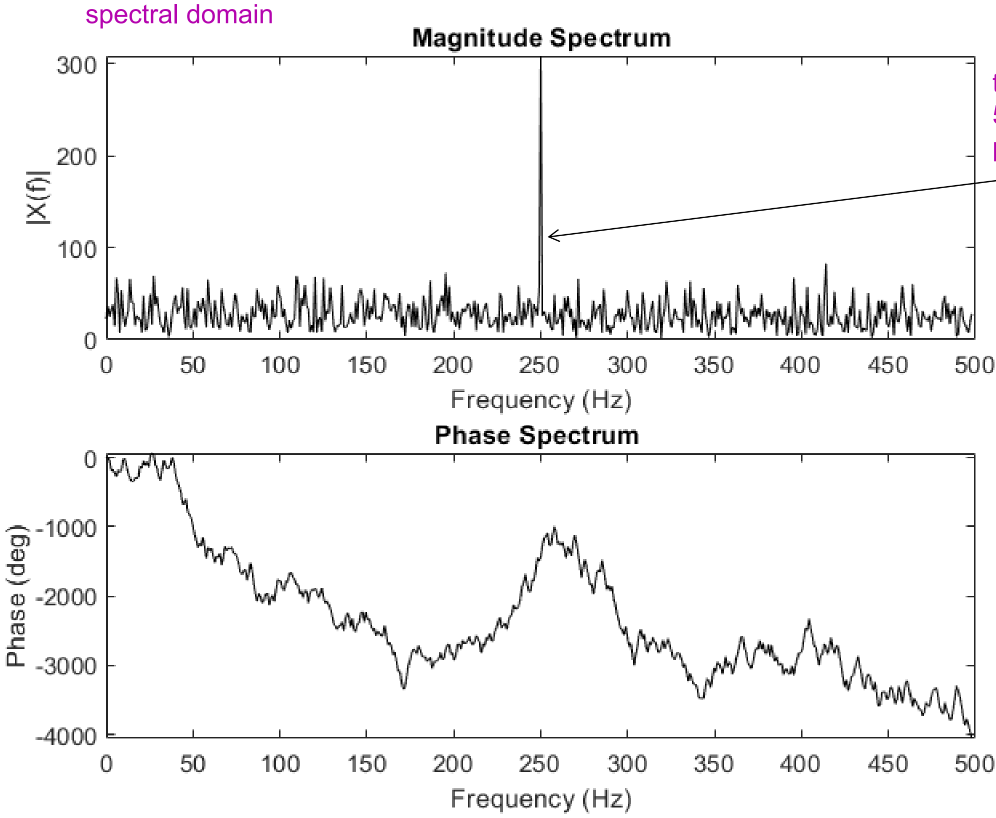
SNR/dB = 20 * log_10 (A_signal / A_noise) = 10 * log_10 (P_signal / P_noise) time domain



the signal is sintetized by adding some white noise to sinewave

```
X = fft(x);           % Calculate FFT
X_mag = abs(X);            % Compute magnitude spectrum
Phase = unwrap(angle(X));  % Phase spectrum unwraped
Phase = Phase*360/(2*pi);  % Convert phase to deg
f = (0:N-1)*fs/N;          % Frequency vector

figure(6)
clf
subplot(2,1,1);
plot(f(1:N2),X_mag(1:N2),'k');      % Plot magnitude spectrum
title('Magnitude Spectrum','FontSize',12);
xlabel('Frequency (Hz)','FontSize',14);
ylabel('|X(f)|','FontSize',14);
subplot(2,1,2);
plot(f(1:N2),Phase(1:N2),'k');      % Plot phase spectrum
title('Phase Spectrum ','FontSize',12);
xlabel('Frequency (Hz)','FontSize',14);
ylabel('Phase (deg)','FontSize',14);
```

spectral domain



the white noise is distributed from 0 to 500, while the sinewave has all of its power concentrated here

Zero padding

Duality in time vs. frequency domain

Higher sampling rate in t ⇔ Broader spectrum in f

Longer signal in t ⇔ Higher resolution in f

Zeropadding (i.e. adding a row of zeros at the end of the signal) allows to increase (artificially) the spectral resolution.

```
% Example 3.4 Generate a 1.0 second wave symmetrical triangle wave.  Make fs = 100 Hz s
% N = 100 points.  Calculate and plot the magnitude spectrum.
% Zero pad so the period is extended to 2 and 8 sec. and recalculate and plot the
% magnitude spectrum.  Since fs = 100 Hz the signal should be padded to 200 and 600 Poi
%
% clear all, close all;
fs = 100;                           % Sample frequencies
N1 = [0 150 750];                   % Padding added to the original 50 point signal
x =[(0:25) (24:-1:0)];              % Generate basic test signal, 50 pts long

figure(7); clf
for k = 1:3
    x1 = [x zeros(1,N1(k))];         % Zero pad signal
    N = length(x1);                  % Data length
    t = (1:N)/fs;
```