Basics of signal processing (2)

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_01_ADC.mlx (cont'd)
    - NEng_1920_02_Stats.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 - Analog to Digital Conversion

Contents:

- sampling, Shannon's or Nyquist's sampling theorem, aliasing, antialiasing filters
- quantization, number of bits, q. noise, (saturation high-pass filter)

See also https://www.dspguide.com/pdfbook.htm, Chapter 3

---------- 01/04/2020 ----------

Sampling

Try the following:

1. Choose a low F_0 (e.g. 5 Hz) and assess that you can visually interpolate the samples to recover the analog waveform
2. Increase F_0 (e.g. 30 Hz) and note that the visual interpolation becomes much harder. Nevertheless the reconstruction using the `sinc()` is successful.
3. Set F_0 just below 50 Hz (e.g. 49 Hz) and note that the visual interpolation is misleading (two low-frequency sinewaves seem to appear). Even in this case the `sinc()` reconstruction is successful.
4. Set F_0 just above 50 Hz (e.g. 51 Hz) and note that the visual interpolation is qualitatively not different from the previous case. Despite this, the `sinc()` reconstruction fails (the reconstructed sinewave seems to be at a lower frequency).
5. Set F_0 just below 100 Hz (e.g. 99 Hz) and note that the visual interpolation is misleading (a very low-frequency sinewave appears). In this case, the `sinc()` reconstruction fails in a way that is consistent with the visual interpolation (the same very low frequency sinewave is reconstructed).

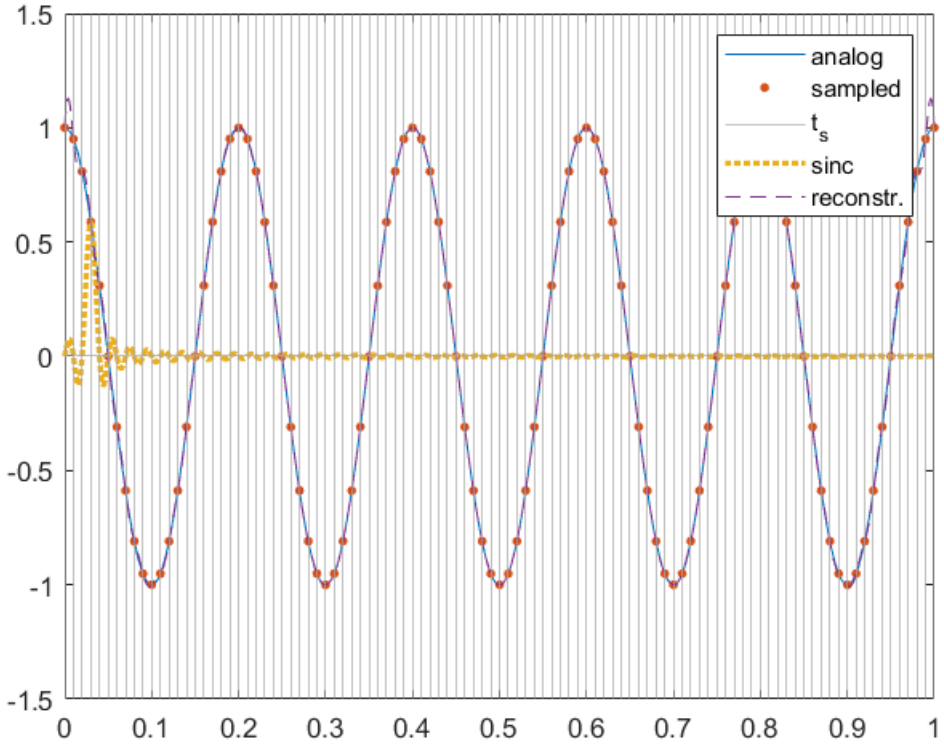
```
% Generate the desired signal (sinewave)
F_0 =5;  % Hz
sig = @(t) cos(2*pi*F_0*t);  % anonymous function

t = (0:.001:1)';    % s, proxy of continuous time
x = sig(t);         % proxy of analog signal for plotting

% Sample the signal
DO_SAMPLING = true;
if DO_SAMPLING
    F_samp = 100;  % Samples/second (S/s), sampling frequeny
    T_samp = 1/F_samp;  % s, sampling interval
    t_sampled = (0:T_samp:1)';   % sampling times
    x_sampled = sig(t_sampled);  % sampled signal
    %     [x_sampled2,t_sampled2] = resample(x,t,F_samp);
end% if

% Reconstruct an analog signal from the samples
```

```
DO_RECONSTRUCTION = true;
if DO_RECONSTRUCTION
    % convoluted syntax to avoid for-cicles for efficiency
    % basically, it generates a sinc() on each sample
    % and sums them all
    [t_grid,ts_grid] = ndgrid(t,t_sampled);
    sincs = sinc((t_grid - ts_grid) / T_samp);
    x_reconstructed = sincs * x_sampled;
end


% === CREATE THE FIGURE ===
figure(1)
clf

% Show the analog signal
plot(t,x, "DisplayName", "analog")
yline(0, "Color", "#aaa",'HandleVisibility','off')
```

```
ans =
  ConstantLine with properties:

    InterceptAxis: 'y'
            Value: 0
            Color: [0.6667 0.6667 0.6667]
        LineStyle: '-'
        LineWidth: 0.5000
            Label: ''
      DisplayName: ''

  Show all properties
```

```
xlim([0,1])  % width of the figure's time axis

% Show the samples
if DO_SAMPLING
    hold on
    plot(t_sampled, x_sampled, '.', "MarkerSize",10, "DisplayName","sampled")
    hold off
    if true% show sampling timepoints
        for tt=t_sampled', h = xline(tt, "Color", "#aaa",'HandleVisibility','off'); end
        h.HandleVisibility = "on";
        h.DisplayName = "t_s";
    end
end% if

% Show the reconstructed analog signal
if DO_RECONSTRUCTION
    hold on
    if true% show one of the sinc's
        SINC_SAMPLE =4;  % which sinc will be plotted
        plot(t, sincs(:,SINC_SAMPLE)*x_sampled(SINC_SAMPLE), ':', ...
            "LineWidth",2, "DisplayName","sinc"),
    end
    % the reconstructed signal is the sum of all sincs
    plot(t, x_reconstructed, '--', "DisplayName","reconstr.")
```

```
    hold off
end
legend
```



Sampling theorem

(a.k.a. Shannon's theorem, a.k.a. Nyquist's theorem):

A continuous signal can be *properly* sampled, only if it does not contain frequency components above one-half of the sampling rate.

(*Properly* meas that the analog signal can be correctly reconstructed from its samples.)

One-half the sampling frequency is named the *Nyquist frequency.*

Failing to sample in the conditions of the Shannon's theorem, produces a phenomenon called **aliasing**, i.e. the appearance of "ghost" frequencies below the Nyquist frequency, which are in fact *mirrored* from higher frequency bands. In this example ($f_{Nyquist} = 50Hz$) , a sinewave at 51 Hz is reconstructed as a sinewave at 49 Hz, and a sinewave at 99 Hz is reconstructed as a sinewave at 1 Hz.

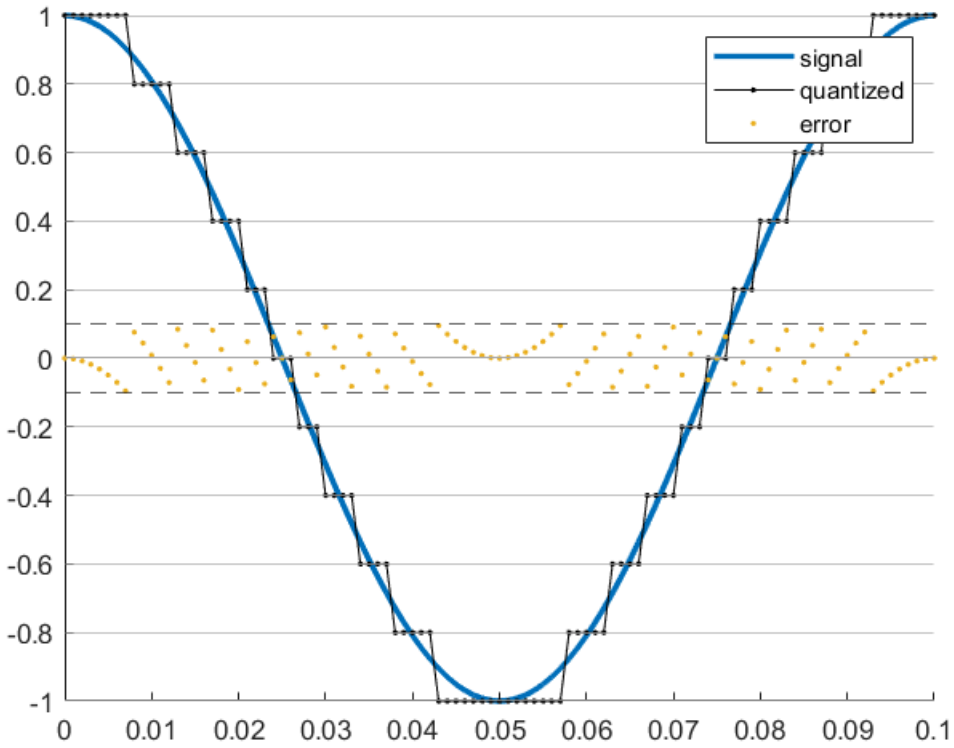
Sampling a signal containing (non-interesting) components with arbitrarly high frequency, requires an *analog* low-pass filter to condition the analog signal (**anti-alias filter**) so that for a given sampling frequency the Shannon's theorem is fulfilled. By doing so, the high frequency component is lost, but at least it does not contaminate through aliasing the useful signal components at lower frequencies.

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

Quantization

```
QUANT_INTERVAL = 0.2;
quantize = @(val) round(val/QUANT_INTERVAL) * QUANT_INTERVAL;
x_quantized = quantize(x);
quant_err = x-x_quantized;

% === DISPLAY QUANTIZATION ===
figure
clf
hold on

plot(t,x, '-', "LineWidth",2)
plot(t,x_quantized, 'k.-')

if true% Display error
    plot(t,quant_err, '.')
    yline(QUANT_INTERVAL/2, '--');
    yline(-QUANT_INTERVAL/2, '--');
end

yline(0, "Color", "#aaa");
for yy = quantize(min(x)):QUANT_INTERVAL:quantize(max(x))
    yline(yy, "color", "#aaa");
end

hold off

xlim([0,.1])
ylim([-1 1])
legend("signal", "quantized", "error")
```

real analog to digital converter are also characterized by a maximum input range



The he quantization noise has zero mean and its standard deviation is proportional to the quantization interval

LSB:

LSB indica il bit meno significativo del quantizzatore

standard deviation

$$\sigma_{quant} = \frac{1}{\sqrt{12}} \cdot LSB$$

quantizing a signal introduces a noise given by this formula

If an Analog to Digital Converter has a range [0,R] (i.e. values of an input signal that does not saturate), and NBIT number of bits, its quantization noise is:

how many levels can we use? this is another parameter of the converter(not independent). Sometimes you can have the number of bits parameter

```
MAX_V_IN = .1;  % Range, millivolt
NBITS = 8;
quant_interval = MAX_V_IN / 2^NBITS;
noise_quant = 1/sqrt(12) * quant_interval;
disp("The std of the quantization noise is "+ sprintf("%.2g",1e3*noise_quant) +" uV.")
```

the first value in the sampling frequency

if our input range is 100 uV and number of bits is 8 which means 256 possible levels

```
The std of the quantization noise is 0.11 uV.
```

this value is acceptable

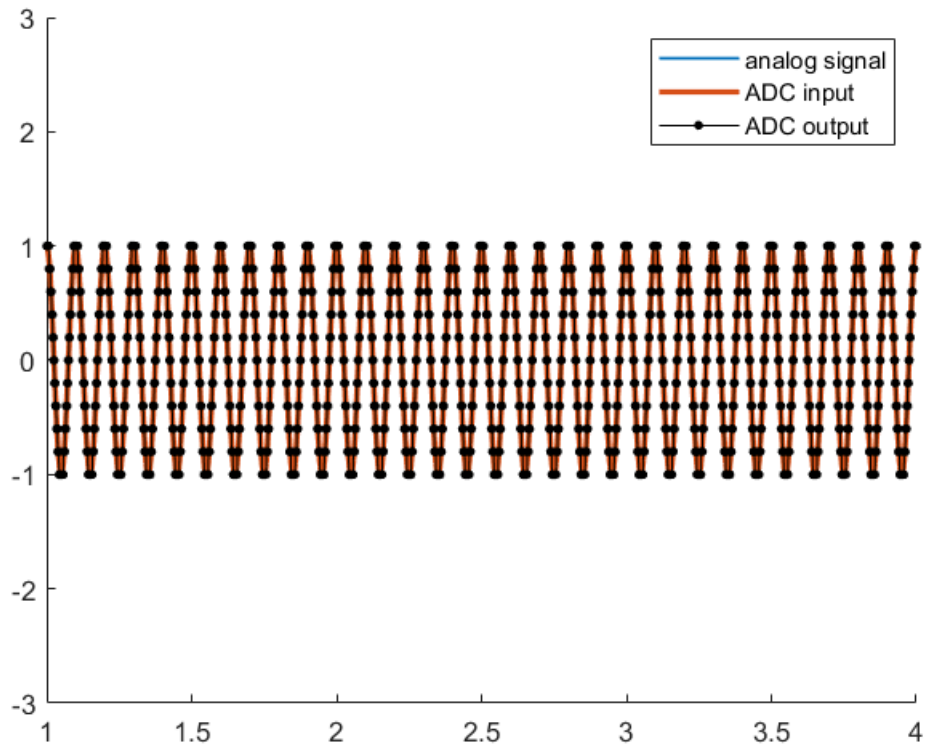
if we have an 8 bit converter, every value of the analogue signal would be converge into an integer number with 8 bits

if the subject that you're recording moves his eyes, we have studied that we have high amplitute and low frequency

Clipping (saturation) due to low frequency / high amplitude artifacts

- Superimpose high amplitude 1 Hz sinewave (artifact) on 10 Hz moderate amplitude sinewave (signal).
- Show effects of saturation.
- Mitigate with high pass filtering.

```
% Signal
F_0 =10;  % Hz, frequency of signal
sig_0 = @(time) cos(2*pi*F_0*time);  % signal

% Artifact
F_SLOW = 1;  % Hz, frequency of low frequency artifact
A_SLOW = 0;  % amplitude of low frequency artifact
sig_SLOW = @(time) A_SLOW * cos(2*pi*F_SLOW*time) ;  % low frequency artifact

% ADC input
sig = @(time) sig_0(time) + sig_SLOW(time) ; artefatto+segnale
t = (0:.001:5)';    % s, proxy of continuous time
% note that in this demonstration we do not care of sampling at F_samp
DO_ANALOG_HIGHPASS = false;
if DO_ANALOG_HIGHPASS
    x = highpass(sig(t));
else
    x = sig(t);
end

% Quantize
QUANT_INTERVAL = 0.1;
ADC_RANGE = [-2 2];  % input range of the ADC converter
% quantize = @(val) round(val/QUANT_INTERVAL) * QUANT_INTERVAL;
clip = @(val, range) min(max(val, min(range)), max(range));
quantize_and_clip = @(val) clip(quantize(val), ADC_RANGE);


% === DISPLAY QUANTIZATION ===
figure
clf
hold on
plot(t,sig(t), '-', "LineWidth",1)
plot(t,x, '-', "LineWidth",2)
plot(t,quantize_and_clip(x), 'k.-', "MarkerSize", 10)
hold off

xlim([1,4])
ylim([-3 3])
legend("analog signal", "ADC input", "ADC output")
```




```
function x = highpass(signal)
N = 8;
Fpass = 8;
Astop = 30;
Apass = .5;
Fs = 1e3;

d = designfilt('highpassiir', ...
  'FilterOrder',N, ...
  'PassbandFrequency',Fpass, ...
  'StopbandAttenuation',Astop,'PassbandRipple',Apass, ...
  'SampleRate',Fs);

    % fvtool(d)
    x = filtfilt(d, signal);

end
```

SULL'ALTRO BLOCCO CI SONO GLI APPUNTI E I DISEGNINI DEL PROF

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.

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):

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

ARV and RMS are applied to the deviation of the signal(signal - its mean)

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):

sample variance computed empirically

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

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

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

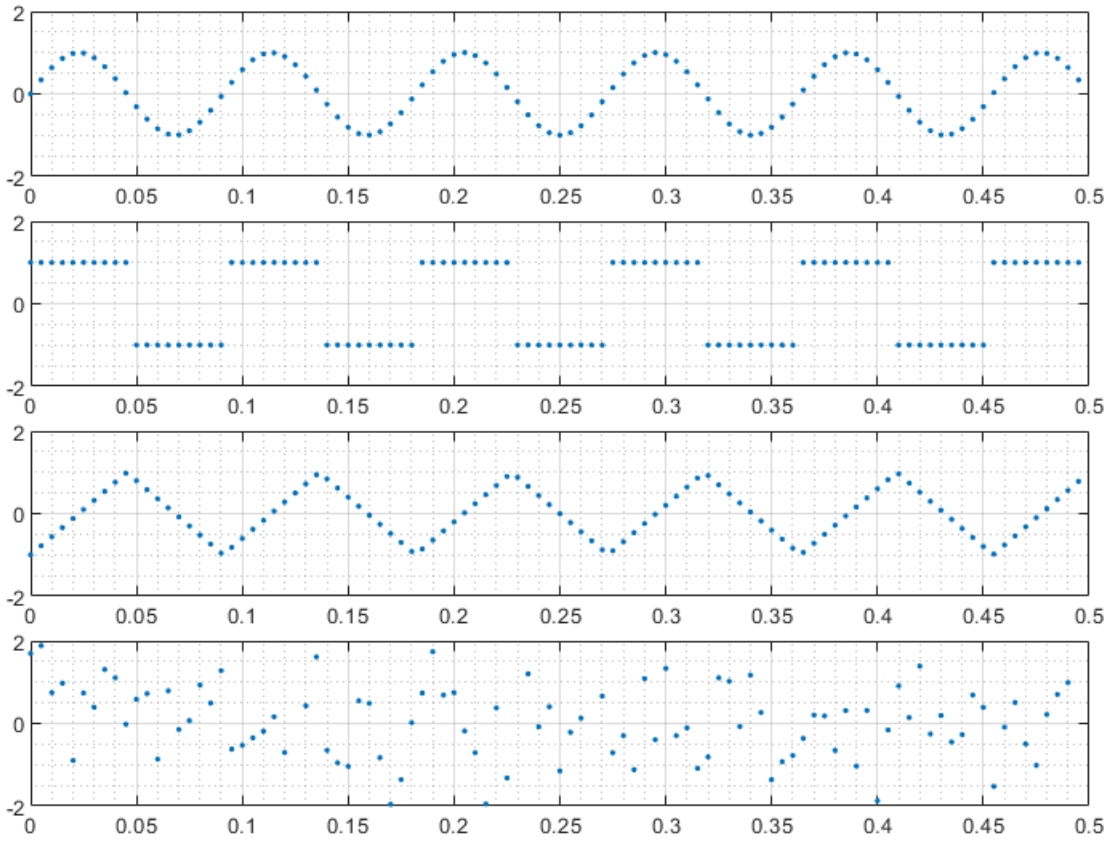
We introduce here four signals that we will analize in the following:

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
```

rand signal with normal distribution



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);
```

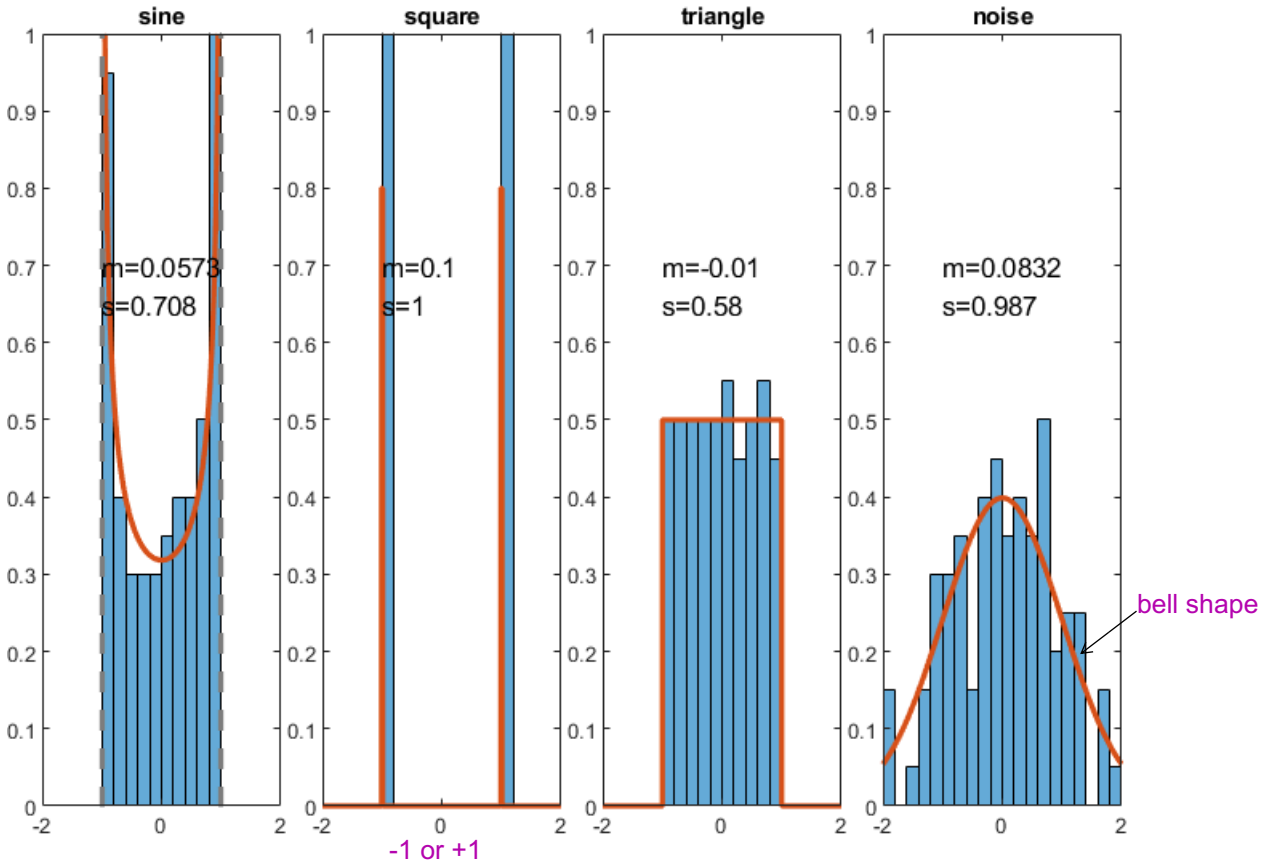
```
Warning: Data clipped when writing file.
```

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 square wave

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

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

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 $\sqrt{N}$:

$$\mu_Z = \mu_X$$

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

$$\sigma_Z = \frac{1}{\sqrt{N}}\sigma_X$$ ← standard deviation of the average potential

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:

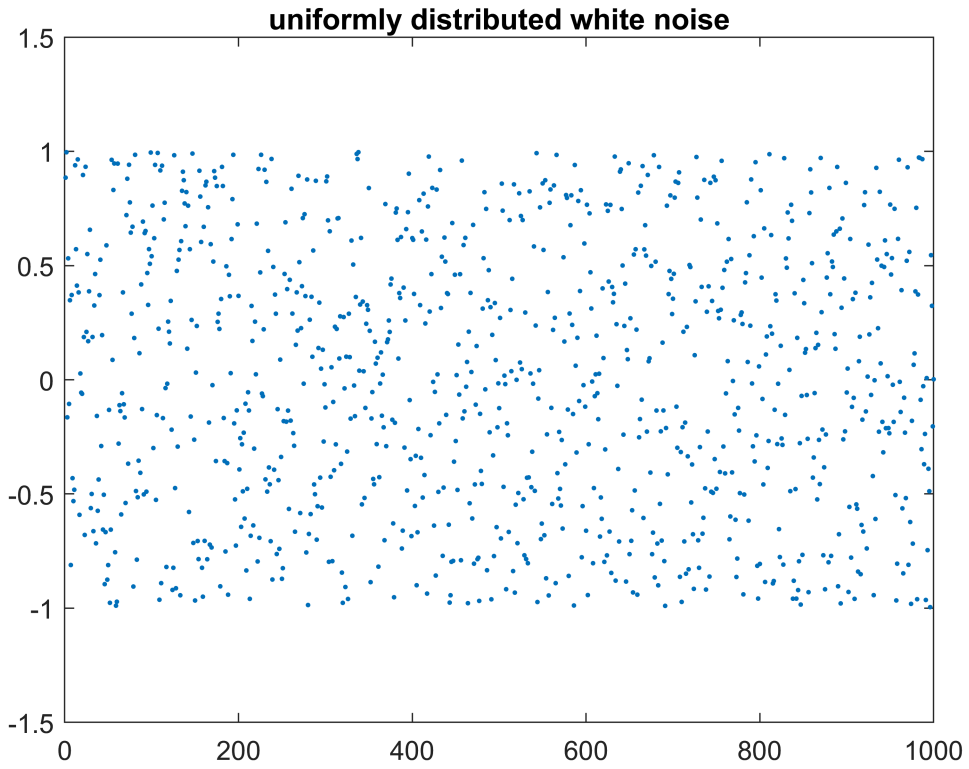
$\mu_X = 0$ (We will later manipulate the expected mean, by adding a small "useful" signal to this noise.)

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

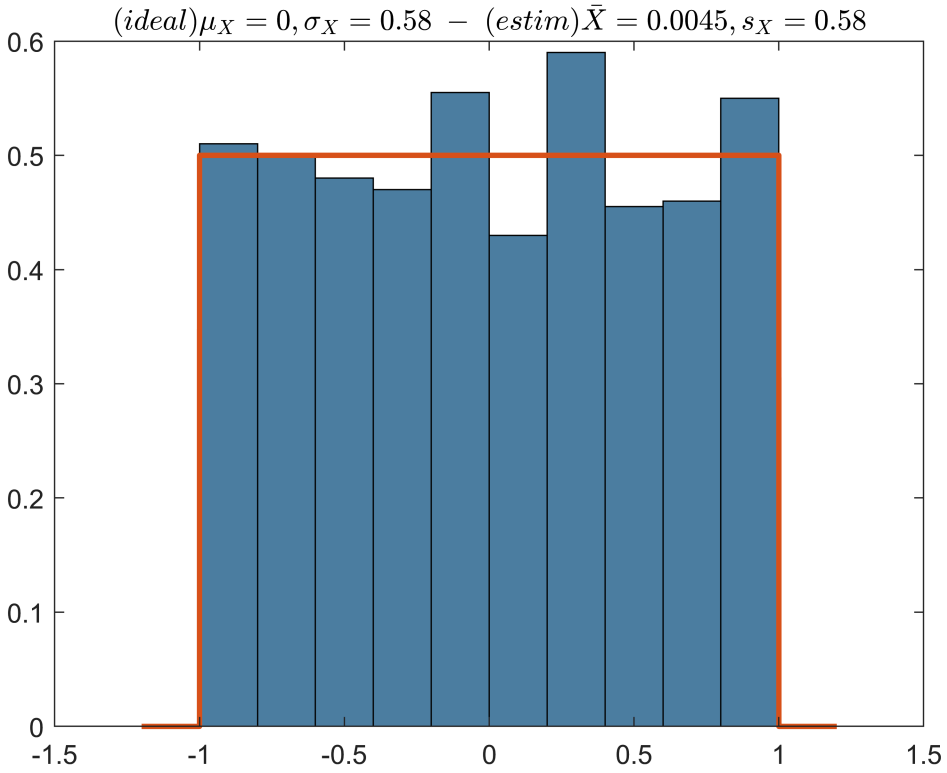
N.B. when we estimate these parameters from empirical data we can only approach the modelled values.

```
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")
```

no randn, so without n; it means that the probability of this signal is constant between 0 and 1



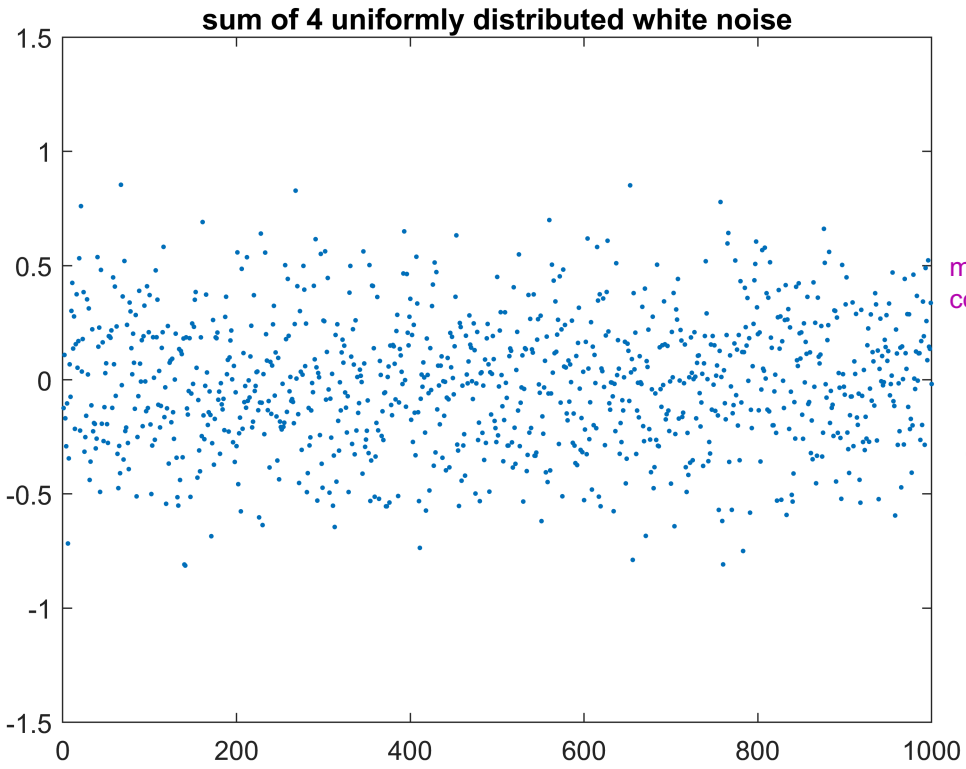

```
figure(4)
clf
histogram(uwnoise, "Normalization","pdf")
hold on
fplot(@(x) 1/2 * double(abs(x)<1), [-1.2 1.2], "LineWidth",2)
hold off
m_x = mean(uwnoise);
s_x = std(uwnoise);
titlestr = "$"+...
    "(ideal) \mu_X=0, \sigma_X="+sprintf("%.2g", 2/sqrt(12)) + "\;-\;" + ...
    "(estim) \bar{X}="+sprintf("%.2g",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



rectangular distribution

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

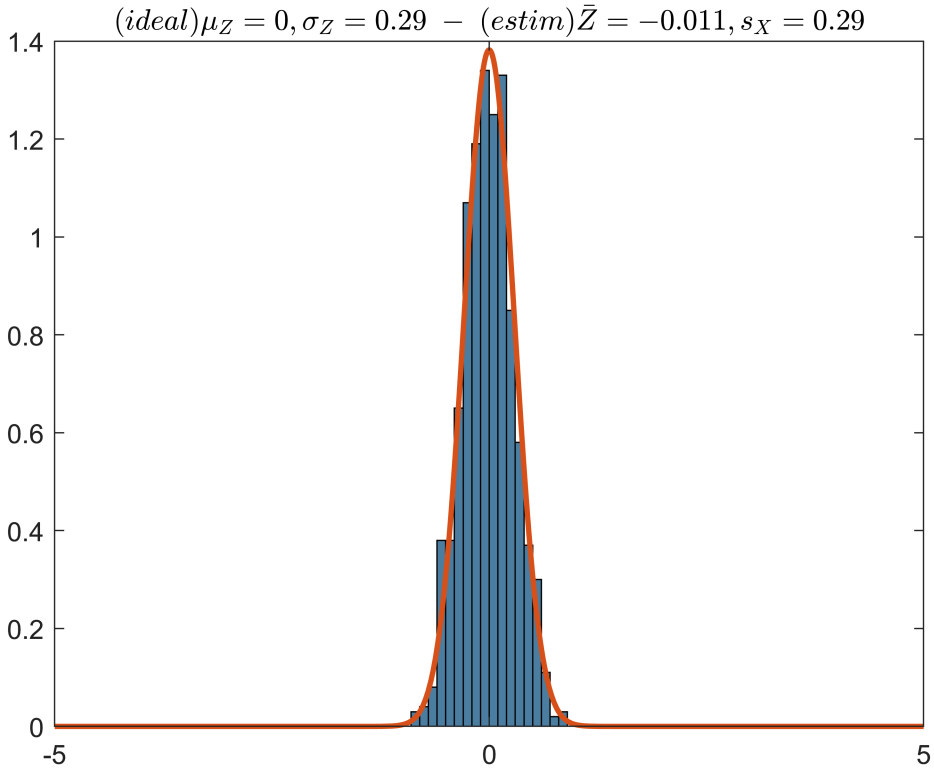
```
N=4; I can change the number of copies of the signal that in this case is 4
noise_copies = -1 + 2*rand(SIGNAL_LEN,N);
vars_xi = signal + noise_copies;
var_z = mean(vars_xi, 2);

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



most of the samples are concentrated around 0

if N increases the samples are more close to 0

```
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")
```



more triangular distribution

if N increases the samples are more similar to a higher gaussian

,