Signal Power, Noise, SNR and Auto and Cross Correlation

Eren Can Gungor
Riccardo Miccini
Technical University of Denmark - DTU

October 23, 2016

Contents

1 Problem 1.1

1.1 `rand` and `randn`

The MATLAB functions `rand` and `randn` are used to generate sequences of random numbers.

rand Uniformly distributed random numbers in the interval $(0, 1)$

randn Normally distributed random numbers with $\mu = 0$ and $\sigma = 1$

The functions support the same combinations of input arguments. They can be used to return a single scalar when called with no parentheses, or a matrix of $n_1 \times n_2 \times \ldots \times n_N$ elements.

1.2 `hist`

The MATLAB function `hist` is used to generate the histogram of a sequence of numbers, i.e. the distribution of the values within a series of non-overlapping intervals called bins. The histogram of a dataset can be used to estimate its probability distribution.

The function supports the following arguments:

x mandatory argument representing the input data

nbins number of bins into which the data is divided

xbins vector containing the center value of each bin

The following return values can be retrieved:

none if no assignment is specified, the function plots the histrogram

count vector containing number of elements in each bin

centers vector containing the center value of each bin; can be used, together with count, to manually plot the histogram

1.3 Examples

Here there will be shown example usages of the two functions, and proofs of their properties. The following code snippets generate a vector of 1000 numbers using `rand` and `randn` respectively, calculates their mean value and plot the histogram of the sequence with 10 bins.

```
a = rand(1000, 1);
mean(a)
hist(a, 10)
```

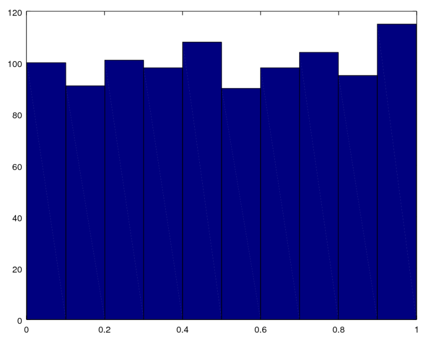

Figure 1: Histogram of rand function

As it can be seen from the figure above, the distribution of the numbers is approximately uniform, with all the values within the interval $(0, 1)$. The mean of the sequence is very close to 0.5.

```
a = randn(1000, 1);
mean(a)
std(a)
hist(a, 10)
```

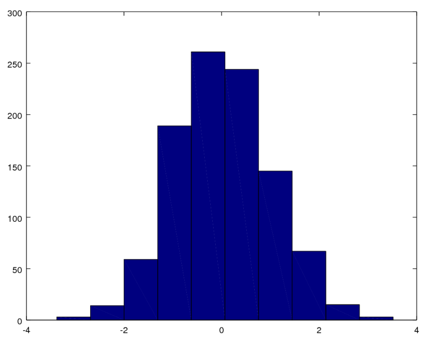

Figure 2: Histogram of randn function

As it can be seen from the figure above, the distribution of the numbers follows a gaussian curve. The mean of the sequence is very close to 0, and its standard deviation to 1.

1.4 `randn` and normal probability density function

In the following section, it will be shown how the values returned by `randn` follow the probability density function of the normal distribution, which is given by the formula:

$$\mathcal{N}(\mu,\sigma^2) = \frac{1}{\sqrt{2 * \sigma^2 * pi}} * \exp(-(x-\mu)^2/(2 * \sigma^2)) \tag{1}$$

The following code generates a sequence of numbers using `randn`, calculates its normalized histogram, and then plots it along with an approximated probability density. These operations are performed with the standard normal distribution, and with example values of μ and σ.

```
clear;
hold off;
clf;

title({'Histogram and probability density'})
figure(1)

% simulation parameters
avg = [0, 2, -5, .5];
sigma = [1, 5, .5, .1];
sigma2 = sigma.^2;
bins = 1000;

for i = 1:length(avg)
  % generate random sequence of normally-distributed numbers
  a = avg(i) + randn(1e6, 1) * sigma(i);
  % generate its histogram
  [hh, xx] = hist(a, bins);

  % generate probability density function
  x = linspace(min(a), max(a), bins);
  px = 1/sqrt(2*sigma2(i)*pi) * exp(-(x-avg(i)).^2 / (2*sigma2(i)));

  % plot histogram and probability density function
  subplot(1,length(avg),i)
  hold on;
  binwidth = xx(2)-xx(1);
  plot(xx, hh/(sum(hh)*binwidth), ":g")
  plot(x, px, ":b")
```

```
    txt = sprintf('{␣\\sl␣N(%.2f,␣%.2f)}', avg(i), sigma2(i));
    legend({"hist()", txt})
    hold off;
end
```

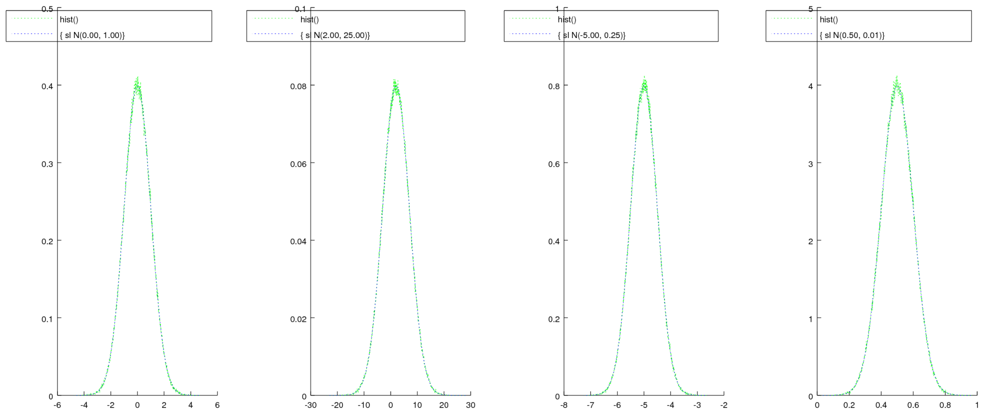

Figure 3: Examples of histograms of normally distributed values and PDF

As it can be seen in the figure above, the curves on each plot closely match, indicating that the assumption hold true.

1.5 Gaussian noise and sinusoidal signals

This section will analyze the effects of noise contamination using sinusoidal signals as reference.

The following code generates three sinewaves with the same frequency and varying phases. These waves are to be considered sampled versions of their continuous equivalents. They are then normalized in order to have a power of 1 mW. Subsequently, four gaussian noise signals are generated using the `randn` function, and scaled to have powers of 1, 10, 100 mW and 1 W respectively. The noise signal are summed to the original sinewaves, and plotted.

```
clear;
hold off;
clf;

title({'Gaussian␣noise␣and␣sinusoidal␣signals'})
figure(1)
```

```
% simulation parameters
fs = 44100;
f = 1000;
ph = [0 pi/2 pi];
pwr = [.001 .01 .1 1];

% create a time vector
n = 0:100-1;

% generate and normalize 3 sine signals
sig = zeros(length(n), 3);
for i = 1:3
  sig(:,i) = sin(2*pi*f*n/fs + ph(i));
  sig_norm(:,i) = sig(:,i) / std(sig(:,i)) * sqrt(1e-3);
end

% plot signals with legend
subplot(length(pwr)+1,1,1)
plot(n, sig_norm)
legend({'{sin(\omega*T_s*n)}',
  '{sin(\omega*T_s*n + \pi/2)}',
  '{sin(\omega*T_s*n + \pi)}'})

for i = 1:length(pwr)
  % generate noise signal
  noise = randn(1, length(n))';
  % normalize signal
  noise_norm = noise / std(noise) * sqrt(pwr(i));

  % sum sines and noise
  sig_noise = sig_norm + noise_norm;
  % calculate SNR
  snr(i,:) = var(sig_norm)/var(noise_norm);

  % plot noisy signals
  subplot(length(pwr)+1,1,i+1)
  plot(n, sig_noise)
end

% print SNRs
```

snr

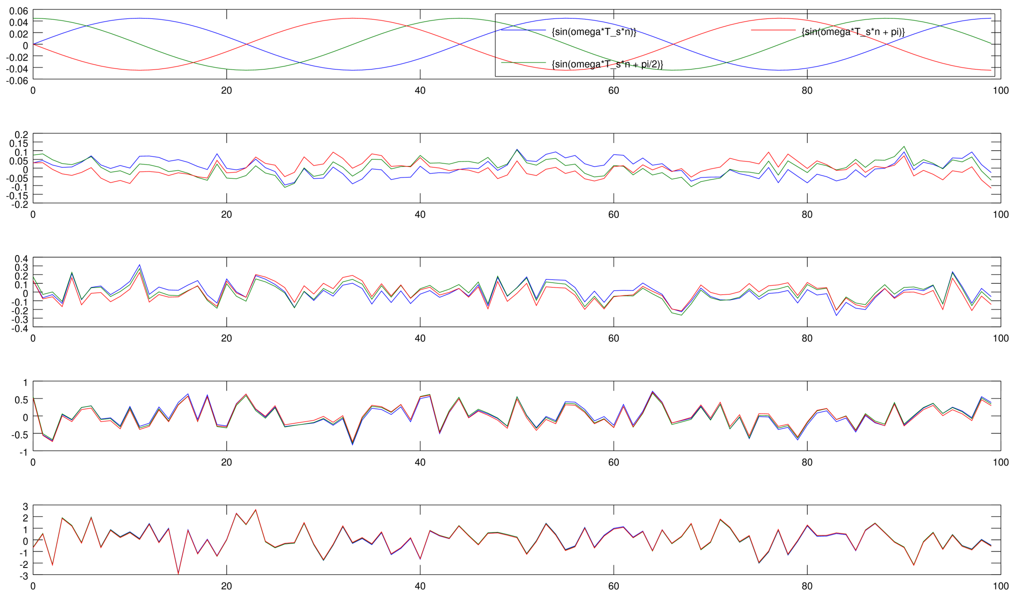

Figure 4: Examples of sinusoidal signals and noise, at various SNR levels

The figure above shows how difficult it becomes to discern the original sine signal from the noise, let alone identifying its phase. Moreover, the calculated SNRs match the expected values of 1, 0.1, 0.01, and 0.001.

2 Problem 1.2

2.1 `xcorr`

The MATLAB function `xcorr` is used to compute the cross-correlation of discrete-time signals. Given two discrete-time signals f_1 and f_2, their cross-correlation is defined as:

$$r_{f_1,f_2}[k] = \sum_{n=-\infty}^{\infty} f_1[n] * f_2[n+k] \tag{2}$$

Where k represents the amount of delay between the two signals. Such operation is particularly useful to verify whether two signals are related, or to identify specific features of a signal within another.

When such function is applied on the same signal, it takes the name of auto-correlation. For $k = 0$, the auto-correlation value of a signal corresponds to its power.

In MATLAB, the following arguments can be specified:

x if only one input vector is passed, its auto-correlation will be computed

y if a second input vector is passed, the cross-correlation between the two will be computed; the vectors must have the same length - if that is not the case, the shortest will be padded with zeros

maxlag optionally, a maximum value of k can be specified

The function returns a vector containing the result of the operation over various values of k.

2.2 Autocorrelation of the noisy sinusoidal signals

In this section, the `xcorr` function will be used to reconstruct the sinusoidal components of the signals from the previous problem.

The following code generates sinusoidal signals with various SNR levels, calculates their auto-correlation, and plots them.

```
clear;
hold off;
clf;
pkg load signal

title({'Autocorrelation␣of␣noisy␣sinusoidal␣signals'})
figure(1)
```

```
% simulation parameters
fs = 44100;
f = 1000;
ph = [0 pi/2 pi];
pwr = [0 .001 .01 .1 1];

% create a time vector
n = 0:100-1;

% generate and normalize 3 sine signals
sig = zeros(length(n), 3);
for i = 1:3
  sig(:,i) = sin(2*pi*f*n/fs + ph(i));
  sig_norm(:,i) = sig(:,i) / std(sig(:,i)) * sqrt(1e-3);
end

for i = 1:length(pwr)
  % generate noise signal
  noise = randn(1, length(n))';
  % normalize signal
  noise_norm = noise / std(noise) * sqrt(pwr(i));

  % sum sines and noise
  sig_noise = sig_norm + noise_norm;

  % generate autocorrelation of noisy signals
  for j = 1:3
    [acor_tmp, lag_tmp] = xcorr(sig_noise(:,j));
    acor(:,j) = acor_tmp;
    lag(:,j) = lag_tmp;
  end

  % plot autocorrelation
  subplot(5,1,i)
  plot(lag, acor)
end
```

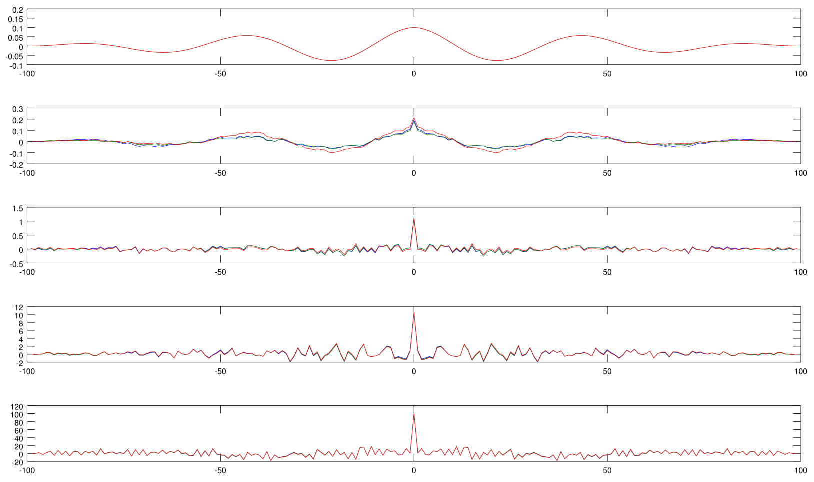

Figure 5: Autocorrelation of noisy sinusoidal signals, at various SNR levels

It can be easily noticed how the periodic peaks in the auto-correlation plots become fainter, as the SNR decreases. In the last two plots, only a major peak at $k = 0$ is visible, showing how the signal is mainly composed of noise and therefore uncorrelated with itself. The distance of the peaks in the first two plots corresponds to the period of the sinusoidal component of the signal.

3 Problem 1.3

3.1 Detection of pulses through cross-correlation

In this section, it will be demonstrated how the `xcorr` function can be used to identify pulses from pulse-train signals with decreasing SNR levels.

The following code generates a random sequence of nonreturn-to-zero pulses between -1 and 1, of given length. Gaussian noise is applied to the pulse-train signal, following a given SNR value. The resulting signal is then compared to a reference pulse between 0 and 1 using the `xcorr` function. A sequence of detected pulses is generated by observing the cross-correlation value at given k intervals, provided by the pulse length. The aforementioned operations are repeated for a selection of SNR levels, and the outcomes are plotted. The chosen SNR values are 10, 0.1, 0.05, and 0.01.

The resulting plot shows the initial sequence of pulses, the noisy signal, its cross-correlation with the reference pulse, and the detected sequence of pulses.

```
clear;
hold off;
clf;
pkg load signal

title({'Detection of pulses through cross-correlation'})
figure(1)

% simulation parameters
pulse_len = 100;
pulse_num = 10;
pulse_dc = 0.5;
snrs = [10 .1 .05 .01];

%generate pulse reference
pulse_ref = zeros(1, pulse_len);
pulse_ref(1:(end*pulse_dc)) = 1;

for i = 1:length(snrs)
  % generate pulse-train signal
  pulses = zeros(1, pulse_num * pulse_len);
  pulses_pos = round(rand(1, pulse_num));
  for j = 1:pulse_num
    pulses((j-1)*pulse_len+1 : j*pulse_len) = pulse_ref * pulses_pos(j
  end
  pulses(pulses==0) = -1;
  % calculate signal power
  pulses_pwr = var(pulses);

  % generate noise signal
  noise = randn(1, pulse_num * pulse_len);
  % normalize signal
  noise = noise / std(noise) * sqrt(pulses_pwr / snrs(i));

  % sum pulse-train and noise
  sig_noise = pulses + noise;

  % generate autocorrelation of noisy signals
  [rxx, lag] = xcorr(sig_noise, pulse_ref);
  % detect pulses:
  % find cross-correlation values for possible pulses
  rxx_pulses = rxx(find(mod(lag, pulse_len) == 0)(pulse_num:end));
  pulses_res = rxx_pulses > 0;

  % plot pulses position
  subplot(length(snrs),4,(i-1)*4+1)
  stem(pulses_pos)
  % plot encoded noisy signal
  subplot(length(snrs),4,(i-1)*4+2)
  plot(sig_noise)
  % plot cross-correlation
  subplot(length(snrs),4,(i-1)*4+3)
  plot(lag(pulse_num*pulse_len:end), rxx(pulse_num*pulse_len:end))
  % plot detected pulses
  subplot(length(snrs),4,i*4)
  stem(pulses_res)
end
```

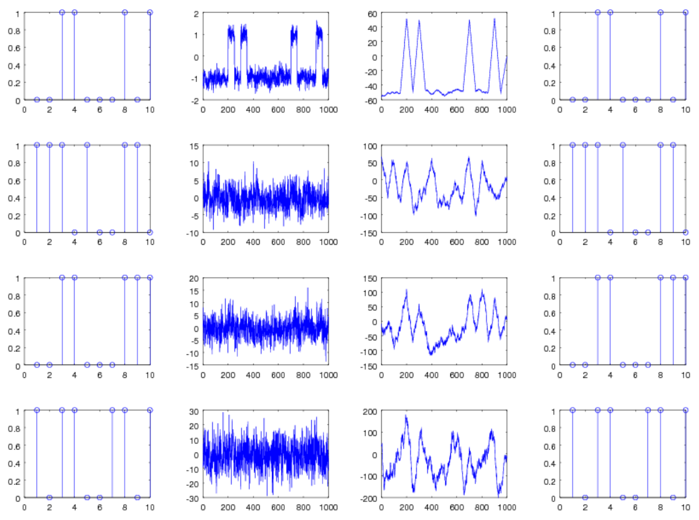

Figure 6: Detection of pulses through cross-correlation, at various SNR levels

The figure above shows how pulses can be successfully extracted from a noisy source using cross-correlation techniques.

The accuracy of this method has been experimentally verified using a slightly modified version of the code above. A series of 25 logarithmically spaced values between 10 and 0.001 (corresponding to a range between -30 and 10 dB) has been chosen as SNR, and the overall number of correctly detected pulses counted over a large number of iterations.

The following plot shows how the percentage of correctly detected pulses is a decreases for small SNR levels, but remains always above 50, which corresponds to pure chance.

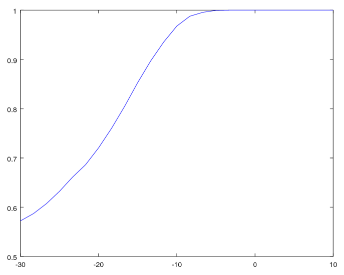

Figure 7: Percentage of successfully detected pulses over a range of SNR between -30 and 10 dB