2-nd practical exercise
BASK signal detection with noise

The purpose of the work is to solve the problem with symbol detection when comparing received signal level with a threshold. Afterwards, BASK modulated signal is studied in the same way.

For this work, there are no individual parameters of system. The report should be prepared and sent to email address: elans.grabs@rtu.lv. The report must include the following:

1. The objective of practical exercise;
2. The full source code of simulation programs;
3. The plots obtained during simulation for different testing scenarios.
4. The conclusion on noise immunity comparison for rectangular pulses and BASK signals.

The tasks to be solved:

1. Save a copy of the first practical exercise and set parameters to work with unipolar rectangular pulses.
2. Reorganize program to draw plots in the same window. Draw the following plots:
    a. Transmitted signal sTx;
    b. Received signal sRx;
    c. Detected signal sD.
3. Perform detection by comparing average symbol level with a threshold. For this purpose, it is necessary to split received signal into segments first.
4. Analyze the accuracy of detection for different SNR values (try 20 dB, 10 dB, 5 dB and -5dB).
5. Increase number of transmitted binary symbols to 10 000. Generate these symbols randomly and add code for counting number of incorrectly received symbols after detection procedure.
6. Perform BASK modulation for a signal by changing carrier signal from constant voltage level to a harmonic signal. Observe the high number of errors when averaging detector is used.
7. Implement correlation detector for BASK signal to improve the accuracy of detection.
8. Analyze the accuracy of detection for different SNR values (try 20 dB, 10 dB, 5 dB and -5dB).
9. Make conclusions on noise immunity of two modulation types: a) unipolar rectangular pulses, b) BASK modulated signal.

The guidelines for practical exercise

1 The last task of the 1-st practical exercise required working with NRZ signal. This introduced two changes into a program:

1. We have converted signal levels from {0, 1} to {+1, -1}, respectively;
2. The threshold has been set to 0 and comparison sign has been changed from “>” to “<”.

In order to return to unipolar pulses, these two changes must be reversed. The first change can be simply commented out (or removed entirely), and for the second change the appropriate code modifications must be made:

```
% sTx = -2*sTx + 1;  % Convert signal to NRZ
…
sD = sRx > Thresh;    % Detecting signal by comparing with a
                      % threshold.
```

Note here, that threshold value has been detected automatically as signal average value, which is:

- 0 for NRZ signal (since levels alternate between +1 and -1);
- 0.5 for unipolar pulses (the half of the "1" symbol amplitude).

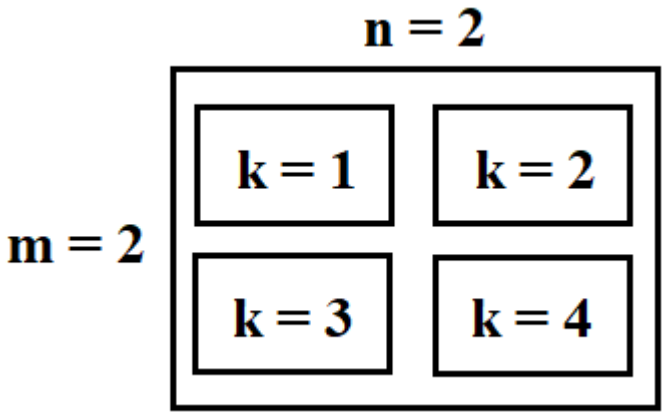
2 In our previous practical work, we were using figure( ) command to create a new window for each plot. In our case, all comparable signals share the same x axis (time), so it would be convenient to align them together by drawing them in the same window in 3 plots one below each other.

Consider, for example, how the plotting has been implemented in 1-st practical work:

```
figure(1), stem(s), grid on
figure(2), plot(sTx), grid on
figure(3), plot(sRx), grid on
figure(4), plot(sD), grid on
```

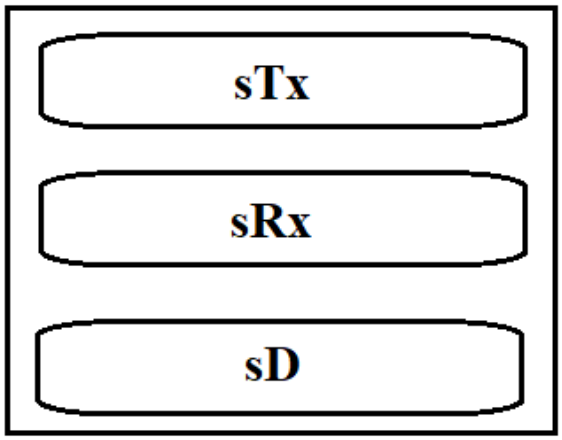
(List 1)

The another possible representation option is to use *subplot(m,n,k)* command before each plotting operation instead of *figure(k)* command. See help command on subplot for more details and illustration below for numbering of plots.



Here **m** is the number of rows (2 in example above),
n is the number of columns (2 in example above);
k is the number of each figure (from 1 to 4 in example above).

For our purpose, the plots will be organized as follows (m = 3, n = 1, k = 1 .. 3):

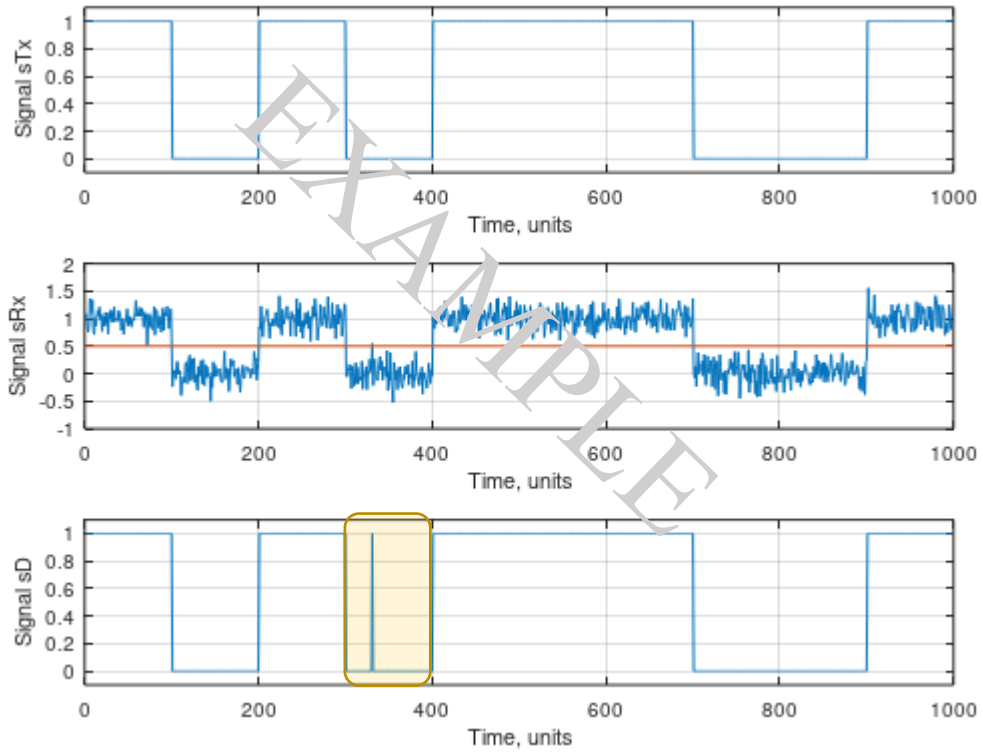


Then, the code List 1 (see above) will be changed as follows:

```
figure(1), stem(s), grid on
subplot(3,1,1), plot(sTx), grid on
subplot(3,1,2), plot(sRx), grid on
subplot(3,1,3), plot(sD), grid on
```

(List 2)

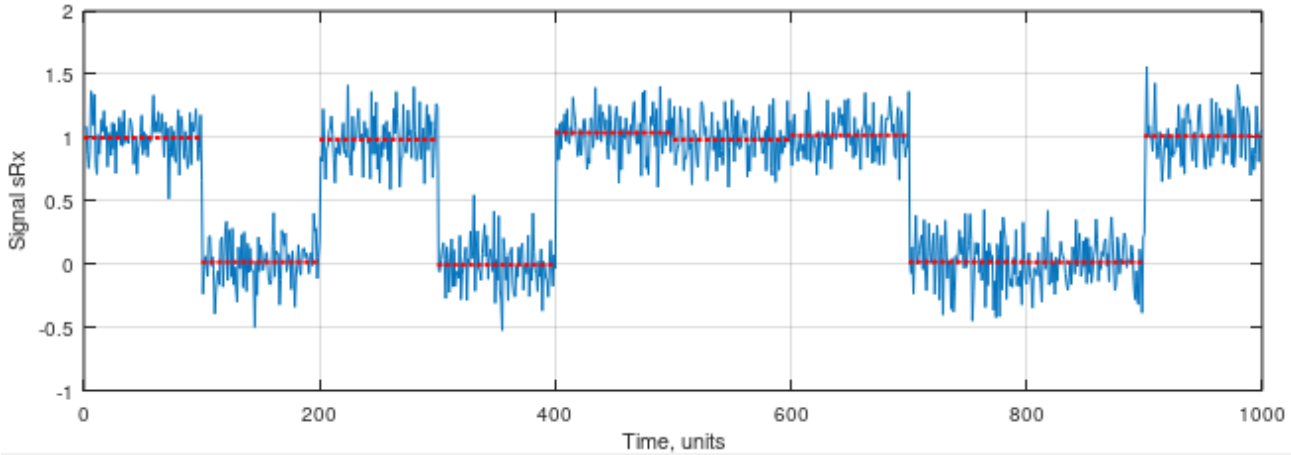
Note, that we are simply skipping the first *stem( )* graph, since it shows the same values as sTx plot, and thus is redundant. The resulting window of plots should look like picture below:



EXAMPLE

3 As it has been observed by analyzing detection accuracy for different SNR values, there are certain points, where detection has been performed incorrectly due to too strong noise. Example of such occurrence is shown in 3-rd plot in figure above (Signal sD).

To improve accuracy of detection, an average signal value can be evaluated over each symbol duration, as is shown in example figure below with a red dotted line:



This average value can be compared with a threshold instead to give more accurate results.

The idea of the averaging is based on additive nature of the noise, which is added to a signal:

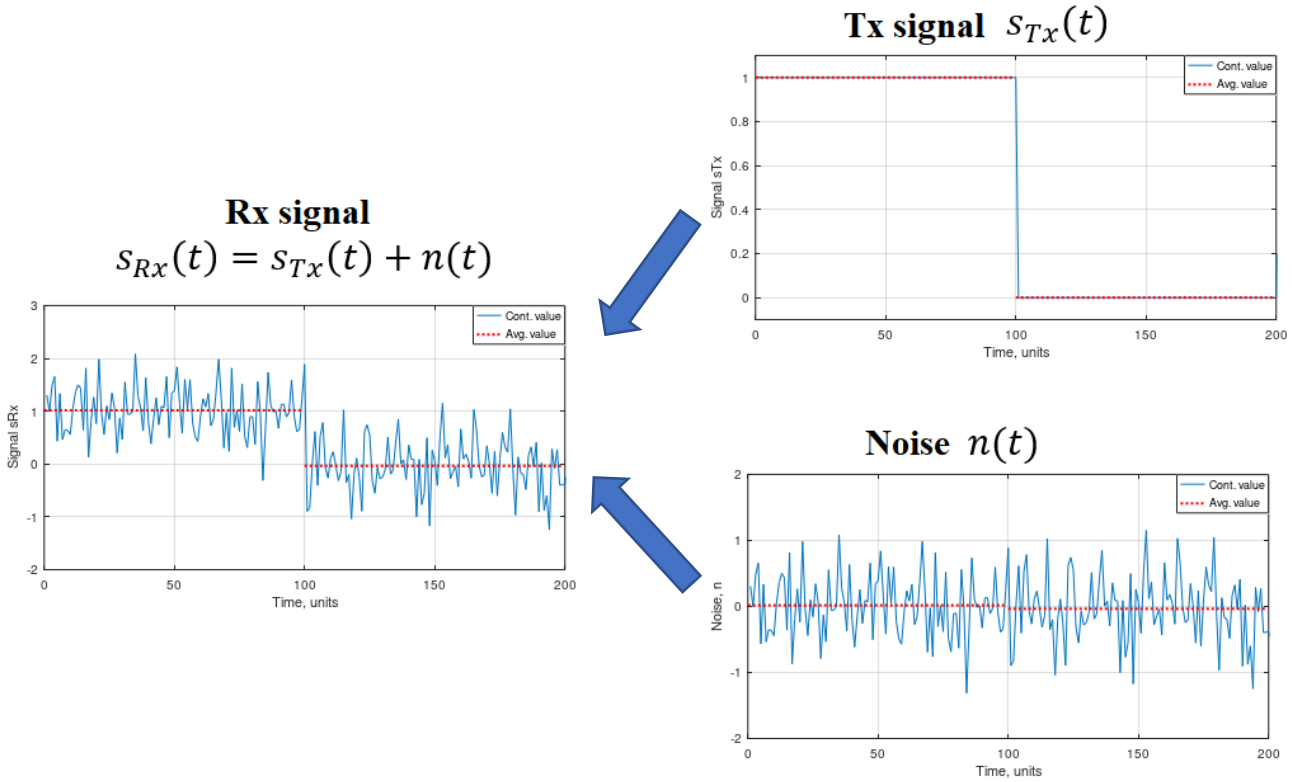
$$s_{Rx}(t) = s_{Tx}(t) + n(t)$$

In this case, the average symbol value (mathematical expectation) of received signal can be also expressed as the sum:

$$E[s_{Rx}(t)] = E[s_{Tx}(t)] + E[n(t)],$$

where it is known, that $E[n(t)] \to 0$. This enables eliminating out the noise from calculations. In practice, however, mostly due to filtering the average value of noise will be greater, than 0. Also, in order to achieve 0 average value, an infinitely long noise process should be considered.

This superposition principle illustration is shown below, where You can see how received signal is described as a sum of noise and transmitted signal.



For this approach to work, the signal must be first segmented into separate symbols, for which average value must be calculated. In Octave/Matlab this signal is represented by a sequence of numerical values, with a total length of 1000 numerical values. There are 10 symbols transmitted in total (defined by us at variable vector *s*):

```
s = [1, 0, 1, 0, 1, 1, 1, 0, 0, 1];   % Digital signal
```

So, each symbol is represented by 1000/10 = 100 samples. This is the value we specified for signal replication via Kroneker multiplication:

```
sTx = kron(s,ones(1,100)); % Create "analog" signal, by sampling each
                           % symbol for 100 samples
```

For calculating each symbol's average value, the matrix of symbol samples should be formed first:

Signal vector	➔	Segmented by symbols
1 x 1000 vector		100 x 10 matrix

Here each column contains 100 samples (with noise) for each of 10 symbols.
Such a conversion can be achieved by using reshape(x,M,N) function, where

- x is initial matrix or vector (in our case, it is sRx vector);
- M is the number of rows of the new matrix (in our case, M = 100),
- N is the number of columns in the new matrix (in our case N = 10).

For easier further modifications, let us reorganize program first, by specifying carrier signal explicitly as a variable. This is performed right before Kroneker multiplication.

```
sTx = kron(s,ones(1,100));
```
(List 3)

After the modifications, the code fragment in List 3 should look like this:

```
N = length(s);      % Number of symbols (count s elements)
M = 100;            % Samples per symbol (we choose)
s0 = ones(1,M);     % Carrier signal (constant level)
sTx = kron(s,s0);   % Upsampled (analog) message signal
```
(List 4)

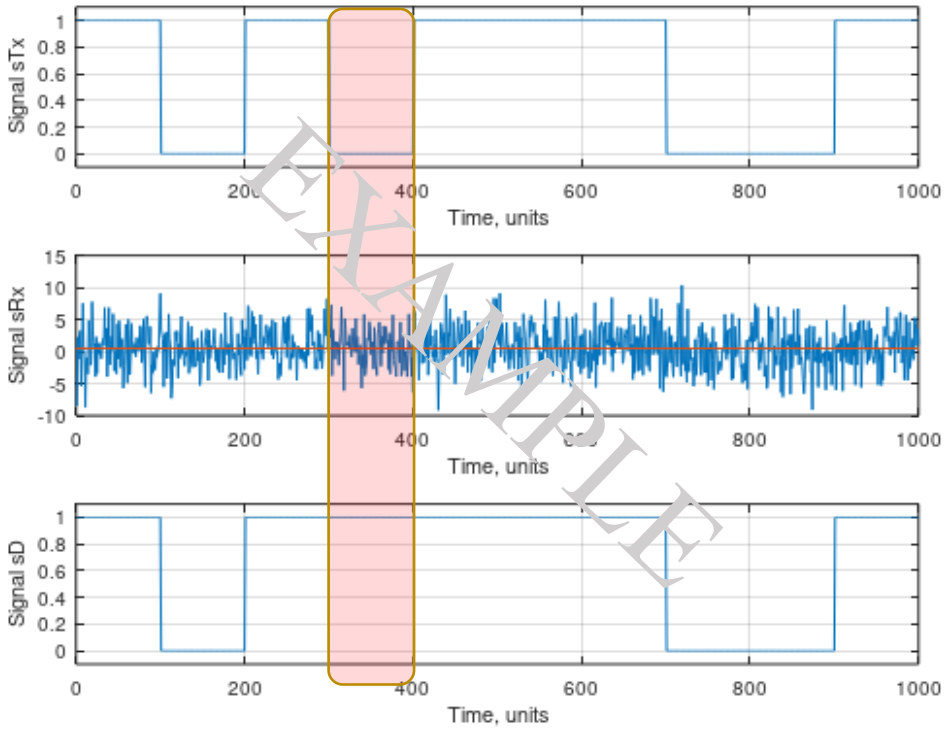
This way, the matrix reshape can be performed right before detection process as follows. Note, that during detection process we compare the average value of all samples (each column):

```
sRx = reshape(sRx,M,N);
sD = mean(sRx) > Thresh;
```
(List 5)

To draw the plot of detected signal sD in the same scale as other signals, the Kroneker multiplication can be used to upsample the signal by a factor of M = 100.

```
sDvec = kron(sD,ones(1,M));        % Upsample detected data
subplot(3,1,3), plot(sDvec), grid on
```

The resulting 3 plots should look like shown below:



EXAMPLE

4 Please, try different SNR values to observe, how noise power can lead to errors during detection. An example of such an error is marked with red color in figure above. Note, that in case of incorrect detection, an entire symbol is received incorrectly (rather than its specific samples, as it was before averaging).
Note: You may need to run program multiple times for the error to occur due to random noise nature.

5

As You may have noticed during task 4, multiple runs of the program can yield different results (number of incorrectly received symbols). From probability theory and statistics, it is well known, that the number of experiments should be as big, as it is possible.

Let us increase the number of generated symbols from N = 10 to N = 10 000. This time, however, let computer generate "1" and "0" values randomly. This can be achieved by using *randi(I_{max},m,n)* function, which generates random integer numbers. Here:

- I_{max} is the maximum possible generated number (from 1 to I_{max});
- *m* is the number of rows in generated matrix;
- *n* is the number of columns in generated matrix.

In our case, we will generate a vector-row consisting of N = 10 000 elements. This will update the code snippet below:

```
s = [1, 0, 1, 0, 1, 1, 1, 0, 0, 1];   % Digital signal
```
(List 6)

to more complete version:

```
N = 10000;              % Number of symbols to transmit
s = randi(2,1,N) - 1; % Generate binary symbols vector
...
N = length(s);        % Number of symbols (count s elements)
```
(List 7)

Note, that this time, we define N manually as 10 000, so there is no need to redefine this value before Kroneker multiplication (it will not be changed anyway). Also, please note, that randi(2,1,N) function generates integer values from 1 to 2, instead of desired result ranging from 0 to 1. That is why we subtract 1 from this function result.

Important:

Please note, that increase in number of symbols will significantly increase plot drawing time. Therefore, it is much advisable to limit output of each plot to some number of the symbols, for example, 25 first symbols. This can be achieved by modifying code (List 2) as follows:

```
subplot(3,1,1), plot(sTx(1:25*M)), grid on
subplot(3,1,2), plot(sRx(1:25*M)), grid on
sDvec = kron(sD,ones(1,M);
subplot(3,1,3), plot(sDvec(1:25*M)), grid on

% Add Threshold line and limit to first 25 symbols
hold on, plot([0 25*M],[Thresh Thresh]), hold off
```
(List 8)

At the end of the code, error counting should be added. Simply speaking, sD should be compared to match s:

```
>> s(1:25)
ans =

   1   1   1   0   1   1   1   0   0   1   0   1   0   0   1   1   1   1   1   0   1   0   1   1   0

>> sD(1:25)
ans =

   1   1   1   0   1   1   1   0   0   1   0   1   0   0   1   1   1   1   1   0   1   0   1   1   0
```

To do it programmatically, the relational operators ("==", "~=", "<", ">", etc.) can be used. Here we need to find any mismatches, therefore operator "~=" (not equal to) should be used.

```
ErrVec = (sD ~= s);          % Comparing recv. data with generated
```

This will generate a vector, consisting of "1" elements at each mismatch location:

```
1  1  1  0  1  1  1  0  0  1  0  1  0  0  1  1  1  1  1  0  1  0  1  1  0  s
1  1  1  0  0  1  1  0  0  1  0  1  0  0  1  1  1  1  1  0  1  0  1  1  0  sD
0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  ErrVec
```

All that is left – to count the number of "1" elements in ErrVec vector. This can be performed by adding up all the numbers of this vector together:

```
NumErr = sum(ErrVec)        % Number of errors
```

Note, that there is no semi-colon at the end of the command, so the result is output at the Octave console:

```
Command Window
NumErr = 0
>>
```

6Let us study modulation scheme with a carrier harmonic signal. The simplest modulation type is ***Binary Amplitude Shift Keying*** (***BASK***). As it has been pointed out during lecture, this modulation type simply generates "ON-OFF" type harmonic signal. The possible signal values are:

- $\sin(2\pi f_0 t)$ if "1" is transmitted;
- 0 if "0" is transmitted.

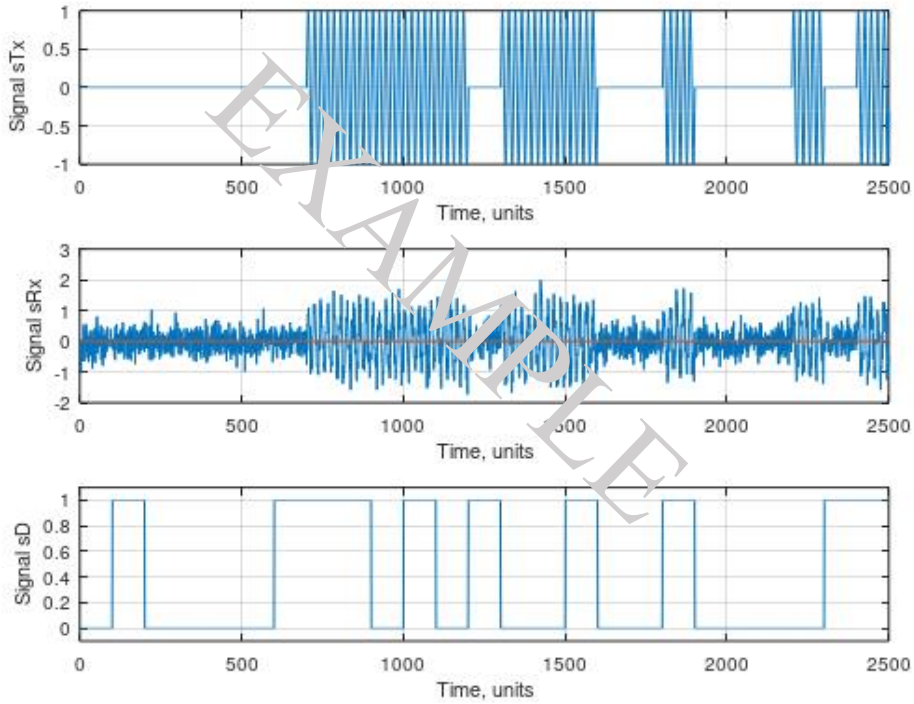
The program we have written so far can be altered by specifying a different carrier signal s0 in (List 4):

```
M = 100;              % Samples per symbol (we choose)
s0 = sin(2*pi*(0:M-1)/20);  % Harmonic carrier signal
sTx = kron(s,s0);     % Upsampled (analog) message signal
```

(List 4)

Here 20 is the number of full periods per each "1" or "0" symbol. Afterwards, this harmonic signal is multiplied either with "1" ("ON" phase) or with "0" ("OFF" phase).

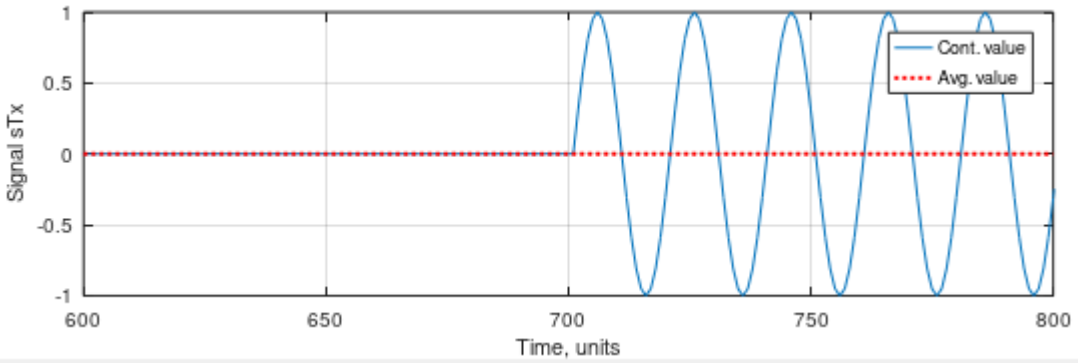
The resulting plots should look like these:



EXAMPLE

Note the high error count:
NumErr = 5012 (more than 50%!).

7 The results obtained in previous task are not depending on SNR value, i.e. even for good channels there are many detection errors. In fact, this is caused by average value of harmonic signal, which is zero. This way, there is no possible option to compare average value of harmonic signal ("1" symbol) and average value of zero voltage level ("0" symbol).



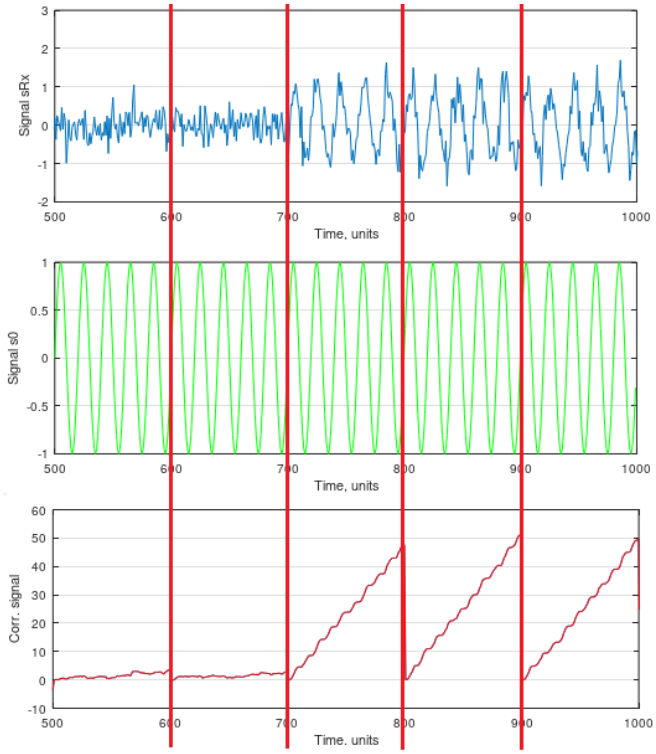
The solution is to use entirely different detection scheme – correlation receiver, which calculates ***correlation integral*** value:

$$s_D(t) = \int_{t_1}^{t_2} s_{Rx}(t) \cdot s_0(t) dt, \quad (1)$$

where:

- $s_{Rx}(t)$ is noisy received signal;
- $s_0(t)$ is "pattern" signal (pure carrier signal);
- t_1 and t_2 are symbol start and end times, accordingly.

The correlation receiver "compares" each received symbol with a pattern. The higher is similarity, the greater correlation integral value will be:



Noisy signal

Carrier signal

Integrated value

There are two possible scenarios, if no noise is present in signal:

1. The signal is “1”, i.e. harmonic signal received. Then the result of integration will be signal **energy**;
2. The signal is “0”, i.e. zero voltage received. Then the result of integration will be **zero**.

This, accordingly, gives us a new threshold value of the half of energy:

```
Thresh = sum(s0.*s0)/2;    % BASK threshold, half of carrier energy
```

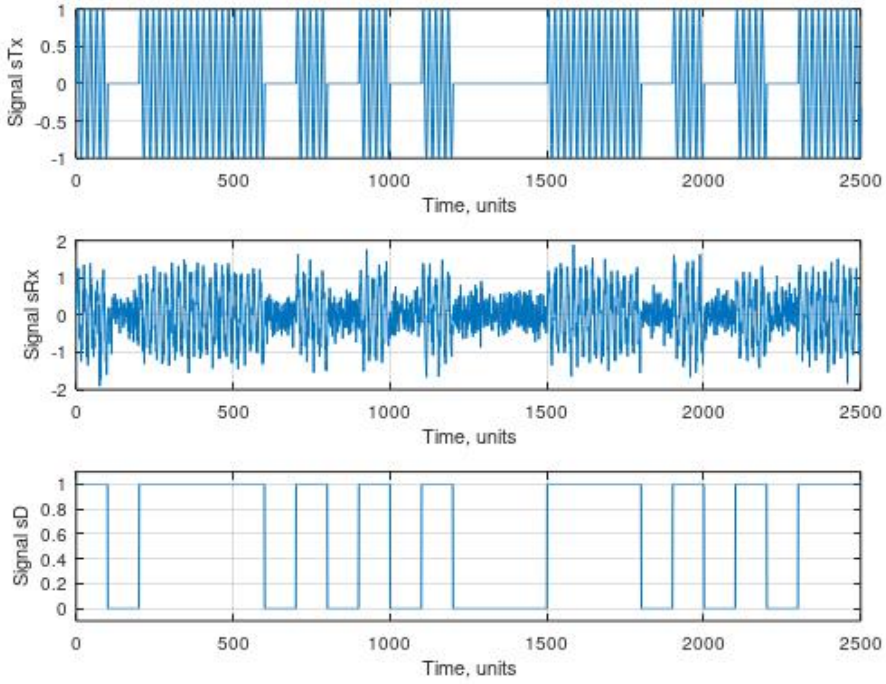
Note, that for discrete model integration is replaced with a finite sum( ).
Likewise, in case of discrete process (which is in Octave memory), the expression (1) changes into:

$$s_D[M] = \sum_{m=0}^{M-1} s_{Rx}[m] \cdot s_0[m] \qquad (2)$$

This means, that detector must be changed accordingly. We already have converted sRx signal into MxN matrix by using reshape. Each symbol is represented as a column of this matrix. Now, we multiply each column by a vector row s0, and thus calculate (2) for each symbol:

```
sD = (s0*sRx) > Thresh;     % Calculate BASK signal correlation for
                            % each symbol-column
```

Now, the detection is performed correctly, and the number of errors depend on SNR value:



8 Please, try different SNR values to observe, how noise power can lead to errors during detection. Note, that there is a strict relation of number of errors to SNR level. This will be further discussed in future lectures. For now, summarize in table the number of errors for multiple SNR values.