Lab Memo for

TSKS02 Telecommunication

Mikael Olofsson

Department of Electrical Engineering (ISY)
Linköping University, SE-581 83 Linköping, Sweden

Autumn 2014

Note:
This lab memo is intended for the course TSKS02 Telecommunication that is given for a multitude of students at Linköping University. Some are electrical engineering Bachelor students, others are engineering Master students not primarily into communication, and yet others are computer science Master students. It is a translation of the Swedish lab memo used in a previous similar course TSEI67 Telecommunication. That in turn was a complete rewrite of various lab instructions used earlier in that course. Many models and signals that are used are the same as before, and previous course responsible and lab responsible have therefore contributed to these labs. Primarily, that is Niclas Wadströmer and Magnus Öberg.

Lab memo for TSKS02 Telecommunication

Department of Electrical Engineering (ISY)
Linköping University
SE-581 83 Linköping
Sweden

This document is written using LaTeX2$_\varepsilon$.

Introduction

The goal of these laboratory exercises is to provide a feeling of how you can empirically analyze telecommunication systems. The tool that is used is the graphical simulation tool Simulink which is a part of Matlab. With this tool, you can create systems using building blocks and set parameters in those blocks. Finally, you can simulate your system with whatever signal you choose.

There are three laboratory exercises in this course, namely

1. Analog modulation. This exercise is also intended to introduce Simulink.
2. Digital modulation.
3. Coding methods.

All these laboratory exercises are scheduled as four-hour sessions and are supposed to be done in pairs. You should be able to finish each exercise in the scheduled four hours. You pass each exercise by reporting your findings to your lab teacher before leaving the lab session. If you are not done within those four hours, you have to finish it by yourselves and then hand in a brief written report to your lab teacher. The laboratory exercises are reported to Ladok as LAB1 when you have passed all three exercises.

Not much theory is presented in this lab memo. The needed related theory is presented in the course book.

It is probably a good idea to create a new folder on the computer for each lab. There are certain files that you need for these laboratory exercises, and they can be downloaded from the course homepage at

`http://www.commsys.isy.liu.se/TSKS02`

under "Laboratory exercises".

Laboratory Exercise 1 Analog Modulation

1.1 Preparation

Read up on analog modulation methods. Make sure that you know the differences between the various versions of amplitude modulation and angle modulation.

1.2 Introduction

Simulink makes it possible to build different types of systems using block diagrams, and to simulate them. In our case we will be building communication systems. The initial part of this laboratory exercise is about getting used to the Simulink interface by doing some simple exercises. Then we will move on to study amplitude and angle modulation. The instructions in the introduction are very detailed, but they will get more brief further on in this memo.

Download the following files from the course web to your lab folder:

```
sound1.mat
amradio1.mdl
amradio2.mdl
fmradio1.mdl
fmradio2.mdl
lab1blocks.mdl
```

1.3 Introduction to Simulink

Begin by launching the Matlab application. If you are on a Windows machine, you do that as usual via the main menu. If you are on a Linux machine, you first start a terminal and then type `matlab` in that terminal. There is a file browser to the left. Use that to navigate to your lab folder. All files you create will end up there.

You can also use the Unix-inspired command `cd` to change directory. You can use the command `pwd` to find out where you are, and the command `ls` to list the contents of the current folder.

Start Simulink

Write the following in Matlab's command window:

```
simulink
```

This starts Simulink and opens the Simulink Library Browser. All Simulink blocks are available via this window.

Building a Model

You are now going to build a simple system and perform a few simple measurements on that system. In the window that was just opened, choose

```
File > New > Model
```

This gives you a new empty window, which is where you will build your model. It is now a good idea to give your model a name by saving it using

```
File > Save As
```

First you need a source that generates a signal. Click on the library `Sources` in the Simulink Library Browser. A new tab window entitled `Library: simulink/Sources` should pop up. Drag the source `Sine Wave` to your model. This creates a copy that you can edit by double-clicking on it. Do that and choose

```
Sine type:   Time based
Amplitude:   1
Bias:        0          - This is the DC component.
Frequency:   100        - Note that this is angular frequency (rad/s)
Phase:       0          - Phase in radians.
Sample time: 0.005      - Sample period
```

Here it is obvious that the simulation is time-discrete; a sample period is mentioned. It is important to make sure that the sampling theorem is fulfilled: The sampling frequency has to be at least twice the largest frequency component of the signal. Sometimes you need a larger marginal than that if there are non-linear blocks involved or modulation will be applied. Such blocks can spread the spectrum of the signal, and the sampling theorem has to hold for all signals in the system.

Answer the following:

- What is the sampling frequency in this case?
- What is the largest frequency component of the signal?
- Is the sampling theorem fulfilled?

There is extensive help available for all blocks, in some cases even too extensive for most uses. Right-click on a block and select `Help`, and Matlab's help window opens with the help text for that block.

Connecting and Using Measurement Instruments

Now that we have a signal, it would be nice to be able to look at it. For that you need an oscilloscope, which of course is available in Simulink. Click on `DSP System Toolbox` in the Simulink Library Browser and open `Sinks` in that tab. Notice that DSP stands for Digital signal processing. Drag `Time Scope` (oscilloscope) from there to your model. Also drag `Spectrum Scope` (frequency analyzer) to your model. That is an instrument that performs DFT (Discrete Fourier Transform) on a signal and shows its amplitude spectrum (absolute value) using a dB scale. Connecting a source (e.g., the sine wave) to the oscilloscope is done by clicking on the output of the source and dragging it to the input of the oscilloscope. To connect the frequency analyzer, drag its input to the newly created connection. A tiny black dot indicates that the connection is successful.

You do not need to make any adjustments on the oscilloscope right now, but you need to open the frequency analyzer and choose `Buffer input`. While, you are there, also select `Specify FFT length` and set `FFT length` to 1024.

Simulation

Before you simulate this simple model, it is a good idea to control the simulation parameters. Choose

```
Simulation > Model Configuration Parameters
```

in your model window. At present there's not much to do here, but make sure that the starting point is 0 seconds and that the end point is 10 seconds. Then close the dialog. This is the time in the reality that we are simulating. Note that the actual simulation can take more or less time. Finally, you start the simulation by choosing

```
Simulation > Run
```

The computer emits a beep when the simulation is done. The oscilloscope and frequency analyzer will show its results in different figures. If these do not pop up automatically, you can double-click on the respective block to open them. To get a reasonable graph from the frequency analyzer, you might need to open it and change the axes (e.g., there is an autoscale).

Answer the following:

- What is the frequency (in Hz) of the signal?
- How can we see that in the oscilloscope?
- How can we see that in the frequency analyzer?

Feel free to experiment with the settings of the frequency analyzer. Also experiment some with the settings of the oscilloscope, via its open window. At least check which different settings, statistics, and measurements there are.

So far, there has not been much of a system in your model. Therefore, open `Math Operators` in the Simulink Library Browser, and drag `Math Function` to your model. This block will be your system. It is a block that specifies a momentary relation between its input and its output. The default relation is `exp`, which means that we have the output $y[n] = e^{x[n]}$, where $x[n]$ is the input. Connect the input of your system to the signal from the source, while keeping the already existing connections. Set `Number of Input Ports` of the oscilloscope settings to 2. This results in a second input to the oscilloscope. Connect the output of the system to that input.

You will now need another frequency analyzer. Right-click on your frequency analyzer and choose `Copy`. Then right-click on the back-ground beside and choose `Paste`. This does not only give you a new frequency analyzer, but one that has the settings of the original as initial settings. Connect the output of your system to the input of this frequency analyzer. Now you have two frequency analyzers. It can be a good idea to separate them by changing their names. You do that by clicking in the text below the block and edit it.

Simulate the model and answer the following:

- What can be said about the output?
- What can be said about the spectrum of the output?

Feel free to experiment with different functions in the system. You do that by first double-clicking on it and re-simulate after you have chosen a new function and closed the dialog. When you are done experimenting enough, reset the function to `exp`.

Measuring Power

Often we are interested in the power of a signal, and then we mean the signal power and not the physical concept of power. The momentary power of a time-discrete signal, $s[n]$, is $s^2[n]$. The signal power is then the average of that. We calculate that as

$$P = \frac{1}{N}\sum_{n=1}^{N} s^2[n]$$

for some relatively large integer N. The variance, σ^2, of a signal is then the power of the signals after we have subtracted its mean

$$m = \frac{1}{N}\sum_{n=1}^{N} s[n].$$

The variance is calculated as

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N} s^2[n] - m^2.$$

There are blocks that calculate the mean and the variance. Then we get the power as

$$P = m^2 + \sigma^2.$$

The two mentioned blocks are `Mean` and `Variance`. Those can be found by opening `Statistics` in the `DSP System Toolbox`. Connect these to the output of the system. Open both and select `Running mean` and `Running variance`, respectively. Finally, in `Sinks` of the `DSP System Toolbox` you find the block `Display`. We need two of those, one each to the blocks `Mean` and `Variance`.

Answer the following:

- What is the mean of the output?
- What is the variance of the output?
- What is the power of the output?

Comment

By now you have probably realized that there is a lot of clicking to get the blocks that you need in a certain model. Therefore we have prepared Simulink models for each laboratory exercise with suitable blocks for those labs. You downloaded those that are needed for this exercise in the beginning.

1.4 Amplitude Modulation

Keep the model that you have just built. You may very well have use of blocks from it later in this exercise. The model `lab1blocks.mdl` that you downloaded initially also contains some useful blocks for this laboratory exercise.

Load the file `sound1.mat` into Matlab, either by double-clicking on it in the file-browser to the left in the Matlab window, or by typing

```
load sound1
```

at Matlab's command window. This gives you the variable `sound1` containing sound, sampled with the sampling frequency 80 kHz. You can listen to the sound by typing

```
soundsc(decimate(sound1,2),40000)
```

at the command window. The computers in the computer rooms might not have loud-speakers connected to them. If you have headphones to your cell phone with an ordinary 3.5mm connector, you are encouraged to connect them.

Then open the model `amradio1.mdl` by double-clicking on it. This model uses the sound in `sound1` as input. Use a frequency analyzer with

```
FFT length                  1024
Number of spectral averages  600
Buffer size                 1024
```

and other measuring blocks to answer the following about the input:

- What is the bandwidth of the signal?
- What is the power of the signal?

Answer the following about the signal that is sent over the channel:

- What modulation form is used?
- What carrier frequency is used?
- What is the bandwidth and power of the signal?

Answer the following about the received signal:

- What does the spectrum look like? Compare to the sent signal.
- What is the variance of the noise? Both according to settings and measured.

The output ends up in the variable `yout`. Again, you can listen to it by typing

```
soundsc(decimate(yout,2),40000)
```

Answer the following about the output:

- What does the spectrum look like? Compare to the input.
- What does the signal sound like? Compare to the input.
- What does the noise spectrum look like? Hint: Set the initial amplification to 0. It is perfectly possible to do that since we are dealing with a linear modulation form.

1.5 Angle Modulation

You are now supposed to do the same thing as above, but now for the model `fmradio1.mdl`. Answer the following about the signal that is sent over the channel:

- What modulation form is used?
- What carrier frequency is used?
- What is the bandwidth and power of the signal?

Answer the following about the received signal:

- What does the spectrum look like? Compare to the sent signal.
- What is the variance of the noise? Both according to settings and measured.

Answer the following about the output:

- What does the spectrum look like? Compare to the input.
- What does the signal sound like? Compare to the input.

This is a non-linear modulation form. Therefore, the resulting noise is different in different situations. You get the noise by subtracting the input from the output. The system delays the input 66 samples, which means that you need to delay the input equally much before subtracting it, to make a comparison fair. We shall now study the noise spectrum for some different settings for the two amplifiers in the system to shed some light on the effects of the non-linear property.

- What does the noise spectrum look like when both amplifications are 1?
- What does the noise spectrum look like when the first amplification is 0 and the second is 1?
- What does the noise spectrum look like when both amplifications are 0?

1.6 If you have time

Repeat Sections 1.4 and 1.5, but use the models `amradio2.mdl` and `fmradio2.mdl` instead.

- Find out how these models differ from the first ones.
- How does that affect the communication?

1.7 If you have even more time

Build an envelope detector and use measurements to show that it works as intended. The description in the compendium uses a diode that charges a capacitor, that is discharged through a resistance. Alternatively, you can rectify the signal followed by an LP filter. Use this alternative approach.

- Does this envelope detector behave differently in any way compared to the description in the compendium?

Laboratory Exercise 2 Digital Modulation

2.1 Preparation

Read up on eye patterns in Section 7.6 in the course book, (i.e., Olofsson, Telecommunication Methods).

Draw the following signals with graded axes and calculate the their signal energies.

$$x_1(t) = \begin{cases} A, & 0 \le t < T, \\ 0, & \text{elsewhere.} \end{cases} \qquad x_3(t) = \begin{cases} A\frac{t}{T}, & 0 \le t \le T, \\ 0, & \text{elsewhere.} \end{cases}$$

$$x_2(t) = \begin{cases} A, & 0 \le t < T/2, \\ -A, & T/2 \le t < T, \\ 0, & \text{elsewhere.} \end{cases} \qquad x_4(t) = \begin{cases} 2A\frac{t}{T}, & 0 \le t < T/2, \\ 2A\frac{T-t}{T}, & T/2 \le t < T, \\ 0, & \text{elsewhere.} \end{cases}$$

Express the signal energies in A and T. The signal energy E of a signal $x(t)$ is given by

$$E = \int_{-\infty}^{\infty} x^2(t)\, dt.$$

2.2 Introduction

Start Matlab and Simulink as the last time and download the following files from the course homepage to your lab folder:

```
lab2blocks.mdl     testmodel.mdl     TSKS0102.mdl
linecodes.mdl      ASK.mdl
```

Type the following at the Matlab prompt to avoid some annoying warnings during this laboratory exercise:

```
warning('off','commblks:commObsoleteBlockLibrary')
```

2.3 Introductory Study – Line Codes

In the Matlab command window, set `Rb` to 1000 and `Spb` to 20. These are parameters that affect several blocks in the models that you will use in this laboratory exercise. `Rb` is bit-rate, (i.e., the number of bits per second), while `Spb` is the number of samples per bit.

Open the model `linecodes.mdl`. It contains a random source that generates bits with probability 1/2 for both 0 and 1. These bits are modulated using six different line codes:

`Unipolar_NRZ`	`Polar_NRZ`	`Manchester`
`Unipolar_RZ`	`Bipolar_RZ`	`Triangular`

Here, `RZ` stands for Return-to-Zero, while `NRZ` stands for Non-Return-to-Zero. Answer the following:

- How are 0 and 1 represented by the different line codes?
- Which of the line codes have a non-zero DC component? How can you see that?
- If we define the bandwidth as the frequency where the first zero occurs in the spectrum, what are the bandwidths of the line codes in this example? Try to explain why they have different bandwidths.

To make the following comparison of the line codes fair, we want them to have the same average signal energy.

- Determine the average signal energy of `Polar_NRZ` in the example above.
- How much do you need to amplify the other line codes to get the same average signal energy as for `Polar_NRZ`?

These amplifications should be used whenever you use a line code in this laboratory exercise.

⧏⧐ **Before you continue, let your lab teacher see your numbers.**

2.4 Test Model

Open the model `testmodel.mdl` and set the values of the variables `Rb` and `Spb` to 1000 and 100 respectively. This model is a communication system that that uses a line code to communicate over an AWGN channel. Initially we let the variance of the channel be 0, to study the eye patterns of the line codes. The receiver consists of a filter that is matched to the chosen line code, followed by a decision block (multidetector) that compares the value of the signal at the sample time with a threshold.

- What do the eye patterns show?

Simulate the system with noise variance 0.1 and then with noise variance 1.

- How are the eye patterns changed?
- Why does the receiver base its decision on the output from the matched filter?
- The multidetector uses the threshold 0 in this case. Why?

Comment: The delays at the inputs of the eye pattern blocks are there so that the eye patterns are presented in the same way as they are described in the course material.

2.5 Syncronization Errors

Set the channel variance to 10. The multidetector has a parameter called *sampling instant*, with which you can set the sampling time. This sampling is rounded to the time resolution that is used. Try therefore to change the sampling time τ to `-k/(Rb*Spb)` for different values of `k` and fill in the table below.

Note: The sampling time τ should be given in the interval `-1/Rb` $< \tau \leq 0$ and is rounded to the time resolution `1/(Rb*Spb)`. If the given sampling time is outside of this interval, then it is reduced into this interval modulo `1/Rb`.

Measured error rate in %

k	**0**	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
`Polar_NRZ`											

The ideal sampling time actually corresponds to $k = 1$. Therefore, use that value henceforth.

2.6 Error Probabilities for Line Codes

We are now going to compare the different line codes. Simulate the system with different noise variance and different line codes. Note that you need to use the correct receiver filter for the different line codes. All necessary blocks are in the model `lab2blocks.mdl`. Do not forget to use the correct amplification for the different line codes.

Before you start filling in the table below, you need to contemplate about the following for each of the line codes:

- What threshold should be used by the receiver? Hint: Simulate the system with noise variance 0 and study the eye pattern for the output of the filter.

And specifically for `Bipolar_RZ`:

- The detector of the receiver has to be modified somewhat. How and why? Hints: Study the eye pattern. How are 0 and 1 represented?

⊲⊳ **Let your lab teacher check your numbers before you continue.**

Measure error rate in %

Noise variance	**5**	**10**	**15**	**20**	**25**
`Unipolar_NRZ`					
`Unipolar_RZ`					
`Polar_NRZ`					
`Bipolar_RZ`					
`Manchester`					
`Triangular`					

You should be able to identify certain differences and similarities in the table.

- What is the explanation to those differences and similarities?
- What properties, other than low error rate, could be desireable for line codes? Where do the considered linecodes stand in that comparison?

2.7 Non-Binary Baseband Modulation

Open the model `ASK.mdl`, which contains a communication system using 4-ASK. First you will study the spectrum of this modulation. For that, we use the parameters `Rb=1000` and `Spb=20`.

To start with, you will study this signalling in the baseband, i.e. the included AM-SC modulation is not to be used. You achieve that by setting the angle frequency of the sine generator to 0, and its phase to `pi/2`.

Make sure that you understand what the different parts in the system do, and how it all works.

- What bit patterns correspond to the four possible signals?
- Determine the amplification needed in the sender amplifier so that this modulation uses the same bit-energy as the binary linecodes that you studied before.
- What thresholds should be used in the detectors of the receiver?
- Using the same bandwidth definition as before, what is the bandwidth of this modulation form?

◁▷ **Let your lab teacher check your numbers before you proceed.**

Time to measure error rates for different channel noise variances. Use the parameters `Rb=1000` and `Spb=100` for that.

Measured error rate in %

Noise variance	**5**	**10**	**15**	**20**	**25**
`4-ASK`					

Now answer this:

- Compare the error rates with the different line codes. Better or worse?
- Does this modulation form have an obvious advantage compared to the line codes?

2.8 Non-Binary Passband Modulation

Usually signals are not sent in the baseband. Instead they are modulated to a passband using AM-SC. Now set the angle frequency of the sine generator to `5*Rb*2*pi`. Use the parameters `Rb=1000` and `Spb=20`.

- What carrier frequency does that correspond to?
- What amplification is now needed in the sender amplifier to maintain the previous bit energy?
- What thresholds should then be used in the detectors of the receiver?
- Essentially the same bandwidth definition as before: The width of the main lobe. What is the bandwidth of this modulation form?

⧏⧐ **Let your lab teacher check your numbers before you continue.**

Error rates again. Use the parameters `Rb=1000` and `Spb=100`.

Measured error rate in %

Noise variance	**5**	**10**	**15**	**20**	**25**
`4-ASK`					

Now answer this:

- Compare this to the baseband case above. Better or worse?

2.9 If you have time left

Based on the ASK system, build a communication system that uses 16-QAM.

Hint: 16-QAM can be seen as a two-dimensional variant of 4-ASK, where 4-ASK is used to modulate each of the two dimensions. The two basis functions have to be orthogonal. Time-limited $\cos(2\pi f_0 t)$ and $\sin(2\pi f_0 t)$ are orthogonal if the frequency f_0 is chosen correctly. – How?

- If the bit-energy is the same, how does 16-QAM compare to 4-ASK?

Laboratory Exercise 3 Coding Methods

Start Matlab and download the following files from the course homepage to your lab folder:

```
lab3blocks.mdl
uncoded.mdl
repetitioncode.mdl
hammingcode.mdl
planet.mat
```

This time, you will primarily measure error rates for error control codes. There is also a short study on Huffman codes.

3.1 Uncoded Communication over a Binary Symmetric Channel

First you get a stripped-down variant of the measuring setup where no error control is used. You find that in the model `uncoded.mdl` and the communication takes place over a binary symmetric channel. The settings of the model results in the simulation of 10^5 sent bits.

Uncoded: Measured BER in %

Channel Error Probability	0.1%	0.3%	1%	3%	10%
`Measured error rate`					

The measured error rate should be close to the chosen error probability of the channel. If not, then something is wrong.

3.2 Repetition Codes

The model `repetitioncode.mdl` contains a communication system that uses a repetition code to communicate over a binary symmetric channel. All blocks in this model use the variable `n` (codeword length), that has to be set in Matlab's workspace using the command window. In that way, you only need to change `n` in one place.

Repetition Coding: Measured BER in %

Channel Error Probability	0.1%	0.3%	1%	3%	10%
$n = 3$					
$n = 5$					
$n = 7$					
$n = 9$					

- Determine the error correction capability of the codes.
- How is the error correction capability reflected in those measurements?

3.3 Hamming Codes

The model `hammingcode.mdl` contains a communication system that uses a Hamming code to communicate over a binary symmetric channel. This model uses two parameters from Matlab's workspace, namely `n` as above and `k` (dimension of the code).

Hamming Codes: Measured BER in %

Channel Error Probability		0.1%	0.3%	1%	3%	10%
$n = 7$	$k =$					
$n = 15$	$k =$					
$n = 31$	$k =$					
$n = 63$	$k =$					

Observe that the repetition code with $(n, k) = (3, 1)$ also is a Hamming code.

- Why does the BER increase with increasing codeword length?

3.4 Binary BCH Codes

BCH codes are a class of linear codes. Among codes over various alphabets it specifically contains binary codes. These codes allow us to choose among different values of the parameters length, dimension and minimum distance. Start with the model `hammingcode.mdl`, but replace the encoder and decoder by corresponding blocks for BCH codes from the model `lab3blocks.mdl`. These blocks also use the variables `n` and `k`.

You cannot choose n and k arbitrarily. Look up the help page for `bchenc` in Matlab's documentation. There you can find some allowed combinations. Note that those BCH codes that have the same parameters as Hamming codes really are Hamming codes.

Try out some combinations of n and k and fill in your results below. It is a good idea to compare codes with approximately the same rate, $R = k/n$. For example, you can compare BCH-(15,5) to the repetition code with $n = 3$, and BCH-(63,36) to Hamming-(7,4).

BCH: Measured BER in %

Channel Error Probability			0.1%	0.3%	1%	3%	10%
n	k	R	Measured BER				

If the rate is kept unchanged it seems like the BER decreases with increasing codeword length, at least when the error probability of the channel is low.

- What could be the reason for that?
- Could you think of a reason why it doesn't hold for high channel error probability?

3.5 Source Coding

Simulink has some support for source coding, but not for Huffman coding. Therefore, we resort to plain Matlab for this part. Load the file `planet.mat` into Matlab. It is a grayscale image in 256 × 256 pixels, with integer values from 0 to 255, where 0 corresponds to black and where 255 corresponds to white.

You can look at the picture using

```
imshow(planet,[0 255])
```

To determine a Huffman code, we need a histogram (probability distribution) of the pixel values. You create one from the picture using

```
prob = hist(planet(:),[0:255])/256^2;
```

and you can look at it using

```
plot(prob)
```

You create the code using

```
[dict,avglen] = huffmandict([0:255],prob);
```

In `dict`, there is a table that describes the code, and `avglen` is the mean codeword length. You get the codeword lengths using

```
L = zeros(1,256);
for i=1:256
  L(i)=length(dict{i,2});
end
```

You might want to look at those lengths using `plot`.

- In principle terms, what is the relation between the codeword lengths and the probabilities?

The entropy

$$-\sum_{i=0}^{255} \Pr\{i\} \log_2 \Pr\{i\}$$

is a lower bound on the mean codeword length. You can determine it like this:

```
-sum(prob(prob>0).*log2(prob(prob>0)))
```

The `prob>0` that you find in the expression above removes all cases of the probability 0, so that Matlab does not have to calculate the logarithm of 0. The redundancy in bit per symbol is then the difference between the mean codeword length and the entropy.

- What is the entropy of this particular probability distribution?

3.6 If you have time left

Start with the model `repetitioncode.mdl`, but replace the channel with the combination

line code `Polar_NRZ` – amplifier – AWGN channel – matched filter – detector,

that is, as it was in Laboratory Exercise 2. This combination is readily available in `lab3blocks.mdl` and gives you a binary symmetric channel.

It may take too long to simulate as many bits in this model as we have done so far in this lab. Therefore reduce that to say 10^3.

Now compare uncoded communication with repetition coding in terms of BER, but make sure that you use the same energy per information bit in both cases. Choose this bit energy and the variance on the channel in such a way that the uncoded system has BER approximately 5%. Adjust the parameter `Spb` such that the number of samples per information bit is the same in both cases.

- What can you observe?
- How can this observation be explained?