Mid Sweden University
Magnus Eriksson

Revised: 2013-11-12

Assignment 2 in Simulation of Telesystems
Laboratory exercise: Introduction to Simulink and Communications Blockset

You are expected to conclude exercise 1-7 within a week, and the rest within two weeks.

Presentation: You do not have to send the lab report to the supervisor, but should demonstrate it during supervision seminars. Save or print out plots and code, and be prepared to demonstrate that the code works and explain the principles. Also write down all command-line Matlab commands and answers to the questions.

Preparation

Do the Matlab introduction exercise. The answers to these are partially presented during a lecture.

1. Help commands

Start the Matlab program. Test the Matlab help commands: **Help**, **helpwin**, **doc**, **lookfor, whatsnew**. Shortly describe the difference between them. Get acquainted with the help text structure. Hint: Write **helpwin help** for help on the help command.

2. Toolboxes and Simulink libraries

Inform yourself on what Matlab toolboxes and Simulink libraries that are installed, by means of the command **ver** and the help texts. Which of the toolboxes and libraries do you expect that you will meet during this course, or may gain your interest? (Nowadays Mathworks do not use the word blockset any more. For example Communications toolbox previously only consisted of .m-files, and Communications blockset of SImulink blocks and models. Nowadays both or part of the product Communication Systems Toolbox).

3. Simulink examples
Run the Matlab command **doc**. Search for **Communications systems toolbox examples**. Browse through the example, and look especially under the headline **Simulink examples**. Run at least one of the Simulink models by clicking on **Start simulation** (the "play" icon). Note that some of the examples are matlab scripts and functions (i.e. .m-files) rather than Simulink models.

Several of the Simulink blocks in each model are subsystems that contain other blocks. Try to see the content of some subsystems. Some subsystems can be opened by double-clicking at the arrow. Others may be opened by clicking on the arrow icon on its symbol. You may also browse among the subsystems by choosing **View** -> **Model** browser options. Finally you may find the helptext for a subsystem from the background menu.

Which model did you try? Which subsystem did you open?

4. Freeware Simulink models
Download, install and try at least one user contributed Simulink model related to a communication systems standard from the "Matlab Central File Exchange". See http://www.mathworks.com/matlabcentral/ -> "File exchange". Search for example for "wireless", "communication" or "network". Be prepared to demonstrate the model to the whole group of students. Note that some models require an old version of Matlab, for example version R2010.

Presentation: Which model did you try? Did it work without errors? Demonstrate it to the supervisor.

5. Create a simple Matlab function
Create a simple Matlab function (i.e. an .m file where the first line starts by the word **function**). The function should take an input argument, calculate something, and reply by an output argument. Save the function file, and call it from the command prompt by setting it in a function expression.

Hint: Use the Matlab commands **edit** and **function**. Read their help texts. You may for example start our from some existing Matlab function by writing

mkdir h:\matlab
cd h:\matlab
edit myfunction.m.

A typical Matlab function starts with

```
function y=function_name(x1, x2)
```

```
% This is a function with two input parameters
% and one output parameter
…
y = … ;
```

or:

```
function [y1, y2]=function_name(x)
% This is a function with one input parameter
% and two output parameter
```

Add a help text for your function after a % character on the second line, below the function declaration. Thus it would be possible to apply the help command on your function by writing `help` *function_name*.

Save the file.

Printout or write down your function.

Make a function call, for example:

```
myfunction(2)+myfunction(3)
```

Presentation: Print out or write down your function, your function call and its result.

6. Create a simple Simulink model

Create a simple Simulink model, by writing the Matlab command **simulink** in the command window, and choosing the menu alternative **File -> New -> Model.** Drag blocks from the **Simulink Library Browser**, and drop them into your model. Your model should at least include one source block, e.g. a signal generator, and one sink, for example a measurement instrument. Connect the source and sink blocks with wires by holding the mouse over a block connector, and dragging it to a connector on another block.

Run the simulation.

You may save your model by means of the menu **File** -> **Save**.

If you create a large Simulink model, it may be practical to organize it into a hierarchy of several subsystems. Create a subsystem by marking several blocks (hold down the Shift button while clicking on each of the blocks), clicking with the right mouse button on the group of blocks, and choosing **Create Subsystem** on the background menu.

Open the preferences of your model by choosing the menu alternative **Simulation** -> **Configuration parameters** -> **Solver**. Here you may set the duration time of each simulation. Enter **inf** (infinite) in the end time field if you want the simulation to run until your click on the stop icon.

You may set the **Sampling time** of the system, i.e. the "clock frequency", in the box "Fixed-step size". If you write "auto" in that field, the sampling time of the blocks in the model is automatically adopted to the sampling time of the source block.

In digital communication applications it is normally necessary to choose sampling time type "Fixed Step", and "Discrete (no continuous states)" solver. In signal processing applications, for example simulation of analogue and digital filters, it is often possible to reduce the simulation time by choosing other settings. In those cases, Simulink may automatically choose longer step-time, i.e. skip the simulation of some samples or clock-cycles, and interpolate the these values after the simulation has been carried out. Still sufficient accuracy will be achieved. Simulink is originally designed for efficient numerical solving of differential equations and difference equations describing analogue and digital filters, control systems, physical devices, etc. This facility can seldom be utilized in digital communication protocols and algorithms, since these involve highly non-linear functions and discrete events, and since the output signal of these blocks can be described as non-continuous functions and cannot be interpolated. However, Simulink is interesting for simulation of communication applications anyway, since it may be utilized as a tool for graphical data-flow oriented programming.

7. User defined functions

The set of blocks in the Simulink library should not be considered as a generic programming language, supplying all conceivable demands. You often have to design your own blocks by means of conventional programming. One common approach is to utilize Matlab functions (.m functions).

Develop and call a Matlab function of your own from Simulink. The Matlab function should somehow modify the signal.

Hint: Use the Simulink block **MATLAB fcn**, which you can find in **Simulink Library Browser** -> **Simulink** -> **User Defined Functions**. Alternatively you may use the Simulink block Function, and write the name of your own .m file in it (except the .m file extension).

8. Automated Simulink calls

A common problem is that you want to compile a plot that shows the relation between input parameters and simulation results, for example how the signal-to-noise ratio (SNR) affects the bit-error rate (BER). The SNR is the x-axis and the BER the y axis. Every ”point” in the plot corresponds to one Simulink simulation run. Some ten points per curve may be required for a nice looking plot. To start Simulink manually for each point may be time consuming. The simulation series may be automated by means of a Matlab script that calls Simulink repeatedly. Hint: You may use the command **sim**. Write “doc sim” for further help, and see:
http://www.mathworks.se/help/simulink/ug/using-the-sim-command.html

a) Write a Matlab script that starts a Simulink simulation, for example an existing communication system model. A recommendation is that you do not use a Matlab function, i.e. the .m-file should *not* start with the keyword ”function”. The reason is that Simulink, as well as Matlab scripts, easily can access variables in the so called Matlab base workspace, while Matlab functions use local (in its own workspace) and global variables.

b) Send some calculation results from a Simulink model back to the Matlab script. It is recommended that you start out from an existing communication system, that you consider to study during your project. You may for example start out from a calculation of the Bit Error Ratio. Hint: Use the Simulink block **Sinks -> To Workspace**, or the command **sim**. The result is a so called Matlab struct. To see what fields that are included in the structr x, write x. To see the content of field y in the struct x, write x.y. The output signal often contains a vector with one value for each sample. Sometimes you want to know the last sample, for example the final bit error ratio, and in this case you may use the Matlab function **end**. Sometimes you want to know the average av the sample values, for example the average signal-to-noise ratio, and in that case you may use the Matlab function **mean**. Illustrate this.

c) Use a Matlab script to control some parameter within a Simulink block, for example the ”gain” of an amplifier, or the noise level of a channel model, and consequently the SNR. If you start out from an existing model of a communication system you may for example change the signal-to-noise ratio, the modulation method (M in MQAM), the error correction code, etc. Hint: To transfer parameters you may use the command **set_param**.

d) Test several cases of the controlled parameter by means of a loop. Plot the controlled parameter on the x axis, and the simulation result (the estimated parameter) on the y axis. If the y axis shows the BER, a logarithmic scale is recommended, typically from 10^{-5} to 10^{0}. If the x axis whos the SNR in dB, a linear scale is recommended, while if it shows the SNR in times, a logarithmic scale is recommended.

Hint: The sequence of input parameter values as well as simulation result values should be saved into two vectors of equal length. Use **for** or **while**, and **end**, to achieve a loop. Use the function **num2str** or **int2str** to convert the numerical inparameter values to strings that you may use in the **set_param** function call.

Use the commands **plot** or **semilogy**, **title**, **xlabel** and **ylabel** to create a nice looking plot.

e) If the simulations take long time, it is practical to develop a separate Matlab script for simulation, and a second script (or function) that presents the simulation results in plots and tables. Develop a simulation script that saves the input parameters as well as the simulation results to the disk by means of the **save** command, successively during the simulation. Also develop a presentation script, or function, that retrieves the data, by means of the **load** command, and presents a plot.

There are several advantages with this approach:

- You may modify the graphical presentation without running the time consuming simulations again.
- You may add some additional parameter values to the result, without simulating the previous results once again.
- If the simulations are interrupted, for example by a system crash or a power failure, the results may be partially saved to the disk, and you can easily modify the script to continue where it was interrupted.
- You may run simulations on several computers in parallel. Each computer runs a version of the simulation script, with different input parameters. The results are stored into several files. The presentation Matlab function compiles the result files into one file, and plots the results.

f) **Voluntary exercise:** Demonstrate the parallelization mentioned above.

g) **Voluntary exercise:** Investigate the effect of two input parameters. A curve is plotted in the same plot for every value of the second parameter. Alternatively, you may compare two Simulink models, for example two communication systems, and present the results in two curves in the same plot. Use the function **legend** to add information to the plot that states what each curve shows.

9. Control the simulation time and simulation accuracy

If the simulation time is too short, the plot may show a "jittery" curve due random fluctuations originating from noise sources or randomized data sources. The BER may be estimated as 0 allthough there are noise and interference, since no bit errors occur during the short simulation. You may increase the simulation time in the Simulink model settings.

a) Choose or develop a Simulink model that involves a random source. Experiment on what is a sufficient simulation time to avoid "jittery" curves (i.e. large differences for every new simulation), or to avoid that the BER becomes zero although there are noise.

What model did you choose? Give a short example on sufficient simulation time, for example to achieve BER>0 for a couple of SNR values.

b) **Voluntary exercise:** Develop a Matlab script that runs the same simulation input parameter case several times (by means of an inner loop). It should store the result from each inner loop iteration in a vector, which we may call *y*, and calculate the average Y = **mean**(*y*). The script should automatically interrupt the inner loop (by means of the **break** command) when a certain condition is fulfilled, for example when the number of bit errors that have occurred is at least 3, or when the simulation time is above a certain number of seconds (by means of the **tic** and **toc** functions).

c) **Voluntary exercise:** Further develop the above script to stop when the accuracy is sufficiently high. The script should estimate the standard deviation *S* of the above estimate *Y*, as *S* = **std(***y***)/(sqrt(length(***y***)-1).** The script should be interrupted when *S* is below a certain threshold value *T*. Those familiar with mathematical statistics may calculate the threshold value *T* for a certain confidence interval of *Y*. For example, you may want a 80% confidence interval of Y that is smaller than 5% of the y axis interval, i.e. the curve jitter amplitude should be less than 1/20 of the height of the y axis. You may assume that the *y* values have Gaussian distribution.

Sometimes the mean result of each Simulation run is different for each simulation run, independently of the simulation time. Such a process is *non-ergodic*, meaning that the time mean value (the average result from one infinitely long simulation run) differs from the ensemble mean value (the average result of all simulation runs for the same input parameter values). In this case it is necessary to add an extra loop that simulates each input parameter case several times, and calculate the average result, as above. The loop may be interrupted when the *standard deviation* of the results is sufficiently low.

11. Voluntary exercise: Use the **bertool** command to repeatedly call a Simulink model of a communication system, and plot the BER vs SNR.

10. Voluntary exercise: Discuss how the computational efficiency of your Matlab code and Simulink model can be improved

Suggestions:

- Find a way to divide the simulation sequence into several jobs that can run on several computers in parallel.
- The **profile** command may generate statistics on what Matlab command lines, and what Simulink blocks, that consume most computational time. You may try to optimize those lines or blocks.
- Analyze a time consuming sub-system, and replace it by a simplified model. This may for example be a *look-up table*, a saved signal vector or data sequence that is reused for every simulation run, or a channel model with a certain bit-error ratio.
- Instead of simulating every combination of input parameters, a *Monte Carlo simulation* may be used, which analyzes random combinations of the input parameters.
- If you compare for example two algorithms, the same random sequence may be used for each simulation. Thus, the relative performance will be more

accurate. One option is to reset the random generator to its initial state before each simulation run, a second option is to save sequence into a file or into a workspace array.

- Use *Real-Time Workshop* and some compiler software to compile a Simulink model into executable code.