**Wireless Receivers: Algorithms and Architectures** **Assignment 00**
Fall Semester
Telecommunications Circuits Laboratory
EPFL

# 0 Warmup

## Introduction

Virtually all modern communication systems are based on digital data transmission, e.g. mobile networks like GSM, UMTS and LTE, or broadcast systems like DVB. To detect the data in the receiver, the distortions that are introduced during the transmission must be estimated and corrected. Today, these tasks are mostly performed by digital signal processors.
In this lab course, you will learn to design digital receivers for realistic wireless systems, with a focus on signal processing aspects. You will start by modelling a very simple transmission system where the signal is only disturbed by additive noise. Realistic channels however introduce many other distortions into the signal, for example an unknown timing offset in the A/D converter. Therefore, in each lab you will adapt the receiver to more and more realistic channels by adding components like synchronization units and estimators for unknown channel parameters. In the end, you will have built a receiver that could be used in a realistic communication system. As we are only concerned with the digital part of the receiver, we start our examination behind the A/D converter. In each lab, you will be given time-discrete signals which model the output of the A/D converter and thus the input into the digital signal processor. The signals contain images as payload data, so by recovering the data and displaying the received image on the screen, you will get a qualitative impression whether your receiver works properly or not. In order to quantify the performance of your receiver, you will also receive signals that contain a known bit sequence so you can calculate the bit error rate.

### MATLAB

Throughout this course we will use MATLAB, which is a programming language specialized on numerical computing. If you are unfamiliar with MATLAB, we suggest that you start with reading the "Getting Started" section in the MATLAB documentation, which can be found at `https://ch.mathworks.com/help/matlab/index.html`.
There are also many tutorials available in the internet, for example `http://www.cyclismo.org/tutorial/matlab/`.
Here is a brief list of some useful commands that you might need during the course:

| | |
|---|---|
| help <functionName> | Displays help for the function functionname |
| zeros$(m, n)$ | Returns the $m \times n$-matrix of zeros |
| ones$(m, n)$ | Returns the $m \times n$-matrix of ones |
| length$(x)$ | The length of the vector x |
| size$(x)$ | The size of the matrix x (number of rows and number of columns) |
| reshape$(x, m, n)$ | Returns the $m \times n$-matrix whose elements are taken columnwise from x |
| repmat$(x, m, n)$ | Repeats the matrix x $m \times n$ times |
| circshift$(x, m)$ | Circularly shifts the elements of the vector x |
| angle$(x)$ | The argument of a complex number |
| max$(x)$ | The maximum element of the vector x |
| sum$(x)$ | The sum of the elements of x |
| mod$(x, y)$ | The remainder of the division $x/y$ |
| plot$(y)$ | Plot the data in the vector y |
| conv$(x, y)$ | Returns the convolution of x and y |
| bi2de(bits) | Converts a vector of bits to an integer |

Furthermore, many standard mathematical functions like abs, sin, exp and so on are available for (complex-valued) matrix arguments; they return a matrix of the same size where the function is applied elementwise. The variable $pi$ is predefined with $\pi$, the variables $i$ and $j$ are predefined with the complex number $\sqrt{-1}$
Most available operators have their usual meaning, as known from other programming languages like C/C++. Some operators, however, have a different meaning when used with vectors or matrices as arguments. For example, $a * b$ denotes a matrix multiplication, whereas $a. * b$ is an elementwise multiplication. Since we are mainly dealing with complex-valued vectors, you also need to differentiate between the transpose operators $'$ and $.'$. The prime $'$ is the hermitian-transpose operator, i.e. $a'$ is the transpose and complex-conjugate of $a$, whereas $a.'$ is simply the transpose of $a$.
MATLAB provides an in-program help for each function, which can be accessed by the help <function> command.
If you wan tto use your own laptop for the lab sessions, you need to install the **Communications Toolbox**, **DSP System Toolbox**, and **Signal Processing Toolbox**.

## MATLAB Grader

To evaluate the correctness of the code written during the lab sessions, we will use MATLAB grader. Once you solved a problem in MATLAB, you can copy and paste the working code in MATLAB grader to get a feedback on your answer. It is advised to use regular MATLAB to debug your code as it runs far quicker than its online counterpart and provides many useful debugging tools.

## 0.1 Familiarizing with Lab Workflow

The main purpose of the first two tasks is to get familiar with the workflow of the lab sessions. The main steps of the lab consists in:

1. Download the MATLAB code for the lab session from moodle and extract its content.
2. Each task will have its corresponding script and, sometime, function.
3. Open the matlab files related to the task you have to solve.
4. Complete the code to solve the task at hand.
5. Copy and paste the code you judge as working in MATLAB grader and submit it.

### 0.1.1 Your Tasks

A0T1 Complete the provided script to compute $y = e^{j2\pi fx}$ for $x = 3$ and $f = 1$

A0T2 Complete the provided function to compute $y = e^{j2\pi fx}$ for any $x$ and $f = 1$

## 0.2 The BSC Channel

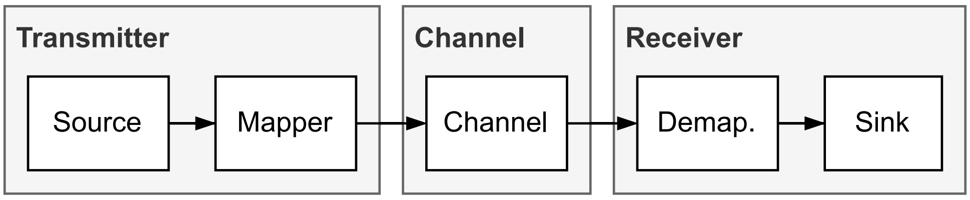

Figure 0.1: Universal System model for Communication Systems

We start our guided project with a review of the very basic communication system model shown in Fig. 0.1. In the transmitter, a source emits a bit stream (representing the information) which is mapped onto a set of symbols. The symbols are then fed into a channel. The output of the channel (which does not necessarily have to be from the same set of symbols) is then demapped back to bits. The set of symbols and the channel model are chosen to replicate (or at least approximate) the physical behavior of the used channel.

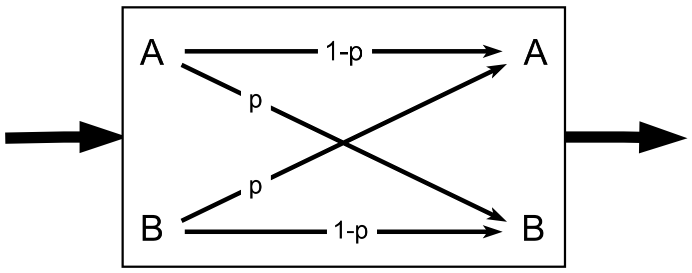

Figure 0.2: The binary symmetric channel (BSC) model

The binary symmetric channel (BSC) shown in Fig.0.2 is one of the easiest to understand channel models in communication theory. The BSC has a discrete binary (and identical) set of input and output symbols. With probability $p$ the channel causes a flip of the transmitted symbol value, otherwise the symbol is transmitted unchanged. Despite its very simplistic nature there are real communication systems (e.g. photons in quantum communication) which can be modeled by it.
Based on student submissions from previous years we have assembled a MATLAB implementation of a BSC simulation. It generates a random bitstream, applies the BSC and calculates the resulting bit error rate (BER).

Please note that this is exceptionally POOR code and combines many of the mistakes most commonly done in MATLAB programming. You should never hand in such code. Nevertheless the script is valid MATLAB code and you can use it as a reference for the language syntax.

## 0.3 Your Tasks

A0T3

– Measure how long it takes to run the original script. The code already contains the tic and toc functions for measuring the execution time. Reduce the number of bits to 1000 and measure the execution time again.

– Implement a better version of the BSC simulation which is smaller and faster. It should **NOT** contain any loops, cell-arrays, duplicated constants or if statements. Compare the improvement in speed for 1000000 and for 1000 bits. You should be at least 20x faster for the first case.

### 0.3.1 Some General MATLAB Advice

MATLAB (MATrix LABoratory) is a very powerful tool for numeric evaluation. It will be our main tool during the course. While the program is very flexible, it is also very slow for general programming tasks. In order to MATLAB offers a lot of highly optimized built-in routines for vector and matrix operations. Writing your MATLAB code in the right way can save you a lot of time and trouble. The second important goal of good MATLAB code is readability. In many cases simpler MATLAB-aware code also leads to faster execution speed. Unfortunately this is not always the case and sometimes you have to find a trade-off between speed and readability. We have assembled some general rules which can help you to achieve good code:

- Use vector or even matrix operations instead of slow loops.
- Write short comments in your code in order to document its behaviour. But keep in mind that code which is well structured and easy to read is even the better documentation.
- Use variables for parameters in order to keep the code reconfigurable.
- Give all your variables meaningful names. It can help to stick to a unified naming pattern.

- Use indention and newlines to keep your code readable. MATLAB offers a built-in smart indent function which provides good results.
- Optimize only performance critical parts, for all other parts readability is more important.