University of Glasgow | School of Engineering

1 Digital Communications 4: Bit Errors and Parity Checking

1.1 Introduction

This laboratory project will introduce you to some issues that occur in digital communication channels. In particular we will study the effects of additive white gaussian noise on the communication channel and how its effects can be mitigated using a simple parity checking and Automatic Repeat-reQuest (ARQ). In your previous laboratories on this course you will have studied modulation formats. Here we will circumvent some of the low-level coding by using the `komm` Python library `https://pypi.org/project/komm/` to provide the appropriate functionality. If you are using your own computer, make sure the Python libraries `scipy`, `numpy`, `matplotlib` and `pillow` are installed. It is recommended that you use a suitable IDE for your project, such as Spyder.

This laboratory will be scheduled over a 3 hour session, where you will be able to ask the lecturer for help. If you do not complete the project in this time, you should undertake to complete it in your own time. The laboratory project should be written up as a short report describing what you've done and the results you have taken along with any conclusions that you draw. Include your python code(s) in the appendices. Make sure your name and registration number is on the report, and, if you have worked as a pair, include the name of your partner. The report should be uploaded to the Moodle assignment by the stated deadline, either using Moodle's inbuilt editor, or as a single PDF.

Figure 1: example 150×100 and 300×200 grayscale images

1.2 Obtaining Digital Data

A number of 8 bit depth grayscale images of various sizes have been provided for you to use in this laboratory project. You may also consider the use of the `numpy.random.randint()`

command to create random binary streams for testing purposes. As the runtime depends on the size of the data, you should generally use the smallest data set first, although in terms of accuracy it may be advisable to use the larger datasets where the smallest images result in a small number of bit errors, say < 25, to improve your accuracy in determining the bit-error-rate.

The following code reads in a grayscale image from a file to a 2 dimensional array of unsigned integer values, and displays the image. If you are using Spyder, you can show graphics in separate windows by setting `Tools>Preferences>IPython console>Graphics>Backend` from `Inline` to `Automatic`. The `unpackbits()` command converts the integer values into a flattened (1D) binary array.

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import komm

tx_im = Image.open("DC4_150x100.pgm")
Npixels = tx_im.size[1]*tx_im.size[0]
plt.figure()
plt.imshow(np.array(tx_im),cmap='gray',vmin=0,vmax=255)
plt.show()
tx_bin = np.unpackbits(np.array(tx_im))
```

1.3 Noisy Channel Simulation

Now we turn to the channel simulation which consists of 3 parts, namely (1) the channel coding, (2) the transmission and reception over a noisy channel, and (3) the channel decoding. We shall start with binary phase-shift keying (BPSK) which has two symbols. We will examine other keying schemes with more symbols later.

The `komm` library provides functions for PSK modulation and demodulation and for simulating an additive white gaussian noise source, as shown below. The modulated psk waveform is by default unit average power. The scalar `snr` specifies the (linear) signal-to-noise ratio for the channel, and the example below corresponds to 6 dB. `tx_data` and `rx_data` will consist of complex arrays.

```
psk = komm.PSKModulation(2)
awgn = komm.AWGNChannel(snr=10**(6./10.))

tx_data = psk.modulate(tx_bin)
rx_data = awgn(tx_data)
rx_bin = psk.demodulate(rx_data)
```

In Spyder, use the variable explorer and write down the value of `Npixels`, and the data type and sizes of `tx_bin`, `tx_data`, `rx_data` and `rx_bin`. Check that the data types are correct, and the comparison of the sizes of `tx_data`, `rx_data` to `tx_bin`, `rx_bin` corresponds to the number of bits per symbol for the modulation scheme used.

1.4 Measuring Bit Error Ratio

Here we will investigate the influence of noise on the received data. An I-Q constellation diagram can be displayed and inspected visually using the following where a sample from `rx_data` has

been copied to `rx_data_samp` so that the detail of the constellation is not obscured.

```
plt.figure()
plt.axes().set_aspect('equal')
plt.scatter(rx_data_samp.real,rx_data_samp.imag,s=1,marker=".")
plt.show()
```

You can also visually inspect the recovered image by rearranging the received data into a decimal valued matrix corresponding to the original image dimensions, and using `imshow` as you did previously.

```
rx_im = np.packbits(rx_bin).reshape(tx_im.size[1],tx_im.size[0])
```

To determine the number of bit errors a bitwise comparison of the transmitted and received binary matrices should be made, summing the case of unequal elements. The bit error ratio (ber) is obtained by dividing the number of errors by the total number of bits, `8*Npixels`. Calculate the ber for a range of decibel values for the signal-to-noise ratio and plot them as $\log_{10} ber$ vs *snr* in decibel. The following python code excerpt can be used to plot the data contained in `x` and `y` arrays (same length) with a logarithmic vertical axis.

```
plt.figure()
plt.scatter(x, y) #plot points
plt.plot(x, y)    #plot lines
plt.yscale('log')
plt.grid(True)
plt.show()
```

Include a curve showing the theoretical dependence for BPSK or QPSK,

$$\frac{1}{2}\mathrm{erfc}\sqrt{\frac{10^{snr(\mathrm{dB})/10}}{k}}$$

where erfc is the complementary error function (available in the python `scipy` library, i.e. `scipy.special.erfc(x)`) and *snr* is the signal to noise in dB with k bits per symbol.

1.5 Parity Check

In this task consider the data as consisting `Npixels`× 8-bit words and replace the least significant bit of each 8-bit word with a parity bit, which doesn't make any discernable difference to your view of the grayscale image. You are free to select whether you use even parity or odd parity: setting the 8th bit to the modulo 2 (`%2` in python) sum of the previous 7 bits corresponds to even parity.

The parity test of the received data is done by looking at the modulo 2 sum of each 8-bit word. If this is different from the parity setting you should do an Automatic Repeat-reQuest (ARQ) and resend the word. Therefore you should amend your code to simulate the transmission of an 8-bit word at a time as a step within a loop. Place a counter in your code to add up the total number of ARQs. If you are unfamiliar with python, please note that control blocks (such as initiated by

```
for i in range(start,stop,step):
```

are defined by indentation, that counters begin at `i=start` and the block is not executed for the final value `i=stop`.

Visually inspect the received image and determine the bit error ratio. Plot the *ber* as a function of *snr* in dB as before. Additionally plot the log of the ratio of the total number of ARQs to the number of pixels as a function of *snr* in dB.

Once you have obtained satisfactory results for BPSK format, repeat the exercise for Quadrature Phase Shift Keying (QPSK) by setting

```
psk = komm.PSKModulation(4,phase_offset=np.pi/4)
```

Check that `psk.bits_per_symbol` is commensurate with your data array sizes. Note also that the `komm` implementation uses gray coding by default (so that symbol errors will give rise to mostly single-bit errors), which you can verify by inspecting `psk.labeling`.

1.6 QAM

Now adapt your code to undertake similar simulations with square QAM: 4-QAM (identical to QPSK so verify that this matches the previous exercise), 16-QAM and 256-QAM. Again, the `komm` implementation uses gray coding by default, which you can verify by inspecting `psk.labeling`. The following code excerpts demonstrates the implementation of 4-QAM.

```
qam = komm.QAModulation(4,base_amplitudes=...)
tx_data = qam.modulate(...)
```

Check that `qam.bits_per_symbol` is commensurate with your data array sizes. `base_amplitudes` is unity by default. However, to draw appropriate comparisons with PSK we should be consistent with the average power per symbol (unity by default with PSK). Therefore inspect the value `qam.energy_per_symbol` and set `base_amplitudes` to a value such that `qam.energy_per_symbol` becomes unity.

1.7 Documentation

Python 3 `https://docs.python.org/3/`

komm `https://komm.readthedocs.io/en/latest/`

numpy and scipy `https://docs.scipy.org/doc/`

matplotlib `https://matplotlib.org/contents.html`

pillow `https://pillow.readthedocs.io`

spyder `https://docs.spyder-ide.org/`

Getting the python libraries

If you are using your own computer, make sure the Python libraries `scipy`, `numpy`, `matplotlib` and `pillow` are installed. These libraries are installed by default with the Anaconda python distribution. It is recommended that you use a suitable IDE for your project, such as Spyder. The `komm` Python library is available from PyPI repository and, if required, can be installed using `pip`. From a python-activated command line (such as Anaconda prompt), use the following command to install in your user App config

```
pip install komm --user
```