University of Glasgow | James Watt School of Engineering

1 Digital Communications 4: Digital Modulation

1.1 Introduction

This coding project will introduce you to modulation and demodulation techniques for digital data. We will be coding in the python programming language in order to make use of a bespoke library for later projects. If you are using your own computer, make sure the Python libraries `scipy`, `numpy`, and `matplotlib` are installed. It is recommended that you use a suitable IDE for your project, such as Spyder, PyCharm or Visual Studio. Tip: if you are using Spyder, you can display, manipulate and save graphics in separate windows by setting

`Tools>Preferences>IPython console>Graphics>Backend`

from `Inline` to `Automatic`.

Each project will be scheduled over a two week period, within which there will be 2 scheduled consultation sessions where you will be able to ask teaching staff for guidance. The 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 student ID number is on the report. The report should be uploaded to the Moodle assignment by the stated deadline, either using Moodle's inbuilt html editor, or as a single PDF file.

1.2 Importing libraries and setting input data

For the input data, you will use the 24 bit binary representation of your student number. The following code imports the useful libraries, and gives an example of converting a decimal number to a numpy array in binary based on the `binary_repr` numpy function (`zfill` fills to required size with zeros and `list` splits the string into individual data).

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import fft
from scipy import signal

def bin_array(num, m):
    """Convert a positive integer num into an m-bit bit vector"""
    return np.array(list(np.binary_repr(num).zfill(m))).astype(np.bool)

# import 24 bit digital data
id_num = 3141592
Nbits = 24
tx_bin = bin_array(id_num, Nbits)
```

Inspect the quantity `tx_bin` using the variable explorer in your IDE and ensure it has the appropriate form.

We will also need to initialise the value of two dimensionless constants scaled to the sample rate: the carrier frequency `fc` and the bit length `bit_len`. Remember that the carrier frequency must be less than 0.5 of the sample rate to meet the Nyquist criterion.

1.3 BPSK modulation

In this section we will be using binary phase shift keying, so the symbol set $s = \pm 1$ is real-valued. Generate the sampled modulated data with 2 nested for loops. We have the outer loop, say `i`, is over the number of bits `Nbits`.

```
for i in range(Nbits):
     {loop code}
```

for which we map the binary data in `tx_bin[i]` to the symbol set `s[i]`. The scope of python for loops are defined by indentation, and in this case run from element 0 to `Nbits-1`. Then we have the inner loop, say `j`, over the bit length `bit_len`. The sampled modulated signal value will be set with an expression of the form,

```
     s[i]*np.cos(2*np.pi*fc*(i*bit_len+j))
```

You should end up with a sampled array of `Nbits*bit_len` values. Plot this array, say `tx_mod`, and the magnitude of its Fourier Transform with,

```
plt.figure()
plt.plot(tx_mod)
plt.show()

plt.figure()
plt.plot(np.abs(fft.fft(tx_mod)))
plt.show()
```

You should observe the π phase jumps in the modulated signal (you may need to zoom in), and the sidebands in the spectrum.

1.4 Demodulation

We will be using coherent detection, so the first step in demodulation is to multiply the modulated signal by the carrier wave again. You should be able to copy-and-paste from your nested loops for modulation and replace the baseband signal with the modulated signal for this. Next you will pass this mixed signal through a low-pass filter. `scipy.signal` library has routines for defining and using digital filtering. The following code excerpt designs a low pass filter by determining the tap weights for `numtaps` and a bandwidth defined in terms of the Nyquist frequency (0.5 sample rate). The accompanying plot should show the magnitude of the frequency transfer function.

```
# low-pass filter
numtaps = 64
b1 = signal.firwin(numtaps, 0.1)
mixed = np.zeros(numtaps)
```

```
w1, h1 = signal.freqz(b1)

plt.title('Digital filter frequency response')
plt.plot(w1/2/np.pi, 20*np.log10(np.abs(h1)))
plt.ylabel('Amplitude Response/dB')
plt.xlabel('Frequency/sample rate')
plt.grid()
plt.show()
```

The filter is applied to a sampled signal `rx_mixed` with

```
rx_lpf = signal.lfilter(b1, 1, rx_mixed)
```

Plot the resulting sampled array and observe the waveform (you may need to zoom in). Try adjusting the filter parameters and observe the effects on the resulting signal.

The final step in the demodulation of digital data is to select an appropriate sample point, and then use a thresholding function to convert floating point into boolean. We shall use the most obvious sample point by taking the midpoint of the bit, i.e. `i*bit_len+bit_len//2`. Therefore construct a for loop over `Nbits` and applying a threshold to the bit midpoint. In this case an appropriate threshold function is the heaviside (step) function

```
rx_bin[i] = np.heaviside(rx_lpf[i], 0)
```

Finally, compare your output binary data with your input data. Do you notice any issues?

You should observe a delay in the output data when compared to the input data. This is because the digital filter comprises delays within each tap, and the filter designs resulting from `signal.firwin` normally have an overall delay of `numtaps//2`. Account for this delay in your code; you will need to add at least `numtaps//2` dummy samples to the end of your binary input data.

1.5 QPSK modulation

Copy your BPSK code to a new file, which you will adapt for QPSK modulation. Here is a summary of the changes you will need to make

1. the modulation will involve the sum of **in-phase** (multiply by cos) and **quadrature** (multiply by sin) components

2. two bits at a time are used in selecting the appropriate symbol from the set of four possibles. Use gray encoding so there is only one bit change to an adjacent symbol.

3. The demodulation will separately do the **in-phase** (multiply by cos) and **quadrature** (multiply by sin) components. Each of these data streams are passed through its own low-pass filter.

4. The thresholding function is done on both LPF data streams, and the corresponding two bit output is identified from the symbol.

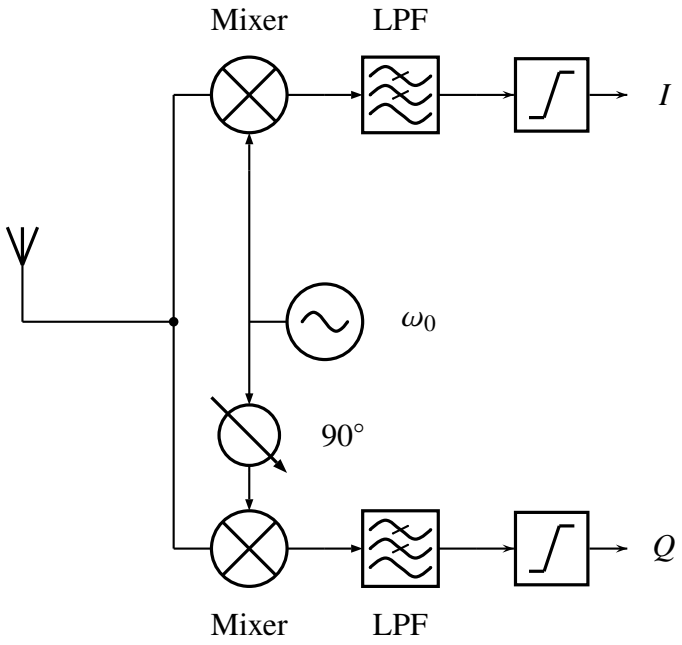

Figure 1: Coherent Demodulation of QPSK

1.6 Documentation

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

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

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

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

Getting the python libraries

If you are using your own computer, make sure the Python libraries `scipy`, `numpy` and `matplotlib` 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.