Advanced techniques in signal processing and communications
2020-2021
Lab #2: Convolutional codes

March 18, 2021

1 Introduction

The goal of this exercise is to implement a convolutional encoder, along with its corresponding decoder, in Python/MATLAB. It entails writing a program that will generate several (independent) sequences of bits, encode each one using a convolutional code, simulate their transmission through a binary symmetric channel (BSC), and decode them using the Viterbi algorithm. The work to be carried out can roughly be structured into:

- Writing a function whose inputs are the sequence of information bits to be transmitted and the generator matrix in polynomial form, and whose output is the encoded binary sequence.
- Writing a function that simulates the transmission through a BSC.
- Writing a function whose input is the *received* binary sequence, that returns the decoded sequence. The function must make use of the **Viterbi algorithm**.

For simplicity we assume the communications system is using a **binary constellation**, i.e., every transmitted symbol carries a single bit.

Some (Python) code is provided as starting point in the directory *convolutional_codes* of the *github* repository `https://github.com/manuvazquez/uc3m_atspc`.

2 Main program

Write a Python/MATLAB script named, e.g., `main.{py,m}`, that simulates the transmission through a communications system using a convolutional code. It must generate several (independent) random sequences of bits and simulate their encoding, transmission and decoding using the implemented functions. In the end, the results obtained for the different sequences are averaged to assess the performance of the coding scheme, which is illustrated by means of a plot.

3 Convolutional encoding

Write a function named, e.g., `conv_encoding.{py,m}`, that performs the convolutional encoding. For that purpose, the function must take as inputs the sequence to be transmitted and the generator matrix, whose elements can be represented as binary vectors with a 1 in the degrees of the polynomial that are present in each output bit, and a 0 in those which are not. For example, polynomial D^3+D+1 can be represented as $[1,0,1,1]$.

Implementation

- an easy way of implementing encoding in
  - Python is using `numpy.convolve`, and in
  - MATLAB is using function `conv`.
- the generator matrix, $\mathbf{G}$, can be represented in
  - Python using lists (of lists of lists), and in
  - MATLAB using `cell` *arrays* ("that can contain data of varying types and sizes").

4 Channel model

Hard decoding is to be implemented, and hence wee need a BSC model. For that goal, we will implement a function called, e.g., `bsc_channel.{py,m}`. Its inputs are a binary sequence and an SNR given by E_b/N_0. The function must compute the probability of error for that particular SNR, and flip some of the bits in the input sequence according to that probability. The output of the function is the sequence with erroneous bits.

The probability of error can be computed using

$$P_e = \mathrm{Q}\left(\sqrt{\frac{2E_s}{N_0}}\right) = \mathrm{Q}\left(\sqrt{\frac{2E_b mR}{N_0}}\right).$$

with m being the number of bits per symbol and R the rate of the code, and where we have used that

$$E_b = \frac{E_s}{mR}.$$

Notice that R is dependent on whether or not you use encoding.

5 Decoding

A function named, e.g., `hard_decoding.{py,m}` implements the Viterbi algorithm using the number of errors as the accumulating metric.

6 Hand-in assignment

The Python/MATLAB source code implementing the requested tasks must be uploaded to AG following the structure outlined above, i.e., you should have separate files for

- the main script,
- encoding implementation,
- transmission simulation,
- decoding implementation, and
- **optionally**, extra functions you might use in the above files.

The file names (`main.{py,m}`, `conv_encoding.{py,m}`,...) are illustrative, but you must respect the stated *functional* division (every file plays a different role).

Additionally, the student must write a small **report** showing the BER curve as a function of E_b/N_0 for a convolutional encoder with generating matrix

$$\mathbf{G}(D) = [D^2 + 1, \quad D^2 + D + 1],$$

along with a curve showing the same values without coding. *Briefly* discuss the results in connection with the theoretical properties of the code.

Important

Regarding implementation, in

- Python only functions from `numpy`, `scipy` and `matplotlib` are allowed (along with any function from the *standard* Python library), and in
- MATLAB no toolbox can be used in the implementation.

7 Assessment criteria

Notice that, in principle, the program must be able to work with *any* matrix G. Failing to comply with this requirement sets the maximum grade down to 9. The report should make it clear whether the code is meant to work on any generating matrix or just the one given in the previous section.

Also important

The report should not exceed 2 (single-sided) pages.

8 Python environment

You can install Python and related dependencies any way you like. However, if you use Anaconda, you can get a working environment (here named `conv`) by issuing from the command line

```
conda env create -f conv_environment.yml
```

Appendix: script

1. Generate a sequence of bits.
2. Compute the channel bit error probability, P_e^c, from the E_b/N_0.
3. Simulate transmission by flipping bits ($0 \to 1$ or $1 \to 0$) according to the above probability of error.
4. Compute the average number of erroneous bits (with no coding) in the received sequence. It should be close to the above *theoretical* value.
5. Encode the sequence of bits generated in `1` using the convolutional code (either using the trellis diagram or via convolutions).
6. Simulate transmission *of the encoded sequence* by flipping bits according the above P_e^c.
7. Decode the received sequence using the Viterbi algorithm.
8. Compute the average number of erroneous bits over de decoded sequence.

Figures should show the values obtained in `4` y `8`.