Técnicas Avanzadas de Tratamiento de Señal en Comunicaciones
2018-2019
Laboratorio #2: Códigos convolucionales

12 de marzo de 2019

1. Introducción

En esta práctica implementaremos un codificador convolucional y su correspondiente decodificador en Python/MATLAB. En resumen, la práctica supondrá completar las siguientes tareas:

- Se implementará un programa que generará varias secuencias de bits (independientes), codificará cada una usando un código convolucional, simulará su transmisión utilizando un canal binario simétrico (BSC), y las decodificará mediante el algoritmo de Viterbi. Este *script* hará uso de las funciones que se deben implementar para las tareas subsiguientes.
- Se implementará una función que tendrá como entrada la secuencia de bits de información a transmitir y la matriz generadora en forma polinomial, y devolverá la secuencia binaria codificada.
- Se implementará una función que simule la transmisión a través de un BSC.
- Se implementará una función que tendrá como entrada una secuencia binaria recibida y devolverá la secuencia decodificada. La función usará el algoritmo de Viterbi para tal fin.

Código (Python) que puede servir como punto de partida se puede encontrar en el directorio *convolutional_codes* del repositorio *github* `https://github.com/manuvazquez/uc3m_atspc`.

2. Programa principal

Escriba un programa en Python/MATLAB llamado, e.g., `main.{py,m}` que simule la transmisión a través de un sistema de comunicaciones que utiliza un código convolucional.

Debe generar varias secuencias de bits aleatorias (independientes), simular su codificación, transmisión y decodificación haciendo uso de las funciones implementadas. Al final, los resultados obtenidos para las distintas tramas se promedian para evaluar el rendimiento del código, que se ilustra mediante una figura.

3. Codificación Convolucional

Implemente una función llamada, e.g., `conv_encoding.{py,m}`, que lleve a cabo la codificación convolucional. Para ello, la función deberá tener como entradas la secuencia a transmitir y la matriz generadora, cuyos elementos se pueden representar con vectores binarios que tienen 1 en los grados del polinomio presentes en cada bit de salida y un 0 en aquéllos que no lo están. Por ejemplo, se puede representar el polinomio $D^3 + D + 1$ como $[1, 0, 1, 1]$.

Implementación

- una forma sencilla de implementar la codificación en
  - Python es utilizando `numpy.convolve`, y en
  - MATLAB es utilizando la función `conv`.
- la matriz generadora, $\mathbf{G}$, se puede representar en
  - Python utilizando listas (de listas), y en
  - MATLAB utilizando `cell` *arrays* (que pueden contener datos de differente tipo y tamaño).

4. Modelo de canal

Se va a implementar una decodificación dura, por lo que necesitaremos un modelo de canal BSC. Para ello implementaremos una función llamda, e.g., `bsc_channel.{py,m}`. Tiene como entradas una secuencia binaria y una SNR dada por E_b/N_0. La función debe calcular la probabilidad de cruce del canal para esta SNR e introducir errores en la secuencia de entrada de acuerdo con dicha probabilidad. La secuencia con errores es la salida de la función.

Para calcular la probabilidad de cruce se puede utilizar

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

siendo m el número de bits por símbolo y R la tasa del código, y donde se ha utilizado que

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

Hay que tener en cuenta que R depende si utilizamos o no codificación.

5. Decodificación

Una función llamda, e.g., `hard_decoding.{py,m}` implementará el algoritmo de Viterbi usando el número de errores como métrica a acumular.

6. Resultados a entregar

El código Python/MATLAB que implementa las tareas propuestas debe ser depositado en AG siguiendo la estructura indicada arriba, por lo que debería tener ficheros separados para

- el programa principal,
- la implementación de la codificación,
- la simulación de la transmisión,
- la implementación de la decodificación y,
- **opcionalmente**, funciones extra que se hicieran necesarias en los ficheros de arriba.

Los nombres de los ficheros (`main.{py,m}`, `conv_encoding.{py,m}`,...) son orientativos, pero se debe respetar estrictamente la división *funcional* indicada (cada fichero cumple una función específica).

Además, se deberá elaborar un pequeño **informe** que muestre la curva de BER en función de la E_b/N_0 para el codificador convolucional con matriz generadora $\mathbf{G}(D) = [D^2 + 1, \quad D^2 + D + 1]$, junto con la curva que muestra los valores sin codificación. Comente *brevemente* los resultados obtenidos en función de las propiedades teóricas de dicho código.

Importante

En cuanto a la implementación, en

- Python solo se pueden utilizar funciones de `numpy`, `scipy` y `matplotlib` (además de cualquier función de la librería *estándar* de Python), y en
- MATLAB no se puede utilizar ningún *toolbox*.

7. Criterios de evaluación

Nótese que, en principio, el programa debería funcionar para *cualquier* matriz G. No cumplir este requisito supone rebajar la calificación máxima a 9.

También importante

La extensioń del informe no debe superar las 2 páginas (a una sola cara).

8. Entorno Python

Se puede instalar Python y sus dependencias de varias formas. Sin embargo, utilizando Anaconda se puede obtener un entorno funcional (llamado `convolutional`) ejecutando desde la línea de comandos

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