TRANSMISIÓN DE DATOS 2007/08			
Práctica 6: Codificación Convolucional		Grupo	
		Puesto	
Apellidos, nombre		Fecha	
Apellidos, nombre		17, 19 y 20 de Diciembre	

El objetivo de esta práctica es familiarizar al alumno con la codificación de canal mediante códigos lineales.

Para llevar a cabo la práctica, desarrolle cada ejercicio en un fichero MATLAB dentro del directorio P6.

Se proporcionan, accesibles desde la página de la asignatura, una serie de ficheros como guía para las funciones que se deben implementar. Todos estos ficheros están accesibles en el fichero CodLineales.zip.

Además del código de las funciones que se piden en cada apartado se debe adjuntar los scripts que se desarrollen para poner a prueba dichas funciones y completar los ejercicios propuestos.

Justo antes de finalizar la práctica, comprima el proyecto en un fichero TxDatosP6GXzz.zip (siendo X el grupo –A, B ó C-, y zz el número de pareja) conéctese al sistema de entrega de prácticas de la Intranet y entréguelo en el grupo que corresponda (A, B o C). Guárdese adicionalmente una copia personal, para posible reutilización del código en prácticas posteriores. Recuerde borrar su trabajo del ordenador del.

1. Códigos Convolucionales

Ejercicio 1: Desarrollo de un codificador convolucional

Es este ejercicio se desarrollará un codificador convolucional no recursivo. Para ello deberá apoyarse en los ficheros suministrados en esta práctica y en los fundamentos explicados en clase de teoría.

A partir del fichero facilitado desarrolle una función con el siguiente interfaz:

```
function [code] = convEncoder(polGenerator,Sec)
```

- Recibe:
  - polGenerator == matriz de 0's y 1's en la que cada fila es un polinomio generador.
  - Sec== Secuencia a codificar. Cada fila se corresponde con una palabra mensaje.
- Devuelve:
  - code==código devuelto.

Nota: *Tenga en cuenta que al conocer los polinomios generadores y el número de columnas de la matriz Sec, somos capaces de inducir la dimensión del buffer del codificador, L, la longitud de palabra código, n, y la longitud de la palabra mensaje, k.*

Ejercicio 2: Codificación Convolucional

Desarrolle un *script* al que llamará `Ejercicio2.m`. Éste, mediante el uso de la función desarrollada en el apartado anterior, codificará la secuencia contenida en el fichero[1] `secuencia.txt`. Se utilizará para tal fin la siguiente matriz generadora:

`[1 1 1 1 1 1; 1 0 1 0 1 0; 1 0 0 0 1 0; 1 1 0 0 0 1]`.

Nota: *Tenga en cuenta que a la salida del codificador habremos de obtener L-1 palabras código más que las palabras mensaje de la secuencia.*

Complete las siguientes tablas:

Mensaje	*Palabra código*

¿Por qué obtenemos más palabras código que palabras mensaje tenemos?

Codifique también, mediante este mismo *script*, la secuencia contenida en `secuencia2.txt`.

Mensaje	*Palabra código*

Comente los resultados:

Nota: *Busque analogías y diferencias entre las dos secuencias y entre sus codificaciones.*

[1] Para la lectura de los ficheros de datos de de esta práctica puede usar la función matlab `dlmread`.

Ejercicio 3: Problemática del decodificador de *Viterbi*.

La decodificación óptima consiste en encontrar el camino a través del diagrama de *Tellis* con la menor distancia a la secuencia recibida (`y`). Con distancia nos referimos a la distancia *Hamming*, implementada en `distanciaHamming.m`

Este cometido se podría cumplir analizando todas las posibles combinaciones de palabras código. Esto es, codificando cada una de ellas (`c`) para luego hallar sus distancias con la secuencia recibida (`y`). De esta forma, en caso de ocurrencia de un error corregible, podríamos recuperar la secuencia original (sería la que menor distancia guardara con la recibida). Esta solución estaría basada en la fuerza bruta y sobra decir que no es la implementada.

Siendo N_1 el número de palabras mensajes posibles y N_2 el número de palabras recibidas, ¿Cuántas secuencias habríamos de codificar para abordar el problema de la forma descrita (en función de N_1 y N_2)?

El algoritmo de *Viterbi*, pese a su elevado coste computacional, supone una importante mejora con respecto a la solución descrita.

En el fichero `convDecoder.m` ofrecemos una implementación de un decodificador *Viterbi* con 2 simplificaciones sobre el algoritmo original:

1) Solo es capaz de decodificar secuencias generadas con `L=2`, si L fuera distinto la función daría error.
2) Si en el proceso de decodificación encontramos 2 supervivientes con la misma distancia acumulada, tan solo nos quedaremos con uno de ellos.

Ejercicio 4: Decodificación óptima por el algoritmo de *Viterbi*.

Desarrolle un *script* al que llamará `Ejercicio4.m`. Este, mediante el uso de las funciones desarrolladas en los apartados anteriores, deberá codificar y decodificar la secuencia contenida en el fichero `secuencia3.txt`. Se utilizará para tal fin la siguiente matriz generadora:
`[0 0 1 1; 0 1 0 1; 0 1 1 1;1 0 0 0;1 0 1 1;1 1 1 1]`.

Mensaje	*Palabra código*	*Mensaje Decodificado*

Ahora haremos lo mismo, pero introduciendo una componente de ruido (`ruido1.txt`) a las palabras código.

Mensaje	*Palabra código*	*Ruido*	*Codigo + Ruido*	*Mensaje Decodificado*

Y una vez más introduciendo el patrón de ruido contenido en el fichero `ruido2.txt`:

Mensaje	*Palabra código*	*Ruido*	*Codigo + Ruido*	*Mensaje Decodificado*

Comente los resultados: