Práctica 4

Decodificación Iterativa

Procesamiento de señal en sistemas de comunicaciones y audiovisuales

Máster Universitario en Ingeniería de Telecomunicación

ETSIT-UPV

Índice

1. Objetivos de la práctica

Con esta práctica se pretende que los alumnos asimilen la teoría estudiada en clase sobre los turbo códigos. Para ello durante esta sesión de prácticas se va a implementar en MATLAB un turbo codificador y un turbo decodificador. En la práctica anterior de la asignatura se utilizaron las funciones `turbo_codif` y `turbo_dec` que implementaban ambas funciones respectivamente. En esa práctica se evaluó el comportamiento de estos códigos y se comparó en terminos de BER con otros códigos de canal. En la sesión de hoy el alumno debe programar ambas funciones y comprobar el correcto funcionamiento, para ello se tendrán que comparar el resultado de diferentes decodificaciones con el resultado obtenido por las funciones `TC` y `TDEC` que se proporcionan en esta práctica (estas dos funciones tienen los mismos parámetros de entrada y salida que las que se van a programar durante la práctica).

2. Turbo Codificador

En primer lugar diseñaremos la función `turbo_codificador`:

```
>>function [c] = turbo_codificador (x,Enrejado);
```

`c`: secuencia de bits codificada.
`x`: secuencia de bits a codificar.
`Enrejado`: Estructura de enrejado que define los codificadores.

El codificador que se debe programar es el representado en la figura 1. A la hora de programar se deben tener en cuenta las siguientes consideraciones:

1. *Definición de los codificadores.*
   Los codificadores convoluciones que utilizaremos serán codificadores sistemáticos y recursivos, los cuales se definen utilizando el siguiente objeto de Matlab:

   ```
   >>hCC=comm.ConvolutionalEncoder(Enrejado,'TerminationMethod',...
         'Terminated');
   ```

   y para codificar se utiliza la instrucción:

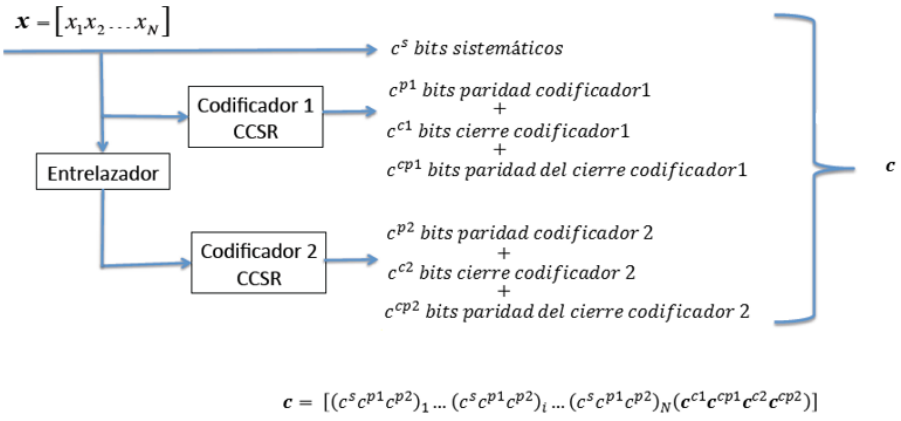

Figura 1: Diagrama de bloques de un Turbo Codificador.

```
x_codif=step(hCC,x);
```

donde Enrejado se define a través de la estructura de trellis de matlab:

```
>>Enrejado=poly2trellis(constraintlength,codegenerator,...
           feedbackconnection);
```

Los campos de esta estructura son los citados a continuación:

- numInputSymbols
- numOutputSymbols
- numStates
- nextStates
- outputs

Para acceder a un campo concreto se debe utilizar la instrucción `Enrejado.nombredelcampo`.

2. *Entrelazador.*
   Para entrelazar los bits sistemáticos que se pasan al segundo codificador se utiliza la función:

```
>>data_entre=randintrlv(data,12);
```

3. *Montaje de la palabra código de salida.*
   A la salida de cada codificador tendremos los bits sistemáticos alternados con los de paridad (en nuestro caso solo habrá un bit de paridad por cada bit sistemático), los bits de cierre y los bits de paridad asociados a esos bits de cierre. Es decir, una vez codifiquemos los bits sistemáticos por cada codificador, a la salida de cada uno de ellos tendremos el vector `x_codif` con la siguiente estructura de los bits:

   $$[(c^s c^p)_1 \cdots (c^s c^p)_i \cdots (c^s c^p)_N (\mathbf{c}^c \mathbf{c}^{cp})].$$

   Donde $(c^s c^p)_i$ son cada uno de los $i = 1, \cdot, N$ bits de entrada al codificador y su bit de paridad asociado, siendo $N = NumInfobits$. Por otro lado $\mathbf{c}^c$ y $\mathbf{c}^{cp}$ son vectores de bits que indican los bits de cierre y los bits de paridad asociados al cierre del decodificador. El número de bits de cierre será el tamaño de memoria de nuestro codificador convolucional el cual podemos averiguar como: $constraintlength - 1$ o $log2(Enrejado.numStates)$.

   Utilizando los vectores de salida de ambos codificadores tendremos que construir el vector de salida del codificador convolucional como:

   $$[(c^s c^{p1} c^{p2})_1 \cdots (c^s c^{p1} c^{p2})_i \cdots (c^s c^{p1} c^{p2})_N (\mathbf{c}^{c1} \mathbf{c}^{cp1} \mathbf{c}^{c2} \mathbf{c}^{cp2})].$$

   Los bits sistemáticos de la palabra código a la salida del codificador convolucional diseñado son los bits de datos, los cuales coindicen con los bits sistematicos del codificador uno. En el caso del segundo codificador los bits sistemáticos son los bits de datos entrelazados.

Para comprobar el correcto funcionamiento del codificador programado genere varias secuencias aleatorias de bits usando:

```
>>x = randi( [0 1], NumInfoBits , 1);
```

donde NumInfoBits puede tomar el valor o valores que desee. Codifique esas secuencias con el algoritmo programado y con la función TC que se proporciona en la práctica y compruebe que producen la misma palabra código. Para ello utilize el siguiente enrejado:

```
>>Enrejado = poly2trellis ( 4, [ 13 15 ], 13 );
```

3. Turbo Decodificador

Una vez se ha programado y validado el correcto funcionamiento del Turbo Codificador, vamos a pasar a programar el Turbo Decodificador correspondiente, el cual queda representado en la figura 2.

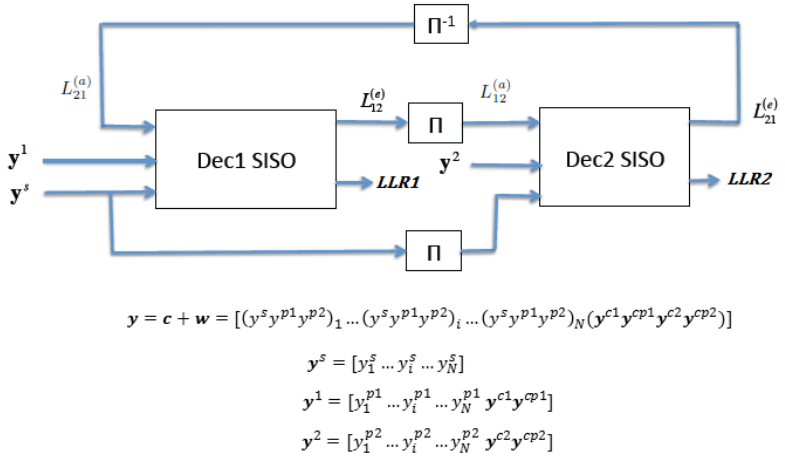

Figura 2: Diagrama de bloques de un Turbo Decodificador.

```
>>function [BitsDec,LLR] = turbo_decodificador (y,Lc,Enrejado,Niter)
```

`BitsDec`: secuencia de bits decodificada.
`LLR`: LLR asociados a la secuencia de bits decodificada.
`y`: secuencia recibida que vamos a decodificar.
`Enrejado`: Estructura de enrejado que define los codificadores.
`Niter`: número de iteraciones de nuestro decodificador.

A la hora de programar se deben tener en cuenta las siguientes consideraciones:

1. *Montaje del vector de entrada de cada decodificador.*
   Como señal de entrada del algoritmo tendremos el vector y que viene definido por

$$\mathbf{y} = [(y^s y^{p1} y^{p2})_1 \cdots (y^s y^{p1} y^{p2})_i \cdots (y^s y^{p1} y^{p2})_N (\mathbf{y}^{c1} \mathbf{y}^{cp1} \mathbf{y}^{c2} \mathbf{y}^{cp2})].$$

   A partir de esta señal, tendremos que montar el vector de entrada de cada codificador los que llamaremos $\mathbf{y}_{c1}$ e $\mathbf{y}_{c2}$ respectivamente. Estos vectores serán una combinación de los vectores $\mathbf{y}^s$, $\mathbf{y}^1$ e $\mathbf{y}^2$ representados en la figura 2. Estos vectores quedarán como:

$$\mathbf{y}_{c1} = [(y^s y^{p1})_1 \cdots (y^s y^{p1})_i \cdots (y^s y^{p1})_N (\mathbf{y}^{c1} \mathbf{y}^{cp1})],$$

$$\mathbf{y}_{c2} = [(\tilde{y^s} y^{p2})_1 \cdots (\tilde{y^s} y^{p2})_i \cdots (\tilde{y^s} y^{p2})_N (\mathbf{y}^{c2} \mathbf{y}^{cp2})].$$

   donde $\tilde{y^s}$ se ha utilizado para denotar los bits sistemáticos recibidos y^s una vez entrelazados, pues recuerde que los bits sistemáticos del codificador 2 son los mismos que el 1 pero entrelazados, para ello utilice la misma función de entrelazado de matlab que la empleada en el codificador.

2. *Desentrelazador.*
   Para desentrelazar los bits es decir realizar la función inversa del entrelezado se utiliza la función:

```
>>data=randdeintrlv(data_entre,12);
```

3. *Definición de los decodificadores.*
   Los decodificadores utilizados serán decodificadores BCJR, para ello Matlab utiliza un objeto que definimos como:

```
>>hDec=comm.APPDecoder('TrellisStructure',Enrejado,...
       'TerminationMethod',Terminated','Algorithm','Max');
```

   para decodificar se utiliza la instrucción:

```
>>LU=step(hDec,LA,LcI);
```

Donde para cada decodificador LA será: $L_{21}^{(a)}$ y $L_{12}^{(a)}$; y LcI será $Lc*y_{c1}$ y $Lc*y_{c2}$ respectivamente. A la salida de estos decodificadores el calculo de $L^{(e)}$ y LLR utilizando se realiza del siguiente modo:

```
>>Le=LU-LcI(1:2:end);
>>LLR=LU+LA;
```

Observese que LcI contiene tanto los bits sistemáticos como los de paridad. Esta forma de calcularlos se debe a que el objeto de matlab (`comm.APPDecoder`) no proporciona a su salida los valores de LLR sino `LU`, siendo `LU=LLR-LA`. Otra cuestión a considerar es que el decodificador calcula también la información de los bits de cierre, por tanto a la hora de entrelazar y desentrelazar la información Le para obtener la La del siguiente decodificador hay que eliminar la contribución de Le y ponerla como ceros en La.

Con las consideraciones presentadas se debe programar el turbo decodificador con los siguientes pasos:

- Construir las señales de entrada a cada codificador.
- Definir los decodificadores.
- Inicializar $L_{21}^{(a)}$ a ceros (en la primera iteración el primer decodificador no tiene información apriori). El tamaño de la del vector de la información a priori es el número de bits sistemáticos (número de bits de datos) más el número de bits de cierre.
- Bucle:
  - Ejecutar el primer decodificador,
  - calcular $L_{12}^{(e)}$,
  - Calcular $L_{12}^{(a)}$, para ello entrelazar ($L_{12}^{(e)}$) con solo los sistemáticos y añadir ceros para los bits de cierre,
  - Ejecutar el segundo decodificador,
  - calcular $L_{21}^{(e)}$ y LLR,
  - Calcular $L_{21}^{(a)}$, para ello desentrelazar ($L_{21}^{(e)}$) con solo los sistemáticos y añadir ceros para los bits de cierre.
- Desentrelazar el LLR de la última iteración (de solo los bits sistemáticos) y calcular partir de él el vector BitsDec (un valor negativo es un cero y un valor positivo un uno).

Para comprobar el correcto funcionamiento del algoritmo programado, decodifique el vector codificado $\mathbf{c}$, utilizando el mismo Enrejado, en este caso utilizaremos $Lc = 1$ (en otros casos donde el vector recibido esté afectado por un ruido Lc sería $\frac{2}{\sigma^2}$, donde σ^2 es la varianza del ruido), y $Niter = 6$. Los bits de salida del decodificador tendran que ser iguales a los bits de información $\mathbf{x}$. Compruebe también que el valor de LLR que obtenemos a la salida es el correcto, para ello utilice la función `TDEC` que se proporciona en la práctica y compruebe que se obtienen los mismos LLR.

4. Resultados a entregar

Como resultado de la práctica se deben entregar las funciones programadas.