5.3. ALGORITMO DE BAUM-WELCH.

Una vez caracterizado el canal de propagación junto a las etapas de modulación y desmodulación tras la simulación estamos preparados para construir un modelo discreto o modelo oculto de Markov. En este modelo existirán un numero fijo de estados entre los cuales ira permutandoel sistema de acuerdo con las probabilidades de transición entre cada estado. A su vez estos estados tienen asociados una determinada probabilidad de generar un error y de esta forma conseguir que el patrón de errores resultado sea lo mas parecido al comportamiento del sistema original del cual sacamos el modelo de cadenas de Markov.

Los parámetros de modelo discreto son obtenidos con la ayuda del algoritmo recursivo de Baum-Welch el cual necesita como ya hemos mencionado la secuencia de errores que caracteriza el sistema original que deseamos discretizar.

Dado que los sistemas de comunicaciones pueden establecer patrones de error complejos ha sido necesario definir un modelo de tres estados, el cual veremos como es capaz de acomodarse apropiadamente al sistema a modelar.

EL algoritmo Baum-Welch es un algoritmo recursivo que necesita de un estado inicial. Este estado inicial compuesto por una matriz de transición de estados y las probabilidades error para cada estado determinan el resultado al que llega el algoritmo pero no así el tiempo que tarda en estabilizarse. El estado inicial determinado por las matrices A y B determinan $\pi=\{\pi_i\}$, $\pi_i=P(i_1=i)$, es decir la probabilidad de estar en el estado i al principio del experimento en t=1. Esto lo obtenemos mediante la función init.m.

```
function pi=init(A)

% Funcion que inicio las probabilidades de estado a partir del
% conocimiento
% de las probabilidades de transiciones entre estados.
%
%       pi=init(A)
%
% pi: Probabilidades de estado.
% A: Matriz transiciones entre estados.

[v,d]=eig(A);
[x,y]=find(d>0.98&d<1.02);
pi=abs(v(:,x))/d(x,x);

return
```

Para presentar el algoritmo se va a utilizar matrices iniciales así como un patrón de errores arbitrario de forma que podamos observar como se comporta el algoritmo ante un número variable de muestras así como el tiempo necesario para estabilizarse. El patrón de errores que utilizaremos es:

```
[271   0   0   0   0   0   2   0   1 248   2 958   0   1  47   1   1  46   0
168   0   3   4   8   0   0   0   0   8   0  14   3   8   0   2   0   2   2   2   0
  0  21   2   2  10   0  92   4  78   0   0   0 107   0 103   1   1   1   0   1
  0  18   0   1   0   0  13   3   0   9 506   0  11   1   0 399   1 223   2   0
```

181 1 2 1 0 1 7 0 0 2 277 0 4 9 1 4 0 273 2 0
2 0 877 1 0 0 223 0 0 0 0 0 198 0 1 0 0 163 0 2]

Cada número de esta secuencia indica el número bits correctos que existen entre dos bits erroneos y es una forma compacta de escribir una secuencia de errores que puede contener un gran número de bits.

El algoritmo lo iniciaremos con la de transición entre estados

$$A = \begin{bmatrix} 0.9967 & 0.1230 & 0.0014 \\ 0.001 & 0.8008 & 0.1877 \\ 0.0022 & 0.0762 & 0.8109 \end{bmatrix}$$

y la matriz de probabilidad de error para cada estado

$$B = \begin{bmatrix} 1 & 0 \\ 0.3847 & 0.6153 \\ 0.9635 & 0.0365 \end{bmatrix}$$

La función que desarrolla el algoritmo de Baum-Welch es baumw.m

```
function [Af,Bf,P,pi]=baumw(A,B,epattern,p)

% Funcion que aplica el algoritmo de Baum-Welch para la obtencion de las
% matrices A y B que definen el modelo de cadenas de Markoff.
%
%      [Af,Bf,P,pi]=baumw(A,B,epattern,p)
%
% Af: Martriz de probabilidades de transiciones entre estados.
% Bf: Matriz de probabilidades de errores para cada estado.
% P: Probabilidad de obtener la secuencia de estados con la matriz
%    resultado.
% pi: probabilidad en ls iteracion n. Necesaria para inicar otra
%    iteracion en algor.m
% A: Martriz de probabilidades de transiciones entre estados inicial.
% B: Matriz de probabilidades de errores para cada estado inicial.
%    epattern: Patron de errores deducido a partir de la simulacion del
%    canal.
% p: Mattriz para introducir manualmente la probabilidad inicial.

if (nargin==4)
   pi=p;
else
   pi=init(A);
end

l=sum(epattern);
N=max(size(A));   % La matriz A debe ser cuadrada.
eburst=zeros(0,0);
len=length(epattern);
for(j=1:len)
```

```
   eburst=[eburst zeros(1,epattern(j)) 1];
end
t=length(eburst);
alfa=zeros(N,t);
beta=alfa;
alfa(:,1)=pi.*B(:,eburst(1)+1);      % Se suma 1 porque no puede ser
                                     % indice  de
beta(:,t)=1;
for(i=2:t)
   alfa(:,i)=(alfa(:,i-1)'*A)'.*B(:,eburst(i)+1);            % Forward
                                                             % variables
   k=t+1-i;
   beta(:,k)=((beta(:,k+1).*B(:,eburst(k+1)+1))'*A')';       % Backward
                                                             % variables
   %beta(:,k)=(beta(:,k+1)'*A)'.*B(:,eburst(k+1)+1);         % Backward
                                                             % variables
end
P=sum(alfa(:,t));


% Step 2: Computo de las probabilidades de transicion

gamma=alfa.*beta/P;
pi=mean(gamma')';
epsi=zeros(N,N,t);
for(j=1:t-1)
   epsi(:,:,j)=(A.*(alfa(:,j)*(B(:,eburst(j+1)+1).*beta(:,j+1))'))/P;
end
Af=zeros(size(A));
Bf=zeros(size(B));
eps=epsi(:,:,1:length(epsi)-1);
gamm=gamma(:,1:length(gamma)-1);
for(i=1:N)
   for(j=1:N)
      Af(i,j)=sum(eps(i,j,:))/sum(gamm(i,:));
   end
end
Af=Af';
a=size(B);
for(i=1:a(1))
   for(j=1:a(2))
      w=zeros(size(eburst));
      aux=find(eburst==j-1);
      w(aux)=1;
      Bf(i,j)=sum(gamma(i,:).*w)/sum(gamma(i,:));
   end
end
return
```

El algoritmo es lanzado y detenido por la función algor.m que establece cuando se han cumplido las condiciones suficientes para estimar que estamos lo suficientemente cerca de la solución.

```
function [X,Y,gra,graP]=algor(epat1)

% Funcion que lanza el algoritmo hasta que se acerca lo suficiente al
```

```
% optimo.
%
%             function [X,Y,gra,graP]=algor(epat1)
%
% X: Martriz de probabilidades de transiciones entre estados.
% Y: Matriz de probabilidades de errores para cada estado.
% epat1: Patron de errores deducido a partir de la simulacion del canal.
% gra: Secuencia en que se ve como evoluciona el algoritmo.
% graP: Secuencia en que se ve como evoluciona el algoritmo.

load algoritmo.mat            % Aqui tenemos Ai y Bi
[A,B,P,pi]=baumw(A,B,epat1);
dif=5;
i=0;
gra=zeros(0,0);
graP=zeros(0,0);
while(abs(dif)>0.01)
   pant=P;
   [A,B,P,pi]=baumw(A,B,epat1,pi);
   dif=((P-pant)/P)*100;       % Variacion porcentual
   i=i+1;
   if(i>400)
      break
   end
   gra=[gra dif];
   graP=[graP P];
end
X=A';
Y=B;

subplot(1,1,1);
plot(1:length(gra),abs(gra))
title('Evolucion del algoritmo')
xlabel('Numero de iteraciones')
ylabel('|dif(%)|')
grid


plot(1:length(graP),log(graP))
title('Probabildad de semejanza')
xlabel('Numero de iteraciones')
ylabel('log(P(O|lambda))')
grid
return
```

En el caso de que solo los seis primeros términos de la secuencia el algoritmo se convierte en inestable:

[271 0 0 0 0 0]

En la gráfica de la izquierda se representa como evoluciona $P(O \mid l)$, es decir, la probabilidad obtener la secuencia de ocurrencia con un sistema $l = (A, B, p)$ establecido en ese paso de iteración. En la grafica de la derecha vemos la evolución de la diferencia existente entre un paso de iteración i y el anterior i-1.

En este caso el algoritmo no evoluciona hacia una solución estable o al menos no en el número de iteraciones que hemos determinado como máximo para llegar a una solución antes de que la diferencia entre dos iteraciones sea menor que el límite que hemos considerado.

Si utilizamos más muestras de la secuencia de errores, el algoritmo empieza a comportar de forma más eficiente y a converger más rapido a la solución. Así para el caso en que introduzcamos [271 0 0 0 0 0 2 0 1] el algoritmo evoluciona de la formar:

En este caso y encontrando soluciones cuya probabilidad es similar la diferncia entre ellas no se estabiliza hasta la iteración 60.

Podemos apreciar que, a consecuencia de la todavía escasa longitud de la secuencia el algoritmo sigue teniendo comportamientos extraños.

Si utilizamos [271 0 0 0 0 0 2 0 1 248 2 958 0 1 47]

Introducimos

[271 0 0 0 0 0 2 0 1 248 2 958 0 1 47 1 1 46
0 168 0 3 4 8 0 0 0 0 8 0 14 3 8 0 2 0 2
2 2 0 0 21 2 2 10 0 92 4 78]

Con la secuencia completa la solución a la que llega el algoritmo es

$$A = \begin{bmatrix} 0.9956 & 0.0035 & 0.0009 \\ 0.0988 & 0.8084 & 0.0927 \\ 0.0000 & 0.1642 & 0.8358 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 0.4817 & 0.5183 \\ 0.9907 & 0.0093 \end{bmatrix}$$

evolucionando el algoritmo de la forma:

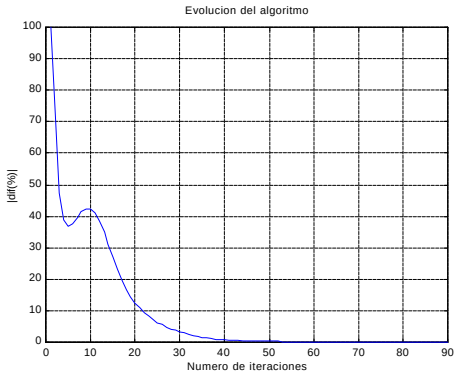
Si utilizamos otra matriz para iniciar el algoritmo

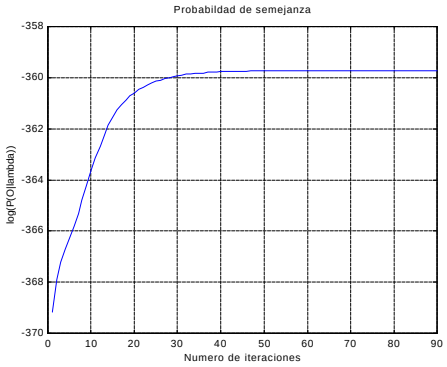
$$A = \begin{bmatrix} 0.98 & 0.01 & 0.01 \\ 0.02 & 0.96 & 0.02 \\ 0.03 & 0.03 & 0.94 \end{bmatrix} B = \begin{bmatrix} 1 & 0 \\ 0.7 & 0.3 \\ 1 & 0 \end{bmatrix}$$

llegamos a la solución

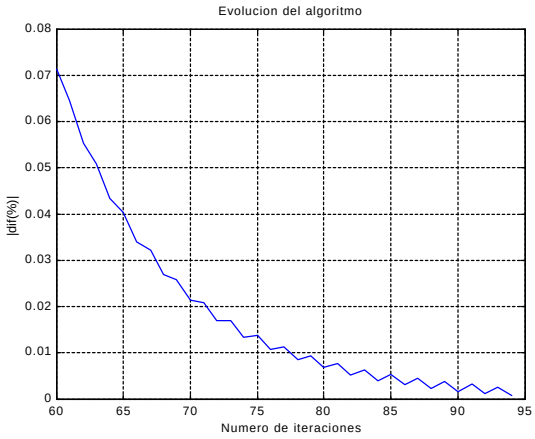
$$A = \begin{bmatrix} 0.9933 & 0.013 & 0.0053 \\ 0.0274 & 0.8126 & 0.1600 \\ 0.0677 & 0.0978 & 0.8344 \end{bmatrix} B = \begin{bmatrix} 1 & 0 \\ 0.4902 & 0.5098 \\ 1 & 0 \end{bmatrix}$$

con los siguientes gráficas de evolución:



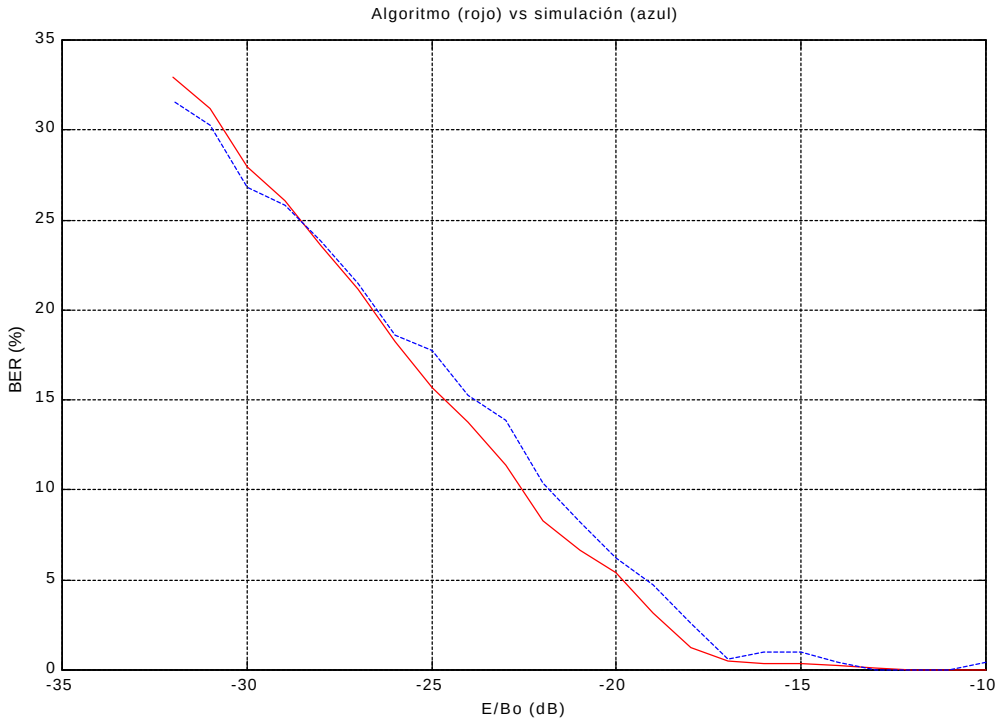


Si observamos algo mas en detalle el final podemos observar como el valor diferencia empieza a oscilar al igual que ocurría con mas claridad con menos longitud de trama.



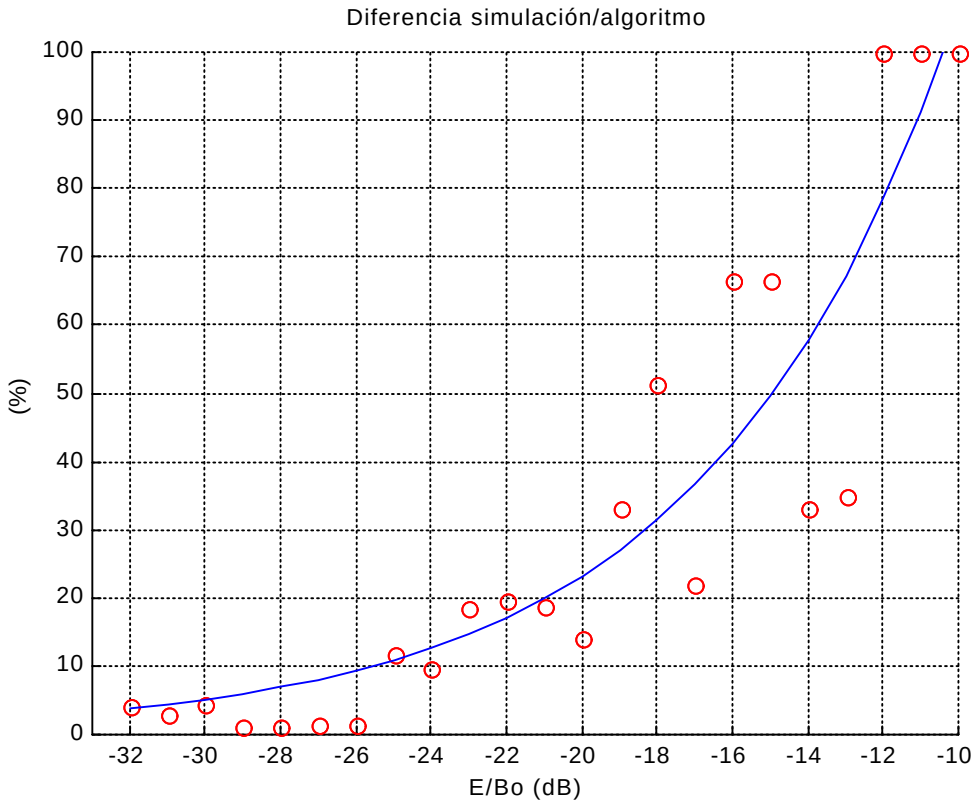
Una vez visto como se comporta el algoritmo utilizamos la secuencia resultado de la simulación del canal este nos proporciona un patrón de errores definiendo al que estemos modelando y en las condiciones en que se encuentre debido a la realción S/N que exista:

Por ultimo podemos ver como el sistema IS-95 se comporta ante una relación señal ruido a la entrada distinta y como a su vez el patrón de errores consecuencia del ruido externo es capaz de producir a partir del algoritmo de Baum-Welch modelos ocultos de Markov que poseen las mismas características a la hora de generar secuencias de errores antes tramas de bits pseudoaleatorias que representarían la señal de voz codificada.



En esta grafica podemos ver como el modelo oculto de Markov se comporta de forma muy similar a como lo hace el sistema simulado a nivel de onda.

La diferencia existente entre ambos se va estabilizando conforme el entorno es más ruidoso.


Diferencia simulación/algoritmo
100
90
80
70
60
50
40
30
20
10
0
(%)
-32
-30
-28
-26
-24
-22
-20
-18
-16
-14
-12
-10
E/Bo (dB)