APÉNDICE 1.

CÓDIGO MATLAB

Estimador de Kay

```
wif=(1/sqrt(2)^(pot(i)))*(randn(1,nsimb) + j*randn(1,nsimb));
cf=rf+wif;
%
%
ventana=(1.5*nsimb/(nsimb^2-1))*(1-(((0:nsimb-1)-(nsimb/2- …
         1))/(nsimb/2)).^2);
r=conj(cf(1:nsimb-1)).*cf(2:nsimb);
frec(i)=(ventana(1:nsimb-1)*angle(r)')/(2*pi*Ts);
```

Estimador de Luise y Reggiannini

```
wif=(1/sqrt(2)^(pot(i)))*(randn(1,nsimb) + j*randn(1,nsimb));
cf=rf+wif;
%
%
m=1;
auxi=0;
while (m<=M)
   in(m)=(1/(nsimb-m))*((cf(1:nsimb-m))*cf(m+1:nsimb)');
   auxi=auxi+ in(m);
   m=m+1;
end
frec(i)=angle(auxi)/((M+1)*pi*Ts);
```

Estimador de Fitz

```
wif=(1/sqrt(2)^(pot(i)))*(randn(1,nsimb) + j*randn(1,nsimb));
cf=rf+wif;
%
```

```
%
m=1;
in=0;
while (m<=M)
   r(m)=sum(conj(cf(m+1:nsimb)).*(cf(1:nsimb-m)))/(nsimb-m);
   in=in+angle(r(m));
   m=m+1;
end
aux(i)=in;
frec(i)=(aux(i))/(M*(M+1)*Ts*pi);
```

Estimador de Lovell y Williamson

```
wif=(1/sqrt(2)^(pot(i)))*(randn(1,nsimb) + j*randn(1,nsimb));
cf=rf+wif;
%
s=1:nsimb-1;
w=6.*s.*(nsimb-s)/(nsimb*(nsimb^2-1));
in=sum(w.*exp(j*(angle(cf(2:nsimb))-angle(cf(1:nsimb-1)))));
frec(i)=angle(in)/(2*pi*Ts);
```

Estimador de Mengali y Morelli

```
wif=(1/sqrt(2)^(pot(i)))*(randn(1,nsimb) + j*randn(1,nsimb));
cf=rf+wif;
%
%
m=1;
while (m<=M)
   in(m)=sum(cf(1:nsimb-m).*conj(cf(m+1:nsimb)))/(nsimb-m);
   m=m+1;
end
s=1:M;
w=3*((nsimb-s).*(nsimb-s+1)-M*(nsimb-M))/(M*(4*M^2-                    …
      6*M*nsimb+3*nsimb^2-1));
x=rem([angle(in(1)) (angle(in(2:M))-angle(in(1:M-1)))]+2*pi,2*pi);
omega=sum(w.*x)/Ts;
```

Simulacion de la BER

```
% Simulacion para analisis de la BER
% Simber
%
%
result=struct('error_frec',{},'long_trama',{},'potencia',{},          …
      'iteraciones',{},'bits_err',{},'bit_tx',{},'berate',{});
i_result=1;
eps=0.05;
%
for ef=5:5:50;
   for pot=0:2:10
      n=8;
      while (n<=14)
         L=2^(n);
         berr=0;
         tab_om=0;
         nbittx=0;
         niter=0;
         ber=0;
         while (berr<100 & nbittx<=1000000)
            [biterr,omega]=simulq1(ef,L,pot);
            niter=niter+1;
            tab_om(niter)=omega;
            berr=berr+biterr;
            nbittx=L*niter;
              [ef pot L berr nbittx niter]
         end
         ber=berr/nbittx;
         festim=mean(tab_om)/(2*pi);
         var=sqrt(sum((tab_om-2*pi*ef).^2)/length(tab_om));
         result(i_result).error_frec=ef;
         result(i_result).long_trama=L;
         result(i_result).potencia=pot;
         result(i_result).iteraciones=niter;
         result(i_result).bits_err=berr;
         result(i_result).bit_tx=nbittx;
         result(i_result).berate=ber;
         result(i_result).frec_estim=festim;
         result(i_result).var_omega=var;
```

```
         save simsinalg.mat
         if (berr<=nbittx*(1/2-eps))
            n=n+1;
            i_result=i_result+1;
         else
            n=17;
            i_result=i_result+1;
         end
      end
   end
end
```

Función *simulq1*

```
% Simulacion.
% SIMULQ1 M&M
% Funcion que simula una transmision QPSK utilizando el algoritmo de …
      Mengali & Morelli
% para la recuperacion de la frecuencia portadora. Como entrada …
      tiene ef- es el error en
% frecuencia que se introduce en el oscilador local, L- es la …
      longitud de los datos que se transmite
% y pot- es el exponente del ruido complejo. Como salida tiene …
      omega- es la estimacion del error
% en frecuencia introducido y nbiterr- es el numero de bits erroneos.
%
function [nbiterr,omega]=simulq1(ef,L,pot)
nbit1=256; %nº de bits que tiene el preambulo
nsimb1=nbit1/2;
lt1=32*(nsimb1+8);
nbit2=L; %nº de bits que tiene la trama de datos
nsimb2=nbit2/2;
lt2=32*(nsimb2+nsimb1+8);
wif2=(1/sqrt(2)^pot)*(randn(1,nsimb2)+j*randn(1,nsimb2));   %ruido    …
      complejo
[sif,c,rcos,wp]=sqpsk1(nsimb1+nsimb2); %esta  funcion  realiza  la    …
      modulacion QPSK
lrc=length(rcos);
wl0=wp+2*pi*ef;
%
```

```
kv=1; %variable que separa el preambulo de los datos
%
if (kv==1)
   sif1=sif(1:lt1); %parte de los simbolos transmitidos que se …
      corresponden con el preambulo
   c1=c(1:nsimb1);
   [omega]=estimmandm1(nsimb1,wl0,rcos,sif1,c1,pot);  %realiza la …
      demodulacion de la señal paso de
      %banda correspondiente al preambulo y la estimacion del error …
      en frecuencia.
   kv=2;
end
if (kv~=1)
   sif2=sif(lt1-lrc+2:length(sif)); %señal paso de banda …
      correspondiente a los datos
   Ts=32/8000;
   t1=0:1/8000:(length(sif2)-1)/8000;
   ic=2*sif2.*cos((wl0-omega)*t1);  %demodulacion
   iq=-2*sif2.*sin((wl0-omega)*t1);
   icq=ic+j*iq;
%Una vez que tenemos las componentes en fase y cuadratura, las
%pasamos por un filtro paso de baja y por el filtro receptor.
   [B,A]=cheby2(9,20,0.25);
   dt=filter(B,A,icq);
   res=conv(rcos,dt);   %paso por el filtro receptor
%Obtenida la señal total lo que queda es muestrear.
%Muestreamos
   k=lrc+1+2;  %el nº 2 es el retraso en muestras que ocasiona el …
      filtro de Chebyshev
   l=1;
   while (k<=length(res)-lrc-1)  %muestreo
      rf(l)=res(k);
      k=k+32;
      l=l+1;
   end
   rf=rf/(mean(sqrt(real(rf).^2+imag(rf).^2))); %se normaliza la …
      señal para que la energia
      %de simbolo sea igual a la unidad.
   rf=rf+wif2; %se añade el ruido complejo
   rcf=decisor(rf);
   c2=c(nsimb1+1:nsimb1+nsimb2); %simbolos que se han transmitido …
      correspondientes a los datos
```

```
   nbiterr=biter(rcf,c2);  %funcion que calcula el nº de bits erroneos
end
```

Función *estimmandm1*

```
%ESTIMMANDM1: Funcion que estima la frecuencia portadora a la …
      recepcion, gracias
%a un preambulo conocido a priori.
%Salida: omega- correccion de la frecuencia
%Entrada: nsimb- nº de simbolos, wl0- frecuencia del oscilador local,
%rcos- funcion raiz cuadrada de coseno alzado, sif- señal paso de …
      banda que se transmite,
%c- simbolos transmitidos y pot- exponente del ruido.
function [omega]=estimmandm1(nsimb,wl0,rcos,sif,c,pot)
%
lrc=length(rcos);
Ts=32/8000;
wif=(1/sqrt(2)^pot)*(randn(1,nsimb)+j*randn(1,nsimb));
t1=0:1/8000:(length(sif)-1)/8000;
%Demodulamos la señal
ic=2*sif.*cos(wl0*t1);
iq=-2*sif.*sin(wl0*t1);
icq=ic+j*iq;
%Una vez que tenemos las componentes en fase y cuadratura, las
%pasamos por un filtro paso de baja y por el filtro receptor.
[B,A]=cheby2(9,20,0.25);
dt=filter(B,A,icq);
res=conv(rcos,dt);
%Obtenida la señal total lo que queda es muestrear.
%Muestreamos
k=lrc+1+2;  %el nº 2 es el retraso en muestras que ocasiona el …
      filtro de Chebyshev
l=1;
while (k<=length(res)-lrc-1)
   rf(l)=res(k);
   k=k+32;
   l=l+1;
end
rf=rf/mean(sqrt(real(rf).^2+imag(rf).^2));   %se normaliza la señal …
      para que la energia
```

```
      %de simbolo sea igual a la unidad.
rf=rf+wif;  %se añade el ruido complejo
%rcf=decisor(rf);
%
cf=conj(c).*rf;   %la señal resultante paso de baja se multiplica por
el conjugado
                  %de los simbolos transmitidos
%Se  procede  al  algoritmo  de  estimacion  del  error  cometido  en
frecuencia
m=1;
M=4;  %parametro de diseño (nº de correlaciones)
while (m<=M)
   in(m)=sum(cf(1:nsimb-m).*conj(cf(m+1:nsimb)))/(nsimb-m);
   %correlacion
   m=m+1;
end
s=1:M;
w=3*((nsimb-s).*(nsimb-s+1)-M*(nsimb-M))/(M*(4*M^2-
6*M*nsimb+3*nsimb^2-1));
x=rem([angle(in(1)) (angle(in(2:M))-angle(in(1:M-1)))]+2*pi,2*pi);
omega=sum(w.*x)/Ts;
```

Función *qpsk1*

```
%Funcion que crea una señal QPSK.
% Se le da como parametro el número de simbolos.
function [sif,c,rcos,wp]=sqpsk1(nsimb)
%
Ts=32/8000; %tiempo de simbolo
%
sec=randn(1,2*nsimb);
sec=ones(1,2*nsimb);
sec(find(sec>=0))=1; %generacion de la secuencia de 1s y 0s
sec(find(sec<0))=0;
k=1;
i=1;
while(i<=nsimb)   %este  bucle  aplica  la  secuencia  binaria  a  la  …
      constelacion QPSK
%para formar la señal QPSK
   if(sec(k)==0)
```

```
      if(sec(k+1)==0)
         c(i)=exp(j*5*pi/4);
      else
         c(i)=exp(j*3*pi/4);
      end
   else
      if(sec(k+1)==0)
         c(i)=exp(j*7*pi/4);
      else
         c(i)=exp(j*pi/4);
      end
   end
   i=i+1;
   k=k+2;
end
rcos=rcmod; %rcmod es la funcion que realiza la funcion raiz …
      cuadrada de
            %coseno alzado.
lrc=length(rcos);
lt=32*(nsimb+8);
k=1;
while(k<=nsimb)   %este bucle se encarga de la conformacion del pulso
raiz
                  %cuadrada de coseno alzado
   fi=c(k)*rcos;
   if(k==1)
      v1=[fi zeros(1,lt-lrc)];
   else
      v1=v1+[zeros(1,(k-1)*32) fi zeros(1,lt-lrc-(k-1)*32)];
   end
   k=k+1;
end
%Ahora se va a trasladar a la frecuencia de portadora
%que en este caso es cero.Ahora el vector de tiempos debe
%tener otra dimension.
t=0:1/8000:(length(v1)-1)/8000;
wp=2*pi*1000;
sif=real(v1.*(cos(wp*t)+j*sin(wp*t))); %modulacion de la señal …
      anterior ya conformada
```

Función *rcmod*

```
function rcos=rcmod();
%Nuevo 3:Segun el doc.IEEE
%Creacion del pulso raiz cuadrada de coseno alzado
%
r=1;
T=1;
alfa=0.36;
t=-4*T:T/32:4*T;
%
a=pi*r*(1-alfa);
b=pi*r*(1+alfa);
c=4*r*alfa;
num=sin(a*t)+c*t.*cos(b*t);
den=pi*r*(1.-c*t).*(1.+c*t).*t;
rcos=num./den;
rcos(32*4+1)=(a+c)/(pi*r);
```

Función *decisor*

```
%Función DECISOR.
function rcf=decisor(cf)
m=1;
tam=size(cf);
while (m<=tam(2))
   if (real(cf(m))>0)
      if (imag(cf(m))>0)
         rcf(m)=exp(j*pi/4);
      else
         rcf(m)=exp(j*7*pi/4);
      end
   else
      if (imag(cf(m))>0)
         rcf(m)=exp(j*3*pi/4);
      else
         rcf(m)=exp(j*5*pi/4);
      end
   end
```

```
   m=m+1;
end
```

Función *biter*

```
% BITER: Función que calcula los simbolos erroneos.
% Entrada: Simbolos enviados- c y recibidos- srec.
% Salida: Nº de simbolos erroneos.
function be=biter(c,srec)
tam=size(srec);
lc=tam(2);
k1=1;
be=0;
while (k1<=lc)
   if (sign(real(srec(k1)))~= sign(real(c(k1))))
      be=be+1;
   end
   if (sign(imag(srec(k1)))~=sign(imag(c(k1))))
      be=be+1;
   end
   k1=k1+1;
end
```