Apéndice I: listados de programas utilizados en las simulaciones.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  DATOS.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Mediante esta función generamos una secuencia aleatoria de símbolos
%QPSK diferencial, que será la emitida por el transmisor. Los símbolos
%son equiprobables. Como parámetro de entrada tenemos la longitud de
%esa secuencia. A la salida, también damos la secuencia
%QPSK antes de la codificación diferencial, con el fin de compararla
%posteriormente con la secuencia detectada.
%Hay que hacer notar que a lo largo de las simulaciones no disponemos
%en ningún momento de la cadena de bits transmitida ni recibida, sino
%que sólo manejamos símbolos QPSK.


function [sec,secdif]=datos(long)


ent=rand(1,long);
%La secuencia QPSK es la generada a continuación:
sec=[ent<0.25]+i*[ent>=0.25&ent<0.5]+(-1)*[ent>=0.5&ent<0.75]+(-
i)*[ent>=0.75];
%Fase de esa secuencia:
fase=angle(sec);
%Codificamos la secuencia diferencialmente:
fasedif(1)=fase(1);
for k=2:long
   fasedif(k)=fase(k)+fasedif(k-1);
end
secdif=exp(i*fasedif);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  DATOS8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Mediante esta función generamos una secuencia aleatoria de símbolos
%8PSK diferencial, que será la emitida por el transmisor. Los símbolos
%son equiprobables. Como parámetro de entrada tenemos la longitud de
%esa secuencia. A la salida, también damos la secuencia
%8PSK antes de la codificación diferencial, con el fin de compararla
%posteriormente con la secuencia detectada.
%Hay que hacer notar que a lo largo de las simulaciones no disponemos
%en ningún momento de la cadena de bits transmitida ni recibida, sino
%que sólo manejamos símbolos 8PSK.


function [sec,secdif]=datos8(long)


ent=rand(1,long);
%La secuencia 8PSK es la generada a continuación:
sec1=[ent<0.125]+i*[ent>=0.25&ent<0.375]+(-1)*[ent>=0.5&ent<0.625]+(-
i)*[ent>=0.75&ent<0.875];
sec2=sqrt(0.5)*((1+i)*[ent>=0.125&ent<0.25]+(-
1+i)*[ent>=0.375&ent<0.5]+(-1-i)*[ent>=0.625&ent<0.75]+(1-
i)*[ent>=0.875]);
sec=sec1+sec2;
%Fase de esa secuencia:
fase=angle(sec);
%Codificamos la secuencia diferencialmente:
fasedif(1)=fase(1);
for k=2:long
   fasedif(k)=fase(k)+fasedif(k-1);
end
secdif=exp(i*fasedif);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  RX.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Señal que nos llega al receptor, ya demodulada;
%Queda la señal en banda base multiplicada por una señal de baja
%frecuencia, rdt, originada por el offset entre la frecuencia de la
%señal portadora y la frecuencia de nuestro oscilador local, que no
%son idénticas. Nuestro objetivo en el esquema completo será reducir
%al máximo ese offset, intentando eliminarlo, para que los datos
%puedan ser recuperados. Se ha utilizado un pulso conformado
%gaussiano. Hemos considerado la existencia de un ruido blanco
%gaussiano en el canal.
%Parámetros: secuencia de datos codificados mediante QPSK u 8-PSK,sec.
%           factor de rolloff del pulso coseno alzado, rolloff.
%           Offset de frecuencia entre la portadora de la señal
%           modulada recibida y la frecuencia del oscilador local, nu;
%           Relación señal a ruido del canal, supuesto que el
%           demodulador no introduce ruido adicional, snr.
```

```
function rdt=rx(secdif,rolloff,nu,snr)
```

```
long=length(secdif);
gdt=pulso(rolloff);
T=length(gdt);
signal=zeros(1,T*long);
for k=1:long
   signal(((k-1)*T+1):k*T)=secdif(k)*gdt(1:T);
end
wif=sqrt(10^(-snr/10)/2)*(randn(1,T*long)+j*randn(1,T*long));
rdt=exp(i*2*pi*nu*(1:T*long)).*signal+wif;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  MULT.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Operación de multiplicación de la señal recibida r(t) por la salida
%del VCO. Este bloque de código lo utilizaremos cuando el sistema
%funcione en bucle abierto.
%Parámetros: señal proveniente del demodulador, rdt; Esta señal puede
%            ser originada por una secuencia de símbolos QPSK o 8-PSK.
%            rdt es la salida del bloque RX.M.
%            Longitud de la secuencia de datos recibida, long;
%            Duración de cada símbolo, T;
%            Offset de frecuencia estimado por el VCO en el periodo
%            anterior, nuestim.


function slp=mult(rdt,long,T,nuestim)


%Señal que proviene del VCO:
vcosc=exp((-i)*2*pi*nuestim*(1:(long*T)));
%Señal de salida del multiplicador:
slp=rdt.*vcosc;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  MATCH.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Pasamos la señal de salida del multiplicador por un filtro FIR
%adaptado a la respuesta del canal, la que aplicamos a las señales en
%el transmisor para conformar los pulsos.
%Este bloque se utiliza indistintamente para sistemas con modulaciones
%QPSK y 8-PSK.
%Parámetros: señal de entrada, slp, y factor de caída del
%filtro,rolloff.


function ydt=match(slp,rolloff)


ydt=1.0191*filter(pulso(rolloff),[1],slp);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  MUESTREO.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Esta función realiza el muestreo de la señal continua que sale del
%filtro adaptado del sistema de sincronización de frecuencia.
%Suponemos la temporización perfectamente conocida, se ha recuperado
%sin errores en algún sistema anterior; así, sabemos que tenemos que
%obtener una muestra de la señal y(t) cada T unidades de tiempo; se
%realiza el muestreo con un cierto retraso tau (constante para todas
%las muestras), que estimamos es el igual al retardo debido al canal y
%que ha sido obviado por ser un parámetro también extraído en la etapa
%de temporización.
%Este bloque lo utilizamos en bucle abierto, tanto para QPSK como para
%8-PSK.
%Parámetros: ydt: salida(continua) del filtro adaptado; es la señal
%           que queremos muestrear.
%           T: periodo de muestro,es el tiempo de símbolo.


function ydk=muestreo(ydt,T)


   ydk=ydt((1:(length(ydt)/T))*T);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  DECISOR.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Bloque que decide qué símbolo QPSK de los cuatro posibles en nuestro
%alfabeto ha sido recibido. Parámetro: señal de entrada al bucle
%debidamente filtrada y muestreada, ydk. A la salida devuelve la fase
%de la secuencia detectada todavía codificada diferencialmente, ang, y
%la secuencia de datos estimada QPSK tras realizar la decodificación
%diferencial.
%Este decisor se utiliza en las operaciones en bucle abierto.


function [secest,ang]=decisor(ydk)


%Calculamos la fase de la secuencia recibida:
fase=angle(ydk);
%Realizamos la operación de decisión:
ang=0*[abs(fase)<=(pi/4)]+(pi/2)*[abs(fase-pi/2)<(pi/4)]+(pi)*[(pi-
fase)<=(pi/4)]-(pi/2)*[abs(fase+pi/2)<(pi/4)]-
(pi)*[(fase+pi)<=(pi/4)];
%Operación inversa a la codificación diferencial de la señal QPSK:
fasestim(1)=ang(1);
for k=2:length(ydk)
   fasestim(k)=ang(k)-ang(k-1);
end
%Obtenemos la secuencia estimada por el decisor.
secest=exp((-i)*fasestim);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  DECISOR8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Bloque que decide qué símbolo 8PSK de los ocho posibles en nuestro
%alfabeto ha sido recibido. Parámetro: señal de entrada al bucle
%debidamente filtrada y muestreada, ydk. A la salida devuelve la fase
%de la secuencia detectada todavía codificada diferencialmente, ang, y
%la secuencia de datos estimada 8PSK tras realizar la decodificación
%diferencial.
%Este decisor se utiliza en las operaciones en bucle abierto.


function [secest,ang]=decisor8(ydk)


%Calculamos la fase de la secuencia recibida:
fase=angle(ydk);
%Realizamos la operación de decisión:
ang1=0*[abs(fase)<=(pi/8)]+(pi/2)*[abs(fase-pi/2)<=(pi/8)]+(pi)*[(pi-
fase)<=(pi/8)]-(pi/2)*[abs(fase+pi/2)<=(pi/8)]-
(pi)*[(fase+pi)<=(pi/8)];
ang2=(pi/4)*[abs(fase-pi/4)<(pi/8)]+(3*pi/4)*[abs(fase-
(3*pi/4))<(pi/8)]-(pi/4)*[abs(fase+pi/4)<(pi/8)]-
(3*pi/4)*[abs(fase+(3*pi/4))<(pi/8)];
ang=ang1+ang2;
%Operación inversa a la codificación diferencial de la señal QPSK:
fasestim(1)=ang(1);
for k=2:length(ydk)
   fasestim(k)=ang(k)-ang(k-1);
end
%Obtenemos la secuencia estimada por el decisor.
secest=exp((-i)*fasestim);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  ZED.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Multiplicador que sigue al muestreador, con entradas desde éste
%y desde el decisor en los sistemas PSK.
%Este bloque conforma el PD del sistema. Recibe como parámetros de
%entrada la salida de la etapa de muestreo, ydk, y la fase de la
%secuencia diferencial detectada, ang. Entre la fase de ydk y ang no
%puede haber una diferencia mayor de ±π/4 si las señales son QPSK y
%±π/8 si son 8-PSK. Obtenemos a la salida una característica zdk que en
%bucle abierto tiene forma de diente de sierra.


function zdk=zed(ydk,ang)


s=exp((-i)*ang);
zdk=ydk.*s;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  ERR.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Bloque generador de error. Este bloque da lugar a una función de
%error obtenida según el algoritmo de Sari y Moridi, para señales
%QPSK. Como parámetro tenemos la señal que recoge la diferencia de
%fase entre el símbolo recibido y el símbolo ideal registrado por el
%bloque detector, esto es, la salida del PD, zdk, y el parámetro beta
%que caracteriza al generador de error QPSK.
%Este programa, junto con ZED.M, simula el comportamiento del PFD.
%Utilizamos este bloque para las simulaciones en bucle abierto.


function edk=err(zdk,beta)


%El parámetro beta elegido para acotar la función de error es:
teta=beta*pi/4;
%Diferencia de fase entre la señal recibida y la salida del detector:
e=angle(zdk);
%Generamos una función de error según el algoritmo citado:
edk(1)=e(1)*[abs(e(1))<=teta]+teta*[abs(e(1))>teta];
for n=2:length(e)
    edk(n)=e(n)*[abs(e(n))<=teta]+edk(n-1)*[abs(e(n))>teta];
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  ERR8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Bloque generador de error. Este bloque da lugar a una función de
%error obtenida según el algoritmo de Sari y Moridi, para señales
%QPSK. Como parámetro tenemos la señal que recoge la diferencia de
%fase entre el símbolo recibido y el símbolo ideal registrado por el
%bloque detector, esto es, la salida del PD, zdk, y el parámetro beta
%que caracteriza al generador de error 8-PSK.
%Este programa, junto con ZED.M, simula el comportamiento del PFD.
%Utilizamos este bloque para las simulaciones en bucle abierto.


function edk=err8(zdk,beta)


%El parámetro beta elegido para acotar la función de error es:
teta=beta*pi/8;
%Diferencia de fase entre la señal recibida y la salida del detector:
e=angle(zdk);
%Generamos una función de error según el algoritmo citado:
edk(1)=e(1)*[abs(e(1))<=teta]+teta*[abs(e(1))>teta];
for n=2:length(e)
    edk(n)=e(n)*[abs(e(n))<=teta]+edk(n-1)*[abs(e(n))>teta];
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  FILTRO.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Este programa simula un filtro LP colocado a la salida del generador
de error; Entra como
%parámetro la señal edk, y calcula su media.
%Se puede utilizar para cualquiera de las dos modulaciones PSK.
%La representación de la salida de este filtro frente a la desviación
%de frecuencia será la característica de salida del PFD, S(fd), la
%cual tendremos que calcular.
%Utilizamos este bloque para las simulaciones en bucle abierto.



function media=filtro(edk);


suma=0;
for i=1:length(edk)
   suma=suma+edk(i);
end;
media=suma/length(edk);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  SIMULQ.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Simulación del sistema QPSK en bucle abierto  hasta la entrada del
%vco; realizamos una simulación de la media de la salida que nos da el
%generador de error en función de la desviación de frecuencia fd,
%que en nuestro caso coincide con el valor de nu, ya que por ahora el
%vco no interviene y por tanto no realiza estimación ni corrección
%alguna. La característica obtenida estará afectada por el ruido.
%Pretendemos calcular el rango de adquisición de frecuencias de
%nuestro sistema y obtener la característica del PFD S(fd).
%Parámetros:   n: número de iteraciones que realizaremos en la
%              simulación.
%              snr: relación señal a ruido a la entrada del receptor.
%Devuelve a la salida las curvas S(fd) para beta=0.5, 0.25 y 0.75.


function [sq5,sq25,sq75]=simulq(n,snr)


clear all;
long=1000;
rolloff=0.5;
nuestim=0;
T=100;
for i=1:n
   %Obtenemos la secuencia QPSK, con y sin codificación diferencial.
   [sec,secdif]=datos(long);
   k=1;
    %Realizamos el resto de las operaciones para cada valor del offset
    de frecuencia dentro del rango de -12% y 12%.
   for nu=-0.0012:0.00001:0.0012
         %Obtenemos la señal demodulada con offset un.
         rdt=rx(secdif,rolloff,nu,snr);
         %En bucle abierto mult.m no realiza ninguna acción.
         slp=mult(rdt,long,T,nuestim);
          %Pasamos la señal por el filtro adaptado a la entrada del
          %dispositivo de sincronzación.
         ydt=match(slp,T);
         %Realizamos el muestreo de la señal.
         ydk=muestreo(ydt,T);
         %Introducimos la señal muestreada y(k) en el decisor.
         [secest,ang]=decisor(ydk);
         %Realizamos la operación del PD.
         zdk=zed(ydk,ang);
          %Introducimos la salida del PD en el generador de error con
          %beta=0.5.
         edk5=err(zdk,0.5);
         %Introducimos la salida del PD en el generador de error con
         %beta=0.25.
         edk25=err(zdk,0.25);
         %Introducimos la salida del PD en el generador de error con
         %beta=0.75.
         edk75=err(zdk,0.75);
         %Introducimos la salida del PFD en el filtro de media.
         simq5(i,k)=filtro(edk5);
         simq25(i,k)=filtro(edk25);
```

```
            simq75(i,k)=filtro(edk75);
            k=k+1;
        end
    end
    %Realizamos una media entre las salidas del filtro en las distintas
    %iteraciones; S(fd) para el caso beta=0.5:
    sq5=sum(simq5,1)/n;
    %S(fd) para el caso beta=0.25:
    sq25=sum(simq25,1)/n;
    %S(fd) para el caso beta=0.75:
    sq75=sum(simq75,1)/n;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  SIMUL8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Simulación del sistema 8PSK en bucle abierto  hasta la entrada del
%vco; realizamos una simulación de la media de la salida que nos da el
%generador de error en función de la desviación de frecuencia fd,
%que en nuestro caso coincide con el valor de nu, ya que por ahora el
%vco no interviene y por tanto no realiza estimación ni corrección
%alguna. La característica obtenida estará afectada por el ruido.
%Pretendemos calcular el rango de adquisición de frecuencias de
%nuestro sistema y obtener la característica del PFD S(fd).
%Parámetros:   n: número de iteraciones que realizaremos en la
%              simulación.
%              snr: relación señal a ruido a la entrada del receptor.
%Devuelve a la salida las curvas S(fd) para beta=0.5, 0.25 y 0.75.


function [s85,s825,s875]=simul8(n,snr)


clear all;
long=1000;
rolloff=0.5;
nuestim=0;
T=100;
for i=1:n
   %Obtenemos la secuencia 8PSK, con y sin codificación diferencial.
   [sec,secdif]=datos8(long);
   k=1;
    %Realizamos el resto de las operaciones para cada valor del offset
    de frecuencia dentro del rango de –12% y 12%.
   for nu=-0.0012:0.00001:0.0012
         %Obtenemos la señal demodulada con offset un.
         rdt=rx(secdif,rolloff,nu,snr);
         %En bucle abierto mult.m no realiza ninguna acción.
         slp=mult(rdt,long,T,nuestim);
          %Pasamos la señal por el filtro adaptado a la entrada del
          %dispositivo de sincronzación.
         ydt=match(slp,T);
         %Realizamos el muestreo de la señal.
         ydk=muestreo(ydt,T);
         %Introducimos la señal muestreada y(k) en el decisor.
         [secest,ang]=decisor8(ydk);
         %Realizamos la operación del PD.
         zdk=zed(ydk,ang);
          %Introducimos la salida del PD en el generador de error con
          %beta=0.5.
         edk5=err8(zdk,0.5);
         %Introducimos la salida del PD en el generador de error con
         %beta=0.25.
         edk25=err8(zdk,0.25);
         %Introducimos la salida del PD en el generador de error con
         %beta=0.75.
         edk75=err8(zdk,0.75);
         %Introducimos la salida del PFD en el filtro de media.
         sim85(i,k)=filtro(edk5);
```

```
            sim825(i,k)=filtro(edk25);
            sim875(i,k)=filtro(edk75);
            k=k+1;
        end
    end
%Realizamos una media entre las salidas del filtro en las distintas
%iteraciones; S(fd) para el caso beta=0.5:
s85=sum(sim85,1)/n;
%S(fd) para el caso beta=0.25:
s825=sum(sim825,1)/n;
%S(fd) para el caso beta=0.75:
s875=sum(sim875,1)/n;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  ERRBA.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Detectamos el número de errores en recepción cuando la señal que
%pretendemos recuperar es una señal digital QPSK en bucle abierto,
%esto es, sin que esté activa la salida del VCO; pretendemos con esto
%comparar la eficacia de un sistema en el que no se aplique una
%corrección de frecuencia (bucle abierto).
%El número de símbolos recibidos es long, el desfase entre la
%frecuencia de la señal portadora y la de la señal local es nu y la
%relación señal a ruido del canal es snr, en decibelios.


function n=errba(nu,long,snr)


%Factor de caída del pulso conformador gaussiano:
rolloff=0.5;
%La salida del VCO no está activada, siempre estima un offset de
frecuencia nulo:
nuestim=0;
%Secuencia de datos QPSK transmitida.
[sec,secdif]=datos(long);
%Duración de cada símbolo:
T=100;
%Señal recibida, ya demodulada y con un offset de frecuencia nu:
rdt=rx(secdif,rolloff,nu,snr);
%La pasamos por el multiplicador:
slp=mult(rdt,long,T,nuestim);
%Salida del filtro adaptado:
ydt=match(slp,rolloff);
%Muestreamos la señal continua:
ydk=muestreo(ydt,T);
%Detección de la secuencia recibida:
[secest,ang]=decisor(ydk);
%Número de errores en la recepción en bucle abierto, en régimen
permanente:
n=length(find(abs(exp(i*angle(sec(51:long)))-exp((-
1)*i*angle(secest(51:long))))>0.0001));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  ERRBA8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Detectamos el número de errores en recepción cuando la señal que
%pretendemos recuperar es una señal digital 8PSK en bucle abierto,
%esto es, sin que esté activa la salida del VCO; pretendemos con esto
%comparar la eficacia de un sistema en el que no se aplique una
%corrección de frecuencia (bucle abierto).
%El número de símbolos recibidos es long, el desfase entre la
%frecuencia de la señal portadora y la de la señal local es nu y la
%relación señal a ruido del canal es snr, en decibelios.


function n=errba(nu,long,snr)


%Factor de caída del pulso conformador gaussiano:
rolloff=0.5;
%La salida del VCO no está activada, siempre estima un offset de
frecuencia nulo:
nuestim=0;
%Secuencia de datos 8PSK transmitida.
[sec,secdif]=datos8(long);
%Duración de cada símbolo:
T=100;
%Señal recibida, ya demodulada y con un offset de frecuencia nu:
rdt=rx(secdif,rolloff,nu,snr);
%La pasamos por el multiplicador:
slp=mult(rdt,long,T,nuestim);
%Salida del filtro adaptado:
ydt=match(slp,rolloff);
%Muestreamos la señal continua:
ydk=muestreo(ydt,T);
%Detección de la secuencia recibida:
[secest,ang]=decisor8(ydk);
%Número de errores en la recepción en bucle abierto, en régimen
permanente:
n=length(find(abs(exp(i*angle(sec(51:long)))-exp((-
1)*i*angle(secest(51:long))))>0.0001));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%  MONTECBAQ.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Función que realiza el análisis de Montecarlo para hallar la
%probabilidad de error del sistema que hemos simulado en bucle
%cerrado; consideramos la transmisión de secuencias de 10000 símbolos,
%y realizamos tantas simulaciones de la recepción de una de esas
%secuencias como requiera el método. El análisis lo realizamos para
%desviaciones de frecuencia del 0%, 2.5%, 5%, 7.5% y 10%, y con ocho
%valores de la relación señal a ruido. Guardamos en memoria los
%vectores obtenidos en la simulación para posteriormente calcular la
%probabilidad de error.


function montecbaq


%Iniciamos los contadores.
t=1;
n=1;
%Longitud de la secuencia de símbolos:
long=10000;
%Vector de desviaciones de frecuencia:
nu=[0,0.00025,0.0005,0.00075,0.001];
%vector que indica el número de iteraciones para cada snr:
num=[1 1 2 5 15 30 100 150];
%Inicializamos los vectores que contendrán los números de errores
%para cada desviación de frecuencia.
nba0e=[0 0 0 0 0 0 0 0];
nba2e=[0 0 0 0 0 0 0 0];
nba5e=[0 0 0 0 0 0 0 0];
nba7e=[0 0 0 0 0 0 0 0];
nba10e=[0 0 0 0 0 0 0 0];

%Iniciamos la simulación:
for snr=7:1:14

   switch snr

%Según el valor de la SNR tendremos un número de iteraciones distinto:
      case 7
         m=num(1);
      case 8
         m=num(2);
      case 9
         m=num(3);
      case 10
         m=num(4);
      case 11
         m=num(5);
      case 12
         m=num(6);
      case 13
         m=num(7);
      case 14
```

```
          m=num(8);
   end

   for n=1:m
%Calculamos el número de símbolos recibidos erróneamente en la
%secuencia:
      nba0=errba(nu(1),long,snr);
      nba0e(t)=nba0e(t)+nba0;
      nba25=errba(nu(2),long,snr);
      nba2e(t)=nba2e(t)+nba25;
      nba5=errba(nu(3),long,snr);
      nba5e(t)=nba5e(t)+nba5;
      nba75=errba(nu(4),long,snr);
      nba7e(t)=nba7e(t)+nba75;
      nba10=errba(nu(5),long,snr);
      nba10e(t)=nba10e(t)+nba10;
%Grabamos el trabajo realizado por el programa en cada iteración:
      save nba0e nba0e
      save nba2e nba2e
      save nba5e nba5e
      save nba7e nba7e
      save nba10e nba10e

   end

end

%Matriz de probabilidades de error; cada fila servirá para construir
%una curva:
pba(1,:)=nba0e./num;
pba(2,:)=nba2e./num;
pba(3,:)=nba5e./num;
pba(4,:)=nba7e./num;
pba(5,:)=nba10e./num;

save pba pba;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%  MONTECBA8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Función que realiza el análisis de Montecarlo para hallar la
%probabilidad de error del sistema 8PSK que hemos simulado en bucle
%abierto; consideramos la transmisión de secuencias de 10000 símbolos,
%y realizamos tantas simulaciones de la recepción de una de esas
%secuencias como requiera el método. El análisis lo realizamos para
%desviaciones de frecuencia del 0%, 1.25%, 2.5%, 3.75% y 5%, y con
%ocho valores de la relación señal a ruido. Guardamos en memoria los
%vectores obtenidos en la simulación para posteriormente calcular la
%probabilidad de error.


function montecba8


%Iniciamos los contadores.
t=1;
n=1;
%Longitud de la secuencia de símbolos:
long=10000;
%Vector de desviaciones de frecuencia:
nu=[0,0.000125,0.00025,0.000375,0.0005];
%vector que indica el número de iteraciones para cada snr:
num=[1 1 2 5 15 50 135 150];
%Inicializamos los vectores que contendrán los números de errores
%para cada desviación de frecuencia.
nba08=[0 0 0 0 0 0 0 0];
nba18=[0 0 0 0 0 0 0 0];
nba28=[0 0 0 0 0 0 0 0];
nba38=[0 0 0 0 0 0 0 0];
nba58=[0 0 0 0 0 0 0 0];

%Iniciamos la simulación:
for snr=13:1:20

   switch snr

%Según el valor de la SNR tendremos un número de iteraciones distinto:
      case 7
         m=num(1);
      case 8
         m=num(2);
      case 9
         m=num(3);
      case 10
         m=num(4);
      case 11
         m=num(5);
      case 12
         m=num(6);
      case 13
         m=num(7);
      case 14
         m=num(8);
```

```
   end

   for n=1:m
%Calculamos el número de símbolos recibidos erróneamente en la
%secuencia:
      nba0=errba8(nu(1),long,snr);
      nba08(t)=nba08(t)+nba0;
      nba1=errba8(nu(2),long,snr);
      nba18(t)=nba18(t)+nba1;
      nba2=errba8(nu(3),long,snr);
      nba28(t)=nba28(t)+nba2;
      nba3=errba8(nu(4),long,snr);
      nba38(t)=nba38(t)+nba3;
      nba5=errba8(nu(5),long,snr);
      nba58(t)=nba58(t)+nba5;
%Grabamos el trabajo realizado por el programa en cada iteración:
      save nba08 nba08
      save nba18 nba18
      save nba28 nba28
      save nba38 nba38
      save nba58 nba58

   end

end


%Matriz de probabilidades de error; cada fila servirá para construir
%una curva:
pba8(1,:)=nba08./num;
pba8(2,:)=nba18./num;
pba8(3,:)=nba28./num;
pba8(4,:)=nba38./num;
pba8(5,:)=nba58./num;

save pba8 pba8;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  MULT2.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Operación de multiplicación de la señal recibida r(t) por la salida
%del VCO. Este bloque de código lo utilizaremos cuando el sistema
%funcione en bucle cerrado.
%Parámetros: señal proveniente del demodulador, rdt; Esta señal puede
%            ser originada por una secuencia de símbolos QPSK o 8-PSK.
%            rdt es la salida del bloque RX.M.
%            Longitud de la secuencia de datos recibida, long;
%            Duración de cada símbolo, T;
%            Offset de frecuencia estimado por el VCO en el periodo
%            anterior, nuestim.
%            lugar que el símbolo ocupa en la secuencia, k.


function slp=mult2(rdt,long,T,nuestim,k)


%Señal que llega del oscilador,
vcosc=exp((-i)*2*pi*nuestim*(((k-1)*T+1):(k*T)));
%Señal de salida del multiplicador:
slp=rdt(((k-1)*T+1):(k*T)).*vcosc;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%  MUESTR2.M  %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Esta función realiza el muestreo de la señal continua que sale del
%filtro adaptado del sistema de sincronización de frecuencia.
%Suponemos la temporización perfectamente conocida, se ha recuperado
%sin errores en algún sistema anterior; así, sabemos que tenemos que
%obtener una muestra de la señal y(t) cada T unidades de tiempo; se
%realiza el muestreo con un cierto retraso tau (constante para todas
%las muestras), que estimamos es el igual al retardo debido al canal y
%que ha sido obviado por ser un parámetro también extraído en la etapa
%de temporización.
%Este bloque lo utilizamos en bucle cerrado, tanto para QPSK como para
%8-PSK.
%Parámetros: ydt: salida(continua) del filtro adaptado; es la señal
%           que queremos muestrear.
%           T: periodo de muestro, es el tiempo de símbolo.
```

```
function ydk=muestr2(ydt,T)
```

```
%Llega una señal y(t) durante un tiempo T, y la muestreo en el
instante kT.
   ydk=ydt(T);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  DECBC.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Bloque que decide qué símbolo QPSK de los cuatro posibles en nuestro
%alfabeto ha sido recibido. A la salida devuelve la fase de la
%secuencia detectada todavía codificada diferencialmente, ang, y la
%secuencia de datos estimada QPSK tras realizar la decodificación
%diferencial.
%Este decisor se utiliza en las operaciones en bucle cerrado.
%Parámetros: señal de entrada al bucle debidamente filtrada y
%            muestreada, ydk.
%            fase del símbolo QPSK diferencial en el instante k-1,
%            fasant. El valor de este parámetro fue el parámetro de
%            salida de este mismo bloque ang en el instante anterior.


function [secest,ang]=decbc(ydk,fasant)


%Calculamos la fase de la secuencia recibida:
fase=angle(ydk);
%Realizamos la operación de decisión:
ang=0*[abs(fase)<=(pi/4)]+(pi/2)*[abs(fase-pi/2)<(pi/4)]+(pi)*[(pi-
fase)<=(pi/4)]-(pi/2)*[abs(fase+pi/2)<(pi/4)]-
(pi)*[(fase+pi)<=(pi/4)];
%Operación inversa a la codificación diferencial de la señal QPSK:
fasestim=(ang-fasant);
%Obtenemos la secuencia estimada por el decisor.
secest=exp((-i)*fasestim);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  DECBC8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Bloque que decide qué símbolo 8PSK de los ocho posibles en nuestro
%alfabeto ha sido recibido. A la salida devuelve la fase de la
%secuencia detectada todavía codificada diferencialmente, ang, y la
%secuencia de datos estimada 8PSK tras realizar la decodificación
%diferencial.
%Este decisor se utiliza en las operaciones en bucle cerrado.
%Parámetros: señal de entrada al bucle debidamente filtrada y
%            muestreada, ydk.
%            fase del símbolo 8PSK diferencial en el instante k-1,
%            fasant. El valor de este parámetro fue el parámetro de
%            salida de este mismo bloque ang en el instante anterior.


function [secest,ang]=decbc8(ydk,fasant)


%Calculamos la fase de la secuencia recibida:
fase=angle(ydk);
%Realizamos la operación de decisión:
ang1=0*[abs(fase)<=(pi/8)]+(pi/2)*[abs(fase-pi/2)<=(pi/8)]+(pi)*[(pi-
fase)<=(pi/8)]-(pi/2)*[abs(fase+pi/2)<=(pi/8)]-
(pi)*[(fase+pi)<=(pi/8)];
ang2=(pi/4)*[abs(fase-pi/4)<(pi/8)]+(3*pi/4)*[abs(fase-
(3*pi/4))<(pi/8)]-(pi/4)*[abs(fase+pi/4)<(pi/8)]-
(3*pi/4)*[abs(fase+(3*pi/4))<(pi/8)];
ang=ang1+ang2;
%Operación inversa a la codificación diferencial de la señal QPSK:
fasestim=(ang-fasant);
%Obtenemos la secuencia estimada por el decisor.
secest=exp((-i)*fasestim);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  ERR2.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Bloque generador de error. Este bloque da lugar a una función de
%error obtenida según el algoritmo de Sari y Moridi, para señales
%QPSK. Como parámetro tenemos la señal que recoge la diferencia de
%fase entre el símbolo recibido y el símbolo ideal registrado por el
%bloque detector, esto es, la salida del PD, zdk, y el parámetro beta
%que caracteriza al generador de error QPSK. También necesita
%información sobre el instante k en el que estamos, para distinguir
%entre el primer símbolo de la secuencia y los restantes.
%Este programa, junto con ZED.M, simula el comportamiento del PFD.
%Utilizamos este bloque para las simulaciones en bucle cerrado.



function edk=err2(zdk,k,ant,beta)


%Damos un valor al parámetro beta, que acotará el valor de salida de
%err2.
teta=beta*pi/4;
%Diferencia de fase entre la señal recibida y la salida del símbolo
%detectado:
e=angle(zdk);
%Generamos a partir de esta fase y del valor que obtuvimos para la
%iteración(k-1), teniendo en cuenta si se trata o no del primer
%símbolo recibido, el valor del error que entrará en el filtro LP
%colocado tras este bloque, según el algoritmo citado:
edk=e*[abs(e)<=alfa]+sign(e)*alfa*[abs(e)>alfa]*[k==1]+ant*[abs(e)>alf
a]*[k>1];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  ERR28.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Bloque generador de error. Este bloque da lugar a una función de
%error obtenida según el algoritmo de Sari y Moridi, para señales
%8PSK. Como parámetro tenemos la señal que recoge la diferencia de
%fase entre el símbolo recibido y el símbolo ideal registrado por el
%bloque detector, esto es, la salida del PD, zdk, y el parámetro beta
%que caracteriza al generador de error 8PSK. También necesita
%información sobre el instante k en el que estamos, para distinguir
%entre el primer símbolo de la secuencia y los restantes.
%Este programa, junto con ZED.M, simula el comportamiento del PFD.
%Utilizamos este bloque para las simulaciones en bucle cerrado.



function edk=err28(zdk,k,ant,beta)


%Damos un valor al parámetro beta, que acotará el valor de salida de
%err2.
teta=beta*pi/8;
%Diferencia de fase entre la señal recibida y la salida del símbolo
%detectado:
e=angle(zdk);
%Generamos a partir de esta fase y del valor que obtuvimos para la
%iteración(k-1), teniendo en cuenta si se trata o no del primer
%símbolo recibido, el valor del error que entrará en el filtro LP
%colocado tras este bloque, según el algoritmo citado:
edk=e*[abs(e)<=alfa]+sign(e)*alfa*[abs(e)>alfa]*[k==1]+ant*[abs(e)>alf
a]*[k>1];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  FILTRO2.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Este programa simula un filtro LP colocado a la salida del generador
%de error; Entra como parámetro (e) la señal edk, y  va calculando la
%media entre todas las muestras recibidas hasta el momento.
%Se puede utilizar para cualquiera de las dos modulaciones PSK.
%La representación de la salida de este filtro frente a la desviación
%de frecuencia será la característica de salida del PFD, S(fd), la
%cual tendremos que calcular.
%Utilizamos este bloque para las simulaciones en bucle abierto.


function media=filtro2(e);


suma=0;
for i=1:length(e)
   suma=(suma+e(i));
end;
media=suma/length(e);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%  VCOBUC.M  %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Obtenemos la frecuencia con la que oscilará el VCO colocado en
%en el sistema QPSK, en función de la tensión de entrada al oscilador,
%que será la media extraída en el filtro LP. Para ello necesitamos
%como parámetros la salida del filtro del bucle, media, y la
%desviación de frecuencia estimada en el instante k-1, nuant.


function nuestim=vcobuc(media,nuant)


nuestim=0.2*nuant+0.0024*media;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%  VCOBUC8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Obtenemos la frecuencia con la que oscilará el VCO colocado en
%en el sistema 8PSK, en función de la tensión de entrada al oscilador,
%que será la media extraída en el filtro LP. Para ello necesitamos
%como parámetros la salida del filtro del bucle, media, y la
%desviación de frecuencia estimada en el instante k-1, nuant.


function nuestim=vcobuc(media,nuant)


nuestim=0.2*nuant+0.0026*media;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%  BUCLECERR.M  %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Sistema QPSK completo, en bucle cerrado. Este sistema se coloca
% detrás de la etapa de demodulación, de la cual nos llega una
% secuencia de símbolos que pretendemos identificar. La secuencia que
% nos transmiten es la variable obtenida sec; La información de estos
% símbolos la recibimos en la fase de la señal obtenida. Esta fase se
% encuentra distorsionada por la presencia de una portadora residual
% nu, fruto de una demodulación no exacta, que modula a muy baja
% frecuencia la señal recibida. Suponemos en todo el proceso que
% conocemos perfectamente los datos de temporización, esto es,
% conocemos T y tau, siendo tau el retraso introducido por el canal,
% que ha sido obviado por el hecho de haber considerado que la
% sincronización del reloj es perfecta. por nuestro sistema una vez
% detectado.
% Contemplamos la presencia de ruido aditivo gaussiano en el canal
% aéreo.
```

```
function n=buclecerr(nu,long,snr)
```

```
%Duración de cada símbolo:
T=100;
%La secuencia de datos codificada diferencialmente es secdif:
[sec,secdif]=datos(long);
rolloff=0.5;
%Inicialmente, la frecuencia estimada por el VCO es 0:
nuestim=0;
%La señal que entra en nuestro sistema es r(t):
rdt=rx(secdif,rolloff,nu,snr);
ant=0;
fasant=0;
nuant=0;
for k=1:long
%La señal se multiplica por la salida del VCO.
   slp=mult2(rdt,long,T,nuestim,k);
%Se pasa la señal con el offset de frecuencia corregido por un filtro
%adaptado.
   ydt=match(slp,rolloff);
%Se realiza un muestreo de la señal de salida del filtro.
   ydk(k)=muestr2(ydt,T);
%La señal muestreada se da como entrada al decisor para que reconozca
%el símbolo recibido.
   [secest(k),ang]=decbc(ydk(k),fasant);
   fasant=ang;
   di(k)=sec(k).*secest(k);
%Nos quedamos con el error entre las fases del símbolo recibido y
%del detectado.
   zdk=zed(ydk(k),ang);
%Pasamos esta muestra obtenida por el generador de error;
   e(k)=err2(zdk,k,ant);
```

```
%Guardamos en un registro el valor obtenido del generador de error,
%que puede ser útil para el algoritmo cuando en el siguiente periodo
%llegue otra muestra.
   ant=e(k);
%Pasamos el resultado del generador de error por un filtro de
%media
   media=filtro2(e);
   m(k)=media;
%En función de la media tendremos la respuesta del VCO.
   nuestim=vcobuc(media,nuant);
   nuant=nuestim;
   fd(k)=nu-nuestim;
end
%Calculamos el número de errores de recepción en régimen permanente.
n=length(find(abs(exp(i*angle(sec(51:long)))-exp((-
1)*i*angle(secest(51:long))))>0.001));

%subplot(2,2,1), plot(e), title('curva error fase')
%subplot(2,2,2),plot(fd), title('fd')
%subplot(2,2,3), plot(m),title('media')
%subplot(2,2,4), plot(angle(di)),title('error Tx-Rx')
```

NOTA: Al final se han añadido cuatro líneas de código que se encuentran desactivadas al aparecer como comentarios; estas líneas, cuando están habilitadas, muestran por pantalla resultados de simulaciones como los que vemos en la figura 5.65, muy útiles para analizar el comportamiento de los distintos bloques del bucle tanto a medida que el sistema va sincroniza la frecuencia como en estado estacionario.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%  BUCLECERR8.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Sistema 8PSK completo, en bucle cerrado. Este sistema se coloca
% detrás de la etapa de demodulación, de la cual nos llega una
% secuencia de símbolos que pretendemos identificar. La secuencia que
% nos transmiten es la variable obtenida sec; La información de estos
% símbolos la recibimos en la fase de la señal obtenida. Esta fase se
% encuentra distorsionada por la presencia de una portadora residual
% nu, fruto de una demodulación no exacta, que modula a muy baja
% frecuencia la señal recibida. Suponemos en todo el proceso que
% conocemos perfectamente los datos de temporización, esto es,
% conocemos T y tau, siendo tau el retraso introducido por el canal,
% que ha sido obviado por el hecho de haber considerado que la
% sincronización del reloj es perfecta. por nuestro sistema una vez
% detectado.
% Contemplamos la presencia de ruido aditivo gaussiano en el canal
% aéreo.


function n=buclecerr8(nu,long,snr)


%Duración de cada símbolo:
T=100;
%La secuencia de datos codificada diferencialmente es secdif:
[sec,secdif]=datos8(long);
rolloff=0.5;
%Inicialmente, la frecuencia estimada por el VCO es 0:
nuestim=0;
%La señal que entra en nuestro sistema es r(t):
rdt=rx(secdif,rolloff,nu,snr);
ant=0;
fasant=0;
nuant=0;
for k=1:long
%La señal se multiplica por la salida del VCO.
   slp=mult2(rdt,long,T,nuestim,k);
%Se pasa la señal con el offset de frecuencia corregido por un filtro
%adaptado.
   ydt=match(slp,rolloff);
%Se realiza un muestreo de la señal de salida del filtro.
   ydk(k)=muestr2(ydt,T);
%La señal muestreada se da como entrada al decisor para que reconozca
%el símbolo recibido.
   [secest(k),ang]=decbc8(ydk(k),fasant);
   fasant=ang;
   di(k)=sec(k).*secest(k);
%Nos quedamos con el error entre las fases del símbolo recibido y
%del detectado.
   zdk=zed(ydk(k),ang);
%Pasamos esta muestra obtenida por el generador de error;
   e(k)=err28(zdk,k,ant);
```

```
%Guardamos en un registro el valor obtenido del generador de error,
%que puede ser útil para el algoritmo cuando en el siguiente periodo
%llegue otra muestra.
   ant=e(k);
%Pasamos el resultado del generador de error por un filtro de
%media
   media=filtro2(e);
   m(k)=media;
%En función de la media tendremos la respuesta del VCO.
   nuestim=vcobuc8(media,nuant);
   nuant=nuestim;
   fd(k)=nu-nuestim;
end
%Calculamos el número de errores de recepción en régimen permanente.
n=length(find(abs(exp(i*angle(sec(51:long)))-exp((-
1)*i*angle(secest(51:long))))>0.001));

%subplot(2,2,1), plot(e), title('curva error fase')
%subplot(2,2,2),plot(fd), title('fd')
%subplot(2,2,3), plot(m),title('media')
%subplot(2,2,4), plot(angle(di)),title('error Tx-Rx')
```

NOTA: Al final se han añadido cuatro líneas de código que se encuentran desactivadas al aparecer como comentarios; estas líneas, cuando están habilitadas, muestran por pantalla resultados de simulaciones como los que vemos en la figura 5.67, muy útiles para analizar el comportamiento de los distintos bloques del bucle tanto a medida que el sistema va sincroniza la frecuencia como en estado estacionario.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  MONTECQ.M  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%Función que realiza el análisis de Montecarlo para hallar la
%probabilidad de error del sistema que hemos simulado en bucle
%cerrado; consideramos la transmisión de secuencias de 10000 símbolos,
%y realizamos tantas simulaciones de la recepción de una de esas
%secuencias como requiera el método. El análisis lo realizamos para
%desviaciones de frecuencia del 0%, 2.5%, 5%, 7.5% y 10%, y con ocho
%valores de la relación señal a ruido. Guardamos en memoria los
%vectores obtenidos en la simulación para posteriormente calcular la
%probabilidad de error.


function montecq


%Iniciamos los contadores.
t=1;
n=1;
%Longitud de la secuencia de símbolos:
long=10000;
%Vector de desviaciones de frecuencia:
nu=[0,0.00025,0.0005,0.00075,0.001];
%vector que indica el número de iteraciones para cada snr:
num=[1 1 2 5 15 30 100 150];
%Inicializamos los vectores que contendrán los números de errores
%para cada desviación de frecuencia.
nbc0e=[0 0 0 0 0 0 0 0];
nbc2e=[0 0 0 0 0 0 0 0];
nbc5e=[0 0 0 0 0 0 0 0];
nbc7e=[0 0 0 0 0 0 0 0];
nbc10e=[0 0 0 0 0 0 0 0];

%Iniciamos la simulación:
for snr=7:1:14

   switch snr

%Según el valor de la SNR tendremos un número de iteraciones distinto:
      case 7
         m=num(1);
      case 8
         m=num(2);
      case 9
         m=num(3);
      case 10
         m=num(4);
      case 11
         m=num(5);
      case 12
         m=num(6);
      case 13
         m=num(7);
      case 14
```

```
          m=num(8);
   end

   for n=1:m
%Calculamos el número de símbolos recibidos erróneamente en la
%secuencia:
      nbc0=buclecerr(nu(1),long,snr);
      nbc0e(t)=nbc0e(t)+nbc0;
      nbc25=buclecerr(nu(2),long,snr);
      nbc2e(t)=nbc2e(t)+nbc25;
      nbc5=buclecerr(nu(3),long,snr);
      nbc5e(t)=nbc5e(t)+nbc5;
      nbc75=buclecerr(nu(4),long,snr);
      nbc7e(t)=nbc7e(t)+nbc75;
      nbc10=buclecerr(nu(5),long,snr);
      nbc10e(t)=nbc10e(t)+nbc10;
%Grabamos el trabajo realizado por el programa en cada iteración:
      save nbc0e nbc0e
      save nbc2e nbc2e
      save nbc5e nbc5e
      save nbc7e nbc7e
      save nbc10e nbc10e

   end

end

%Matriz de probabilidades de error; cada fila servirá para construir
%una curva:
p(1,:)=nbc0e./num;
p(2,:)=nbc2e./num;
p(3,:)=nbc5e./num;
p(4,:)=nbc7e./num;
p(5,:)=nbc10e./num;

save p p;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Función que realiza el análisis de Montecarlo para hallar la
%probabilidad de error del sistema 8PSK que hemos simulado en bucle
%cerrado; consideramos la transmisión de secuencias de 10000 símbolos,
%y realizamos tantas simulaciones de la recepción de una de esas
%secuencias como requiera el método. El análisis lo realizamos para
%desviaciones de frecuencia del 0%, 1.25%, 2.5%, 3.75% y 5%, y con
%ocho valores de la relación señal a ruido. Guardamos en memoria los
%vectores obtenidos en la simulación para posteriormente calcular la
%probabilidad de error.


function montec8


%Iniciamos los contadores.
t=1;
n=1;
%Longitud de la secuencia de símbolos:
long=10000;
%Vector de desviaciones de frecuencia:
nu=[0,0.000125,0.00025,0.000375,0.0005];
%vector que indica el número de iteraciones para cada snr:
num=[1 1 2 5 15 50 135 150];
%Inicializamos los vectores que contendrán los números de errores
%para cada desviación de frecuencia.
nbc08=[0 0 0 0 0 0 0 0];
nbc18=[0 0 0 0 0 0 0 0];
nbc28=[0 0 0 0 0 0 0 0];
nbc38=[0 0 0 0 0 0 0 0];
nbc58=[0 0 0 0 0 0 0 0];

%Iniciamos la simulación:
for snr=13:1:20

   switch snr

%Según el valor de la SNR tendremos un número de iteraciones distinto:
      case 7
         m=num(1);
      case 8
         m=num(2);
      case 9
         m=num(3);
      case 10
         m=num(4);
      case 11
         m=num(5);
      case 12
         m=num(6);
      case 13
         m=num(7);
      case 14
         m=num(8);
```

```
   end

   for n=1:m
%Calculamos el número de símbolos recibidos erróneamente en la
%secuencia:
      nbc0=buclecerr8(nu(1),long,snr);
      nbc08(t)=nbc08(t)+nbc0;
      nbc1=buclecerr8(nu(2),long,snr);
      nbc18(t)=nbc18(t)+nbc1;
      nbc2=buclecerr8(nu(3),long,snr);
      nbc28(t)=nbc28(t)+nbc2;
      nbc3=buclecerr8(nu(4),long,snr);
      nbc38(t)=nbc38(t)+nbc3;
      nbc5=buclecerr8(nu(5),long,snr);
      nbc58(t)=nbc58(t)+nbc5;
%Grabamos el trabajo realizado por el programa en cada iteración:
      save nbc08 nbc08
      save nbc18 nbc18
      save nbc28 nbc28
      save nbc38 nbc38
      save nbc58 nbc58

   end

end


%Matriz de probabilidades de error; cada fila servirá para construir
%una curva:
p8(1,:)=nbc08./num;
p8(2,:)=nbc18./num;
p8(3,:)=nbc28./num;
p8(4,:)=nbc38./num;
p8(5,:)=nbc58./num;

save p8 p8;
```