TRABAJO FIN DE GRADO

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Escuela de Ciencias Técnicas e Ingenierías

Módulo de ecualizador adaptativo para mitigar el efecto de la propagación multitrayecto para el laboratorio virtual de comunicaciones comLAB

Autor: Rubén Guzmán Serrano

Directores: José María Díaz Nafría y Antonio Jesús Muñoz Montoro

Curso 2024/2025

Índice de contenidos

RESUMEN

En este Trabajo Fin de Grado se aborda el modelado de la propagación multitrayecto en un sistema de transmisión digital, así como el de la técnica para mitigar su efecto: la ecualización adaptativa. Para ese propósito se recurrirá a comLAB, laboratorio virtual de comunicaciones desarrollado en MATLAB/Simulink, que servirá de base para implementar los modelos correspondientes a dichos procesos en el sistema de transmisión. Se pretende mostrar cuál es el efecto de la propagación multitrayecto sobre la señal transmitida y cómo de efectiva es la ecualización adaptativa para su compensación.

Palabras clave

Propagación multitrayecto, Ecualización adaptativa, Sistema de transmisión digital, Simulación en comunicaciones, MATLAB, Simulink, comLAB, Comunicaciones inalámbricas.

ABSTRACT

In this Bachelor's thesis, the modeling of multipath propagation in a digital transmission system is addressed, as well as the technique to mitigate its effect: adaptive equalization. For this purpose, comLAB, a virtual communications laboratory developed in MATLAB/Simulink, will be used as the basis for implementing the models corresponding to these processes within the transmission system. The aim is to show the effect of multipath propagation on the transmitted signal and how effective adaptive equalization is in compensating for it.

Keywords

Multipath propagation, Adaptive equalization, Digital transmission system, Communication simulation, MATLAB, Simulink, comLAB, Wireless communications.

CAPÍTULO I. INTRODUCCIÓN

1.1. Motivación del proyecto

El estudio de la propagación multitrayecto es una parte fundamental en el diseño de los sistemas de comunicación inalámbricos. La reflexión, refracción y dispersión de las ondas electromagnéticas en obstáculos provocan que estas lleguen al receptor a través de múltiples trayectos con distintos retardos y atenuaciones. La superposición de estas componentes de señal en el receptor puede producir desvanecimiento selectivo en frecuencia e interferencia intersimbólica, dificultando el proceso de demodulación e incrementando las tasas de error.

Además, cabe considerar que las condiciones del multitrayecto son variantes en el tiempo, de forma que no basta con compensar la respuesta del canal con una ecualización estática. Para contrarrestar estas variaciones se suelen incorporar técnicas de ecualización adaptativa en el receptor. Estas técnicas permiten ajustar dinámicamente los coeficientes del filtro ecualizador, de forma que el receptor puede adaptarse a las condiciones cambiantes del canal de comunicación.

En este contexto, el presente Trabajo de Fin de Grado se centra en analizar el impacto de la propagación multitrayecto sobre un sistema de transmisión digital, y comprobar la eficacia de la ecualización adaptativa para mitigar sus efectos. Para ello, simularemos un ciclo de transmisión digital completo en el laboratorio virtual comLAB, desde el muestreo de la señal en origen, hasta su reproducción en el destino (ver figura 1).

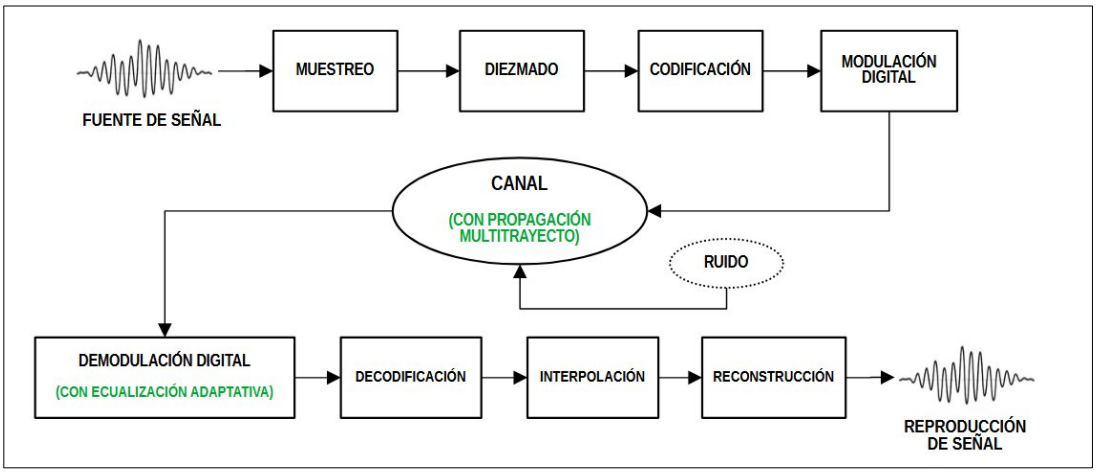

Figura 1. Ciclo de transmisión digital, contemplando la propagación multitrayecto y la ecualización adaptativa.

comLAB es un laboratorio virtual de comunicaciones, implementado en Simulink, donde se representan las distintas etapas del sistema de transmisión mediante bloques interconectados. La representación gráfica del circuito mediante bloques facilita la comprensión del flujo de la señal y permite realizar modificaciones de forma más intuitiva. En este marco, como objetivo de este proyecto, se diseñarán e incorporarán los bloques necesarios para simular la propagación multitrayecto y la ecualización adaptativa.

Modelar estos fenómenos en MATLAB/Simulink nos va a permitir reproducir escenarios de propagación, analizar el efecto de distintas condiciones del canal y evaluar el algoritmo de ecualización adaptativa, sin necesidad de recurrir a equipos físicos. De este modo, dispondremos de una herramienta para el análisis técnico y un recurso que nos facilitará la comprensión de los conceptos teóricos de la transmisión digital.

En primer lugar, se recreará el proceso de transmisión mediante funciones ejecutadas desde la consola de MATLAB, lo que nos permitirá realizar un análisis de la señal en bloque y comprobar la validez de los modelos. Además, las funciones implementadas en código MATLAB servirán de base para la traslación a Simulink de dichos modelos. Posteriormente, con el propósito de realizar una simulación que se acerque más al análisis en tiempo real, pasaremos al entorno de comLAB.

Previamente, antes del desarrollo de las simulaciones, estableceremos el marco teórico que ayude a entender los conceptos fundamentales que se abordan en este trabajo: el desvanecimiento multitrayecto y las técnicas de ecualización adaptativa. A su vez, se incluirá un estado de la cuestión que recoja la evolución de la simulación por ordenador en el ámbito de las telecomunicaciones, destacando el papel de MATLAB/Simulink, y justificando su uso para este proyecto.

1.2. Objetivos

A continuación, se definen los objetivos del proyecto:

- Definición del marco teórico que sirva de base a los elementos de simulación que se van a diseñar: desvanecimiento multitrayecto y ecualización adaptativa. Clarificación de estos conceptos en glossaLAB y vinculación con los bloques correspondientes diseñados en comLAB.

- Diseño en MATLAB de las funciones que modelan la propagación multitrayecto y la ecualización adaptativa. Integración de estas funciones en la simulación de un ciclo de transmisión y análisis de su efecto.

- Diseño en Simulink de los bloques que modelan la propagación multitrayecto y la ecualización adaptativa. Integración de estos bloques en un ciclo de transmisión en comLAB y análisis de su efecto.

CAPÍTULO II. MARCO TEÓRICO

2.1. Ecualización del canal

En un sistema de transmisión digital, la **ecualización** es una técnica utilizada en el receptor para compensar las distorsiones que sufre la señal durante su propagación por el canal de comunicación. Su objetivo principal es revertir o mitigar los efectos del canal para que la señal recibida se parezca lo máximo posible a la señal transmitida.

Partimos de un sistema de transmisión digital como el de la figura 2 (Sklar y Harris, 2021), donde una señal digital $x(n)$ está representada por un tren de impulsos separados por intervalos de tiempo T. Esta señal se transforma, mediante la modulación, en un tren de pulsos de una forma determinada, adecuada para atravesar el canal de transmisión, que estará afectado por el ruido. Cada pulso representa un símbolo, y la separación entre ellos es el periodo de símbolo T.

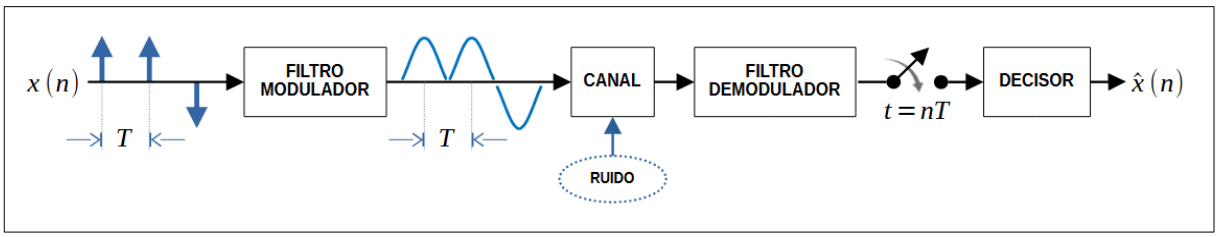

Figura 2. Sistema de transmisión digital.

En recepción, la señal pasa por un filtro demodulador, cuyo objetivo es transformar la señal recibida de forma que, al ser muestreada posteriormente en los instantes de tiempo óptimos $n\,T$, se maximice la relación señal a ruido (SNR) y se minimice la interferencia entre símbolos (ISI), con el objetivo de minimizar la probabilidad de error en la fase de decisión, que nos devuelve los valores estimados de la señal a la salida $\hat{x}(n)$.

La pareja de filtros de modulación y demodulación está diseñada para que la ISI sea mínima. Para conseguir esto, se elige como demodulador un filtro adaptado al pulso generado por el modulador, de forma que si el modulador genera pulsos basados en un pulso base $s(t)$, la respuesta al impulso del demodulador sea $h(t) = k\, s(T - t)$. Este diseño supone que el canal no introduce distorsión.

Cuando el canal introduce distorsión, esto tiene consecuencia directa sobre la ISI. Para compensar el efecto del canal se recurre a la ecualización (ver figura 3).

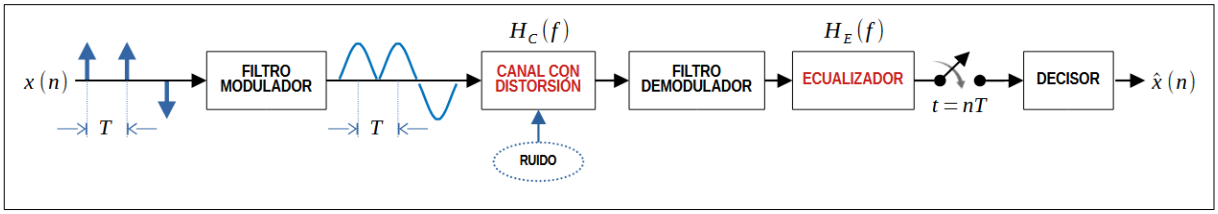

Figura 3. Sistema de transmisión digital con ecualización de canal.

Si podemos caracterizar el canal como un sistema con respuesta en frecuencia $H_C(f) = |H_C(f)|\, e^{j\theta_c(f)}$, para compensar su efecto, el filtro ecualizador debería tener la siguiente función de transferencia:

$$H_E(f) = H_C(f)^{-1} = \frac{1}{|H_C(f)|}\, e^{-j\theta_c(f)}$$

Este resultado es óptimo, como veremos más adelante, solo cuando se considera un canal no ruidoso.

A partir de la respuesta al impulso estimada del canal, es posible diseñar un ecualizador que compense sus efectos. Este proceso suele comenzar con el envío de una secuencia de entrenamiento conocida por el receptor. Comparando la señal recibida con la esperada, el sistema puede estimar los coeficientes que caracterizan la distorsión introducida por el canal.

Los ecualizadores pueden ser lineales, si únicamente incluyen elementos de avance, conocidos como ecualizadores transversales, o bien no lineales, si además incorporan elementos de retroalimentación, como ocurre en los ecualizadores de decisión retroalimentada (DFE, *Digital Feedback Equalizers*). Estos últimos mejoran el rendimiento al utilizar decisiones anteriores para cancelar la ISI, aunque pueden verse afectados por errores de propagación. En este proyecto nos centraremos en la versión adaptativa de un ecualizador transversal.

El ecualizador transversal consiste en un filtro FIR (*Finite Impulse Response*, respuesta finita al impulso) que procesa la señal recibida aplicando una combinación lineal ponderada de muestras pasadas. El diseño de este tipo de ecualizador puede abordarse mediante dos enfoques principales.

Por una parte, puede adoptarse una solución determinista, conocida como *Zero-Forcing*, que busca anular por completo la ISI en los instantes de muestreo, forzando a que la respuesta total del sistema formado por el canal y el ecualizador sea ideal. Sin embargo, esta solución puede amplificar excesivamente el ruido si la respuesta en frecuencia del canal tiene ceros muy pronunciados, que se intentarán compensar con altas ganancias en el ecualizador que también se aplicarán sobre el ruido.

Por otra parte, se puede optar por una solución estadística, denominada MMSE (*Minimum Mean*

Square Error, mínimo error cuadrático medio*)* , que no elimina completamente la ISI, pero encuentra un compromiso óptimo entre la cancelación de la interferencia y la amplificación del ruido. Este enfoque minimiza el valor esperado del error cuadrático entre la señal transmitida y la señal estimada, ofreciendo mejor desempeño en canales ruidosos.

Este tipo de soluciones suponen que el canal es invariante en el tiempo, pero en muchos de los escenarios reales esto no se cumple. En entornos cambiantes, la ecualización tradicional basada en una caracterización fija del canal resulta insuficiente. Por ello, es necesario recurrir a técnicas adaptativas, capaces de ajustar dinámicamente los coeficientes del ecualizador en función de las condiciones del canal. Este tipo de estrategias serán abordadas en el apartado 2.3, pero antes se introduce un fenómeno clave en este contexto: el desvanecimiento por propagación multitrayecto.

2.2. Desvanecimiento multitrayecto

En el análisis de un sistema de comunicación radioeléctrico, uno de los fenómenos que se debe considerar en la caracterización del canal es el producido cuando las señales transmitidas llegan al receptor a través de múltiples trayectorias, debidas a reflexiones, difracciones y dispersiones en el entorno (ver figura 4). La recepción de estas señales, sumadas a la onda directa con distintos retardos, amplitudes y fases, puede degradar la calidad del enlace. A este efecto de degradación de la señal se le conoce como **desvanecimiento multitrayecto**.

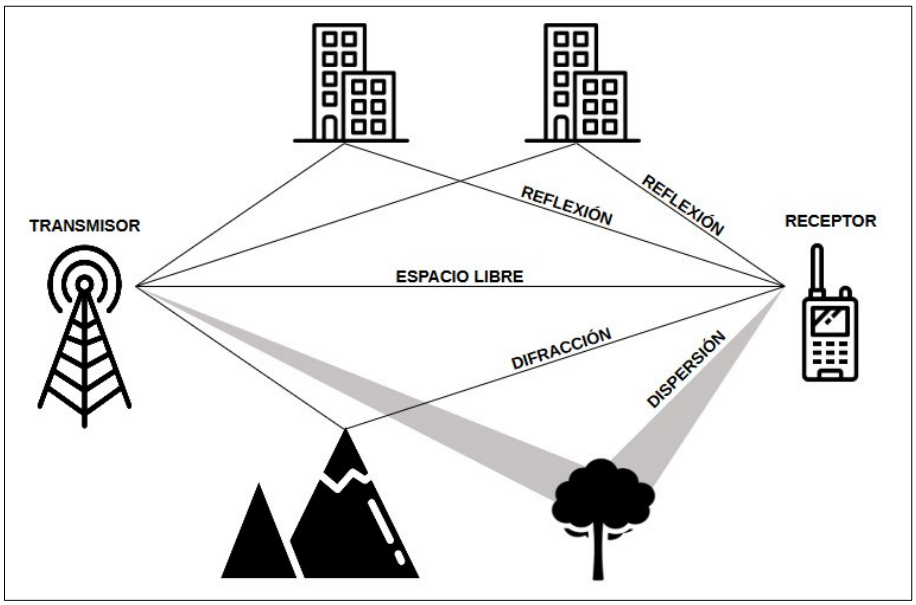

Figura 4. Propagación multitrayecto.

La consecuencia directa de la recepción de señales con distintas amplitudes y retardos es la posible atenuación de la potencia recibida, por la recepción de señales en contrafase. Pero además, hay que considerar un factor que agrava la degradación del enlace en la propagación multitrayecto: la dispersión temporal. En una transmisión digital, la llegada de las señales retardadas va a producir un ensanchamiento de los símbolos recibidos que va a aumentar considerablemente la interferencia entre símbolos y en consecuencia los errores en la detección de estos. Sumado a eso, esta dispersión temporal influye directamente en la respuesta en frecuencia del canal. El aumento de la dispersión temporal reduce el ancho de banda de coherencia del canal, que se traduce en una respuesta más selectiva en frecuencia.

El estudio del efecto del multitrayecto es especialmente sensible en los servicios de comunicaciones móviles, donde, además de la impredecibilidad de los condicionantes que afectan a la caracterización del canal, se añade que el receptor puede estar en movimiento. En las bandas de frecuencia utilizadas por estos sistemas, del orden de microondas, las longitudes de onda son centimétricas, por lo que pequeñas variaciones de posición del receptor suponen cambios bruscos de fase, que pueden llevar a desvanecimientos súbitos. Por ello, se considera que el canal varía en el tiempo de forma aleatoria. Para modelar su respuesta se recurre a procesos aleatorios dependientes del tiempo.

2.2.1. Caracterización de la propagación en comunicaciones móviles

Para la caracterización del canal en comunicaciones móviles se puede distinguir entre el *desvanecimiento a gran escala* y el *desvanecimiento a pequeña escala* (Sklar y Harris, 2021).

Se considera como *desvanecimiento a gran escala*, la variación de la señal recibida considerando las pérdidas por propagación en espacio libre y las pérdidas producidas por la interposición de obstáculos de gran tamaño respecto a la longitud de onda. Este desvanecimiento tiene variaciones lentas sobre su media y depende principalmente de la distancia entre transmisor y receptor.

Hata (1980), a partir del trabajo previo de Okumura (1968), desarrolló un modelo matemático para las pérdidas de propagación en canales multitrayecto en función del entorno. A partir de este modelo, se pueden representar las pérdidas del enlace por *desvanecimiento a gran escala*, como una variable aleatoria L_P dependiente de la distancia d:

$$L_P(d) = L_S(d_0) + 10\, n \log\left(\frac{d}{d_0}\right) + X_\sigma \quad [dB]$$

siendo d_0 una distancia de referencia en campo lejano, sobre la que se calculan las pérdidas por propagación en espacio libre L_S, según $L_S = (4\,\pi\,d/\lambda)^2$; n es el exponente de pérdidas de propagación, que depende del tipo de entorno (rural, suburbano, urbano), la frecuencia y la altura de las antenas; y X_σ es una variable aleatoria con distribución gaussiana de media 0, que representa las pérdidas específicas por la topología del área.

El *desvanecimiento a pequeña escala* se refiere a los cambios súbitos de nivel de señal motivados por pequeños cambios de posición del receptor, de hasta media longitud de onda. Las variaciones de señal por estos pequeños cambios se modelan con diferentes tipos de distribuciones de probabilidad dependiendo de la visibilidad entre receptor y emisor:

- *Desvanecimiento de Rayleigh.* Cuando no hay visibilidad directa, no hay un componente dominante en el conjunto de señales recibidas de cada trayectoria, como suele ocurrir en entornos altamente urbanizados o en entornos *indoor*, donde tenemos un alto número de ondas reflejadas y no hay onda directa. En este caso, se modela estadísticamente la señal recibida con una distribución de probabilidad de Rayleigh.

- *Desvanecimiento de Rice.* En caso de no estar obstaculizada la línea de visión directa entre receptor y emisor, la onda directa es un componente dominante en el conjunto de señales recibidas de cada trayectoria. En este caso, se modela el nivel de señal recibida con una distribución de probabilidad de Rice.

La suma del *desvanecimiento a gran escala* y el *desvanecimiento a pequeña escala* caracteriza el nivel de señal en recepción en escenarios de comunicaciones móviles.

2.2.2. Dispersión temporal

La propagación por diferentes caminos va a producir que una señal transmitida llegue a su destino en diferentes instantes de tiempo. Se define el *retardo máximo* T_m como el tiempo transcurrido entre la recepción de la primera versión de la señal y la última, suponiendo un umbral de potencia por debajo del cuál se desprecian las posteriores (Sklar y Harris, 2021) .

Se define el *ancho de banda de coherencia* f_0 como el rango de frecuencias donde el canal deja pasar todas las componentes de la señal con la misma amplitud y fase lineal, por tanto, las componentes pertenecientes al mismo rango son afectadas por el canal de la mima forma ante el desvanecimiento. Como aproximación se puede considerar la siguiente relación entre el *retardo máximo* y el *ancho de*

banda de coherencia: $f_0 \approx 1/T_m$.

Suponiendo una transmisión digital en la que la señal transmitida tiene un símbolo de periodo T_s y ancho de banda aproximado $W \approx 1/T_s$, podemos clasificar el desvanecimiento en función de la relación entre el canal y el símbolo transmitido de la siguiente forma:

- *Desvanecimiento selectivo en frecuencia* (cuando $T_m > T_s$, $f_0 < W$): Las componentes de la señal multitrayecto se extienden en el tiempo más que el periodo del símbolo, provocando interferencia entre símbolos que causan distorsión en la señal recibida. La separación entre las componentes facilita su distinción por parte del receptor y puede mitigar su efecto utilizando ecualización adaptativa. Desde el punto de vista espectral, el ancho de banda de coherencia del canal es más estrecho que el de la señal, por lo que las diferentes componentes espectrales de la señal se ven afectados por el canal de forma distinta.

- *Desvanecimiento plano* (cuando $T_m < T_s$, $f_0 > W$): Las componentes de la señal multitrayecto se reciben dentro del tiempo de un símbolo, por lo que el canal no introduce interferencia ente símbolos, aunque sí que puede mermar la relación señal a ruido. En este caso, el *ancho de banda de coherencia* del canal es mayor que el ancho de banda de la señal, por lo que todas sus componentes espectrales son tratadas por igual.

2.2.3. Caracterización del canal multitrayecto en la transmisión digital

En un escenario de transmisión digital, se parte de un símbolo representado en banda base por $s_l(t)$ y de su versión modulada $s(t)$, que se puede expresar como $s(t) = \Re\{ s_l(t) e^{j 2\pi f_c t} \}$, siendo f_c la frecuencia de la portadora (Proakis y Salehi, 2008).

Al enviar este símbolo a través del canal multitrayecto, en recepción obtenemos una respuesta que es la suma del símbolo atenuado y retrasado por los diferentes trayectos:

$$r(t) = \sum_k \alpha_k(t)\, s(t - \tau_k(t))$$

siendo k el número de trayectos, y $\alpha_k(t)$ y $\tau_k(t)$, respectivamente, las variaciones temporales de atenuación y retardo de cada trayectoria.

Si expresamos la señal recibida en función del símbolo en banda base obtenemos:

$$r(t) = \Re \{ [\sum_k \alpha_k(t) e^{-j2\pi f_c \tau_k(t)} s_l(t - \tau_k(t))] e^{j2\pi f_c t} \}$$

La demodulación de esta señal nos lleva a la expresión del símbolo recibido en banda base:

$$z(t) = \sum_k \alpha_k(t) e^{-j2\pi f_c \tau_k(t)} s_l(t - \tau_k(t))$$

De la que deducimos la expresión de la respuesta al impulso del canal multitrayecto en banda base:

$$h(\tau, t) = \sum_k \alpha_k(t) e^{-j2\pi f_c \tau_k(t)} \delta(\tau - \tau_k(t))$$

Donde k, $\alpha_k(t)$ y $\tau_k(t)$ son procesos aleatorios dependientes del tiempo.

2.2.4. Discretización del modelo de respuesta impulsional

Para facilitar el análisis del modelo en sistemas de computación, podemos representar su equivalente suponiendo señales y respuesta al impulso discretas. El número de trayectos $p(t)$, los retardos $\tau_k(t)$, las amplitudes $\rho_k(t)$ y las fases $\theta_k(t)$ de las señales que llegan al receptor se consideran procesos aleatorios dependientes del tiempo. Por lo tanto, la respuesta del canal es un proceso aleatorio que queda representado por (Díaz-Nafría, 2021) :

$$h(n, t) = \sum_{k=1}^{p(t)} \rho_k(t) e^{j\theta_k(t)} \delta(n - \tau_k(t))$$

2.3. Ecualización adaptativa

La **ecualización adaptativa** es una técnica de procesamiento de señales, donde un filtro tiene la capacidad de adaptar sus coeficientes de forma dinámica ante las condiciones cambiantes de la señal o de su entorno. Para ello, los filtros adaptativos implementan algoritmos que buscan la minimización de una función de error determinada.

Cuando las características de las señales y el entorno son variantes, debemos recurrir a métodos con capacidad de adaptación y seguimiento a estos cambios. Los filtros adaptativos son soluciones ideales para sistemas que deben trabajar en tiempo real en entornos no estacionarios. Estas son algunas de las múltiples aplicaciones que implementan técnicas de ecualización adaptativo:

- *Identificación de sistemas*. La comparación de las señales a la salida de un filtro adaptativo en paralelo con un sistema desconocido variante en el tiempo nos permitirá identificar la respuesta del sistema en base a los coeficientes obtenidos en el filtro.

- *Ecualización adaptativa del canal*. Utilizado para mitigar la distorsión producida en la señal por un canal de comunicación de condiciones variantes en el tiempo. Este es el caso de las comunicaciones móviles, donde debido al multitrayecto, la distorsión de los símbolos transmitidos aumenta la interferencia entre estos.

- *Cancelación de eco*. En las comunicaciones *full-duplex*, como la telefónica, parte de la señal del emisor puede introducirse en el canal de vuelta. Ésta se puede suprimir mediante ecualización adaptativa.

En nuestro proyecto nos centramos en la ecualización adaptativa ante las condiciones variantes del canal de comunicación. En la figura 5 podemos observar como se incorpora el ecualizador adaptativo en el esquema del receptor digital. La señal recibida $x(t)$ pasa por el filtro adaptado y se muestrea en los instantes óptimos de muestreo nT. Las muestras resultantes $z(nT)$ pasan por el ecualizador adaptativo, consistente en un filtro que actualiza sus coeficientes en función del algoritmo adaptativo, basado en el error de decisión (diferencia entre las decisiones tomadas a la salida del decisor $d(n)$ y las salidas del propio ecualizador $y(n)$). El algoritmo actualiza los coeficientes muestra a muestra.

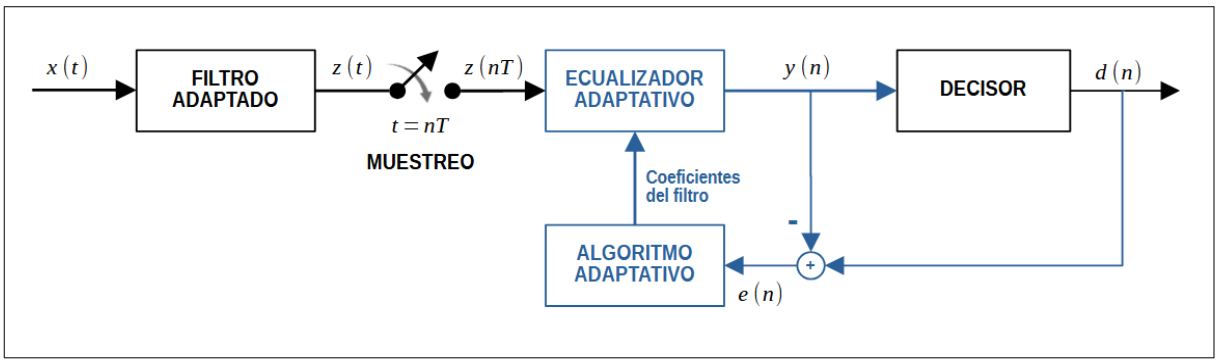

Figura 5. Modelo de receptor digital con la inserción de la ecualización adaptativa.

Existen dos algoritmos fundamentales en el filtrado adaptativo: **LMS** (*Least Mean Squares*, Mínimos cuadrados medios) y **RLS** (*Recursive Least Squares*, Mínimos cuadrados recursivos). Ambos buscan la minimización del error cuadrático medio entre la señal decidida y la señal ecualizada, pero lo abordan de forma diferente. En líneas generales, LMS implementa una solución iterativa por el método de gradiente descendiente, mientras RLS utiliza una método recursivo de actualización de la solución óptima en base a todas las muestras previas. RLS requiere de mayor complejidad y coste

computacional, pero a cambio converge de forma más rápida a los coeficientes óptimos (Proakis y Manolakis, 2007).

2.3.1. Implementación con filtros FIR

Los filtros FIR son los más ampliamente utilizados en el filtrado adaptativo. Por lo tanto, partimos de un filtro FIR de coeficientes adaptativos implementado en su estructura directa (ver figura 6), donde $x(n)$ es la señal a la entrada del filtro, $y(n)$ la señal a su salida, $d(n)$ la señal deseada y $h(n)$ es la respuesta al impulso del filtro de M coeficientes $h(0), h(1), \ldots, h(M-1)$.

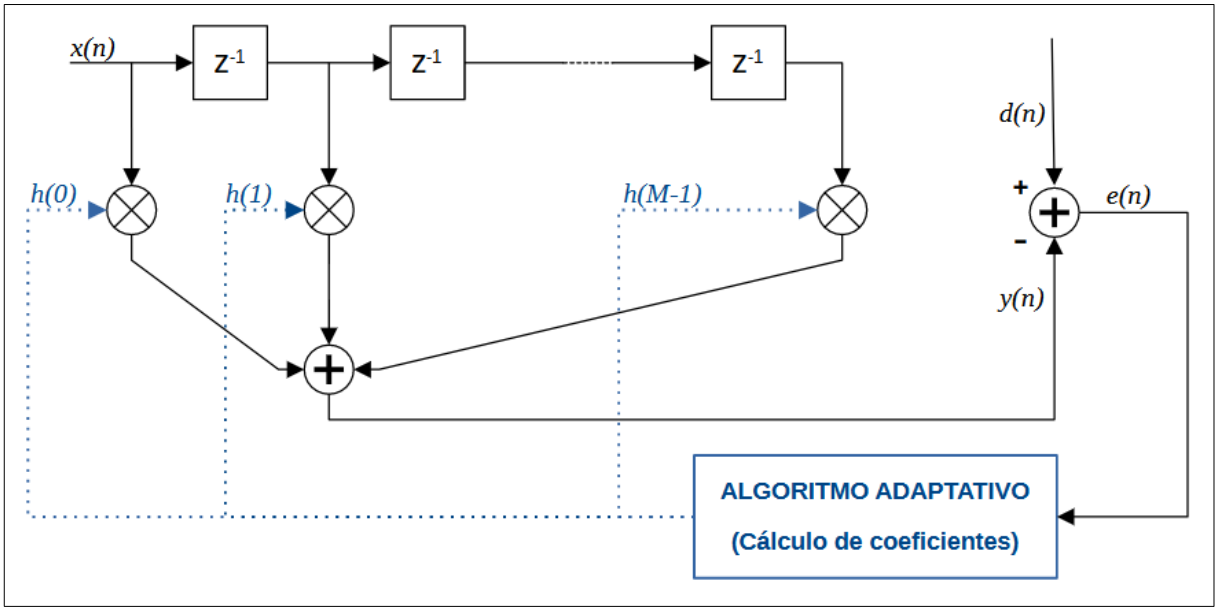

Figura 6. Filtro FIR adaptativo genérico.

La señal a salida del filtro se puede expresar como la convolución entre la señal a la entrada y la respuesta al impulso: $y(n) = \sum_{k=0}^{M-1} x(k)\, h(n-k)$.

El error está determinado por $e(n) = d(n) - y(n)$, y el objetivo es encontrar el valor de los parámetros del filtro que minimicen el error cuadrático medio, según la siguiente función de costo:

$$J = E[e^2(n)] = E[(d(n) - y(n))^2] = E\left[\left(d(n) - \sum_{k=0}^{M-1} x(k)\, h(n-k)\right)^2\right]$$

La optimización de esta función nos lleva al siguiente conjunto de ecuaciones lineales, conocido como

la *ecuación de Wiener-Hopf* (Proakis y Manolakis, 2007):

$$\sum_{k=0}^{M-1} h(k)\,\gamma_{xx}(l-k) = \gamma_{dx}(l) \qquad l = 0, 1, \ldots, M-1$$

En esta ecuación se consideran $x(n)$ y $d(n)$ como procesos aleatorios, siendo $\gamma_{xx}(l)$ la función de autocorrelación de $x(n)$, y $\gamma_{dx}(l)$ la función de autocorrelación cruzada entre $d(n)$ y $x(n)$. La solución a este ecuación da como resultado los coeficientes $h(n)$ *óptimos* del filtro que minimizan el error cuadrático medio, pero desconocemos los valores estadísticos de $x(n)$ y $d(n)$ para poder resolverla. Sin embargo, podemos considerar una ecuación muy similar a partir de los valores reales de las señales:

$$\sum_{k=0}^{M-1} h(k)\,r_{xx}(l-k) = r_{dx}(l) \qquad l = 0, 1, \ldots, M-1$$

donde $r_{xx}(l)$ es la autocorrelación de la secuencia $x(n)$, y $r_{dx}(l)$ la correlación cruzada entre las secuencias $d(n)$ y $x(n)$. Estos valores reales son estimaciones de los valores estadísticos de las señales y la resolución de este sistema de ecuaciones, muestra a muestra, dará como resultado los coeficientes $h(n)$ *estimados* en cada instante.

El objetivo de los algoritmos adaptativos como el LMS es acercarse a la solución *óptima* de la ecuación de Wiener-Hopf a partir de los datos reales de las secuencias, haciendo que los datos *estimados* obtenidos a partir de estos converjan hacia la solución *óptima*.

2.3.2. Algoritmo LMS

El algoritmo LMS (*Least Mean Squares*, Mínimos cuadrados medios) es un método iterativo basado en el gradiente descendente para adaptar los coeficientes de un filtro y minimizar el error cuadrático medio entre la salida filtrada y la señal deseada.

Para ello, propone la actualización iterativa de los coeficientes en dirección del gradiente negativo según la regla:

$$\boldsymbol{h}(n+1) = \boldsymbol{h}(n) + \mu\, e(n)\, \boldsymbol{x}(n)$$

siendo:

- $\boldsymbol{h}(n)=[h_0(n), h_1(n), \ldots, h_{M-1}(n)]^T$, el vector de coeficientes del filtro en el instante n.

- $\boldsymbol{x}(n)=[x(n), x(n-1), \ldots, x(n-M+1)]^T$, el vector de entrada deslizante de M muestras de la señal $x(n)$ en el instante n.

- $e(n)=d(n)-y(n)$, el error en el instante n. Siendo $d(n)$ el valor deseado de la muestra, e $y(n)$ el valor de la salida del filtro en ese instante, donde $y(n)=\sum_{k=0}^{M-1} x(k)\,h(n-k)=\boldsymbol{x}^{\boldsymbol{T}}(n)\,\boldsymbol{h}(n)$.

- μ, la *tasa de aprendizaje*, parámetro que condiciona la velocidad de convergencia y la estabilidad del algoritmo.

La elección de μ es determinante. Una tasa de aprendizaje mayor produce una convergencia más rápida, pero no puede superar la condición de estabilidad del algoritmo dada por $0<\mu<\frac{2}{\lambda_{máx}}$. Donde $\lambda_{máx}$ es el autovalor máximo de $\boldsymbol{R}_{\boldsymbol{xx}}$ (matriz de autocorrelación de la secuencia $x(n)$) . Ante la dificultad de conocer $\lambda_{máx}$, se suele utilizar la aproximación dada por $0<\mu<\frac{1}{M\,P_x}$, donde M es el número de coeficientes del filtro y P_x es la potencia media de la señal de entrada (Proakis y Manolakis, 2007).

El procedimiento del algoritmo se puede sintetizar en los siguientes pasos:

1. Inicialización de los coeficientes del filtro $\boldsymbol{h}(0)=[h_0(0), h_1(0), \ldots, h_{M-1}(0)]^T$. Si fuera posible se tomarían los valores calculados previamente a partir de una señal de referencia conocida por el receptor. Si no, se inicializan normalmente a 0.

2. Entrada de datos. Se toma la muestra $x(n)$ y se construye el vector $\boldsymbol{x}(n)=[x(n), x(n-1), \ldots, x(n-M+1)]^T$.

3. Cálculo de la salida estimada del filtro mediante $y(n)=\sum_{k=0}^{M-1} x(k)\,h(n-k)=\boldsymbol{x}^{\boldsymbol{T}}(n)\,\boldsymbol{h}(n)$.

4. Cálculo del error mediante $e(n)=d(n)-y(n)$.

5. Actualización del vector de coeficientes. Según la regla $\boldsymbol{h}(n+1)=\boldsymbol{h}(n)+\mu\,e(n)\,\boldsymbol{x}(n)$.

6. Incremento del índice temporal $n=n+1$ y salto al paso 2.

2.3.3. Algoritmo RLS

El algoritmo RLS (*Recursive Least Squares*, Mínimos cuadrados recursivos) actualiza los coeficientes del filtro mediante la minimización de la suma del error cuadrático hacia el pasado según una ponderación decreciente exponencial. RLS tiene una convergencia más rápida, pero a costa de mayor complejidad computacional.

RLS trata de encontrar los coeficientes que minimizan la función de costo siguiente:

$$J = \sum_{l=0}^{n} \lambda^{n-l} e^2(l) = \sum_{l=0}^{n} \lambda^{n-l} (d(l) - y(l))^2 = \sum_{l=0}^{n} \lambda^{n-l} (d(l) - \boldsymbol{x}^T(l)\,\boldsymbol{h}(n))^2$$

donde: n es el instante actual; l es el índice que recorre las muestras pasadas desde 0 hasta n, suponiendo $x(n) = 0$ para $n < 0$; $d(l)$ e $y(l)$ son el valor deseado de la muestra y el valor a la salida del filtro, respectivamente; y λ es un parámetro conocido como *factor de olvido* ($0 < \lambda < 1$, típicamente muy cercano a 1, por encima de 0.9) que pondera las muestras de forma exponencial decreciente desde la actual a la más antigua.

Se puede observar que a diferencia del planteamiento inicial que llevó a LMS, donde la función de costo era estadística, RLS plantea una función de costo determinista.

La optimización de la dicha función nos llevará a la resolución de un conjunto de ecuaciones lineales $\boldsymbol{R}_{xx}\,\boldsymbol{h} = \boldsymbol{r}_{xl}$, con el que obtendremos el vector de coeficientes del filtro para cada instante $\boldsymbol{h}(n) = [h_0(n), h_1(n), \ldots, h_{M-1}(n)]^T$, siendo $\boldsymbol{R}_{xx}$ la matriz de correlación de dimensión $M \times M$ a partir de la secuencia $x(n)$, y $\boldsymbol{r}_{xl}$ el vector columna de dimensión $M \times 1$ dado por la correlación cruzada de las secuencias $x(n)$ y $d(n)$.

Para ello, se emplea un método recursivo en el cuál la *ecuación de actualización* de los coeficientes viene dada por $\boldsymbol{h}(n) = \boldsymbol{h}(n-1) + \boldsymbol{k}(n)\,e(n)$, donde $e(n) = d(n) - y(n)$ es el error entre el valor de la muestra deseada y el valor a la salida del filtro en el instante n, y $\boldsymbol{k}(n)$ es el *vector de ganancia de Kalman* (Proakis y Manolakis, 2007), dado por la expresión:

$$\boldsymbol{k}(n) = \frac{\boldsymbol{R}_{xx}^{-1}(n-1)\,\boldsymbol{x}^*(n)}{\lambda + \boldsymbol{x}^T(n)\,\boldsymbol{R}_{xx}^{-1}(n-1)\,\boldsymbol{x}^*(n)}$$

El procedimiento recursivo del cálculo de coeficientes, según el algoritmo RLS, se puede sintetizar como:

1. Inicialización de $\boldsymbol{h}(-1)=\mathbf{0}$ y de $\boldsymbol{R}_{xx}^{-1}(-1)=(1/\delta)\,\boldsymbol{I}$ (δ número positivo pequeño, $\boldsymbol{I}$ matriz identidad M x M).

2. Entrada de datos. Se toma la muestra $x(n)$ y se construye el vector $\boldsymbol{x}(n)=[x(n),\, x(n-1),\, \ldots,\, x(n-M+1)]^T$.

3. Cálculo de la salida estimada del filtro mediante $y(n)=\boldsymbol{x}^T(n)\,\boldsymbol{h}(n-1)$.

4. Cálculo del error mediante $e(n)=d(n)-y(n)$.

5. Cálculo del vector de ganancia de Kalman. Según la expresión
$$\boldsymbol{k}(n)=\frac{\boldsymbol{R}_{xx}^{-1}(n-1)\,\boldsymbol{x}^*(n)}{\lambda+\boldsymbol{x}^T(n)\,\boldsymbol{R}_{xx}^{-1}(n-1)\,\boldsymbol{x}^*(n)}.$$

6. Actualización de la inversa de la matriz de autocorrelación. Según
$$\boldsymbol{R}_{xx}^{-1}(n)=\frac{\boldsymbol{R}_{xx}^{-1}(n-1)-\boldsymbol{k}(n)\,\boldsymbol{x}^T(n)\,\boldsymbol{R}_{xx}^{-1}(n-1)}{\lambda}.$$

7. Actualización del vector de coeficientes. Según la regla $\boldsymbol{h}(n)=\boldsymbol{h}(n-1)+\boldsymbol{k}(n)\,e(n)$.

8. Incremento del índice temporal $n=n+1$ y salto al paso 2.

El algoritmo RLS es ampliamente recomendado para el seguimiento y ecualización de canales móviles multitrayecto debido a su rápida convergencia y su capacidad para adaptarse a variaciones temporales en el canal (Proakis y Salehi, 2008). Éste será el algoritmo que implementaremos en nuestro proyecto de simulación.

CAPÍTULO III. ESTADO DE LA CUESTIÓN

3.1. La simulación computacional en la ingeniería de telecomunicación

La simulación por ordenador ha desempeñado un papel clave en el desarrollo de la ingeniería moderna, y en particular en el ámbito de las telecomunicaciones. Su relevancia radica en su capacidad para modelar sistemas de diversa complejidad de manera eficiente, permitiendo evaluar su comportamiento bajo diversas condiciones sin tener que recurrir a implementaciones físicas, con la reducción de costes y tiempo que eso supone. El diseño y la evaluación del rendimiento de un sistema de comunicación es habitualmente complejo, debido a la cantidad de parámetros y subsistemas implicados. A pesar de que existen métodos potentes disponibles para el diseño y el análisis teórico por parte del ingeniero, en muchos casos la simulación por ordenador de un sistema de comunicación completo, así como de algunas de sus partes, es un enfoque práctico conveniente, y en muchas ocasiones la única opción práctica posible. A lo largo de las últimas décadas, el progreso en las telecomunicaciones ha estado estrechamente vinculado al desarrollo de técnicas de simulación computacional, las cuales han evolucionado desde modelos básicos de circuitos hasta sofisticados entornos capaces de simular redes completas con alto grado de fidelidad entre el sistema y el modelo.

En el contexto de la ingeniería de telecomunicación, la simulación ha permitido analizar y predecir el comportamiento de los sistemas de telecomunicación. Desde sus inicios, en las décadas de 1950 y 1960, se utilizaron simulaciones numéricas para resolver ecuaciones diferenciales que modelaban circuitos eléctricos, filtros y canales analógicos. En esta primera etapa, se aplicaban métodos clásicos como Runge-Kutta o diferencias finitas sobre modelos simplificados, debido a las limitaciones computacionales de la época (Cellier y Kofman, 2006).

Con el aumento de la capacidad de cómputo en las décadas siguientes, las simulaciones comenzaron a abordar fenómenos más complejos como la propagación de ondas electromagnéticas, la respuesta de sistemas no lineales, y la interacción entre distintos subsistemas de comunicación. La aparición de herramientas como MATLAB (y su entorno Simulink) o SPICE facilitó el modelado de circuitos, canales y sistemas de transmisión tanto analógicos como digitales. A partir de los años 1980, la simulación se convirtió en un paso imprescindible para validar diseños antes de su implementación física, permitiendo, por ejemplo, probar esquemas de modulación, técnicas de codificación de canal o filtros digitales sin necesidad de recurrir a la implementación de prototipos costosos.

A lo largo de los años 1990 y 2000, la evolución de la simulación en telecomunicaciones estuvo fuertemente impulsada por la digitalización completa de las señales y por la proliferación de nuevos estándares en redes móviles y de datos. En este contexto, cobró especial relevancia la simulación de redes de comunicación, que permitió modelar de forma precisa el comportamiento de protocolos, arquitecturas de red y tráfico de datos en condiciones dinámicas. El uso de técnicas de simulación de eventos discretos se consolidó como el enfoque predominante para este tipo de análisis, dando lugar al desarrollo y adopción de herramientas especializadas como NS2, NS3 u OMNeT++ (Patel y Kamboj, 2015). En paralelo, surgen herramientas específicas centradas en la propagación de ondas electromagnéticas como HFSS, para analizar el comportamiento de sistemas de antenas y circuitos de microondas. Durante esta etapa también se consolidó el uso de modelos probabilísticos para representar el ruido, la interferencia y los errores en canales reales, lo que llevó al desarrollo de simulaciones Monte Carlo como método estándar para evaluar el rendimiento de sistemas de comunicación digital (Law, 2013).

Actualmente, la simulación computacional en ingeniería se encuentra en una fase de integración con tecnologías emergentes. El paradigma de los gemelos digitales permite replicar digitalmente sistemas reales y simular su comportamiento en tiempo real. Un gemelo digital es un modelo virtual que reproduce de forma precisa el comportamiento y las características de un objeto o sistema físico. Esta réplica digital evoluciona junto con su contraparte real, incorporando datos operativos en tiempo real y empleando técnicas como la simulación, el aprendizaje automático y el análisis predictivo para apoyar la toma de decisiones a lo largo de su ciclo de vida (IBM, 2021).

Los entornos de desarrollo se han ampliado hacia la co-simulación entre plataformas (por ejemplo, entre MATLAB y Python, o entre Simulink y sistemas embebidos como Rapsberry Pi o Arduino). Asimismo, se exploran enfoques basados en inteligencia artificial, como redes neuronales o algoritmos de aprendizaje profundo, para acelerar la generación de modelos y la predicción de resultados. También se destaca el avance de la simulación en la nube, que permite realizar análisis complejos sin necesidad de infraestructura local, y la tendencia hacia entornos colaborativos donde múltiples disciplinas e ingenieros interactúan sobre un mismo modelo compartido.

La evolución de la simulación ha transformado profundamente el proceso: de una herramienta de análisis a posteriori a un componente estratégico del ciclo de vida del producto, desde el diseño hasta la implementación. Esta trayectoria refleja no solo el desarrollo tecnológico, sino también un cambio en la cultura del diseño, orientada cada vez más hacia modelos virtuales, predictivos y adaptativos.

3.2. Simulink y alternativas de simulación modular

Simulink es la plataforma elegida por comLAB para el modelado del ciclo de transmisión completo. Se trata de un entorno de simulación y diseño basado en diagramas de bloques, integrado dentro de MATLAB y desarrollado por MathWorks. Su arquitectura modular y visual permite representar sistemas dinámicos de forma intuitiva, facilitando el modelado, simulación y análisis de sistemas de comunicación complejos. Simulink se integra directamente con MATLAB, lo que permite la incorporación de funciones totalmente personalizadas. Además, dispone de bibliotecas especializadas, como *Communications Toolbox*, que proporciona bloques para tareas fundamentales en sistemas de comunicación: modulación digital, codificación de canal, generación de ruido, simulación de canal, etc. (MathWorks, s.f.a).

Su uso está ampliamente extendido no solo en el ámbito profesional, sino también en el ámbito académico. En universidades de todo el mundo, Simulink y MATLAB son herramientas habituales en asignaturas relacionadas con el tratamiento digital de señales y la teoría de la comunicación. Su enfoque visual y modular facilita la comprensión de conceptos abstractos, permitiendo a los estudiantes construir y probar sus propios modelos de forma interactiva. Además, ofrece la posibilidad de integrar la simulación con hardware real mediante técnicas de *Hardware-in-the-Loop* (HiL), lo que permite validar sistemas de control o procesamiento en condiciones casi reales sin necesidad de contar con todo el entorno físico implementado. Esta funcionalidad resulta especialmente útil tanto en entornos industriales como académicos, ya que permite probar controladores embebidos, sistemas de comunicaciones o algoritmos de procesamiento digital conectando el modelo simulado con dispositivos reales. Esto permite el paso pasar de la simulación al prototipo funcional de forma progresiva, en un única plataforma.

Existen alternativas que pueden ser más adecuadas dependiendo del tipo de sistema que queramos simular. Entre ellas se encuentran GNU Radio**,** plataforma de código abierto especialmente diseñada para el desarrollo de sistemas de radio definidos por software, que permite implementar y probar esquemas de modulación, transmisión y recepción en tiempo real, siendo muy utilizada en investigación experimental y en entornos donde se requiere trabajar directamente con señales de radiofrecuencia. Por otro lado, LabVIEW, desarrollado por National Instruments, está más orientado a la instrumentación virtual y a la adquisición de datos en entornos de medida y control, y se utiliza ampliamente en laboratorios industriales y académicos donde se requiere una integración estrecha con sensores y equipos físicos.

Por último, existe un auge en el uso de lenguajes de propósito general como Python o C++ para la

simulación de sistemas de comunicaciones, impulsado por la flexibilidad que ofrecen y la amplia disponibilidad de bibliotecas especializadas. Python, en particular, ha ganado gran popularidad en el ámbito académico y de investigación gracias a su sintaxis sencilla y a paquetes como *NumPy*, *SciPy*, *Matplotlib* o *SimPy*, que permiten construir simulaciones numéricas, modelos estocásticos o sistemas discretos con relativa facilidad. En el caso de C++, aunque más exigente en cuanto a complejidad sintáctica, permite desarrollar simulaciones altamente eficientes desde el punto de vista computacional y se utiliza frecuentemente en entornos donde el rendimiento es crítico. No obstante, aunque estos lenguajes ofrecen un alto grado de control y personalización, su enfoque textual y la necesidad de programar desde cero muchas de las funciones hacen que carezcan de la estructura modular y del entorno visual intuitivo que proporcionan herramientas como Simulink. Por ello, su uso suele estar más orientado a entornos donde se requiere máxima flexibilidad, automatización o integración con otros sistemas, mientras que herramientas como Simulink resultan más adecuadas cuando se busca una construcción modular, una visualización clara del flujo de señal y una rápida validación funcional, tanto en contextos académicos como en aplicaciones industriales (IONOS, 2023).

3.3. Simulación por tramas y por muestras

Simulink permite configurar el tratamiento de las señales en distintos modos según la estructura temporal de los datos. Las dos modalidades principales en la simulación de señales discretas son la simulación por muestras (*sample-based*) y la simulación por tramas (*frame-based*). Cada una de estas técnicas se ajusta mejor a determinadas aplicaciones, y su elección tiene un impacto significativo en el rendimiento computacional y en la funcionalidad del modelo.

En la simulación por muestras, los datos se procesan de forma secuencial, muestra a muestra. Este enfoque es conceptualmente más cercano al funcionamiento físico del canal de comunicación, que actúa de manera continua sobre la señal recibida, sin conocimiento de estructuras como tramas o bloques, y responde más a lo que consideramos un simulación de manera intuitiva (una muestra en cada instante). Es adecuado para el análisis de circuitos en tiempo real, o para el estudio detallado de fenómenos dependientes del tiempo como oscilaciones o efectos transitorios.

Por otro lado, en la simulación por tramas, las señales se agrupan en bloques de datos (tramas) que se procesan de manera conjunta en cada paso de simulación. Aunque este enfoque introduce una abstracción temporal, resulta mucho más eficiente computacionalmente, ya que Simulink está optimizado para trabajar con matrices y vectores. Además, muchas operaciones propias de los sistemas digitales (como la codificación de canal o la modulación) requieren necesariamente el tratamiento de

bloques de datos. De hecho, muchos de los bloques del *Communications Toolbox* de Simulink están diseñados para operar exclusivamente en modo *frame-based*.

En el caso particular de la simulación de un ciclo completo de comunicación digital, como el que se aborda en este trabajo (incluyendo muestreo, cuantificación, codificación de fuente, codificación de canal, modulación, canal de transmisión, demodulación, decodificación y reconstrucción), la simulación por tramas se revela como la opción más adecuada por su eficiencia computacional y por la coherencia con los subsistemas que trabajan sobre bloques de datos (MathWorks, s.f.b).

CAPÍTULO IV. LÍNEA BASE

4.1. El entorno de comLAB

comLAB es un laboratorio virtual de telecomunicaciones con fines didácticos, implementado sobre la plataforma Simulink/MATLAB. El objetivo de comLAB es el de proporcionar a los estudiantes un entorno visual interactivo, donde se puedan aplicar los conocimientos aprendidos en diversas asignaturas relacionadas con la teoría de la comunicación y la transmisión digital. comLAB forma parte de glossaLAB, que se define como una plataforma interactiva y abierta para la clarificación conceptual y la colaboración interdisciplinar en los sistemas de información (glossaLAB, s.f.).

En la implementación de la que partimos, comLAB representa un sistema de transmisión digital completo de forma modular, simulando el efecto de cada una de las etapas del ciclo de transmisión mediante un bloque de Simulink. Se parte de una señal de audio seleccionable sobre la que se pueden aplicar módulos de muestreo, diezmado, cuantificación, codificación de canal, modulación, canal ruidoso, demodulación, decodificación de canal, reconstrucción e interpolación (ver figura 7).

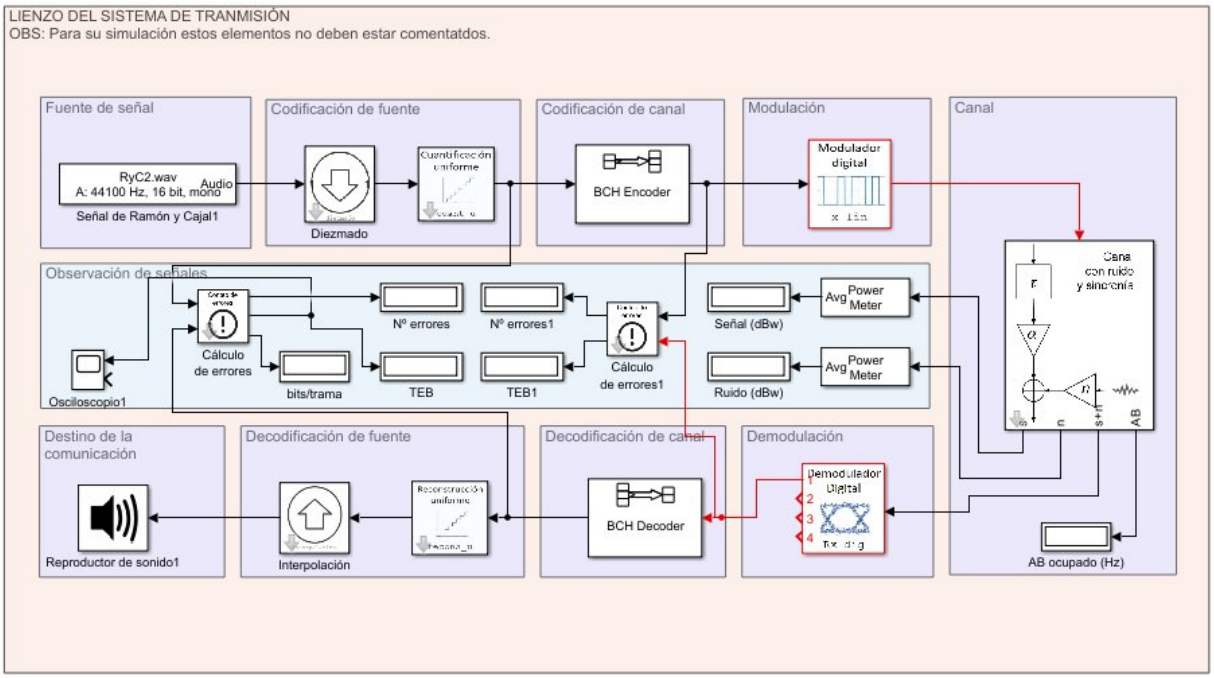

Figura 7. Sistema de transmisión digital en comLAB.

La elección de los módulos que forman parte del sistema de transmisión y su interconexión es editable

por el usuario, en función de los objetivos de aprendizaje y de la parte del ciclo de transmisión sobre la que se desee enfatizar.

A su vez, cada uno de los módulos es configurable a través de un cuadro de diálogo, que aparece al hacer doble click sobre ellos. Simulink denomina este cuadro de diálogo como *mask* (máscara). Como ejemplo, en la figura 8 podemos observar la configuración del bloque de modulación digital, donde además de tener una breve explicación del mismo, podemos seleccionar los parámetros implicados, desde el tipo de pulso elegido y los detalles de su configuración (número de muestras por pulso, o el nivel de entrelazado y factor de redondeo para pulsos de raíz cuadrada de coseno alzado) hasta los valores de la constelación de símbolos y la codificación que representa cada uno de ellos.

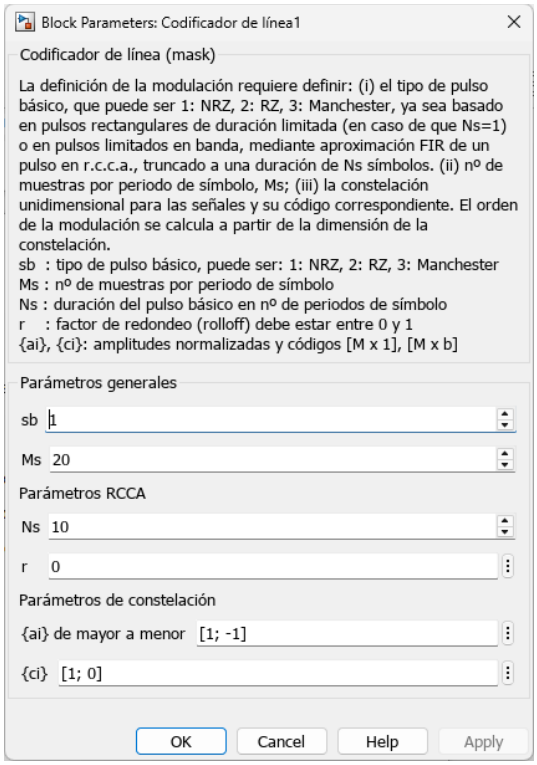

Figura 8. Ejemplo de cuadro de diálogo de configuración de bloque en comLAB: Modulador digital.

El aspecto visual de los cuadros de diálogo la establece el programador, a través de la edición de la máscara, permitiéndole personalizar la interfaz de usuario. En la configuración de la propia máscara se define la acción que se realiza al pulsar en el botón de ayuda. En el caso de comLAB, al acceder a la ayuda de cada bloque, se nos redirige a la entrada de glossaLAB.org donde se clarifica el concepto relacionado con el bloque en cuestión. En esta entrada se incluye un apartado donde se ejemplifica el

concepto mediante un código realizado en MATLAB. Este código suele ser la base de la codificación empleada para crear el propio bloque en Simulink.

Hay que señalar que comLAB está constituido, en su mayoría, por bloques del tipo *MATLAB S-Functions* de nivel 2 . Estas funciones definen las propiedades y comportamiento de un bloque específico en Simulink, e incluyen los métodos que son ejecutados durante la simulación del modelo. Estos métodos se encargan de inicializar el bloque, calcular sus salidas y gestionar su lógica durante la ejecución (MathWorks, s.f.c). Las S-Functions se programan en código MATLAB, y pueden realizar llamadas a funciones nativas o personalizadas.

También existen bloques que actúan como subsistemas, incluyendo varios bloques dentro de una misma unidad funcional. Como ejemplo, en la figura 9 mostramos el subsistema definido por el bloque *canal con ruido y sincronía*.

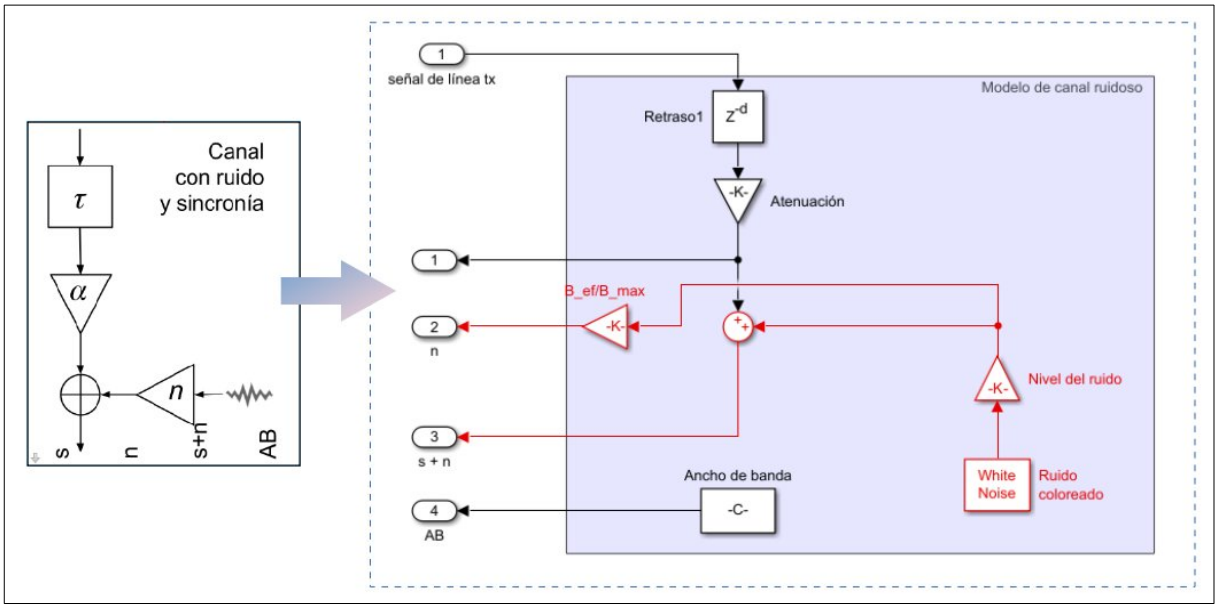

Figura 9. Ejemplo de bloque como subsistema: Canal con ruido y sincronía.

comLAB permite incorporar instrumentos de medida para analizar las señales en distintos puntos del sistema. En el sistema representado en la figura 7 se pueden ver algunos de estos instrumentos situados en la parte central del lienzo.

Disponemos de osciloscopios, para representar las señales en el dominio temporal, y de analizadores de espectro, para su representación en el dominio de la frecuencia. Además, se pueden incorporar medidores de error, para comparar las señales entre puntos simétricos del ciclo de transmisión (por ejemplo, antes de la codificación de canal y después de la decodificación de canal, o antes de la

modulación digital y después del receptor digital). Con esto, podemos observar la degradación que introduce el circuito sobre la señal en las distintas etapas. Para llevar a cabo estas mediciones de error, a nivel de muestra, a nivel de bit o a nivel de byte, dependiendo del punto en el que se realicen, es fundamental que se sincronicen los datos a la entrada del medidor. Para ello, se tiene que considerar el retardo que introduce cada bloque en la señal y compensarlo a la entrada del medidor.

4.1.1. Método de solapamiento y almacenamiento en la simulación por tramas

En comLAB la simulación se realiza por tramas, de forma que la señal continua muestreada al principio del ciclo se trocea en tramas del mismo tamaño. El número de muestras de la trama se define en el bloque de la señal de entrada, y va a condicionar las dimensiones de las entradas y salidas del resto de bloques.

A pesar de esta división por tramas, es importante mantener la coherencia con la naturaleza continua de la señal original. Es decir, si el procesamiento en un bloque genera una salida que se extiende más allá del final de la trama actual, dicho exceso debe conservarse y transferirse a la siguiente trama. Esto implica que el procesamiento debe ser trama a trama, pero con memoria entre tramas para evitar discontinuidades en la señal. Este procedimiento de acarreo de datos entre tramas es conocido como *método de solapamiento y almacenamiento* (Proakis y Manolakis, 2007).

Por ejemplo, en un bloque que simule un filtro FIR de orden M, la salida en un instante determinado no depende solo de la entrada en ese mismo instante, sino también de los $M-1$ valores anteriores. Por tanto, si el filtrado comienza en la primera muestra de una nueva trama, se deben incluir también las últimas $M-1$ muestras de la trama anterior como parte del cálculo. Esto asegura que el filtrado tenga continuidad y que el resultado sea equivalente al que se obtendría si se procesara la señal completa sin dividirla en tramas. En la primera trama, donde no existe acarreo de la trama anterior, se considera que los $M-1$ valores heredados son cero.

4.2. Funciones de partida en MATLAB

Antes de comenzar con el diseño de los bloques en el entorno comLAB, es esencial verificar previamente la validez de los códigos mediante simulaciones en la consola de MATLAB. En esa fase inicial, se comprobará el correcto funcionamiento del ciclo completo de transmisión digital, incluyendo tanto la propagación multitrayecto como la ecualización adaptativa, a través de funciones

de MATLAB que posteriormente se integrarán en el modelo de Simulink.

Para ello, partimos de la actividad de aprendizaje *Aplicaciones de filtrado adaptativo* (Díaz-Nafría, 2021), perteneciente a la asignatura *Tratamiento digital de la señal,* que es la semilla de este proyecto. Esta actividad tenía como objetivo la simulación en MATLAB de una cadena de transmisión digital afectada por un canal multitrayecto variante en el tiempo, y el diseño de una función decisora con ecualización adaptativa. Algunas de las funciones de partida de dicha actividad, en sus versiones más recientes, han sido reutilizadas para el presente proyecto.

A continuación, se describen las funciones de MATLAB empleadas. En el anexo I se recogen los códigos de cada una de estas funciones.

- **`filtro_AA()`**: Realiza un filtrado *antialiasing* sobre la secuencia de entrada, mediante un filtro de Butterworth de orden 20, utilizando como frecuencia de corte normalizada $1/M$, siendo M el orden de diezmado (o factor de submuestreo) utilizado posteriormente en **`muestreo()`**.

- **`muestreo()`**: Submuestrea la secuencia de entrada en un factor M, quedándose con las muestras $n{\cdot}M$. Devuelve también la base de tiempos de la secuencia de salida y la nueva frecuencia de muestreo.

- **`compr_A()`**: Aplica la ley A de compresión a la secuencia de entrada.

- **`cuant_u()`**: Realiza una cuantificación uniforme de la secuencia de entrada, normalizada a $[-1, 1]$, usando b bits, con bit de signo y 0 como umbral de decisión. La entrada es una secuencia de valores tipo *double* y la salida una secuencia de bits codificados como variables tipo *single*.

- **`cod_dat()`**: Codifica un fragmento de la señal seleccionada, aplicando sucesivamente **`filtro_AA()`**, **`muestreo()`**, **`compr_A()`** y **`cuant_u()`**. Devuelve también, además de la señal codificada, el fragmento de de la señal de entrada.

- **`s_b()`**: Genera un símbolo básico de M_S muestras para varios códigos de línea: NRZ, RZ o Manchester. El parámetro de entrada N_S define el nivel de entrelazado, considerando pulsos de raíz cuadrada de coseno alzado, y r el factor de redondeo (*roll-off*). En caso de seleccionar $N_S = 1$, se consideran pulsos rectangulares.

- **`x_lin()`**: Genera una señal de línea a partir de secuencia de datos binarios y de los parámetros

de definición del pulso, recurriendo a la función **`s_b()`**.

- **`s_linea()`**: Genera la señal de línea correspondiente a un fragmento de la señal de entrada seleccionada, con los parámetros de definición del pulso como entrada. Recurre a las funciones **`cod_dat()`** y **`x_lin()`**.

- **`multitrayecto()`**: Aplica la distorsión correspondiente al canal multitrayecto sobre la señal de línea de entrada. Como parámetros de entrada se consideran: el número de trayectos, los retardos mínimo y máximo, la desviación típica de las amplitudes de los trayectos secundarios y la longitud de las tramas (en muestras) en las que las condiciones del canal no varían. Esta función sirve como punto de partida a nueva versión de la misma, que llamaremos **`multipath()`**, para distinguirla de la original, y cuyo diseño es uno de los objetivos del proyecto.

- **`ruido_blanco()`**: Genera una secuencia de ruido de una longitud determinada, a partir de la densidad espectral de potencia de ruido deseada y de la frecuencia de muestreo (que determina el ancho de banda).

- **`Rx_dig()`**: Realiza una recepción óptima de la señal de línea recibida, devolviendo los datos detectados. De está función solo aprovecharemos la primera parte del código, que corresponde a la aplicación del filtro adaptado y su muestreo, tratando la decisión por separado. La nueva función, que solo realiza el filtrado adaptado y devuelve las muestras en los instantes óptimos, la renombramos como **`FiltroAdaptado()`**.

- **`decisor()`**: Realiza la decisión a partir de la señal muestreada tras el filtrado adaptado. Devuelve la señal reconstruida y señal codificada, en función de la constelación unidimensional de entrada y de los códigos asociados a cada símbolo.

CAPÍTULO V. DESARROLLO

5.1. Simulación de la cadena de transmisión en MATLAB

Como primer paso, antes de implementar en Simulink los módulos correspondientes al canal multitrayecto y a la ecualización adaptativa, se pretenden validar los algoritmos y los códigos desarrollados a través de la consola de MATLAB, que nos servirá como banco de prueba del ciclo de transmisión completo, antes de trasladar la solución a comLAB.

Para ello, partiendo de los códigos descritos en la sección 4.2, generaremos una señal de línea a partir de una señal de entrada, que en muestro caso es un archivo de audio que contiene la grabación de un fragmento de un discurso de Ramón y Cajal (**`RyC.wav`**). La señal de línea, configurada conforme a los parámetros de modulación elegidos, será sometida a la función que simula la propagación multitrayecto y posteriormente sufrirá la adición de ruido blanco, simulando el canal de comunicación. En recepción, la señal atravesará la función que realiza el filtrado adaptado y el muestreo en los instantes óptimos. Las muestras resultantes se enviarán a la función que realiza la decisión y la reconstrucción, que incorpora ecualización adaptativa. A este último bloque lo denominamos *decisor adaptativo*. El diseño de las funciones que modelan la propagación multitrayecto y la decisión adaptativa son los principales objetivos de esta parte del proyecto.

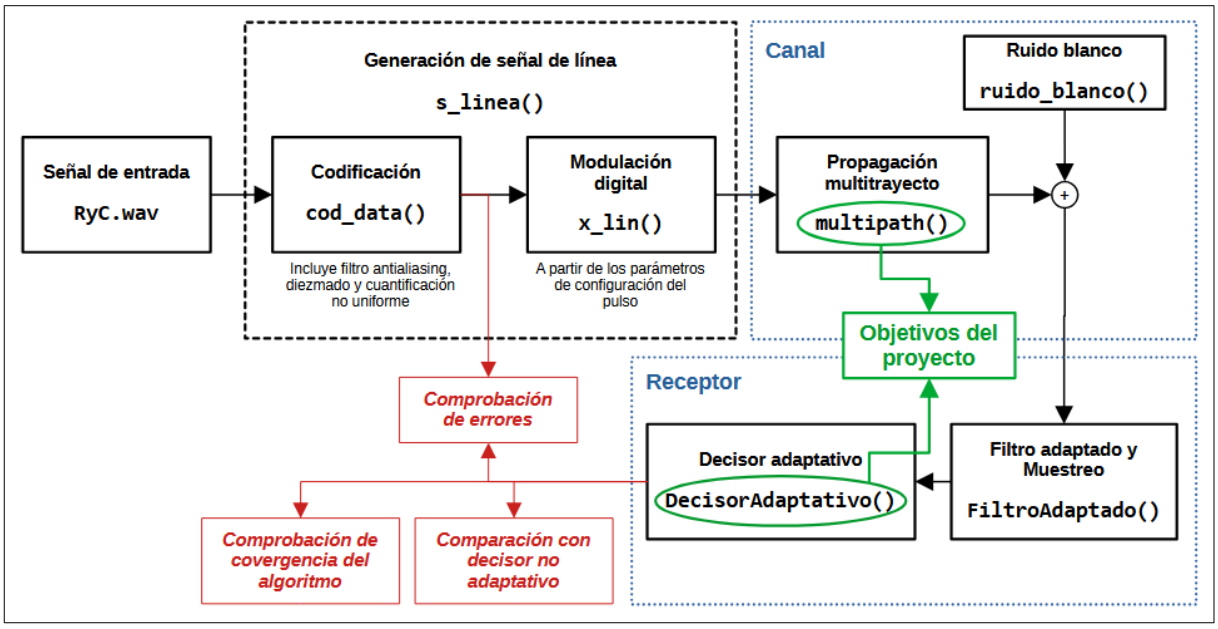

Figura 10. Esquema del ciclo de transmisión con funciones MATLAB

Finalmente, realizaremos las comprobaciones adecuadas para verificar la solvencia del decisor adaptativo, mediante la comparación de las secuencias codificadas de entrada y salida, la comparación con el resultado de un decisor no adaptativo y la constatación de la convergencia del algoritmo RLS. En la figura 10 se muestra el esquema del ciclo de transmisión completo y las funciones de MATLAB implicadas.

5.1.1. Generación de la señal de línea

En primer lugar, se va a generar la señal de línea a partir de un archivo de audio, que representa la señal de entrada a nuestro sistema. Este archivo digital de entrada, muestreado a 44100 Hz y codificado con 16 bits, será sometido a un proceso de diezmado (submuestreo), acorde a las características espectrales de la señal. Según el teorema de muestreo, para evitar el *aliasing*, la frecuencia de muestreo debe ser superior al doble de la frecuencia máxima de la señal, $F_s > 2\,F_{max}$ (Proakis y Manolakis, 2007). Para limitar el contenido espectral por debajo de esta frecuencia máxima con información relevante, aplicaremos previamente un filtrado *antialiasing*.

Posteriormente, se cuantificarán estas muestras, con el número de bits que consideremos adecuado para mantener la relación de señal a ruido de cuantificación, *SQNR*, por encima del rango dinámico de la señal. Para ello, se utilizará cuantificación no uniforme, aplicando en primer lugar compresión y posteriormente cuantificación uniforme. Estos datos se serializarán para obtener la señal codificada.

A partir de la señal codificada, y de los parámetros de modulación digital elegidos, se obtendrá la señal de línea.

Para este propósito, utilizamos la función **`s_linea()`**, cuyo código está incluido en el anexo I. Está función utiliza el siguiente formato de entradas y salidas:

```
[datos,x_l,t] = s_linea(s,Rb,signal,inicio,fin,Ms,tipo,{Ns,r})
```

siendo las entradas:

- **`s`**: Potencia media de la señal de salida. Para simplificar el cálculo, tomaremos normalmente 1W.
- **`Rb`**: Tasa binaria de transmisión en bits/segundo. Este parámetro es significativo para confeccionar la base de tiempos. Tomaremos normalmente 1 Mbps (10^6 bits/segundo).

- **`signal`**: Con este parámetro elegimos la señal de entrada para efectuar las pruebas. En nuestro caso tomaremos la señal 1, que corresponde a **`RyC.wav`**.

- **`inicio`** y **`fin`**: Determinan el fragmento de muestras de la señal de entrada que queremos seleccionar.

- **`Ms`**: Número de muestras por periodo del pulso elegido.

- **`tipo`**: Tipo de pulso utilizado: 1 – NRZ (*Non Return to Zero*), 2 – RZ (*Return to Zero*), 3 – Manchester.

- **`Ns`** y **`r`**: Estos parámetros son opcionales, y su omisión supone el uso de pulsos rectangulares. En caso de incluirse, se supone el uso de pulsos de raíz cuadrada de coseno alzado (RCCA), y representan el número de símbolos entrelazados (**`Ns`**) y el factor de redondeo (o *roll-off*, **`r`**).

y las salidas:

- **`datos`**: Datos codificados y serializados previos a la modulación. Esta salida la conservaremos para compararla con la señal codificada resultante al final del ciclo, y calcular la tasa de error de bit correspondiente al proceso.

- **`x_l`**: Señal de línea. Esta salida será la que pasemos a la siguiente etapa de la cadena de transmisión.

- **`t`**: Base de tiempos para la señal de línea.

La función **`s_linea()`** recurre a internamente a otras dos funciones: **`cod_data()`** y **`x_lin()`**, cuyos códigos están disponibles en el anexo I.

`cod_data()` se encarga de generar la señal codificada del fragmento delimitado por **`inicio`** y **`fin`**. Para ello, en función de la señal seleccionada, aplica un proceso distinto de filtrado antialiasing, diezmado, compresión y cuantificación uniforme, mediante las funciones **`filtro_AA()`**, **`muestreo()`**, **`compr_A()`** y **`cuant_u()`**, respectivamente.

Para el caso de nuestra señal de entrada (**`RyC.wav`**), que se trata de una señal de voz, su contenido espectral relevante está por debajo de los 4000 Hz, por lo que es adecuado aplicar un diezmado de orden 5, que deja la frecuencia de muestreo en $44100/5 = 8820\ Hz$, que cumple con el teorema de muestreo. El filtro antialiasing se aplicará con una frecuencia de corte de $44100/(2 \cdot 5) = 4410\ Hz$.

Posteriormente, se ejerce la compresión según la ley A y la cuantificación uniforme con 8 bits, equivalente a realizar la cuantificación no uniforme, y suficiente para mantener el margen dinámico de la señal por debajo de la SQNR.

Por otra parte, **`x_lin()`** se encarga de generar la señal modulada, a partir de la potencia media deseada (**`s`**), de los parámetros propios de la modulación (**`tipo`** y **`Ms)`** y los opcionales para pulsos RCCA: **`Ns`** y **`r`**. Esta función recurre internamente a la función **`s_b()`**, fundamental en todo este proceso, ya que es la encargada de generar el pulso básico.

`s_b()` tiene el formato de entrada **`s_b(tipo,Ms,Ns,r)`**, siendo los parámetros de entrada los ya comentados anteriormente, considerándose pulsos rectangulares cuando **`Ns=1`**, si no, se consideran pulsos RCCA. En la figura 11 representamos ejemplos de pulsos básicos generados mediante la función **`s_b()`**.

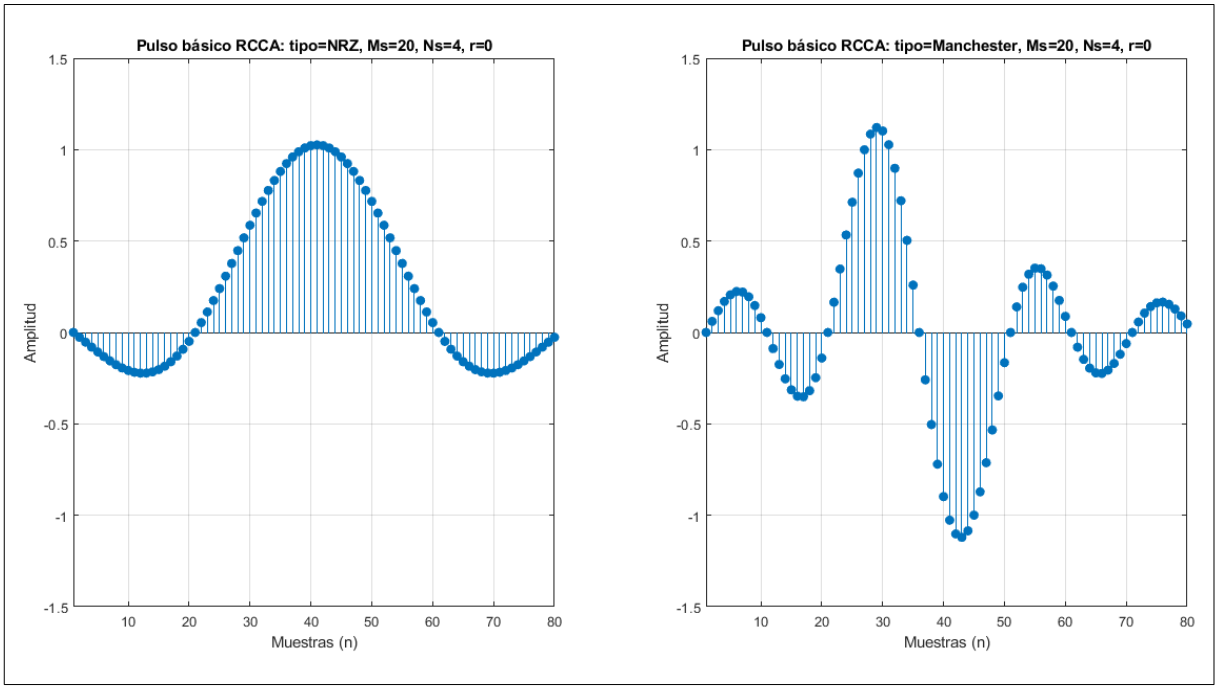

Figura 11. Representación de pulsos básicos RCCA con **`Ms=20`**, **`Ns=4`** *y* **`r=0`**, *para tipos NRZ y Manchester, mediante* **`s_b(1,20,4,0)`** *y* **`s_b(3,20,4,0)`**, *respectivamente.*

A continuación, representamos un mismo fragmento de la señal de línea correspondiente a **`RyC.wav`** con dos modulaciones diferentes. En un caso utilizando pulsos de raíz cuadrada de coseno alzado (**`tipo=1`** (NRZ), **`Ms=16`**, **`Ns=4`**, **`r=0`**), y en el otro pulsos rectangulares (**`tipo=1`** (NRZ), **`Ms=16`**). Se puede apreciar la diferencia en las dimensiones de la señal de salida, motivadas por la diferencia de tamaño del pulso modulador.

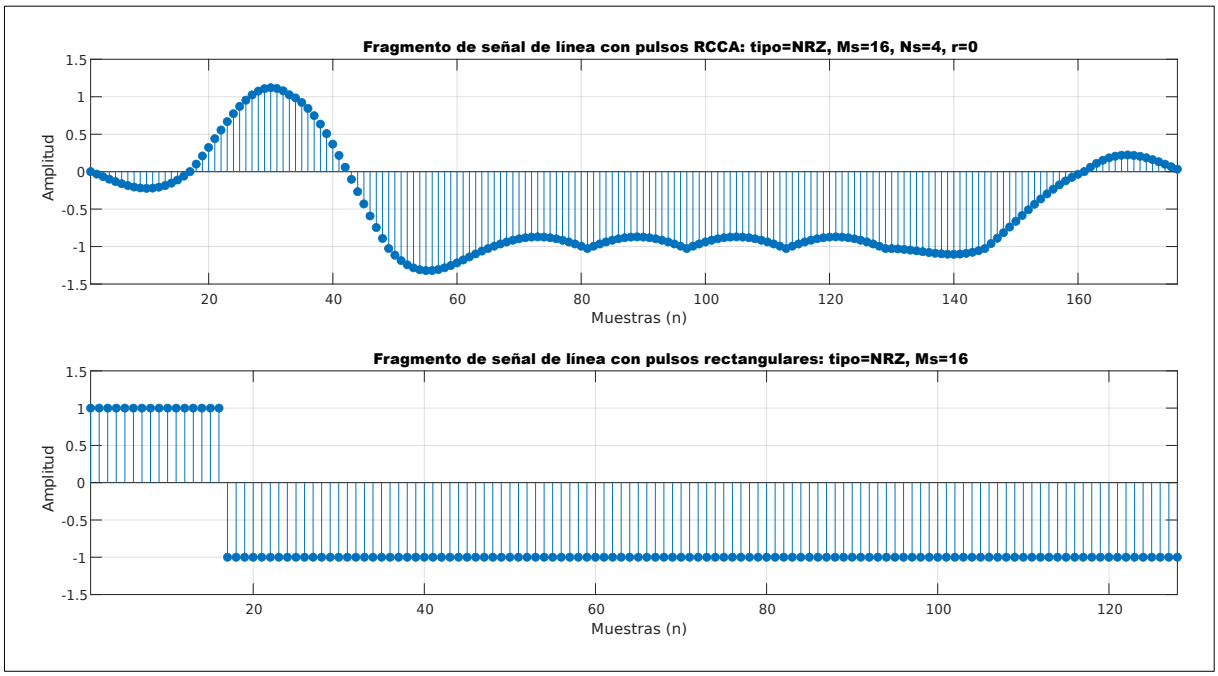

Figura 12. Fragmentos de señal de línea que representan la misma muestra de la señal de entrada con dos modulaciones diferentes: pulsos RCCA y pulsos rectangulares.

La señal de línea es el resultado de convolucionar la señal codificada, representada en un tren de deltas espaciadas M_S muestras, con el pulso modulador. El pulso modulador tiene una longitud de $M_S \cdot N_S$ muestras, siendo $N_S = 1$ en pulsos rectangulares. Si la señal codificada es de N bits, el tren de deltas codificado tendrá una dimensión de $M_S \cdot N$ muestras. Por lo tanto, la señal de línea tendrá una dimensión de $M_S \cdot N + M_S \cdot N_S - 1$ muestras. En consecuencia, la diferencia entre una señal modulada con pulsos RCCA de entrelazado N_S y otra modulada con pulsos rectangulares ($N_S = 1$), con el mismo número de muestras por periodo, es de $M_S(N_S - 1)$ muestras.

Las consideraciones sobre la dimensión de la señal de línea tiene especial relevancia más adelante, cuando en recepción se traten de encontrar los instantes óptimos de muestreo.

En el ejemplo citado, la diferencia entre las dos señales será de $M_S(N_S - 1) = 16(4 - 1) = 48$ muestras, como se observa en la figura 12.

5.1.2. La función multitrayecto

En este punto, nos encontramos ante uno de los objetivos de diseño de este proyecto: el modelado de la propagación multitrayecto. En primer lugar en la consola de MATLAB, y más adelante en el

entorno de comLAB en Simulink.

Para ello, consideramos el canal multitrayecto como un sistema discreto, cuyo comportamiento podemos caracterizar por su respuesta impulsional según el modelo presentado en la sección 2.2.4:

$$h(n,t)=\sum_{k=1}^{p(t)} \rho_k(t)\, e^{j\theta_k(t)}\, \delta(n-\tau_k(t))$$

donde el número de trayectos $p(t)$, los retardos $\tau_k(t)$, las amplitudes $\rho_k(t)$ y las fases $\theta_k(t)$ de las señales que llegan al receptor se consideran variables aleatorias dependientes del tiempo.

Como simplificación, vamos a tomar el primer trayecto como referencia, de forma que vamos a considerar que los retardos y las fases son relativas al trayecto principal, es decir, el retardo y la fase del primer trayecto son nulos. Además, normalizamos la respuesta al impulso respecto a la amplitud del primer trayecto, es decir, su amplitud es la unidad. Por lo tanto, el modelo queda de la siguiente forma:

$$h(n,t)=\delta(n)+\sum_{k=2}^{p(t)} \frac{\rho_k(t)}{\rho_1(t)}\, e^{j\theta_k(t)}\, \delta(n-\tau_k(t))$$

Para simplificar la nomenclatura, renombramos $\frac{\rho_k(t)}{\rho_1(t)}$ como $\rho_k(t)$, por lo tanto, a partir de ahora $\rho_k(t)$ representa las amplitudes de los trayectos secundarios normalizadas respecto a la amplitud del primer trayecto, quedando la representación matemática como:

$$h(n,t)=\delta(n)+\sum_{k=2}^{p(t)} \rho_k(t)\, e^{j\theta_k(t)}\, \delta(n-\tau_k(t))$$

A continuación, vamos a definir como son las distribuciones de probabilidad de las variables aleatorias implicadas, para caracterizar el comportamiento estadístico del canal.

En primer lugar, establecemos un número máximo de trayectos posibles P_{max}. De esta forma podemos definir que $p(t)$ tendrá una distribución uniforme en el intervalo $[1, P_{max}]$.

A su vez, definimos un retardo máximo τ_{max}, estando cada $\tau_k(t)$ uniformemente distribuida en el intervalo $[1, \tau_{max}]$.

En cuanto a las fases $\theta_k(t)$, consideramos que siguen una distribución uniforme en el intervalo $[-\pi, \pi]$.

Para las amplitudes de los trayectos secundarios $\rho_k(t)$, suponemos una distribución normal con desviación típica σ y media nula.

Los parámetros P_{max}, τ_{max} y σ serán necesarios para caracterizar el canal, y se considerarán parámetros de entrada de la función multitrayecto como **`Pmax`**, **`Tmax`** y **`sigma`**, respectivamente.

Otro aspecto que debemos tener en cuenta en la caracterización del canal es su variabilidad temporal. Hay que considerar que las variables aleatorias implicadas son dependientes de t, y por tanto debemos tener presente el factor temporal en nuestra función. Para ello, vamos a introducir el parámetro **`Nt`**, que se define como el número de subtramas, dentro de la trama principal de entrada, donde las condiciones del canal van a ser diferentes. Es decir, el vector **`x,`** que representa la señal de entrada al canal, se verá afectado por **`Nt`** repuestas al impulso diferentes en el transcurso del tiempo. Si **`Nt=1`**, las condiciones del canal serán constantes para toda la señal de entrada. Como veremos más adelante, como simplificación, consideraremos que las subtramas tienen el mismo tamaño.

Por lo tanto, este será el formato de nuestra función multitrayecto:

```
x_mt = multipath(x,Pmax,Tmax,sigma,Nt)
```

siendo:

- **`x`**: Vector que representa la señal de línea generada en la etapa anterior de la cadena de transmisión, que es la entrada de nuestro canal.
- **`Pmax`**: Número máximo de trayectos que puede recorrer la señal.
- **`Tmax`**: Retardo máximo de cualquier trayecto, respecto al trayecto principal.
- **`sigma`**: Desviación típica de la amplitud de los trayectos secundarios.
- **`Nt`**: Número de condiciones diferentes del canal a las que se ve afectada la señal de entrada.
- **`x_mt`**: Vector que representa la señal de salida tras verse afectada por la propagación multitrayecto.

Una vez definidas las entradas y la salida, pasamos a describir la estructura interna de la función.

Para cada **`Nt`**, es decir, para cada situación diferente del canal, tenemos que definir un vector de dimensión **`Tmax`**, que defina la respuesta el impulso del canal $h(n)$ en ese momento.

Con esto formamos una matriz **B** de dimensiones **Nt** x **Tmax** que contendrá las respuestas al impulso del canal para cada situación distinta. Esta matriz será recorrida fila por fila para aplicar la respuesta al impulso del canal a la subtrama correspondiente del vector de entrada.

Para la construcción de **B** tendremos que recurrir a las variables aleatorias que definen cada $h(n)$: p, ρ_k, θ_k y τ_k. Nótese que se han suprimido los índices t, ya que nos estamos refiriendo a $h(n)$ para un t determinado.

Para cada una de estas variables aleatorias, creamos su variable correspondiente en la función:

- **P** para el número de trayectos p.
- **r** para las amplitudes de los trayectos secundarios ρ_k.
- **theta** para las fases de los trayectos secundarios θ_k.
- **tau** para los retardos, en muestras, respecto al trayecto principal τ_k.

P será un vector de dimensión **Nt** x **1**, que asociará a cada subtrama el número de trayectos correspondientes, conforme a una distribución de probabilidad uniforme entre **1** y **Pmax**, según se ha comentado con anterioridad. Para ello recurrimos a la función de MATLAB **randi()**, que nos devuelve una matriz con valores enteros aleatorios, según una distribución uniforme en el intervalo definido, con las dimensiones elegidas en la entrada (MathWorks, s.f.d). En nuestro caso:

```
P = randi([1,Pmax],Nt,1)
```

tau será una matriz de dimensión **Nt** x **(Pmax-1)**, que asociará a cada subtrama (fila) los retardos correspondientes a cada uno de los trayectos secundarios (columna). Los valores de **tau** seguirán una distribución uniforme entre **2** y **Tmax**, ya que el valor **1** (primer índice en la respuesta al impulso) está reservado para el primer trayecto, que corresponde con un retardo nulo en el modelo matemático. Volviendo a recurrir a **randi()**, se define **tau** como:

```
tau = randi([2,Tmax],Nt,Pmax-1)
```

De forma análoga, se establecen **r** y **theta**, como las matrices que asociarán a cada subtrama (fila) las amplitudes y fases, respectivamente, correspondientes a cada uno de los trayectos secundarios (columna). Los valores de **theta** seguirán un distribución uniforme entre **-pi** y **pi**, para lo que utilizaremos la función **rand()**, que nos devuelve una matriz, con las dimensiones elegidas, de valores reales uniformemente distribuidos en el intervalo $[0, 1]$ (MathWorks, s.f.e). Para **r** utilizamos una

distribución normal de media nula y desviación típica **sigma**. Para ello, recurrimos a la función **randn()**, que devuelve una matriz, con las dimensiones seleccionadas, de valores aleatorios de una distribución normal tipificada ($\mu = 0$, $\sigma = 1$) (MathWorks, s.f.f). Por tanto, definimos **theta** y **r** como:

```
theta = rand(Nt,Pmax-1)*2*pi-pi

r = randn(Nt,Pmax-1)*sigma
```

Por comodidad para su acceso, agrupamos las amplitudes y las fases en una sola expresión compleja de la forma $|\rho_k(t)|\, e^{j\theta_k(t)}$. Con lo que nos queda la matriz **r_complejo**:

```
r_complejo = abs(r).*exp(1j*theta)
```

Una vez definidas las matrices **P**, **tau**, **theta** y **r_complejo**, procedemos a la construcción de la matriz **B**. En primer lugar, la primera columna de **B**, correspondiente al trayecto principal, se asigna toda con el valor **1**. Posteriormente, recorremos las filas de **B** en un bucle de índice **n** desde **1** a **Nt**. Para cada fila buscamos el número de trayectos correspondientes en **P** (**P(n)**, con valores entre **1** y **Pmax**), y recorremos las columnas **k** con otro bucle desde **1** a **P(n)-1** (solo trayectos secundarios), que busca en **tau** el retardo correspondiente para cada trayecto (entre **2** y **Tmax**). Este valor de **tau(n,k)** es el índice (columna) donde tenemos que escribir el valor de **r_complejo** correspondiente a los índices del bucle. En caso de que tau tenga valores repetidos en la misma fila se suman los valores de **r_complejo**.

En la figura 13 se muestra un ejemplo de la matriz **B** (respuesta al impulso por subtrama).

B

15x8 complex double

	1	2	3	4	5	6	7	8
1	1.0000 + 0.0000i	0.0000 + 0.0000i	-0.0913 + 0.1712i	-0.2309 - 0.1881i	0.0000 + 0.0000i	-0.0265 + 0.1968i	0.0000 + 0.0000i	0.0000 + 0.0000i
2	1.0000 + 0.0000i	0.1263 + 0.2842i	-0.0740 + 0.0854i	0.0000 + 0.0000i	0.0000 + 0.0000i	-0.2261 + 0.1975i	0.4650 - 0.5131i	0.0000 + 0.0000i
3	1.0000 + 0.0000i	-0.0759 + 0.2282i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
4	1.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
5	1.0000 + 0.0000i	-0.3923 - 0.1562i	0.0824 - 0.2232i	-0.3646 - 0.2693i	0.0000 + 0.0000i	0.0000 + 0.0000i	-0.0266 - 0.0985i	0.0091 - 0.0282i
6	1.0000 + 0.0000i	0.0000 + 0.0000i	-0.2708 + 0.2192i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0802 - 0.2504i	0.0000 + 0.0000i	0.0000 + 0.0000i
7	1.0000 + 0.0000i	-0.2604 - 0.1774i	0.0000 + 0.0000i	-0.2223 - 0.1254i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
8	1.0000 + 0.0000i	0.0779 - 0.2574i	0.0996 + 0.0719i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	-0.1188 + 0.0007i
9	1.0000 + 0.0000i	-0.0466 + 0.0010i	0.0000 + 0.0000i	-0.0780 + 0.0870i	0.0000 + 0.0000i	0.0210 + 0.1209i	0.0000 + 0.0000i	0.0556 - 0.0129i
10	1.0000 + 0.0000i	-0.0073 - 0.0228i	-0.0218 - 0.0323i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
11	1.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.1280 + 0.1195i	0.0000 + 0.0000i	0.0000 + 0.0000i
12	1.0000 + 0.0000i	0.0000 + 0.0000i	0.0939 + 0.2438i	0.0000 + 0.0000i	0.0000 + 0.0000i	-0.4872 - 0.0229i	0.0000 + 0.0000i	0.0000 + 0.0000i
13	1.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	-0.0008 + 0.0840i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
14	1.0000 + 0.0000i	0.0949 - 0.2968i	0.1977 - 0.0077i	0.0000 + 0.0000i	-0.0222 - 0.0522i	0.0000 + 0.0000i	0.1862 + 0.4224i	0.0000 + 0.0000i
15	1.0000 + 0.0000i	-0.0003 - 0.0228i	0.0000 + 0.0000i	-0.0252 + 0.2727i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.1224 + 0.3515i	0.0000 + 0.0000i

Figura 13. Ejemplo de matriz **B** *con* **Nt=15**, **Pmax=6**, **Tmax=8**, **sigma=0.25**

En la figura 14 se muestra la representación de varias respuestas al impulso para otro ejemplo de construcción de **B**.

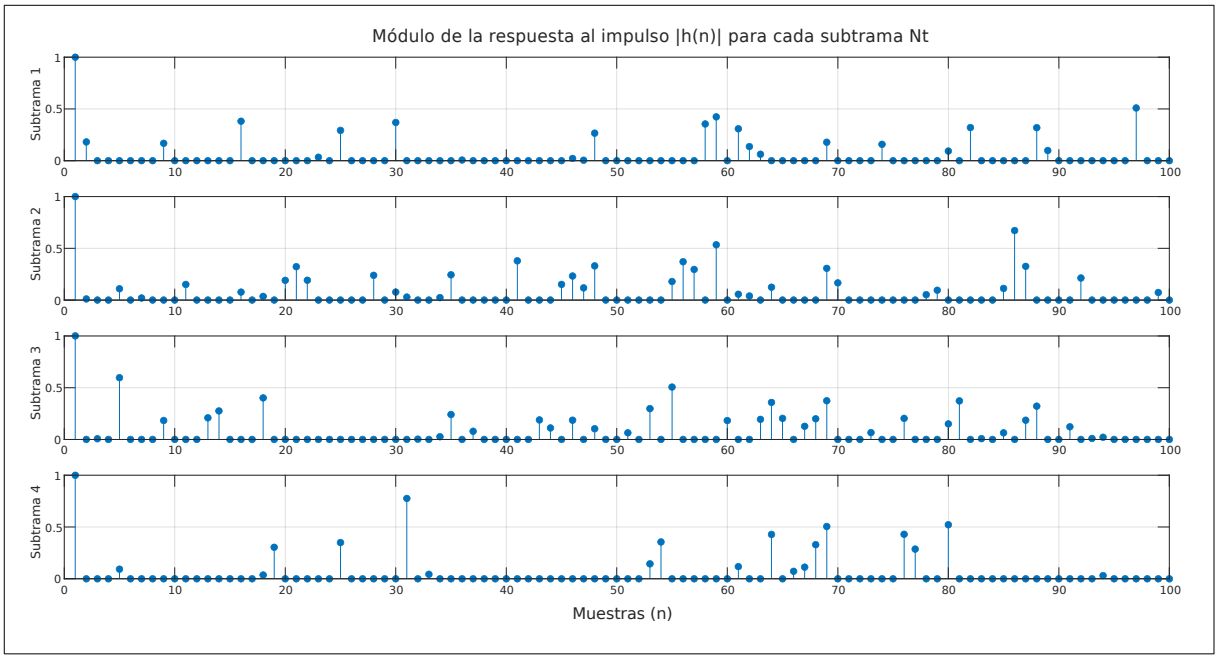

Figura 14. Representación del módulo de la respuesta al impulso (cada fila de **B***) para cada subtrama, con* **Nt=4**, **Pmax=50**, **Tmax=100**, **sigma=0.25**

Una vez definidas las respuestas al impulso, aplicaremos éstas a las subtramas de datos correspondientes.

En primer lugar, definimos el tamaño de las subtramas. Por simplificación del modelo, vamos a tomar todas las subtramas iguales excepto la última, que incluirá el resto de la división. Así, cada subtrama de la primera a la penúltima tendrá como dimensión $Parte\ entera\ de\left(\frac{Longitud\ de\ trama\ de\ entrada}{Número\ de\ subtramas}\right)$. La última tendrá dimensión $Parte\ entera\ de\left(\frac{Longitud\ de\ trama\ de\ entrada}{Número\ de\ subtramas}\right) + Resto\ de\left(\frac{Longitud\ de\ trama\ de\ entrada}{Número\ de\ subtramas}\right)$.

A continuación, para aplicar las respuestas al impulso, contenidas en **B**, a cada subtrama, recurrimos a la función **filter(b,a,x)**. Esta función de MATLAB realiza el filtrado de los datos de entrada **x**, aplicando la función de transferencia definida por los coeficientes **b** y **a**. En nuestro caso, equivalente a un filtro FIR, **b** coincide con la respuesta al impulso (cada fila de **B**) y **a** es **1**. Además, **filter()** permite añadir las condiciones iniciales de los datos de entrada **zi** como entrada a la función, y nos devuelve las condiciones finales **zf** como salida (MathWorks, s.f.g). Esto nos va a permitir dar continuidad a nuestra respuesta, ya que las condiciones finales de la subtrama n serán la condiciones iniciales de la subtrama $n+1$. Los vectores de condiciones iniciales y finales tendrán la dimensión de

b menos **1**, que en nuestro caso será **Tmax-1**. Para la primera subtrama aplicaremos un vector de condiciones iniciales nulas, mediante la siguiente expresión:

```
zi = zeros(1,Tmax-1)
```

Ejecutaremos **filter()**, mediante un bucle de índice **i** que recorra las subtramas, de la siguiente forma:

```
[y,zf]=filter(h,1,x_subtrama,zi)
```

Donde **h** es la respuesta al impulso correspondiente a la fila **i** de **B** (**h = B(i,:)**), y **x_subtrama** es la selección de la trama de datos de entrada correspondiente a la subtrama **i**.

Las salidas **y** de **filter()** se encadenan formando el vector de salida, del cuál nos quedamos con su parte real.

En la figura 15 mostramos ejemplos del efecto de la función **multipath()** sobre un tren de pulsos rectangulares y sobre un pulso básico RCCA. En el capítulo VI, dedicado a los resultados del proyecto, retomaremos la función **multipath()** para ejemplificar diferentes situaciones de la cadena de transmisión sobre las que extraer conclusiones.

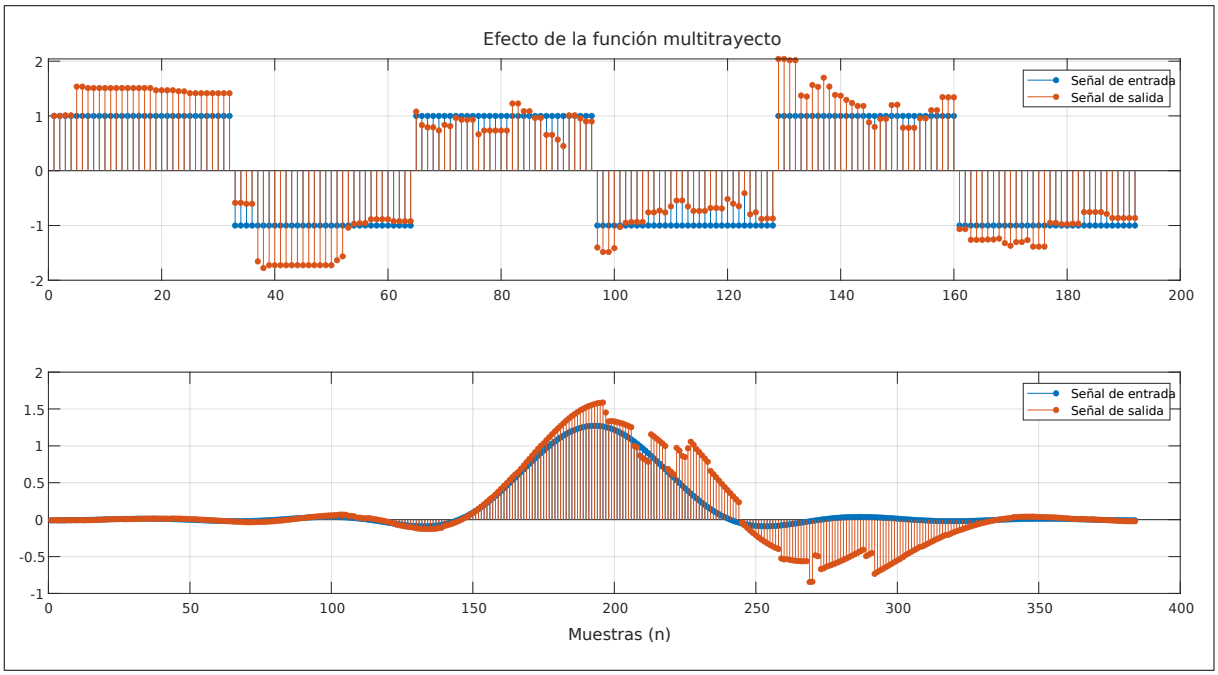

Figura 15. Efecto de la función **multipath()** *sobre diversas señales de entrada, con* **Nt=15**, **Pmax=40**, **Tmax=100**, **sigma=0.2**

Finalmente, realizamos una pequeña variación sobre las salidas que entrega **`multipath()`**, que nos permitirá observar la evolución de las respuestas al impulso del canal. Para ello, añadimos la matriz **`B`** como salida, quedando el formato final de la función como:

```
[x_mt,B] = multipath(x,Pmax,Tmax,sigma,Nt)
```

El código completo de la función **`multipath()`**, que acabamos de describir, está disponible en el anexo II.

5.1.2.1. Suavizado del cambio de las respuestas al impulso

Los cambios de las condiciones del canal, tal como los hemos considerado hasta ahora, se producen de forma brusca, es decir, no hay relación de continuidad entre filas sucesivas de la matriz **`B`**. Cada una de las respuestas al impulso se calcula de forma independiente, a partir de las mismas variables aleatorias, pero sin vinculación entre una y la posterior. Esta caracterización modela una propagación multitrayecto con cambios muy severos, que no refleja adecuadamente el comportamiento de la realidad.

La consideración de estas condiciones más desfavorables es válida en cuanto a objetivos de diseño para el ecualizador adaptativo, ya que permite evaluar el rendimiento del sistema bajo escenarios adversos y garantiza mayor robustez frente a este tipo de fluctuaciones.

No obstante, en la mayoría de entornos reales los cambios en las características del canal presentan cierta correlación temporal. Esto implica que las respuestas al impulso en instantes cercanos no son completamente independientes, sino que existe una continuidad estadística entre ellas.

Para modelar el sistema de manera más real, tomamos la solución de Díaz-Nafría (2021) en el código de **`multitrayecto()`**, consistente en aplicar un filtrado paso bajo sobre las variables aleatorias que definen los parámetros del canal (**`tau`**, **`theta`** y **`r`**). Este procedimiento introduce correlación entre valores sucesivos y suaviza la variación de las respuestas al impulso, reduciendo la brusquedad de los cambios. Sin embargo, hay que tener en cuenta que se requiere de un número elevado de variaciones del canal (**`Nt`**), para que el efecto del filtro sea apreciable. Si el número de filas de la matriz es reducido, la respuesta inicial del filtro se ve afectada por el efecto transitorio, lo que provoca que los valores filtrados no sean representativos.

El código de filtrado, mostrado a continuación, está *comentado* en nuestra versión de **`multipath()`**, pero haremos uso de él en simulaciones posteriores donde el número de variaciones sea significativo,

para ver su efecto.

```
% Filtro de suavizado sobre tau, theta y r

[b,a] = butter(6,0.1); % Filtro para suavizar las variaciones de las variables aleatorias
tau = filter(b,a,tau);
tau = max(2, round(tau)); % Para asegurar que tau sea entero y >=2
theta = filter(b,a,theta);
r = filter(b,a,r);
```

5.1.3. Adición de ruido blanco

El siguiente paso en la cadena de transmisión, como parte de la modelización del canal, consiste en la adición de ruido blanco gaussiano (AWGN, *Additive White Gaussian Noise*).

Para tal fin, recurrimos a la función **`ruido_blanco()`**, que genera una secuencia de ruido de longitud N y densidad espectral de potencia N_0. Estos dos parámetros se suministran como entradas junto con la frecuencia de muestreo f_m, la cual determina el ancho de banda de ruido considerado. En este modelo se asume que el contenido espectral de la señal se encuentra acotado por la frecuencia de Nyquist, definida como $f_m/2$.

Según este modelo, la frecuencia de muestreo a considerar para al cálculo de la potencia de ruido vendrá dada por $f_m = (f_S/D) \cdot b \cdot M_S$, siendo: f_S la frecuencia de muestreo de la señal de la que partimos ($44.1\,kHz$), D el factor de diezmado aplicado, b los bits utilizados en la cuantificación, que para la señal **`RyC.wav`** es son 5 y 8, respectivamente, y M_S las muestras por símbolo. Dando como resultado $f_m = 70560 \cdot M_S$.

El formato de la función es el siguiente: **`nx = ruido_blanco(No,N,fm)`**.

Si **`x_mt`** es la señal de salida después de aplicar la función multitrayecto a la señal de línea, según **`[x_mt,~] = multipath(x,Pmax,Tmax,sigma,Nt)`**, entonces debemos asegurarnos que **`N`** tenga la misma longitud que **`x_mt`**, mediante **`N = length(x_mt)`**.

Posteriormente, solo quedaría sumar los dos vectores **`x_mt + nx`** para obtener la señal a la salida del canal, afectada por la propagación multitrayecto y el ruido blanco gaussiano aditivo

En la figura 16, retomamos las señales de línea mostradas en la figura 12, pero esta vez representándolas junto con el resultado de las mismas al atravesar el canal ruidoso con propagación multitrayecto.

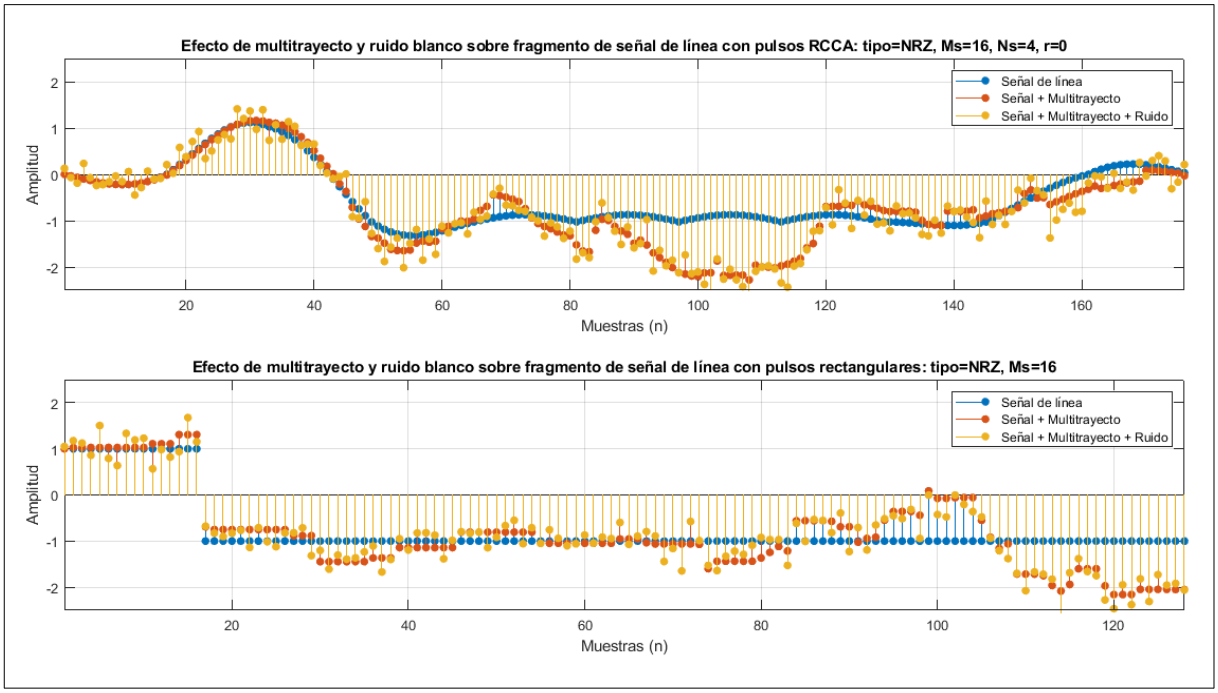

Figura 16. Efecto de multitrayecto (`Pmax=20`, `Tmax=50`, `sigma=0.2`, `Nt=5`) y ruido blanco (`No=1e-7`) sobre señal de línea.

5.1.4. Filtrado adaptado y muestreo en instantes óptimos

Una vez superado el canal de comunicación, llegamos a la fase de recepción, donde la señal pasará por varias etapas que sintetizaremos en dos funciones: **`FiltroAdaptado()`**, que ejecutará el filtro adaptado sobre la señal recibida y tomará muestras en los instantes óptimos, y **`DecisorAdaptativo()`**, que tomará la decisión sobre las muestras, aplicando técnicas de ecualización adaptativa (ver figura 17).

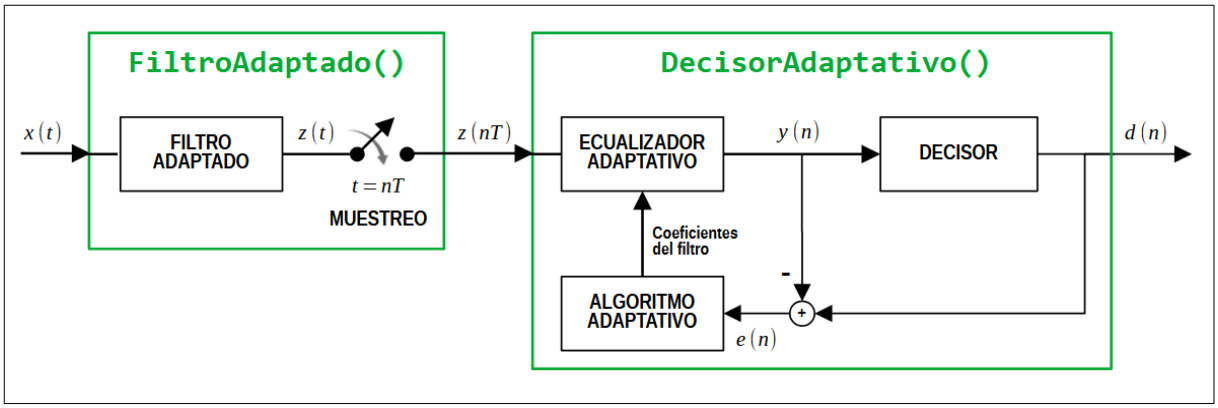

Figura 17. Esquema de la fase de recepción

El objetivo del filtro adaptado es proporcionar la máxima relación señal a ruido a su salida en los instantes óptimos de muestreo. A tal efecto, su respuesta al impulso se diseña como una versión temporalmente invertida y desplazada en T_P, de la forma de onda del pulso básico con la que se construyen los símbolos transmitidos, siendo T_P la duración de dicho pulso.

Esta configuración asegura que, al realizar la convolución con la señal recibida, la energía de cada símbolo se concentre en los instantes de muestreo correspondientes a $T_P + n \cdot T$, siendo T el periodo del símbolo y $n = \{ 0 , 1 , 2 , ... \}$. De esta forma, se minimiza la interferencia entre símbolos (ISI) y se maximiza la separación entre niveles en presencia de ruido aditivo (Sklar y Harris, 2021).

Así, si el pulso básico se define como $s(t)$, la respuesta al impulso del filtro adaptado tiene la siguiente forma: $h(t) = s(T_P - t)$.

La función **`FiltroAdaptado()`** es una versión reducida de la función de partida **`Rx_dig()`**, prescindiendo de la parte de la decisión, y quedándose con el filtrado adaptado y el muestro. Los códigos de ambas funciones están disponibles en el anexo I.

En la función, la definición de la respuesta al impulso como $h(t) = s(T_P - t)$ se realiza mediante la expresión **`h = fliplr(s_b(tipo,Ms,Ns,r))`**, que invierte temporalmente la expresión del pulso definida por la función conocida **`s_b()`**.

Posteriormente, se realiza la convolución de la señal de entrada **`x_l`** con **`h`**, obteniendo un vector de longitud **`[length(x_l)+length(h)-1]`**, o su equivalente **`[length(x_l)+Ms*Ns-1]`**. Este vector debe ser muestreado en los instantes óptimos, correspondientes a $T_P + n \cdot T$, con $n = \{ 0 , 1 , 2 , ... \}$.

El primer instante de muestreo corresponderá a T_P, o lo que es lo mismo **`Ms*Ns`**. El último instante de muestreo corresponderá a **`[length(x_l)+Ms*Ns-1]-[Ms*Ns-1]`**, es decir **`length(x_l)`**. La separación entre muestras debe ser T, o su equivalente **`Ms`**.

Esta selección de muestras óptimas se realiza en el código con las expresiones:

```
y = y(Ms*Ns:end-(Ms*Ns-1))
ym = y(1:Ms:end)
```

siendo **`y`** el resultado de la convolución, que se recorta entre el primer y el último instante de muestreo, e **`ym`** el vector de muestras. Estos dos vectores son las salidas de la función.

En las figuras 18 y 19, representamos la evolución de dos señales de línea de 8 bits desde su

generación hasta la salida del filtro adaptado y su muestreo en los instantes óptimos.

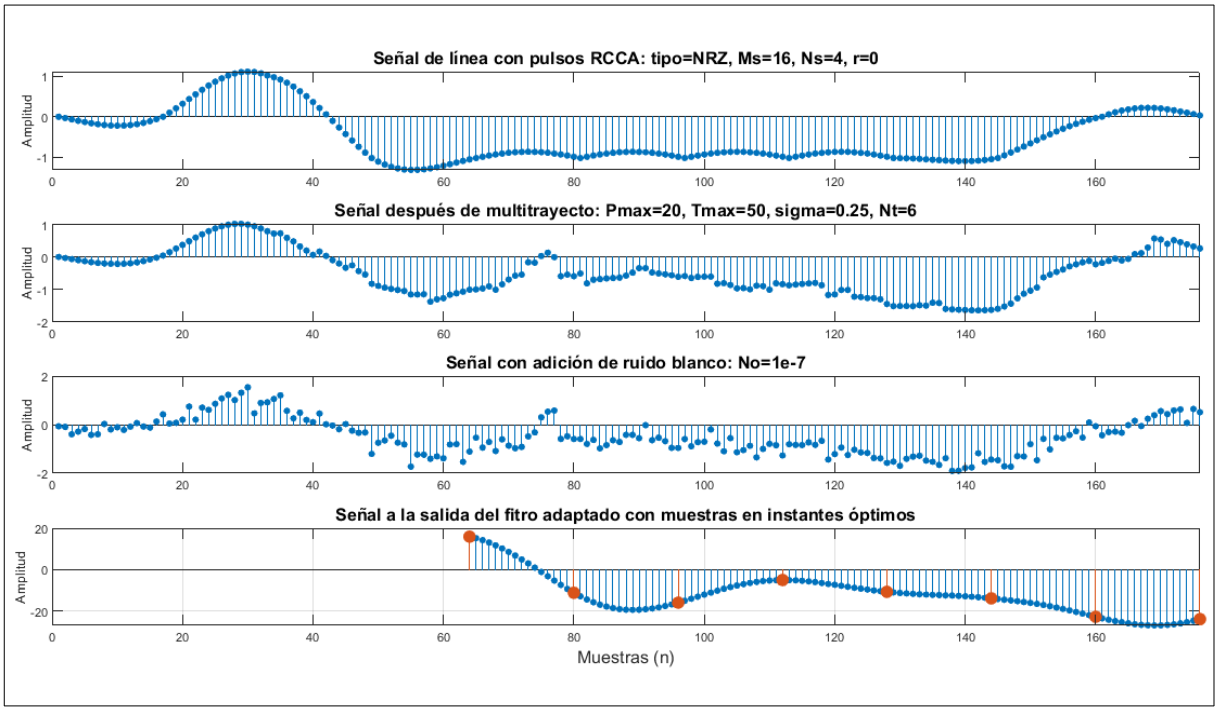

Figura 18. Evolución de señal de línea con pulsos RCCA, desde su generación hasta la salida del filtro adaptado. Las muestras de la salida del filtro están desplazadas para hacerlas coincidir temporalmente.

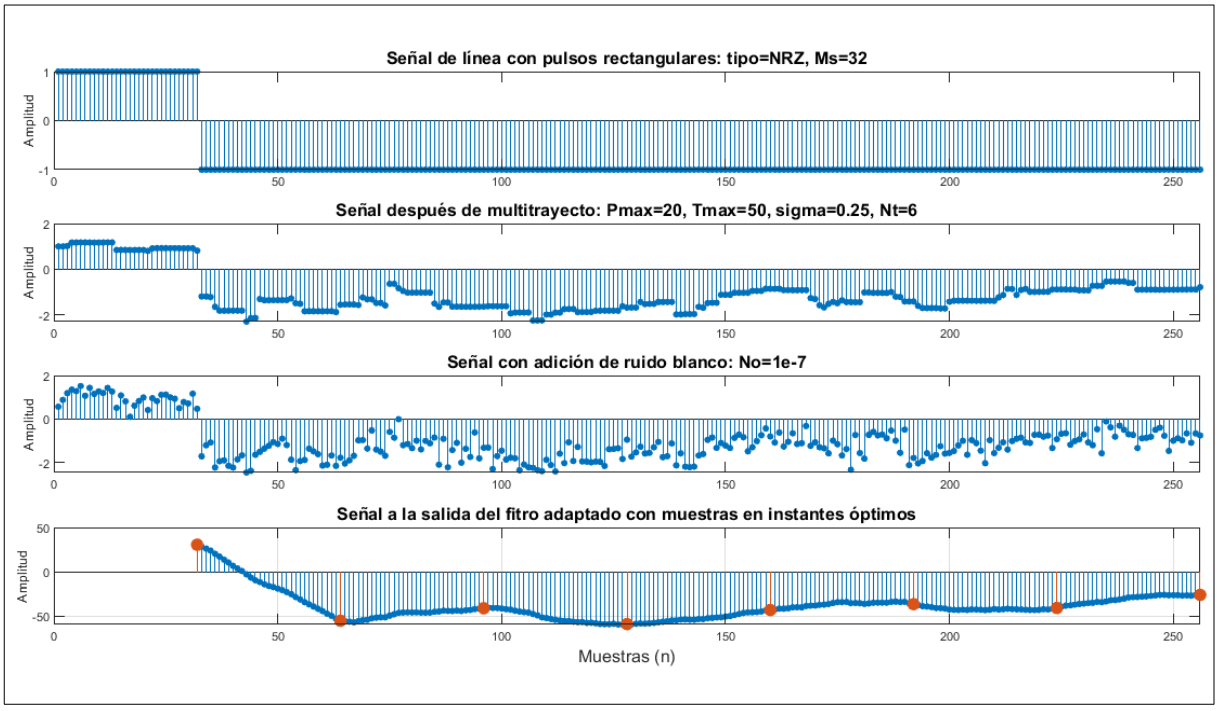

Figura 19. Evolución de señal de línea con pulsos rectangulares, desde su generación hasta la salida del filtro adaptado. Las muestras de la salida del filtro están desplazadas para hacerlas coincidir temporalmente.

5.1.5. Ecualización adaptativa y decisión

El siguiente paso en la fase de recepción, tras la obtención de las muestras a la salida del filtro adaptado, es la etapa de decisión. En esta etapa se determina el símbolo recibido, identificando el punto de la constelación más próximo a la muestra obtenida. La constelación en recepción está formada por los valores esperados de la señal en los instantes de muestreo óptimos. Estos valores tendrán asociados una codificación binaria, con la que construiremos la secuencia digital decodificada de salida.

La función de partida **`decisor()`** se encarga de generar la secuencia decodificada a partir de las muestras, según el siguiente formato:

```
[d,y] = decisor(xm,a,c)
```

Siendo:

- **`xm`**: Secuencia de muestras de entrada.
- **`a`**: Constelación de los valores esperados de mayor a menor, como vector columna.
- **`c`**: Códigos correspondientes a los valores de **`a`**, como vector columna.
- **`y`**: Secuencia de valores reconstruidos en base a la constelación **`a`**.
- **`d`**: Secuencia de datos decodificados de salida, a partir de **`y`** y de **`c`**.

Hay que señalar que la función **`s_b()`**, con la que construimos los pulsos básicos, genera símbolos de energía **`Ms`**, de forma que la señal generada mediante estos símbolos tenga potencia media unitaria. Nótese que **`Ms`** equivale a T (periodo del símbolo), y la energía del símbolo está determinada por $E = P \cdot T$, siendo P la potencia. Para generar señales de potencia $P = 1$, se debe cumplir que $E = T$.

En la función generadora de la señal de línea **`s_linea()`** se establece la potencia de la señal de línea **`s`**, mediante la multiplicación de los símbolos **`s_b()`** por la amplitud, que viene determinada por la raíz cuadrada de la potencia **`sqrt(s)`**. Asumiendo una modulación binaria bipolar, que es la que utiliza **`s_linea()`**, tendremos símbolos **`sqrt(s)*s_b()`** y **`-sqrt(s)*s_b()`**, que, considerando un canal sin pérdidas, nos llevará a unos valores esperados en el instante de muestreo de **`sqrt(s)*Ms`** y **`-sqrt(s)*Ms`**. En nuestras simulaciones, por simplicidad, estamos asumiendo que la potencia de la señal de línea es de 1 W, es decir, **`s=1`**, por lo que la constelación de valores esperados en recepción será **`a = [Ms;-Ms]`**, correspondientes a los códigos **`c = [1;0]`**.

La utilización del decisor inmediatamente después del filtro adaptado y del muestreo constituye un modelo válido cuando el canal no introduce distorsión. Sin embargo, tal y como hemos visto en el capítulo II, para compensar la degradación de la señal provocada por el canal es necesario incorporar técnicas de ecualización en el receptor. Además, si consideramos que el canal tiene un comportamiento variable en el tiempo, debemos implementar la ecualización mediante algoritmos adaptativos, capaces de ajustar dinámicamente los coeficientes del ecualizador para adaptarse a las variaciones del canal.

El diseño de una función en MATLAB que incorpore la ecualización adaptativa como parte del decisor es uno de los objetivos de este proyecto. Con este fin, vamos a incorporar un filtro FIR a nuestro decisor, que adapte sus coeficientes siguiendo el algoritmo RLS, según lo expuesto en la sección 2.3.

La función diseñada **`DecisorAdaptativo()`** añade la ecualización adaptativa a **`decisor()`**, que sigue encargándose de realizar la toma de decisión sobre la señal ecualizada por el filtro FIR, según el esquema de la figura 20.

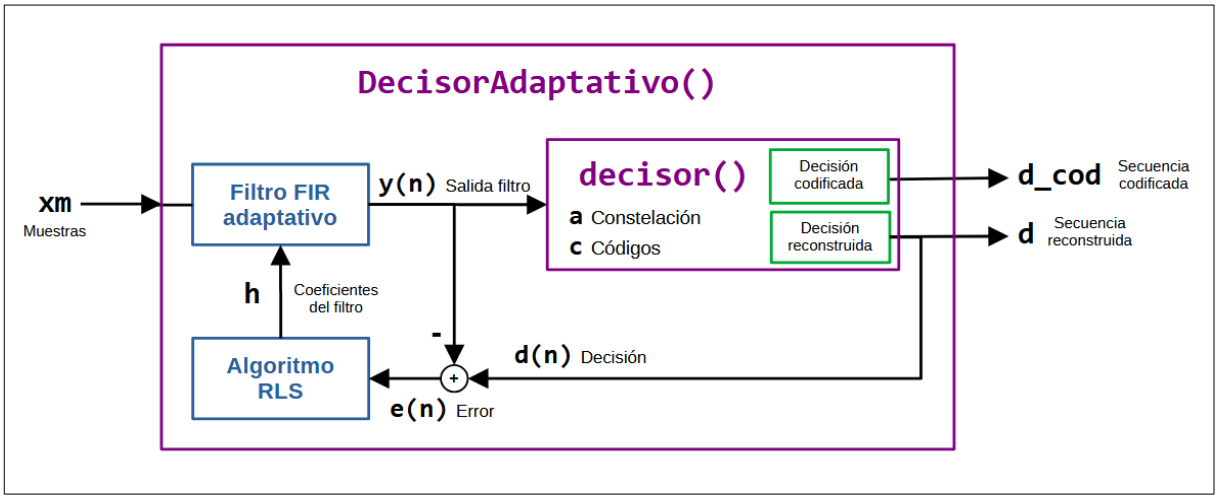

Figura 20. Esquema de la función **`DecisorAdaptativo()`**

El formato completo de la función es el siguiente:

```
[d_cod,d,y,e,h_out,P_out,h_hist] = DecisorAdaptativo(xm,a,c,M,lambda,h,P)
```

siendo las entradas:

- **`xm`**: Secuencia de muestras de entrada.
- **`a`**: Constelación de los valores esperados de mayor a menor, como vector columna.
- **`c`**: Códigos correspondientes a los valores de **`a`**, como vector columna.

- **`M`**: Orden del filtro FIR que realiza la ecualización.

- **`lambda`**: Factor de olvido λ del algoritmo RLS. Sirve para ponderar las muestras de forma exponencial decreciente, desde la más reciente a la más antigua. Valor típico: $\lambda = 0.99$.

- **`h`**: Inicialización del vector de coeficientes del filtro $\boldsymbol{h} = [h_0, h_1, \ldots, h_{M-1}]^T$, de dimensión M. El algoritmo RLS define un vector inicial nulo de longitud M, pero en nuestro caso vamos a utilizar un vector $\boldsymbol{h} = [1, 0, \ldots, 0]^T$, que no altera la entrada en la primera iteración.

- **`P`**: Inicialización de la matriz $\boldsymbol{P}$, inversa de la matriz de autocorrelación: $\boldsymbol{P} = \boldsymbol{R}_{xx}^{-1}$. $\boldsymbol{P}$ tiene dimensión $M \times M$ y se inicializa con la expresión $\boldsymbol{P} = (1/\delta)\,\boldsymbol{I}$, siendo $\boldsymbol{I}$ la matriz identidad de dimensión $M \times M$, y δ un número positivo pequeño, de valor típico 0.01.

y las salidas:

- **`d_cod`**: Secuencia de datos codificados de salida. Codificación de las decisiones reconstruidas según **`c`**.

- **`d`**: Secuencia de decisiones reconstruidas en base a la constelación **`a`**.

- **`y`**: Secuencia de salidas del ecualizador adaptativo, previas al decisor.

- **`e`**: Vector de error entre **`d`** e **`y`** (**`e=d-y`**).

- **`h_out`**: Último valor del vector **`h`**. Utilizado como entrada de la siguiente trama, en caso de estar trabajando con tramas encadenadas.

- **`P_out`**: Último valor de la matriz **`P`**. Utilizado como entrada de la siguiente trama, en caso de estar trabajando con tramas encadenadas.

- **`h_hist`**: Matriz que almacena el histórico del vector **`h`** a lo largo del proceso. Esta salida no tiene fines operativos. Se utiliza para poder mostrar la evolución de los coeficientes del filtro en el tiempo.

Es importante señalar que la ecualización funciona como un filtro FIR de orden M, por lo que debemos añadir un vector de $M-1$ condiciones iniciales a la secuencia de entrada, desde $x(-M+1)$ hasta $x(-1)$. Este vector, que normalmente va a ser un vector nulo de dimensión $M-1$, se añade a la secuencia de entrada antes de llamar la función. El motivo de esto se entenderá más adelante, cuando diseñemos el bloque de comLAB correspondiente a la decisión adaptativa, por la manera en la que este

bloque hace la llamada a la función. A efectos prácticos, esto no va afectar a la simulación en MATLAB. Por lo tanto, antes de ejecutar la función sobre la secuencia de entrada **xm**, debemos concatenarla con **M-1** ceros al principio:

```
xm = [zeros(1,M-1),xm]
```

A nivel interno, la función sigue el algoritmo RLS descrito en la sección 2.3.3.

Mediante un bucle de índice **n** se recorre el vector de entrada, y el ecualizador procesa la señal muestra a muestra. Para cada instante, se selecciona una ventana de **M** muestras de la secuencia de entrada que alimenta el filtro FIR, invirtiendo su orden para que las muestras más recientes coincidan con los coeficientes más bajos del filtro. A continuación, se calcula la salida del ecualizador como el producto escalar entre la ventana de muestras **x** y los coeficientes actuales **h**. Esta salida **y(n)** se pasa por la función **decisor()**, que determina el símbolo de la constelación **a** más cercano y devuelve tanto su valor decidido **d(n)** como los bits asociados a partir de **c**. Estos datos se van escribiendo sobre los vectores de salida **d_cod** y **d** en cada iteración.

Después, se calcula el error **e(n)** como la diferencia entre la decisión **d(n)** y la salida del ecualizador **y(n)**. La corrección de los coeficientes **h** para la siguiente iteración dependerá de **e(n)** y del vector de ganancia de Kalman **k**, que se calcula a partir de la matriz **P** y de la ventana de entrada **x**. La matriz **P** también se actualiza para la siguiente iteración, a partir de **lambda**, **k** y **x**.

Además de **d_cod** y **d**, en cada iteración se van construyendo los vectores de salida **d**, **y** y **e**, y la matriz **h_hist**.

Los últimos valores de **h** y **P** se devuelven como las salidas **h_out** y **P_out**, respectivamente. Estas salidas, serán de utilidad para dar continuidad cuando se procesen señales continuas divididas en tramas.

El código completo de **DecisorAdaptativo()** está disponible en el anexo II. En el siguiente apartado, dedicado a ejemplificar el ciclo de transmisión completo, se ilustrará el uso de la función.

5.1.6. Ejemplo de simulación del ciclo de transmisión y medidas de rendimiento

La simulación del ciclo de transmisión completo, según la figura 10, consiste en la aplicación sucesiva de las distintas funciones vistas hasta el momento sobre la señal de entrada seleccionada. Al final,

añadiremos diversas medidas y representaciones gráficas que nos permitan comprobar la validez de los modelos empleados.

En primer lugar, definimos los parámetros correspondientes y generamos la señal de línea:

```
% Generación de señal de línea

s=1; % Potencia 1 W
Rb=1e6; % Tasa binaria 1 Mpbs
signal=1; % Señal elegida Ryc.wav
inicio=10000; % Primera muestra
fin=40000; % Última muestra
Ms=32; % Muestras por símbolo
tipo=1; % Tipo de codificación de línea NRZ
Ns=1; % Elegimos pulsos rectangulares
r=0; % Roll-off (no relevante en pulsos rectangulares)

[DATOS_IN,X1,~]=s_linea(s,Rb,signal,inicio,fin,Ms,tipo,Ns,r);
```

Se seleccionan 30001 muestras de la señal **`RyC.wav`**, que tras aplicar el diezmado de orden 5 y la cuantificación de 8 bits, generan una secuencia binaria de 48008 bits, que recogemos en la salida **`DATOS_IN`** para su uso posterior. La salida con la señal de línea de pulsos rectangulares es **`X1`**, que es una secuencia de 1536256 muestras.

A continuación, utilizamos **`X1`** como entrada de la función que simula la propagación multitrayecto, definiendo previamente los parámetros deseados:

```
% Paso por canal multitrayecto

Pmax=20; % Número máximo de trayectos
Tmax=320; % Retardo máximo (en muestras)
sigma=0.25; % Desviación típica de la amplitud de los pulsos secundarios
Nt=4; % Número de subtramas (variaciones del canal)

[X2,B]=multipath(X1,Pmax,Tmax,sigma,Nt);
```

La elección de **`Nt`** bajo tiene el único objetivo de facilitar la representación gráfica. El retardo máximo de 320 muestras hará que el efecto de los trayectos secundarios pueda notarse hasta en 10 bits posteriores. La matriz **`B`** de salida guarda las respuestas al impulso de cada variación del canal, para su posterior representación. **`X2`** es la secuencia de salida a la que añadimos ruido blanco en la siguiente etapa:

```
% Adición de ruido blanco

No=1e-7; % DEP de ruido
fm=(44100/5)*8*Ms; % Para calcular el AB de ruido
N=length(X2); % Longitud de trama de entrada para ruido_blanco()

X3=X2+ruido_blanco(No,N,fm);
```

La secuencia **X3** es la señal resultante de atravesar el canal con propagación multitrayecto y ruido blanco. La dimensión de **X3** sigue siendo la misma que **X1** y **X2**, 1536256 muestras. Esta secuencia sirve de entrada a la primera etapa de la recepción:

```
% Paso por filtro adaptado y muestreo

X4=FiltroAdaptado(X3,tipo,Ms,Ns,r);
```

La salida **X4**, que corresponde a las muestras en los instantes óptimos tras el filtrado adaptado, tiene una dimensión 48008 muestras, coincidente con la dimensión de **DATOS_IN**, ya que estamos utilizando modulación binaria. Estas muestras pasan a la etapa de decisión, que nos devolverá la secuencia decodificada de salida:

```
% Etapa de decisión
M=16; % Orden del filtro adaptativo
lambda=0.99; % Factor de olvido del algoritmo RLS
delta=0.01; % Para inicializar la matriz P
P=(1/delta)*eye(M); % Inicialización de matriz P
h = [1;zeros(M-1,1)]; % Inicialización de h como [1;0;...;0]
X4_ext = [zeros(1,M-1),X4]; % Se añaden M-1 valores iniciales a cero
a=[Ms,-Ms]';c=[1,0]'; % Amplitudes y códigos para decisor

% Salida del decisor (sin ecualización adaptativa)
[DATOS_SOLO_DECISOR,d_SOLO_DECISOR] = decisor(X4,a,c);

% Salida del decisor adaptativo
[DATOS_DECIS_ADAP,d_DECIS_ADAP,y,e,h_out,P_out,h_hist] = DecisorAdaptativo(X4_ext,a,c,M,lambda,h,P);
```

La decisión se realiza con y sin ecualización adaptativa, con el fin de evaluar el efecto de esta última. Las secuencias decodificadas son, respectivamente, **DATOS_SOLO_DECISOR** y **DATOS_DECIS_ADAP**, que utilizaremos para comparar con los datos de entrada **DATOS_IN** y comprobar el número de errores totales en la transmisión y la tasa de error de bit. Para ello, utilizamos el siguiente código que contrasta los resultados:

```
% Con decisor adaptativo

ERRORES_DECIS_ADAP = sum(logical(DATOS_IN') ~= DATOS_DECIS_ADAP); % Errores
BER_DECIS_ADAP=ERRORES_DECIS_ADAP/length(DATOS_IN); % Tasa de error de bit

% Solo decisor
ERRORES_SOLO_DECISOR = sum(logical(DATOS_IN') ~= DATOS_SOLO_DECISOR); % Errores
BER_SOLO_DECISOR=ERRORES_SOLO_DECISOR/length(DATOS_IN); % Tasa de error de bit
```

En la simulación de transmisión realizada como ejemplo obtenemos los siguientes resultados:

- Con el decisor adaptativo:
  - Número de errores en la transmisión: **`ERRORES_DECIS_ADAP = 55`**.
  - Tasa de error de bit (BER): **`BER_DECIS_ADAP = 0.0011`**.
- Sólo con el decisor, sin ecualización adaptativa:
  - Número de errores en la transmisión: **`ERRORES_SOLO_DECISOR = 2511`**.
  - Tasa de error de bit (BER): **`BER_SOLO_DECISOR = 0.0523`**.

En vista de los datos obtenidos en este ejemplo, podemos concluir que la ecualización adaptativa ha conseguido mitigar sensiblemente, aunque no del todo, los efectos de la distorsión del canal.

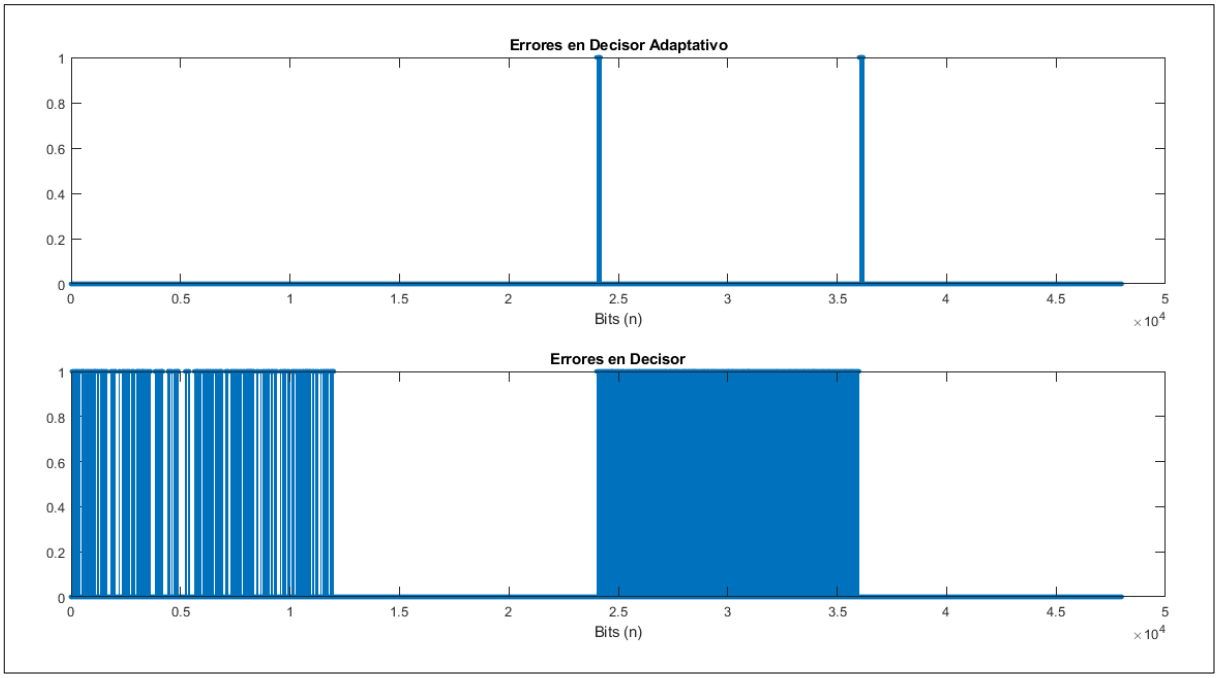

Figura 21. Comparación de errores en la decisión, con y sin ecualizador adaptativa.

Para analizar más en detalle los motivos de los errores en la transmisión, y las diferencias entre los dos decisores, mostramos una gráfica donde identificamos dónde se han producido las diferencias (ver figura 21).

Se puede observar que los errores del decisor no adaptativo se encuentran concentrados en dos bloques bien diferenciados. Hay que recordar que el parámetro que controla las variaciones del canal se estableció como **`Nt=4`**, y que las subtramas con condiciones distintas de canal tienen aproximadamente el mismo tamaño. En este ejemplo, cada subtrama tiene 384064 muestras, que corresponden a 12002 bits. Por tanto, los errores del decisor no adaptativo se concentran en la primera y tercera subtrama.

A su vez, los pocos errores del decisor adaptativo se producen al principio de la tercera y la cuarta subtrama.

Para ver las condiciones del canal en esos instantes, en la figura 22 representamos la respuesta al impulso del canal para cada **`Nt`**, mediante la matriz **`B`**, que obtuvimos como salida en la función **`multipath()`**.

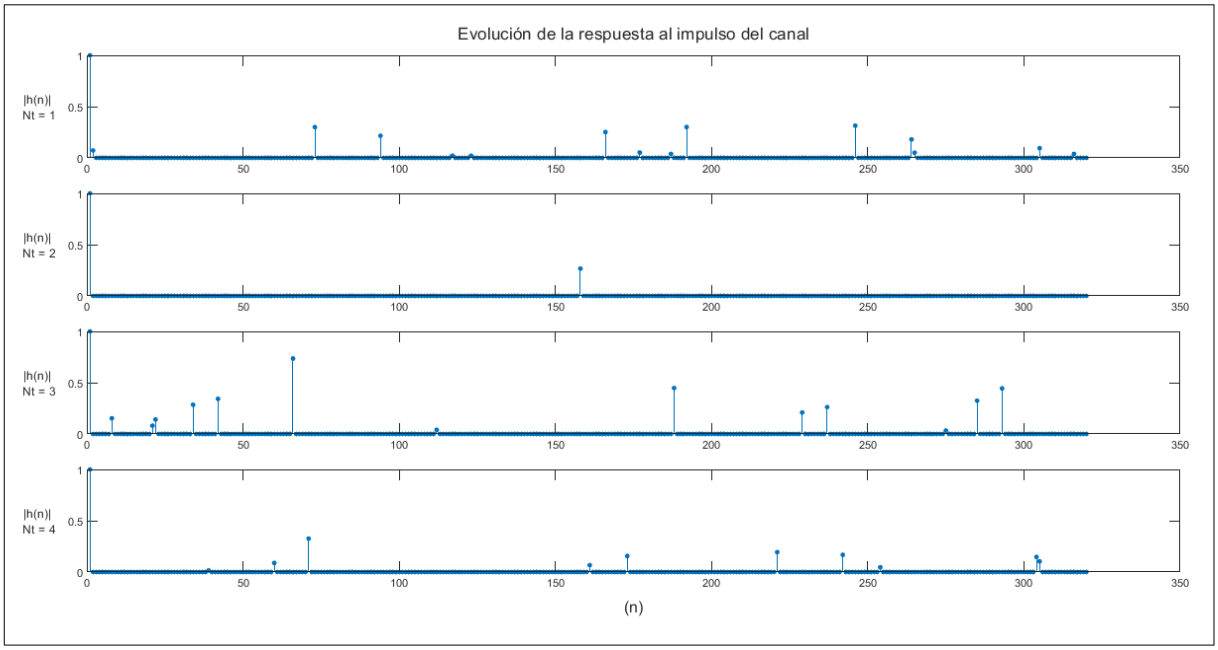

Figura 22. Evolución del módulo de la respuesta al impulso del canal en cada variación.

A simple vista, se puede apreciar como el peso de los trayectos secundarios es más significativo para las subtramas primera y tercera, especialmente esta última, que tiene contribuciones relevantes con retardos elevados. Esto explica que el decisor no adaptativo acumule errores en estas subtramas, especialmente en la tercera. Sin embargo, en las subtramas segunda y cuarta, que tienen respuestas al impulso menos desfavorables en cuanto a distorsión, las decisiones son correctas.

El decisor sin ecualización tiene más errores cuando las contribuciones secundarias relevantes están más alejadas, especialmente si los retardos de estas contribuciones superan el periodo de símbolo. Este caso se corresponde con el *desvanecimiento selectivo en frecuencia*, que se planteó en la sección 2.2.2.

En cambio, el decisor adaptativo es capaz de compensar en gran medida los efectos introducidos por el canal. Aunque requiere un periodo inicial de adaptación, que en el caso de las subtramas tercera y cuarta supone decisiones erróneas al principio, logra ajustar sus coeficientes para mitigar la interferencia producida por los trayectos secundarios y optimizar las decisiones y la tasa de error.

En la figura 23, a partir de los vectores de salida **`d`** (decisiones), **`y`** (salidas del filtro) y **`e`** (error entre **`d`** e **`y`**), construimos dos representaciones gráficas que muestran la evolución del error cuadrático y la comparación entre las decisiones y las salidas del ecualizador.

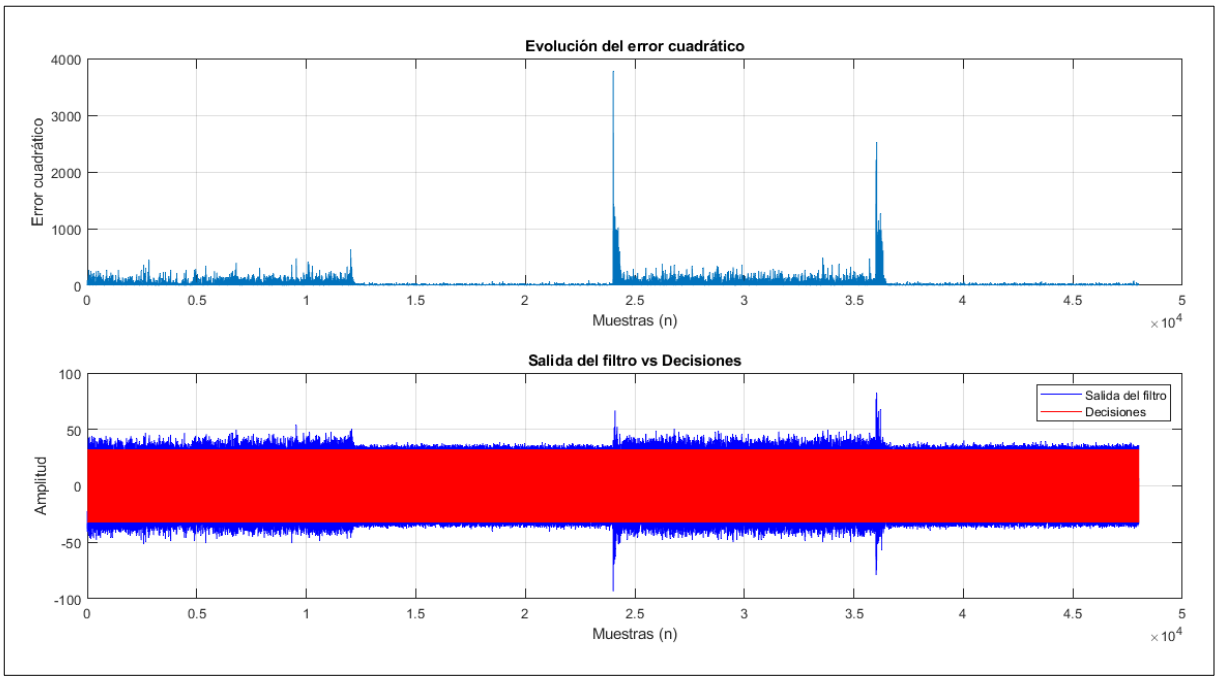

Figura 23. Progreso del error cuadrático y comparativa entre decisiones y salidas del filtro FIR.

En la gráfica se aprecia como el error cuadrático tarda más en converger para las subtramas tercera y cuarta, lo que provoca los errores de decisión. En la primera trama, a pesar de que el error se mantiene por encima de los valores de la cuarta, y parecidos a los de la tercera, la convergencia se produce mucho más rápido, y se mantiene controlado por debajo de valores que no producen decisiones equivocadas.

Por último, resulta relevante observar la evolución de los coeficientes del filtro FIR a lo largo del proceso, observando su margen de variación y su convergencia. Para ello, utilizamos la matriz **`h_hist`**,

de la que representamos las cuatro primeras filas, a modo de ejemplo, correspondientes a h_0 , h_1 , h_2 , h_3 (ver figura 24).

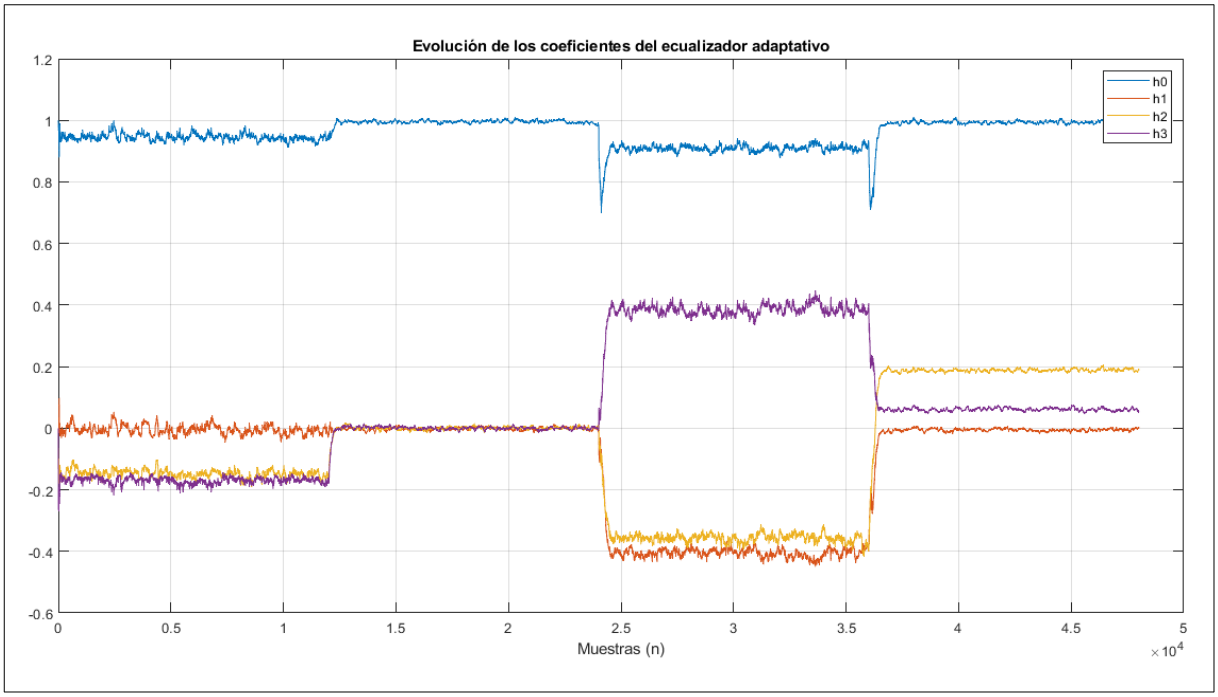

Figura 24. Evolución de los cuatro primeros coeficientes del filtro FIR adaptativo.

El comportamiento de los coeficientes guarda similitud con la evolución el error cuadrático. La convergencia es más rápida en las subtramas primera y segunda, sin embargo, la evolución es más estable en las subtramas segunda y cuarta, coincidente con los periodos donde el error cuadrático se mantenía más controlado.

El script de MATLAB, correspondiente al ejemplo descrito en este apartado y a las representaciones gráficas, está disponible en el anexo II como **`Ciclo_de_transmision.m`**

En el capítulo VI se retomarán las simulaciones de este ciclo de transmisión, enfocándose en analizar la influencia de distintos parámetros sobre los resultados, con el propósito de extraer conclusiones relevantes para el proyecto.

5.2. Implementación del modelo en comLAB (Simulink)

Una vez desarrollados los modelos de propagación multitrayecto y decisión adaptativa para su simulación en la consola de MATLAB, el siguiente paso del proyecto es trasladarlos al entorno gráfico de comLAB, implementado sobre la plataforma Simulink.

El factor clave en la traslación de los modelos de MATLAB a Simulink es cómo se tratan las secuencias de datos en uno y otro entorno. Mientras que en MATLAB tomábamos las señales completas, para su procesado a través del ciclo de simulación, en Simulink vamos a trocear las señales en tramas de tamaño fijo, que se procesarán de forma secuencial, una tras otra. Este enfoque por tramas nos permite acercarnos a lo que sería la simulación en tiempo real, en la que los datos llegan de forma continua. Para mantener esa continuidad, es necesario acarrear información de estado entre tramas consecutivas, de modo que el procesamiento de una trama en un bloque determinado está condicionada por los datos heredados de la trama anterior.

El punto de partida en el entorno de comLAB es el modelo de cadena de transmisión que mostrábamos en la figura 7. A partir de este modelo, vamos a simplificar la representación suprimiendo por el momento la parte de monitorización, centrándonos sólo en los bloques propios del ciclo de transmisión.

A su vez, vamos a deshabilitar los bloques de este ciclo que no sean estrictamente necesarios para el objetivo de nuestro proyecto. Tanto el canal multitrayecto como la decisión con ecualización adaptativa se sitúan en la cadena de transmisión después de la generación de la señal de línea por parte del modulador, a partir de la señal codificada. Cómo se codifica la señal de origen no es relevante para nuestro estudio, por lo que sólo vamos a dejar la parte de codificación estrictamente necesaria para generar la señal codificada: la señal original muestreada y la cuantificación. Por lo tanto, deshabilitamos los bloques de diezmado/interpolación y codificación/decodificación de canal. El lienzo del que partimos, con las simplificaciones asumidas, se muestra en la figura 25.

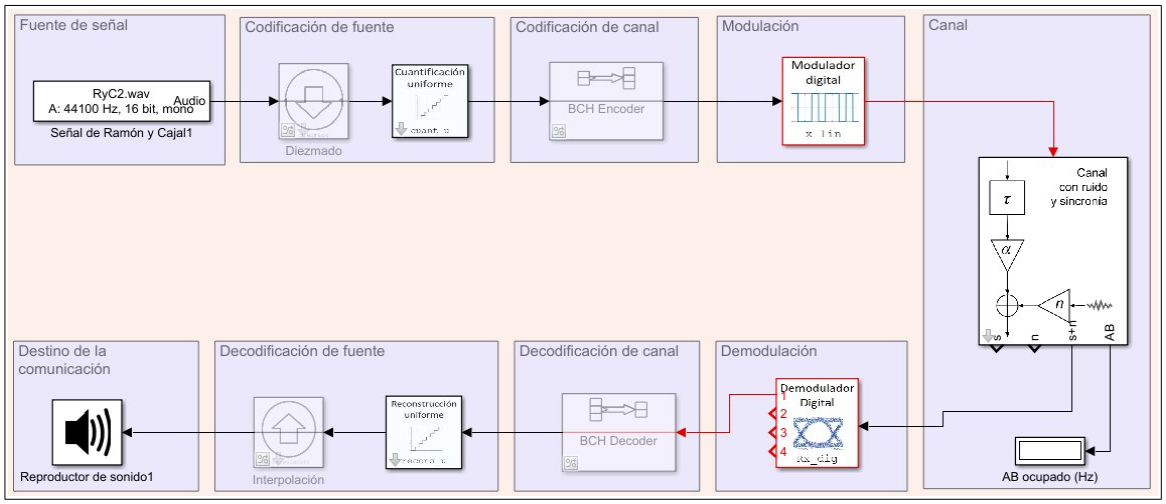

Figura 25. Sistema de transmisión simplificado.

Sobre este sistema se van a incorporar los bloques diseñados para simular la propagación multitrayecto y la decisión adaptativa, que son los objetivos principales de este proyecto.

Nuestro modelo de multitrayecto solo considera el retardo y la atenuación de los trayectos secundarios respecto al principal, por lo que es complementario con un modelo de canal que asuma el retardo y la atenuación del trayecto principal, además de la adición de ruido. Por este motivo, en comLAB podemos intercalar nuestro bloque de multitrayecto entre el bloque de modulación, que genera la señal de línea, y el bloque de canal ruidoso con atenuación.

Respecto al bloque de decisión adaptativa, su entrada debe ser las muestras realizadas después de aplicar el filtro adaptado a la salida del canal. El bloque demodulador de comLAB incorpora estas muestras como una de sus salidas (puerto 3), que aprovecharemos como entrada de nuestro bloque. Así, en vez de sustituir el bloque demodulador, lo aprovecharemos como filtro adaptado y muestreador en instantes óptimos. Además, la salida de su primer puerto, que devuelve la señal decodificada tras realizar la decisión simple y asociar los códigos correspondientes a estas decisiones, la podemos aprovechar para comparar las salidas decodificadas de los dos decisores, con y sin ecualización adaptativa. En la figura 26 se muestra el esquema del sistema objetivo, con la incorporación de los nuevos bloques.

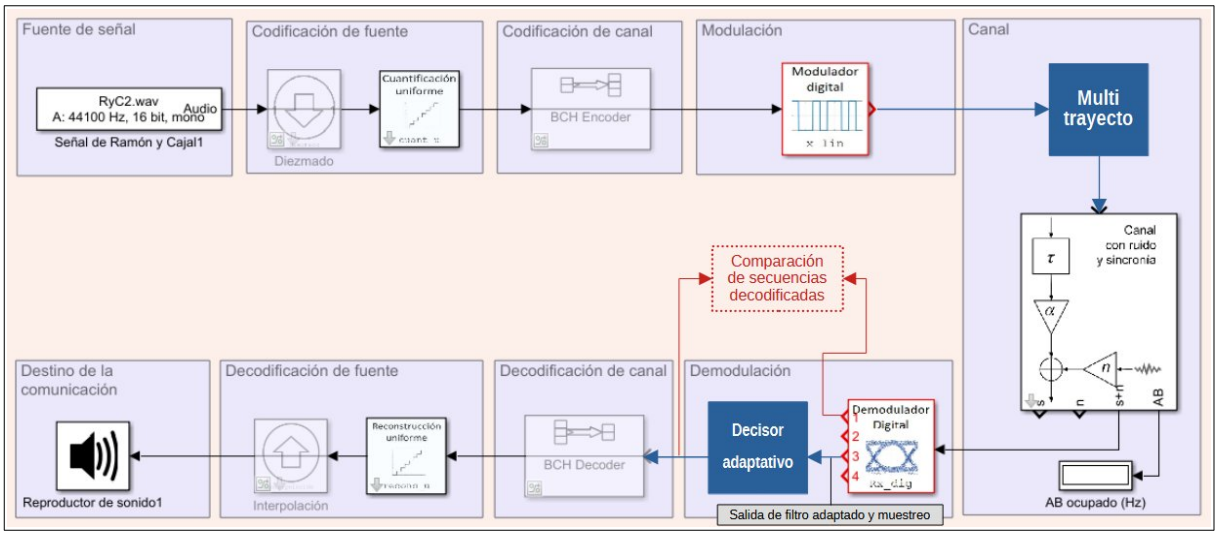

Figura 26. Sistema de transmisión con los nuevos bloques.

A continuación, pasamos a describir la implementación de cada uno de los nuevos bloques mediante *MATLAB S-Functions* de nivel 2, que es el formato utilizado en comLAB.

5.2.1. Bloque de multitrayecto

La *S-Function* de nivel 2, que define el funcionamiento del bloque que modela la propagación multitrayecto, es una traslación directa de la función desarrollada para ser ejecutada en la consola de MATLAB, **`multipath()`**, que se expuso en el apartado 5.1.2.

La función **`multipath()`** basaba su funcionamiento en la llamada a la función **`filter()`** para cada una de las subtramas en la que se dividía la secuencia de entrada, aplicando en cada ejecución una respuesta al impulso diferente, que se obtenía de la matriz **`B`**, generada previamente. En cada ejecución de **`filter()`** se generaban unas condiciones finales, que se acarreaban como condiciones iniciales de la subtrama posterior.

La principal diferencia en la función para Simulink, que nombramos como **`simulink_Multipath.m`**, radica en que, al trabajar con la secuencia troceada en tramas, además de acarrear las condiciones finales de la ejecución de **`filter()`** entre subtramas, también lo debemos hacer entre las diferentes tramas. Para ello, recurrimos a los vectores *Dwork*, que permiten almacenar valores entre iteraciones del simulador, es decir, entre la ejecución de las diferentes tramas por parte del bloque (MathWorks, s.f.h).

Además, en la traslación de la función de MATLAB, sólo la entrada correspondiente a la señal de línea **`x`** se mantiene como entrada en el bloque de Simulink. El resto de entradas (**`Pmax`**, **`Tmax`**, **`sigma`** y **`Nt`**) pasan a ser parámetros del bloque, que se introducirán desde la máscara que crearemos para ello.

En la función se invocan cuatro métodos:

- **`SetInputPortDimensions`**: Establece las dimensiones de entrada y salida del bloque en función del tamaño de la señal de entrada. Asegura que entrada y salida tengan dimensiones coherentes.

- **`PostPropagationSetup`**: Se ejecuta una vez determinada la estructura de la simulación. En este caso, se define un vector *Dwork*, para almacenar el estado final de la última ejecución de **`filter()`** en cada trama. Esta información se usará como estado inicial en la siguiente.

- **`Start`**: Se ejecuta una vez al comienzo de la simulación (primera trama). En este caso se inicializa el vector *Dwork* a un vector todos ceros de dimensión **`Tmax-1`**, que se utilizará como condiciones iniciales en la primera ejecución de **`filter()`**, de la misma forma que hacíamos en **`multipath()`**.

- **`Outputs`**: Es la parte donde se define el funcionamiento del bloque. Se ejecuta en cada paso de la simulación para procesar la señal de entrada y calcular la salida. Su estructura es casi idéntica a la función **`multipath()`**. En primer lugar, se crean las matrices **`P`**, **`tau`**, **`theta`** y **`r`**, a partir de valores aleatorios. Con esto se construye la matriz **`B`**, que contiene las respuestas el impulso para cada variación del canal. Mediante un bucle, se aplica la función **`filter()`** a cada subtrama con su respuesta el impulso correspondiente, pasando el estado final como estado inicial de la siguiente iteración, y guardando el resultado de cada ejecución en el vector **`x_mt`**. Finalmente, se guarda el último estado final del filtro en el vector *Dwork* y se entrega el vector resultante **`x_mt`** como salida del bloque.

El código completo de **`simulink_Multipath.m`** está disponible en el anexo III.

Para incorporar el nuevo módulo en comLAB, añadimos desde la librería de Simulink un bloque del tipo *Level-2 MATLAB S-Function* y en su configuración lo asociamos con **`simulink_Multipath`**. Posterioremente, creamos la máscara del bloque mediante *Mask/Create Mask*, donde asociamos los parámetros de la función con entradas de diálogo de la máscara, para permitir que el usuario los pueda modificar de forma sencilla. Además, se insertan comentarios para describir el módulo y los propios parámetros.

El botón de ayuda del cuadro de diálogo se asocia al artículo creado en glossaLAB sobre *Desvanecimiento Multitrayecto*, de forma que al pulsar el botón se abra el artículo creado en el navegador (https://www.glossalab.org/wiki/Draft:Desvanecimiento_multitrayecto). Ver figura 27.

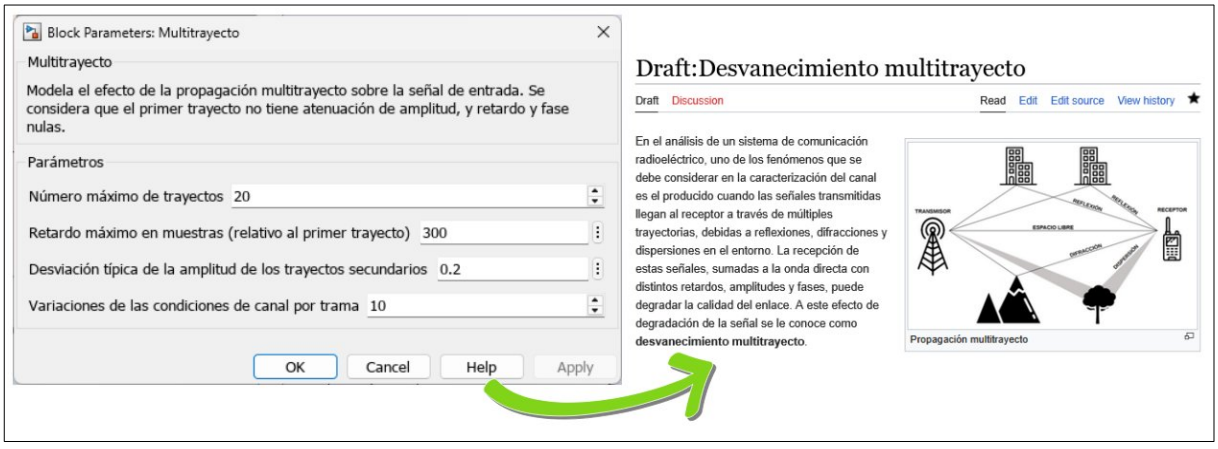

Figura 27. Cuadro de diálogo de configuración del bloque y asociación con el artículo de www,glossalab.org.

Seguidamente, modificamos el icono del bloque , mediante *Mask/Add Icon Image*, y situamos el módulo entre los bloques de modulación y canal , con las conexiones pertinentes (ver figura 28).

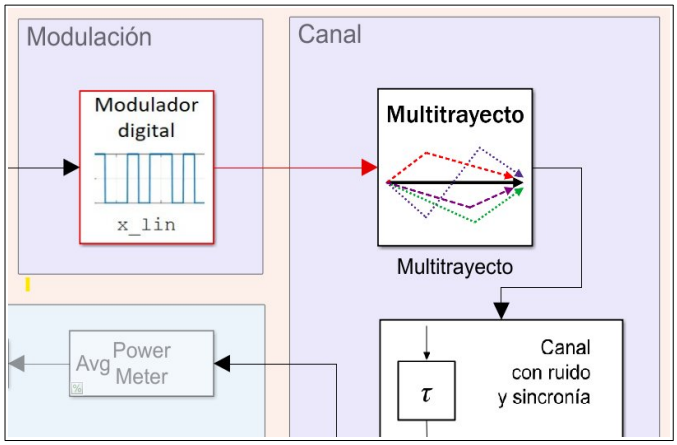

Figura 28. Integración del bloque de multitrayecto en el entorno de comLAB.

5.2.2. Bloque de decisión adaptativa

Para la construcción de este bloque, procedemos de forma similar al apartado anterior. Es decir, nos basamos en la función creada para la simulación en la consola de MATLAB, **`DecisorAdaptativo()`**. En este caso, de forma más evidente, ya que se va a invocar a la propia función desde la *S-function* que define el comportamiento del bloque **`simulink_DecisorAdaptativo.m`**,

La entrada de este bloque será la secuencia de muestras **`xm`** que nos entrega el puerto 3 del bloque de demodulación, que son las muestras en los instantes óptimos tras pasar por el filtro adaptado.

A través de la máscara, se introducen los parámetros de entrada, ya conocidos, por su uso en la simulación en MATLAB:

- **`a`**: Constelación de los valores esperados de mayor a menor, como vector columna.
- **`c`**: Códigos correspondientes a los valores de **`a`**, como vector columna.
- **`M`**: Orden del filtro FIR que actúa como ecualizador.
- **`lambda`**: Factor de olvido λ del algoritmo RLS.
- **`delta`**: Factor de inicialización δ de la inversa de la matriz de correlación **`P`**.

En este bloque tenemos cuatro salidas que corresponden con las de la función **`DecisorAdaptativo()`**:

- *Puerto 1* (**`d_cod`**): Secuencia de datos codificados de salida. Codificación de las decisiones reconstruidas según **`c`**.
- *Puerto 2* (**`d`**): Secuencia de decisiones reconstruidas en base a la constelación **`a`**.
- *Puerto 3* (**`y`**): Secuencia de salidas del ecualizador adaptativo, previas al decisor.
- *Puerto 4* (**`e`**): Vector de error entre **`d`** e **`y`** (**`e=d-y`**).

Por otra parte, el almacenamiento de información entre pasos de ejecución es ligeramente más complejo que en el caso anterior. Manejamos tres vectores *Dwork* con la siguiente información:

- *Solape para la trama posterior* (**`solape`**): Siguiendo el *método de solape y almacenamiento*, visto en el apartado 4.1.1, la salida correspondiente a cada trama dependerá de los valores finales de la trama anterior, en concreto los últimos $M-1$ valores, siendo M el orden del filtro FIR. Por lo tanto, tenemos que almacenar estos valores para garantizar la continuidad en el paso entre tramas.
- *Estado de los coeficientes del filtro FIR* (**`h`**): En cada llamada a la función **`DecisorAdaptativo()`** debemos pasarle el estado inicial del vector de coeficientes del filtro **`h`**. Excepto para la primera trama, estos coeficientes deben ser los últimos coeficientes de la trama anterior, por lo que deben ser almacenados para su uso entre tramas consecutivas.
- *Estado de la matriz P* (**`P`**): El estado inicial de la inversa de la matriz de correlación también se pasa como entrada a la función **`DecisorAdaptativo()`**, Por lo tanto, cada trama debe heredar la última matriz **`P`** de la trama anterior. El almacenamiento, en este caso, tiene una pequeña salvedad, ya que *Dwork* solo permite almacenar vectores, por lo que tenemos que convertir la matriz en vector para almacenarla mediante **`reshape(matriz,[],1)`**, y volver a reconstruirla como matriz cuando vayamos a utilizarla, mediante **`reshape(vector,[M, M])`**.

Se utilizan los mismos métodos que en el bloque de multitrayecto. En **`SetInputPortDimensions`** y **`PostPropagationSetup`** se mantienen las mismas mismas estructuras, para garantizar la coherencia entre dimensiones de puertos de entrada y salida y definir los vectores de almacenamiento *Dwork*. Con la salvedad de que aquí tenemos más puertos de salida y más vectores de almacenamiento, con dimensiones diferentes. La matriz **`P`** de dimensiones $M \times M$ se define como un vector de dimensión $M \cdot M$, por la limitación, ya comentada, de que sólo se permite el uso de vectores en *Dwork*.

En **`Start`** se inicializan los vectores *Dwork* para la primera ejecución con los siguientes valores:

- **`solape`**: Vector todo ceros de dimensión $M-1$, para establecer las condiciones iniciales para la primera trama.
- **`h`**: Vector que establece los coeficientes iniciales el filtro FIR. Tomamos un vector de dimensión M con un uno y $M-1$ ceros.
- **`P`**: Se parte de una matriz de dimensión $M \times M$, definida como $\boldsymbol{P} = (1/\delta)\,\boldsymbol{I}$, donde $\boldsymbol{I}$ es la matriz identidad. Esta matriz se redimensiona como vector con **`reshape(matriz,[],1)`**, para que sea aceptada por *Dwork*.

En el método **`Outputs`**, la implementación es bastante sencilla, ya que lo único que se hace es tomar la trama de entrada, los parámetros de la máscara y los valores heredados de *Dwork* y realizar la llamada a la función **`DecisorAdaptativo()`**. Previamente, se construye el vector de entrada a la función como concatenación del solape y la trama de entrada. Posteriormente, se asignan los valores de salida para los cuatro puertos y se actualizan los vectores *Dwork* para el siguiente paso.

El diseño de la máscara del bloque, y la asociación del botón de ayuda con el artículo sobre *Filtrado Adaptativo* desarrollado en glossaLAB (https://www.glossalab.org/wiki/Draft:Filtrado_adaptativo), siguen un proceso idéntico el utilizado en el anterior bloque. En la figura 29 mostramos el resultado.

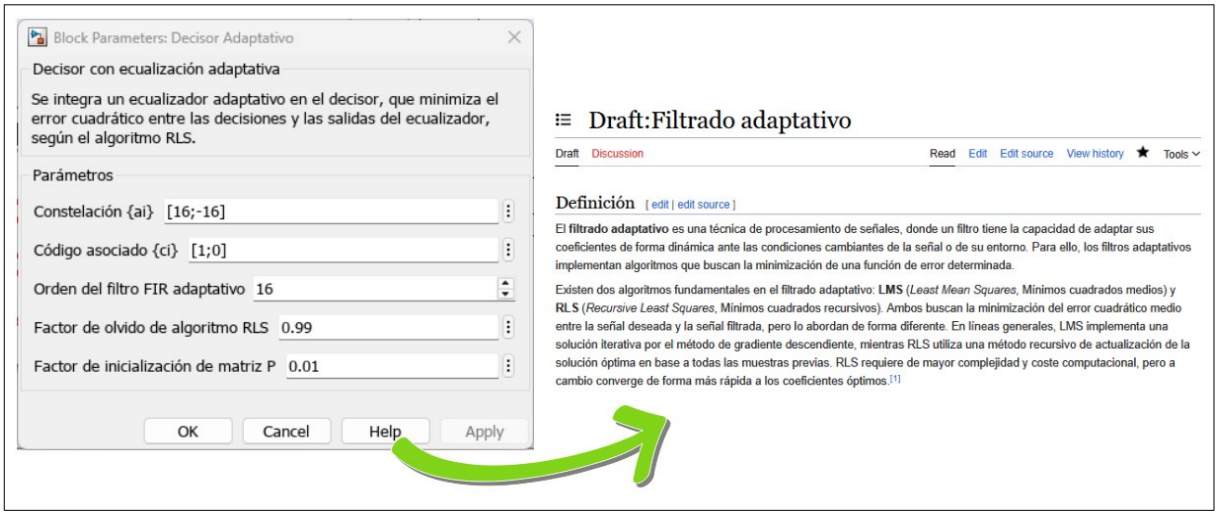

Figura 29. Cuadro de diálogo de configuración del bloque y asociación con el artículo de www,glossalab.org.

Finalmente, creamos un icono para el bloque y lo integramos y conectamos en el lienzo de comLAB (ver figura 30).

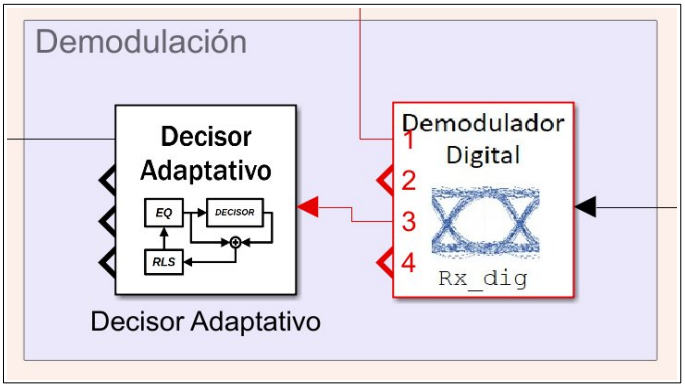

Figura 30. Integración del bloque de decisión adaptativa en el entorno de comLAB.

5.2.3. Elementos de medida y monitorización de señales

5.2.3.1. Medidores de error

comLAB dispone de bloques que permiten comparar dos secuencias de datos binarios. Estos bloques disponen de dos entradas, correspondientes a las dos secuencias, y dos salidas que van a dos bloques de tipo *display*. Estos displays muestran los errores acumulados en la comparación de secuencias durante la simulación y el BER correspondiente a estos errores (ver figura 31).

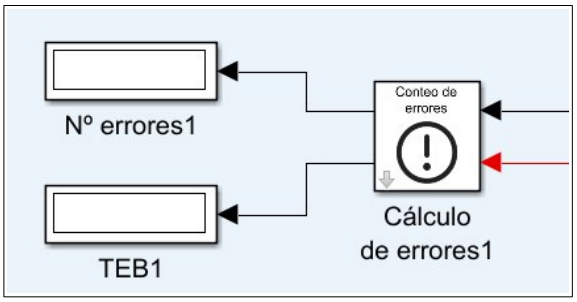

Figura 31. Medición del error entre secuencias.

Para poder realizar la comparación entre secuencias, estas deben estar temporalmente sincronizadas. Para ello, sobre la primera entrada se aplica un retardo que compensa el retardo acumulado por la segunda en el transcurso del circuito. El cálculo apropiado de este retardo es fundamental para disponer de unas medidas fiables.

Para comprobar la eficacia de nuestros decisores, en función de las condiciones del multitrayecto,

vamos a incorporar dos medidores que comparen la secuencia codificada antes de entrar al modulador, con las secuencias decodificadas a la salida del decisor simple (puerto 1 del demodulador) y a la salida del decisor adaptativo, según la figura 32.

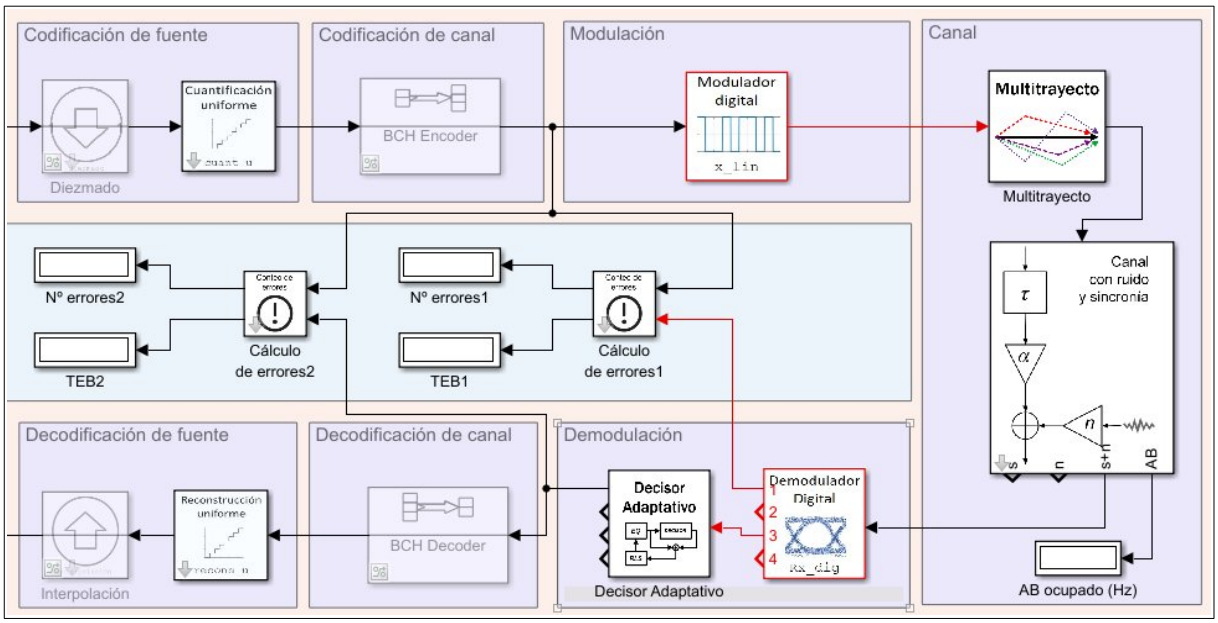

Figura 32. Medida del error a la salida de los decisores

El cálculo del retardo introducido por el circuito en cada caso va a depender de los módulos intercalados entre los puertos de medida. Estos son los retardos introducidos por cada uno de los módulos:

***Modulador y demodulador digital*:**

La pareja modulador/demodulador introduce un retardo de $N_S \cdot k$ bits siendo k el número de bits por símbolo. Por tanto, para modulación binaria consideramos un retardo de N_S bits, correspondientes a $N_S \cdot M_S$ muestras de señal de línea, siendo M_S el número de muestras por periodo de bit.

***Multitrayecto*:**

No introduce retardo. A pesar de basar su funcionamiento en un filtro, en la función **`filter()`** el resultado se alinea temporalmente con la entrada, es decir, la primera muestra de salida corresponde a la primera de entrada.

***Decisor adaptativo*:**

Basa su funcionamiento en un filtro FIR de M coeficientes, por lo tanto, se introduce un retardo de

$M-1$ bits.

***Canal con ruido y sincronía*:**

El canal esta diseñado para introducir un retardo en muestras, que cuadre la señal a nivel de bit (a la salida del decodificador de canal) y de byte (a la salida del reconstructor). De forma que el retardo absoluto a la salida del decodificador de canal se pueda expresar en un número entero de bits, y además el retardo absoluto después de la reconstrucción se pueda expresar en un número entero de bytes.

Considerando únicamente el retardo introducido por el bloque modulador/demodulador, tal como ocurre en el circuito original, si asumimos modulación binaria, el retardo añadido por el canal, expresado en muestras, debe ajustarse de manera que el retardo total sea múltiplo de la longitud de un byte: $Retardo\ canal = (b - mod\,(N_s\,,\,b))\,M_S$. Siendo b el número de bits por byte.

Si consideramos los $M-1$ bits de retardo que añade el decisor adaptativo, el canal debe añadir el siguiente retardo (en muestras):

$$Retardo\ canal = (b - mod\,(N_s + M - 1\,,\,b))\,M_S$$

Por lo tanto, en el primer medidor, que compara la secuencia binaria a la entrada del modulador con la de la salida del demodulador, debemos introducir un retardo en la primera entrada que compense el retardo del canal y el retardo de la pareja modulador/demodulador (en bits):

$$Retardo\ medidor\ 1 = N_s + b - mod\,(N_s + M - 1\,,\,b)$$

Además, para el segundo medidor debemos considerar el retardo añadido por el bloque de decisión adaptativa:

$$Retardo\ medidor\ 2 = M - 1 + N_s + b - mod\,(N_s + M - 1\,,\,b)$$

Con estas consideraciones, las secuencias entrantes en los dos medidores están alineadas temporalmente y las medidas de error reflejan las diferencias entre ambas. Para facilitar el cálculo de estos retardos en el circuito, las máscaras integran estas fórmulas y calculan el retardo a partir de los parámetros introducidos en ellas. Para considerar también el parámetro M, lo añadimos a las máscaras de los medidores (ver figura 33).

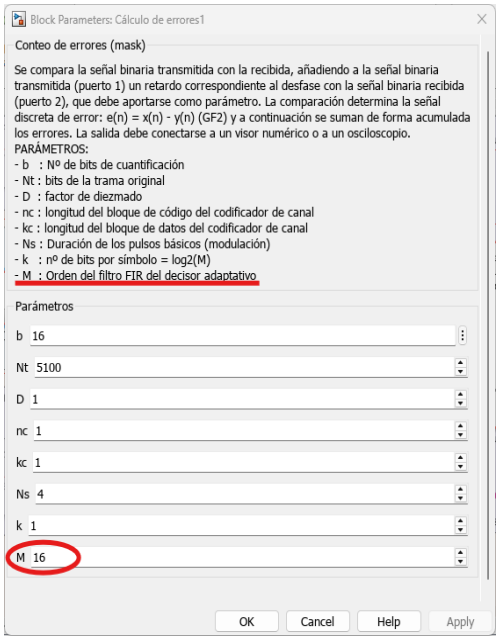

Figura 33. Consideración del orden del filtro adaptativo en la máscara del medidor

5.2.3.2. Representación de señales

comLAB dispone de bloques de representación de señales, que nos van a permitir analizar su comportamiento en los dominios temporal y espectral en diferentes puntos del circuito.

Para observar el efecto del canal en el dominio del tiempo, incorporamos un *osciloscopio* con tres entradas, correspondientes a la salida del modulador, la salida del bloque de multitrayecto y la salida del canal ruidoso (ver figura 34).

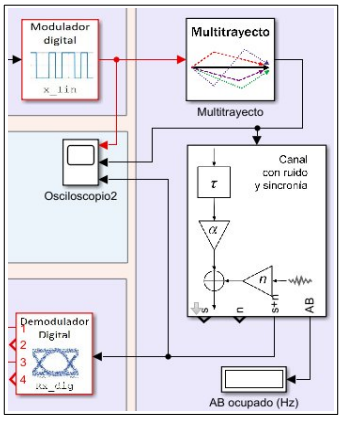

Figura 34. Osciloscopio para representar el efecto del canal.

En la figura 35 se muestra un ejemplo de representación, considerando una modulación con pulsos RCCA de nivel de entrelazado 4, un bloque de multitrayecto con 300 muestras de retardo máximo y $\sigma = 0.4$, y un canal ruidoso con $N_0 = 10^{-8}\ W/Hz$ y atenuación de 6 dB. Nótese que la señal a la salida del canal ruidoso está desplazada temporalmente respecto a las otras dos, debido al retardo introducido por el bloque de canal para sincronizar a nivel de bit y byte.

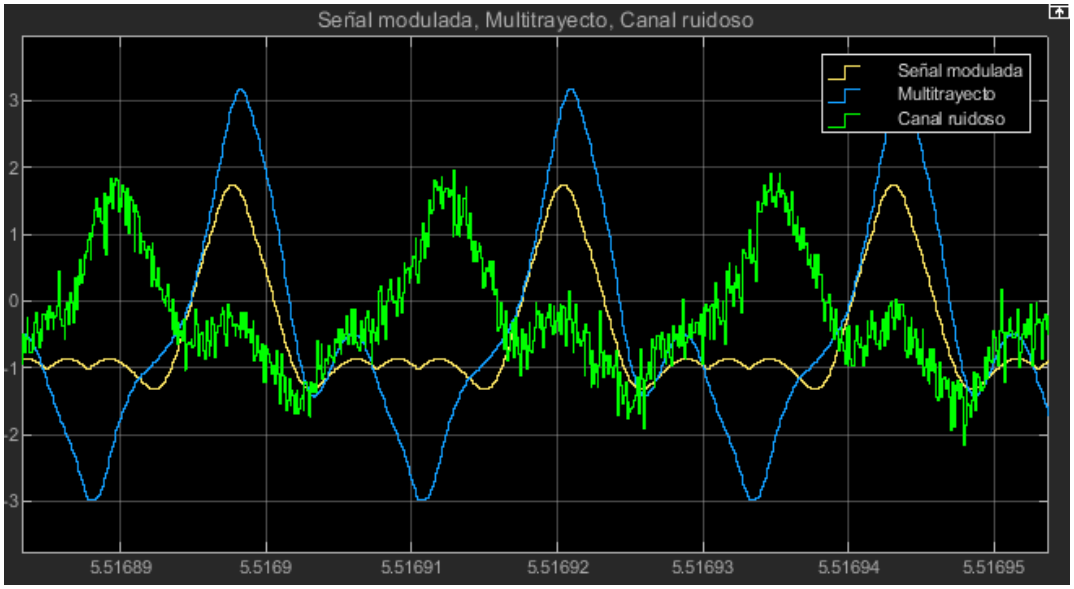

Figura 35. Representación en osciloscopio de señal modulada, señal tras multitrayecto y señal a la salida del canal.

En el dominio de la frecuencia, es de interés ver cómo afecta el bloque de multitrayecto a la densidad espectral de potencia de la señal. Para ello, se insertan dos bloques de *analizador de espectros* antes y después del módulo de multitrayecto, con el fin de comparar los resultados (ver figura 36).

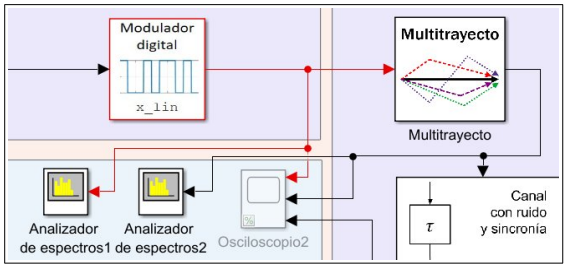

Figura 36. Analizador de espectros antes y después del bloque de multitrayecto.

En la figura 37 se representan las dos D.E.P. Podemos observar cómo, en este caso, se acentúan las frecuencias más bajas de la señal cuando se atraviesa el bloque de multitrayecto. En este ejemplo,

partimos de una señal con codificación de línea NRZ y pulsos RCCA con factor de redondeo 0. El bloque de multitrayecto está configurado con un retardo máximo de 300 muestras, un máximo de 10 trayectos y $\sigma = 0.4$.

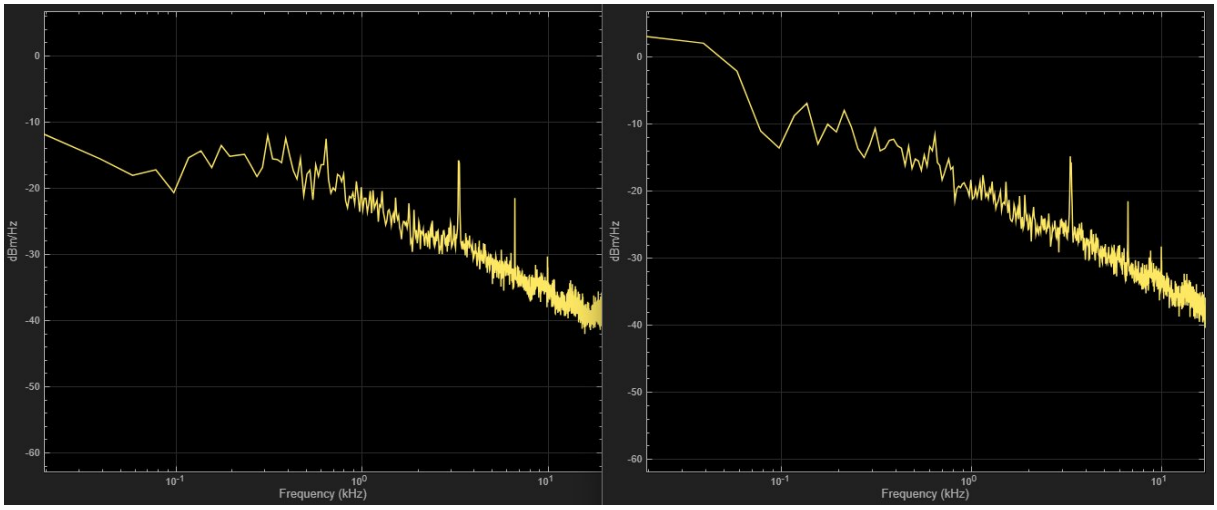

Figura 37. D.E.P. de la señal a la salida del modulador (dcha.) y a salida del bloque multitrayecto (izq.).

Como último elemento de representación, incorporamos un osciloscopio para mostrar la evolución del error cuadrático entre las decisiones y las salidas del filtro adaptativo. Para ello, conectamos la salida del error que nos entrega el decisor adaptativo a un bloque que eleva al cuadrado y posteriormente al osciloscopio (ver figura 38). Esto nos ayudará a apreciar, de forma gráfica, la convergencia del ecualizador adaptativo.

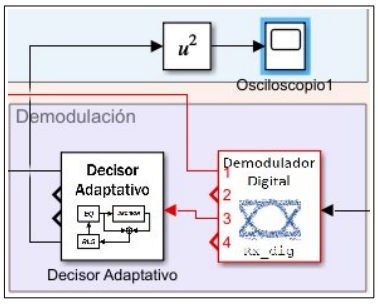

Figura 38. Representación del error cuadrático entre decisiones y salidas del filtro adaptativo.

CAPÍTULO VI. RESULTADOS

6.1. Efecto del filtro de suavizado en la función multitrayecto

En esta sección analizamos la utilidad de aplicar el filtro de suavizado de la sección 5.1.2.1 en la función **`multipath()`**. Su uso, en combinación con un número elevado de variaciones del canal, va a permitir que la simulación reproduzca de manera más realista el comportamiento de la propagación multitrayecto. Previamente, es conveniente revisar cómo se generan las respuestas al impulso del canal, ya que dependen de la generación aleatoria de números.

En MATLAB, la generación de números aleatorios funciona mediante algoritmos pseudoaleatorios, lo que significa que, aunque los números parecen aleatorios, en realidad siguen una secuencia determinista calculada a partir de un valor inicial llamado *seed* (semilla). Por defecto, MATLAB utiliza el algoritmo *Mersenne Twister* y la semilla 0. La función **`rng(seed)`** nos permite fijar la semilla. Cuando se ejecuta, se reinicia el estado interno del generador, lo que garantiza que, en la siguiente llamada a funciones como **`rand()`**, **`randn()`** o **`randi()`**, se obtendrá la misma secuencia de números. Sin embargo, es importante destacar que si se generan más números aleatorios después de la primera llamada, la secuencia continuará de manera predecible pero no se repetirá automáticamente. Es decir, si se ejecuta **`rng(123)`** y luego **`rand(1)`** dos veces, la primera llamada dará un valor fijo (por ejemplo, 0.6965), y la segunda un valor distinto pero también determinado (por ejemplo, 0.2861). La repetición exacta solo ocurre si se vuelve a resetear la semilla con **`rng(seed)`** antes de cada generación. Si no se reinicia la semilla, cada nueva llamada a funciones aleatorias seguirá avanzando en la secuencia predefinida, produciendo resultados distintos (MathWorks, s.f.i).

Por tanto, si fijamos la semilla al inicio de cada simulación, nos va a permitir repetir las mismas condiciones aleatorias en simulaciones distintas. De este modo, será posible comparar el efecto de la función multitrayecto sin y con filtro de suavizado.

Para comprobar el efecto del filtro de suavizado, guardamos una versión de **`multipath()`** con el código de filtrado no comentado y la llamamos **`multipath_fil()`**. Aplicamos las dos versiones sobre una señal de línea **`x`**, fijando la semilla antes de la ejecución de cada una. De este modo, los valores aleatorios de partida **`tau`**, **`theta`** y **`r`** serán los mismos en las dos ejecuciones. La diferencia estará en las matrices **`B`** de respuestas al impulso construidas. A continuación, se muestran los parámetros utilizados y la ejecución de las funciones. Nótese el valor más elevado de **`Nt`**, respecto a la simulación

de la sección 5.1.6, para superar el efecto transitorio del filtro:

```
Pmax=15; % Número máximo de trayectos
Tmax=300; % Retardo máximo (en muestras)
sigma=0.3; % Desviación típica de la amplitud de los pulsos secundarios
Nt=300; % Número de subtramas (variaciones del canal)
rng(456);
[~,B]=multipath(x,Pmax,Tmax,sigma,Nt); % SIN suavizado
rng(456);
[~,B_fil]=multipath_fil(x,Pmax,Tmax,sigma,Nt); % CON_suavizado
```

Si comparamos las matrices **B** obtenidas, podemos apreciar el efecto del suavizado. A simple vista, se observa el impacto del filtro en la figura 39, donde se comparan cuatro respuestas al impulso consecutivas, correspondientes a las subtramas del final del proceso. Se puede advertir que las respuestas filtradas (a la derecha) tienen menos fluctuación en la amplitud y en el retardo, que las no filtradas (a la izquierda).

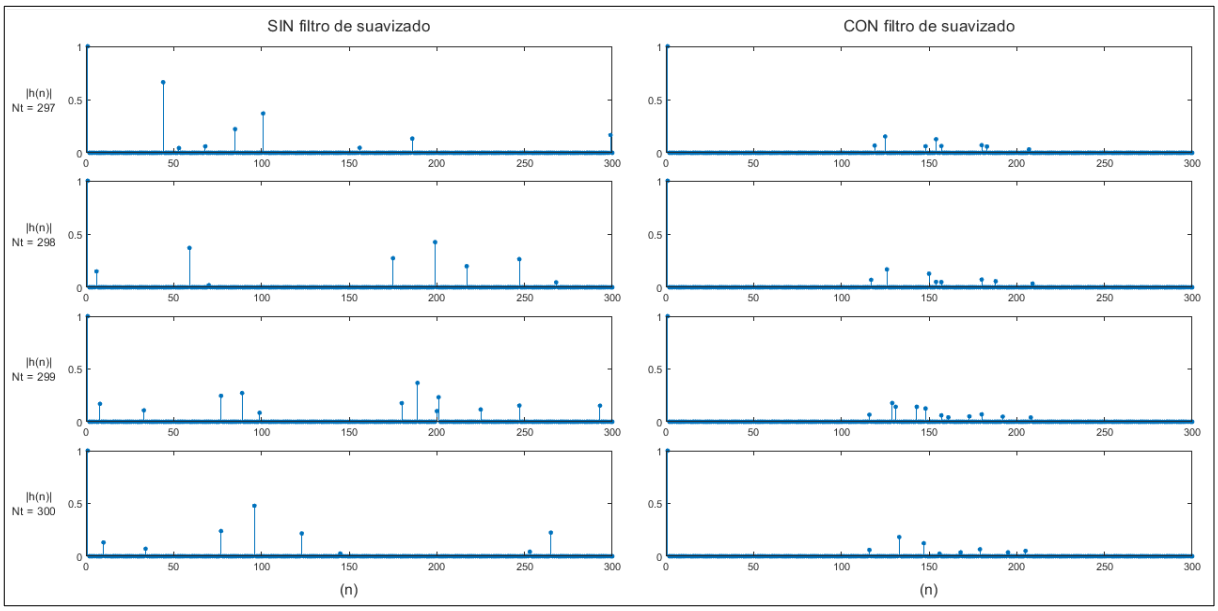

Figura 39. Respuestas al impulso consecutivas SIN y CON filtro de suavizado.

Como medida de la variabilidad entre respuestas consecutivas, podemos utilizar el coeficiente de correlación mediante la función de MATLAB **`corr()`**. Aplicamos esta función a las filas consecutivas, de dos en dos, de las matrices **B** obtenidas, con transposición previa, ya que para calcular el coeficiente de correlación entre vectores con **`corr()`** deben estar dispuestos como columnas. Con esto, obtenemos la gráfica de la figura 40, donde apreciamos que las respuestas consecutivas después del suavizado

tienen un grado de correlación sensiblemente superior que las no suavizadas, por estar sus coeficientes de correlación más próximos a 1.

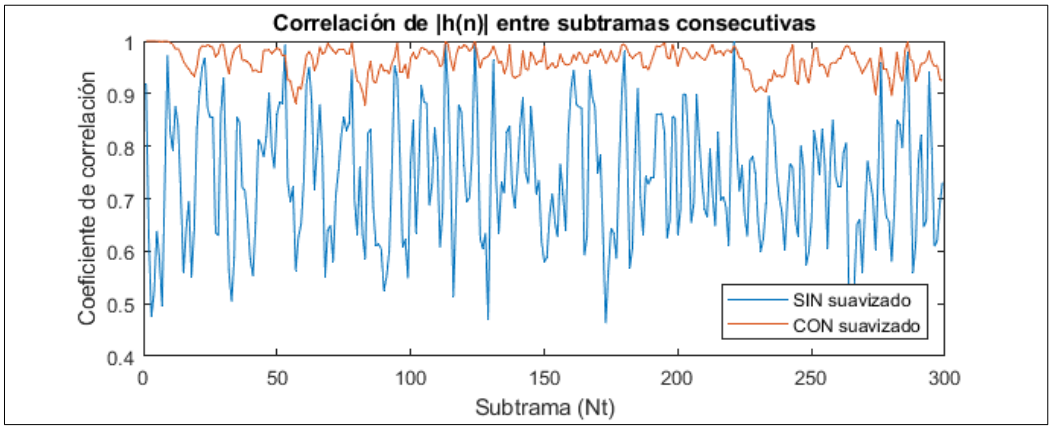

Figura 40. Coeficiente de correlación de respuestas al impulso consecutivas.

Este comportamiento se acerca más a la realidad, y será el modelo utilizado en **`multipath()`** en las próximas simulaciones, cuando el valor de **`Nt`** lo permita.

El *script* de MATLAB utilizado en esta simulación y en las representaciones gráficas está disponible en el anexo II como **`Efecto_filtro_suavizado.m`**.

Si continuamos las simulaciones hasta el final del ciclo con estas condiciones de propagación multitrayecto, como era predecible, los resultados de tasa de error de bit son mejores para el multitrayecto con suavizado.

Consideramos los siguientes parámetros en el resto de la simulación:

```
No=1e-7; % DEP de ruido
M=16; % Orden del filtro adaptativo
lambda=0.99; % Factor de olvido del algoritmo RLS
delta=0.01; % Para inicializar la matriz P
```

Partiendo de 500000 muestras de la señal de origen, modulada con pulsos rectangulares NRZ de 16 muestras por periodo, obtenemos los siguientes resultados:

- Sin filtro de suavizado:
  - BER con decisor no adaptativo: **`BER_SOLO_DECISOR = 0.0159`**.
  - BER con decisor adaptativo: **`BER_DECIS_ADAP = 0.0052`**.

- Con filtro de suavizado:
  - BER con decisor no adaptativo: **`BER_SOLO_DECISOR = 9.7e-04`**.
  - BER con decisor adaptativo: **`BER_DECIS_ADAP = 3.0e-05`**.

6.2. Resultados en función del retardo máximo

El retardo máximo T_m, entendido como el tiempo transcurrido entre la recepción de la primera versión de la señal y la última, es un parámetro fundamental en la caracterización de la propagación multitrayecto, como ya señalamos en el apartado 2.2. Su valor define la extensión temporal de la respuesta al impulso y es determinante en la interferencia entre símbolos. En este apartado vamos a mostrar como influye el retardo máximo en la tasa de error de bit, según nuestro modelo de simulación.

Para ello, vamos a variar el parámetro **`Tmax`** en distintas simulaciones, manteniendo el resto de parámetros constantes, y observando su influencia en la tasa de error de bit, tanto para el decisor adaptativo como para el decisor simple. Hay que señalar que **`Tmax`** no es exactamente el retardo máximo, sino el valor máximo de la variable aleatoria uniformemente distribuida que representa el retardo, por lo tanto, **`Tmax`** no coincide con T_m, pero la repetición de la simulación con distintas semillas generadoras de aleatoriedad y el cálculo de la media nos dará una noción de la variación de la tasa de error en función del retardo máximo.

Realizamos un bucle de simulaciones con **`Tmax`** variable, y la misma semilla en cada simulación, con los siguientes parámetros:

```
inicio=1; % Primera muestra
fin=500000; % Última muestra
Ms=32; % Muestras por símbolo
tipo=1; % Tipo de codificación de línea NRZ
Ns=1; % Elegimos pulsos rectangulares
Pmax=10; % Número máximo de trayectos
sigma=0.3; % Desviación típica de la amplitud de los pulsos secundarios
Nt=500; % Número de subtramas (variaciones del canal)
No=1e-7; % DEP de ruido
M=16; % Orden del filtro adaptativo
lambda=0.99; % Factor de olvido del algoritmo RLS
delta=0.01; % Para inicializar la matriz P
```

`Tmax` tomará valores desde `8` hasta `512` en saltos de `8` muestras, de forma que, siendo T el periodo de símbolo (`Ms` muestras), la simulación cubra desde 0.25 T hasta 16 T.

Repetimos este bucle una veintena de veces, cambiando cada vez la semilla generadora, y calculamos la media de los valores obtenidos, obteniendo los resultados de la figura 41.

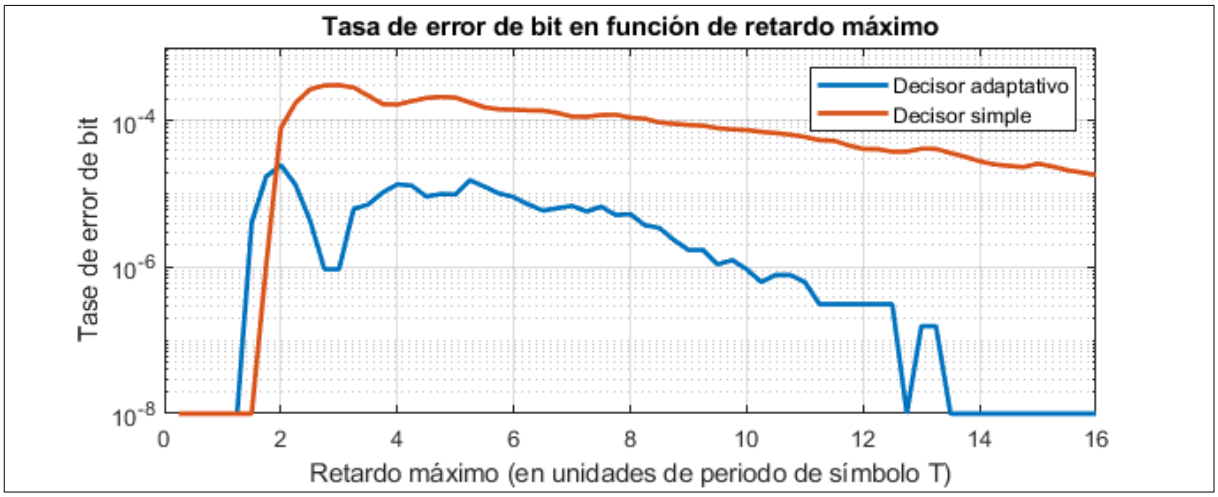

Figura 41. BER en función de `Tmax` *para decisor adaptativo y decisor simple.*

Se muestra el BER en escala logarítmica, poniendo los valores nulos a 10^{-8}, para facilitar la representación.

Se observa que para retardos pequeños, por debajo de T, ambos decisores aciertan siempre. Esta es la zona de *desvanecimiento plano*, que en el apartado 2.2 definíamos como el caso donde el *ancho de banda de coherencia* del canal es mayor que el ancho de banda de la señal, por lo que todas sus componentes espectrales son tratadas por igual.

A partir de 1.5 T aumenta el error en los dos decisores, siendo destacable que en el periodo entre 1.25 T y 1.5 T el decisor simple es más eficiente que el adaptativo. Cuando T_m es mayor que T entramos en la zona de *desvanecimiento selectivo en frecuencia*, que como veíamos en 2.2, se produce al ser el ancho de banda de coherencia del canal es más estrecho que el de la señal, por lo que cada componente espectral de la señal se ve afectada de forma distinta por el canal, incrementando la interferencia entre símbolos.

El decisor simple alcanza su máximo error para 3 T, mientras que el adaptativo lo hace en 2 T. A partir de esos momentos, ambos descienden su tasa de error, siendo la caída en el decisor simple menos pronunciada. Este descenso se puede explicar porque a pesar de que los retardos son más largos, con el perjuicio que esto conlleva, el número de trayectos sigue siendo el mismo, lo que hace que las

contribuciones secundarias estén más espaciadas. Por tanto, la concentración de trayectos en $3\,T$ es más perjudicial que su espaciado en $16\,T$.

La caída del error más pronunciada en el decisor adaptativo, llegando hasta cero, se puede explicar por la naturaleza del propio ecualizador, que no deja de ser un filtro FIR de M coeficientes. Cuando los retardos de los trayectos están espaciados, las réplicas de la señal llegan con suficiente separación temporal, lo que hace que su energía se concentre en posiciones del retardo que coinciden mejor con los coeficientes del FIR. De esta forma, el ecualizador puede ajustar los pesos con mayor precisión para eliminar la ISI. Cuando los retardos están muy próximos, las energías se solapan, dificultando la convergencia y la cancelación del efecto del canal.

La siguiente simulación está relacionada, precisamente, con el orden del filtro adaptativo. Tomamos uno de los bucles de simulación anteriores, y lo volvemos a repetir con las mismas condiciones, excepto por el orden del filtro FIR, que cambiamos de 16 a 8. En la figura 42 se muestran los resultados de los dos casos. Se observa que cuando los retardos de los trayectos superan el rango temporal que cubre el orden del filtro FIR, este deja de ser capaz de compensarlos. Los ecos con retardos mayores quedan fuera del alcance del filtro y actúan como interferencia sobre símbolos posteriores sin posibilidad de cancelación. Esto se traduce en el aumento de la tasa de error. Para compensar estos retardos, sería necesario aumentar el orden del filtro.

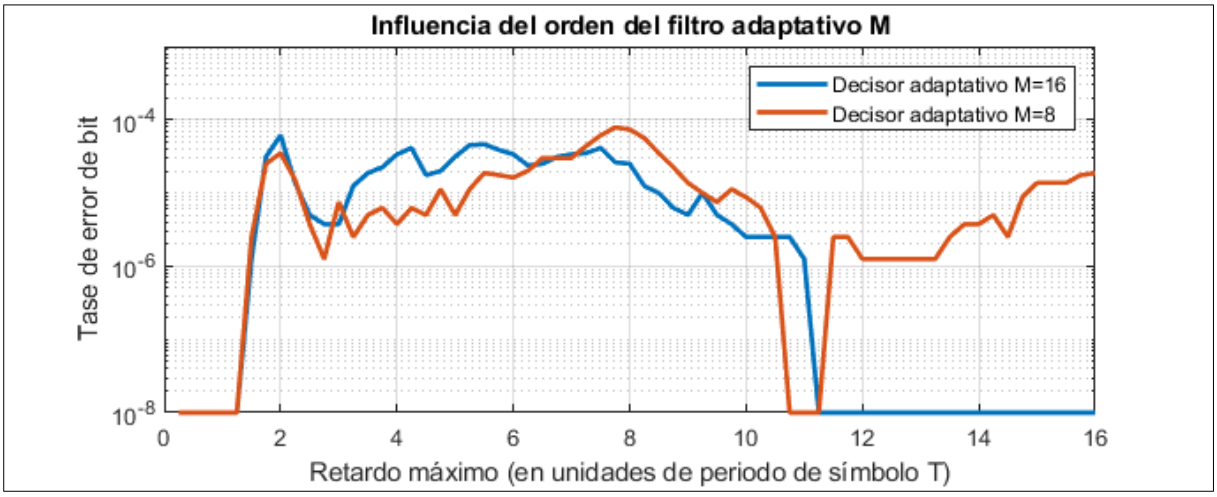

Figura 42. Evolución del BER en función de `Tmax` *para filtros FIR de distinto orden.*

6.3. Resultados en función de la modulación utilizada

Para esta prueba, vamos a realizar una simulación con unos parámetros determinados, que repetiremos bajo las mismas condiciones de aleatoriedad, pero cambiando cada vez el tipo de modulación (pulsos rectangulares o RCCA, con distintos factores de redondeo) y el código de línea (NRZ, RZ o Manchester). Los resultados no pretenden generalizar la influencia de la modulación en la tasa de error de bit, ya que se va simular en unas condiciones determinadas. Simplemente, tratamos de mostrar los distintos resultados para casos particulares, que pueden indicar una tendencia.

Para ello, creamos un ciclo de simulación con los siguientes parámetros fijos:

```
s=1; % Potencia 1 W
Rb=1e6; % Tasa binaria 1 Mpbs
signal=1; % Señal elegida Ryc.wav
inicio=1; % Primera muestra
fin=500000; % Última muestra
Ms=32; % Muestras por símbolo
Pmax=10; % Número máximo de trayectos
Tmax=300; % Retardo máximo (en muestras)
sigma=0.35; % Desviación típica de la amplitud de los pulsos secundarios
Nt=500; % Número de subtramas (variaciones del canal)
No=1e-7; % DEP de ruido
M=16; % Orden del filtro adaptativo
lambda=0.99; % Factor de olvido del algoritmo RLS
delta=0.01; % Para inicializar la matriz P
```

En cada simulación tomaremos distintos tipos de modulación y codificación de línea, mediante la variación de los parámetros **`tipo`**, **`Ns`** y **`r`**. Para que los resultados sean más representativos, repetiremos la simulación con los mismos parámetros una veintena de veces, cambiando cada vez la semilla generadora, y calculando al final el BER medio de las repeticiones realizadas. Para ello, utilizaremos un bucle del tipo:

```
for i=1:20
        rng(i); % Para fijar semilla
        {Simulación con parámetros tipo, Ns y r determinados}
        {Almacenar resultado en vector BER con índice i}
end
{Calcular media de vector BER}
```

El código utilizado se puede encontrar en el anexo II como **`Efecto_modulacion.m`**.

Posteriormente, repetimos las simulaciones con otros valores de **`sigma`** y **`Tmax`**, para ver si varían los resultados en otras condiciones.

Los resultados obtenidos se muestran en la tabla 1.

TASA DE ERROR DE BIT		**`sigma=0.35 Tmax=300`**		**`sigma=0.4 Tmax=300`**		**`sigma=0.4 Tmax=200`**	
Modulación	**Código de línea**	*Decisor adaptativo*	*Sólo decisor*	*Decisor adaptativo*	*Sólo decisor*	*Decisor adaptativo*	*Sólo decisor*
Rectangular	**NRZ**	1,49E-05	6,57E-04	9,71E-05	2,16E-03	9,11E-05	3,33E-03
	RZ	0,00E+00	0,00E+00	3,75E-07	2,25E-06	3,00E-06	1,38E-06
	Manchester	0,00E+00	0,00E+00	0,00E+00	1,25E-07	0,00E+00	2,50E-07
RCCA `Ns=4` `r=0`	**NRZ**	1,46E-04	9,57E-03	4,83E-04	1,52E-02	7,61E-04	1,86E-02
	RZ	3,75E-07	9,20E-05	3,38E-06	2,86E-04	2,79E-04	5,73E-03
	Manchester	6,50E-06	6,71E-04	1,81E-05	1,36E-03	1,36E-05	1,71E-03
RCCA `Ns=4` `r=0.5`	**NRZ**	3,91E-05	1,28E-03	1,96E-04	3,58E-03	2,79E-04	5,73E-03
	RZ	2,50E-07	6,25E-07	6,25E-07	2,55E-05	5,63E-06	2,99E-05
	Manchester	5,00E-07	3,75E-07	2,50E-06	1,85E-05	2,00E-06	1,30E-05
RCCA `Ns=4` `r=1`	**NRZ**	1,83E-05	7,54E-04	1,25E-04	2,48E-03	1,33E-04	3,76E-03
	RZ	1,25E-07	3,75E-07	3,75E-07	1,76E-05	4,88E-06	2,58E-05
	Manchester	0,00E+00	1,25E-07	7,50E-07	4,63E-06	5,00E-07	4,13E-06
RCCA `Ns=8` `r=1`	**NRZ**	2,20E-05	7,38E-04	1,14E-04	2,48E-03	1,23E-04	3,66E-03
	RZ	3,75E-07	1,25E-07	5,00E-07	1,71E-05	5,13E-06	2,63E-05
	Manchester	1,25E-07	1,25E-07	1,00E-06	5,13E-06	7,50E-07	3,75E-06

Tabla 1. BER obtenido en distintas simulaciones, en función de la modulación.

En cada columna se aprecia una tendencia muy similar en los resultados. Para poder apreciarla mejor,

representamos una de ellas en formato de diagrama de barras (ver figura 43).

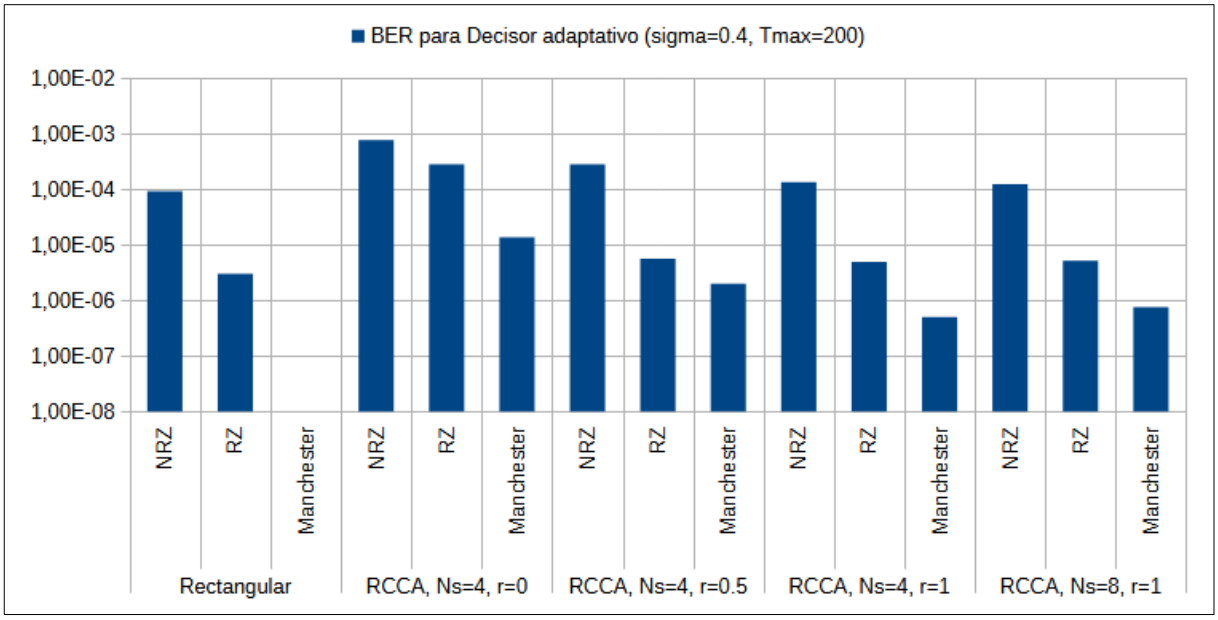

Figura 43. Tasa de error de bit en función de la modulación para **`sigma=0.4`**, **`Tmax=200`**.

En cuanto a la codificación de línea, se observa que Manchester es la que obtiene mejores resultados, seguida de cerca por RZ. En contraposición, NRZ es, sin duda, la que peor se comporta ante la distorsión provocada por el multitrayecto, con tasas de error de bit de órdenes superiores a las dos anteriores.

En cuanto a la modulación, la rectangular obtiene mejores resultados que cualquiera de las de raíz cuadrada de coseno alzado. Entre las RCCA, la tasa de error es mejor cuando el factor de redondeo es 1, y peor cuando se acerca a 0. No se aprecia diferencia significativa cuando se aumenta el nivel de entrelazado (**`Ns`**), por lo que este factor no parece ser relevante en este caso.

La explicación a estos resultados se puede hacer desde dos puntos de vista: el espectral y el temporal.

El desvanecimiento multitrayecto es selectivo en frecuencia cuando el retardo máximo supera al periodo de símbolo, como vimos en el apartado 2.2. Estos desvanecimientos en frecuencia tienen un impacto más negativo sobre las señales con menor ancho de banda, ya que su energía está más concentrada en el dominio espectral. Como consecuencia, cualquier atenuación localizada en frecuencia perjudica una proporción mayor de su contenido útil. Esto explica que las codificaciones Manchester y RZ presenten mejor rendimiento que NRZ**.** Al tener el doble de ancho de banda, su energía está más dispersa, lo que las hace menos vulnerables a las caídas selectivas en frecuencia. Por el mismo motivo, se justifica que la modulación con pulsos rectangulares sea menos sensible al

multitrayecto que la modulación con pulsos RCCA, y que se obtengan mejores resultados cuando el *roll-off* es mayor, ya que esto implica mayor ancho de banda y mayor robustez ante los desvanecimientos selectivos de frecuencia.

En el domino temporal, las modulaciones más afectadas por la interferencia entre símbolos, producida por la dispersión temporal del multitrayecto, son las que tienen pulsos más anchos. La extensión temporal de la energía del pulso incrementa el solapamiento entre símbolos cuando existen múltiples trayectos con retardos diferentes, agravando el efecto de la ISI. Por este motivo, la modulación rectangular y las codificaciones RZ y Manchester obtienen las mejores tasas de error.

Hay que señalar que estos resultados se han obtenido suponiendo un canal sin limitación de ancho de banda. En un escenario real, con un canal limitado espectralmente, los resultados y su justificación estarían condicionados de forma determinante por esta circunstancia.

6.4. Simulaciones en comLAB

Se parte del circuito de transmisión digital de comLAB, con la incorporación de los nuevos módulos de multitrayecto y decisión adaptativa, así como los elementos de medida y representación que se mostró en la sección 5.2.3 (ver figura 44).

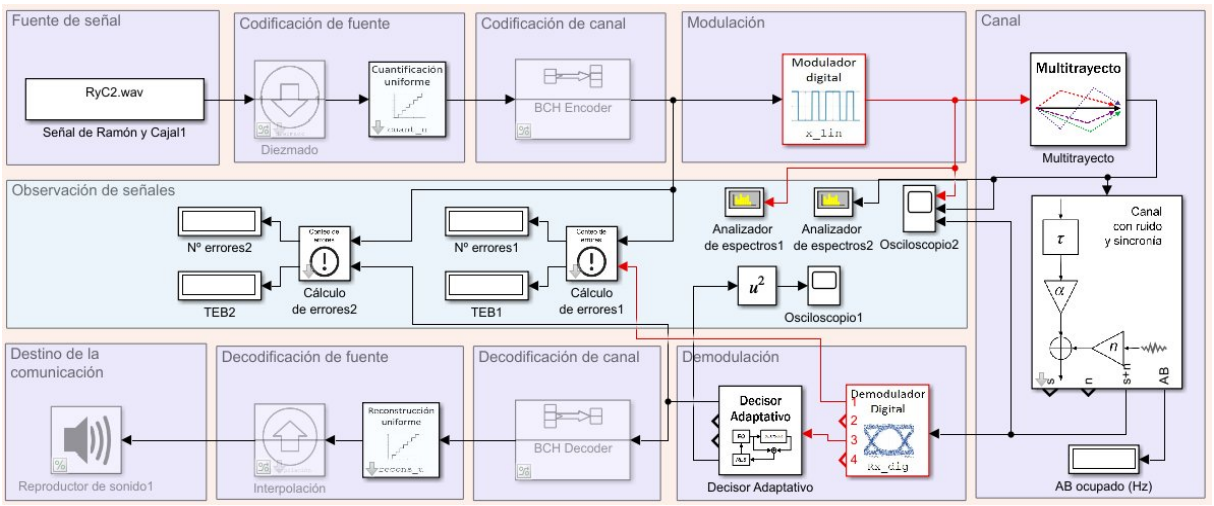

Figura 44. Cadena de transmisión con los nuevos módulos y los elementos de medida y representación.

En primer lugar, para validar la eficacia del decisor adaptativo como elemento de decisión , se procede a comprobar su funcionamiento sin el efecto del bloque de multitrayecto y reduciendo la densidad espectral de potencia de ruido a un valor despreciable. En tales circunstancias, sin distorsión ni ruido,

la tasa de error de bit debería ser nula.

Lamentablemente, los resultados de la simulación en dichas condiciones no son los esperados. Mientras que la tasa de error a la salida del decisor no adaptativo es del orden de 10^{-7}, en el decisor adaptativo no se consiguen valores inferiores a 10^{-3}.

Para entender la posible causa de estos resultados hay que recordar como se realiza el sincronismo en comLAB. El receptor no dispone de mecanismos de sincronización por si mismo. Es el canal el que introduce el retardo oportuno en muestras para lograr el sincronismo a nivel de byte y bit (ver sección 5.2.3.1).

El retardo introducido por el canal son muestras de valor nulo que se añaden al principio de la secuencia. Estas muestras nulas, sumadas al bajo nivel de ruido del canal, producen unos valores que oscilan ligeramente alrededor de cero. Al pasar esta secuencia por el filtro adaptado y muestrear el resultado en los instantes óptimos, se obtiene, al principio, una serie de muestras transitorias cercanas a cero. Estas muestras limitadas, que corresponden al retardo y no a la señal, son susceptibles de ser malinterpretadas por el decisor no adaptativo, y producir errores. Pero los errores no se extenderán más allá del retardo introducido. Por ello, con el decisor no adaptativo se obtiene una tasa casi despreciable del orden de 10^{-7}, correspondiente a los errores transitorios del principio.

Sin embargo, el efecto de estas muestras transitorias es más perjudicial para el decisor adaptativo. En este caso, las muestras próximas a cero, alejadas de los valores esperados, son utilizadas por el algoritmo adaptativo (RLS) para actualizar los coeficientes del ecualizador. Dado que estas muestras no contienen información válida de la señal transmitida, el algoritmo interpreta valores esencialmente aleatorios y ajusta los coeficientes en direcciones erróneas. Esto provoca que el ecualizador se desvíe de la solución óptima antes de que comience a llegar la señal útil.

Cuando finalmente se reciben las primeras muestras válidas de la señal, el ecualizador parte de un estado distorsionado debido a la fase de adaptación sobre datos no útiles. Este error inicial puede reforzarse, ya que las decisiones tomadas sobre las primeras muestras válidas están influenciadas por un ecualizador mal ajustado, lo que genera errores de decisión que retroalimentan el algoritmo y dificultan la convergencia.

Este fenómeno explica por qué la tasa de error del decisor adaptativo no se reduce hasta valores despreciables en condiciones ideales. Aun sin ruido ni distorsión del canal, la propia dinámica de adaptación sobre muestras transitorias degrada la tasa de error de bit.

Es evidente que esta circunstancia, producida por la introducción de retardo en el canal para lograr el sincronismo, también afecta de la misma forma cuando se activa el bloque de multitrayecto. El efecto transitorio puede dificultar la convergencia del algoritmo RLS, impidiendo el correcto funcionamiento del bloque de decisión adaptativa en el entrono de comLAB.

Para la incorporación funcional en el entrono de comLAB del bloque de decisión adaptativa, cuya efectividad se ha comprobado por separado en las simulaciones en MATLAB, se hace necesario incorporar técnicas de sincronismo en el receptor, o incorporar mecanismos de adaptación a la transitoriedad en el decisor adaptativo. Estas mejoras se contemplan como posibles desarrollos en la sección de líneas futuras.

CAPÍTULO VII. CONCLUSIONES Y TRABAJO FUTURO

7.1. Limitaciones del estudio

Antes de extraer conclusiones a partir de las simulaciones realizadas, cabe señalar las limitaciones de este estudio. El objetivo principal de este proyecto ha sido el desarrollo de un escenario de simulación en MATLAB/Simulink que permitiera observar el efecto de la propagación multitrayecto y su minimización mediante la ecualización adaptativa. El trabajo no tiene como objetivo obtener resultados concluyentes sobre los fenómenos simulados, sino proporcionar una plataforma que facilite la experimentación y el aprendizaje.

Los resultados obtenidos nos pueden servir como una referencia para detectar tendencias, que pueden confirmar lo esperado según el marco teórico, pero no deben interpretarse como conclusiones definitivas, debido a las simplificaciones adoptadas en el modelo.

Un estudio detallado de la propagación multitrayecto requeriría un modelo de mayor complejidad, que distinguiera entre entornos *indoor* y *outdoor*, o entre zonas rurales y urbanas, aplicando distribuciones probabilísticas acordes a cada circunstancia. También se deberían considerar efectos relacionados con el velocidad de desplazamiento del receptor, como el *efecto Doppler* o el desvanecimiento rápido (*fast fading*). Nuestro modelo matemático engloba la variabilidad del canal de forma general, mediante una respuesta al impulso condicionada a variables aleatorias. Para obtener mayor fidelidad respecto al comportamiento real, las distribuciones de las variables consideradas deberían ajustarse a las circunstancias de la propagación en cada caso. Un análisis más exhaustivo se puede encontrar en el capítulo 13 del libro referenciado de Proakis y Salehi (2008).

A su vez, en la ecualización adaptativa sólo se ha implementado un filtro FIR de coeficientes variables según el algoritmo de Kalman (RLS), por lo que no podemos generalizar los resultados obtenidos a otras técnicas de ecualización adaptativa.

7.2. Conclusiones

El número de posibles condiciones de simulación, variando los parámetros implicados, es muy amplio. Por ello, las resultados que se presentan deben entenderse como ejemplos significativos de los comportamientos observados, y no como un análisis de todos los escenarios posibles. A continuación,

se enumeran algunas de las conclusiones que pueden extraerse de las simulaciones realizadas. Algunas de ellas ya se adelantaron en el capítulo VI.

Eficacia del decisor adaptativo respecto al decisor simple

La mejora en la tasa de error de bit al usar el decisor adaptativo está directamente relacionada con el retardo máximo de los trayectos secundarios respecto al principal T_m, y con el orden del filtro FIR adaptativo M, considerando que el número máximo de trayectos no varía.

Siendo T el periodo de símbolo, para un retardo máximo inferior a $2\,T$, el decisor adaptativo no muestra mejora aparente respecto al decisor simple. A partir de $2\,T$, la mejora aumenta progresivamente, alcanzando su máximo para $M{\cdot}T$.

Esto se puede explicar por dos causas:

- *Separación temporal de los ecos*: Cuando el retardo de los trayectos secundarios es suficientemente grande en relación a T , el ecualizador puede identificarlos y compensar el efecto de interferencia entre símbolos. En cambio, si los retardos son muy pequeños, el decisor adaptativo los percibe como un único trayecto con distorsión. En este caso, el decisor simple es igual de efectivo.

- *Orden del filtro*: Cuando el retardo se va acercando al límite establecido por el orden del filtro, el ecualizador es más efectivo, ya que utiliza más coeficientes para compensar el efecto del multitrayecto. Nótese que estamos considerando que el número máximo de trayectos es el mismo.

Cuando el retardo supera $M{\cdot}T$, hay ecos que quedan fuera del efecto del filtro, por lo que no se corrigen. Esto hace que el decisor adaptativo pierda parte de su eficacia y vuelva a aumentar la tasa de error.

Tasa de error en función del retardo máximo

Como se adelantó en el apartado 6.2, si consideramos que el número máximo de trayectos no varía, se puede apreciar una tendencia en la relación entre la tasa de error de bit y el retardo máximo de los trayectos secundarios respecto al principal.

Tanto el decisor simple como el adaptativo no se ven afectados por los errores cuando el retardo máximo está por debajo de $1.5\,T$. La concentración de trayectos secundarios cerca del periodo de

símbolo hacen que la ISI no sea lo suficientemente alta para confundir al decisor. A partir de 1.5 T hay una tendencia creciente en la tasa de error en los dos decisores, alcanzándose el máximo para 2 T en el decisor adaptativo y 3 T para el decisor simple.

A partir de ahí, hay un descenso en la tase de error, explicable por la mayor separación de las contribuciones secundarias, que hace que la ISI no esté tan concentrada. En el decisor adaptativo, el descenso es más significativo a medida que aumenta el retardo, ya que el ecualizador puede ajustar sus coeficientes con mayor precisión.

Influencia de la codificación de línea y la modulación utilizada ante la propagación multitrayecto

Como vimos en el apartado 6.3, la codificación de línea y la modulación utilizada son relevantes en las tasas de error obtenidas cuando se atraviesa un canal multitrayecto.

En líneas generales, los mejores resultados se obtuvieron con codificaciones Manchester y RZ. Mientras que en el caso de la modulación, el mejor resultado se obtuvo la modulación con pulsos rectangulares, seguida de la modulación con pulsos RCCA con factor de redondeo 1.

Se concluyó que tanto en la modulación como en la codificación, la mayor dispersión espectral de la señal es beneficiosa ante la propagación multitrayecto. El desvanecimiento selectivo en frecuencia, propio del multitrayecto, tiene mayor impacto sobre las señales con el ancho de banda más concentrado, afectando a mayor contenido útil de la señal.

7.3. Líneas futuras

Los modelos planteados en este proyecto son aproximaciones relativamente sencillas a escenarios realmente complejos, por lo que existe una evidente línea de mejora aumentando el detalle de los modelos empleados, lo que enriquecería las simulaciones. Además, comLAB es una plataforma en continuo desarrollo, por lo progresivamente integrará nuevas herramientas y funcionalidades que la harán una herramienta de simulación más potente. A continuación, se plantean algunos de los ámbitos de mejora:

- *Incorporación de un mecanismo de sincronismo en el receptor de comLAB*. Actualmente, el receptor no dispone de sincronización propia, sino que depende del retardo artificial introducido por el canal. Integrar un sistema de sincronismo automático permitiría al receptor adaptarse a diferentes retardos de canal y evitaría los errores iniciales provocados por las

muestras transitorias, mejorando la fiabilidad de las simulaciones.

- *Gestión del retardo inicial en la adaptación del ecualizador.* Para evitar que el algoritmo RLS se adapte sobre muestras transitorias no representativas de la señal transmitida, sería conveniente incorporar un mecanismo que inhiba la actualización de los coeficientes durante el retardo de sincronismo. Esta mejora facilitaría la convergencia y permitiría evaluar de manera más precisa el desempeño del decisor adaptativo en el entorno de comLAB.

- *Implementación de modulaciones paso-banda*. A pesar de que la propagación multitrayecto solo tiene sentido en señales paso-banda, por considerar comunicaciones inalámbricas, en nuestras simulaciones hemos obviado esta circunstancia. Se ha asumido que hay implícito un proceso de modulación y demodulación en radiofrecuencia antes y después del canal, y sólo consideramos las señales en banda-base. La incorporación de la modulación y demodulación en radiofrecuencia a nuestro modelo daría una aproximación más realista al sistema de transmisión.

- *Incorporación de filtro de suavizado en el bloque de multitrayecto*. El filtro de suavizado que incorporamos en nuestras simulaciones en la consola de MATLAB, que proporcionaba variaciones de las respuestas al impulso del canal más reales, no se ha incorporado a la simulación en comLAB. Esta sería un mejora considerable del modelo.

- *Consideración de modulaciones M-arias*. En las simulaciones realizadas siempre hemos considerado modulaciones binarias. La consideración de otras constelaciones no binarias puede completar el estudio.

- *Desarrollo de un modelo de multitrayecto más complejo*. El modelo de multitrayecto se puede enriquecer, diferenciando el entorno de propagación (*indoor*, *outdoor*, urbano, rural, etc.) y aplicando distribuciones de probabilidad más acordes (distribución de Rayleigh, distribución de Rice). A su vez, se pueden considerar el *efecto Doppler*, el *fast fading* o fuentes de ruido impulsivo.

- *Implementación de otras técnicas de ecualización adaptativa*. En este proyecto nos hemos limitado a modelar el algoritmo RLS sobre un filtro FIR transversal, pero existen numerosos algoritmos cuya simulación puede ser de interés (LMS, NLMS o Fast RLS), combinándolos con diferentes estructuras (transversal, en celosía, o de decisión retroalimentada, DFE).

REFERENCIAS

- Cellier, F. E. y Kofman, E. (2006). *Continuous System Simulation*. Nueva York, NY: Springer Science+Business Media.

- Díaz Nafría, J.M. (2021). Aplicaciones de filtrado adaptativo. *Enunciado de actividad de la asignatura Tratamiento Digital de la Señal*. UDIMA.

- glossaLAB (s.f.). *Co-Creating Interdisciplinary Knowledge*. Recuperado de https://www.glossalab.org/wiki/Main_Page

- Hata, M. (1980). Empirical Formulae for Propagation Loss in Land Mobile Radio Services, *IEEE Transactions on Vehicular Technology, VT-29*(3), 317–325.

- IBM (5 de agosto de 2021). *¿Qué es un gemelo digital?*. Recuperado de https://www.ibm.com/es-es/think/topics/what-is-a-digital-twin

- IONOS (18 de octubre de 2023). *Python vs. Matlab: Which language is right for you?*. Recuperado de https://www.ionos.com/digitalguide/websites/web-development/python-vs-matlab/

- Law, A.M. (2013). *Simulation Modeling and Analysis* (5ª edición). Nueva York, NY: McGraw-Hill Education.

- MathWorks (s.f.a). *Simulación y diseño basado en modelos con Simulink*. Recuperado de https://es.mathworks.com/products/simulink.html

- MathWorks (s.f.b). *Sample- and Frame-Based Concepts*. Recuperado de https://es.mathworks.com/help/dsp/ug/sample-and-frame-based-concepts.html?searchHighlight=frame-based&s_tid=srchtitle_support_results_1_frame-based

- MathWorks (s.f.c). *Write Level-2 MATLAB S-Functions*. Recuperado de https://es.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html?searchHighlight=S-functions&s_tid=srchtitle_support_results_9_S-functions

- MathWorks (s.f.d). *randi - Uniformly distributed random integers*. Recuperado de https://es.mathworks.com/help/matlab/ref/double.randi.html?searchHighlight=randi&s_tid=srchtitle_support_results_1_randi

- MathWorks (s.f.e). *rand - Uniformly distributed random numbers*. Recuperado de https://es.mathworks.com/help/matlab/ref/double.rand.html?searchHighlight=rand&s_tid=srchtitle_support_results_1_rand

- MathWorks (s.f.f). *randn - Normally distributed random numbers*. Recuperado de https://es.mathworks.com/help/matlab/ref/double.randn.html?searchHighlight=randn&s_tid=srchtitle_support_results_1_randn

- MathWorks (s.f.g). *filter - Filtrado digital 1D* . Recuperado de https://es.mathworks.com/help/matlab/ref/filter.html?searchHighlight=filter&s_tid=srchtitle_support_results_1_filter

- MathWorks (s.f.h).*Use DWork Vectors in S-Functions*. Recuperado de https://es.mathworks.com/help/simulink/sfg/dwork-matlab.html?searchHighlight=Dwork&s_tid=srchtitle_support_results_2_Dwork

- MathWorks (s.f.i). *rng - Controlar el generador de números aleatorios* . Recuperado de https://es.mathworks.com/help/matlab/ref/rng.html?searchHighlight=rng&s_tid=srchtitle_support_results_1_rng

- Okumura, Y.; Ohmori, E.; Kawano, T. y Fukuda, K. (1968). Field Strength and Its Variability in VHF and UHF Land Mobile Radio Service. *Review of the Electrical Communication Laboratory, 16*(9-10), 825-873.

- Patel, R. y Kamboj P. (2015). Investigation of Network Simulation Tools and Comparison Study: NS3 vs NS2. *Journal of Network Communications and Emerging Technologies, 5*(2).

- Proakis, J.G. y Manolakis, D.G. (2007). *Tratamiento digital de señales* (4ª edición). Madrid: Pearson Educación.

- Proakis, J.G. y Salehi, S. (2008). *Digital Communicactions* (5ª edición). Nueva York, NY: McGraw-Hill.

- Sklar, B. y Harris, F. (2021). *Digital Communications: Fundamentals and Applications* (3ª edición). Londres: Pearson Education.

ANEXOS

Anexo I. Códigos de MATLAB de las funciones de partida

Filtro antialiasing

```
function y = filtro_AA (x,M)
% y = filtro_AA (x,M)
% Realiza un fitrado antialiasing basado en filtrado de Butterworth de
% orden 20 (que minimiza la distorsión en la banda de paso y ofrece una
% transición suficientemente abrupta), haciendo coincidir la frecuencia
% de corte de submuestreo fsm/2 = fm/2M, donde fm: frec muestreo original.
% ENTRADAS
% x...secuencia de entrada
% M...factor de submuestreo
% SALIDA
% y...secuencia de salida filtrada (sin modificar la fm)
% AUTORES: J.M.Díaz-Nafría
[B,A] = butter (20,1/M); % Frec. corte normalizada (fm/2M)/(fm/2)=1/M
y = filter (B,A,x); % Filtra la señal de entrada usando el filtro
end
```

Submuestreo (diezmado)

```
function [y,tm,fsm] = muestreo (x,t,fm,M)
%% [y,tm,fsm] = muestreo (x,t,fm,M)
% Submuestrea la secuencia x en un factor M, quedándose con las muestras
% nM, y devuelve además la base de tiempos correspondientes así como el
% valor de la nueva frecuencia de muestreo tras el diezmado.
% ENTRADA
% x....secuencia de entrada (debe ser un vector)
% t....base de tiempos para de x
% fm...frecuencia de muestreo de la señal original
% M....factor de submuestreo (debe ser un entero)
% SALIDA
% y....secuencia submuestreada
% tm...base de tiempos de y
% fsm..nueva frecuencia de muestreo
% AUTORES: J.M.Díaz-Nafría
fsm = fm/M; % frecuencia de muestreo de la señal submuestreada
tm = t(1:M:end); % tm: vector compuesto por t(i) para i = M*n
y = x(1:M:end); % y: vector compuesto por x(i) para i = M*n
end
```

Compresión de ley A

```
function y = compr_A(x)
%% y = compr_A(x)
% Aplica la relación de compresión de la ley A (de compansión)
% y(x)= f1(x) = x*(A/(1+log(A)) si {abs(x) < 1/A}
% f2(x) = sign(x)*(1+log(A*abs(x)))/(1+log(A)) si {1/A <= abs(x) <= 1}
% f3(x) = sign(x) si {abs(x) > 1}
% ENTRADAS
% x.....entrada comprimida [-1, 1]
% SALIDAS
% y...salida expandida
% AUTORES: J.M.Díaz-Nafría
% Para la cuantificación de todo el vector de muestras de la señal de
% entrada de una sola vez (es decir, no elemento a elemento) se aplica el
% siguiente método:
% (1) se construye un vector que solo es 1 en los valores de x comprendidos
% en cada dominio de validez de las subfunciones (f1, f2, f3) de y(x),
% (2) éstos vectores se multiplica por la relación evaluada en todos los
% valores de x. El vector resultante sera 0 fuera de cada subdominio de
% fi(x).
% (3) Los pasos (1) y (2) se aplican a cada subfunción fi(x) y se suman los
% vectores obtenidos de la aplicación de fi(x) en su subdominio.
A = 87.6;
y = (abs(x)<1/A).*x*(A/(1+log(A))); % valores del primer segmento
% La siguiente modificación del vector x se hace para evitar que se produzca
% una indeterminación de la función logarítima en x=0 que se evalúa en la
% parte derecha de: [pertenencia subdominio] .* [f(x1), f(x2)... f(xn)]
x = (x==0)*1e-3 + x; % para evitar luego NaN en x=0
y = y + ((abs(x)>=1/A)&(abs(x)<1)).*sign(x).*(1+log(A*abs(x)))/(1+log(A))+...
(abs(x)>=1).*sign(x); % por encima de la saturación
end
```

Cuantificación uniforme

```
function xbb = cuant_u (xc,b)
%% [xqb] = cuant_u (xc,b)
% Cuantifica uniformemente la secuencia de entrada (normalizada a [-1, 1]
% UTN)usando un bit de signo, con 0 como umbral de decisión.
% ENTRADAS
% xc....secuencia muestreada en UTN
% b.....nº de bits de cuantificación
% SALIDAS
% xbb...secuencia cuantificada en binario como serie de bits (tipo single)
% AUTORES: J.M.Díaz Nafría
if max(abs(xc))>1
warning('La entrada transgrede los niveles de saturación de cuantificación [-1,1]');
end
%% CUANTIFICACIÓN UNIFORME | Cero: valor de decisión, Entrada: [-1,+1] UTN
N = 2^b; % No de niveles de cuantificación
xq =(abs(xc)<1).*floor(abs(xc)*N/2)+... % Cuantif.unif. entre umbrales de saturación
(abs(xc)>=1)*(N/2-1); % Cuantificación uniforme a partir de saturación
```

```
%% Generación de la secuencia binaria de la señal codificada
% Como matriz en los que cada fila: [S B1 B2 ... ], donde S: bit de signo
xqb = [int8((-sign(xc)+1)/2)',... % Bit de signo: 0 si x>0
de2bi(xq,'left-msb',b-1)]; % El bit más significativo a la izq
% Conversión a secuencia unidimensional (vector fila)
xbb = single(reshape(xqb',1,[])); % Secuencia de bits (vble tipo single)
end
```

Codificación de fragmento de señal

```
function [d,x] = cod_dat(signal,inicio,fin)
%% [datos,x] = cod_dat(signal,inicio,fin)
% Codifica la señal y fragmento elegido como secuencia de datos binarios
% (en serie) para su ulterior modulación. Las señales de sonido: se submuestrean
% de modo que se preserva hasta 4.4 KHz para la voz y 11 KHz para la
% música. Para ambas se usa cuantificación no uniforme basada ley A, usando
% 8 bits para la voz y 16 para la música. La señal digital de imagen solo se
% serializa para su ulterior codificación de línea.
% ENTRADAS
% signal...1: voz de Ramón y Cajal, 2: música de H.Dune, 3: imagen de Meucci
% inicio...primera muestra de la señal y fragmento elegido
% fin......última muestra de la señal y fragmento elegido
% SALIDAS
% datos....secuencia de datos binarios (en serie)
% x........señal original elegida (antes de la codificación)
% AUTOR: J.M.Díaz Nafría
%% Lectura y codificación de fuente (muestreo y cuantificación no uniforme)
switch signal
case 1 % Señal de voz de Ramón y Cajal
[x,fm] = audioread("RyC.wav");
t = (inicio:fin)/fm;
x = x(inicio:fin)*4; % Corrección de amplitud para [-1, 1]
y = filtro_AA(x,5); % Filtrado antialiasing
y = muestreo(y,t,fm,5); % Submuestreo
y = compr_A(y); % Compresión con ley A
d = cuant_u(y,8); % Cuantificación uniforme
case 2 % Señal musical de Herman Dune
[x,fm] = audioread("H_D.mp3");
t = (inicio:fin)/fm;
x = x(inicio:fin);
y = filtro_AA(x,2); % Filtrado antialiasing
y = muestreo(y,t,fm,2); % Submuestreo
y = compr_A(y); % Compresión con ley A
d = cuant_u(y,16); % Cuantificación uniforme
case 3 % Señal fotográfica de Meucci
x = imread('Antonio_Meucci.gif'); % x es un array de enteros [400 x 400]
y = x';
d = de2bi(y(inicio:fin)); % d es un array [160.000 x 8] lin.hor.
d = reshape(d,1,[]); % Serializa los datos
end
end
```

Símbolo básico

```
function x_b = s_b(tipo,Ms,Ns,r)
%% x_b = s_b(tipo,Ms,Ns,r)
% Genera el símbolo básico para varios códigos de línea de duración finita
% o de AB finito. En caso de seleccionar Ns=1, se asume que se trata de
% duración finita basados en formas rectangulares, y en cualquier otro
% caso en versiones truncadas de pulsos de Raíz Cuadrada de Coseno Alzado
% (RCCA) de duración NsTs.
% La secuencia x_b generada es de energía Ms (o su repetición P=1).
% ENTRADA
% tipo : 1:'NRZ', 2:'RZ', 3:'Manchester'
% Ms : Nº de muestras por símbolo (debe ser par si se trata de códigos
% con cambio dentro de símbolo)
% Ns : duración del pulso básico en nº de periodos de símbolo
% r : Factor de redondeo (roll-off) en caso de tratarse de pulsos en RCCA
% SALIDA
% x_b : Pulso básico
% DESARROLLO
% 6-7/12/24: se modifica para poderlo integrar con el desarrollo de
% simulink_codificador.m y en conjunto poder integrar NRZ, RZ y
% Manchester, a la vez que los parámetros de constelación y código definen
% la constelación binaria (unipolar, bipolar) o de orden superior. Lo que
% supone la definición de un modulador digital 1D.
% En el futuro se podría:
% (i) incorporar pulsos básicos gaussianos;
% (ii) combinarlo con otro módulo para permitir codificación diferencial;
% (iii) combinar bloques de codif./mod. para realizar modul. de orden superior.
% (iv) incorporar pulsos básicos paso banda a partir de s_b (paso bajo)
% (v) hacer combinaciones como en (iii) para dar lugar a mod.paso banda
% PSK, QPSK, FSK, OFDM
% (v) incorporar pulsos MSK y GMSK para poder generar dichas modulaciones.
% AUTOR: J.M.Díaz Nafría
%% Duración del pulso sub-básico (para la conformación del pulso básico)
% Los Pulsos con Retorno a Cero (RZ y Manchester) se obtienen mediante
% combinación de dos pulsos sub-básicos de duración Ms/2 (o con pasos por
% cero en Ms/2 para AB finito), uno de ellos retrasado Ms/2. En estos casos
% devuelve error en este caso si Ms no es par. En el resto de los códigos
% contemplados se asume una duración Ms/2.
% Se emplea k para determinar el periódo sub-básico D = Ms/k (distancia entre
% ceros) y la duración del pulso sub-básico en nº de periodos, 2 Ns.
% k=1 para NRZ, k=2 para RZ o Manchester
% MENSAJE DE ERROR: Si el tipo de código no estuviera entre los contemplados.
if tipo==2 || tipo==3 % Pulsos RZ
if mod(Ms,2)==1 % Ms no es par
error ('El nº de puntos para la simulación de un símbolo del tipo elegido debe ser par');
end
k = 2; % se emplea para considerar 2 Ns en el pulso sub-básico
elseif tipo==1 % Pulsos NRZ
k = 1; % duración del pulso sub-básico: todo el símbolo
else
error('El tipo especificado no está contemplado, debe ser 1, 2 o 3. 1: NRZ, 2: RZ, 3: Manchester');
end
```

```
D = Ms/k;
%% Generación de los Pulsos sub-básicos
% Pulsos de duración Ts (AB ilimitado, forma RECTANGULAR)
if Ns==1 % Pulsos de duración finita (rectangulares)
n = 1:Ms; % Índices hasta fin de símbolo
x_b = (n<=D); % Pulsos rectangulares
x_b = x_b*sqrt((k-(tipo==3)*1)/Ms);
else % Si se introducen más de dos argumento == rcca
% Pulsos de ancho de banda finito (RCCA)
if r>1 || r<0
error('El factor de redondeo del coseno alzado debe estar comprendido en [0,1]');
end
% En la siguiente línea se ha de incluir el parámetro 'normal' al final
% para generar pulsos en coseno alzado, de lo contrario son en RCCA.
x_b = rcosdesign(r,(Ns+1)*k,D); % ,'normal'); % Pulsos en RCCA, Es=1
% Se extiende a Ns+1 para que el efecto de truncamiento reduzca la
% distorsión, lo que hará que la Es tras el truncamiento sea < 1.
end
%% Código MANCHESTER
if tipo==3
ns = 1:Ms/2+1;
d_b = ((ns==1)-(ns==Ms/2+1)); % Delta(n)-Delta(n-Ms/2)
if Ns==1 % Si es de duración limitada (RECTANG.)
x_b = x_b(1:Ms/2); % Se recorta para limitar duración a Ts
else % Si es de ancho de banda limitado (RCCA)
d_b = d_b/sqrt(2); % normaliza / E_s=1 para pulsos de AB finito
end
x_b = conv(d_b,x_b); % Conformación del pulso básico
% x_b = x_b(1+Ms:Ns*Ms+Ms+1); % truncamiento para que la longitud sea Ns*Ms+1
% OBS: Con este truncamiento la duración anterior y posterior no es
% simétrica, sin embargo al eliminar un lóbulo completo la señal
% empieza y acaba en cero. En la recepción tras el FA el máximo se logra
% en cualquier caso en y(Ms*Ns).
end
%% Truncamiento de los pulsos de AB finito (RCCA)
% Aquí se emplea k de forma diferente: para determinar el origen y fin del
% truncamiento. En los códigos Manchester k=1 para lograr que los extremos
% no presenten transiciones bruscas (aunque el pulso truncado no sea
% simétrico) con objeto de reducir la IES. En los NRZ y RZ no hace falta y
% k=2, haciendo que el pulso resultante sea más simétrico y minimice la IES.
% La duración del pulso truncado en cualquier caso es Ns*Ms. En la recepción
% tras el FA el máximo se logra en cualquier caso en y(Ms*Ns).
if Ns>1
if tipo == 3
k = 1;
else
k = 2;
end
x_b = x_b(1+Ms/k:Ns*Ms+Ms/k);
end
%% Normalización a P=1 (se compensa la pérdida de energía del truncamiento)
E = sum(x_b.^2); % Se calcula la E discreta
x_b = x_b*sqrt(Ms/E); % Para que la potencia sea 1 [UPN]
```

```
%% Representación del pulso básico (desmarquese para representar el pulso básico)
% t = (0:length(x_b)-1)/Ms; % tiempo normalizado al tiempo de símbolo
% plot(t,x_b); grid on;
% xlabel('{\itt} [normalizado a {\itT_s}]');
% ylabel('{\its_b}({\itt}) [UTN], E_s = Ts');
end
```

Generador de señal de línea

```
function [x_lin] = x_lin(Repr,datos,Ms,tipo,varargin)
%% [x_lin] = x_lin (Repr,datos,Ms,tipo,{Ns,r})
% Genera una señal de línea usando códigos básicos y pulsos de duración
% finita. Los parámetros de entrada Ns y r son opcionales, y en caso
% de indicarse la función considera que se trata de un pulso de AB finito
% (basado en pulsos en raíz cuadrada de coseno alzado RCCA).
% Además representa gráficamente la señal de linea.
% ENTRADA
% Repr...1/0: Hace / No hace representación gráfica de la señal
% datos..Secuencia de datos
% Ms.....Nº de muestras por símbolo (debe ser par en códigos con cambio
% dentro de símbolo)
% tipo : 1:'NRZ', 2:'RZ', 3:'Manchester'
% {Ns}..Nº de símbolos entrelazados (extensión del pulso de AB limitado)
% {r}...Factor de redondeo (rolloff) en caso de tratarse de pulsos en RCCA
% Donde {.} son variables opcionales. Si no se introducen se asume pulsos
% de duración limitada.
% SALIDA
% x_lin..Señal de línea correspondiente a los datos
% AUTOR: J.M.Díaz Nafría
Nd = numel(datos);
%% Generación de los pulsos básicos
if nargin==4
x_b = s_b(tipo,Ms,1,0);
else
x_b = s_b(tipo,Ms,varargin{:}); % Evita tipos no contemplados
end
%% Generación de la señal de línea
d = (datos*2-1)'*((1:Ms)==1);
d = reshape(d',1,[]); % Deltas moduladas con los datos
d = d(1:Ms*(Nd-1)+1); % Recorta los ceros tras la última delta
x_lin = conv(d,x_b); % Señal de linea d_dat(t)*s_b(t)
%% REPRESENTACIÓN GRÁFICA de la señal de línea implementada
if Repr == 1 % Representación condicional
Nx = numel(x_lin); % Nº de elementos de la señal de línea
if nargin==4 % Pulsos rectangulares
t = (0:(Nx)-1)/(Nx-1)*Nd;
else % Pulsos limitados en banda con Ns, r (rolloff)
Ns = varargin{1};
% Dependiendo del tipo de código de linea la longitud de la señal es
% diferente, lo que afecta a la base de tiempos usada en la
% representación de la señal.
if tipo==1 % Códigos NRZ (lim en AB)
```

```
t = (-Ms*Ns/2:(Nd-1)*Ms+Ms*Ns/2)/Ms;
elseif tipo==2 % Códigos RZ (lim en AB)
t = (-Ms*Ns/4:(Nd-1)*Ms+Ms*Ns/4)/Ms;
else
t = (-Ms*Ns/4:(Nd-1)*Ms+Ms/2+Ms*Ns/4)/Ms; % Manchester (lim en AB)
end
end
plot(t,x_lin); grid on;
ylabel('{\itv_s}({\itt}) [UTN]'); % Etiqueta del eje de ordenadas
% Se oculta la siguiente linea si se desea evitar la etiqueta del eje
% horizontal para una representación en paralelo de varias subgráficas.
xlabel('{\itt} x {\itT_b} [s]'); % Etiqueta del eje de ordenadas
end
end
```

Generador de fragmento de señal de línea

```
function [datos, x_l, t] = s_linea(s,Rb,signal,inicio,fin,Ms,tipo,varargin)
%% [datos,x_l,t] = s_linea(s,Rb,signal,inicio,fin,Ms,tipo,{Ns,r})
% Genera la señal de línea para la señal de información y fragmento elegido
% ENTRADAS
% s........Potencia de la señal [W]
% Rb.......Régimen binario de transmisión (bps)
% signal...1: voz de Ramón y Cajal, 2: música de H.Dune, 3: imagen de Meucci
% inicio...primera muestra de la señal y fragmento elegido
% fin......última muestra de la señal y fragmento elegido
% tipo : 1:'NRZ', 2:'RZ', 3:'Manchester'
% {Ns}..Nº de símbolos entrelazados (extensión del pulso de AB limitado)
% {r}...Factor de redondeo (rolloff) en caso de tratarse de pulsos en RCCA
% Donde {.} son variables opcionales. Si no se introducen se asume pulsos
% de duración limitada.
% SALIDAS
% datos....Datos codificados a los que corresponde la señal de línea
% x_l......Señal de línea con potencia s; amplitud en [UTN] / P(t)=x(t)^2
% t........Base de tiempos de la señal de línea (s)
% FUNCIONES requeridas:
% cod_dat, muestreo, filtro_AA, compr_A, cuant_u, x_lin, s_b
% AUTOR: J.M. Díaz Nafría
%% generación de los datos del fragmento elegido
datos = cod_dat(signal,inicio,fin); % Codificación de señal de voz (sec. de bits)
datos = single(datos);
%% generación de la señal de línea
if nargin == 7 % Pulsos rectangulares (limitados en el tiempo)
x_l = sqrt(s)*x_lin(0,datos,Ms,tipo);
else % Pulsos en raíz cuadrada de coseno alzado
Ns = varargin{1}; r = varargin{2};
x_l = sqrt(s)*x_lin(0,datos,Ms,tipo,Ns,r);
end
t = (1:numel(x_l))/(Rb*Ms); % Base de tiempos para la señal de línea (s)
end
```

Multitrayecto

```
function x_mt = multitrayecto(x,P,Mmin,Mmax,sigma,T)
%% funtion x_mt = multititrayecto(x,P,Mmin,Mmax,sigma,T)
% Aplica una distorsión multitrayecto dinámica correspondiente a una secuencia de N
% impulsos cuya amplitud, fase y retardo es aleatorio. La amplitud se considera
% gaussiana de media nula, la fase tiene una distribución uniforme, así
% como los retardos entre un valor mínimo (Mmin) y otro máximo (Mmax). Todos
% los retardos se determinan en tiempo discreto.
% El retardo, amplitud y fase del trayecto directo es cte = Mmin,1,0.
% Para reducir la complejidad se subdivide la señal en tramas de longitud
% cte, P, y se cambian los parámetros del multitrayecto {r},{theta},{tau}
% en cada trama de forma aleatoria, luego se filtran a wc = 0.1 para evitar
% transiciones bruscas, por tanto, la variación de multitrayectos
% independientes es de: Tm*T*10 = (Tb/Ms)*T*10 = T*10/(Ms*Rs)
% ENTRADAS
% x......señal de entrada
% P......Nº de componentes (pulsos) del multitrayecto
% Mmin...Retardo mínimo en nº de muestras de la señal de línea
% Mmax...Retardo máximo en nº de muestras de la señal de línea
% sigma..Desviación de la amplitud de los impulsos secundarios
% T......longitud de las tramas en las que el multitrayecto es cte
% SALIDAS
% x_mt...Señal con distorsión multitrayecto dinámico
% AUTOR: J.M. Díaz Nafría
%% Modelo de multitrayecto dinámico
% Se subdivide la señal en tramas de longitud T, considerándose estable por
% el multitrayecto en cada trama y variando aleatoriamente entre estas. La
% secuencia de parámetros se filtra a wc=0.1.
Nx = length(x); % Longitud de la secuencia x(n)
Nt = ceil(Nx/T); % Nº de tramas
ext = 0; % Entrelazado de tramas
x_mt = zeros(1,Nt*T+2*ext); % Salida vacía
x_mt(ext+1:Nx+ext) = x;
[b,a] = butter(6,0.1); % filtro para suavizar las variaciones
tau = filter(b,a,randi([Mmin,Mmax],Nt,P-1));% retardos respecto al impulso ppal
theta = filter(b,a,rand(Nt,P-1)*2*pi-pi); % fase de los impulsos secundarios
r = filter(b,a,randn(Nt,P-1)*sigma); % amplitud de los impulsos secundarios
% Matrices tau, theta y r de dimensión: (P x Nt)
tau = [ones(1,Nt)*Mmin; tau']; % Se añade tau del principal = Mmin
theta = [zeros(1,Nt); theta']; % Se añade theta del principal = 0
r = [ones(1,Nt); r']; % Se añade amplitud el principal = 1
%% Aplicación de H_m_k(f) por cada trama (k=1...Nº de tramas)
% Se solapan las tramas para evitar efecto de transición entre tramas y
% solo se preserva la parte correspondiente a la trama: ext | trama | ext
for k = 1:Nt
w = (0:T+2*ext-1)/(T+2*ext)*2*pi; % Frecuencias angulares
H = (ones(length(w),1)*(r(:,k)'.*exp(-1i*(theta(:,k)')))).*...
exp(-1i*w'*tau(:,k)'); % [1 1...]x[H0,H1...Hp]->
H = sum(H,2); % Suma de los trayectos
Y = fft(x_mt((k-1)*T+1:k*T+2*ext)).*H'; % Y(w) = X{trama k} * H(w)
x_ext = real(ifft(Y));
x_mt(((k-1)*T+1:k*T)+ext) = x_ext((1:T)+ext); % y{trama k} = IFFT(Y)
```

```
end
%% filtro para suavizar saltos de trama (desmarcar 2 líneas para hacer el filtrado)
% [b,a] = butter(10,0.25); % filtro para suavizar saltos de trama
% x_mt = filter(b,a,x_mt); % Suaviza transiciones de trama
x_mt = x_mt((1:Nx)+ext); % Recorta ceros' inicio y fin
%% Desmarcar para representar gráficamente el efecto sobre una señal
% x = filter(b,a,x); % Para comparar con x_m suavizado
% plot([x',x_mt']), grid on
% if Nx>2000, xlim([max(Nx)-1100, max(Nx)-100]), end
end
```

Ruido blanco

```
function nx = ruido_blanco(No,N,fm)
%% nx = ruido_blanco(No,N,fm)
% Genera una secuencia de ruido de d.e.p. No y longitud N.
% Se aplica un filtrado a fm/8=fn/4 para suavizar la forma del ruido asumiendo
% que el AB de las señales afectadas por el ruido es inferior a esta frecuencia.
% Observe que si el muestreo no cumpliera esta condición debería eliminarse
% el filtrado o modificarlo.
% ENTRADA
% No....d.e.p. de ruido (W/Hz)
% N.....nº de puntos de la secuencia de ruido
% fm....frecuencia de muestreo
% SALIDA
% nx....secuencia de ruido de potencia n y longitud N
nx = randn(1,N)*sqrt(No*fm/2); % La varianza se ajusta a la potencia de
% ruido para un AB a fm/2 (frec. Nyquist)
% [b,a] = butter(10,0.25); % Se filtra a fm/8=fN/4 asumiendo que el muestreo
% nx = filter(b,a,nx); % está bastante por encima de f_Nyq de las señales
end
```

Receptor digital

```
function [d_Rx,y,z,rec] = Rx_dig(x_l,FA,Ms,Ns,a,c)
% Realiza una recepción óptima de la señal de línea recibida, devolviendo
% los datos detectados.
% x_l(t) ->{FA}-> y(t) ->{Muestreo(kT)}-> {MAP}-> d_Rx
% ENTRADAS
% x_l : Señal de línea recibida
% FA : Filtro adaptado (no es necesario indicar el cód. de línea)
% Ms : Nº de muestras por símbolo
% Ns : Nº de símbolos entrelazados (extensión del pulso de AB limitado)
% a : amplitudes esperadas de mayor a menor (constelación) [M x 1]
% c : códigos correspondientes para cada amplitud [M x b]
% SALIDA
% d_Rx : Datos detectados como resultado de la detección óptima
% y : Salida del FA: y
% z : muestras de y: z
% rec : valores de reconstrucción
% FUNCIONES requeridas: decisor()
```

```
% AUTOR: J.M. Díaz Nafría
%% Filtrado adaptado-----------------------------------------------------------
y = conv(x_l, FA); % convolución completa incl. fuera de trama
y = y(Ms*Ns:end-(Ms*Ns-1)); % Parte de la señal que interesa (en trama)
%% Decisor (MAP) basado en muestras de salida de FA en instantes óptimos----
z = y(1:Ms:end); % Muestras óptimas de la señal
[d_Rx,rec] = decisor(z,a,c); % función DECISOR: MAP usando {ai} y {ci}
%% OBSERVACIÓN SOBRE LA PRESERVACIÓN DE LA SEÑAL DE INTERÉS
% Si representamos gráficamente el cálculo de la convolución paso a paso:
%
% 1) Comienza la convolución (n = 1):
%
% |-solape-|-trama-actual----------------------|
% |-s_b----|
%
% 2) Llegamos a la convolución completa del solape (n = MsNs+1):
%
% |-solape-|-trama-actual----------------------|
% |-s_b----|
% En este punto ambos vectores ya se superponen totalmente.
%
% 3) Termina la convolución completa (n = MsNs+1+TMs):
%
% |-solape-|-trama-actual----------------------|
% |-s_b----|
%
% 4) Termina la convolución incompleta (n = MsNs+1+TMs+MsNs):
%
% |-solape-|-trama-actual----------------------|
% |-s_b----|
%
% Solo interesa el resultado entre (2) y (3) tanto como señal que se unirá
% a la trama siguiente para dar lugar a una señal continua, como para tomar
% las muestras en las que se basa la decisión.
```

Filtro adaptado

```
function [ym,y] = FiltroAdaptado(x_l,tipo,Ms,varargin)

%% [ym,y] = FiltroAdaptado(x_l,tipo,Ms,{Ns,r})
% Realiza un filtrado y muestreo óptimo de la señal de línea recibida, en
% los instantes teóricamente óptimos de detección.
% ENTRADAS
% x_l....Señal de línea recibida
% tipo : 1:'NRZ', 2:'RZ', 3:'Manchester'
% Ms.....Nº de muestras por símbolo
% {Ns}...Nº de símbolos entrelazados (extensión del pulso de AB limitado)
% {r}....Factor de redondeo (rolloff) en caso de tratarse de pulsos en RCCA
% Donde {.} son variables opcionales. Si no se introducen se asume pulsos
% de duración limitada.
% SALIDA
% ym.....Señal muestreada tras filtro
```

```
% y......Señal filtrada y centrada (descartando colas)
% AUTOR: J.M. Díaz Nafría
%% Respuesta al impulso
if nargin==3 % Pulsos rectangulares
Ns = 1; r = 0;
else % Pulsos RCCA
Ns = varargin{1}; r = varargin{2};
end
h = fliplr(s_b(tipo,Ms,Ns,r)); % Respuesta al impulso de FA
%% Filtrado adaptado
y = conv(x_l,h); % Filtrado
y = y(Ms*Ns:end-(Ms*Ns-1)); % Parte de la señal que interesa (en trama)
%% Muestreo en instantes óptimos
ym = y(1:Ms:end); % Muestras óptimas de la señal
```

Decisor

```
function [d,y] = decisor(xm,a,c)
% [d,y] = decisor(xm,a,c)
% Realiza la decisión unidimensional a partir de los valores de las muestras
% y de acuerdo a la constelación ORDENADA {a} y su correspondiente código {c}.
% ENTRADA
% xm : secuencia de muestras [M x 1] ó [1 x M]
% a : amplitudes esperadas de mayor a menor (constelación) [M x 1]
% c : códigos correspondientes para cada amplitud [M x b]
% SALIDAS
% d : secuencia de datos decodificados (códigos en {c})
% y : secuencia de amplitudes reconstruidas (incluidas en {a})
% AUTOR: J.M. Díaz Nafría
xm = reshape(xm,1,[]); % xm en vector fila
u = (a(1:end-1)+a(2:end))/2; % Umbrales de decisión
M = length(a); % Orden de la modulación
% A continuación se hace una comparación de todas las muestras con los
% umbrales de cada cada subintervalo ordenados de mayor a menor y se
% reconstruye con los valores correspondientes a cada región de decisión.
for k=1:M
if k == 1 % Reconstrucción en el intervalo superior
y = (xm>u(1))*a(1); % Reconstrucción del valor de la muestra
d = (xm>u(1))'*c(1,:); % Reconstrucción del código
elseif k < M % Reconstrucción en los intervalos intermedios
y = y+((xm>u(k))&(xm<=u(k-1)))*a(k);
d = d+((xm>u(k))&(xm<=u(k-1)))'*c(k,:);
else % Reconstrucción en el intervalo inferior
y = y+(xm<=u(M-1))*a(M);
d = d+(xm<=u(M-1))'*c(M,:);
end
end
d = boolean(reshape(d',[],1)); % Salida en vector columna
y = reshape(y,[],1); % Salida en vector columna
end
```

Anexo II. Códigos de MATLAB desarrollados en el proyecto

Multipath

```
function [x_mt,B] = multipath(x,Pmax,Tmax,sigma,Nt)
%% Función que simula la propagación multitrayecto
% Divide la señal de entrada en subtramas, a las que aplica respuestas al
% impulso variantes, mediante la función filter().
% Cada respuesta al impulso caracteriza la propagación multitrayecto en un
% intervalo de tiempo diferente.
% La caracterización se realiza mediante las variables aleatorias:
% - Número de trayectos (P): Distribución uniforme [1,Pmax]
% - Amplitud de trayectos secundarios (r): Distribución normal con d. típica sigma
% - Fase de trayectos secundarios (theta): Distribución uniforme [-pi,pi]
% - Retardo de los trayectos secundarios (tau): Distribución normal [2,Tmax]
% Se asume que el trayecto principal tiene amplitud 1, fase 0 y retardo 1.
% ENTRADAS
% - x......Señal de entrada
% - Pmax...Nº máximo de componentes del multitrayecto
% - Tmax...Retardo máximo en nº de muestras
% - sigma..Desviación típica de la amplitud de los trayectos secundarios
% - Nt.....Número de subtramas con características de canal diferente
% SALIDAS
% - x_mt...Señal con distorsión multitrayecto
% - B......Matriz de dimensiones [Nt x Tmax] con las respuestas al impulso para cada subtrama
Nx = length(x); % Longitud de la secuencia de entrada
T = floor(Nx/Nt); % Longitud de las Nt-1 primeras subtramas. La última incluirá el resto
% Vector de trayectos correspondientes a cada subtrama
P = randi([1,Pmax],Nt,1);
% Generación de matrices tau, theta y r
tau = randi([2,Tmax],Nt,Pmax-1); % Matriz de retardos de los trayectos secundarios
theta = rand(Nt,Pmax-1)*2*pi-pi; % Matriz de fases de los trayectos secundarios
r = randn(Nt,Pmax-1)*sigma; % Matriz de amplitudes de los trayectos secundarios
% Filtro de suavizado sobre tau, theta y r
% [b,a] = butter(6,0.1); % Filtro para suavizar las variaciones de las variables aleatorias
% tau = filter(b,a,tau);
% tau = max(2, round(tau)); % Para asegurar que tau sea entero y >=2
% theta = filter(b,a,theta);
% r = filter(b,a,r);
r_complejo = abs(r).*exp(1j*theta); % Matriz de amplitudes y fases conjuntas
% Construcción de matriz B (coeficientes del filtro FIR) [Nt x Mmax]
B = zeros(Nt, Tmax); % Inicializar B con ceros
B(:,1)=1; % Primera columna todo unos (trayecto principal)
% Recorrido de tau y asignación de B(indice según tau)=r_complejo
for n = 1:Nt % Recorremos filas
for k = 1:(P(n)-1) % Recorremos columnas solo hasta trayectos secundarios activos
col = tau(n,k); % Retardo como índice
B(n,col) = B(n,col) + r_complejo(n,k); % Si se repite valor de tau se añade
end
end
```

```
zi = zeros(1,Tmax-1); % Valores iniciales del estado del filtro
x_mt = zeros (1,length(x)); % Inicialización de salidas
% Aplicación de filter() a subtramas (de primera a penúltima)
for i=1:Nt-1 % Esto se salta si Nt=1
x_subtrama = x(((i-1)*T+1):i*T); % Recorrido de x mediante subtramas de tamaño T
h = B(i,:); % Coeficientes del filtro correspondientes a cada subtrama
[y,zf]=filter(h,1,x_subtrama,zi); % Llamada al filtro para cada subtrama
x_mt(((i-1)*T+1):i*T) = real(y); % Resultado del filtro (parte real) a vector de salida
zi=zf; % Paso de estado para siguiente iteración
end
% Última subtrama. También es el caso de solo 1 subtrama (Nt=1)
x_subtrama = x(((Nt-1)*T+1):end); % Última subtrama (incluye resto)
h = B(Nt,:); % Última fila
[y,~]=filter(h,1,x_subtrama,zi); % Llamada al filtro para la última subtrama
x_mt(((Nt-1)*T+1):end) = real(y); % Resultado del filtro (parte real) a vector de salida
end
```

Decisor adaptativo

```
function [d_cod,d,y,e,h_out,P_out,h_hist] = DecisorAdaptativo(xm,a,c,M,lambda,h,P)
%% DecisorAdaptativo
% Realiza la decisión, a partir de las muestras en instantes óptimos tras el
% filtro adaptado, utilizando ecualización adaptativa. Mediante un filtro FIR
% que adapta sus coeficientes según el algoritmo RLS.
% ENTRADAS:
% xm: Señal recibida de canal tras pasar por filtro adaptado + muestreo
% a: Vector columna de amplitudes de la constelación (Mx1)
% c: Códigos correspondientes a cada amplitud (Mxb)
% M: Orden del filtro FIR
% lambda: Factor de olvido de algoritmo RLS
% h: Valores iniciales de los coeficientes del ecualizador (heredados de trama anterior)
% P: Valores iniciales de matriz P (heredados de trama anterior)
% SALIDAS:
% d_cod: Vector de decisiones codificado según c
% d: Vector de decisiones después de ecualizador adaptativo
% y: Vector de salidas de ecualizador adaptativo, previo a decisor
% e: Error entre decisiones y salida
% h_out: Últimos valores de h (para siguiente trama)
% P_out: Últimos valores de P (para siguiente trama)
N = length(xm) - (M-1); % Longitud de trama de entrada - Solape (condiciones iniciales)
b = log2(numel(a)); % Bits por símbolo
xm=xm(:); % Forzar entrada como columna
% Inicialización de vectores:
y = zeros(N,1); % Vector de salida de ecualizador adaptativo
d = zeros(N,1); % Vector de decisiones
e = zeros(N,1); % Vector de errores
d_cod = zeros(N*b, 1); % Vector de decisiones codificado
% Para guardar evolución de h
h_hist = zeros(M,N);
% Muestras desde 1 a N:
for n=1:N
x = flipud(xm(n : n+M-1)); % Selección de entrada al filtro de x(n) a x(n-M+1)
```

```
y(n)=x'*h; % Salida de ecualizador para coeficientes actuales
[bits_n, d(n)]=decisor(y(n),a,c); % Decisión para salida de ecualizador
d_cod((n-1)*b+1 : n*b)=bits_n; % Escritura en vector de decisiones codificado
e(n)=d(n)-y(n); % Error (Señal decidida - Señal ecualizada)
k=(P*x)/(lambda + x'*P*x); % Vector de ganancia de Kalman
P=(1/lambda)*(P-k*x'*P); % Actualización de matriz P
h_hist(:,n) = h; % Actualización del histórico de h
h=h+k*e(n); % Actualización de coeficientes para siguiente iteración
end
% Salidas para siguiente trama
h_out=h;
P_out=P;
%Salida decodificada como boolean
d_cod = boolean(d_cod);
end
```

Ejemplo de ciclo de transmisión completo

Ciclo_de_transmision.m

```
% Ejemplo de ciclo de transmisión completo
clear; close all; clc;
% Generación de señal de línea
s=1; % Potencia 1 W
Rb=1e6; % Tasa binaria 1 Mpbs
signal=1; % Señal elegida Ryc.wav
inicio=10000; % Primera muestra
fin=40000; % Última muestra
Ms=32; % Muestras por símbolo
tipo=1; % Tipo de codificación de línea NRZ
Ns=1; % Elegimos pulsos rectangulares
r=0; % Roll-off (no relevante en pulsos rectangulares)
[DATOS_IN,X1,~]=s_linea(s,Rb,signal,inicio,fin,Ms,tipo,Ns,r);
% Paso por canal multitrayecto
Pmax=20; % Número máximo de trayectos
Tmax=320; % Retardo máximo (en muestras)
sigma=0.25; % Desviación típica de la amplitud de los pulsos secundarios
Nt=4; % Número de subtramas (variaciones del canal)
[X2,B]=multipath(X1,Pmax,Tmax,sigma,Nt);
% Adición de ruido blanco
No=1e-7; % DEP de ruido
fm=(44100/5)*8*Ms; % Para calcular el AB de ruido
N=length(X2); % Longitud de trama de entrada para ruido_blanco()
X3=X2+ruido_blanco(No,N,fm);
% Paso por filtro adaptado y muestreo
X4=FiltroAdaptado(X3,tipo,Ms,Ns,r);
% Etapa de decisión
M=16; % Orden del filtro adaptativo
lambda=0.99; % Factor de olvido del algoritmo RLS
delta=0.01; % Para inicializar la matriz P
P=(1/delta)*eye(M); % Inicialización de matriz P
```

```
h = [1;zeros(M-1,1)]; % Inicialización de h como [1;0;...;0]
X4_ext = [zeros(1,M-1),X4]; % Se añaden M-1 valores iniciales a cero
a=[Ms,-Ms]';c=[1,0]'; % Amplitudes y códigos para decisor
% Salida del decisor (sin ecualización adaptativa)
[DATOS_SOLO_DECISOR,d_SOLO_DECISOR] = decisor(X4,a,c);
% Salida del decisor adaptativo
[DATOS_DECIS_ADAP,d_DECIS_ADAP,y,e,h_out,P_out,h_hist] = DecisorAdaptativo(X4_ext,a,c,M,lambda,h,P);
% Con decisor adaptativo
ERRORES_DECIS_ADAP = sum(logical(DATOS_IN') ~= DATOS_DECIS_ADAP); % Errores
BER_DECIS_ADAP=ERRORES_DECIS_ADAP/length(DATOS_IN); % Tasa de error de bit
% Solo decisor
ERRORES_SOLO_DECISOR = sum(logical(DATOS_IN') ~= DATOS_SOLO_DECISOR); % Errores
BER_SOLO_DECISOR=ERRORES_SOLO_DECISOR/length(DATOS_IN); % Tasa de error de bit
% Representación de errores
figure;
subplot(2,1,1);
stem(logical(DATOS_IN') ~= DATOS_DECIS_ADAP,'filled','markersize',3);
title('Errores en Decisor Adaptativo');
xlabel('Bits (n)');
subplot(2,1,2);
stem(logical(DATOS_IN') ~= DATOS_SOLO_DECISOR,'filled','markersize',3);
title('Errores en Decisor');
xlabel('Bits (n)');
% Representación de evolución del canal
figure
t = tiledlayout(Nt,1);
for i=1:Nt
nexttile;
stem(abs(B(i,:)),'filled','markersize',3); hold on;
yl = ylabel(sprintf('|h(n)| \nNt = %d ', i));
yl.Rotation = 0;
end
title(t,'Evolución de la respuesta al impulso del canal');
xlabel(t,'(n)');
% Representación de error cuadrático y comparativa entre 'd' e 'y'
figure;
subplot(2,1,1);
plot(e.^2);
xlabel('Muestras (n)');
ylabel('Error cuadrático');
title('Evolución del error cuadrático');
grid on;
subplot(2,1,2);
plot(y, 'b');
hold on;
plot(d_DECIS_ADAP, 'r');
xlabel('Muestras (n)');
ylabel('Amplitud');
legend('Salida del filtro', 'Decisiones');
title('Salida del filtro vs Decisiones');
grid on;
% Representación de la evolución de los coefiecientes del filtro
figure
```

```
plot(h_hist(1,:));hold on
plot(h_hist(2,:));
plot(h_hist(3,:));
plot(h_hist(4,:));
xlabel('Muestras (n)');
title('Evolución de los coeficientes del ecualizador adaptativo');
legend('h0','h1','h2','h3');
grid on;
```

Efecto del filtro de suavizado

Efectro_filtro_suavizado.m

```
% Comparativa SIN y CON filtro de suavizado
clear; close all; clc;
% Generación de señal de línea
s=1; % Potencia 1 W
Rb=1e6; % Tasa binaria 1 Mpbs
signal=1; % Señal elegida Ryc.wav
inicio=1; % Primera muestra
fin=500000; % Última muestra
Ms=16; % Muestras por símbolo
tipo=1; % Tipo de codificación de línea NRZ
Ns=1; % Elegimos pulsos rectangulares
r=0; % Roll-off (no relevante en pulsos rectangulares)
[~,x,~]=s_linea(s,Rb,signal,inicio,fin,Ms,tipo,Ns,r);
% Paso por canal multitrayecto
Pmax=15; % Número máximo de trayectos
Tmax=300; % Retardo máximo (en muestras)
sigma=0.25; % Desviación típica de la amplitud de los pulsos secundarios
Nt=300; % Número de subtramas (variaciones del canal)
rng(456);
[X2,B]=multipath(x,Pmax,Tmax,sigma,Nt); % SIN suavizado
rng(456);
[X2_fil,B_fil]=multipath_fil(x,Pmax,Tmax,sigma,Nt); % CON_suavizado
% Representación de 4 respuestas al impulso
figure;
t1 = tiledlayout(4,1,'TileSpacing','compact','Padding','compact');
for i=297:300
nexttile;
stem(abs(B(i,:)),'filled','markersize',3); hold on;
yl = ylabel(sprintf('|h(n)| \nNt = %d ', i));
yl.Rotation = 0;
end
title(t1,'SIN filtro de suavizado');
xlabel(t1,'(n)');
figure;
t2 = tiledlayout(4,1,'TileSpacing','compact','Padding','compact');
for i=297:300
nexttile;
stem(abs(B_fil(i,:)),'filled','markersize',3); hold on;
```

```
yl = ylabel(sprintf('|h(n)| \nNt = %d ', i));
yl.Rotation = 0;
end
title(t2,'CON filtro de suavizado');
xlabel(t2,'(n)');
% Correlación de respuestas consecutivas
corr_mod = zeros(Nt-1,1); % Vector de correlaciones SIN suavizado
corr_mod_fil = zeros(Nt-1,1); % Vector de correlaciones CON suavizado
for i = 1:Nt-1
mod_i = abs(B(i,:));
mod_ip1 = abs(B(i+1,:));
corr_mod(i) = corr(mod_i.', mod_ip1.');
mod_i_fil = abs(B_fil(i,:));
mod_ip1_fil = abs(B_fil(i+1,:));
corr_mod_fil(i) = corr(mod_i_fil.', mod_ip1_fil.');
end
figure;
plot(corr_mod);hold on;
plot(corr_mod_fil);
xlabel('Subtrama (Nt)');
ylabel('Coeficiente de correlación');
title('Correlación de |h(n)| entre subtramas consecutivas');
legend('SIN suavizado','CON suavizado');
```

Efecto de la modulación

Efecto_modulacion.m

```
% Repetición de ciclo de transmisión y cálculo de BER medio

clear; close all; clc;
BER_DA=zeros(1,20); % Inicialización de vector BER_DA
BER_SD=zeros(1,20); % Inicialización de vector BER_SD
for I=1:20 % Bucle para diferentes semillas
% Generación de señal de línea
s=1; % Potencia 1 W
Rb=1e6; % Tasa binaria 1 Mpbs
signal=1; % Señal elegida Ryc.wav
inicio=1; % Primera muestra
fin=500000; % Última muestra
Ms=32; % Muestras por símbolo
% Valores variables en cada simulación:
tipo=3;
Ns=8;
r=1;
[DATOS_IN,X1,~]=s_linea(s,Rb,signal,inicio,fin,Ms,tipo,Ns,r);
% Paso por canal multitrayecto
Pmax=10; % Número máximo de trayectos
Tmax=300; % Retardo máximo (en muestras)
sigma=0.35; % Desviación típica de la amplitud de los pulsos secundarios
Nt=500; % Número de subtramas (variaciones del canal)
rng(I); % Se fija la semilla en cada iteración
```

```
[X2,~]=multipath_fil(X1,Pmax,Tmax,sigma,Nt);
% Adición de ruido blanco
No=1e-7; % DEP de ruido
fm=(44100/5)*8*Ms; % Para calcular el AB de ruido
N=length(X2); % Longitud de trama de entrada para ruido_blanco()
X3=X2+ruido_blanco(No,N,fm);
% Paso por filtro adaptado y muestreo
X4=FiltroAdaptado(X3,tipo,Ms,Ns,r);
% Etapa de decisión
M=16; % Orden del filtro adaptativo
lambda=0.99; % Factor de olvido del algoritmo RLS
delta=0.01; % Para inicializar la matriz P
P=(1/delta)*eye(M); % Inicialización de matriz P
h = [1;zeros(M-1,1)]; % Inicialización de h como [1;0;...;0]
X4_ext = [zeros(1,M-1),X4]; % Se añaden M-1 valores iniciales a cero
a=[Ms,-Ms]';c=[1,0]'; % Amplitudes y códigos para decisor
% Salida del decisor (sin ecualización adaptativa)
[DATOS_SOLO_DECISOR,d_SOLO_DECISOR] = decisor(X4,a,c);
% Salida del decisor adaptativo
[DATOS_DECIS_ADAP,d_DECIS_ADAP,y,e,h_out,P_out,h_hist] = DecisorAdaptativo(X4_ext,a,c,M,lambda,h,P);
% Con decisor adaptativo
ERRORES_DECIS_ADAP = sum(logical(DATOS_IN') ~= DATOS_DECIS_ADAP); % Errores
BER_DECIS_ADAP=ERRORES_DECIS_ADAP/length(DATOS_IN); % Tasa de error de bit
% Solo decisor
ERRORES_SOLO_DECISOR = sum(logical(DATOS_IN') ~= DATOS_SOLO_DECISOR); % Errores
BER_SOLO_DECISOR=ERRORES_SOLO_DECISOR/length(DATOS_IN); % Tasa de error de bit
% Almacenamiento en vectores
BER_DA(I)= BER_DECIS_ADAP;
BER_SD(I)= BER_SOLO_DECISOR;
end
% Cálculo de medias
BER_DA_MEDIA=mean(BER_DA);
BER_SD_MEDIA=mean(BER_SD);
```

Anexo III. Códigos de las funciones de los bloques de Simulink

Multitrayecto

```
function simulink_Multipath(block)
%% Bloque de propagación multitrayecto
% Se construye matriz de respuestas al impulso para cada variación del
% canal, a partir de valores aleatorios de número de trayectos, amplitudes,
% fases y retardos.
% Cálculo de la respuesta correspondiente a cada subtrama con filter().
% Se almacena el estado final de filter entre pasos de ejecución.
%
% Level-2 MATLAB file S-Function
```

```
%
% INPUTS :
% - PORT 1: Señal de línea
% - DATA TYPE: double
% - DOMAIN : (-Inf, Inf)
% OUTPUTS :
% - PORT 1: Señal tras canal multitrayecto
% - DATA TYPE: double
% - DOMAIN : (-Inf, Inf)
% PARAMETERS:
% 1 - Pmax: Nº máximo de componentes del multitrayecto
% 2 - Tmax: Retardo máximo en nº de muestras
% 3 - sigma: Desviación típica de la amplitud de los trayectos secundarios
% 4 - Nt: Número de subtramas con características de canal diferente
setup(block);
end
%% BLOCK SET UP %%
function setup(block)
% Puertos
block.NumInputPorts = 1;
block.NumOutputPorts = 1;
% Parámetros: Pmax, Tmax, sigma, Nt
block.NumDialogPrms = 4;
block.DialogPrmsTunable = {'Nontunable', 'Nontunable', 'Nontunable', 'Nontunable'};
% Dimensiones dinámicas
block.SetPreCompInpPortInfoToDynamic;
block.SetPreCompOutPortInfoToDynamic;
% Propiedades de puertos de entrada
block.InputPort(1).DatatypeID = 0; % double
block.InputPort(1).Complexity = 'Real';
% Propiedades de puertos de salida
block.OutputPort(1).DatatypeID = 0; % double
block.OutputPort(1).Complexity = 'Real';
% Sample time
block.SampleTimes = [-1 0]; % Inherited sample time
% In Accelerator mode, the block runs on TLC.
block.SetAccelRunOnTLC(true);
% Métodos
block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
block.RegBlockMethod('PostPropagationSetup', @DoPostPropSetup);
block.RegBlockMethod('Outputs', @Output);
block.RegBlockMethod('Start', @Start);
end
%% FORWARD PROPAGATION SET UP %%
function SetInpPortDims(block, idx, di)
% Set the port dimensions for forward propagation of the dimensions.
block.InputPort(idx).Dimensions = di;
block.OutputPort(1).Dimensions = [di(1) 1];
end
%% POST PROPAGATION SET UP %%
function DoPostPropSetup(block)
Tmax = block.DialogPrm(2).Data; % Para el cálculo de dimensiones de Estado
block.NumDworks = 1;
```

```
% DWork(1): Estado final del filtro -> estado inicial para la trama posterior
block.Dwork(1).Name = 'estado';
block.Dwork(1).DatatypeID = 0;
block.Dwork(1).Complexity = 'Complex';
block.Dwork(1).UsedAsDiscState = true;
block.Dwork(1).Dimensions = Tmax-1;
end
%% VALORES INICIALES %%
function Start(block)
Tmax = block.DialogPrm(2).Data; % Para el cálculo del estado inicial
block.Dwork(1).Data = zeros(1, Tmax-1); % Valores iniciales del estado del filtro
end
%% OUTPUT %%
function Output(block)
% Datos de entrada
x = block.InputPort(1).Data;
% Parámetros
Pmax = block.DialogPrm(1).Data;
Tmax = block.DialogPrm(2).Data;
sigma = block.DialogPrm(3).Data;
Nt = block.DialogPrm(4).Data;
% Datos heredados de la trama anterior
zi = block.Dwork(1).Data; % Estado inicial para filter
Nx = length(x); % Longitud de la secuencia de entrada
T = floor(Nx/Nt); % Longitud de las Nt-1 primeras subtramas (las que son iguales)
% Vector de trayectos correspondientes a cada subtrama
P = randi([1 Pmax], Nt, 1);
% Generación de matrices tau, theta y r
tau = randi([2,Tmax],Nt,Pmax-1); % Matriz de retardos respecto al trayecto principal
theta = rand(Nt,Pmax-1)*2*pi-pi; % Matriz de fases de los impulsos secundarios
r = randn(Nt,Pmax-1)*sigma; % Matriz de amplitudes de los impulsos secundarios
r_complejo = abs(r).*exp(1j*theta); % Matriz de amplitudes y fases conjuntas
% Construcción de matriz B (coeficientes del filtro FIR) Nt x Tmax
B = zeros(Nt, Tmax); % Inicializar B con ceros
B(:,1) = 1; % Primera columna todo 1 (trayecto principal)
% Recorrido de tau y asignación de B(indice según tau)=r_complejo
for n = 1:Nt % Recorremos filas
for k = 1:(P(n)-1) % Recorremos columnas solo hasta trayectos secundarios activos
col = tau(n,k); % Retardo como índice
B(n,col) = B(n,col) + r_complejo(n,k); % Si se repite valor de tau se añade
end
end
x_mt = zeros (1,length(x)); % Inicialización de salidas
% Aplicación de filter a subtramas (de 1 a penúltima)
for i=1:Nt-1 % Esto se salta si Nt=1
x_subtrama = x(((i-1)*T+1):i*T); % Recorrido de x mediante subtramas de tamaño T
h = B(i,:); % Coeficientes del filtro correspondientes a cada subtrama
[y,zf]=filter(h,1,x_subtrama,zi); % Llamada al filtro para cada subtrama
x_mt(((i-1)*T+1):i*T) = real(y); % Resultado del filtro (parte real) a vector de salida
zi=zf; % Paso de estado para siguiente iteración
end
% Última subtrama. También es el caso de solo 1 subtrama (Nt=1)
x_subtrama = x(((Nt-1)*T+1):end); % Última subtrama (incluye resto)
```

```
h = B(Nt,:); % Última fila
[y,zf]=filter(h,1,x_subtrama,zi); % Llamada al filtro para la última subtrama
x_mt(((Nt-1)*T+1):end) = real(y); % Resultado del filtro (parte real) a vector de salida
% Asignar estado para siguiente trama
block.Dwork(1).Data = zf;
% Asignar salida
block.OutputPort(1).Data = x_mt(:); % Salida como columna
end
```

Decisor adaptativo

```
function simulink_DecisorAdaptativo(block)

%% Bloque de decisión con ecualización adaptativa
% Realiza la decisión, a partir de las muestras en instantes óptimos tras el
% filtro adaptado, utilizando ecualización adaptativa. Mediante un filtro FIR
% que adapta sus coeficientes según el algoritmo RLS. Para ello, se invoca a
% la función DecisorAdaptativo.m.
%
% Level-2 MATLAB file S-Function
%
% INPUTS :
% - PORT 1: Señal de línea
% - DATA TYPE: double
% - DOMAIN : (-Inf, Inf)
% OUTPUTS :
% - PORT 1 : Secuencia decodificada según decisiones (d_cod)
% : type: boolean | domain: (0, 1)
% - PORT 2 : Decisiones (d)
% : type: double | domain: (-Inf, Inf)
% - PORT 3 : Salida del ecualizador adaptativo (y)
% : type: double | domain: (-Inf, Inf)
% - PORT 4 : Error entre decisiones y salidas del ecualizador (e)
% : type: double | domain: (-Inf, Inf)
% PARAMETERS:
% 1 - a: Vector columna de amplitudes de la constelación (Mx1)
% 2 - c: Códigos correspondientes a cada amplitud (Mxb)
% 3 - M: Orden del filtro FIR
% 4 - lambda: Factor de olvido de algoritmo RLS
% 5 - delta: Factor de inicialización de matriz P
setup(block);
end
%% BLOCK SET UP %%
function setup(block)
% Puertos
block.NumInputPorts = 1;
block.NumOutputPorts = 4;
% Parámetros: a, c, M, lambda, delta
block.NumDialogPrms = 5;
block.DialogPrmsTunable = {'Nontunable', 'Nontunable', 'Nontunable', 'Nontunable', 'Nontunable'};
% Dimensiones dinámicas
block.SetPreCompInpPortInfoToDynamic;
```

```
block.SetPreCompOutPortInfoToDynamic;
% Propiedades de puertos de entrada
block.InputPort(1).DatatypeID = 0; % DOUBLE
block.InputPort(1).Complexity = 'Real';
block.InputPort(1).SamplingMode = 'Sample';
% Propiedades de puertos de salida
block.OutputPort(1).DatatypeID = 8; % Decisiones codificadas (d_cod) BOOLEAN
block.OutputPort(1).Complexity = 'Real';
block.OutputPort(1).SamplingMode = 'Sample';
block.OutputPort(2).DatatypeID = 0; % Decisiones (d) DOUBLE
block.OutputPort(2).Complexity = 'Real';
block.OutputPort(2).SamplingMode = 'Sample';
block.OutputPort(3).DatatypeID = 0; % Salida del ecualizador adaptativo (y) DOUBLE
block.OutputPort(3).Complexity = 'Real';
block.OutputPort(3).SamplingMode = 'Sample';
block.OutputPort(4).DatatypeID = 0; % Errores (e = d - y) DOUBLE
block.OutputPort(4).Complexity = 'Real';
block.OutputPort(4).SamplingMode = 'Sample';
% Sample time
block.SampleTimes = [-1 0]; % Inherited sample time
% In Accelerator mode, the block runs on TLC.
block.SetAccelRunOnTLC(true);
% Métodos
block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
block.RegBlockMethod('PostPropagationSetup', @DoPostPropSetup);
block.RegBlockMethod('Outputs', @Output);
block.RegBlockMethod('Start', @Start);
end
%% FORWARD PROPAGATION SET UP %%
function SetInpPortDims(block, idx, di) % SET FORWARD PORT DIMENSIONS
% Set the port dimensions for forward propagation of the dimensions.
OrdenMod = numel(block.DialogPrm(1).Data); % Orden de la modulación
b=log2(OrdenMod); % Bits por símbolo
block.InputPort(idx).Dimensions = di;
block.OutputPort(1).Dimensions = [di(1)*b 1];
block.OutputPort(2).Dimensions = [di(1) 1];
block.OutputPort(3).Dimensions = [di(1) 1];
block.OutputPort(4).Dimensions = [di(1) 1];
end
%% POST PROPAGATION SET UP %%
function DoPostPropSetup(block)
M = block.DialogPrm(3).Data; % Orden del filtro ecualizador
block.NumDworks = 3;
% DWork(1): Solape para la trama posterior
block.Dwork(1).Name = 'solape';
block.Dwork(1).DatatypeID = 0;
block.Dwork(1).Complexity = 'Real';
block.Dwork(1).UsedAsDiscState = true;
block.Dwork(1).Dimensions = M-1;
% DWork(2): Coeficientes del filtro ecualizador para trama posterior
block.Dwork(2).Name = 'h';
block.Dwork(2).DatatypeID = 0;
block.Dwork(2).Complexity = 'Real';
```

```
block.Dwork(2).UsedAsDiscState = true;
block.Dwork(2).Dimensions = M;
% DWork(3): Matriz P para trama posterior
block.Dwork(3).Name = 'P';
block.Dwork(3).DatatypeID = 0;
block.Dwork(3).Complexity = 'Real';
block.Dwork(3).UsedAsDiscState = true;
block.Dwork(3).Dimensions = M*M;
end
%% VALORES INICIALES %%
function Start(block)
M = block.DialogPrm(3).Data; % Orden del filtro ecualizador
delta = block.DialogPrm(5).Data; % Factor de inicialización de P
% Inicialización del solape con ceros (para la primera trama)
block.Dwork(1).Data = zeros(M-1, 1);
% Inicialización de h con para la primera trama
block.Dwork(2).Data = zeros(M, 1);
%block.Dwork(2).Data = [1;zeros(M-1, 1)];
% Inicialización de P con delta (para la primera trama)
block.Dwork(3).Data = reshape((1/delta) * eye(M), [], 1);
end
%% OUTPUT %%
function Output(block)
% Datos de entrada
inpData = block.InputPort(1).Data;
% Parámetros
a = block.DialogPrm(1).Data;
c = block.DialogPrm(2).Data;
M = block.DialogPrm(3).Data;
lambda = block.DialogPrm(4).Data;
% Datos heredados de la trama anterior
solape = block.Dwork(1).Data;
h = block.Dwork(2).Data;
P = reshape(block.Dwork(3).Data, [M, M]);
% Concatenación de solape heredado con entrada
xm = vertcat(solape, inpData);
% Llamada a la función DecisorAdaptativo()
[d_cod,d,y,e,h_out,P_out,~]= DecisorAdaptativo(xm,a,c,M,lambda,h,P);
% Actualización de datos para la siguiente llamada
block.Dwork(1).Data = xm(end-M+2:end); % Final de trama, últimos M-1
block.Dwork(2).Data = h_out; % Último valor de h
block.Dwork(3).Data = reshape(P_out, [], 1); % Matriz P redimensionada
% Asignar a la salida
block.OutputPort(1).Data = logical(d_cod(:)); % Conversión a boolean
block.OutputPort(2).Data = d(:); % Decisiones
block.OutputPort(3).Data = y(:); % Salida del filtro FIR
block.OutputPort(4).Data = e(:); % Error (d-y)
end
```

AUTORIZACIÓN DE DIFUSIÓN

El/la abajo firmante, matriculado/a en el Grado de Ingeniería en Tecnologías y Servicios de Telecomunicación, autorizará a la Universidad a Distancia de Madrid (UDIMA) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado titulado: "Módulo de ecualizador adaptativo para mitigar el efecto de la propagación multitrayecto para el laboratorio virtual de comunicaciones comLAB", realizado durante el curso académico 2024-2025 bajo la dirección de José María Díaz Nafría y Antonio Jesús Muñoz Montoro en el Departamento de Ingeniería de Telecomunicaciones, y a la Biblioteca de la UDIMA a depositarlo en el Archivo Institucional con el objeto de incrementar la difusión, uso e impacto del trabajo y garantizar su preservación y acceso a largo plazo.

DECLARACIÓN DE ORIGINALIDAD

El autor de este trabajo, Rubén Guzmán Serrano, con D.N.I. 28996007E, declara que el contenido de este trabajo de fin de grado es original, y en el caso de haber utilizado las ideas o contenidos de otros trabajos de otros autores están convenientemente citados, de acuerdo con las normas de citación establecidas.

FIRMA DEL AUTOR DEL TRABAJO Y FECHA

Firma: Rubén Guzmán Serrano,

En Denia, a 17 de septiembre 2025