Capítulo 4: Análisis de los algoritmos estudiados

4.1 Introducción

A lo largo del presente documento se han ido estableciendo las bases que constituyen este proyecto. Se ha dado una idea del funcionamiento de la modulación OFDM, explicando sus ventajas e inconvenientes. Se ha explicado el por qué de su importancia en el mercado actual de telecomunicaciones, dando así razones para la elaboración de este proyecto. A continuación se explicó como mejorar su rendimiento usando técnicas de estimación de canal, se dio una perspectiva de las mismas y se explicó en que consistían los algoritmos ciegos de estimación. Por último se han explicado 4 algoritmos ciegos de estimación de canal a nivel matemático para introducir al lector en materia antes de proceder a su análisis.

Una vez definido el contexto en el que se va centrar el proyecto y documentado al lector sobre el mismo, procederemos a la exposición de la realización del mismo. Para implementar y testar los algoritmos de estimación de canal se ha elegido la plataforma de simulación MATLAB. Se

trata de un programa de simulación matemática en el que mediante variables, ecuaciones y bucles se han desarrollado los algoritmos deseados. Básicamente nuestro esquema de simulación consiste en implementar el esquema de transmisión básica de OFDM, empezando con los bits a transmitir y terminando con los bits recibidos. Luego, sobre este sistema, implementaremos los algoritmos ciegos de estimación de canal, obteniendo los estimadores de canal. Mediante estos últimos y el canal original se ha calculará el *NMSE* o error cuadrático medio normalizado, que será el punto de partida para evaluar los algoritmos. También es objeto de este capítulo describir y explicar los códigos de MATLAB creados para llevar a cabo estas simulaciones. Para clarificar la exposición de las simulaciones realizadas hemos estructurado este capítulo de la siguiente forma:

- Sistema de comunicación OFDM: dada la información previa dada sobre la modulación OFDM, acotaremos su definición de forma que nos permita establecer las bases teóricas para programar el transmisor, el canal y el receptor de forma eficiente. Esto nos servirá para poner a prueba nuestros algoritmos de estimación de canal.

- Implementación y análisis de los algoritmos: se procederá a la explicación de cómo se han desarrollado los algoritmos en el programa, así como una exposición de los resultados de los análisis a los que se han sometido los mismos.

- Comparativa entre algoritmos: Una vez sometidos los algoritmos a todo tipo de pruebas, se optimizarán sus parámetros simulados y se compararán entre ellos atendiendo a criterios como convergencia, coste computacional o error cuadrático medio normalizado.

4.2 Sistema de comunicaciones OFDM

Para llevar a cabo nuestro estudio, en primer lugar necesitaremos implementar un sistema de telecomunicaciones basado en la modulación OFDM. Partiremos del modelo propuesto en el capítulo 2, no obstante realizaremos una serie de modificaciones que simplificarán el sistema, aunque permaneciendo válido para nuestros objetivos. Plantearemos un modelo discreto de la

modulación. El esquema de nuestro sistema propuesto se puede ver en la figura 4.1.

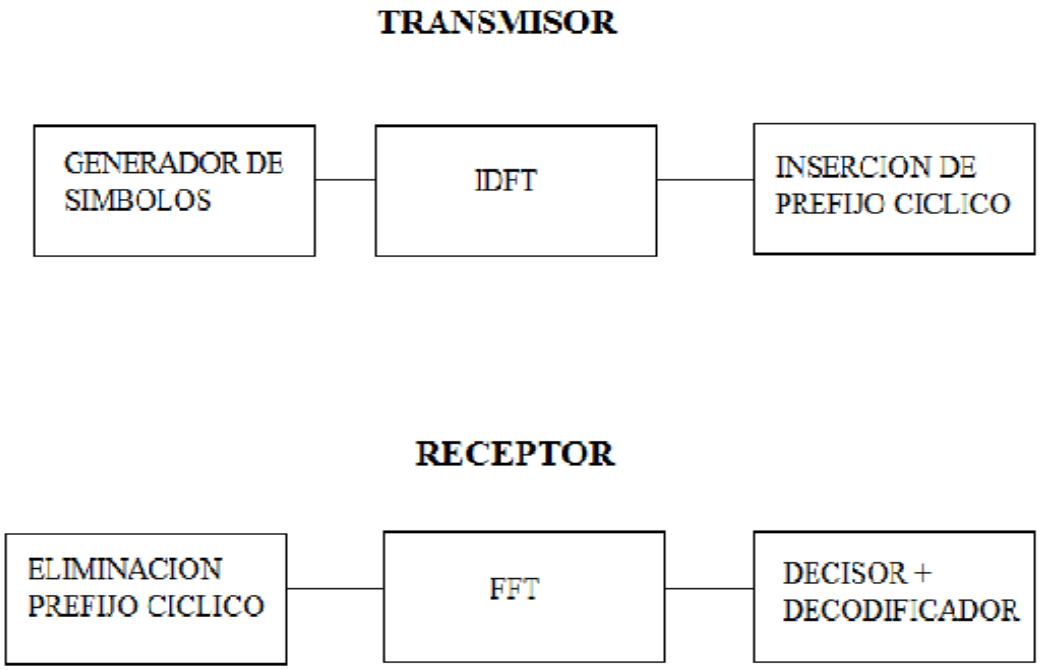

Figura 4.1 Modelo en tiempo discreto del sistema OFDM implementado

Nuestro sistema se ha organizado en dos funciones diferentes, transmisor y receptor. La primera de ellas, incluye desde la generación de bits, hasta la inclusión del prefijo cíclico. La función receptor incluye también las funciones de canal, esto es, aplica el canal a la señal y añade el ruido blanco gausiano pertinente. Tras esto, realiza las funciones de recepción de la señal y extracción de los bits de información. Como se puede observar el esquema de la modulación OFDM en sí es bastante simple, se limita al tratamiento matemático de los símbolos mediante la transformación DFT y la inserción y extracción del prefijo cíclico. Veamos como funcionan las dos funciones programadas en MATLAB detalladamente.

4.2.1 Transmisor

La estructura general de la función se constituye por una primera etapa donde se definen los

parámetros a usar, algunos de dichos parámetros se usarán posteriormente para testar los algoritmos. A continuación se encuentra el generador de símbolos, compuesto por un generador aleatorio de bits, seguido de su correspondiente modulador en amplitud. Por último tenemos la etapa de aplicación de la IDFT y la inserción del prefijo cíclico, ambos se han situado en la misma etapa debido a que van incluidos en el mismo bucle.

- Parámetros:

En esta primera parte se han definido una serie de parámetros predefinidos que regularán la modulación OFDM y el sistema de telecomunicaciones en general. Dichos parámetros son:

```
%Parámetros

N=64;                   %numero de subportadoras y tamaño de la FFT
M=16;                   %tamaño de la constelación
t=50;                   %numero de bloques a transmitir
delta=21;               %tamaño del prefijo cíclico y del canal de transmisión
Ns=delta+N;             %Muestras total de una secuencia de odfm
snr=20;                 %relación señal a ruido deseada en el canal gausiano
a=0.25;                 %índice de autocorrelación
```

Por supuesto el valor que se les ha dado a estos parámetros es sólo inicial, ya que se variarán posteriormente para las simulaciones. Se han realizado algunas simplificaciones para facilitar los cálculos. En primer lugar se ha tomado el número de portadoras a usar como potencia de dos de manera que coincida con el tamaño de la FFT a usar, cuando normalmente se toma un número de subportadoras inferior al tamaño de la FFT aplicada y se realiza zero-padding, pues bien con esto se evita tener que rellenar la secuencia y eliminar los efectos del relleno en el receptor.

Por otra parte se ha escogido un tamaño de prefijo cíclico y de longitud de canal iguales, de manera que la convolución circular que se produce al atravesar el canal sea perfecta, sin ninguna alteración. Realmente esto es irrealizable porque en los sistemas reales no se sabe a priori la

longitud del canal y por lo tanto es imposible ajustar el prefijo cíclico a conveniencia. En realidad se toma un prefijo generalmente mayor a la longitud del canal y luego se ajusta la secuencia al llegar al receptor. Dichas simplificaciones no afectan en ninguna manera a los resultados que se han obtenido, pero si han simplificado mucho el proceso permitiendo acelerarlo.

– Generador de símbolos:

Una vez definido los parámetros del sistema, se comienza simulando una fuente de información aleatoria, en primer lugar se generan una serie de bits con igual probabilidad y luego se codifican según el tamaño de la constelación adoptada. No obstante MATLAB incorpora una serie de funciones que facilitan dicha generación:

```
%Generador de símbolos

simb=randint(N*t,1,M);                    %generación de los símbolos
data=qammod(simb,M,[],'gray');            %codificación de los símbolos
```

El primer comando genera los símbolos con igual probabilidad usando representantes numéricos de 0 hasta *M*-1, sin ningún valor matemático. Como se puede observar se generan todos los símbolos que se van a transmitir durante la simulación, *N* símbolos por *t* bloques. El segundo comando codifica los símbolos anteriores a símbolos QAM siguiendo una distribución Gray. De esta manera ya tenemos la secuencia total de símbolos codificados según una constelación QAM de *M* puntos que se usará en nuestra simulación.

– Transformada IDFT e inserción del prefijo cíclico:

Antes de realizar estas operaciones es necesaria una división en bloques OFDM de la secuencia anteriormente generada. Esto se consigue mediante un bucle programado para que tome elementos de la cadena anterior de *N* en *N* elementos, de forma que constituyan cada uno un bloque:

```
%Generación de subportadoras e inserción del prefijo cíclico

for i=1:t

    port=ifft(data,N);                          %aplicación de ifft a cada bloque
    port_pc=[port(N-(delta-1):N);port];         %adición del prefijo cíclico
    s((i-1)*Ns+1:Ns*i)=port_pc;                 %reconstrucción de la trama

end
```

Una vez realizada la conversión el proceso es muy simple, se aplica el algoritmo IFFT de tamaño *N* a cada bloque realizando la modulación OFDM, a continuación se copian las *delta* últimas muestras del bloque y se añaden al principio de este. Por último se vuelve a reconstruir la trama en una secuencia única modulada en OFDM, dicha secuencia está lista para transmitirse.

4.2.2 Receptor

En esta función se han implementado el canal y el receptor, tal y como se muestra en la figura 4.1. El canal está formado por la convolución entre la señal transmitida y el canal de transmisión y la adición del ruido gausiano. Por otro lado en el receptor se encuentran las funciones pertinentes para llevar a cabo los bloques definidos. En primer lugar se procede a eliminar el término redundante resultante de la convolución cíclica, este viene provocado por la inserción del prefijo cíclico en el transmisor. A continuación se demodula la señal OFDM aplicando el algoritmo FFT extrayendo así los símbolos de las portadoras. Los símbolos resultantes están afectados por el canal y el ruido, de manera que es necesaria una última etapa de detección de los mismos estimando los símbolos recibidos. Por último se procede a la decodificación de los mismos obteniendo así la información introducida en el transmisor.

– Canal:

En esta parte se han simulado los efectos que sufre la señal debido al canal, por una parte la convolución con el mismo y por otra la adición del ruido, se ha implementado de la siguiente forma:

```
%Aplicación del canal

y=conv(s,h);        %convolución de la señal y el canal en el domino del tiempo
y_r=awgn(y,snr,'measured');   %aplicación del ruido
H=fft(h,N);                    %paso del canal al dominio de la frecuencia
```

El primer comando realiza la función de convolución entre la señal y el canal en el dominio del tiempo, **h** en el programa, el segundo le añade el ruido para una *SNR* dada, por último se obtiene el canal en el dominio de la frecuencia para posteriores usos.

Este aspecto del modelo es muy importante debido a que el canal es el objeto de análisis de este proyecto, por lo cuál se ha escogido el modelo cuidadosamente para evitar posibles desviaciones de los resultados de este proyecto. Para modelar el canal se ha usado una función provista por MATLAB llamada stdchan, dicha función genera canales aleatorios en el dominio del tiempo partiendo de una serie de parámetros. En nuestro caso se ha optado por usar un modelo de canal 802.11a, con un período de muestreo de 50ns y un τ_{rms} de 100ns. Teniendo en cuenta el carácter aleatorio de los canales se ha generado una batería de 100 canales diferentes con los mismos parámetros para así promediar los resultados, pudiendo así realizar una comparación entre algoritmos más fiable.

– Extracción del prefijo cíclico y supresión de portadoras:

Al igual que ocurría en el transmisor, estas dos funciones se han unido en un bucle con objeto de su aplicación individual a cada bloque:

```
for i=1:t

%separación de la trama en bloques

y_s=y_r((i-1)*Ns+1:i*Ns);

%eliminación del termino redundante

r_s=y_s(delta+1:Ns);

%supresión de portadoras

r_info=fft(r_s,N);

end
```

Se extrae cada bloque y se le aplican las dos operaciones, la extracción de las *delta* muestras sobrantes y la FFT que elimina las portadoras de nuestra señal, dejando los símbolos recibidos.

– Detección y decodificación:

Por último se realizan las funciones de detección y decodificación de los símbolos, de esta operación obtenemos los símbolos estimados que se generaron al principio. Estas dos funciones se realizan mediante un solo comando de MATLAB:

```
%detector y decodificador

r_simb=qamdemod(r_info,M,[],'gray');
```

Así queda definido el sistema de telecomunicaciones sobre el cuál aplicaremos los algoritmos de estimación de canal, por supuesto los parámetros del mismo son totalmente susceptibles de cambio, de hecho es el objetivo de este proyecto. Una vez implementemos los algoritmos sobre este esquema, se procederá a variar los parámetros del mismo para observar como

afectan al rendimiento del algoritmo.

4.3 Implementación y análisis de los algoritmos

Una vez realizado el sistema de telecomunicación, el siguiente paso es implementar los ya descritos algoritmos de estimación de canal y comprobar su efectividad sobre nuestro sistema.

4.3.1 Consideraciones previas

Comenzaremos detallando el funcionamiento del código realizado para la simulación de estos. Se explicará como se han interpretado las ecuaciones de estos y como se han implementado mediante lineas de código. Existen varios aspectos comunes a los códigos realizados para todos los algoritmos, procederemos a comentarlos en primer lugar.

- Autocorrelación:

Existe un cálculo común en el funcionamiento de cada algoritmo, del que se basan para hacer la estimación posterior, se trata un proceso de autocorrelación de las señales recibidas. En la práctica se realiza el cálculo de la autocorrelación mediante un proceso adaptativo que según la ecuación 3.3. Este proceso permite promediar las estimaciones del canal realizadas en cada momento, propiciará que una vez procesados cierto número de bloques OFDM la estimación del canal se haga cada vez más precisa convergiendo esta hacia la estimación más precisa que el algoritmo en cuestión sea capaz de realizar. La línea de código que permite realizar esta operación es:

```
R=(1-a)*R+a*r_info*r_info';              %calculo de la autocorrelación
```

– Constante α:

Los algoritmos sirven para ofrecer una estimación del canal a falta de una constante, por lo que dada una estimación del canal $\overline{\mathbf{H}}$, la relación con el canal original cumpliría:

$$\mathbf{H} = \alpha \bar{\mathbf{H}} \tag{4.1}$$

Donde α es una constante compleja. Existen diversos métodos establecidos para la estimación de dicha constante, no obstante el error introducido por dicha estimación se añadiría al introducido por los algoritmos de estimación de canal perjudicando los resultados, por lo tanto no es objeto de este proyecto. Para calcular la constante de forma exacta la deduciremos matemáticamente dando por supuesto que conocemos el canal de transmisión real a priori, aunque sabemos que es imposible nos tomaremos esta libertad con objeto de analizar los algoritmos de la forma más objetiva posible.

Para la deducción de la constante partiremos de la ecuación 4.1, nuestro objetivo es conseguir una α que cumpla dicha ecuación. Por supuesto $\mathbf{H}$ y $\overline{\mathbf{H}}$ son vectores, y en nuestro caso $\overline{\mathbf{H}}$ va a contener cierto ruido, por lo cuál será imposible encontrar una constante que cumpla exactamente la relación. Así que lo que tendremos que buscar es el valor de α que cumpla con mayor fidelidad la ecuación 4.1, o de otra forma, un valor de α que minimice:

$$\mathbf{J} = \mathbf{H} - \alpha \bar{\mathbf{H}} \tag{4.2}$$

Para resolver esto, dado que se trata de matrices, en primer lugar lo multiplicaremos por su hermítico para manejar escalares:

$$\mathbf{J}^{\mathbf{H}}\mathbf{J} = \left[\mathbf{H} - \alpha \bar{\mathbf{H}}\right]^{\mathrm{H}}\left[\mathbf{H} - \alpha \bar{\mathbf{H}}\right] = \mathbf{H}^{\mathbf{H}}\mathbf{H} - \alpha \bar{\mathbf{H}}^{\mathbf{H}}\mathbf{H} - \alpha^{H}\mathbf{H}^{\mathbf{H}}\bar{\mathbf{H}} + |\alpha|^{2}\bar{\mathbf{H}}^{\mathbf{H}}\bar{\mathbf{H}} \tag{4.3}$$

A partir de ahí se trata de minimizar una función cualquiera, por lo que derivaremos respecto

de α e igualaremos a cero:

$$\frac{\delta \mathbf{J}^{\mathbf{H}}\mathbf{J}}{\delta \alpha} = -\mathbf{H}^{\mathbf{H}}\bar{\mathbf{H}} + \alpha^{H}\bar{\mathbf{H}}^{\mathbf{H}}\bar{\mathbf{H}} = 0 \qquad 4.4$$

Ya sólo nos queda despejar:

$$\alpha = \frac{\boldsymbol{H}^{\boldsymbol{H}}\bar{\boldsymbol{H}}}{\bar{\boldsymbol{H}}^{\boldsymbol{H}}\bar{\boldsymbol{H}}} \qquad 4.5$$

Esta ecuación se implementa en el código mediante las siguientes líneas:

```
alf=(H2'*H)/(H2'*H2);                    %calculo de la constante alfa
H3=alf*H2;                               %estimación del canal
```

– Cálculo del error cometido:

Una vez disponemos de nuestra estimación completa del canal, ahora sólo falta calcular el error que hemos cometido, para ello usaremos el estimador *NMSE* según la ecuación 3.4:

```
e=H-H3;
NMSE(i)=norm(e)^2/norm(H)^2;             %estimación del error
```

– Simulaciones realizadas:

A la hora de describir los códigos de los algoritmos utilizados nos quedaremos en la estimación del canal a falta de la constante, ya que el resto del proceso ya se ha explicado y es siempre el mismo.

Tras la exposición del código se expondrán los resultados derivados de los análisis realizados. Para estos se ha tenido en cuenta el carácter aleatorio de diversos elementos de nuestro sistema de comunicación. Para evitar que estos interfieran con posibles anomalías poco probables,

se han realizado varias simulaciones para cada uno de estos elementos y se han promediado sus resultados.

En primer lugar tenemos el canal de transmisión, se ha optado por generar una batería de 100 canales diferentes con objeto de realizar las simulaciones para cada uno de ellos para realizar luego un promedio de los resultados. Otro elemento aleatorio importante es el generador de símbolos y el de ruido blanco gausiano. Nuestro objetivo es poder caracterizar los algoritmos de la forma más objetiva posible. Por lo cuál debemos evitar un posible patrón poco probable de símbolos que afecte a los resultados e incluso una secuencia de ruido que sea excesivamente perjudicial o demasiado favorable para la estimación del canal. Es por esto que los resultados se promediaran 50 veces, utilizando diferentes secuencias de símbolos y ruido aleatorias. Hay que tener en cuenta que este proyecto debe evaluar los algoritmos de estimación de canal de la forma más objetiva posible.

También es necesario dar un espacio conveniente de tiempo de simulación para que el algoritmo converja a su valor mínimo, observando así su comportamiento a largo plazo. El tiempo en nuestro sistema se traduce en número de bloques transmitidos. Este aspecto de la simulación es muy importante ya que nos permitirá ver la evolución de la *NMSE* en función de los bloques que se van transmitiendo. Para nuestro caso tomaremos un valor base de 50 bloques, aumentándolo si fuera necesario en el caso de que nos topásemos con un algoritmo más lento de la cuenta. El error cometido es un parámetro muy importante de estas simulaciones, pero también lo es casi con la misma importancia el tiempo que tardan los resultados en converger a su valor mínimo. Para medir este parámetro consideraremos el número de bloques recibidos tal que el *NMSE* alcanza su valor mínimo más un 10% del mismo.

Otro punto de vista importante a la hora de la utilización de los algoritmos es una evaluación previa de su coste computacional, para ello se dispondrá de una herramienta del simulador que calcula el tiempo que un algoritmo ocupa el procesador. En nuestro caso sólo disponemos del código del sistema completo, por lo que en primer lugar observaremos el coste del código del sistema original y lo compararemos con el de cada algoritmo, de esta forma veremos el coste

adicional introducido por el algoritmo. También hay que tener en cuenta que los parámetros del sistema han de ser los mismos para todos los algoritmos, de forma que no exista discrepancia sobre la cantidad de operaciones que realiza cada uno. Por lo tanto dispondremos de un método objetivo de evaluación del coste computacional de los algoritmos en cuestión.

Los parámetros que influyen en el tiempo de proceso se fijarán para la estimación del coste:

```
N=4;                    %numero de subportadoras y tamaño de la FFT
M=4;                    %tamaño de la constelación
t=1;                    %numero de bloques a transmitir
delta=3;                %tamaño del prefijo cíclico y del canal de transmisión
```

El tiempo transcurrido es de 5.842ms, ahora sólo será necesario compararlo con cada algoritmo para observar el peso de estos.

4.3.2 Algoritmo nº1, de Bangwon Seo y Hyun Gyu Chung

- **Código**:

Tal y como se indica en la descripción del mismo, en primer lugar se genera la matriz precodificadora tal y como se indican en las ecuaciones 3.6 y 3.7:

```
C=c*ones(N)+b*eye(N);           %matriz precodificadora
```

Donde *b* y *c* se tomarán reales y distintos de cero. Esta línea de código crea una matriz de tamaño *NxN* en la que todos sus elementos tienen valor *c* excepto los de la diagonal, cuyo valor es *c+b*. De esta forma nos aseguramos que las columnas de la misma sean no ortogonales entre sí y el módulo de las mismas tenga siempre el mismo valor. Por supuesto existe un rango mucho más variado de matrices que cumplen los requisitos anteriormente descritos, no obstante variando los valores b y c lograremos un elenco variado de matrices para testar de una forma rápida y sencilla.

El siguiente paso es aplicar la matriz a los datos generados después de realizar la codificación QAM, tal y como se muestra en la ecuación 3.5:

```
data_c=C'*data(i:i+N-1);
```

Una vez codificados los datos, el transmisor OFDM los transmite de forma totalmente transparente a la precodificación. Ya en el receptor se recibe la señal y se extraen los símbolos de esta, es ahí donde procede el proceso de estimación del canal. Este comienza con el cálculo de la autocorrelación de los símbolos extraídos de las portadoras recibidas. Ahora, aplicamos el algoritmo propuesto en el artículo, para ello proseguiremos calculando la matriz de autocorrelación modificada según la ecuación 3.13, posteriormente calcularemos los autovectores de dicha matriz y escogeremos el principal, como se indica en la ecuación 3.14:

```
A=C'*C;
Q=R./A;                    %calculo de la matriz de autocorrelación modificada
[V,D]=eig(Q);              %calculo de los autovectores de la matriz
H2=V(:,N);                 %extracción del autovector principal
```

El autovector principal es este que corresponde al mayor autovalor, como se puede ver en el código, el comando eig de MATLAB ordena los autovectores de menor a mayor según sus autovalores. El vector H2 corresponde a nuestra estimación de canal a falta sólo de una constante.

- **Análisis**:

El siguiente paso una vez completado el código es variar una serie de parámetros para ver la influencia de los mismos sobre los resultados del algoritmo. Para ello se ha diseñado una función que permite realizar las simulaciones oportunas variando los parámetros. En primer lugar estableceremos los parámetros a variar dándoles un valor base en función a las recomendaciones dadas por el artículo original, nuestro objetivo será desviarnos de esas recomendaciones observando

como afectan estas a la eficiencia del algoritmo:

```
b=1;                    %parámetro 1 de la matriz precodificadora
c=0.5;                  %parámetro 2 de la matriz precodificadora
snr=20;                 %relación señal a ruido
N=64;                   %número de portadoras a utilizar
M=4;                    %tamaño de la constelación
a=0.25;                 %factor de memoria de la autocorrelacion
```

Dados estos valores, observaremos como responde el algoritmo si se transmiten 50 bloques OFDM:

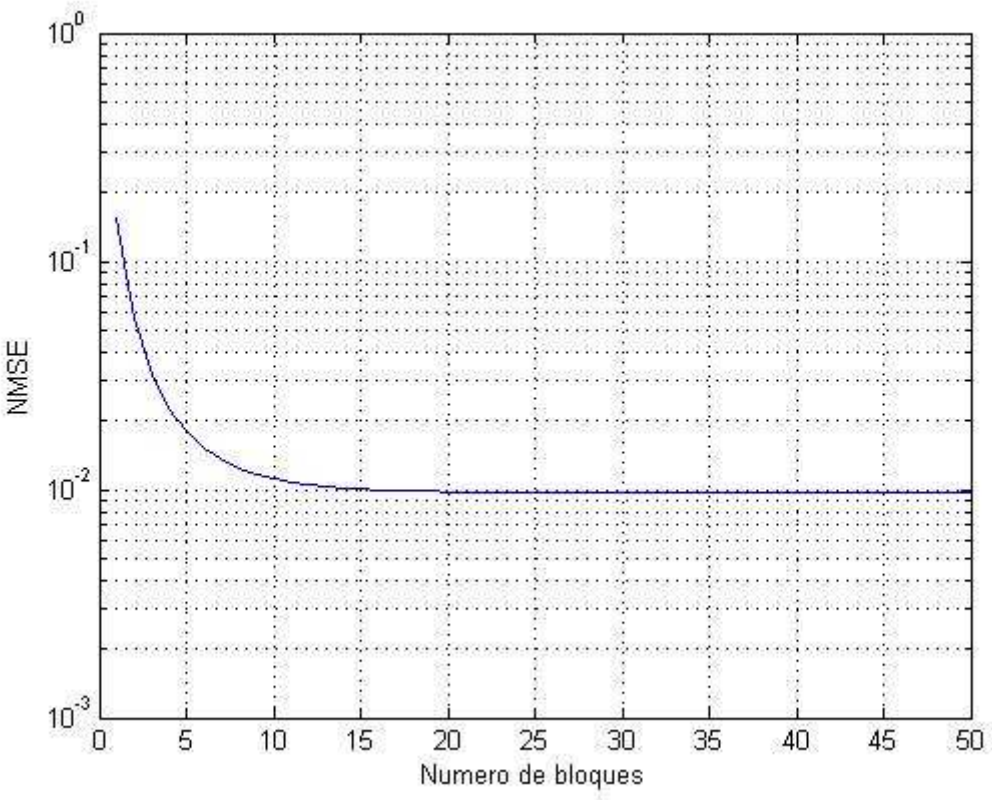

Figura 4.2 NMSE para el algoritmo nº1

Como se puede observar en la figura 4.2, el algoritmo da un *NMSE* de 0.01 a partir del bloque número quince. Esto se traduce en que una vez pasado un tiempo transitorio el error con el

que este algoritmo estimará el canal será del 1%. En los siguientes apartados veremos como responde el algoritmo a la variación de los parámetros anteriormente expuestos.

– Relación señal a ruido, SNR:

En este apartado observaremos la dependencia del algoritmo respecto a la *SNR* existente en el sistema. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintas *SNR's*, se ha simulado con valores entre SNR=0 dB y SNR=30dB con un paso de 2dB.

En la figura 4.3 podemos ver como una *SNR* baja puede distorsionar en gran medida nuestra estimación del canal, llegando hasta errores del 10% en régimen permanente, no obstante conforme aumentamos la *SNR* desde posiciones bajas, el *NMSE* se hace cada vez más pequeña obteniendo una relación casi lineal entre ambas. Dicha relación se cumple hasta los 20 dB, a partir del cuál el *NMSE* se vuelve menos dependiente de la *SNR* tendiendo a ser constante, se puede observar que la variación entre 20 dB y 30 dB es muy pequeña.

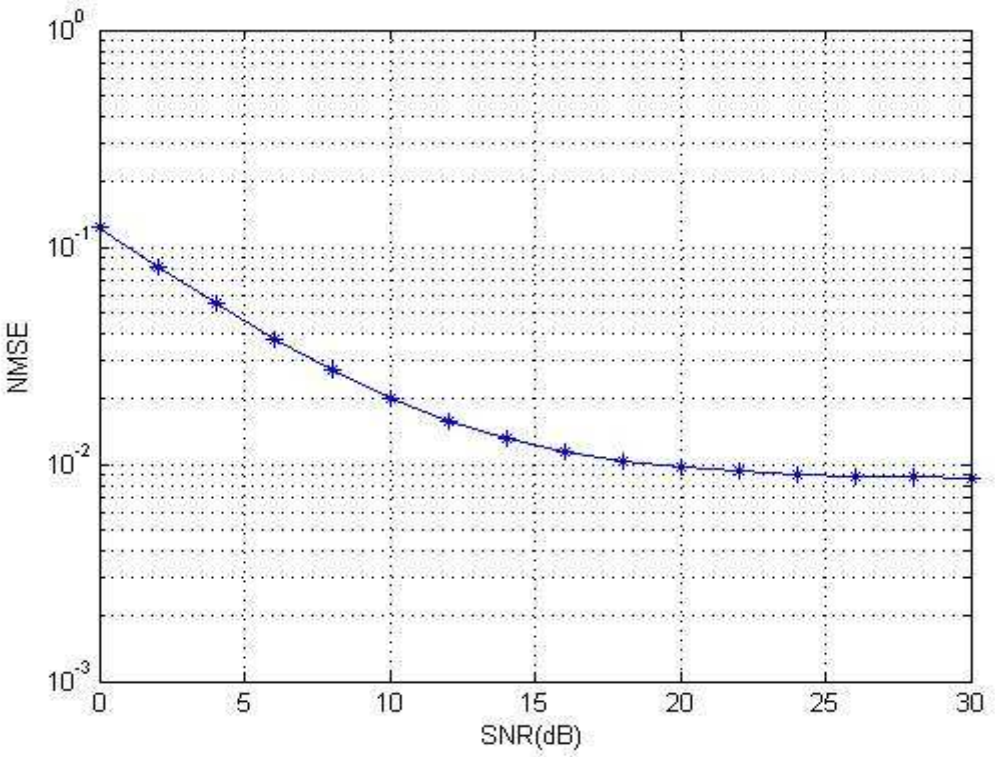

Figura 4.3 NMSE del algoritmo nº1 con respecto a la SNR

En la figura 4.4 se representan la adaptación del algoritmo para distintas *SNR's*, aquí se confirma lo que se veía en la figura anterior, como partiendo desde cero, pequeños aumentos de la *SNR* provocan grandes disminuciones del *NMSE*, hasta llegar a los 20 dB donde se tiende al mismo valor. Esta figura también aporta otro punto de vista, como el tiempo de convergencia del sistema no se ve afectado en absoluto por la relación señal a ruido, tal y como se estableció al principio el algoritmo pasa a régimen permanente una vez transmitidos 15 bloques.

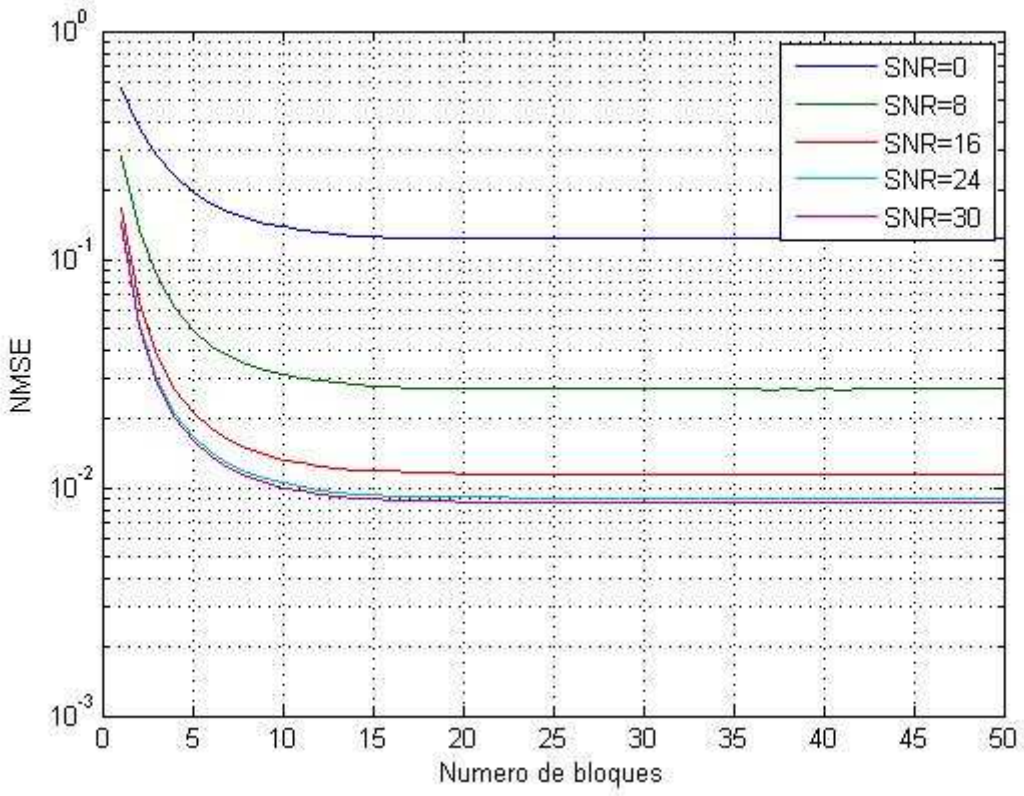

Figura 4.4 NMSE del algoritmo nº1 para distintas SNR

– Matriz de precodificación:

Otro elemento importante dentro de nuestro sistema es la matriz de precodificación del algoritmo. Donde los elementos c_{ij} de la matriz se regían por la siguiente ecuación:

$$\begin{aligned} c_{ij} &= c \quad para\, i \neq j \\ c_{ij} &= c + b \quad para\, i = j \end{aligned} \tag{4.6}$$

En este apartado observaremos la dependencia del algoritmo respecto a la matriz de precodificación del sistema. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintos valores de *b* y *c*. Se ha simulado con valores de *b* entre *b*=-4 y *b*=4, con un paso de 0.5, para el caso de *c* entre *c*=-4.6 y *c*=4.4, con un paso de 0.5 también.

Las figuras 4.5 y 4.6 muestran los resultados para el parámetro *c*. Podemos observar como mejoran los resultados exponencialmente conforme nos alejamos de *c*=0. Esto es, para *c*=0 son los peores que nos podemos encontrar, del orden de *NMSE*=0.3, mientras que si nos alejamos por cualquier lado, dada la simetría de la figura, los resultados tienden a mejorar llegando a un régimen permanente en el que estos se estabilizan tendiendo a *NMSE*=10^{-3}. También es apreciable como este parámetro no afecta al tiempo de régimen transitorio del algoritmo.

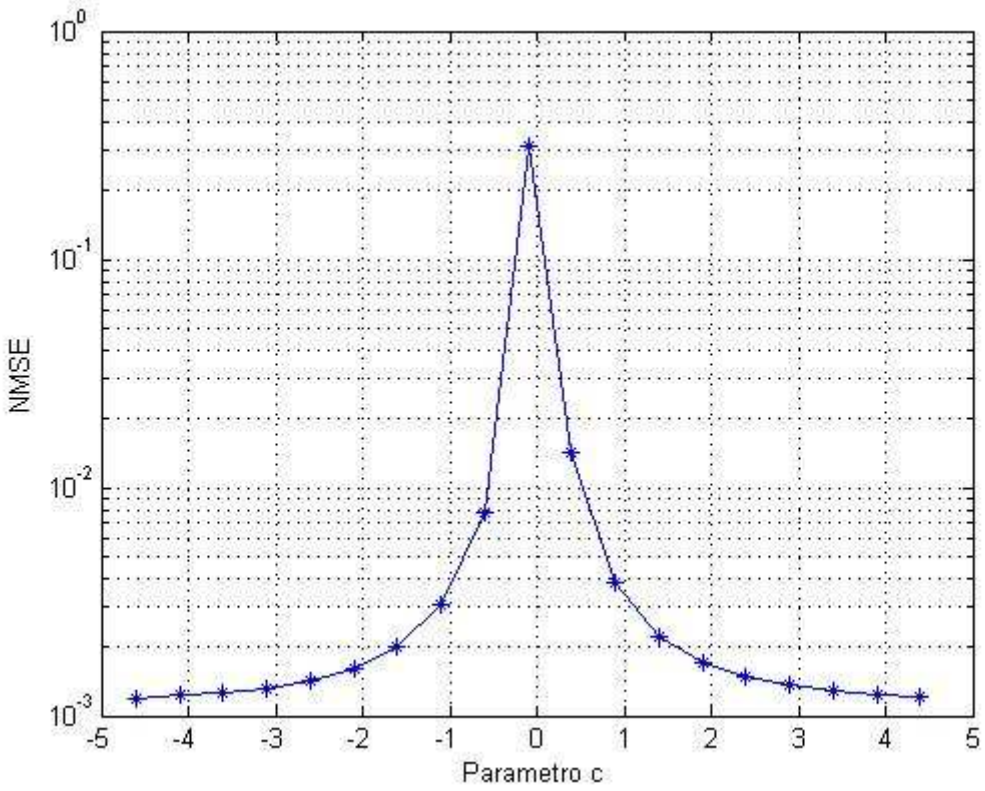

Figura 4.5 NMSE del algoritmo nº1 con respecto al parámetro c

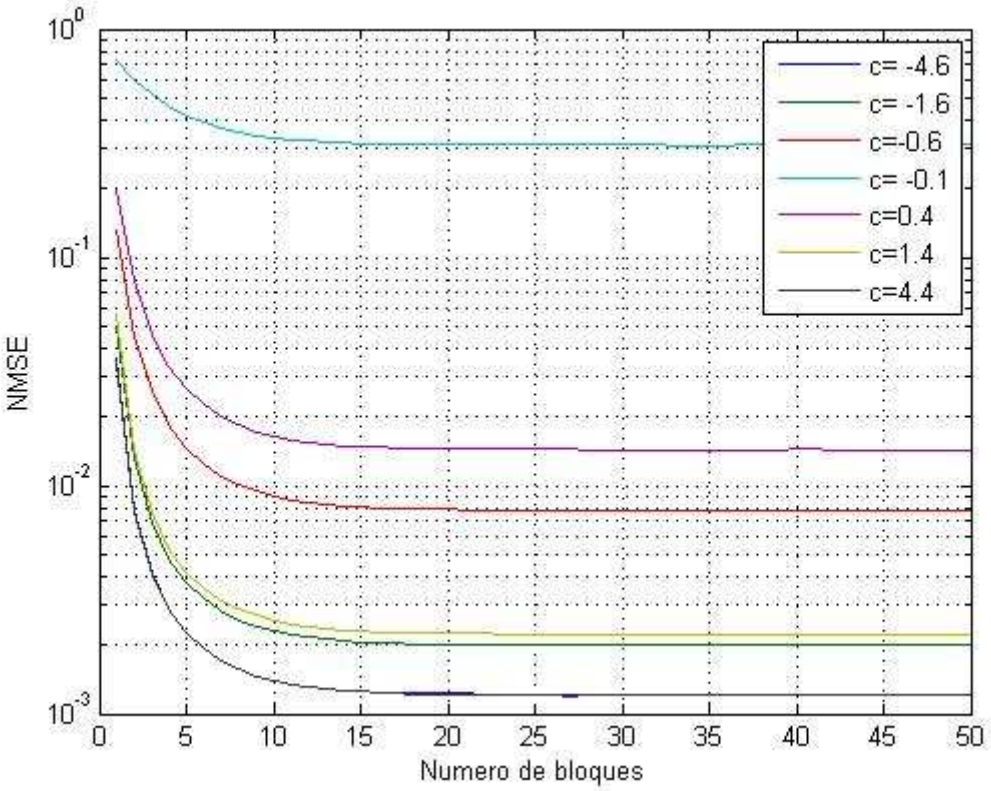

Figura 4.6 NMSE del algoritmo nº1 para distintos valores de c

Por otro lado tenemos el parámetro *b*, para el cuál se han realizado unos análisis semejantes, los resultados se muestran en las figuras 4.7 y 4.8. En este podemos observar cierta similitud con el parámetro anterior en lo que respecta a la simetría de la figura con centro en *b*=0. A diferencia del anterior es aquí donde se encuentra el mínimo de nuestro parámetro, con un *NMSE* de 10^{-3}. Conforme nos vamos alejando de este punto el *NMSE* empieza a crecer con la misma velocidad en ambos sentidos, en la figura se ve que llega a *NMSE*=0.2, se puede observar como va aumentando, pero tendiendo hacia régimen permanente.

Es obvio que el mejor valor para nuestro sistema es b=0, lo cual si lo aplicamos a nuestra matriz obtendremos que la mejor opción es una matriz uniforme, es decir, con todos sus elementos iguales.

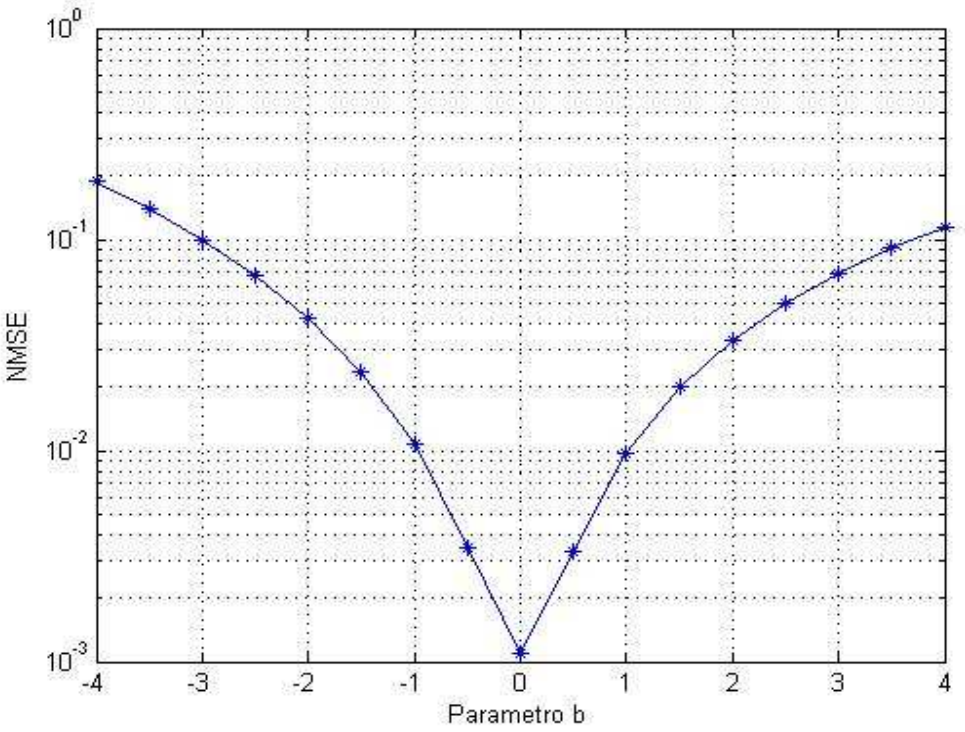

Figura 4.7 Figura 4.5 NMSE del algoritmo nº1 con respecto al parámetro b

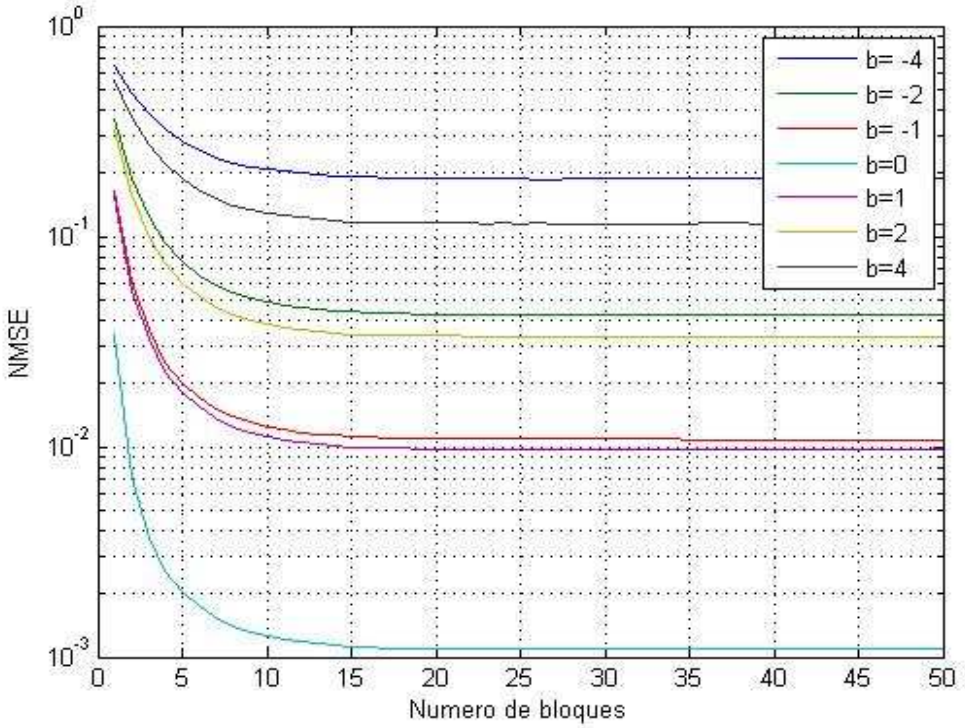

Figura 4.8 NMSE del algoritmo nº1 para distintos valores de b

– Tamaño de la constelación:

En este apartado observaremos la dependencia del algoritmo respecto al parámetro *M*, el cuál define el número de símbolos posibles en los que podemos codificar la trama de bits recibida para luego modularlas ortogonalmente. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintos valores de *M*, se ha simulado con la siguiente tabla de valores: [2, 4, 8, 16, 32, 64, 128, 256].

Los resultados de las simulaciones se muestran en las figuras 4.9 y 4.10. Es fácilmente observable en las gráficas que el tamaño de la constelación apenas influye en el *NMSE*. No obstante según la figura 4.10, en régimen permanente, ambas permanecen casi idénticas siendo para *M*=256 un poco un poco más efectivo el algoritmo.

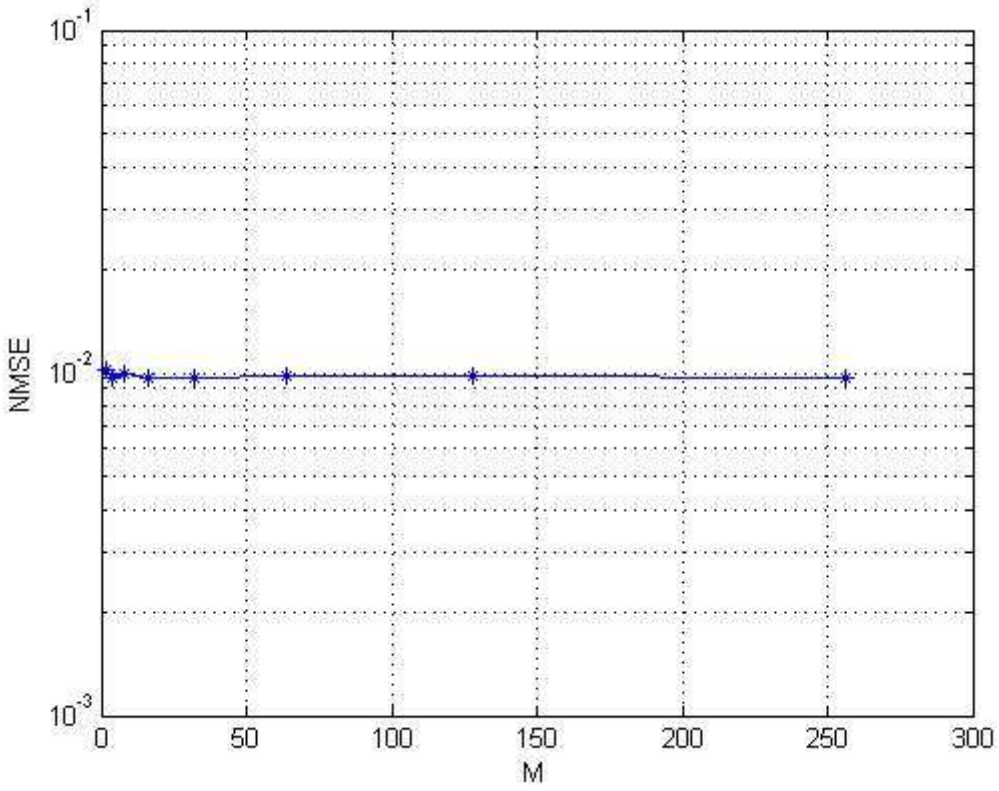

Figura 4.9 NMSE del algoritmo nº1 con respecto al parámetro M

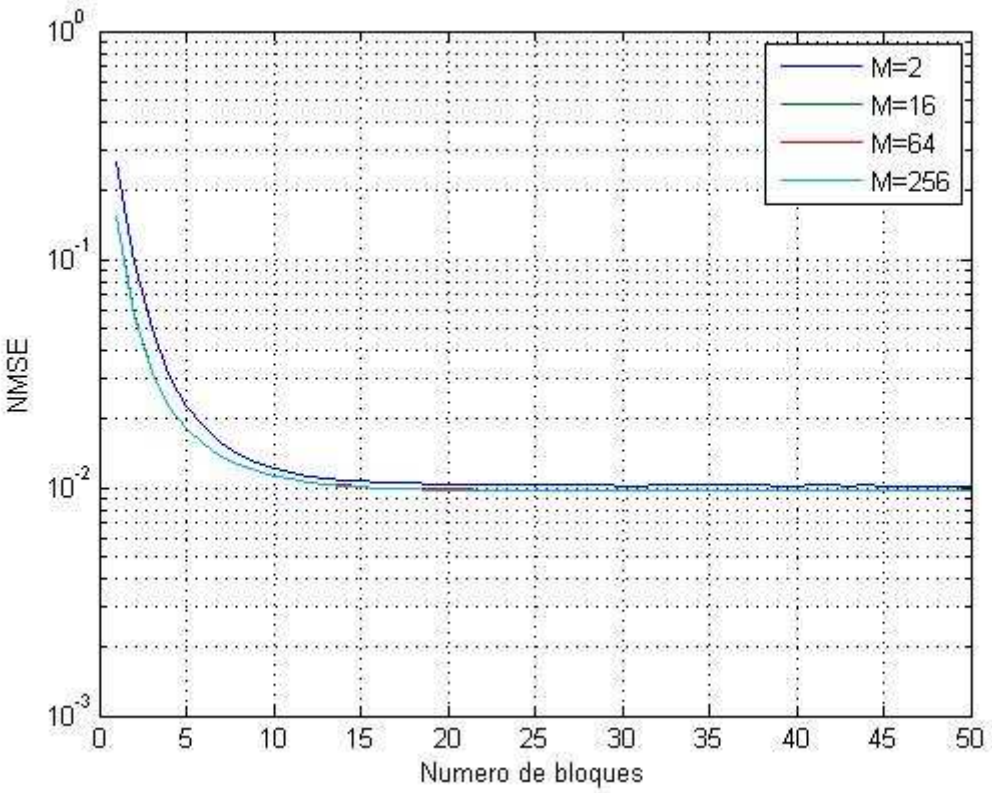

Figura 4.10 NMSE del algoritmo nº1 para distintos valores de M

- Numero de portadoras:

En este apartado observaremos la dependencia del algoritmo respecto al número de portadoras, por número de portadoras entendemos el parámetro *N*. Este indica el número de portadoras que se usarán para transmitir los símbolos en cada bloque OFDM, o en términos matemáticos el tamaño de la FFT utilizada. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintos valores de *N*, se ha simulado con la siguiente tabla de valores: [32, 64, 128, 256].

Como se observa en las gráficas 4.11 y 4.12 el parámetro *N* en valores entorno a 64 portadoras, esta se manifiesta con la mejora del algoritmo a medida que se aumenta el número de portadoras a utilizar. La última muestra de la que disponemos es de *N*=256, se puede deducir que el algoritmo alcanzará un régimen permanente en lo que a *NMSE* dependiente de *N* se refiere. Podemos comprobar esto a partir de como disminuye el salto de NMSE entre valores de N.

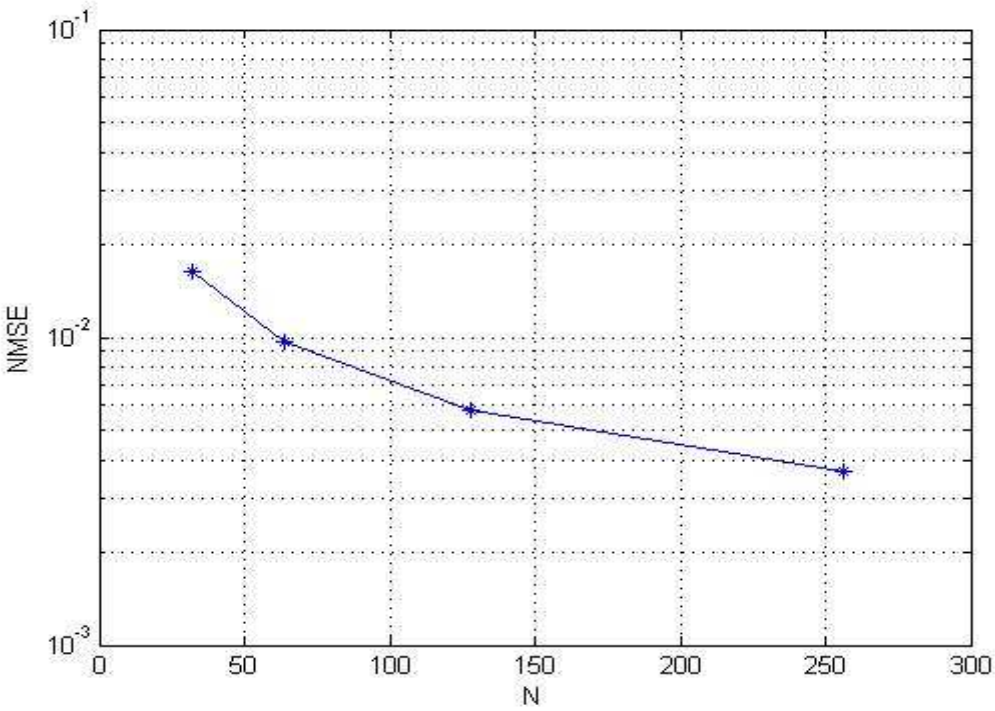

Figura 4.11 NMSE del algoritmo nº1 con respecto al parámetro N

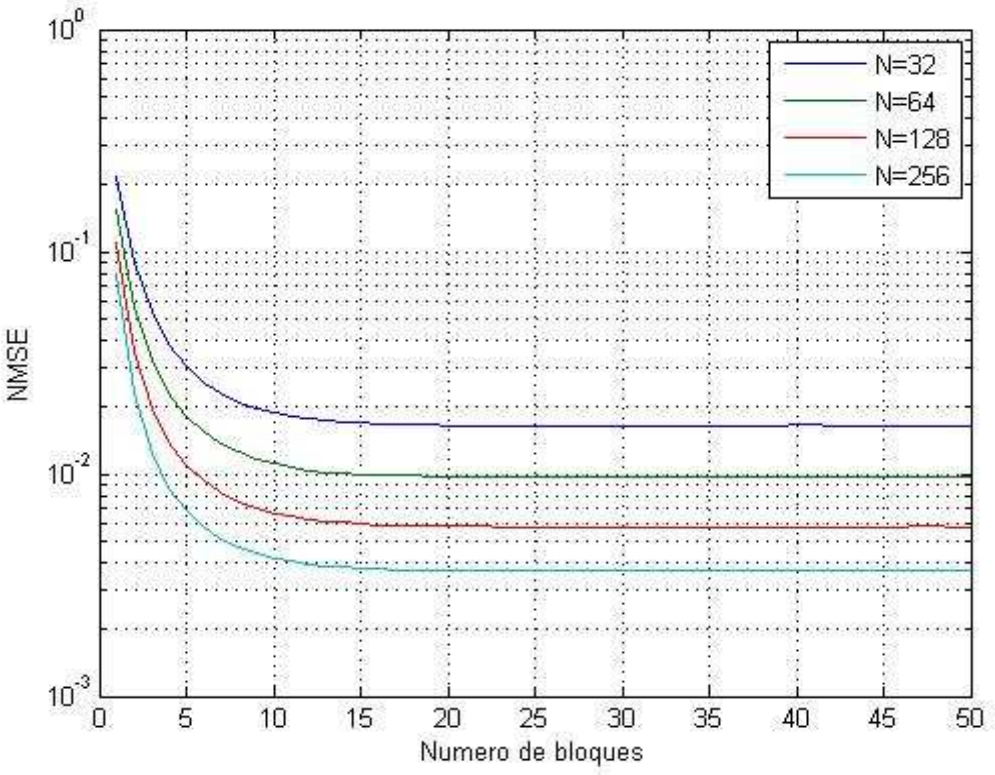

Figura 4.12 NMSE del algoritmo nº1 para distintos valores de N

– Factor de memoria de autocorrelación:

En este apartado observaremos la dependencia del algoritmo respecto al factor de memoria de la autocorrelación, *a*. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintas *a's*, se ha simulado con valores entre a=0.1 y a=0.9 con un paso de 0.1.

En la figura 4.13, podemos ver como el *NMSE* tiene una dependencia casi lineal con respecto al índice de autocorrelación. Conforme aumentamos este último el *NMSE* aumenta también. Es un parámetro bastante influyente, ya que el resultado puede variar desde *NMSE*=0.09 hasta *NMSE*=0.002.

Por otro lado, si nos fijamos en la figura 4.14, se puede ver que este parámetro no sólo afecta al *NMSE*, sino también a la velocidad de convergencia. Esto es, como no podía ser de otra forma, mientras mejoramos el *NMSE* aumentamos el tiempo en el cuál el algoritmo converge hacia su resultado óptimo. La elección de un valor u otro de este parámetro vendrá definida por las necesidades de convergencia de nuestro sistema.

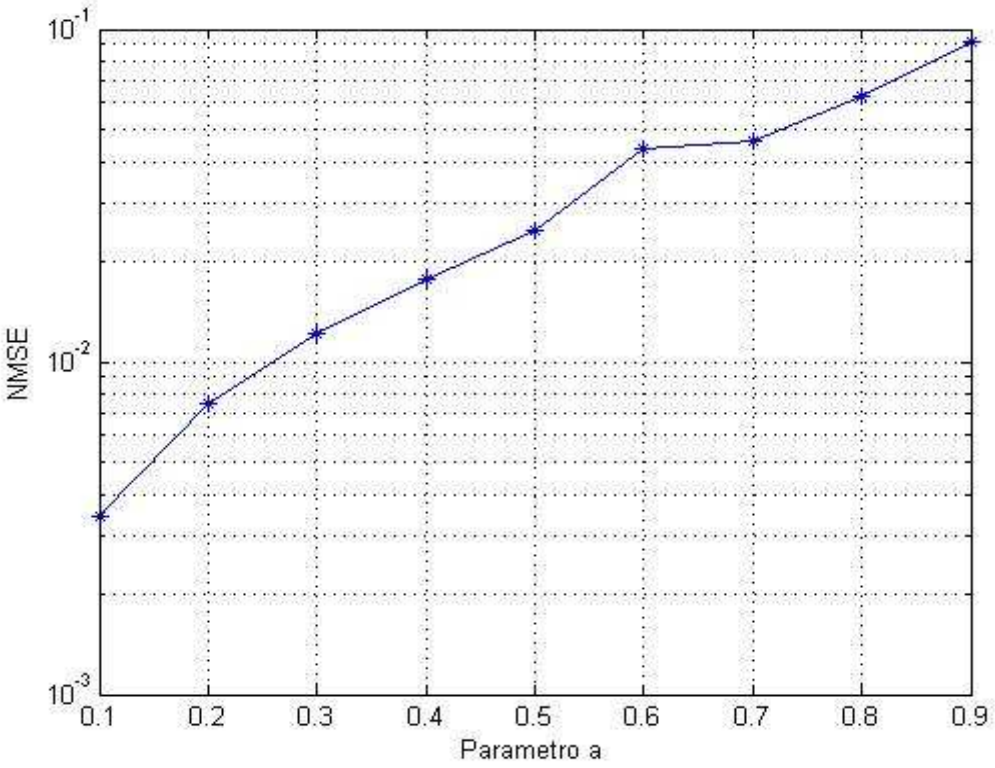

Figura 4.13 NMSE del algoritmo nº1 con respecto al parámetro a

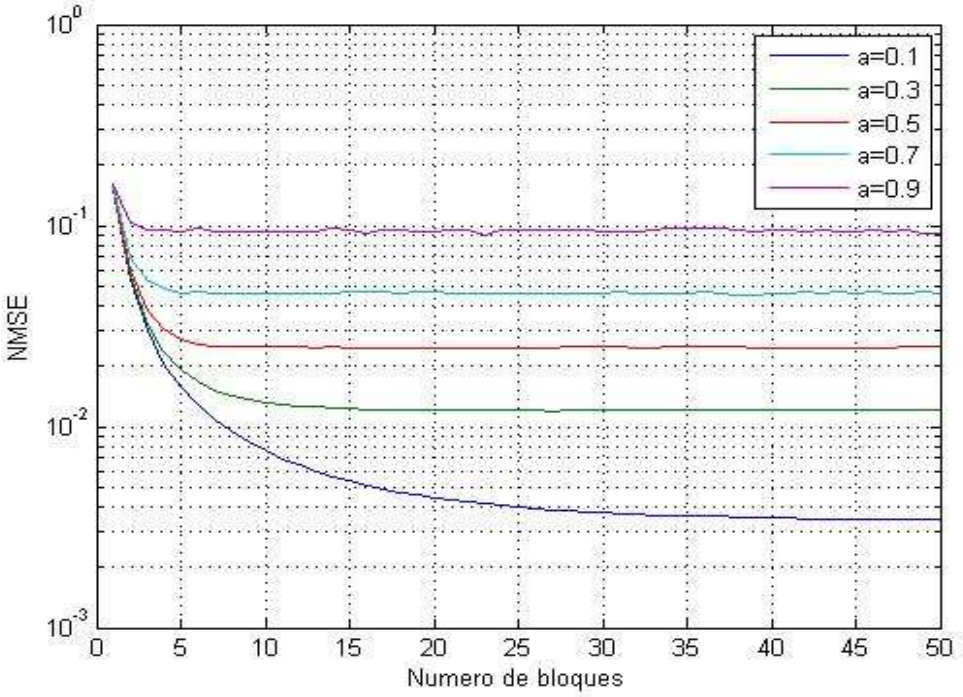

Figura 4.14 NMSE del algoritmo nº1 para distintos valores de a

– Coste computacional:

En primer lugar veremos cuál es el coste del algoritmo: 12.548ms, teniendo en cuenta que el coste original era 5.842ms, por lo tanto tenemos que el coste introducido es: 6.706ms.

4.3.3 Algoritmo nº2, de Ruifeng Zhang

– **Código:**

Tal y como se indica en la descripción del mismo, en primer lugar se genera la matriz precodificadora según la ecuación 3.16:

```
cof1=sqrt(rob);
cof2=sqrt((1-rob)/(N-1));
A=cof2*(ones(N)-eye(N))+cof1*eye(N);              %matriz precodificadora
```

Donde *rob* se encuentra en el intervalo (0,1). Mediante estas líneas de código generamos la matriz impuesta por el algoritmo en función del parámetro *rob*.

El siguiente paso es aplicar la matriz a los datos generados una vez después de realizar la codificación QAM, tal y como se muestra en la ecuación 3.17:

```
data_c=A*data(i:i+N-1);                    %codificación de datos
```

Para la aplicación del algoritmo proseguiremos calculando la matriz de autocorrelación modificada según la ecuación 3.21, posteriormente realizaremos la descomposición en valores singulares de dicha matriz y escogeremos el vector singular:

```
W=R./F;                          %cálculo de la matriz autocorrelación modificada
[U,D,V]=svd(W);                  %descomposición en valores singulares
d=diag(D);
may=max(d);
w=find(d==may);                  %obteniendo el puesto del vector singular
H2=V(:,w);                      %extracción del vector singular
```

El vector singular es el que corresponde al mayor valor, como se puede ver en el código. El vector H2 corresponde a nuestra estimación de canal sólo a falta de una constante.

- **Análisis:**

En primer lugar estableceremos los parámetros a variar:

```
rob=0.9;                %parámetro de la matriz precodificadora
snr=20;                 %relación señal a ruido
N=64;                   %número de portadoras a utilizar
M=4;                    %tamaño de la constelación
a=0.25;                 %factor de memoria de la autocorrelacion
```

Dados estos valores, observaremos como responde el algoritmo si se transmiten 50 bloques OFDM:

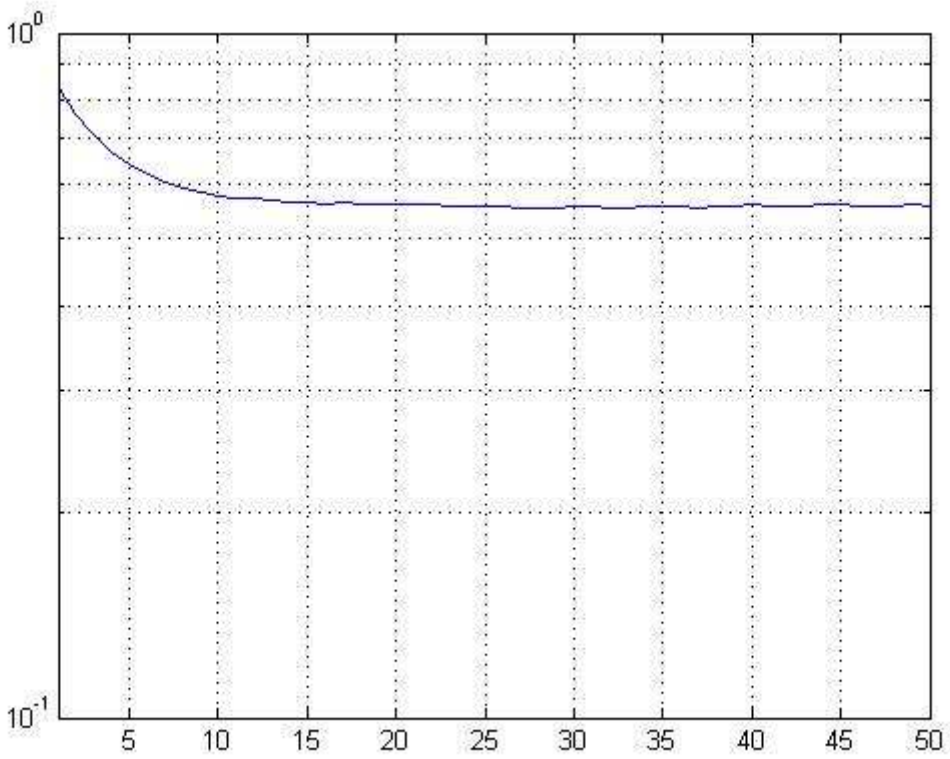

Figura 4.15 NMSE para el algoritmo nº2

Como se puede ver en la figura 4.15, el comportamiento de este algoritmo es bastante peculiar en el sentido de los resultados obtenidos. Aunque en principio parece seguir con el ejemplo de los algoritmos que veníamos analizando en el sentido de la convergencia hacia cierto valor, los resultados son totalmente inaceptables debido a su valor. Este algoritmo ofrece un error del 55% en sus estimaciones de canal.

Durante la implementación de este algoritmo se ha observado un efecto muy perjudicial sobre el mismo cuando se aumenta el número de portadoras, llegando a los valores mostrados en la figura 4.15. No obstante con un número de portadoras pequeño los resultados son aceptables como se observa en la figura 4.16, del orden de los que venimos teniendo. El inconveniente es que nuestra batería de canales tenía una longitud de *delta*=21, esto impide que se use un numero de portadoras mayor que 21, o 32 (ya que el número de portadoras que veníamos utilizando era potencia de 2). Así que no obtendríamos resultados objetivos simulando este algoritmo bajo unas condiciones diferentes a las del resto.

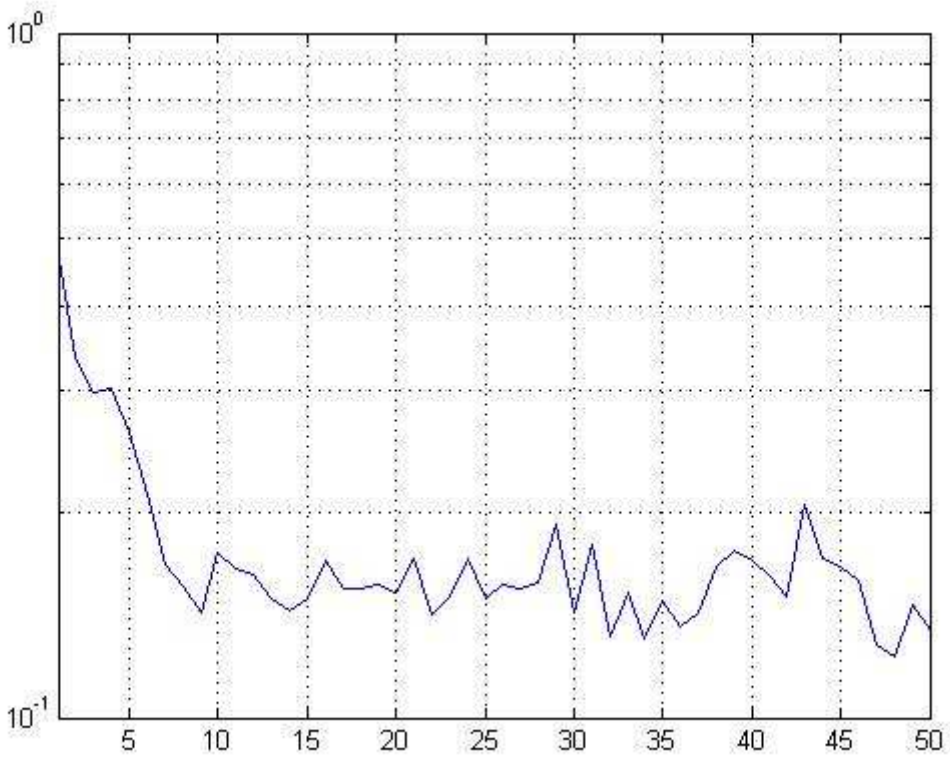

Figura 4.16 NMSE del algoritmo nº2 para N=4

En la figura 4.16 se observa como para *N*=4 ofrece unos valores mas o menos aceptables, cerca del 10% de error. Además de su limitado rango de validez, presenta un comportamiento muy inestable en régimen permanente. A pesar de la poca claridad de valores, se puede estimar una velocidad de convergencia del algoritmo de unos 15 bloques.

- Coste computacional:

En primer lugar veremos cuál es el coste del algoritmo: 9.466ms, teniendo en cuenta que el coste original era 5.842ms, por lo tanto tenemos que el coste introducido es: 3.624ms.

4.3.4 Algoritmo nº 3, de Ruifeng Zhang

– Código:

Tal y como se indica en la descripción del mismo, en primer lugar se genera la matriz precodificadora de forma aleatoria:

```
A=2*rand(N,m)-1;                %matriz precodificadora
```

Se trata de una matriz cuyos elementos son aleatorios entre -1 y 1, dicha matriz se mantendrá fija para cada simulación, variándose y promediándose junto al generador aleatorio y al ruido blanco gausiano, consiguiendo así un resultado objetivo independiente de la precodificación. Esta matriz tiene tamaño *Nxm* tal y como se describió en el capítulo 3.

El siguiente paso es aplicar la matriz a los datos generados una vez después de realizar la codificación QAM, tal y como se muestra en la ecuación 3.22:

```
data_c=A*data(i:i+m-1);             %aplicación de la matriz codificadora Nxm
```

Para la aplicación del algoritmo, en primer lugar realizaremos la extracción de los autovectores cuyos autovalores son menores con objeto de obtener el subespacio ruidoso, definidos como *em*. A continuación, la creación de la matriz *fi* acorde a la ecuación 3.28. El último paso por dar es la extracción del autovector principal de la matriz *fi'*fi*.

```
[V,D]=eig(R);                           %calculo de los autovectores
em=V(:,m+1:N);                          %extracción de los autovectores de ruido
fi=zeros(m*(N-m),N);
for j=1:N-m

    fi(((j-1)*m)+1:j*m,:)=(diag(em(:,j))*conj(A))';

end
[V2,D2]=eig(fi'*fi);                    %calculo del autovector mínimo
H2=V2(:,1);
```

El autovector principal es este que corresponde al mayor autovalor, como se puede ver en el código, el comando eig de MATLAB ordena los autovectores de menor a mayor según sus autovalores. El vector H2 corresponde a nuestra estimación de canal sólo a falta de una constante.

- **Análisis:**

En primer lugar estableceremos los parámetros a variar:

```
m=6;                    %tamaño de los bloques de símbolos
snr=20;                  %relación señal a ruido
N=64;                    %número de portadoras a utilizar
M=4;                     %tamaño de la constelación
a=0.25;                  %factor de memoria de la autocorrelacion
```

Dados estos valores, observaremos como responde el algoritmo si se transmiten 70 bloques OFDM:

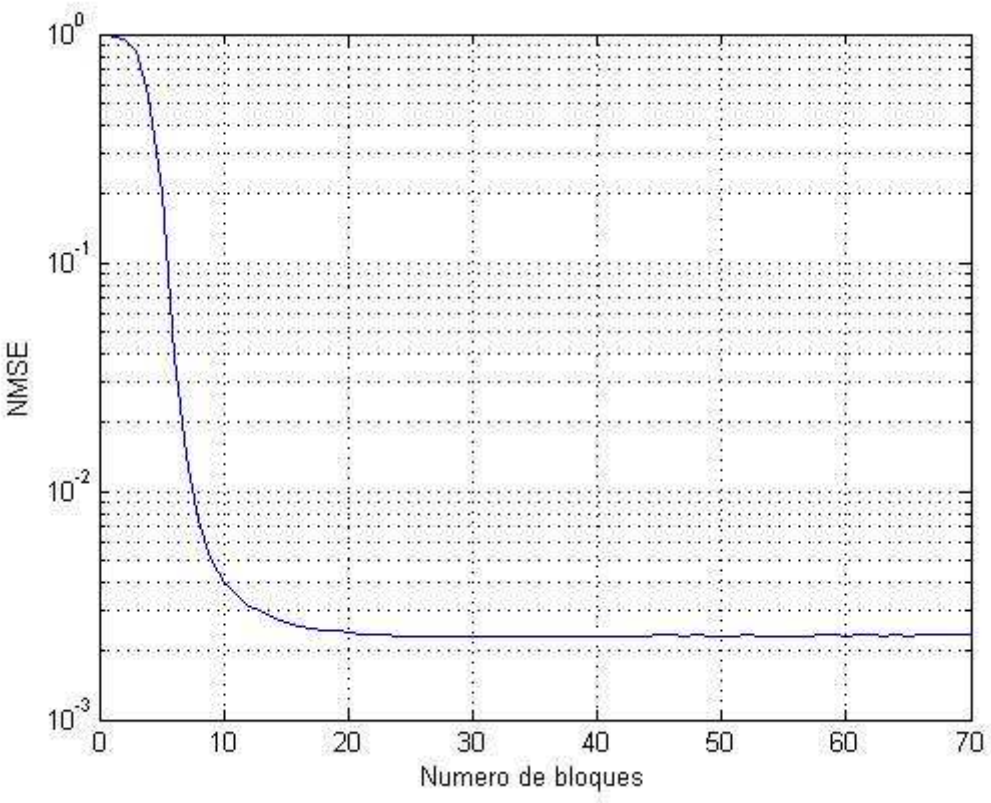

Figura 4.17 NMSE para el algoritmo nº3

Como se puede observar en la figura 4.17, el algoritmo da una *NMSE* de menos de 0.003 a partir del bloque número 25 más o menos. Esto se traduce en que una vez pasado un tiempo transitorio, el error con el que este algoritmo estimará el canal será menor de un 0.3%. En los siguientes apartados veremos como responde el algoritmo a la variación de los parámetros anteriormente expuestos.

– Relación señal a ruido, SNR:

En este apartado observaremos la dependencia del algoritmo respecto a la *SNR* existente en el sistema. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintas *SNR's*, se ha simulado con valores entre *SNR*=0 dB y *SNR*=30dB con un paso de 2dB.

En la figura 4.18 podemos ver como una *SNR* baja puede distorsionar en gran medida nuestra estimación del canal, llegando hasta errores de casi un 100% en régimen permanente, no obstante, conforme aumentamos la *SNR* desde posiciones bajas, el *NMSE* se hace cada vez más pequeño obteniendo una relación casi lineal entre ambos. Dicha relación se cumple hasta los 30 dB simulados con un *NMSE*=0.0002, se puede predecir que la relación en los próximos valores de *SNR* irá decreciendo linealmente. Este hecho es bastante positivo para sistemas en los que se puede obtener una relación señal a ruido grande en detrimento de otros parámetros.

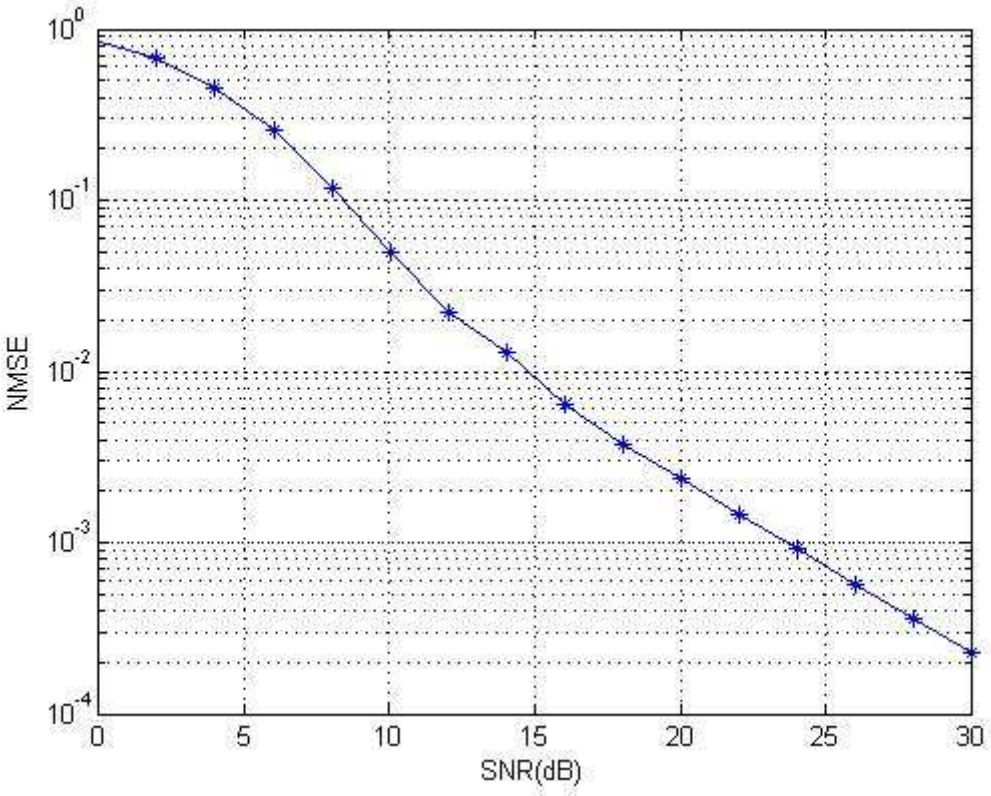

Figura 4.18 NMSE del algoritmo nº3 con respecto a la SNR

En la figura 4.19 se representan la adaptación del algoritmo para distintas *SNR's*, aquí se confirma lo que se veía en la figura anterior, se puede observar un espaciado entre curvas casi idéntico para idénticos incrementos de *SNR*. Esta figura también aporta otro punto de vista, como el tiempo de convergencia del sistema no se ve afectado en absoluto por la relación señal a ruido, tal y como se estableció al principio, el algoritmo pasa a régimen permanente una vez transmitido 25 bloques.

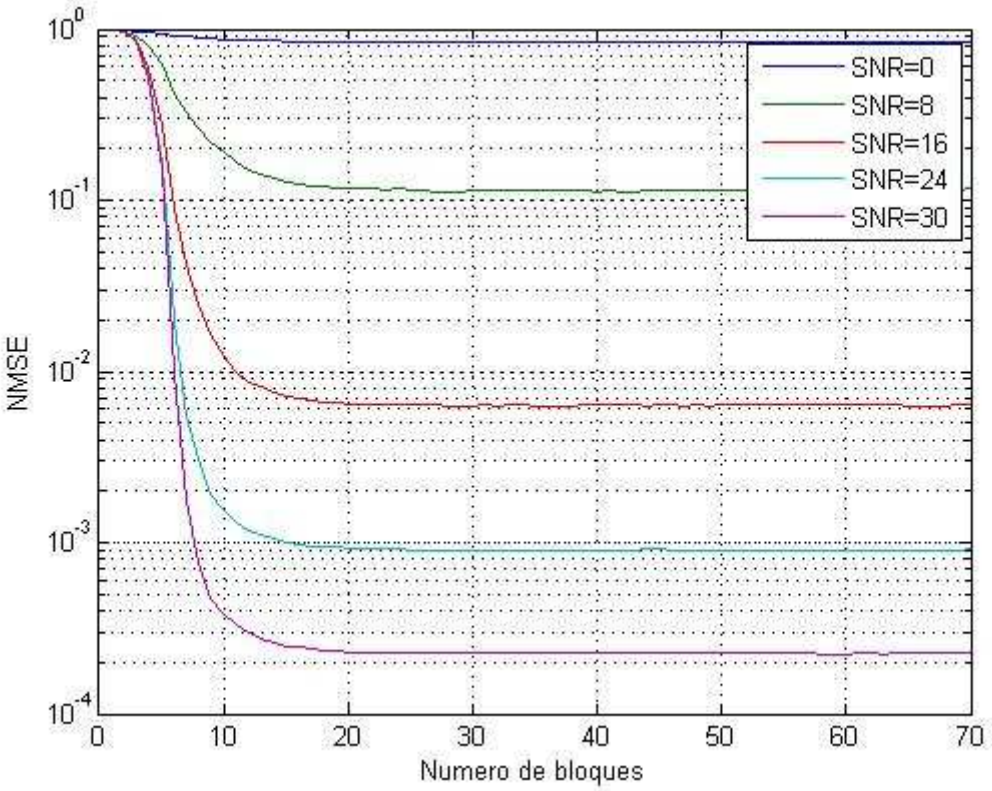

Figura 4.19 NMSE del algoritmo nº3 para distintas SNR

– Tamaño de la matriz precodificadora:

Tal y como se explicó en el análisis de este algoritmo, la matriz precodificadora tenía una cierta peculiaridad, las propiedades introducidas por esta vienen dadas por el tamaño de la misma no por su contenido. En realidad el contenido de la matriz es puramente aleatorio, este hecho se ha tenido en cuenta para que la posibilidad de que una matriz en concreto con ciertas propiedades no previstas modifique los resultados. Se han promediado los resultados de diferentes matrices junto al ruido gausiano y al generador de símbolos.

No obstante, este hecho tiene una implicación bastante importante en la secuencia transmitida, esto es, el parámetro *m* indica el número de símbolos reales que se transmiten por secuencia. Estos símbolos se convertirán en *N* símbolos OFDM tras ser precodificados y modulados, un requisito indispensable de este algoritmo es $m<N$, lo cuál conlleva a un desperdicio

de ancho de banda de *(N-m)/N* x100.

En las figuras 4.20 y 4.21 podemos ver los diferentes efectos del parámetro *m* sobre el *NMSE* del algoritmo. En primer lugar, puntualizar como para un valor muy pequeño como *m*=2, el error aumenta mucho, del orden de *NMSE*=0.05, manteniendo una velocidad de convergencia más que aceptable del orden de unos 15 bloques. No obstante el desperdicio de ancho de banda es enorme con respecto a las pocas prestaciones. Una vez nos centramos en el parámetro *m*=6, los resultados se estabilizan, desde aquí en adelante se sigue un progresión lineal tanto de aumento de error como de velocidad de convergencia llegando a un *NMSE*=0.008, con una velocidad de convergencia de unos 45 bloques para *m*=30. Estos es, conforme aumentamos el parámetro *m*, perdemos precisión en la estimación de canal y ralentizamos el ritmo de convergencia de nuestro sistema. Aunque por otro lado, mientras más símbolos logremos meter en un bloque OFDM, menos ancho de banda de transmisión estaremos desperdiciando.

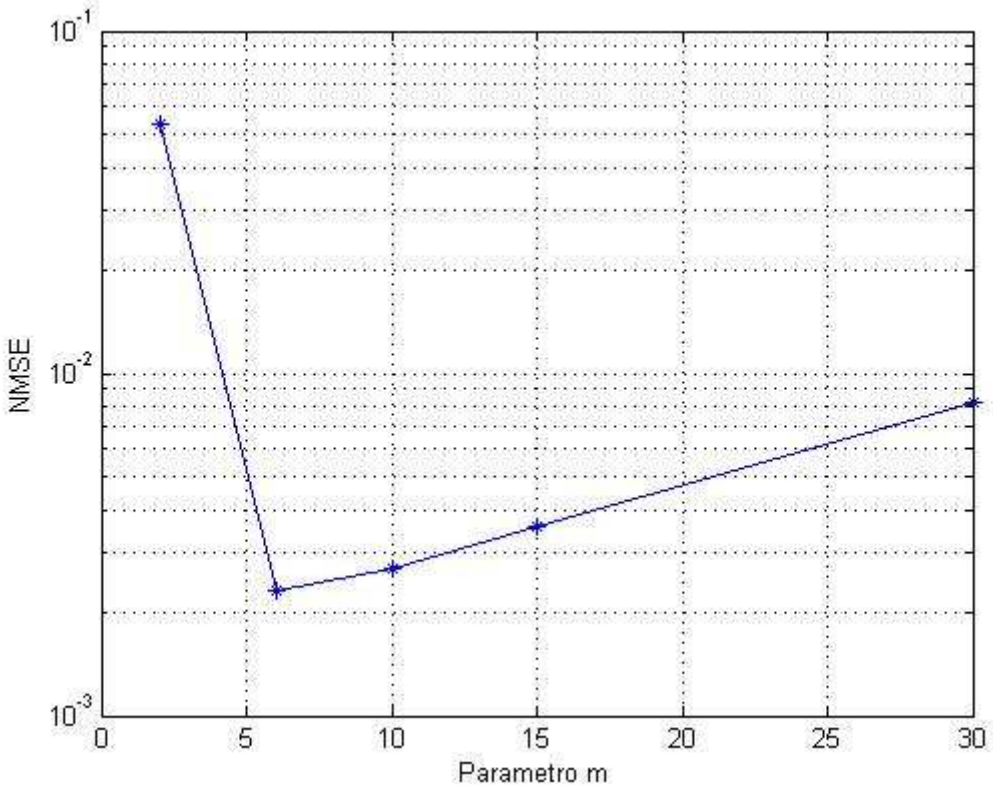

Figura 4.20 NMSE del algoritmo nº3 con respecto al parámetro m

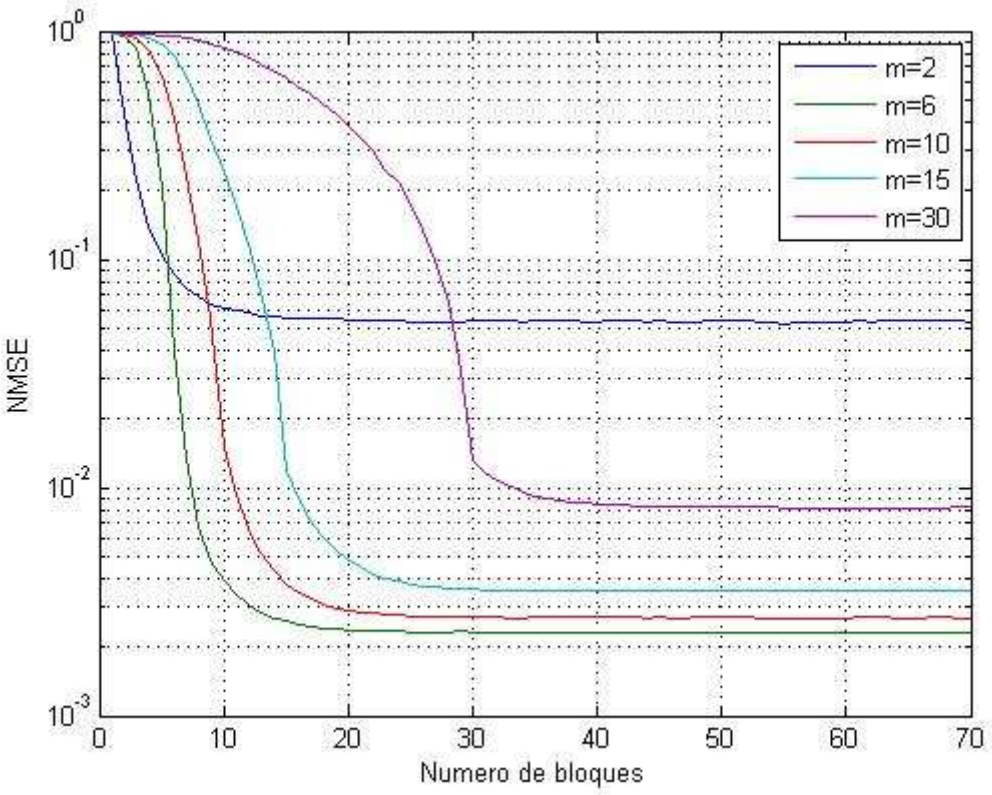

Figura 4.21 NMSE del algoritmo nº3 para distintos valores de m

– Tamaño de la constelación:

En este apartado observaremos la dependencia del algoritmo respecto al parámetro *M*, el cuál define el número de símbolos posibles en los que podemos codificar la trama de bits recibida para luego modularla ortogonalmente. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintos valores de *M*, se ha simulado con la siguiente tabla de valores: [2, 4, 8, 16, 32, 64, 128, 256].

Los resultados de las simulaciones se muestran en las figuras 4.22 y 4.23. Es fácilmente observable en las gráficas que el tamaño de la constelación apenas influye en el *NMSE*. No obstante según la figura 4.23, en régimen permanente, ambas permanecen casi idénticas siendo para *M*=256 un poco un poco más efectivo el algoritmo.

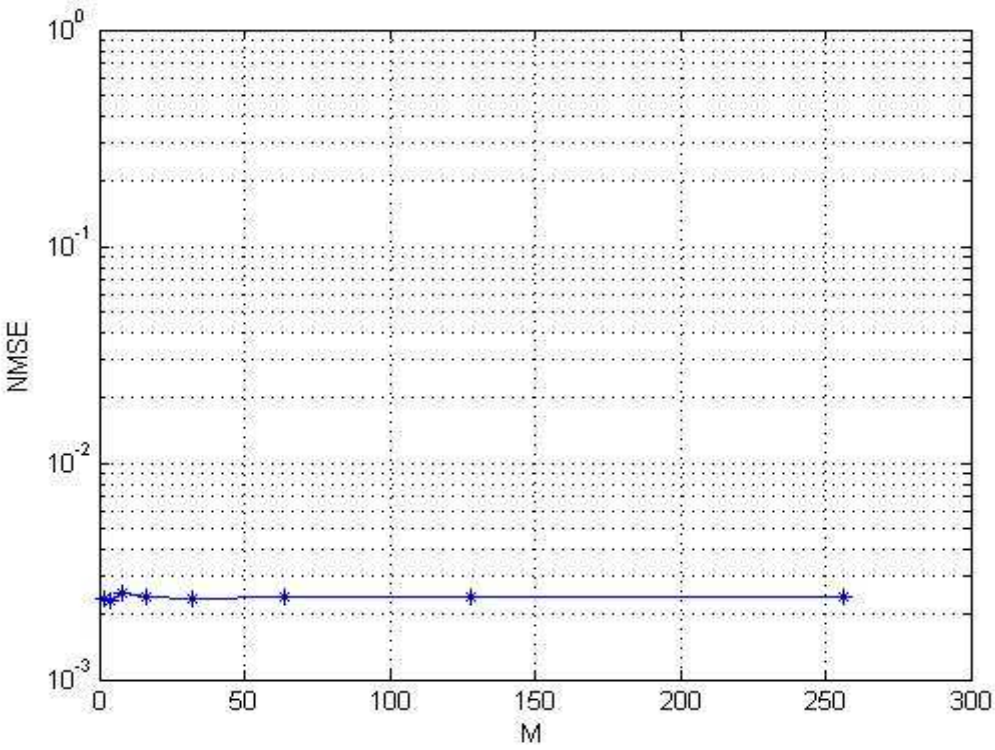

Figura 4.22 NMSE del algoritmo nº3 con respecto al parámetro M

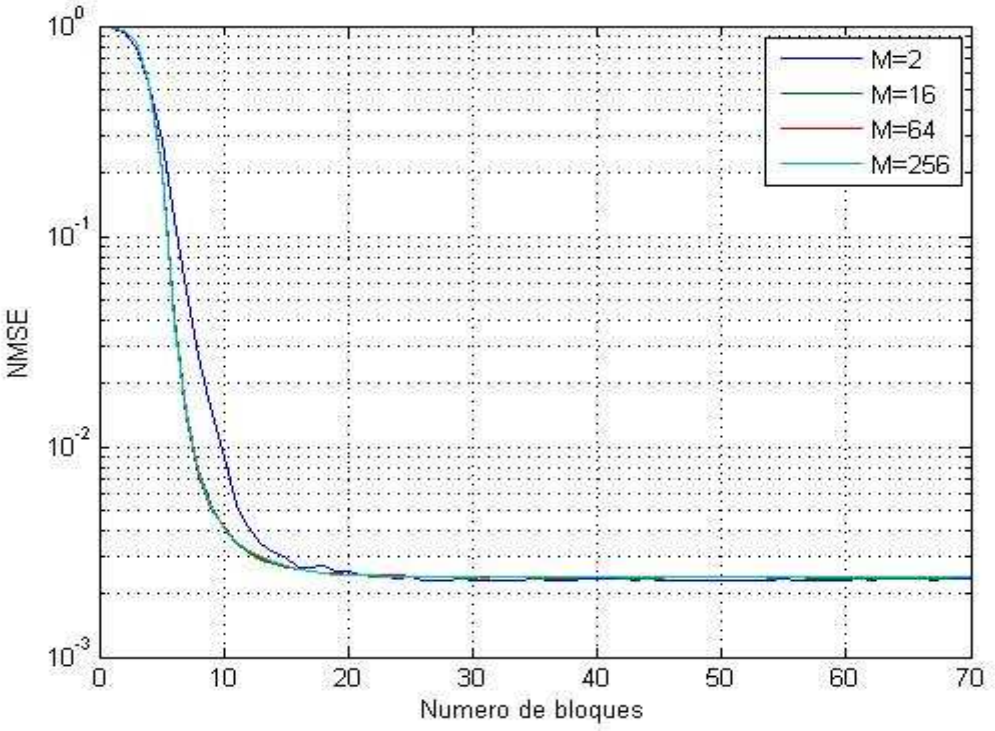

Figura 4.23 NMSE del algoritmo nº3 para distintos valores de M

– Numero de portadoras:

En este apartado observaremos la dependencia del algoritmo respecto al número de portadoras, por número de portadoras entendemos el parámetro *N*. Este indica el número de portadoras que se usarán para transmitir los símbolos en cada bloque OFDM, o en términos matemáticos el tamaño de la FFT utilizada. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintos valores de *N*, se ha simulado con la siguiente tabla de valores: [32, 64, 128, 256].

Al igual que el parámetro anterior, el número de portadoras no parece afectar mucho a los resultados obtenidos. Si se observan detenidamente las figuras 4.24 y 4.25, el *NMSE* tiende a reducirse aunque la diferencia no sea apreciable. El hecho de que no influya este parámetro indica que a este algoritmo no le afectará el número de portadoras que se utilice en el sistema de telecomunicaciones.

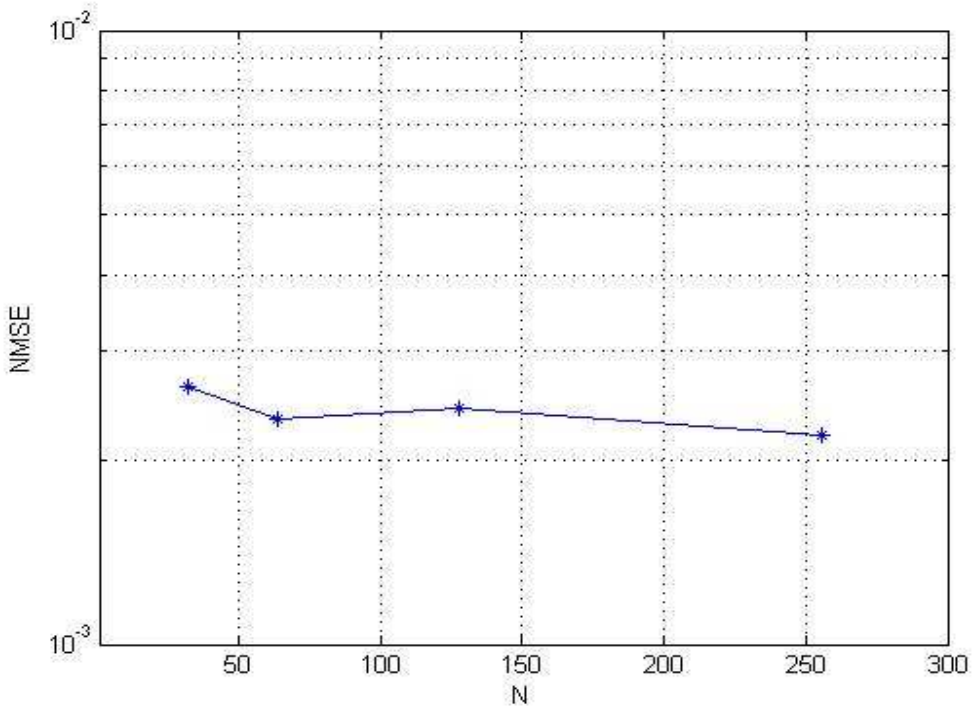

Figura 4.24 NMSE del algoritmo nº3 con respecto al parámetro N

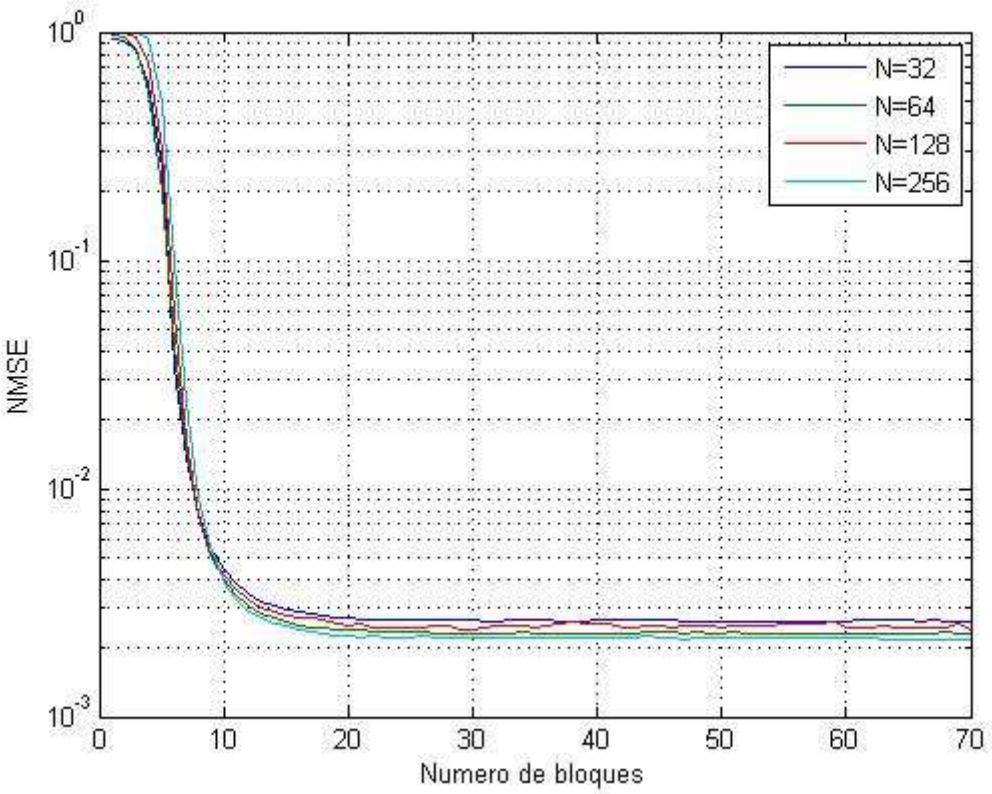

Figura 4.25 NMSE del algoritmo nº3 para distintos valores de N

– Factor de memoria de autocorrelación;

En este apartado observaremos la dependencia del algoritmo respecto al factor de memoria de la autocorrelación, *a*. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintas *a's*, se ha simulado con valores entre *a*=0.1 y *a*=0.9 con un paso de 0.1.

En la figura 4.26, podemos ver como el *NMSE* tiene una dependencia casi lineal con respecto al índice de autocorrelación, conforme aumentamos este último el *NMSE* aumenta también. Es un parámetro bastante influyente, ya que el resultado puede variar desde *NMSE*=0.1 hasta *NMSE*=0.0008.

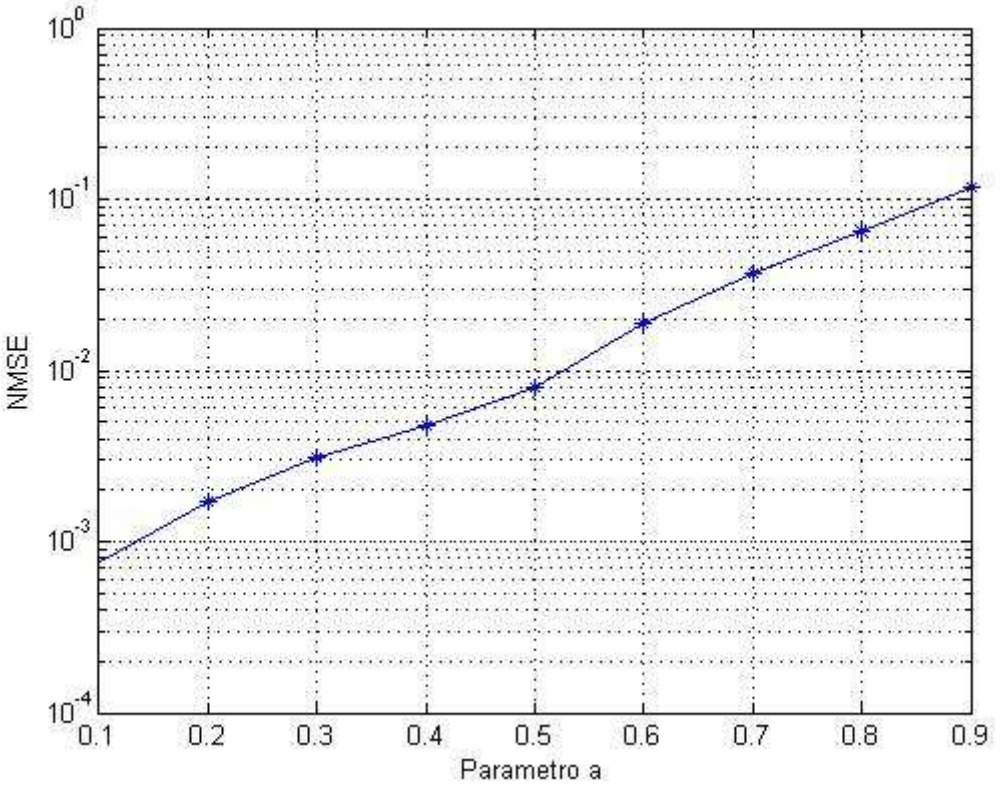

Figura 4.26 NMSE del algoritmo nº3 con respecto al parámetro a

Por otro lado, si nos fijamos en la figura 4.27, se puede ver que este parámetro no sólo afecta al *NMSE*, sino también a la velocidad de convergencia. Esto es, como no podía ser de otra forma, mientras mejoramos el *NMSE* aumentamos el tiempo en el cuál el algoritmo converge hacia su resultado óptimo. La elección de un valor u otro de este parámetro vendrá definida por las necesidades de convergencia de nuestro sistema. Este caso es particular ya que para los valores de a=0.5 y a=0.25 el algoritmo converge mucho más rápido que para el resto de valores. Se puede ver en figura como a=0.5 provoca una caída mucho más rápida que el resto. También se observa como para el resto de valores la convergencia es más lenta de lo que viene siendo en el resto del análisis del algoritmo, esto es porque durante el análisis se viene usando el valor estándar a=0.25.

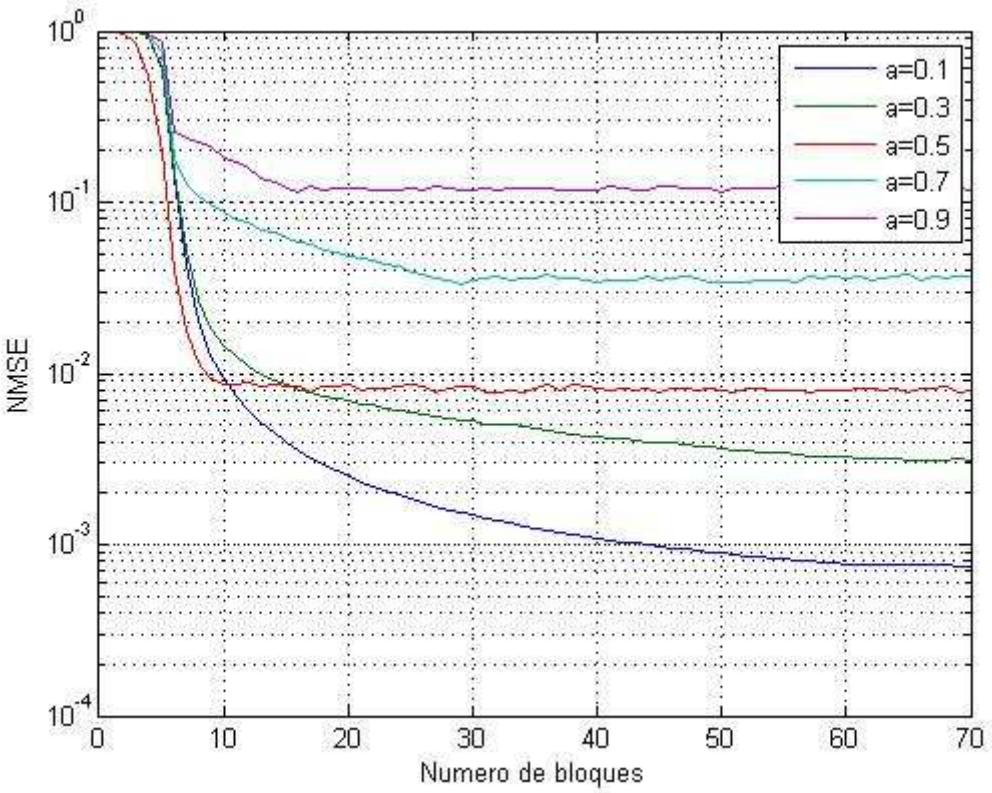

Figura 4.27 NMSE del algoritmo nº3 para distintos valores de a

- Coste computacional:

En primer lugar veremos cuál es el coste del algoritmo: 49.889ms, teniendo en cuenta que el coste original era 5.842ms, por lo tanto tenemos que el coste introducido es: 44.047ms.

4.3.5 Algoritmo nº 4, de Feifei Gao y Arumugam Nallanathan

- **Código:**

Tal y como se indica en la descripción del mismo, en primer lugar se genera la matriz precodificadora en función del parámetro *p*, según se indica en las ecuaciones 3.33 y 3.34.

```
pe=p*ones(N)+(1-p)*eye(N);
W=sqrtm(pe);                    %matriz precodificadora
```

En el código se calcula en primer lugar la matriz **pe**, a partir de la cuál se obtiene la matriz de precodificación que se usará en el algoritmo. Esta matriz variará en función de un parámetro *p*, que se variará durante las simulaciones para ver sus efectos sobre el *NMSE*.

El siguiente paso es aplicar la matriz a los datos generados una vez después de realizar la codificación QAM, tal y como se muestra en la ecuación 3.30:

```
data_c=W*data(i:i+N-1)            %aplicación de la matriz precodificadora
```

El algoritmo comienza con el cálculo de la autocorrelación de los símbolos extraídos de las portadoras recibidas. Ahora, aplicamos el algoritmo propuesto en el artículo, en primer lugar generamos la matriz de transformación de la DFT según las ecuaciones 3.41 y 3.42:

```
fm=exp(-2*pi*sqrt(-1)/N);            %variable de transformación

F=ones(N);

for i=1:N
    for j=1:N

        F(i,j)=fm^((i-1)*(j-1));      %generación de los elementos de la matriz

    end
end
```

A continuación aplicaremos el algoritmo del artículo para realizar la estimación del canal, el código está basado en las ecuaciones 3.39, 3.40, 3.43, 3.44.

```
for q=1:N
```

```
    Fq=[F(1:q-1,1:delta).',F(q+1:N,1:delta).'].';
    rq=[R(1:q-1,q).',R(q+1:N,q).'].';

    heq=heq+pinv(Fq)*rq/sqrt(N);  %estimación en el dominio del tiempo

end

Heq=sqrt(N)*F(:,1:delta)*heq/N;  %estimación en el dominio de la frecuencia
```

El vector **Heq** corresponde a nuestra estimación de canal sólo a falta de una constante.

- **Análisis:**

En primer lugar estableceremos los parámetros a variar:

```
p=0.5;                  %parámetro de la matriz de precodificación
snr=20;                 %relación señal a ruido
N=64;                   %número de portadoras a utilizar
M=4;                    %tamaño de la constelación
a=0.25;                 %factor de memoria de la autocorrelacion
```

Dados estos valores, observaremos como responde el algoritmo si se transmiten 50 bloques OFDM:

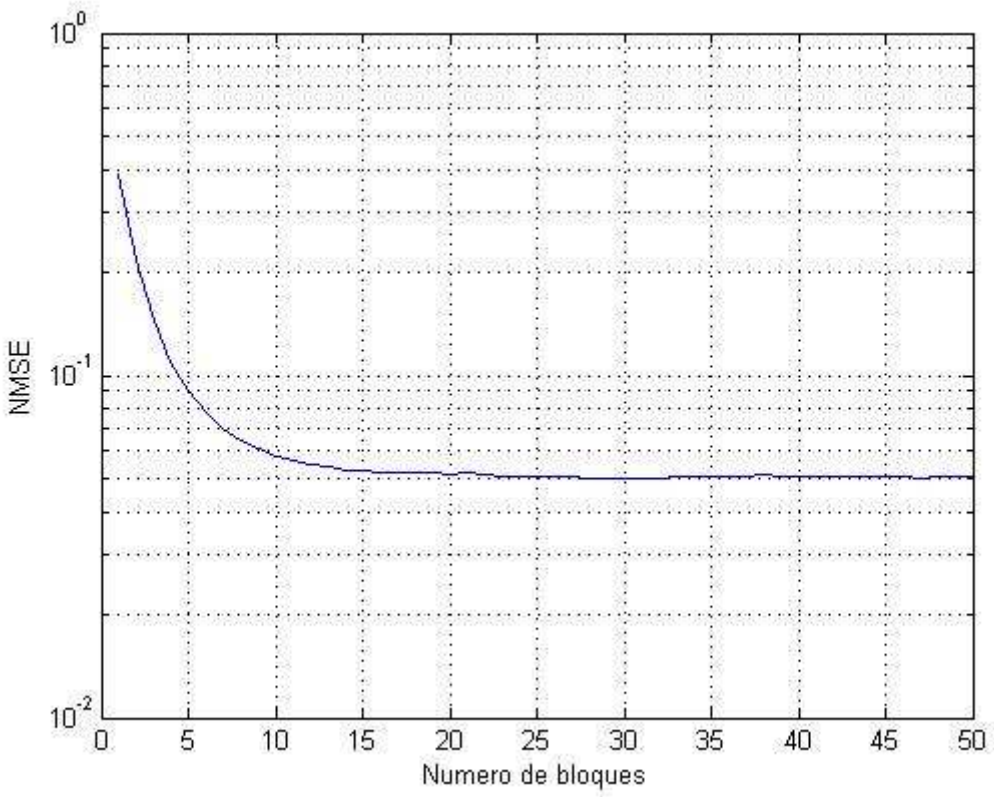

Figura 4.28 NMSE para el algoritmo nº4

Como se puede observar en la figura 4.28, el algoritmo da un *NMSE* de 0.05 a partir del bloque número quince. Esto se traduce en que una vez pasado un tiempo transitorio el error con el que este algoritmo estimará el canal será del 5%. En los siguientes apartados veremos como responde el algoritmo a la variación de los parámetros anteriormente expuestos.

- Relación señal a ruido, SNR:

En este apartado observaremos la dependencia del algoritmo respecto a la *SNR* existente en el sistema. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintas *SNR's*, se ha simulado con valores entre *SNR*=0 dB y *SNR*=30dB con un paso de 2dB.

En la figura 4.29 podemos ver como una *SNR* baja puede distorsionar en gran medida nuestra estimación del canal, llegando hasta errores del 15% en régimen permanente, no obstante

conforme aumentamos la SNR desde posiciones bajas, el *NMSE* se hace cada vez más pequeño obteniendo una relación casi lineal entre ambos. Dicha relación se cumple hasta los 10 dB, a partir del cuál el *NMSE* se vuelve menos dependiente de la *SNR* tendiendo a ser constante, se puede observar que la variación entre 10 dB y 30 dB es muy pequeña.

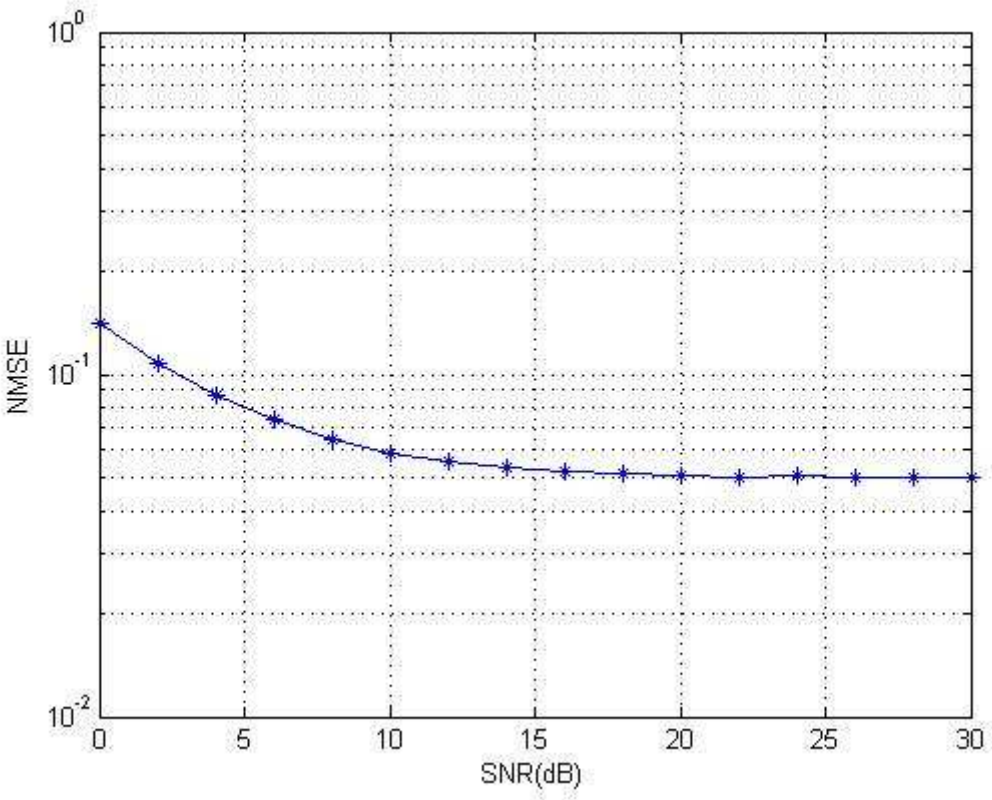

Figura 4.29 NMSE del algoritmo nº4 con respecto a la SNR

En la figura 4.30 se representan la adaptación del algoritmo para distintas *SNR's*, aquí se confirma lo que se veía en la figura anterior, como partiendo desde cero, pequeños aumentos de la SNR provocan grandes disminuciones del *NMSE*, hasta llegar a los 10 dB donde se tiende al mismo valor. Esta figura también aporta otro punto de vista, como el tiempo de convergencia del sistema no se ve afectado en absoluto por la relación señal a ruido, tal y como se estableció al principio el algoritmo pasa a régimen permanente una vez transmitido 15 bloques.

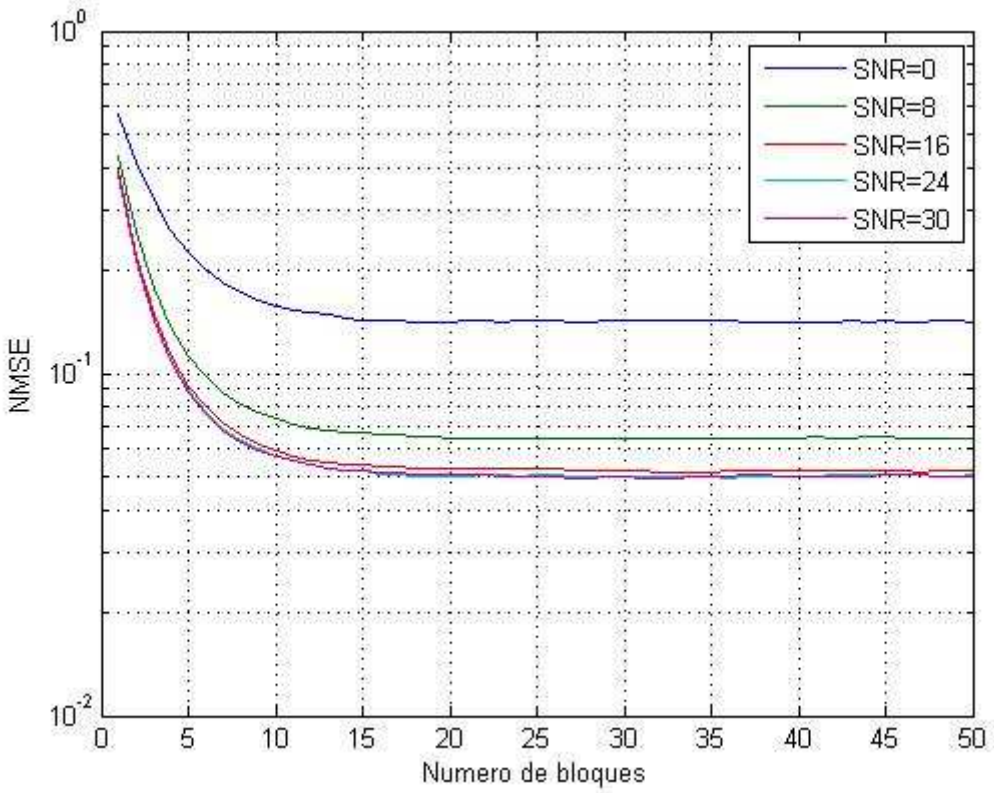

Figura 4.30 NMSE del algoritmo nº4 para distintas SNR

– Matriz de precodificación:

Otro elemento importante dentro de nuestro sistema es la matriz de precodificación del algoritmo. En este apartado observaremos la dependencia del algoritmo respecto a la matriz de precodificación del sistema. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintos valores de *p*. Se ha simulado con valores de *p* entre *p*=0 y *p*=1, con un paso de 0.1. Se ha añadido también a la simulación el valor *p*=-0.15 ya que el algoritmo permitía su uso.

Las figuras 4.31 y 4.32 muestran los resultados para el parámetro *p*. Se puede comprobar que el peor valor posible sería el *p*=0, obteniéndose un NMSE del 100%. A partir de ahí, si vamos incrementando el valor de *p* observamos una mejora del *NMSE* lineal al principio con bastante pendiente, para terminar decayendo bruscamente cuando nos acercamos a *p*=1, con un valor de 0.0003 aproximadamente. No obstante *p*=1 no está contemplado por el algoritmo como un valor

válido, así que un buen valor posible sería p=0.99. Por otro lado si le damos un valor negativo observamos como también se mejora el resultado.

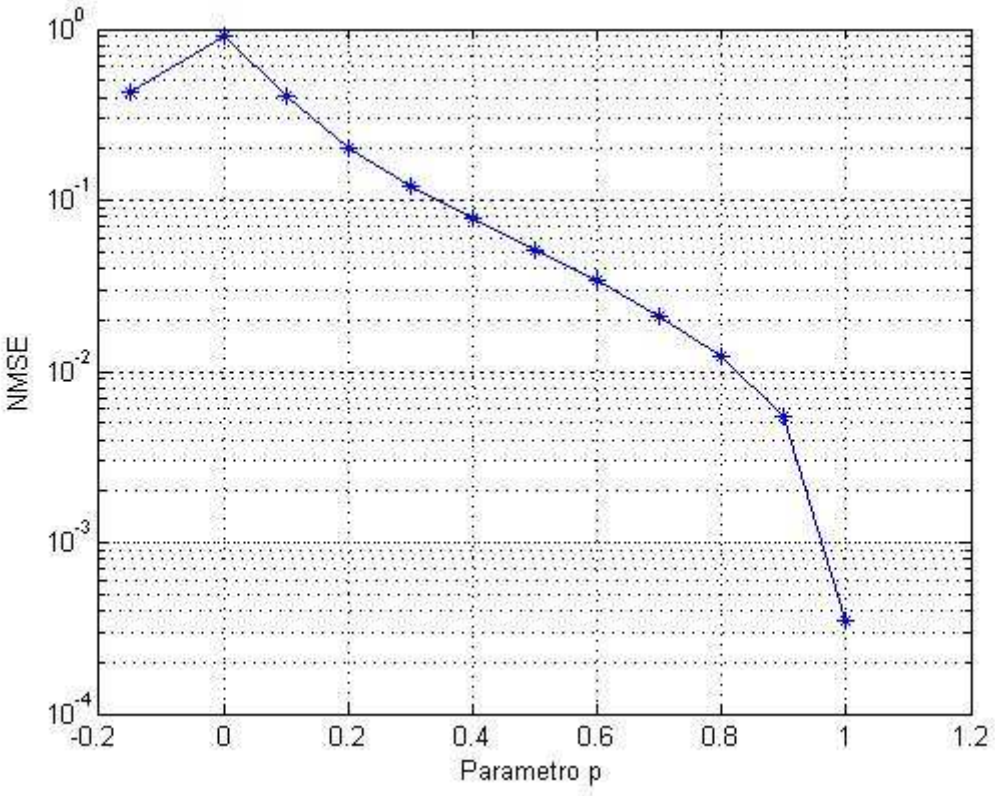

Figura 4.31 NMSE del algoritmo nº4 con respecto al parámetro p

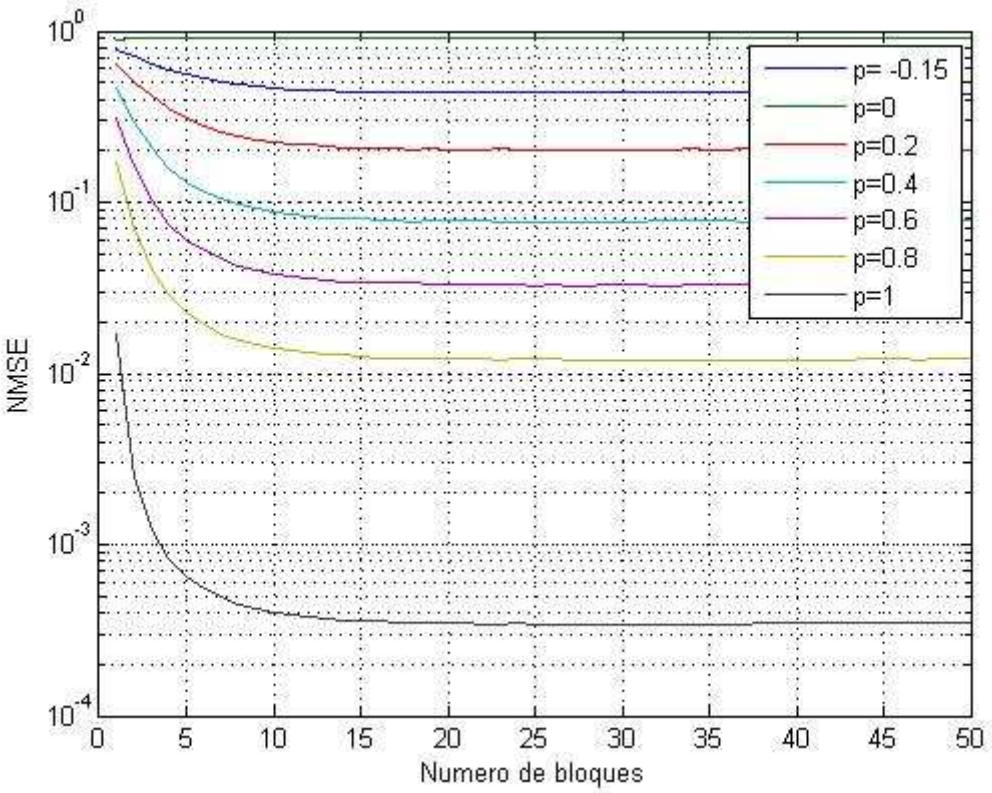

Figura 4.32 NMSE del algoritmo nº4 para distintos valores de p

– Tamaño de la constelación:

En este apartado observaremos la dependencia del algoritmo respecto al parámetro *M*, el cuál define el número de símbolos posibles en los que podemos codificar la trama de bits recibida para luego modularlas ortogonalmente. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintos valores de *M*, se ha simulado con la siguiente tabla de valores: [2, 4, 8, 16, 32, 64, 128, 256].

Los resultados de las simulaciones se muestran en las figuras 4.33 y 4.34. Es fácilmente observable en las gráficas que el tamaño de la constelación apenas influye en el *NMSE*. No obstante según la figura 4.34, en régimen permanente, ambas permanecen casi idénticas siendo para *M*=256 un poco un poco más efectivo el algoritmo.

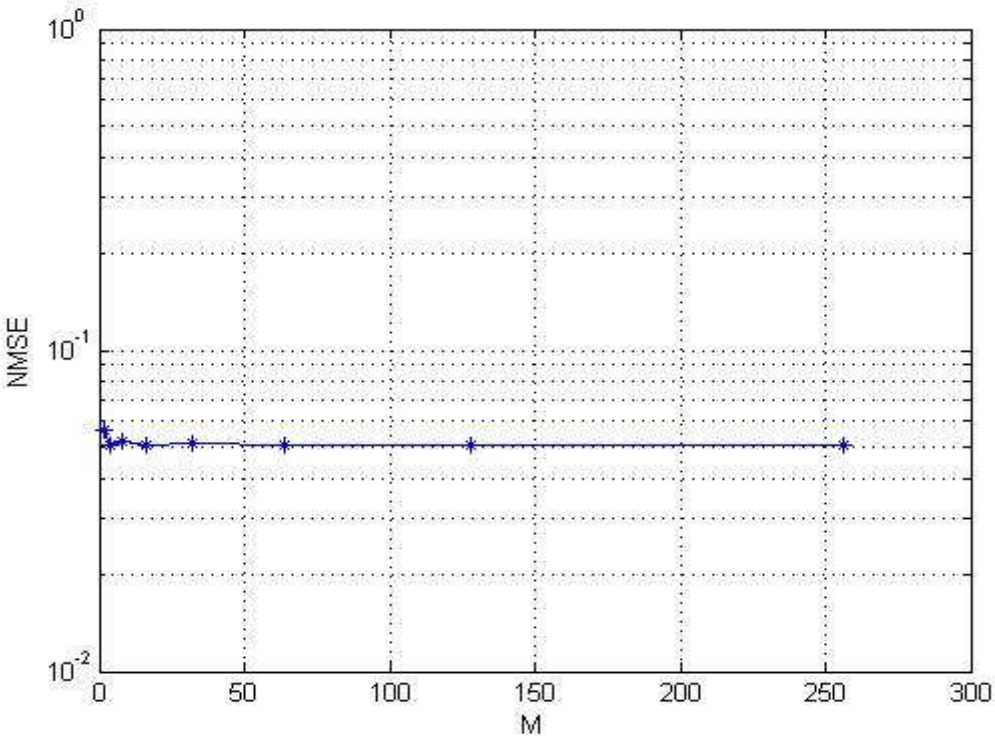

Figura 4.33 NMSE del algoritmo nº4 con respecto al parámetro M

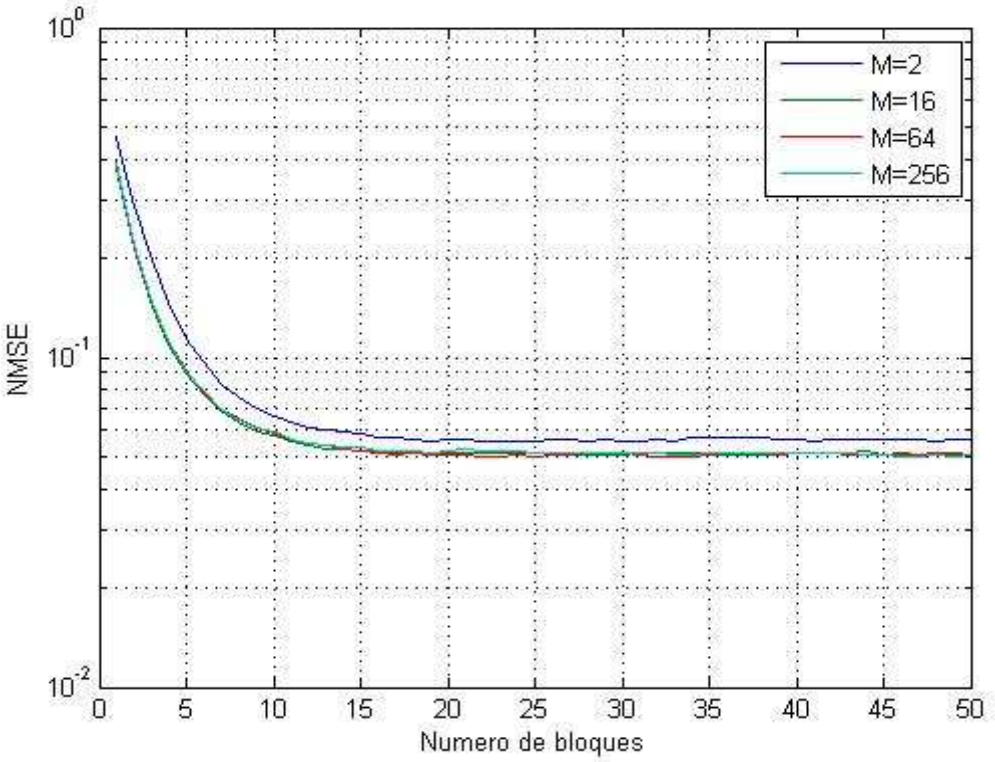

Figura 4.34 NMSE del algoritmo nº4 para distintos valores de M

– Numero de portadoras:

En este apartado observaremos la dependencia del algoritmo respecto al número de portadoras, por número de portadoras entendemos el parámetro *N*. Este indica el número de portadoras que se usarán para transmitir los símbolos en cada bloque OFDM, o en términos matemáticos el tamaño de la FFT utilizada. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintos valores de *N*, se ha simulado con la siguiente tabla de valores: [32, 64, 128, 256].

Como se observa en las figuras 4.35 y 4.36, el parámetro *N* en valores en torno a 64 portadoras, esta se manifiesta con la mejora del algoritmo a medida que se aumenta el número de portadoras a utilizar. Si se observan las cifras se puede observar como el *NMSE* disminuye a la mitad aproximadamente, conforme el número de portadoras se dobla.

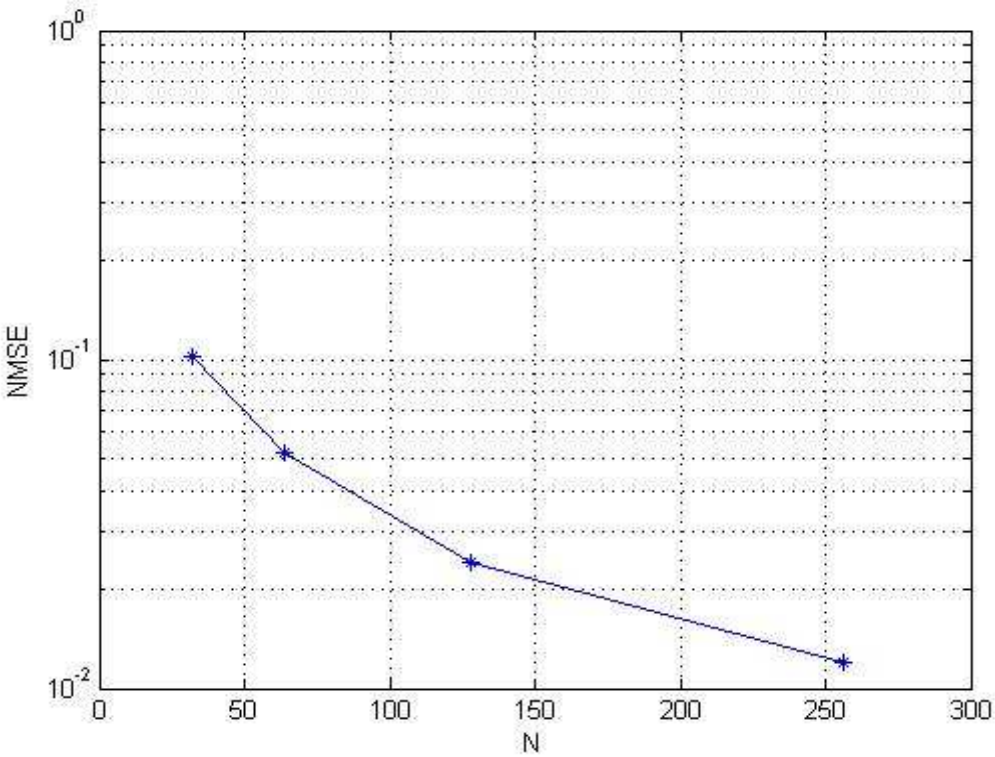

Figura 4.35 NMSE del algoritmo nº4 con respecto al parámetro N

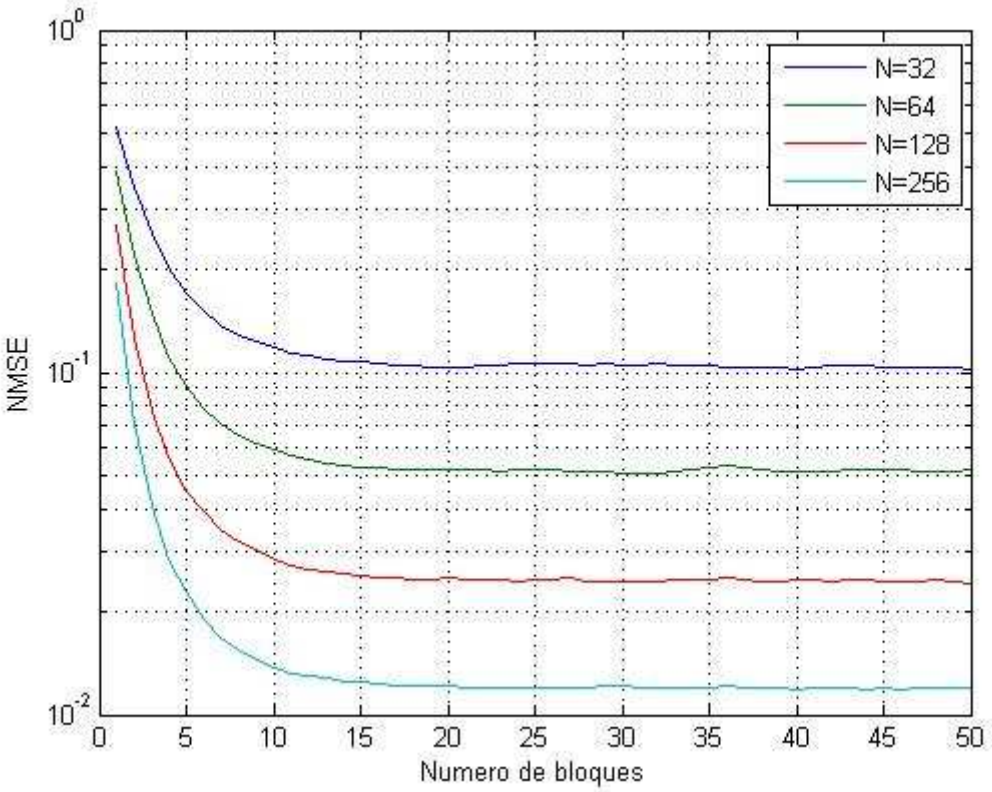

Figura 4.36 NMSE del algoritmo nº4 para distintos valores de N

– Factor de memoria de autocorrelación:

En este apartado observaremos la dependencia del algoritmo respecto al factor de memoria de la autocorrelación, *a*. Para ello realizaremos simulaciones y calcularemos el *NMSE* del algoritmo para distintas *a's*, se ha simulado con valores entre *a*=0.1 y *a*=0.9 con un paso de 0.1.

En la figura 4.37, podemos ver como el *NMSE* tiene una dependencia casi lineal con respecto al índice de autocorrelación, conforme aumentamos este último el *NMSE* aumenta también. Es un parámetro bastante influyente, ya que el resultado puede variar desde *NMSE*=0.3 hasta *NMSE*=0.02.

Por otro lado, si nos fijamos en la figura 4.38, se puede ver que este parámetro no sólo afecta al *NMSE*, sino también a la velocidad de convergencia. Esto es, como no podía ser de otra forma, mientras mejoramos el *NMSE* aumentamos el tiempo en el cuál el algoritmo converge hacia su resultado óptimo. La elección de un valor u otro de este parámetro vendrá definida por las necesidades de convergencia de nuestro sistema. Esta puede variar entre 3 bloques y más de 50 bloques, es decir podría multiplicar casi por 20 el tiempo de convergencia del algoritmo.

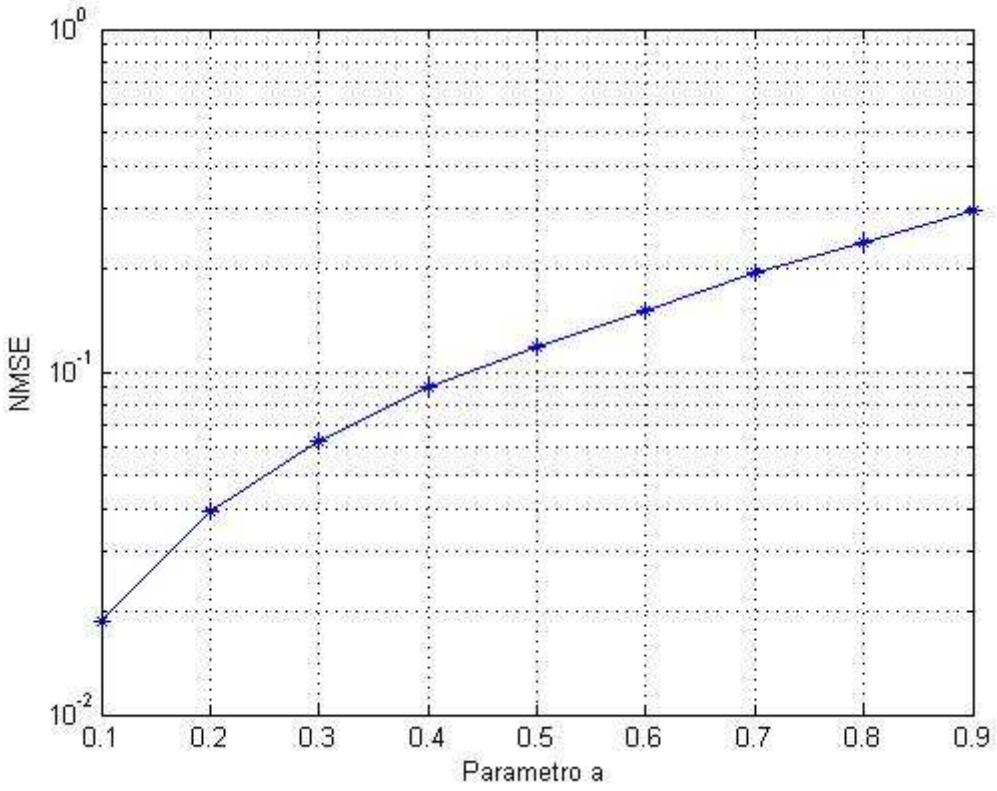

Figura 4.37 NMSE del algoritmo nº4 con respecto al parámetro a

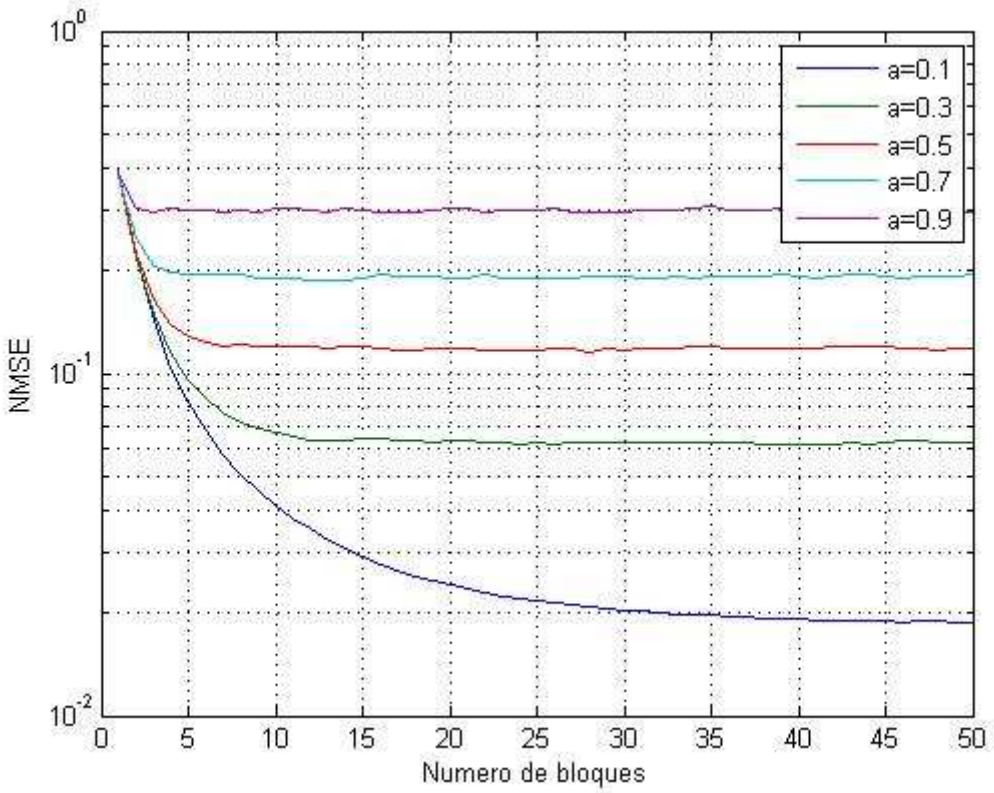

Figura 4.38 NMSE del algoritmo nº4 para distintos valores de a

– Coste computacional:

En primer lugar veremos cuál es el coste del algoritmo: 249.499ms, teniendo en cuenta que el coste original era 5.842ms, por lo tanto tenemos que el coste introducido es: 243.657ms.

4.4 Conclusiones

Este apartado recoge las impresiones más importantes para cada algoritmo, esto es, sus puntos fuertes y débiles así como los resultados de lo que son capaces cada uno de ellos. Además se ofrece una comparativa de los algoritmos permitiendo evaluarlos en un contexto apropiado con otros algoritmos de prestaciones similares.

Estos resultados se hallan en la tabla 4.1, antes de comentar los resultados de cada algoritmo conviene explicar las entradas de la tabla.

- *NMSE* mínimo: indica el mínimo *NMSE* que puede ser obtenido configurando adecuadamente el algoritmo mediante la matriz de precodificación.

- Tiempo de bajada estándar: indica el tiempo de bajada del algoritmo para a=0.25, esto permite comparar la velocidad de convergencia entre algoritmos.

- Tiempo de bajada mínimo: indica el tiempo de bajada mínimo factible para cada algoritmo,aunque optimizar la velocidad de convergencia tendrá efectos negativos sobre la precisión de la estimación del algoritmo.

- Coste Computacional: indica la carga computacional adicional si se introduce el algoritmo, se expresa en milisegundos.

- Ventaja/Inconveniente adicional: aquí figura cualquier ventaja o inconveniente extra que no haya sido expresada en la tabla y sólo incumba al algoritmo en cuestión.

	Algoritmo nº1	**Algoritmo nº2**	**Algoritmo nº3**	**Algoritmo nº4**
NMSE mínimo	10^{-3}	10^{-1}	10^{-3}	10^{-3}
Tiempo de bajada estándar	12 bloques	15 bloques	18 bloques	12 bloques
Tiempo de bajada mínimo	2 bloques	-	10 bloques	2 bloques
Coste comput.	6,706ms	3,624ms	44.047ms	243,657ms
Ventaja adicional	-	-	Relación NMSE/SNR totalmente lineal	Gran dependencia NMSE/N
Inconveniente adicional	-	Muy inefectivo para N>8 portadoras	Uso poco eficiente del ancho de banda de transmisión	-

Tabla 4.1 Comparativa de algoritmos

En la tabla se puede ver como cada algoritmo ofrece unas prestaciones diferentes, todos están balanceados por *NMSE*, velocidad de convergencia o coste computacional. Además, algunos ofrecen características especiales que pueden ser aprovechadas según nuestro sistema. Veamos cada algoritmo por separado:

- El algoritmo nº1 es el más equilibrado de todos, presenta un buen *NMSE* y velocidad de convergencia alta a un coste computacional aceptable sin cargas adicionales.

- El algoritmo nº2 es el que menos coste tiene, no obstante su *NMSE* es bastante pequeño y presenta el inconveniente de ser ineficiente para sistemas con un número elevado de portadoras.

- El algoritmo nº3 presenta un *NMSE* aceptable y coste medio, tiene la ventaja de poder reducir el *NMSE* en sistemas en los que la *SNR* fuese bastante grande. Por contra tenemos una velocidad de convergencia pequeña en comparación a los demás y un uso ineficiente del ancho de banda de transmisión.

- El algoritmo nº4 es presenta unas características similares al primer algoritmo en lo que a *NMSE* y velocidad de convergencia se refieren. A diferencia del algoritmo nº1 tiene la posibilidad de reducir en gran medida el *NMSE* en sistemas con un número de portadoras elevado, no obstante esto requiere un coste computacional bastante grande.