Trabajo Fin de Máster
Máster en electrónica, tratamiento de señal y comunicaciones

Revisión de algoritmos de detección para sistemas MIMO de alto orden

Autor: Irene Santos Velázquez
Tutor: Juan José Murillo Fuentes

Dep. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2014

Trabajo Fin de Máster
Máster en electrónica, tratamiento de señal y comunicaciones

Revisión de algoritmos de detección para sistemas MIMO de alto orden

Autor:
Irene Santos Velázquez

Tutor:
Juan José Murillo Fuentes
Profesor titular

Dep. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2014

Trabajo Fin de Máster: Revisión de algoritmos de detección para sistemas MIMO de alto orden

Autor: Irene Santos Velázquez

Tutor: Juan José Murillo Fuentes

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2014

El Secretario del Tribunal

Resumen

Con este trabajo se pretende comparar varios algoritmos de detección para sistemas de comunicación MIMO y constelaciones de gran orden, dado que la utilización del detector óptimo de máxima verosimilitud (ML) es inabordable computacionalmente. Para este fin, se han creado varias funciones en código matlab que se han publicado como toolbox en mathworks:

http://www.mathworks.com/matlabcentral/fileexchange/48212-detection-mimo-toolbox

Entre los algoritmos de detección destacan el ZF, MMSE y sus versiones con cancelación de interferencias (SIC), las cuales proporcionan una mejora de la probabilidad de error a cambio de una mayor carga computacional. Sin embargo, este trabajo se centra tanto en el algoritmo GTA [1], [2] como en el algoritmo EP [13] (publicada recientemente su aplicación a los sistemas MIMO).

- El algoritmo GTA (aproximación de árbol gaussiana) se basa en aproximar la distribución de probabilidad a posteriori gaussiana exacta mediante un árbol libre de bucles. A continuación, se aplica la restricción de que los símbolos pertenecen a un conjunto finito de datos (determinados por el tipo y tamaño de la constelación) para obtener una aproximación de la distribución de probabilidad discreta libre de bucles, la cual se maximiza mediante un algoritmo de paso de mensajes, i.e, el algoritmo BP.
- El algoritmo EP (expectation propagation) está basado en buscar una aproximación a la distribución de probabilidad a posteriori a través de una distribución gaussiana cuya media y varianza estén lo más próximas posible a la media y la varianza de la distribución exacta original. Esto se consigue a través de un método iterativo.

Los resultados de simulación muestran que la probabilidad de error del algoritmo GTA (en concreto, su versión con cancelación de interfencias) y de EP, son bastante mejores si se comparan con los algoritmos ZF y MMSE, tanto en su versión estándar como SIC.

Abstract

In this work we compare various detection algorithms for MIMO communication systems and high-order constellations, due to optimal solution, i.e; the maximum likelihood (ML) solution is computationally unfeasible. For this purpose, we have done some matlab functions which have been uploaded as a toolbox in mathworks:

http://www.mathworks.com/matlabcentral/fileexchange/48212-detection-mimo-toolbox

We revise zero-forcing (ZF), minimum mean square error (MMSE) algorithms and their versions with successive interference cancelation (SIC), which improve error probability but the complexity of computing increases. However, this work is focused on GTA [1], [2] and EP [13] (recently published its application to MIMO systems) algorithms.

- GTA (Gaussian tree approximation) algorithm is based on approximating the posterior exact Gaussian probability distribution using a cycle-free tree graph. Then, it is applied the restriction of finite-set constraint (determined by the type and size of constellation) to obtain a discrete loop free approximation of probability distribution, which is maximized using a message-passing algorithm, i.e, BP algorithm.
- EP (expectation propagation) algorithm is based on looking for an approximation of the posterior probability distribution, which is a Gaussian distribution whose mean and variance is as near as possible to the mean and variance of the exact distribution. This is an iterative method.

Simulation results show that error probability of GTA (specifically the version with successive interference cancelation) and EP algorithm are much better tan ZF and MMSE algorithms (in standard and SIC version).

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

e	Rama de un árbol de expresión
E_S	Energía media de símbolo
σ^2	Varianza del ruido
$\mathbf{C}$	Matriz de covarianzas
$\mathbf{I}$	Matriz identidad
I(f,g)	Información mutua entre f y g
h(·)	Entropía diferencial
ρ_{ij}	Coeficiente de correlación entre i y j
P_e	Probabilidad de error
$\mathbf{H}$	Matriz de canal
$\mathbf{x}$	Vector columna de símbolos transmitidos
$\mathbf{y}$	Vector columna de la señal recibida
$\mathbf{v}$	Vector columna de ruido
$\hat{\mathbf{x}}$	Aproximación del vector $\mathbf{x}$
$\mathbf{H}^T$	Matriz $\mathbf{H}$ traspuesta
$\mathbf{H}^\dagger$	Matriz pseudoinversa de $\mathbf{H}$
$(\mathbf{H})^{-1}$	Matriz inversa de $\mathbf{H}$
$\mathbf{H}$i	Columna *i* de la matriz $\mathbf{H}$
$\langle \mathbf{H} \rangle_i$	Fila *i* de la matriz $\mathbf{H}$
$\mathbf{H}_{\bar{\mathbf{i}}}$	Matriz que resulta de hacer cero las columnas 1, 2, …, *i* de la matriz $\mathbf{H}$
■	Como queríamos demostrar
$\sum_{\sim\{x\}}(\cdot)$	Operador no suma (o sumario)
x_i	Elemento *i* del vector $\mathbf{x}$
H_{ij}	Elemento de la fila *i* y columna *j* de la matriz $\mathbf{H}$
$\mu_{b\to c}$	Mensaje enviado de 'b' a 'c'
$\mathcal{N}(\mathbf{x}: \boldsymbol{\mu}, \sum)$	Distribución gaussiana multivariable de media $\boldsymbol{\mu}$ y matriz de covarianzas $\sum$
p(x|y)	Distribución de probabilidad a posteriori
D(f ‖ g)	Divergencia KL entre f y g
$\mathcal{O}(\cdot)$	Orden
Q(·)	Operación de cuantización
IRe	Parte real
IIm	Parte imaginaria
\	Excepto

Glosario

MIMO	Multiple input multiple output
ML	Maximum likelihood
ZF	Zero-forcing
MMSE	Minimum mean square error
GTA	Gaussian tree approximation
BP	Belief propagation
EP	Expectation propagation
SIC	Successive interference cancelation
MRF	Markov random field
LDPC	Low density parity check
KL	Kullback-Leibler
SER	Signal error rate
SNR	Signal-to-noise ratio
i.i.d.	Independiente e idénticamente distribuido
e.o.c.	En otro caso
i.e.	Es decir

1 INTRODUCCIÓN

Uno de los problemas en los sistemas de comunicaciones MIMO es encontrar un detector, computacionalmente realizable, que proporcione una probabilidad de error similar a la obtenida con un detector óptimo de máxima verosimilitud (ML).

Existen varios detectores que ofrecen soluciones por debajo de la óptima, como el zero-forcing (ZF) y el MMSE, que son simples de implementar y su complejidad sólo depende del cálculo de una inversa. En 1999 surgió una mejora de dichos métodos conocida como cancelación sucesiva de interferencias [5], basado en la decodificación de forma secuencial con ordenamiento óptimo, i.e, se decodifican primero los símbolos más seguros (los que tienen menor varianza) y en cada iteración se cancela el efecto del símbolo decodificado y se repite el proceso. Esta variante dio lugar a los algoritmos de detección denominados ZF-SIC y MMSE-SIC, que ofrecen una probabilidad de error menor que sus versiones estándar, a cambio de aumentar un orden la complejidad de implementación.

Este trabajo se centra en el estudio de otros algoritmos de detección, en concreto GTA (aproximación de árbol gaussiano) [1]-[2] y EP (expectation propagation) [13], que consiguen probabilidades de error menores que los algoritmos anteriores, y por tanto más cercanas a la óptima, a cambio de aumentar la complejidad computacional. En la Figura 1-1 se representa un ejemplo de la mejora en probabilidad de error que pueden aportar estos algoritmos (en rojo y celeste) respecto a los algoritmos estándar (verde y azul), sin llegar a alcanzar la solución óptima (negro).

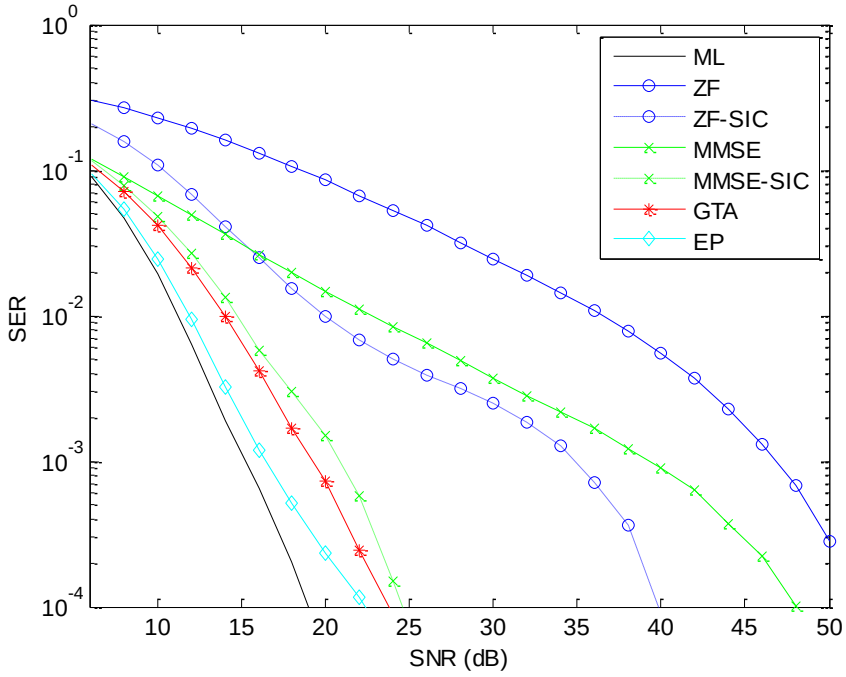

Figura 1-1. Ejemplo de probabilidades de error dadas por distintos algoritmos de detección en canales MIMO.

El algoritmo GTA se basa en encontrar una aproximación a la función de probabilidad exacta gaussiana p(**x**|**y**), donde **x** denota los símbolos transmitidos e **y** la señal recibida. Esta aproximación está asociada al sistema sin restricciones, i.e, aún no se ha tenido en cuenta que los símbolos pertenecen a un conjunto finito de datos (dados por la constelación utilizada). El siguiente paso es aplicar el algoritmo BP, un algoritmo de paso de mensajes. Sin embargo, el grafo de factores asociado a este problema presenta muchos bucles y, en esta situación, el algoritmo BP ofrece unos resultados muy pobres. Por este motivo, primero se busca una aproximación del sistema libre de bucles y se le aplica la restricción de que los símbolos pertenecen a un conjunto finito de datos ($\mathbf{x} \in \mathcal{A}^N$ siendo N el tamaño de la constelación) para obtener una distribución discreta sin bucles de p(**x**|**y**) la cual se maximiza mediante la aplicación del algoritmo BP. De esta forma, se obtiene una aproximación libre de bucles de la función de probabilidad de exacta, que se denotará como $\hat{\mathrm{p}}(\mathbf{x}|\mathbf{y})$. Una vez realizado el paso de mensajes en ambos sentidos, la predicción de la variable se calcula como el producto de todos los mensajes enviados a dicha variable desde su padre y sus hijos (si los tiene), obteniéndose un vector de variables predichas. El algoritmo BP garantiza que dicho vector es, exactamente, la distribución marginal $\hat{\mathrm{p}}(x_i|\mathbf{y})$ de la distribución aproximada $\hat{\mathrm{p}}(\mathbf{x}|\mathbf{y})$ [1]. Por último, el decodificador tendría que elegir el símbolo que proporciona una probabilidad a posteriori $\hat{\mathrm{p}}(x_i|\mathbf{y})$ máxima.

Por otra parte, el algoritmo EP surgió en 2001 de la tesis doctoral de T. Minka [15], pero hasta este año no ha sido aplicado a los sistemas MIMO [13]. Este método consiste en aproximar la distribución a posteriori exacta p(**x**|**y**), mediante la búsqueda de una aproximación de ella q(**x**) formada por un producto de exponenciales $\tilde{t}_i(\mathbf{x})$. Esta aproximación busca, de forma iterativa, que su media y varianza coincidan con las de la distribución original. En concreto, EP va redefiniendo, secuencial e independientemente, los factores $\tilde{t}_i(\mathbf{x})$ de la aproximación quitándolos de la distribución $q(\mathbf{x})$ y asegurando que el producto de todos los factores esté cercano a la distribución original $p(\mathbf{x})$.

1.1 Notación

Sea un sistema de comunicaciones MIMO con *N* antenas transmisoras y *M* antenas receptoras donde $N, M \in \mathcal{N}\backslash\{0\}$. El peso entre la antena transmisora i y la antena receptora j se denota como H_{ij}, siendo H_{ij} el elemento de la fila i y la columna j de la matriz **H**. Para cada matriz de canal **H**, se transmite un vector de símbolos **x**, *N*-dimensional, cuyos elementos (símbolos) pertenecen a un conjunto finito de números reales ($\mathbf{x} \in \mathcal{A}^N$) dados por una constelación *L*-PAM, siendo L el tamaño de dicha constelación. La señal recibida se denota como el vector **y**, M-dimensional, y está dada por la expresión

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v} \tag{1-1}$$

En dicha expresión, el ruido se modela como una vector aleatorio **v**, *M*-dimensional, de valores gaussianos con media 0 y varianza σ^2. La matriz de canal **H**, de dimensiones *M*x*N*, está formada por elementos i.i.d de una distribución normal de varianza unidad. El problema de detección MIMO consiste en encontrar el vector (desconocido) de símbolos transmitidos **x** dado una determinada matriz de canal **H** y señal recibida **y**. Esta estimación encontrada por los diferentes detectores al problema de detección MIMO se denota como $\hat{\mathbf{x}}$.

El modelo mostrado no tiene por qué ser real, aunque en este trabajo sólo se va a trabajar con valores reales. En el Apartado 1.1.1 se muestra una posible solución a este problema en caso de que los valores fueran complejos.

1.1.1 Sistema complejo

En caso de que el sistema sea complejo, se puede reemplazar por su equivalente de valores reales. Para ello, se define el vector **s** de dimensión $n=2N$ y los vectores **r** y **z** de dimensiones $m=2M$, como aquellos vectores formados por la parte real e imaginaria de **x**, **y** y **v**, respectivamente

$$\mathbf{s} = [\mathrm{IRe}(\mathbf{x})^{\mathrm{T}} \quad \mathrm{IIm}(\mathbf{x})^{\mathrm{T}}]^{\mathrm{T}}, \quad \mathbf{r} = [\mathrm{IRe}(\mathbf{y})^{\mathrm{T}} \quad \mathrm{IIm}(\mathbf{y})^{\mathrm{T}}]^{\mathrm{T}}, \quad \mathbf{z} = [\mathrm{IRe}(\mathbf{v})^{\mathrm{T}} \quad \mathrm{IIm}(\mathbf{v})^{\mathrm{T}}]^{\mathrm{T}}$$

Y la matriz de canal de dimensiones *mxn*

$$\begin{bmatrix} \mathrm{IRe}(\mathbf{H}) & \mathrm{IIm}(\mathbf{H}) \\ -\mathrm{IIm}(\mathbf{H}) & \mathrm{IRe}(\mathbf{H}) \end{bmatrix}$$

De esta forma, el equivalente de valores reales al modelo (1-1) está dado por

$$\begin{bmatrix}\mathrm{IRe}(\mathbf{y})\\ \mathrm{IIm}(\mathbf{y})\end{bmatrix}=\begin{bmatrix}\mathrm{IRe}(\mathbf{H}) & \mathrm{IIm}(\mathbf{H})\\ -\mathrm{IIm}(\mathbf{H}) & \mathrm{IRe}(\mathbf{H})\end{bmatrix}\begin{bmatrix}\mathrm{IRe}(\mathbf{x})\\ \mathrm{IIm}(\mathbf{x})\end{bmatrix}+\begin{bmatrix}\mathrm{IRe}(\mathbf{v})\\ \mathrm{IIm}(\mathbf{v})\end{bmatrix} \tag{1-2}$$

En el receptor, un detector obtiene una estimación de los símbolos transmitidos, $\hat{\mathbf{s}}$.

1.2 Revisión de algoritmos de detección

El detector óptimo minimiza la probabilidad media de error, es decir, $\mathrm{P}(\hat{\mathbf{x}} \neq \mathbf{x})$. Esto se consigue con el diseño de máxima verosimilitud (ML – maximum likelihood) el cual proporciona la estimación

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}\in\mathcal{A}^N} \|\mathbf{y}-\mathbf{H}\mathbf{x}\|^2 \tag{1-3}$$

donde $\mathcal{A}^N$ denota la red de posibles símbolos en una constelación L-PAM.

Encontrar la solución exacta al problema (1-3) tiene complejidad NP-completo (NP-hard). Por este motivo, los sistemas de comunicaciones emplean algunas aproximaciones para obtener complejidades computacionales manejables. A continuación se detallan algunas de ellas.

1.2.1 ZF (zero-forcing)

Una solución sub-óptima simple, conocida como el algoritmo zero forcing (o mínimos cuadrados), se basa en una decisión lineal

$$\mathbf{z} = \mathbf{H}^{\dagger}\mathbf{y} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} \tag{1-4}$$

donde $\mathbf{H}^{\dagger}$ denota a la pseudoinversa de $\mathbf{H}$.

Dejando de lado la correlación entre símbolos, se puede encontrar el punto más cercano en $\mathcal{A}$ para cada símbolo de forma independiente

$$\hat{x}_i = \arg\min_{a\in\mathcal{A}} |z_i - a| \tag{1-5}$$

A este estimador se le suele llamar estimador de Babai [11].

La complejidad de este estimador está determinada por la complejidad de encontrar la pseudoinversa de la matriz $\mathbf{H}$ en (1-4). La manera más simple de calcular la pseudoinversa es a través de la factorización QR, $\mathbf{H}$=$\mathbf{QR}$. También puede calcularse de una forma más estable (que evita invertir la matriz triangular superior R) por medio de la descomposición en valores singulares (SVD) de $\mathbf{H}$. En cualquier caso, asumiendo que $\mathbf{H}$ es cuadrada[1] (es decir, N=M), la complejidad de calcular este estimador es de orden cúbico, $\mathcal{O}(N^3)$.

1.2.2 MMSE (minimum mean-square error)

Se puede conseguir un mejor rendimiento utilizando una estimación bayesiana del mínimo error cuadrático medio (MMSE) para el sistema lineal continuo. Sea E_S la energía media de símbolo y σ^2 la varianza del ruido. En ese caso, el estimador MMSE es

$$\mathbf{E}(\mathbf{x}|\mathbf{y}) = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} \tag{1-6}$$

La solución se obtiene encontrando el punto más cercano para cada componente de forma independiente

$$\hat{x}_i = \arg\min_{a\in\mathcal{A}} |\mathrm{E}(\mathbf{x}|y_i) - a| \tag{1-7}$$

[1] Si M<N entonces $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ es singular y no tiene inversa.

1.2.3 Anulación y cancelación

En este método, el estimador se usa sólo sobre uno de los elementos de **x**, por ejemplo el primero. En ese caso, se calcula dicho elemento, x_1 y se cancela su efecto reduciendo el orden del problema de enteros de mínimos cuadrados a *N-1* incógnitas. Este proceso se repite para encontrar x_2, etc.

Para notaciones posteriores, se denotan las columnas y filas de **H** como sigue

$$\mathbf{H} = [\mathbf{H}_1 \quad \mathbf{H}_2 \quad \dots \quad \mathbf{H}_\mathrm{N}] = \begin{bmatrix} \langle \mathbf{H} \rangle_1 \\ \langle \mathbf{H} \rangle_2 \\ \vdots \\ \langle \mathbf{H} \rangle_\mathrm{M} \end{bmatrix}$$

El algoritmo de anulación y cancelación puede implementarse mediante el pseudocódigo de la Figura 1-2, donde Q(·) denota la operación de cuantización apropiada para la constelación en uso.

$\boldsymbol{y_1} \coloneqq \boldsymbol{y}$
$\boldsymbol{for}\ k = 0\ \boldsymbol{to}\ N-1$
  encontrar el vector (columna) de pesos $\boldsymbol{\omega_{N-k}}$
  $\hat{x}_{N-k} \coloneqq \mathrm{Q}\left(\boldsymbol{\omega}_{\mathbf{N-k}}^{\mathrm{T}} \mathbf{y_{k+1}}\right)$
  $\mathbf{y_{k+2}} = \mathbf{y_{k+1}} - \mathbf{H_{N-k}} \hat{x}_{N-k}$
$\boldsymbol{end}$

Figura 1-2. Pseudocódigo del algoritmo de anulación y cancelación.

En el algoritmo, para cada valor del índice *k*, los elementos del vector auxiliar $\mathbf{y_{k+1}}$ son ponderados por las componentes del vector de pesos $\boldsymbol{\omega_{N-k}}$ y combinados linealmente para reducir el efecto de la interferencia. Dependiendo del criterio elegido en el diseño de $\boldsymbol{\omega_{N-k}}$, se pueden distinguir dos casos [6]:

(i) Anulación zero forcing (ZF nulling)

El vector *k*-ésimo de anulación ZF se define como el único vector de norma mínima que satisface

$$\boldsymbol{\omega}_\mathbf{k}^{\mathrm{T}} \mathbf{H}_\mathrm{j} = \begin{cases} 0 & j > k \\ 1 & j = k \end{cases} \tag{1-8}$$

De esta forma, el vector *k*-ésimo de anulación ZF es ortogonal al subespacio formado por las contribuciones de $\mathbf{y_k}$ debida a aquellos símbolos aún no estimados ni cancelados. El único vector que satisface la ecuación (1-8) es la fila *k*-ésima de la matriz $\mathbf{H}^{\dagger}_{\overline{\mathbf{k-1}}}$ donde la notación $\mathbf{H}_{\bar{\mathbf{k}}}$ corresponde a la matriz obtenida tras hacer cero las columnas 1, 2, …, *k* de **H**, es decir

$$\boldsymbol{\omega}_\mathbf{k}^{\mathrm{T}} = \langle \mathbf{H}^{\dagger}_{\overline{\mathbf{k-1}}} \rangle_\mathrm{k} \tag{1-9}$$

(ii) Anulación mínimo error cuadrático medio (MMSE nulling)

El objetivo de la anulación MMSE es minimizar el error cuadrático medio esperado entre la estimación del receptor y el símbolo transmitido.

Además, se asume que la decisión anterior que realiza el detector era correcta, es decir, $\hat{x}_{N-k} = x_{N-k}$. Definiendo

$$\mathbf{x_{1:N-k}} = [x_1 \quad x_2 \quad \dots \quad x_{N-k}]$$

se puede escribir

$$\mathbf{y_{k+1}} = \mathbf{H}_{\mathrm{N-k}} \mathbf{x_{1:N-k}} - \mathbf{v}$$

donde **v** es el vector de ruido de (1-1). Asumiendo que la secuencia de símbolos transmitidos es independiente y de energía E_S, se obtiene como vector de pesos para la anulación MMSE

$$\boldsymbol{\omega}_\mathbf{k}^{\mathrm{T}} = \langle \mathbf{G_k} \rangle_\mathrm{k}$$

siendo

$$\mathbf{G_k} = \left(\mathbf{H}_{\bar{\mathbf{k}}}^{\mathrm{T}} \mathbf{H}_{\bar{\mathbf{k}}} + \frac{\sigma^2}{E_s} \mathbf{I} \right)^{-1} \mathbf{H}_{\bar{\mathbf{k}}}^{\mathrm{T}} \tag{1-10}$$

La complejidad computacional está determinada nuevamente por el cálculo de la pseudo-inversa en cada iteración del algoritmo. Cuando *M=N*, se necesita evaluar la pseudo-inversa de una serie de matrices con dimensiones $N \; x \; (N-k)$, donde *k=N*, *N*-1, …, 1. La complejidad computacional para realizar esta serie de operaciones es de cuarto orden, es decir, $\mathcal{O}(N^4)$, un orden de magnitud mayor que la complejidad de obtener el estimador de Babay.

1.2.4 Anulación y cancelación con ordenamiento óptimo

El algoritmo de anulación y cancelación puede provocar propagación de errores: si x_1 se estima de forma incorrecta puede producir un efecto adverso en la estimación de los aún desconocidos x_2, x_3, etc. Para reducir los efectos de la propagación de errores, es conveniente realizar la anulación y cancelación desde la señal más "fuerte" a la más "débil". Este fue el método propuesto por V-BLAST [5].

Para obtener el orden óptimo, se considera la matriz de convarianzas del error de estimación $\mathbf{x} - \hat{\mathbf{x}}$,

$$\mathbf{C} = \mathbf{E}(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^* = \left(\mathbf{H}^{\mathrm{T}} \mathbf{H} + \frac{\sigma^2}{E_s} \mathbf{I} \right)^{-1} \tag{1-11}$$

La señal más "fuerte", que corresponde a la mejor estimación, es la que tiene menor varianza, es decir, x_i para el cual C_{ii} es el más pequeño. Si se reordenan los elementos de **x** de tal forma que la señal más fuerte es x_m, entonces la estimación $\hat{x}_m$ será la mejor estimación que se pueda obtener para cualquier otro orden de los elementos de **x**. El orden de los elementos de **x** se va obteniendo en cada iteración del algoritmo.

La complejidad computacional del algoritmo es la misma que la complejidad del algoritmo estándar de anulación y cancelación, es decir, $\mathcal{O}(N^4)$, aumentada por la complejidad del cálculo del orden, la cual, para un conjunto de *k* elementos, es $\mathcal{O}(k^3)$.

Este algoritmo de anulación y cancelación con ordenamiento óptimo puede implementarse mediante el código de la Figura 1-3, donde Q(·) denota la operación de cuantización apropiada para la constelación en uso.

$$\begin{aligned}
&\mathbf{y_1} \coloneqq \mathbf{y} \\
&\mathbf{G_1} = \begin{cases} \mathbf{H}^{\dagger} & \text{si ZF} \\ \left(\mathbf{H}^{\mathrm{T}} \mathbf{H} + \frac{\sigma^2}{E_s} \mathbf{I} \right)^{-1} \mathbf{H}^{\mathrm{T}} & \text{si MMSE} \end{cases} \\
&\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ N \\
&\quad k_i = \arg\min_{\mathrm{j} \notin (\mathrm{k}_1, \ldots, \mathrm{k}_{\mathrm{i}-1})} \left\| \langle \mathbf{G_i} \rangle_{\mathrm{j}} \right\|^2 \\
&\quad \boldsymbol{\omega}_{\mathbf{k_i}}^{\mathrm{T}} = \langle \mathbf{G_i} \rangle_{\mathrm{k_i}} \\
&\quad \hat{x}_{k_i} \coloneqq \mathrm{Q}\left(\boldsymbol{\omega}_{\mathbf{k_i}}^{\mathrm{T}} \mathbf{y_i} \right) \\
&\quad \mathbf{y_{i+1}} = \mathbf{y_i} - \mathbf{H}_{\mathrm{k_i}} \hat{x}_{k_i} \\
&\quad \mathbf{G_{i+1}} = \begin{cases} \mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}}^{\dagger} & \text{si ZF} \\ \left(\mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}}^{\mathrm{T}} \mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}} + \frac{\sigma^2}{E_s} \mathbf{I} \right)^{-1} \mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}}^{\mathrm{T}} & \text{si MMSE} \end{cases} \\
&\boldsymbol{end}
\end{aligned}$$

Figura 1-3. Pseudocódigo del algoritmo de anulación y cancelación con ordenamiento óptimo.

1.3 Estructura del trabajo

La memoria del trabajo se estructura de la siguiente manera:

- En el Capítulo 2 se describe una de las variantes del algoritmo BP, conocida como algoritmo Suma-Producto y se muestran varios ejemplos.

- En el Capítulo 3 se representan las distintas formas gráficas por las que se pueden describir un sistema MIMO y se explica la problemática de aplicar el algoritmo BP a un grafo con bucles.
- En el Capítulo 4 se muestra el primer algoritmo objetivo de este trabajo, denominado GTA (gaussian tree aproximation – aproximación de árbol gaussiana), tanto en su versión estándar como su versión con sucesivas cancelaciones de interferencias (GTA-SIC).
- En el Capítulo 5 se muestra el otro algoritmo de estudio en este trabajo, denominado EP (expectation propagation). Primero se describe el método en general y luego su aplicación a los sistemas MIMO.
- En el Capítulo 6 se muestran algunas simulaciones de estos algoritmos y en el Capítulo 7 la estructura de las funciones implementadas para dichas simulaciones.
- En el Capítulo 8, se resumen las conclusiones y líneas futuras de trabajo tras realizar la presente investigación.
- Por último, en el Anexo A se demuestran algunas de las ecuaciones más importantes utilizadas a lo largo de este trabajo. En el Anexo B se deja el pdf explicativo publicado en el toolbox resultado de este trabajo, cuya dirección de internet es:

  http://www.mathworks.com/matlabcentral/fileexchange/48212-detection-mimo-toolbox

  Los Anexos C y D contienen el código Matlab de las funciones realizadas para la simulación y una explicación de ellas, respectivamente.

2 ALGORITMO BP (SUMA-PRODUCTO)

Los algoritmos que tratan con funciones globales complicadas de varias variables suelen buscar la forma de expresar los factores de las funciones dadas como un producto de funciones "locales", donde cada una de ellas depende de un subconjunto de variables. Esta factorización se puede visualizar como un grafo bipartito que se denomina factor graph (o grafo de factores).

En este capítulo se presenta un algoritmo genérico de paso de mensajes, el algoritmo suma-producto o algoritmo BP [3], que opera en un grafo de factores. Siguiendo una única regla, simple computacionalmente, este algoritmo calcula (de forma exacta o aproximada) todas las funciones marginales que se derivan de la función global. En el caso de grafos sin bucles, el algoritmo siempre converge a la solución exacta mientras que, para grafos con bucles es posible que no converja o, en caso contrario, puede no ser exacta la solución.

2.1 Notación utilizada

A lo largo de este capítulo se utilizarán funciones de varias variables. Sea $x_1, x_2, \dots, x_N$ un conjunto de variables en el que, para cada i, x_i toma valores de un cierto dominio (o alfabeto) $\mathcal{A}_i$. Sea $g(x_1, x_2, \dots, x_N)$ una función $\mathbb{Z}$-evaluada de estas variables, es decir, una función de dominio

$$\mathcal{S} = \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_N$$

y codominio $\mathbb{Z}$. El dominio $\mathcal{S}$ de g se llama espacio de configuración para el conjunto de variables dadas, y cada elementos de $\mathcal{S}$ es una configuración concreta de estas variables, es decir, una asignación de un valor a cada variable. A lo largo del capítulo se supondrá que $\mathbb{Z}$ es el conjunto de números enteros.

Se supone que la suma en $\mathbb{Z}$ es bien definida y que hay *N* funciones marginales $g_i(x_i)$ asociadas a cada función $g(x_1, x_2, \dots, x_N)$. Para cada $a \in \mathcal{A}_i$, el valor de $g_i(a)$ se obtiene sumando el valor de $g(x_1, x_2, \dots, x_N)$ sobre todas las configuraciones de las variables que tienen $x_i = a$.

Debida a la importancia de este tipo de sumas, se introduce una notación no estándar denominada "no-suma" o sumario[2]. En vez de indicar las variables sobre las que se suma, se indican aquellas variables sobre las que *no* se suma. Por ejemplo, si h es una función de tres variables x_1, x_2 y x_3, entonces el "sumario para x_2" se denota como

$$\sum_{\sim\{x_2\}} h(x_1, x_2, x_3) \coloneqq \sum_{x_1 \in \mathcal{A}_1} \sum_{x_3 \in \mathcal{A}_3} h(x_1, x_2, x_3)$$

En esta notación se tiene

$$g_i(x_i) \coloneqq \sum_{\sim\{x_i\}} g(x_1, \dots, x_N) \tag{2-1}$$

[2] Se denomina como operador sumario (en inglés, summary) aquel operador que indica las variables sobre las que *no* se suma [3].

Es decir, la i-ésima función marginal asociada con $g(x_1, \ldots, x_N)$ es el sumario para x_i de g.

2.2 Grafo de factores (o factor graph)

Supóngase que $g(x_1, x_2, \ldots, x_N)$ se puede factorizar como un producto de varias funciones locales, cada una de ellas con argumentos pertenecientes a algún subconjunto de $\{x_1, x_2, \ldots, x_N\}$, es decir

$$g(x_1, x_2, \ldots, x_N) = \prod_{j \in \mathcal{J}} f_j(\mathcal{X}_j) \tag{2-2}$$

donde $\mathcal{J}$ es un conjunto discreto de índices, $\mathcal{X}_j$ es un subconjunto de $\{x_1, x_2, \ldots, x_N\}$ y $f_j(\mathcal{X}_j)$ es una función que tiene como argumentos los elementos de $\mathcal{X}_j$.

Los grafos de factores permiten representar ese tipo de expresiones (2-2) de forma eficiente. Son grafos bipartitos en los que hay un nodo variable (variable node) para cada variable x_i, un nodo factor para cada función local f_j y una arista (o rama) que conecta el nodo variable x_i al nodo factor f_j si y solo si x_i es un argumento de f_j. En resumen, un grafo de factores es un gráfico bipartito que representa una relación matemática, en este caso, la relación "es un argumento de" entre variables y funciones locales.

2.2.1 Ejemplo de grafo de factores

Sea $g(x_1, x_2, x_3, x_4, x_5)$ una función de cinco variables que puede ser expresada como el siguiente producto de cinco factores

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5) \tag{2-3}$$

de tal forma que $\mathcal{J} = \{A, B, C, D, E\}$, $\mathcal{X}_A = \{x_1\}$, $\mathcal{X}_B = \{x_2\}$, $\mathcal{X}_C = \{x_1, x_2, x_3\}$, $\mathcal{X}_D = \{x_3, x_4\}$ y $\mathcal{X}_E = \{x_3, x_5\}$. El grafo de factores correspondiente a (2-3) se muestra en la Figura 2-1.

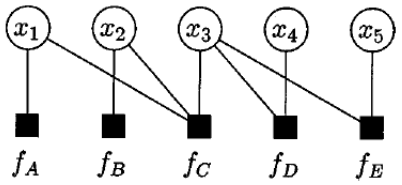

Figura 2-1. Grafo de factores del producto $f_A(x_1)f_B(x_2)f_C(x_1,x_2,x_3)f_D(x_3,x_4)f_E(x_3,x_5)$.

Este grafo de factores también se puede expresar como un árbol que tiene por nodo raíz algún x_i ($i=1, \ldots, 5$). En la Figura 2-2 se muestra una forma de representar el grafo de factores de la Figura 2-1 como un árbol que tiene por nodos raíces x_1 y x_3, respectivamente. Cuando un grafo de factores está libre de bucles, dicho grafo de factores no sólo codifica en su estructura la factorización de la función global, sino también las expresiones aritméticas mediante las cuales se pueden calcular las funciones marginales asociadas a la función global.

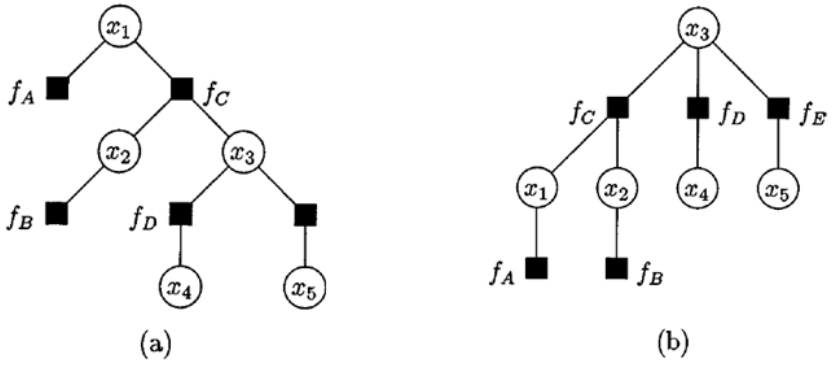

Figura 2-2. Grafo de factores de la Figura 2-1, redibujado como un árbol con nodo raíz: (a) x_1 y (b) x_3.

2.3 Árboles de expresión ("Expression Trees")

En muchas situaciones (por ejemplo, cuando $g(x_1, x_2, x_3, x_4, x_5)$ representa un función de probabilidad conjunta) interesa calcular las funciones marginales de $g_i(x_i)$. Se puede obtener una expresión de cada función marginal usando (2-3) y explotando la propiedad distributiva.

Para ilustrar lo anterior, se puede escribir $g_1(x_1)$ del ejemplo del Apartado 2.2.1 como

$$g_1(x_1) = f_A(x_1)\left(\sum_{x_2} f_B(x_2)\left(\sum_{x_3} f_C(x_1, x_2, x_3)\left(\sum_{x_4} f_D(x_3, x_4)\right)\left(\sum_{x_5} f_E(x_3, x_5)\right)\right)\right)$$

o, en notación sumaria (es decir, partiendo de la definición (2-1))

$$\begin{aligned} g_1(x_1) &= \sum_{\sim\{x_1\}} g(x_1, x_2, x_3, x_4, x_5) \\ &= f_A(x_1) \\ &\quad \times \sum_{\sim\{x_1\}}\left(f_B(x_2) f_C(x_1, x_2, x_3) \times \left(\sum_{\sim\{x_3\}} f_D(x_3, x_4)\right) \times \left(\sum_{\sim\{x_3\}} f_E(x_3, x_5)\right)\right) \end{aligned} \tag{2-4}$$

De forma similar, se puede encontrar que

$$g_3(x_3) = \left(\sum_{\sim\{x_3\}} f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3)\right) \times \left(\sum_{\sim\{x_3\}} f_D(x_3, x_4)\right) \times \left(\sum_{\sim\{x_3\}} f_E(x_3, x_5)\right) \tag{2-5}$$

Las expresiones matemáticas como las del segundo miembro de las ecuaciones (2-4) y (2-5) se suelen representar por árboles de expresión, en los que los vértices interiores (es decir, vértices con descendientes) representan a los operadores aritméticos (sumas, multiplicaciones, negaciones, etc) y los vértices extremos (es decir, vértices sin descendientes) representan a las funciones, variables o constantes. En la Figura 2-3 se representan, sin ambigüedades, el segundo miembro de las ecuaciones (2-4) y (2-5), respectivamente. Los operadores mostrados en estas figuras son productos de funciones y sumarios, que tienen como argumentos varias funciones locales.

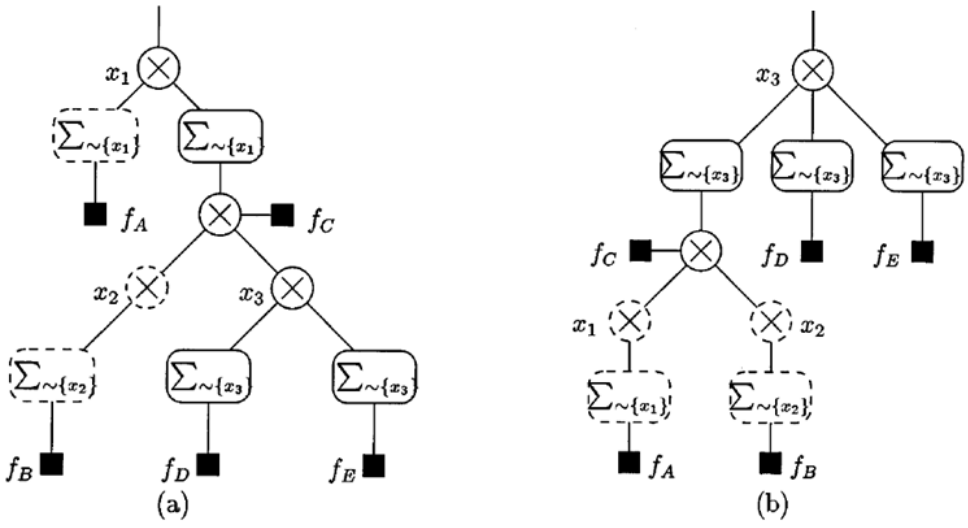

Figura 2-3. Representación en árbol de expresión del segundo miembro de las ecuaciones: (a) (2-4) y (b) (2-5).

2.3.1 Conversión de grafo de factores a árbol de expresión

Para convertir un grafo de factores sin bucles que representa una función $g(x_1, \dots, x_N)$ en su correspondiente árbol de expresión para $g_i(x_i)\ \forall i$, se dibuja el grafo de factores como un árbol con nodo raíz x_i, tal y como se representa en la Figura 2-2. De esta forma, cada nodo v del grafo de factores tiene claramente definido un nodo padre. A continuación, se reemplaza cada nodo variable del grafo de factores por un operador producto. También se reemplaza cada nodo factor del grafo de factores por un operador "producto multiplicado además por f", y entre el nodo factor f y su padre x, se inserta un operador sumario $\sum_{\sim\{x\}}$. Estas transformaciones locales se muestran en la Figura 2-4(a) para un nodo variable y en la Figura 2-4(b) para un nodo factor f con su padre x.

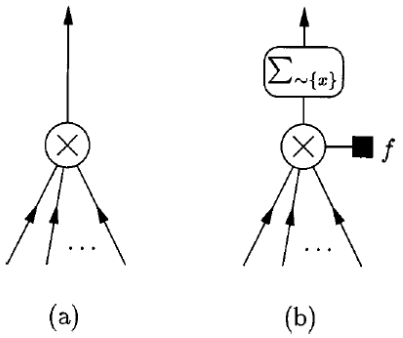

Figura 2-4. Sustituciones locales que transforman una función marginal (a) y un nodo factor (b) de un grafo de factores sin bucles en un árbol de expresión.

Los productos triviales (aquellos con uno o ningún operando) actúan como operadores identidad, o pueden omitirse si son nodos extremos en el árbol de expresión. Un operador sumario $\sum_{\sim\{x\}}$ aplicado a una función con un único argumento x es también una operación trivial y puede omitirse. Aplicando esta transformación a los grafos de factores de la Figura 2-2 se consigue el árbol de expresión de la Figura 2-3 (los operadores triviales se indican en esta figura con líneas discontinuas). También puede observarse en la Figura 2-5.

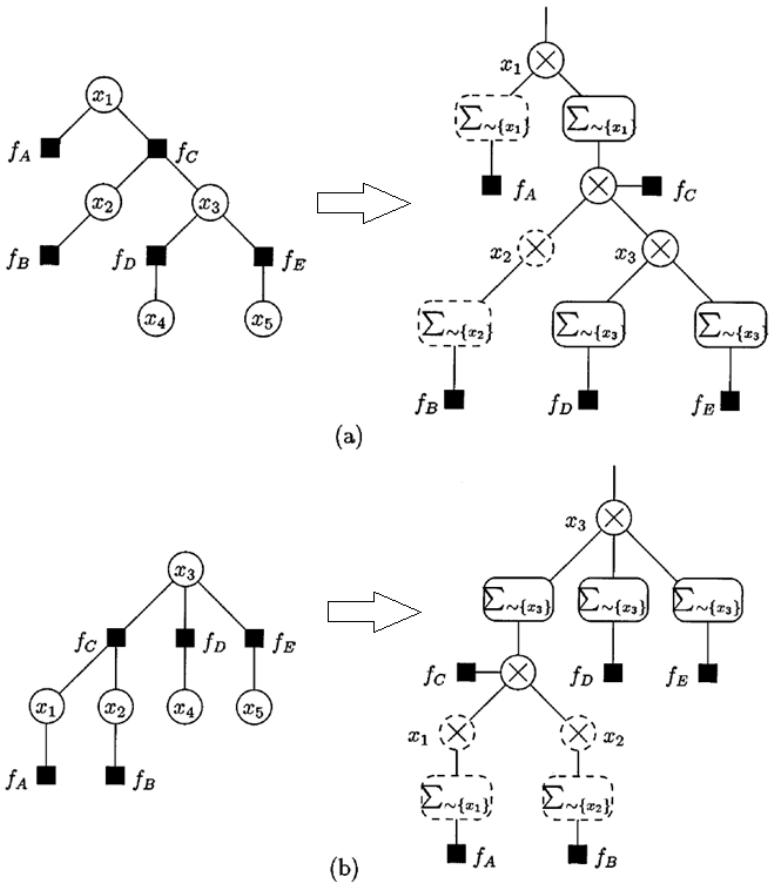

Figura 2-5. Conversión de los grafos de factores de la Figura 2-2 a sus respectivos árboles de expresión.

2.4 Cálculo de una función marginal

Cada árbol de expresión representa un algoritmo para calcular la expresión correspondiente. Este algoritmo se suele describir como un procedimiento "de abajo a arriba" que comienza en los extremos del árbol, con cada vértice operador combinando sus operandos y transmitiendo el resultado como un operando a su padre.

En vez de trabajar con el árbol de expresión, es más simple y directo describir dicho algoritmo de marginalización en términos de su correspondiente grafo de factores. Para entender mejor el algoritmo, es útil imaginar que las ramas (aristas) del grafo de factores representan canales por los que se comunican los vértices. Los "mensajes" enviados entre vértices son simplemente descripciones apropiadas de alguna función marginal.

A continuación se describe el algoritmo de paso de mensajes denominado, temporalmente, "algoritmo suma-producto para un único i", ya que calcula, para un único valor de i, la función marginal $g_i(x_i)$ en un grafo de factores sin bucles, con nodo raíz x_i.

El cálculo comienza en los extremos (u hojas) del grafo de factores. Cada nodo variable extremo envía un mensaje trivial ("función identidad") a su padre, y cada nodo factor extremo f envía una descripción de f a su padre. Cada vértice espera a que le lleguen los mensajes de todos sus hijos antes de calcular el mensaje que tiene que enviar a su padre. Este cálculo se realiza de acuerdo a las transformaciones que se muestran en la Figura 2-4, es decir, un nodo variable simplemente envía el producto de los mensajes recibidos desde sus hijos, mientras que un nodo factor f de padre x realiza el producto de f con los mensajes recibidos desde sus hijos, y luego aplica el operador sumario $\sum_{\sim\{x\}}$ al resultado.

El cálculo termina en el nodo raíz x_i, donde se obtiene la función marginal $g_i(x_i)$ como el producto de todos los mensajes recibidos en x_i.

Es importante recalcar que un mensaje que pasa por la rama $\{x, f\}$, ya sea desde la variable x al factor f, o viceversa, es una función de un único argumento de x, la variable asociada a dicha rama. Esto es así ya que, en cada nodo factor, las operaciones sumario se realizan sobre la variable asociada con la rama por la que se pasa el mensaje. Del mismo modo, en un nodo variable, todos los mensajes son funciones de esa variable, al igual que cualquier producto de estos mensajes.

El mensaje que se pasa por una rama durante la operación del algoritmo suma-producto para un único i puede interpretarse como sigue. Si $e = \{x, f\}$ es una rama del árbol, donde x es un nodo variable y f un nodo factor, entonces el mensaje que pasa por e durante la operación del algoritmo suma-producto es simplemente un sumario de x del producto de las funciones locales descendientes del vértice que origina el mensaje.

2.5 Cálculo de todas las funciones marginales

Muchas veces lo que interesa es calcular $g_i(x_i)$ para más de un valor de i. Este cálculo podría realizarse aplicando el algoritmo para un único i para cada valor deseado de i por separado, pero este enfoque no es eficiente ya que muchos de los cálculos intermedios realizados para cada valor de i serán los mismos. El cálculo de $g_i(x_i)$ para todos los i simultáneamente puede lograrse de forma eficiente por "superposición" en un único grafo de factores de todos los casos del algoritmo para cada i. No se toma ningún vértice en particular como raíz, por lo que no hay una relación fija padre/hijo entre los vértices vecinos. En vez de eso, cada vecino w de cualquier vértice dado v es, en algún momento, considerado como padre de v. El mensaje enviado desde v a w se calcula cómo en el algoritmo para un único i, es decir, como si w fuera el padre de v y todos los demás vecinos de v fueran sus hijos.

Como en el algoritmo para un único i, el paso de mensajes se inicializa en los extremos. Cada vértice v permanece inactivo hasta que le llegan los mensajes de todas las ramas incidentes en v excepto una (que se entiende como su padre de forma temporal). Al igual que en el algoritmo para un único i, una vez que llegan estos mensajes, v es capaz de calcular un mensaje que envía a su vértice vecino restante, considerado temporalmente como su padre, como en el algoritmo para un único i. Por comodidad, se denotará al padre temporal como el vértice w. Después de enviar el mensaje a w, el vértice v vuelve al estado inactivo, esperando un "mensaje de respuesta" procedente de w. Una vez que llega este mensaje, el vértice es capaz de calcular y enviar mensajes a cada uno de sus vecinos (excepto a w), siendo considerados, a su vez, cada uno de ellos como un padre. El algoritmo termina una vez que han pasado dos mensajes (uno en cada dirección) por

cada una de las ramas. En el nodo variable x_i, el producto de todos los mensajes que le llegan es la función marginal $g_i(x_i)$, al igual que en el algoritmo para un único i. Como en este algoritmo es necesario el cálculo de varias sumas y productos, se le denomina algoritmo suma-producto.

Regla (Algoritmo suma-producto): El mensaje enviado desde un nodo v por una rama e es el producto de la función local en v (o la función unidad si v es un nodo variable) con todos los mensajes recibidos en v por otras ramas distintas a e, sumadas (con el operador sumario) sobre la variable asociada con e.

Sea $\mu_{x\to f}(x)$ el mensaje enviado desde el nodo x a el nodo f en el algoritmo suma-producto y sea $\mu_{f\to x}(x)$ el mensaje enviado desde el nodo f al nodo x. Sea también $n(v)$ el conjunto de vecinos de un nodo v en un grafo de factores. Entonces, como se muestra en la Figura 2-6, los mensajes que se calculan (e intercambian) en el algoritmo suma-producto se pueden expresar como sigue

- De variable a función local

$$\mu_{x\to f}(x) = \prod_{h\in n(x)\setminus\{f\}} \mu_{h\to x}(x) \tag{2-6}$$

- De función local a variable

$$\mu_{f\to x}(x) = \sum_{\sim\{x\}} \left(f(\mathcal{X}) \prod_{y\in n(f)\setminus\{x\}} \mu_{y\to f}(y) \right) \tag{2-7}$$

donde $\mathcal{X} = n(f)$ es el conjunto de argumentos de la función f.

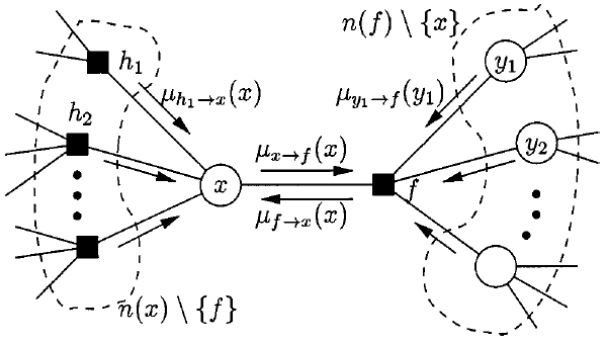

Figura 2-6. Fragmento de un grafo de factores que muestra las reglas del algoritmo suma-producto.

El mensaje desde un nodo variable x toma una forma simple dada por la ecuación (2-6) porque no hay ninguna función local que incluir. Por otro lado, el mensaje desde un nodo función local, dado por la ecuación (2-7), en general implica multiplicaciones de funciones no triviales, seguidas de un operador sumario.

2.6 Ejemplo detallado

La Figura 2-7 muestra el flujo de mensajes que se genera en el algoritmo suma-producto aplicado al grafo de factores de la Figura 2-1.

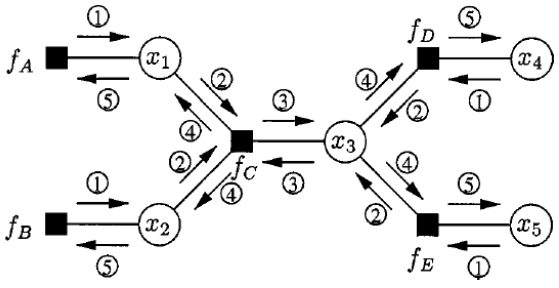

Figura 2-7. Mensajes generados en cada paso del algoritmo suma-producto.

Los mensajes se generan en cinco pasos, que se muestran a continuación de forma detallada.

- Paso 1

$$\mu_{f_A \to x_1}(x_1) = \sum_{\sim\{x_1\}} f_A(x_1) = f_A(x_1)$$
$$\mu_{f_B \to x_2}(x_2) = \sum_{\sim\{x_2\}} f_B(x_2) = f_B(x_2)$$
$$\mu_{x_4 \to f_D}(x_4) = 1$$
$$\mu_{x_5 \to f_E}(x_5) = 1$$

- Paso 2

$$\mu_{x_1 \to f_C}(x_1) = \mu_{f_A \to x_1}(x_1)$$
$$\mu_{x_2 \to f_C}(x_2) = \mu_{f_B \to x_2}(x_2)$$
$$\mu_{f_D \to x_3}(x_3) = \sum_{\sim\{x_3\}} \mu_{x_4 \to f_D}(x_4) f_D(x_3, x_4)$$
$$\mu_{f_E \to x_3}(x_3) = \sum_{\sim\{x_3\}} \mu_{x_5 \to f_E}(x_5) f_E(x_3, x_5)$$

- Paso 3

$$\mu_{f_C \to x_3}(x_3) = \sum_{\sim\{x_3\}} \mu_{x_1 \to f_C}(x_1) \mu_{x_2 \to f_C}(x_2) f_C(x_1, x_2, x_3)$$
$$\mu_{x_3 \to f_C}(x_3) = \mu_{f_D \to x_3}(x_3) \mu_{f_E \to x_3}(x_3)$$

- Paso 4

$$\mu_{f_C \to x_1}(x_1) = \sum_{\sim\{x_1\}} \mu_{x_3 \to f_C}(x_3) \mu_{x_2 \to f_C}(x_2) f_C(x_1, x_2, x_3)$$
$$\mu_{f_C \to x_2}(x_2) = \sum_{\sim\{x_2\}} \mu_{x_3 \to f_C}(x_3) \mu_{x_1 \to f_C}(x_1) f_C(x_1, x_2, x_3)$$
$$\mu_{x_3 \to f_D}(x_3) = \mu_{f_C \to x_3}(x_3) \mu_{f_E \to x_3}(x_3)$$
$$\mu_{x_3 \to f_E}(x_3) = \mu_{f_C \to x_3}(x_3) \mu_{f_D \to x_3}(x_3)$$

- Paso 5

$$\mu_{x_1 \to f_A}(x_1) = \mu_{f_C \to x_1}(x_1)$$
$$\mu_{x_2 \to f_B}(x_2) = \mu_{f_C \to x_2}(x_2)$$
$$\mu_{f_D \to x_4}(x_4) = \sum_{\sim\{x_4\}} \mu_{x_3 \to f_D}(x_3) f_D(x_3, x_4)$$
$$\mu_{f_E \to x_5}(x_5) = \sum_{\sim\{x_5\}} \mu_{x_3 \to f_E}(x_3) f_E(x_3, x_5)$$

- Último paso

$$g_1(x_1) = \mu_{f_A \to x_1}(x_1) \mu_{f_C \to x_1}(x_1)$$

$$g_2(x_2) = \mu_{f_B \to x_2}(x_2)\mu_{f_C \to x_2}(x_2)$$
$$g_3(x_3) = \mu_{f_C \to x_3}(x_3)\mu_{f_D \to x_3}(x_3)\mu_{f_E \to x_3}(x_3)$$
$$g_4(x_4) = \mu_{f_D \to x_4}(x_4)$$
$$g_5(x_5) = \mu_{f_E \to x_5}(x_5)$$

En el último paso, se calcula $g_i(x_i)$ como el producto de todos mensajes dirigidos a x_i.

Este algoritmo BP converge a la solución y es exacto si el grafo de factores no tiene bucles. En caso de tener bucles, el algoritmo no tiene por qué converger, y si converge, puede no hacerlo a la solución.

3 MODELO GRÁFICO DEL SISTEMA

Una vez visto como funciona el algoritmo BP para obtener funciones marginales en grafos, en este capítulo se va a expresar el sistema (1-1) de dos maneras distintas [1], [2]: como un grafo MRF y como un grafo de factores, mostrando cómo se realiza el paso de mensajes y las desventajas de cada uno de ellos.

3.1 Representación como grafo MRF

Dado el sistema lineal $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v}$, y una distribución previa uniforme de $\mathbf{x}$ sobre un conjunto finito de puntos $\mathcal{A}$, la función de probabilidad a posteriori del vector aleatorio discreto $\mathbf{x}$ dado $\mathbf{y}$ es

$$p(\mathbf{x}|\mathbf{y}) \propto \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{H}\mathbf{x}-\mathbf{y}\|^2\right) \qquad \mathbf{x} \in \mathcal{A}^N \tag{3-1}$$

Como $\|\mathbf{H}\mathbf{x}-\mathbf{y}\|^2$ es una expresión cuadrática, $p(\mathbf{x}|\mathbf{y})$ puede factorizarse como un producto de dos expresiones

$$p(x_1, \dots, x_n|\mathbf{y}) \propto \prod_i \psi_i(x_i) \prod_{i<j} \psi_{ij}(x_i, x_j) \tag{3-2}$$

tal que

$$\psi_i(x_i) = \exp\left(-\frac{1}{2\sigma^2}\mathbf{y}^T\mathbf{H}_i x_i\right) \tag{3-3}$$

$$\psi_{ij}(x_i, x_j) = \exp\left(-\frac{1}{\sigma^2}\mathbf{H}_i{}^T\mathbf{H}_j x_i x_j\right) \tag{3-4}$$

Donde $\mathbf{H}_i$ es la i-ésima columna de la matriz $\mathbf{H}$. Como los factores obtenidos son funciones de pares de variables aleatorias, se obtiene una representación de campo aleatorio de Markov (MRF – Markov random field). En los sistemas MIMO, la matriz (conocida) $\mathbf{H}$ es aleatoria por lo que el grafo MRF suele ser un grafo totalmente conectado, como se muestra en la Figura 3-1.

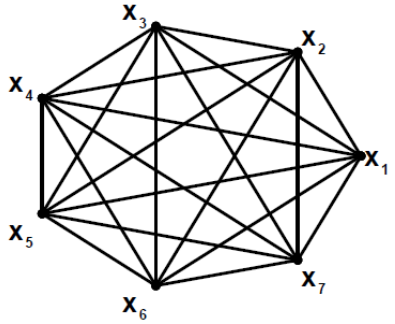

Figura 3-1. Grafo MRF correspondiente al problema de detección MIMO para N=7.

El algoritmo BP trata de resolver los problemas de inferencia mediante la propagación de información a través de este grafo MRF, es decir, mediante una serie de mensajes enviados entre los nodos vecinos. Al aplicar el algoritmo BP (suma-producto) al grafo MRF (3-2), el mensaje de x_j a x_i es

$$m_{j\to i}(x_i) = \sum_{x_j \in \mathcal{A}} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\neq i,j} m_{k\to j}(x_j) \qquad x_i \in \mathcal{A} \tag{3-5}$$

En cada iteración, se puede calcular una estimación de la distribución marginal posterior (que se denomina "belief") para cada variable, multiplicando todos los mensajes entrantes desde todos los demás nodos

$$b_i(x_i) = \psi_i(x_i) \prod_{k\neq i} m_{k\to i}(x_i) \qquad x_i \in \mathcal{A} \tag{3-6}$$

Una variante del algoritmo suma-producto es el algoritmo máximo-producto en el que el sumatorio de (3-5) se reemplaza por una maximización sobre todos los símbolos de $\mathcal{A}$. En un grafo MRF sin bucles el algoritmo BP (tanto la variante suma-producto como máximo-producto) siempre converge a las probabilidades marginales exactas (que en el caso de detección MIMO corresponde a $p(x_i|\mathbf{y})$). Para grafos sin bucles, únicamente es necesario el paso de mensajes entre nodos vecinos. Computacionalmente, es sencillo aplicar esta misma regla de intercambio de mensajes en grafos con bucles. Sin embargo, en la mayoría de estos modelos, el algoritmo BP con bucles no calcula la distribución marginal exacta, por lo que no hay ninguna justificación teórica para aplicar este algoritmo.

Aun así, se ha comprobado que el algoritmo BP aplicado a grafos con bucles tiene éxito en algunas aplicaciones, por ejemplo, en decodificación de códigos LDPC (dado que los grafos son dispersos, i.e, los bucles de los grafos son largos y comparten las propiedades de un árbol ya que los mensajes son prácticamente independientes). Sin embargo, dado que en la mayoría de modelos los grafos MRF de canales MIMO son grafos completamente conectados y el rendimiento de detección asociado es pobre, el algoritmo BP no es una buena opción para el problema MIMO.

Como método alternativo, existe el algoritmo BP generalizado (GBP – generalized belief propagation) que funciona bien con matrices dispersas si las regiones del algoritmo se elijen cuidadosamente. Sin embargo, en el modelo de canal MIMO se asume que los elementos de la matriz de canal son independientes, idénticamente distribuidos y gaussianos, por lo que no se puede asumir que la matriz de canal **H** sea dispersa.

3.2 Representación como grafo de factores

Existe también la posibilidad de expresar el problema de detección MIMO (3-1) como un grafo de factores. Se puede verificar que[3]

$$p(\mathbf{x}|\mathbf{y}) \propto \prod_i p(y_i|\mathbf{x}) \propto \prod_i \exp\left(-\frac{1}{2\sigma^2}(\langle\mathbf{H}\rangle_i\mathbf{x} - y_i)^2\right) \qquad \mathbf{x} \in \mathcal{A}^N \tag{3-7}$$

siendo $\langle\mathbf{H}\rangle_i$ la *i*-ésima fila de la matriz **H**. Cada factor corresponde a una única ecuación lineal y por tanto cada factor es una función de todas las variables desconocidas. En la Figura 3-2 se muestra el grafo de factores resultante. Las variables del grafo de factores $x_1, x_2, \ldots, x_N$ corresponden a las antenas transmisoras y los factores $y_1, y_2, \ldots, y_M$ corresponden a las antenas receptoras.

[3] La demostración de (3-7) se encuentra en el Apartado A.1.

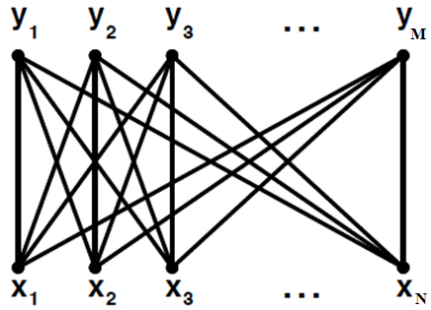

Figura 3-2. Grafo de factores de un sistema de detección MIMO.

Hay que recalcar que la representación anterior del grafo de factores es un grafo bipartito completamente conectado. Aplicando el algoritmo BP al grafo de factores de la Figura 3-2 se obtienen los siguientes mensajes:

- Mensaje BP desde el factor asociado con y_j a la variable x_i

$$m_{j\to i}(a) = \sum_{\mathbf{x}|x_i=a} \exp\left(-\frac{1}{2\sigma^2}\left(\langle \mathbf{H}\rangle_j \mathbf{x} - y_j\right)^2\right) \prod_{k\neq i} m_{k\to j}(x_k) \qquad a \in \mathcal{A} \tag{3-8}$$

- Mensaje BP desde la variable x_i al factor y_j

$$m_{i\to j}(a) = \prod_{k\neq j} m_{k\to i}(a) \qquad a \in \mathcal{A} \tag{3-9}$$

Un inconveniente importante de aplicar BP al grafo de factores (3-7) es la complejidad de calcular los mensajes BP desde los factores a las variables (3-8), la cual es exponencial respecto al número de variables N.

4 ALGORITMO GTA

El algoritmo GTA trata de encontrar una aproximación de la función de probabilidad exacta[4] de un vector discreto aleatorio **x** dado **y**, que permita implementar con éxito el algoritmo BP [1]-[2] y [12]. Esta función de probabilidad exacta es de la forma

$$p(\mathbf{x}|\mathbf{y}) \propto \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{H}\mathbf{x}-\mathbf{y}\|^2\right) \qquad \mathbf{x} \in \mathcal{A}^N \tag{4-1}$$

Como se ha visto en el anterior capítulo, el algoritmo BP es óptimo en grafos de factores sin bucles (árboles), por lo que la idea es encontrar una aproximación en árbol óptima de la distribución exacta (4-1).

Dada una distribución f(**x**), el método de Chow-Liu permite encontrar una aproximación en árbol de dicha distribución, g(**x**), cuya divergencia de Kullback-Leibler (KL) es mínima respecto a la distribución dada f(**x**) [8]. Este método permite encontrar el árbol óptimo, que es el árbol de expansión (spanning tree) de mayor peso en el grafo completo de *N* vértices, donde el peso de cada rama está dado por la información mutua de las dos variables aleatorias asociadas con los extremos de la rama. El problema del algoritmo de Chow-Liu es que está basado en el producto de distribuciones marginales de segundo orden y encontrar la distribución marginal de la función de probabilidad (4-1) tiene complejidad NP-completo.

Para solventar este problema, el algoritmo GTA se basa en aplicar el algoritmo de Chow-Liu a la distribución sin restricciones[5]

$$p(\mathbf{x}|\mathbf{y}) \propto \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{H}\mathbf{x}-\mathbf{y}\|^2\right) \qquad \mathbf{x} \in \mathcal{R}^N \tag{4-2}$$

Esta distribución es gaussiana y, por tanto, resulta sencillo calcular las distribuciones marginales de segundo orden. Dada la aproximación en árbol gaussiana, el siguiente paso es aplicar la restricción de conjunto finito para obtener una aproximación discreta sin bucles de $p(\mathbf{x}|\mathbf{y})$, la cual puede maximizarse de forma eficiente usando el algoritmo BP.

Ejemplo 4–1. *Obtención del ZF a partir de una aproximación gaussiana*[6].

En este ejemplo, se va a obtener la solución ZF vista en (1-5) a partir de la distribución gaussiana (4-1). Esta distribución (4-1) puede expresarse como una gaussiana[7] de media $\mathbf{z}(\mathbf{y}) = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y}$ (es decir, el estimador de mínimos cuadrados de (1-4)) y matriz de covarianzas $\mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}$

[4] Se dice que la distribución es exacta porque en ella se tiene en cuenta que los símbolos pertenecen a un conjunto finito de datos ($x \in \mathcal{A}^N$), dados por el tipo y tamaño de la constelación.
[5] La distribución sin restricciones es aquella que no tiene en cuenta que los símbolos pertenecen a un conjunto finito de datos, i.e. $x \in \mathcal{R}^N$.
[6] Este ejemplo es una versión simplificada de la aproximación en árbol gaussiana utilizada en el Apartado 4.1.2.2.
[7] La demostración de la ecuación (4-3) se encuentra en el Apartado A.2.

$$p(\mathbf{x}|\mathbf{y}) \propto f(\mathbf{x};\mathbf{z},\mathbf{C}) = \frac{1}{\sqrt{(2\pi)^N|\mathbf{C}|}}\exp\left(-\frac{1}{2}(\mathbf{z}-\mathbf{x})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{z}-\mathbf{x})\right) \qquad \mathbf{x}\in\mathcal{A}^N \tag{4-3}$$

donde $f(\mathbf{x};\mathbf{z},\mathbf{C})$ es una densidad gaussiana con media $\mathbf{z}$ y matriz de covarianzas $\mathbf{C}$. Hay que destacar que (4-3) sólo está definida para $M \geq N$ ya que, en otro caso, la matriz $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ es singular y $\mathbf{C}$ es indefinida. A continuación, en vez de marginalizar la distribución exacta $p(\mathbf{x}|\mathbf{y})$ (4-3) que tiene complejidad NP-completo, se aproxima por el producto de las marginales de la densidad gaussiana $f(\mathbf{x};\mathbf{z},\mathbf{C})$

$$f(\mathbf{x};\mathbf{z},\mathbf{C}) \approx \prod_i f(x_i;z_i,C_{ii}) = \prod_i \frac{1}{\sqrt{2\pi C_{ii}}}\exp\left(-\frac{(z_i-x_i)^2}{2C_{ii}}\right) \tag{4-4}$$

A continuación se aplica la restricción de conjunto finito. De la aproximación gaussiana (4-4) se puede extraer la aproximación discreta

$$\hat{p}(\mathbf{x}|\mathbf{y}) \propto \prod_i \exp\left(-\frac{(z_i-x_i)^2}{2C_{ii}}\right) \qquad \mathbf{x}\in\mathcal{A}^N \tag{4-5}$$

Dado que esta función de probabilidad conjunta se obtiene como un producto de probabilidades marginales, se puede decodificar cada variable por separado

$$\hat{p}(x_i=a|\mathbf{y}) \propto \exp\left(-\frac{(z_i-a)^2}{2C_{ii}}\right) \qquad a\in\mathcal{A} \tag{4-6}$$

Cogiendo el símbolo más probable se obtiene la solución sub-óptima ZF (1-5).

$$\begin{aligned}\hat{x}_i = \arg\max_{a\in\mathcal{A}}\left(\hat{p}(x_i=a|\mathbf{y})\right) &= \arg\max_{a\in\mathcal{A}}\left(\exp\left(-\frac{(z_i-a)^2}{2C_{ii}}\right)\right) = \arg\min_{a\in\mathcal{A}}\left(\frac{(z_i-a)^2}{2C_{ii}}\right)\\ &= \arg\min_{a\in\mathcal{A}}((z_i-a)^2) = \arg\min_{a\in\mathcal{A}}|z_i-a|\end{aligned} \tag{4-7}$$

En el siguiente apartado se explica con detalle el algoritmo GTA paso a paso.

4.1 Pasos a seguir

4.1.1 Búsqueda de la aproximación gaussiana en árbol óptima

Sea $\{p(i)\}_{i=1}^N$ el conjunto de todos los padres de un árbol sin bucles de *N* nodos donde $p(i)$ es el padre del nodo i. Para simplificar la notación no se va a describir por separado al nodo raíz, sino que se asume que el padre del nodo raíz es el conjunto vacío. Una distribución $g(x_1,\ldots,x_N)$ representada por un árbol $\{p(i)\}$ se puede escribir como $g(\mathbf{x}) = \prod_{i=1}^N g\left(x_i|x_{p(i)}\right)$. Lo primero es definir una fórmula para la divergencia KL entre una distribución gaussiana $f(\mathbf{x})$ y una distribución $g(\mathbf{x})$ que representa un árbol y está definida en el mismo espacio.

Teorema 4.1

Sea $f(\mathbf{x}) = f(x_1,\ldots,x_N)$ una distribución gaussiana de múltiples variables y sea $g(\mathbf{x}) = \prod_{i=1}^N g\left(x_i|x_{p(i)}\right)$ otra distribución representada por un árbol (sin bucles). En ese caso, la divergencia KL entre f y g es

$$D(f \parallel g) = \sum_{i=1}^N D\left(f\left(x_i|x_{p(i)}\right) \parallel g\left(x_i|x_{p(i)}\right)\right) - h(\mathbf{x}) + \sum_{i=1}^N \left(h(x_i) - I\left(x_i;x_{p(i)}\right)\right) \tag{4-8}$$

donde I es la información mutua y h es la entropía diferencial, basada en la distribución $f(\mathbf{x})$.

De la ecuación (4-8) se extrae que, si se tiene un árbol concreto $\{p(i)\}_{i=1}^{N}$, la distribución del árbol cuya divergencia KL de $f(\mathbf{x})$ es mínima, es $g(\mathbf{x}) = \prod_{i=1}^{N} f\left(x_i|x_{p(i)}\right)$, es decir, la mejor aproximación del árbol de $f(\mathbf{x})$ se construye a partir de las distribuciones condicionales de $f(\mathbf{x})$. Para esa aproximación del árbol se tiene

$$D\left(f \parallel \prod_{i=1}^{N} f\left(x_i|x_{p(i)}\right)\right) = \sum_{i=1}^{N} h(x_i) - h(\mathbf{x}) - \sum_{i=1}^{N} I\left(x_i; x_{p(i)}\right) \tag{4-9}$$

Demostración:

$$\begin{aligned}
\mathrm{D}(f(\mathbf{x}) \parallel g(\mathbf{x})) &= \mathrm{D}\left(f(\mathbf{x}) \parallel \prod_{i=1}^{N} g\left(x_i|x_{p(i)}\right)\right) = \int f(\mathbf{x}) \log f(\mathbf{x})\, d\mathbf{x} - \int f(\mathbf{x}) \log \prod_{i=1}^{N} g\left(x_i|x_{p(i)}\right) d\mathbf{x} \\
&= -h(\mathbf{x}) - \int f(\mathbf{x}) \sum_{i=1}^{N} \log g\left(x_i|x_{p(i)}\right) d\mathbf{x} \\
&= -h(\mathbf{x}) - \sum_{i=1}^{N} \iint f(x_i, x_{p(i)}) \log g\left(x_i|x_{p(i)}\right) dx_i\, dx_{p(i)} \\
&= -h(\mathbf{x}) + \sum_{i=1}^{N} \iint f(x_i, x_{p(i)}) \log \left(\frac{1}{g\left(x_i|x_{p(i)}\right)} \frac{f\left(x_i|x_{p(i)}\right)}{f\left(x_i|x_{p(i)}\right)}\right) dx_i\, dx_{p(i)} \\
&= -h(\mathbf{x}) + \sum_{i=1}^{N} \iint f(x_i, x_{p(i)}) \log \left(\frac{f\left(x_i|x_{p(i)}\right)}{g\left(x_i|x_{p(i)}\right)}\right) dx_i dx_{p(i)} \\
&\quad - \sum_{i=1}^{N} \iint f(x_i, x_{p(i)}) \log f\left(x_i|x_{p(i)}\right) dx_i dx_{p(i)} \\
&= -h(\mathbf{x}) + \sum_{i=1}^{N} \mathrm{D}\left(f\left(x_i|x_{p(i)}\right) \parallel g\left(x_i|x_{p(i)}\right)\right) \\
&\quad - \sum_{i=1}^{N} \iint f(x_i, x_{p(i)}) \log f\left(x_i|x_{p(i)}\right) dx_i dx_{p(i)} \overset{Ec\ 1.111\ de\ [16]}{=} \\
&= -h(\mathbf{x}) + \sum_{i=1}^{N} \mathrm{D}\left(f\left(x_i|x_{p(i)}\right) \parallel g\left(x_i|x_{p(i)}\right)\right) + \sum_{i=1}^{N} h\left(x_i|x_{p(i)}\right) \\
&= -h(\mathbf{x}) + \sum_{i=1}^{N} \mathrm{D}\left(f\left(x_i|x_{p(i)}\right) \parallel g\left(x_i|x_{p(i)}\right)\right) - \sum_{i=1}^{N} \left(I\left(x_i; x_{p(i)}\right) - h(x_i)\right)
\end{aligned}$$

■

Además, en la ecuación anterior tanto $h(\mathbf{x})$ como $h(x_i)$ no dependen de la estructura del árbol, por lo que la topología del árbol $\{p(i)\}$ que mejor aproxima $f(\mathbf{x})$ es aquella que maximiza la suma

$$\sum_{i=1}^{N} I\left(x_i; x_{p(i)}\right) \tag{4-10}$$

Un árbol de expansión (spanning tree) de un grafo dado es un subgrafo que contiene todos los vértices y es un árbol. Suponiendo que las ramas del grafo tienen pesos entonces, el peso de un árbol de expansión es simplemente la suma de los pesos de sus ramas. La ecuación (4-10) revela que el problema de encontrar la mejor aproximación de árbol de la distribución gaussiana $f(\mathbf{x})$ se reduce a encontrar el máximo árbol de expansión del grafo con *N*-nodos donde el peso de la rama i-j es la información mutua entre x_i y x_j [8]. La información mutua entre dos variables aleatorias conjuntamente gaussianas x_i y x_j es

$$I(x_i, x_j) = -\log(1 - \rho_{ij}^2) \tag{4-11}$$

donde ρ_{ij} es el coeficiente de correlación entre x_i y x_j.

Existen varios algoritmos para encontrar un árbol de expansión máximo. En este trabajo de utilizará el algoritmo de Prim [9], el cual es eficiente y muy fácil de implementar. El algoritmo de Prim comienza con algunos vértices v de un grafo dado, definiendo el conjunto inicial de vértices T. Luego, en cada iteración, se elije la rama de peso máximo de entre todas ramas (u, v), donde u está fuera de T y v dentro de T. A continuación el vértice u se incluye en T. Este proceso se repite hasta que se forma el árbol de expansión. La complejidad del algoritmo de Prim para encontrar el árbol de expansión de peso máximo de un grafo de N vértices es $\mathcal{O}(N^2)$.

Como el algoritmo de Prim sólo necesita el orden de los pesos y no sus valores concretos, para encontrar la aproximación gaussiana óptima del árbol se pueden usar los pesos ρ_{ij}^2 en vez de $I(x_i, x_j) = -\log(1 - \rho_{ij}^2)$. De esta forma, el árbol gaussiano óptimo es el único que maximiza la suma de los coeficientes de correlación al cuadrado entre nodos adyacentes. En resumen, el algoritmo que encuentra la mejor aproximación de árbol gaussiana consiste en lo siguiente. Se define x_1 como raíz. A continuación se busca la rama que conecta un vértice del interior del árbol con un vértice que está fuera del árbol, tal que su coeficiente de correlación al cuadrado sea el mayor y se añade dicha rama al árbol. Este procedimiento continúa hasta que se obtiene el árbol de expansión.

4.1.2 Aplicación de BP en la aproximación de árbol

Sea $\hat{f}(\mathbf{x})$ la aproximación Chow-Liu óptima de árbol de cualquier aproximación gaussiana $f(\mathbf{x}; \mathbf{z}, \mathbf{C})$ con media $\mathbf{z}$ y matriz de covarianzas $\mathbf{C}$, es decir

$$f(\mathbf{x}; \mathbf{z}, \mathbf{C}) = \frac{1}{\sqrt{(2\pi)^N |\mathbf{C}|}} \exp\left(-\frac{1}{2}(\mathbf{z} - \mathbf{x})^{\mathrm{T}} \mathbf{C}^{-1} (\mathbf{z} - \mathbf{x})\right) \qquad \mathbf{x} \in \mathcal{R}^N \tag{4-12}$$

Se puede asumir, sin pérdida de generalidad, que $\hat{f}(\mathbf{x})$ tiene x_1 como raíz. Por tanto, $\hat{f}(\mathbf{x})$ es una distribución gaussiana sin bucles en $x_1, \ldots, x_N$ y con nodo raíz x_1, es decir

$$\hat{f}(\mathbf{x}) = f(x_1; \mathbf{z}, \mathbf{C}) \prod_{i=2}^{N} f(x_i | x_{p(i)}; \mathbf{z}, \mathbf{C}) \qquad \mathbf{x} \in \mathcal{R}^N \tag{4-13}$$

donde $p(i)$ es el padre del nodo i en el árbol óptimo.

El algoritmo de Chow-Liu garantiza que $\hat{f}(\mathbf{x})$ es la aproximación óptima gaussiana de árbol de $f(\mathbf{x}; \mathbf{z}, \mathbf{C})$ en el sentido de que la divergencia KL $D(f \parallel \hat{f})$ es mínima.

Dada la aproximación de árbol gaussiana, el siguiente paso es aplicar la restricción de conjunto finito para obtener una aproximación discreta sin bucles de $p(\mathbf{x}|\mathbf{y})$, la cual puede maximizarse de forma eficiente usando el algoritmo BP, es decir, se busca aproximar la distribución exacta $p(\mathbf{x}|\mathbf{y})$ por la aproximación

$$\hat{p}(\mathbf{x}|\mathbf{y}) \propto \hat{f}(\mathbf{x}) = f(x_1; \mathbf{z}, \mathbf{C}) \prod_{i=2}^{N} f(x_i | x_{p(i)}; \mathbf{z}, \mathbf{C}) \qquad \mathbf{x} \in \mathcal{A}^N \tag{4-14}$$

A continuación se describe como se realiza esta maximización. Como la función de probabilidad $\hat{p}(\mathbf{x}|\mathbf{y})$ es un grafo de factores sin bucles, se puede aplicar el algoritmo BP para encontrar la configuración más probable. Para aplicar BP de forma óptima, es necesario que por cada vértice pasen dos mensajes, uno en cada sentido [3]. Los mensajes BP se envían primero desde las variables de los extremos hacia la variable raíz ("hacia abajo"), es decir, el cálculo comienza en los extremos del grafo. Cada nodo extremo envía un mensaje a su padre. Cada uno de los vértices espera a que le lleguen los mensajes de todos sus hijos antes de calcular el mensaje hacia su padre. El mensaje BP "hacia abajo" de una variable x_i a su variable padre $x_{p(i)}$ se calcula teniendo en cuenta todos los mensajes que x_i recibe de sus hijos

$$m_{i\to p(i)}\left(x_{p(i)}\right)=\sum_{x_i\in\mathcal{A}} f\left(x_i|x_{p(i)};\mathbf{z},\mathbf{C}\right)\prod_{j\,|p(j)=i} m_{j\to i}(x_i) \qquad x_{p(i)}\in\mathcal{A} \tag{4-15}$$

Si x_i es un nodo extremo del árbol entonces el mensaje anterior se simplifica

$$m_{i\to p(i)}\left(x_{p(i)}\right)=\sum_{x_i\in\mathcal{A}} f\left(x_i|x_{p(i)};\mathbf{z},\mathbf{C}\right) \qquad x_{p(i)}\in\mathcal{A} \tag{4-16}$$

El cálculo "hacia abajo" termina en el nodo raíz.

A continuación, se envían mensajes BP de vuelta a los extremos ("hacia arriba"). El cálculo comienza en el nodo raíz del grafo. Cada vértice espera un mensaje de su padre antes de calcular el mensaje que debe enviar a cada uno de sus hijos. El mensaje BP "hacia arriba" desde una variable padre $x_{p(i)}$ a su variable hijo x_i se calcula teniendo en cuenta el mensaje "hacia arriba" que $x_{p(i)}$ recibe de su padre $x_{p(p(i))}$ y el mensaje "hacia abajo" que $x_{p(i)}$ recibe de todos los hermanos de x_i (es decir, de todos los hijos de $x_{p(i)}$ distintos a x_i)

$$\begin{aligned} &m_{p(i)\to i}(x_i) \\ &=\sum_{x_{p(i)}\in\mathcal{A}} f\left(x_i|x_{p(i)};\mathbf{z},\mathbf{C}\right)m_{p(p(i))\to p(i)}\left(x_{p(i)}\right)\prod_{\{j\,|j\neq i,p(j)=p(i)\}} m_{j\to p(i)}\left(x_{p(i)}\right) \qquad x_i\in\mathcal{A} \end{aligned} \tag{4-17}$$

Si $x_{p(i)}$ es el nodo raíz del árbol entonces el mensaje anterior se simplifica

$$m_{p(i)\to i}(x_i)=\sum_{x_{p(i)}\in\mathcal{A}} f\left(x_i|x_{p(i)};\mathbf{z},\mathbf{C}\right)\prod_{\{j\,|j\neq i,p(j)=p(i)\}} m_{j\to p(i)}\left(x_{p(i)}\right) \qquad x_i\in\mathcal{A} \tag{4-18}$$

Una vez que este procedimiento de paso de mensajes "hacia abajo" y "hacia arriba" finaliza, se puede calcular cada variable "predicha" ("belief") como el producto de todos los mensajes enviados a la variable desde su padre y desde sus hijos (si los tiene)

$$belief_i(x_i)=m_{p(i)\to i}(x_i)\prod_{j\,|p(j)=i} m_{j\to i}(x_i) \qquad x_i\in\mathcal{A} \tag{4-19}$$

En el caso de que x_i sea el nodo raíz, la variable "predicha" se calcula como sigue

$$belief_i(x_i)=f(x_i;\mathbf{z},\mathbf{C})\prod_{j\,|p(j)=i} m_{j\to i}(x_i) \qquad x_i\in\mathcal{A} \tag{4-20}$$

Como la distribución aproximada $\hat{p}(\mathbf{x}|\mathbf{y})$ (4-14) no tiene bucles, la teoría general de BP garantiza que el vector "predicho" es exactamente la distribución marginal $\hat{p}(x_i|\mathbf{y})$ de la distribución aproximada $\hat{p}(\mathbf{x}|\mathbf{y})$ (4-14). Para obtener el valor decodificado, se toma el símbolo cuya probabilidad posterior es máxima

$$\hat{x}_i=\arg\max_a belief_i(a) \qquad a\in\mathcal{A} \tag{4-21}$$

Aunque sólo se ha descrito la versión suma-producto del algoritmo BP que calcula las probabilidades marginales $\hat{p}(x_i|\mathbf{y})$ existe otra variante de esta versión, conocida como algoritmo máximo-producto, en la cual se sustituiría el sumatorio de (4-15)-(4-18) por una maximización sobre todos los símbolos de $\mathcal{A}$.

4.1.2.1. Versión ZF de una aproximación en árbol

Para obtener una versión ZF (1-4) del algoritmo GTA, basta con expresar la aproximación gaussiana (4-12) como una distribución gaussiana con media $\mathbf{z}=\left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y}$ y matriz de covarianzas $\mathbf{C}=\sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}$. Dado que en la solución ZF la matriz $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ es singular para $M<N$ (y por tanto no se puede aplicar la aproximación de árbol gaussiana), lo más conveniente es utilizar una versión MMSE (explicada en el siguiente

apartado).

4.1.2.2. Versión MMSE de una aproximación en árbol

Como ya se ha comentado, la utilización de una detección bayesiana MMSE (1-6) es mejor que la solución ZF (1-4) ya que incorpora información sobre el ruido. En este caso, la aproximación gaussiana (4-12) se expresa como

$$f_{(\mathbf{x}|\mathbf{y})}(\mathbf{x}|\mathbf{y}) = \frac{1}{\sqrt{(2\pi)^N|\mathbf{C}|}}\exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{z})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{x}-\mathbf{z})\right) \tag{4-22}$$

donde $\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \sigma^2/E_s\,\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y}$ y $\mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \sigma^2/E_s\,\mathbf{I}\right)^{-1}$.

Aplicando la aproximación Chow-Liu de árbol a la distribución gaussiana en (4-22) se obtendría la aproximación 'bayesiana' de árbol gaussiano de $p(\mathbf{x}|\mathbf{y})$. Aplicando la restricción de que los símbolos pertencen a un conjunto finito de datos, se obtiene la aproximación discreta de la distribución $p(\mathbf{x}|\mathbf{y})$, que se maximiza mediante el algoritmo BP. Al contrario que ocurría con la versión ZF, si $M < N$ la solución MMSE y la versión MMSE de la aproximación de árbol gaussiana sí son válidas.

En resumen, la solución al problema de decodificación MIMO estudiada en este capítulo se basa en aplicar el algoritmo BP en una versión discreta de la aproximación de árbol gaussiana de la versión bayesiana de la solución continua de mínimos cuadrados. Este método se denomina "algoritmo de aproximación de árbol gaussiana (GTA)". La complejidad del cálculo de la matriz de covarianzas $\sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \sigma^2/E_s\mathbf{I}\right)^{-1}$ es $\mathcal{O}(N^2M)$. La complejidad del algoritmo de Chow-Liu (basado en el algoritmo de Prim para encontrar el árbol de expansión máximo) es $\mathcal{O}(N^2)$ y la complejidad del algoritmo BP $\mathcal{O}(|\mathcal{A}|^2N)$ (independiente del número de antenas receptoras).

4.2 Resumen del algoritmo GTA

El algoritmo GTA (versión MMSE) se resume en la Figura 4-1.

- Entradas:
  Un sistema lineal $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v}$, un nivel de ruido σ^2 y un conjunto finito de símbolos $\mathcal{A}$ cuya energía media de símbolo es E_s
- Algoritmo:
  - Se calculan
    $$\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} \quad , \quad \mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}$$
  - Se denota
    $$f(x_i;\mathbf{z},\mathbf{C}) = \exp\left(-\frac{1}{2}\frac{(x_i-z_i)^2}{C_{ii}}\right) \quad , \quad f(x_i|x_j;\mathbf{z},\mathbf{C}) = \exp\left(-\frac{1}{2}\frac{\left((x_i-z_i)-\frac{C_{ij}}{C_{jj}}(x_j-z_j)\right)^2}{C_{ii}-\frac{C_{ij}^2}{C_{jj}}}\right)$$
  - Se calcula el árbol de expansión máximo del grafo de *N* nodos donde el peso de la rama i-j es el cuadrado del coeficiente de correlación: $\rho_{ij}^2 = C_{ij}^2/\left(C_{ii}\,C_{jj}\right)$
  - Asumiendo que el árbol tiene como raíz el nodo '1' y denotando al padre del nodo i como $p(i)$, se aplica el algoritmo BP a la distribución sin bucles
    $$\hat{p}(x_1,\ldots,x_N|\mathbf{y}) \propto f(x_1;\mathbf{z},\mathbf{C})\prod_{i=2}^{N} f\left(x_i|x_{p(i)};\mathbf{z},\mathbf{C}\right)$$
    Para encontrar la (aproximación a la) configuración más probable.

Figura 4-1. Resumen del algoritmo GTA.

4.3 Ejemplo numérico

4.3.1 Datos

Sea una constelación BPSK (es decir, $\mathbf{x} \in \mathcal{A} = \{-1,1\}$) con *N*=6 antenas transmisoras y *M*=6 antenas receptoras.

La energía de una señal BPSK es la unidad, es decir, $E_s = 1$.

Se supone también una varianza del ruido de $\sigma^2 = 6e^{-5}$ (que equivale a una relación señal a ruido de SNR=50dB para una señal con energía unidad y 6 antenas transmisoras).

Sea la matriz de canal (de distribución normal y varianza unidad) y el vector recibido siguientes:

$$\mathbf{H} = \begin{bmatrix} 0.73 & 1.41 & 0.49 & 0.89 & 0.33 & 0.32 \\ -0.06 & 1.42 & 1.03 & -1.15 & -0.75 & 0.31 \\ 0.71 & 0.67 & 0.73 & -1.07 & 1.37 & -0.86 \\ -0.21 & -1.21 & -0.3 & -0.81 & -1.71 & -0.03 \\ -0.12 & 0.72 & 0.29 & -2.94 & -0.1 & -0.16 \\ 1.49 & 1.63 & -0.79 & 1.44 & -0.24 & 0.63 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 2.54 \\ -1.89 \\ 1.85 \\ -3.58 \\ -2.59 \\ 4.5 \end{bmatrix}$$

Los símbolos reales transmitidos son:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

4.3.2 Obtención del máximo árbol de expresión (versión MMSE)

Se calcula la matriz de covarianzas

$$\mathbf{C} = \sigma^2 \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I} \right)^{-1} = \begin{bmatrix} 0.0011 & -0.0011 & 0.0008 & -0.0004 & 0.0006 & 0.0022 \\ -0.0011 & 0.0011 & -0.0007 & 0.0004 & -0.0006 & -0.0022 \\ 0.0008 & -0.0007 & 0.0006 & -0.0003 & 0.0004 & 0.0015 \\ -0.0004 & 0.0004 & -0.0003 & 0.0002 & -0.0002 & -0.0008 \\ 0.0006 & -0.0006 & 0.0004 & -0.0002 & 0.0004 & 0.0013 \\ 0.0022 & -0.0022 & 0.0015 & -0.0008 & 0.0013 & 0.0045 \end{bmatrix}$$

Los coeficientes de correlación son

$$\rho_{ij}^2 = \frac{C_{ij}^2}{C_{ii}\,C_{jj}} \quad \Rightarrow \quad \begin{bmatrix} \rho_{11}^2 & \rho_{12}^2 & \rho_{13}^2 & \rho_{14}^2 & \rho_{15}^2 & \rho_{16}^2 \\ \rho_{21}^2 & \rho_{22}^2 & \rho_{23}^2 & \rho_{24}^2 & \rho_{25}^2 & \rho_{26}^2 \\ \rho_{31}^2 & \rho_{32}^2 & \rho_{33}^2 & \rho_{34}^2 & \rho_{35}^2 & \rho_{36}^2 \\ \rho_{41}^2 & \rho_{42}^2 & \rho_{43}^2 & \rho_{44}^2 & \rho_{45}^2 & \rho_{46}^2 \\ \rho_{51}^2 & \rho_{52}^2 & \rho_{53}^2 & \rho_{54}^2 & \rho_{55}^2 & \rho_{56}^2 \\ \rho_{61}^2 & \rho_{62}^2 & \rho_{63}^2 & \rho_{64}^2 & \rho_{65}^2 & \rho_{66}^2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0.9761 & 0.9152 & 0.9082 & 0.8620 & 0.9300 \\ 0.9761 & 1 & 0.9311 & 0.9307 & 0.9198 & 0.9721 \\ 0.9152 & 0.9311 & 1 & 0.7953 & 0.8002 & 0.8874 \\ 0.9082 & 0.9307 & 0.7953 & 1 & 0.9339 & 0.9554 \\ 0.8620 & 0.9198 & 0.8002 & 0.9339 & 1 & 0.9600 \\ 0.9300 & 0.9721 & 0.8874 & 0.9554 & 0.9600 & 1 \end{bmatrix}$$

Se toma como nodo raíz el nodo ‘1’ y se añade al árbol

$$árbol = \{1\}$$

A continuación comienza el procedimiento iterativo para calcular el árbol.

- Iteración 1: se busca la rama de mayor peso que tiene al nodo '1' en uno de sus extremos y se añade el nodo que conecta el otro extremo de la rama al árbol

$$\max\left(\rho_{12}^2, \rho_{13}^2, \rho_{14}^2, \rho_{15}^2, \rho_{16}^2\right) = \rho_{12}^2 = 0.9761 \quad \Rightarrow \quad árbol = \{1,2\}$$

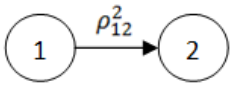

Figura 4-2. Árbol de expansión formado en la iteración 1.

Se eliminan todas las ramas entre los nodos que hay actualmente en el árbol, es decir, habría que eliminar los coeficientes ρ_{12}^2, ρ_{21}^2 (ya que estas ramas no pueden volver a incluirse en el árbol, para evitar bucles)

- Iteración 2: se busca la siguiente rama de mayor peso que tenga en uno de sus extremos a algunos de los nodos del árbol formado hasta ahora (es decir, al nodo '1' o '2') y se añade al árbol

$$\max\left(\rho_{13}^2, \rho_{14}^2, \rho_{15}^2, \rho_{16}^2, \rho_{23}^2, \rho_{24}^2, \rho_{25}^2, \rho_{26}^2\right) = \rho_{26}^2 = 0.9721 \quad \Rightarrow \quad árbol = \{1,2,6\}$$

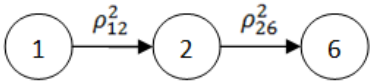

Figura 4-3. Árbol de expansión formado en la iteración 2.

Se eliminan todas las ramas entre los nodos que hay actualmente en el árbol, es decir, habría que eliminar los coeficientes $\rho_{12}^2, \rho_{21}^2, \rho_{16}^2, \rho_{61}^2, \rho_{26}^2, \rho_{62}^2$(ya que estas ramas no pueden volver a incluirse en el árbol, para evitar bucles)

- Iteración 3: se busca la siguiente rama de mayor peso que tenga en uno de sus extremos a algunos de los nodos del árbol formado hasta ahora (es decir, al nodo '1', '2' o '6') y se añade al árbol

$$\max\left(\rho_{13}^2, \rho_{14}^2, \rho_{15}^2, \rho_{23}^2, \rho_{24}^2, \rho_{25}^2, \rho_{63}^2, \rho_{64}^2, \rho_{65}^2\right) = \rho_{65}^2 = 0.9600 \quad \Rightarrow \quad árbol = \{1,2,6,5\}$$

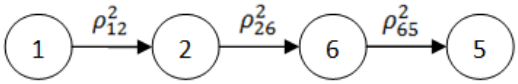

Figura 4-4. Árbol de expansión formado en la iteración 3.

Se eliminan todas las ramas entre los nodos que hay actualmente en el árbol, es decir, habría que eliminar los coeficientes $\rho_{12}^2, \rho_{21}^2, \rho_{16}^2, \rho_{61}^2, \rho_{26}^2, \rho_{62}^2, \rho_{15}^2, \rho_{51}^2, \rho_{25}^2, \rho_{52}^2, \rho_{65}^2, \rho_{56}^2$ (ya que estas ramas no pueden volver a incluirse en el árbol, para evitar bucles)

- Iteración 4: se busca la siguiente rama de mayor peso que tenga en uno de sus extremos a algunos de los nodos del árbol formado hasta ahora (es decir, al nodo '1', '2', '6' o '5') y se añade al árbol

$$\max\left(\rho_{13}^2, \rho_{14}^2, \rho_{23}^2, \rho_{24}^2, \rho_{63}^2, \rho_{64}^2, \rho_{53}^2, \rho_{54}^2\right) = \rho_{64}^2 = 0.9554 \quad \Rightarrow \quad árbol = \{1,2,6,5,4\}$$

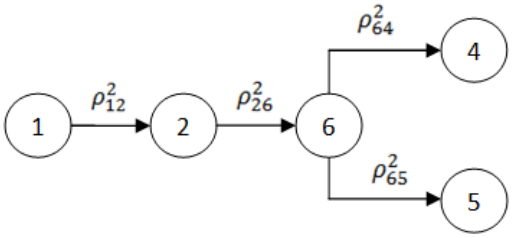

Figura 4-5. Árbol de expansión formado en la iteración 4.

Se eliminan todas las ramas entre los nodos que hay actualmente en el árbol, es decir, habría que eliminar los coeficientes $\rho_{12}^2, \rho_{21}^2, \rho_{16}^2, \rho_{61}^2, \rho_{26}^2, \rho_{62}^2, \rho_{15}^2, \rho_{51}^2, \rho_{25}^2, \rho_{52}^2, \rho_{65}^2, \rho_{56}^2, \rho_{14}^2, \rho_{41}^2, \rho_{24}^2, \rho_{42}^2, \rho_{64}^2, \rho_{46}^2, \rho_{54}^2, \rho_{45}^2$ (ya que estas ramas no pueden volver a incluirse en el árbol, para evitar bucles)

- Iteración 5 (última): se busca la siguiente rama de mayor peso que tenga en uno de sus extremos a algunos de los nodos del árbol formado hasta ahora (es decir, al nodo '1', '2', '6', '5' o '4') y se añade al árbol

$$\max\left(\rho_{13}^2, \rho_{23}^2, \rho_{63}^2, \rho_{53}^2, \rho_{43}^2\right) = \rho_{23}^2 = 0.9311 \quad \Rightarrow \quad árbol = \{1,2,6,5,4,3\}$$

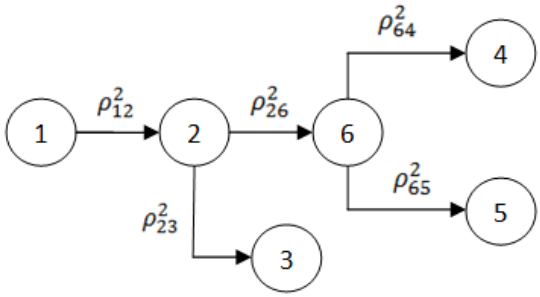

Figura 4-6. Árbol de expansión formado en la última iteración.

En la Figura 4-6 se muestra el árbol de expansión máximo de 6 nodos del ejemplo dado. Los padres y nodos extremos de este árbol son los siguientes

$$\{p(i)\} = \begin{cases} raiz & si\ i = 1 \\ 1 & si\ i = 2 \\ 2 & si\ i = 3 \\ 6 & si\ i = 4 \\ 6 & si\ i = 5 \\ 2 & si\ i = 6 \end{cases} \quad , \qquad extremos = \{3,4,5\}$$

4.3.3 Aplicación del algoritmo BP (versión MMSE)

Se calcula el estimador MMSE

$$\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} = \begin{bmatrix} 0.98 \\ 1.04 \\ -1.03 \\ 1.02 \\ 0.95 \\ -1.13 \end{bmatrix}$$

Se denotan las distribuciones marginales como

$$f(x_i; \mathbf{z}, \mathbf{C}) = \exp\left(-\frac{1}{2}\frac{(x_i - z_i)^2}{C_{ii}}\right)$$

$$f\left(x_i | x_j; \mathbf{z}, \mathbf{C}\right) = \exp\left(-\frac{1}{2}\frac{\left((x_i - z_i) - \frac{C_{ij}}{C_{jj}}\left(x_j - z_j\right)\right)^2}{C_{ii} - \frac{C_{ij}^2}{C_{jj}}}\right)$$

Se comienza el paso de mensajes "hacia abajo", es decir, desde los nodos extremos hacia el nodo raíz. Estos mensajes de muestran, con un indicador en rojo para indicar el orden de los mensajes, en la Figura 4-7.

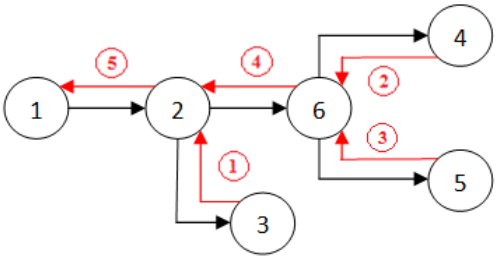

Figura 4-7. Mensajes "hacia abajo" generados en el algoritmo suma-producto.

Los mensajes generados en cada una de las iteraciones se calculan según las ecuaciones (4-15)-(4-16) y son los siguientes (la primera iteración y la cuarta se muestran de forma detallada y en las demás únicamente el resultado)

- Iteración 1 (nodo extremo):

$$m_{3\to 2}(x_2) = \sum_{x_3\in\mathcal{A}=\{-1,1\}} f(x_3|x_2;\mathbf{z},\mathbf{C})$$

$$= \sum_{x_3\in\{-1,1\}} \exp\left(-\frac{1}{2}\frac{\left((x_3-z_3)-\frac{c_{32}}{c_{22}}(x_2-z_2)\right)^2}{c_{33}-\frac{c_{32}^2}{c_{22}}}\right)$$

$$= \sum_{x_3\in\{-1,1\}} \exp\left(-\frac{1}{2}\frac{\left((x_3+1.03)+0.7083(x_2-1.04)\right)^2}{3.9e^{-5}}\right)$$

$$= \exp\left(-\frac{1}{2}\frac{\left((-1+1.03)+0.7083(x_2-1.04)\right)^2}{3.9e^{-5}}\right)$$

$$+ \exp\left(-\frac{1}{2}\frac{\left((1+1.03)+0.7083(x_2-1.04)\right)^2}{3.9e^{-5}}\right) = \begin{cases} 0 & si\ x_2=-1 \\ 0.8493 & si\ x_2=1 \end{cases}$$

- Iteración 2 (nodo extremo):

$$m_{4\to 6}(x_6) = \sum_{x_4\in\mathcal{A}} f(x_4|x_6;\mathbf{z},\mathbf{C}) = \begin{cases} 0.5690 & si\ x_6=-1 \\ 0 & si\ x_6=1 \end{cases}$$

- Iteración 3 (nodo extremo):

$$m_{5\to 6}(x_6) = \sum_{x_5\in\mathcal{A}} f(x_5|x_6;\mathbf{z},\mathbf{C}) = \begin{cases} 0.1500 & si\ x_6=-1 \\ 0 & si\ x_6=1 \end{cases}$$

- Iteración 4:

$$m_{6\to 2}(x_2)=\sum_{x_6\in\mathcal{A}} f(x_6|x_2;\mathbf{z},\mathbf{C})\prod_{j\,|\,p(j)=6} m_{j\to 6}(x_6)=\sum_{x_6\in\mathcal{A}} f(x_6|x_2;\mathbf{z},\mathbf{C})\, m_{4\to 6}(x_6)m_{5\to 6}(x_6)$$

$$=\sum_{x_6\in\mathcal{A}}\exp\left(-\frac{1}{2}\frac{\left((x_6-z_6)-\frac{c_{62}}{c_{22}}(x_2-z_2)\right)^2}{C_{66}-\frac{c_{62}^2}{c_{22}}}\right)m_{4\to 6}(x_6)m_{5\to 6}(x_6)$$

$$=\sum_{x_6\in\mathcal{A}}\exp\left(-\frac{1}{2}\frac{\left((x_6+1.13)+2.04(x_2-1.04)\right)^2}{1.27e^{-4}}\right)m_{4\to 6}(x_6)m_{5\to 6}(x_6)$$

$$=\exp\left(-\frac{1}{2}\frac{\left((-1+1.13)+2.04(x_2-1.04)\right)^2}{1.27e^{-4}}\right)m_{4\to 6}(-1)m_{5\to 6}(-1)$$

$$+\exp\left(-\frac{1}{2}\frac{\left((1+1.13)+2.04(x_2-1.04)\right)^2}{1.27e^{-4}}\right)m_{4\to 6}(1)m_{5\to 6}(1)$$

$$=\begin{cases}0 & si\ x_2=-1\\ 6.07e^{-5} & si\ x_2=1\end{cases}$$

- Iteración 5:

$$m_{2\to 1}(x_1)=\sum_{x_2\in\mathcal{A}} f(x_2|x_1;\mathbf{z},\mathbf{C})\prod_{j\,|\,p(j)=2} m_{j\to 2}(x_2)=\sum_{x_2\in\mathcal{A}} f(x_2|x_1;\mathbf{z},\mathbf{C})\, m_{3\to 2}(x_2)m_{6\to 2}(x_2)$$

$$=\begin{cases}0 & si\ x_1=-1\\ 8.82e^{-11} & si\ x_1=1\end{cases}$$

A continuación se prosigue con el paso de mensajes "hacia arriba", es decir, desde el nodo raíz hacia los extremos. Estos mensajes de muestran, con un indicador en rojo para indicar el orden de los mensajes, en la Figura 4-8.

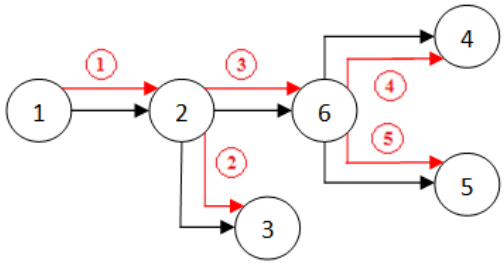

Figura 4-8. Mensajes "hacia arriba" generados en el algoritmo suma-producto.

Los mensajes generados en cada una de las iteraciones se calculan según las ecuaciones (4-17)-(4-18) y son los siguientes

- Iteración 1 (nodo raíz):

$$m_{1\to 2}(x_2)=\sum_{x_1\in\mathcal{A}} f(x_2|x_1;\mathbf{z},\mathbf{C})\prod_{\{j\,|\,j\neq 2,p(j)=1\}} m_{j\to 1}(x_1)=\sum_{x_1\in\mathcal{A}} f(x_2|x_1;\mathbf{z},\mathbf{C})=\begin{cases}0 & si\ x_2=-1\\ 1.71e^{-6} & si\ x_2=1\end{cases}$$

- Iteración 2:

$$m_{2\to3}(x_3) = \sum_{x_2\in\mathcal{A}} f(x_3|x_2;\mathbf{z},\mathbf{C}) m_{p(2)\to2}(x_2) \prod_{\{j\,|\,j\neq3, p(j)=2\}} m_{j\to2}(x_2)$$
$$= \sum_{x_2\in\mathcal{A}} f(x_3|x_2;\mathbf{z},\mathbf{C}) m_{1\to2}(x_2)\, m_{6\to2}(x_2) = \begin{cases} 8.82e^{-11} & si\ x_3 = -1 \\ 0 & si\ x_3 = 1 \end{cases}$$

- Iteración 3:

$$m_{2\to6}(x_6) = \sum_{x_2\in\mathcal{A}} f(x_6|x_2;\mathbf{z},\mathbf{C}) m_{p(2)\to2}(x_2) \prod_{\{j\,|\,j\neq6, p(j)=2\}} m_{j\to2}(x_2)$$
$$= \sum_{x_2\in\mathcal{A}} f(x_6|x_2;\mathbf{z},\mathbf{C}) m_{1\to2}(x_2)\, m_{3\to2}(x_2) = \begin{cases} 1.03e^{-9} & si\ x_6 = -1 \\ 0 & si\ x_6 = 1 \end{cases}$$

- Iteración 4:

$$m_{6\to4}(x_4) = \sum_{x_6\in\mathcal{A}} f(x_4|x_6;\mathbf{z},\mathbf{C}) m_{p(6)\to6}(x_6) \prod_{\{j\,|\,j\neq4, p(j)=6\}} m_{j\to6}(x_6)$$
$$= \sum_{x_6\in\mathcal{A}} f(x_4|x_6;\mathbf{z},\mathbf{C}) m_{2\to6}(x_6)\, m_{5\to6}(x_6) = \begin{cases} 0 & si\ x_4 = -1 \\ 8.82e^{-11} & si\ x_4 = 1 \end{cases}$$

- Iteración 5:

$$m_{6\to5}(x_5) = \sum_{x_6\in\mathcal{A}} f(x_5|x_6;\mathbf{z},\mathbf{C}) m_{p(6)\to6}(x_6) \prod_{\{j\,|\,j\neq5, p(j)=6\}} m_{j\to6}(x_6)$$
$$= \sum_{x_6\in\mathcal{A}} f(x_5|x_6;\mathbf{z},\mathbf{C}) m_{2\to6}(x_6)\, m_{4\to6}(x_6) = \begin{cases} 0 & si\ x_5 = -1 \\ 8.82e^{-11} & si\ x_5 = 1 \end{cases}$$

Una vez realizado todo el paso de mensajes, se predice la variable según las ecuaciones (4-19)-(4-20).

- Nodo raíz

$$belief_1(x_1) = f(x_1;\mathbf{z},\mathbf{C}) \prod_{j\,|\,p(j)=1} m_{j\to1}(x_1) = f(x_1;\mathbf{z},\mathbf{C}) m_{2\to1}(x_1) = \begin{cases} 0 & si\ x_1 = -1 \\ 7.68e^{-11} & si\ x_1 = 1 \end{cases}$$

- Nodos distintos al de raíz

$$belief_2(x_2) = m_{p(2)\to2}(x_2) \prod_{j\,|\,p(j)=2} m_{j\to2}(x_2) = m_{1\to2}(x_2) m_{3\to2}(x_2) m_{6\to2}(x_2)$$
$$= \begin{cases} 0 & si\ x_2 = -1 \\ 8.82e^{-11} & si\ x_2 = 1 \end{cases}$$
$$belief_3(x_3) = m_{p(3)\to3}(x_3) \prod_{j\,|\,p(j)=3} m_{j\to3}(x_3) = m_{2\to3}(x_3) = \begin{cases} 8.82e^{-11} & si\ x_3 = -1 \\ 0 & si\ x_3 = 1 \end{cases}$$
$$belief_4(x_4) = m_{p(4)\to4}(x_4) \prod_{j\,|\,p(j)=4} m_{j\to4}(x_4) = m_{6\to4}(x_4) = \begin{cases} 0 & si\ x_4 = -1 \\ 8.82e^{-11} & si\ x_4 = 1 \end{cases}$$
$$belief_5(x_5) = m_{p(5)\to5}(x_5) \prod_{j\,|\,p(j)=5} m_{j\to5}(x_5) = m_{6\to5}(x_5) = \begin{cases} 0 & si\ x_5 = -1 \\ 8.82e^{-11} & si\ x_5 = 1 \end{cases}$$

$$belief_6(x_6) = m_{p(6)\to 6}(x_6) \prod_{j\,|\,p(j)=6} m_{j\to 6}(x_6) = m_{2\to 6}(x_6) m_{4\to 6}(x_6) m_{5\to 6}(x_6)$$
$$= \begin{cases} 8.82e^{-11} & si\ x_6 = -1 \\ 0 & si\ x_6 = 1 \end{cases}$$

Por último se decodifican estos datos, es decir, se elige como símbolo transmitido el que maximiza las variables predichas anteriores, tal y como se expresa en la ecuación (4-21)

$$\hat{\mathbf{x}} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

Como era de esperar, al tomar una SNR muy elevada el ruido afecta muy poco y la decodificación es perfecta.

4.4 Versión SIC del algoritmo GTA

Al igual que se hizo con los algoritmos ZF y MMSE en el Apartado 1.2.4, es posible implementar una versión con cancelación sucesiva de interferencias (SIC) para el algoritmo GTA. La idea es similar a la vista en ese apartado. En cada iteración (i=1,…,N) se aplica el algoritmo GTA y se obtiene una estimación del símbolo más fiable x_i, es decir, aquel que tiene menor varianza. A continuación se cancela su efecto, obteniéndose un sistema con N-i incógnitas (es decir, en cada iteración se reduce en uno el número de variables desconocidas). Este procedimiento se repite hasta que todos los símbolos han sido decodificados, es decir, N-1 veces.

En la versión estándar del algoritmo GTA, se aplicaba el algoritmo BP a un árbol previamente calculado que requería el paso de mensajes en dos sentidos: hacia abajo (desde los extremos al nodo raíz) y hacia arriba (desde el nodo raíz a los extremos). Después del intercambio de mensajes, se predecían las variables de cada uno de los nodos. Sin embargo, en la versión SIC del algoritmo GTA sólo se predice la variable de un único nodo por iteración. El orden de detección de estos nodos es importante: se detectan de menor a mayor covarianza. Sin pérdida de generalidad, se considerará como nodo raíz aquel nodo que se está estimando. De esta forma, sólo se necesita predecir la variable del nodo raíz y no es necesario realizar el paso de mensajes hacía arriba ya que, como se vio en (4-20), la variable predicha[8] en el nodo raíz x_i es

$$belief_{k_i}(x_{k_i}) = f(x_{k_i}; \mathbf{z}, \mathbf{C}) \prod_{j\,|\,p(j)=k_i} m_{j\to k_i}(x_{k_i}) \qquad x_{k_i} \in \mathcal{A} \tag{4-23}$$

Para obtener el valor decodificado, se toma el símbolo cuya probabilidad a posteriori (4-23) es máxima

$$\hat{x}_{k_i} = \arg\max_a belief_{k_i}(a) \qquad a \in \mathcal{A} \tag{4-24}$$

Una vez obtenida la estimación $\hat{x}_i$, ésta se elimina del sistema y se determina, nuevamente, la variable con menor covarianza, siendo ésta el próximo nodo raíz y por tanto la nueva variable a decodificar. Este proceso se repite hasta que se han decodificado las N variables.

4.4.1 Resumen del algoritmo GTA-SIC (versión MMSE)

El algoritmo GTA-SIC (versión MMSE) se resume en la Figura 4-9.

[8] Cuando se habla de variable predicha (*belief*$_i$(x_i)), ésta coincide con la distribución marginal de la aproximación de la distribución de probabilidad (4-14).

- Entradas:
  Un sistema lineal $\mathbf{y} = \mathbf{Hx} + \mathbf{v}$, un nivel de ruido σ^2 y un conjunto finito de símbolos $\mathcal{A}$ cuya energía media de símbolo es E_s

- Algoritmo:
  $\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ N$
  - Se calculan
    $$\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} \quad , \quad \mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}$$
  - Se denota
    $$f(x_k; \mathbf{z}, \mathbf{C}) = \exp\left(-\frac{1}{2}\frac{(x_k - z_k)^2}{C_{kk}}\right)$$
    $$f(x_k|x_j; \mathbf{z}, \mathbf{C}) = \exp\left(-\frac{1}{2}\frac{\left((x_k - z_k) - \frac{C_{kj}}{C_{jj}}(x_j - z_j)\right)^2}{C_{kk} - \frac{C_{kj}^2}{C_{jj}}}\right)$$
  - Ordenamiento óptimo:
    $$k_i = \arg\min_{j \notin (k_1, \ldots, k_{i-1})} C_{jj}$$
  - Árbol óptimo: Se calcula el árbol de expansión máximo (con nodo raíz x_{k_i}) del grafo de $N - (i-1)$ nodos donde el peso de la rama k-j es el cuadrado del coeficiente de correlación
    $$\rho_{kj}^2 = \frac{C_{kj}^2}{C_{kk}C_{jj}}$$
  - Algoritmo BP: Se realiza el intercambio de mensajes hacia abajo (desde los extremos al nodo raíz). El mensaje de *k* a su padre *j=p(k)* es
    $$m_{k\to p(k)}(x_{p(k)}) = \sum_{x_k \in \mathcal{A}} f(x_k|x_{p(k)}; \mathbf{z}, \mathbf{C}) \prod_{j\,|\,p(j)=k} m_{j\to k}(x_k) \qquad x_{p(k)} \in \mathcal{A}$$
  - Decisor:
    $$\hat{x}_{k_i} = \arg\max_a \left(f(a; \mathbf{z}, \mathbf{C}) \prod_{j\,|\,p(j)=k_i} m_{j\to k_i}(a)\right) \qquad a \in \mathcal{A}$$
  - Cancelación de interferencias:
    $$\mathbf{y} = \mathbf{y} - \mathbf{H}_{\mathbf{k_i}}\hat{x}_{k_i}$$
    $$\mathbf{H} = \mathbf{H}_{\overline{\mathbf{k_i}}}$$

  $\boldsymbol{end}$

Figura 4-9. Resumen del algoritmo GTA-SIC.

5 ALGORITMO EP

El método EP (expectation propagation) [14]-[15] permite obtener una aproximación de la función de distribución a posteriori mediante distribuciones de la familia exponencial, es decir, distribuciones definidas de la siguiente forma

$$f(\mathbf{x}) = h(\mathbf{x})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^{\mathrm{T}}\Phi(\mathbf{x})\right) \propto \exp\left(\boldsymbol{\eta}^{\mathrm{T}}\Phi(\mathbf{x})\right) \tag{5-1}$$

Sea un modelo estadístico en el que la variable **x** es desconocida y se puede factorizar como

$$p(\mathbf{x}) \propto f(\mathbf{x}) \prod_{i=1}^{N} t_i(\mathbf{x}) \tag{5-2}$$

donde $f(\mathbf{x})$ pertecene a una familia exponencial $\mathcal{F}$ con estadísticos suficientes $\Phi(\mathbf{x}) = \{\Phi_1(\mathbf{x}), \dots, \Phi_{\mathrm{N}}(\mathbf{x})\}$ y $t_i(\mathbf{x}) \quad i = 1, \dots, N$ son factores no negativos. Dado que es muy costoso extraer información de la distribución $p(\mathbf{x})$ (5-2), el método EP ofrece una aproximación de $p(\mathbf{x})$ a través de una distribución $q(\mathbf{x})$ de $\mathcal{F}$, que está también dada por un producto de factores

$$q(\mathbf{x}) = f(\mathbf{x}) \prod_{i=1}^{N} \tilde{t}_i(\mathbf{x}) \tag{5-3}$$

donde $\tilde{t}_i(\mathbf{x}) \in \mathcal{F}$ para $i = 1, \dots, N$ es una aproximación a los factores $t_i(\mathbf{x})$ de (5-2) y que presenta los mismos momentos que $p(\mathbf{x})$, es decir

$$\mathbb{E}_{q(\mathbf{x})}\left[\Phi_{\mathrm{j}}(\mathbf{x})\right] = \mathbb{E}_{p(\mathbf{x})}\left[\Phi_{\mathrm{j}}(\mathbf{x})\right] \quad j = 1, \dots, N \tag{5-4}$$

siendo $\mathbb{E}_{q(\mathbf{x})}[\cdot]$ la esperanza respecto a la distribución $q(\mathbf{x})$. Esta ecuación se conoce como *moment matching condition* y es equivalente a encontrar la distribución $q(\mathbf{x})$ que minimiza la divergencia KL con $p(\mathbf{x})$

$$q(\mathbf{x}) = \arg \min_{q'(\mathbf{x}) \in \mathcal{F}} D\left(p(\mathbf{x}) \parallel q'(\mathbf{x})\right) \tag{5-5}$$

En caso de que la distribución $q(\mathbf{x})$ sea gaussiana $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \sum)$, la ecuación (5-5) se reduce a igualar la media de $q(\mathbf{x})$ con la de $p(\mathbf{x})$ y la covarianza de $q(\mathbf{x})$ con la de $p(\mathbf{x})$.

Como no es posible inferir información de $p(\mathbf{x})$, la idea principal de EP reside en extraer información de una distribución en la que sólo está presente uno de los factores $t_i(\mathbf{x})$ para $i = 1, \dots, N$ de (5-2) que no pertenecen a la familia de exponenciales $\mathcal{F}$, es decir

$$\hat{p}_i(\mathbf{x}) \propto f(\mathbf{x})t_i(\mathbf{x}) \tag{5-6}$$

De esta forma, EP va redefiniendo, secuencial e independientemente, los factores $\tilde{t}_i(\mathbf{x})$ quitándolos de la

distribución $q(\mathbf{x})$ y asegurando que el producto de todos los factores esté cercano a la distribución original $p(\mathbf{x})$ [14]-[16]. Por ejemplo, si se quiere redefinir el factor $\tilde{t}_j(\mathbf{x})$, primero se elimina de la distribución $q(\mathbf{x})$ (es decir, se divide $q(\mathbf{x})$ entre $\tilde{t}_j(\mathbf{x})$) y se determina un nuevo factor $\tilde{t}_j(\mathbf{x})$ que asegure que el producto

$$q^{new}(\mathbf{x}) \propto \tilde{t}_j(\mathbf{x}) \prod_{\substack{i=1 \\ i \neq j}}^{N} \tilde{t}_i(\mathbf{x}) \tag{5-7}$$

esté lo más cercano posible a

$$\hat{p}(\mathbf{x}) \propto t_j(\mathbf{x}) \prod_{\substack{i=1 \\ i \neq j}}^{N} \tilde{t}_i(\mathbf{x}) \tag{5-8}$$

Este proceso se repite, de forma iterativa, hasta alcanzar la convergencia de todos los $\tilde{t}_i(\mathbf{x})$, es decir, EP va aproximando los momentos de (5-4) de forma recursiva.

El algoritmo EP secuencial se puede resumir de la siguiente manera [16]:

1. Inicializar todos los factores aproximados $\tilde{t}_i^{(0)}(\mathbf{x})$.
2. Inicializar la aproximación a posteriori según

$$q^{(0)}(\mathbf{x}) = f(\mathbf{x}) \prod_{i=1}^{N} \tilde{t}_i^{(0)}(\mathbf{x}) \tag{5-9}$$

   Para denotar la aproximación a posteriori y los factores aproximados en la iteración ℓ, se utilizará la notación $q^{(\ell)}(\mathbf{x})$ y $\tilde{t}_i^{(\ell)}(\mathbf{x})$, respectivamente. Como se muestra en el paso 3, $q^{(\ell+1)}(\mathbf{x})$ se obtiene actualizando cada uno de los factores $\tilde{t}_i^{(\ell)}(\mathbf{x})$ de forma independiente.
3. Para cada $j = 1, \ldots, N$, hay que repetir los siguientes pasos hasta la convergencia (o un número máximo de iteraciones)
   a. Se elimina el factor $\tilde{t}_j^{(\ell)}(\mathbf{x})$ (siendo éste el factor a redefinir) de la aproximación $q^{(\ell)}(\mathbf{x})$ mediante su división

$$q^{(\ell)\setminus j}(\mathbf{x}) = \frac{q^{(\ell)}(\mathbf{x})}{\tilde{t}_j^{(\ell)}(\mathbf{x})} \in \mathcal{F} \tag{5-10}$$

   b. Se calcula la distribución $\hat{p}_j^{(\ell)}(\mathbf{x}) \propto t_j(\mathbf{x}) q^{(\ell)\setminus j}(\mathbf{x})$ y se calculan sus momentos, es decir,

$$\mathbb{E}_{\hat{p}_j^{(\ell)}(\mathbf{x})}\left[\Phi_j(\mathbf{x})\right] \quad j = 1, \ldots, N \tag{5-11}$$

   Para ello, hay que incluir la evaluación de la constante de normalización

$$K_j = \int t_j(\mathbf{x}) q^{(\ell)\setminus j}(\mathbf{x}) \, \mathrm{d}\mathbf{x} \tag{5-12}$$

   c. Se redefine el nuevo factor $\tilde{t}_j^{(\ell+1)}(\mathbf{x})$ de tal forma que, el momento de la nueva aproximación a posteriori $q^{(\ell+1)}(\mathbf{x}) \propto \tilde{t}_j^{(\ell+1)}(\mathbf{x}) q^{(\ell)\setminus j}(\mathbf{x})$, esto es

$$\mathbb{E}_{q^{(\ell+1)}(\mathbf{x})}\left[\Phi_j(\mathbf{x})\right] \quad j = 1, \ldots, N \tag{5-13}$$

   coincida con (5-11) para $j = 1, \ldots, N$. Este nuevo factor de aproximación es

$$\tilde{t}_j^{\,(\ell+1)}(\mathbf{x}) = K_j \frac{q^{(\ell+1)}(\mathbf{x})}{q^{(\ell)\setminus j}(\mathbf{x})} \tag{5-14}$$

A continuación se verá como se aplica este algoritmo al problema de detección MIMO.

5.1 Aplicación de EP a detectores MIMO

El objetivo de este método [13] es encontrar una aproximación a la distribución de probabilidad a posteriori p(**x**|**y**) mediante una aproximación gaussiana $q_{EP}(\mathbf{x})$ que se obtiene de forma iterativa y va buscando que la divergencia de Kullback-Leibler de ambas distribuciones sea mínima. En concreto, se trata de buscar la aproximación gaussiana

$$q_{EP}(\mathbf{x}) = \mathcal{N}(\mathbf{x}\colon \boldsymbol{\mu}_{\mathbf{EP}}, \sum_{\mathbf{EP}}) \tag{5-15}$$

cuya media y varianza son

$$\boldsymbol{\mu}_{\mathbf{EP}} = \mathbb{E}_{\mathrm{p}(\mathbf{x}|\mathbf{y})}[\mathbf{x}] \tag{5-16}$$

$$\sum_{\mathbf{EP}} = \mathrm{CoVar}_{\mathrm{p}(\mathbf{x}|\mathbf{y})}[\mathbf{x}] \tag{5-17}$$

En este caso, la función de probabilidad a posteriori que se quiere aproximar está dada por (4-1), es decir

$$p(\mathbf{x}|\mathbf{y}) \propto \mathcal{N}(\mathbf{y}\colon \mathbf{H}\mathbf{x}, \sigma^2\mathbf{I}) \qquad \mathbf{x} \in \mathcal{A}^N \tag{5-18}$$

En vez de expresar la restricción de que los datos pertecen a un conjunto finito de datos como $\mathbf{x} \in \mathcal{A}^N$, se puede escribir de forma equivalente

$$p(\mathbf{x}|\mathbf{y}) \propto \mathcal{N}(\mathbf{y}\colon \mathbf{H}\mathbf{x}, \sigma^2\mathbf{I}) \prod_{i=1}^{N} \mathbb{I}_{x_i \in \mathcal{A}} \tag{5-19}$$

donde la función $\mathbb{I}_{x_i \in \mathcal{A}}$ es aquella que toma valor 1 si $x_i \in \mathcal{A}$ y cero en otro caso.

Como el término $\prod_{i=1}^{N} \mathbb{I}_{x_i \in \mathcal{A}}$ es una uniforme discreta, el algoritmo EP busca una aproximación que sustituya esta uniforme por una gaussiana, dando lugar a una distribución gaussiana definida en el lemma 5.1.

Lemma 5.1 (Distribución q(x) por la cual se aproxima p(x) en el algoritmo EP aplicado a MIMO)

La distribución $q_{EP}(\mathbf{x})$ es un producto de gaussianas, es decir, una única gaussiana de la forma

$$q_{EP}(\mathbf{x}) \propto \mathcal{N}(\mathbf{y}\colon \mathbf{H}\mathbf{x}, \sigma^2\mathbf{I}) \prod_{i=1}^{N} \exp\left(\gamma_i x_i - \frac{1}{2}\Lambda_i {x_i}^2\right) \propto \mathcal{N}(\mathbf{x}\colon \boldsymbol{\mu}_{\mathbf{EP}}, \sum_{\mathbf{EP}}) \tag{5-20}$$

donde

$$\sum_{\mathbf{EP}} = \left(\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{H} + \mathrm{diag}(\boldsymbol{\Lambda})\right)^{-1} \tag{5-21}$$

$$\boldsymbol{\mu}_{\mathbf{EP}} = \sum_{\mathbf{EP}}\left(\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{y} + \boldsymbol{\gamma}\right) \tag{5-22}$$

Demostración: La demostración de este lemma se encuentra en el Apéndice A.3.

■

El objetivo del algoritmo EP es actualizar de forma recursiva los valores de (γ_i, Λ_i) para *i*=1, …, *N* e ir recalculando el vector de medias y matriz de covarianzas de (5-21) y (5-22).

Una vez terminado el proceso iterativo (en el que se obtiene el vector de medias y matriz de covarianzas lo más parecido posible al de la distribución exacta a posteriori), la estimación de cada variable se obtiene decodificando cada variable por separado, es decir

$$\hat{x}_{i,EP} = \arg\min_{a\in\mathcal{A}} \left|\mu_{i,EP} - a\right| \qquad i = 1,\dots,N \tag{5-23}$$

Este algoritmo se resume en la Figura 5-1.

1. Inicializar ${\gamma_i}^{(0)} = 0$ y ${\Lambda_i}^{(0)} = {E_s}^{-1}$ para todo $i = 1,\dots,N$. Si no se continuara con los pasos siguientes, se obtendría la solución MMSE.

   Al igual que en el apartado anterior, se denotará con (ℓ) las iteraciones del algoritmo EP.

2. **for** $\ell = 0$ **to** L_{EP}

   a. Calcular las marginales *i*-ésimas de la distribución $q^{(\ell)}(\mathbf{x})$ de (5-20) como ${q_i}^{(\ell)}(x_i) = \mathcal{N}(x_i : {\mu_i}^{(\ell)}, {\sigma_i}^{2(\ell)})$ donde ${\mu_i}^{(\ell)}$ y ${\sigma_i}^{2(\ell)}$ se obtienen de sustituir $\boldsymbol{\gamma}^{(\ell)}$ y $\boldsymbol{\Lambda}^{(\ell)}$ en (5-21) y (5-22).

   b. Eliminar las exponenciales, de una en una e independientemente, de la distribución $q^{(\ell)}(\mathbf{x})$

$$q^{(\ell)\backslash i}(x_i) = \frac{{q_i}^{(\ell)}(x_i)}{\exp\left(\gamma_i{}^{(\ell)} x_i - \frac{1}{2}\Lambda_i{}^{(\ell)} {x_i}^2\right)} \sim \mathcal{N}(x_i : {t_i}^{(\ell)}, {h_i}^{2(\ell)}) \tag{5-24}$$

   donde[9]

$${h_i}^{2(\ell)} = \frac{{\sigma_i}^{2(\ell)}}{1 - {\sigma_i}^{2(\ell)}\Lambda_i{}^{(\ell)}} \tag{5-25}$$

$${t_i}^{(\ell)} = {h_i}^{2(\ell)}\left(\frac{{\mu_i}^{(\ell)}}{{\sigma_i}^{2(\ell)}} - \gamma_i{}^{(\ell)}\right) \tag{5-26}$$

   c. Calcular la media ${\mu_{p_i}}^{(\ell)}$ y la varianza ${\sigma_{p_i}}^{2(\ell)}$ de la distribución

$$\hat{p}_i{}^{(\ell)}(x_i) \propto q^{(\ell)\backslash i}(x_i)\mathbb{I}_{x_i\in\mathcal{A}} = \begin{cases} q^{(\ell)\backslash i}(x_i) & si\ x_i \in \mathcal{A} \\ 0 & e.o.c \end{cases} \tag{5-27}$$

   d. Actualizar la pareja $(\gamma_i{}^{(\ell+1)}, \Lambda_i{}^{(\ell+1)})$ de tal forma que la distribución gaussiana (sin normalizar)

$$q^{(\ell)\backslash i}(x_i)\exp\left(\gamma_i{}^{(\ell+1)} x_i - \frac{1}{2}\Lambda_i{}^{(\ell+1)} {x_i}^2\right) \tag{5-28}$$

   tenga de media ${\mu_{p_i}}^{(\ell)}$ y varianza ${\sigma_{p_i}}^{2(\ell)}$. La solución está dada por[10]

$$\Lambda_i{}^{(\ell+1)} = \frac{1}{{\sigma_{p_i}}^{2(\ell)}} - \frac{1}{{h_i}^{2(\ell)}} \tag{5-29}$$

$$\gamma_i{}^{(\ell+1)} = \frac{{\mu_{p_i}}^{(\ell)}}{{\sigma_{p_i}}^{2(\ell)}} - \frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}} \tag{5-30}$$

end

Figura 5-1. Resumen del algoritmo EP.

[9] La demostración de (5-24) se encuentra en el Apartado A.4.
[10] La demostración de (5-29) y (5-30) se encuentran en el Apartado A.5.

La actualización de Λ_i en (5-29) puede devolver un valor negativo para ${\Lambda_i}^{(\ell+1)}$, lo cual no tiene sentido porque al tratarse de un término de precisión (inverso de la varianza) siempre debería ser positivo. Esto indica que no existe ninguna pareja $({\gamma_i}^{(\ell+1)}, {\Lambda_i}^{(\ell+1)})$ que consiga igualar la varianza de la distribución gaussiana en (5-28) a ${\sigma_{p_i}}^{2(\ell)}$. En ese caso, se mantienen los valores de la iteración anterior para dichos parámetros, es decir, ${\gamma_i}^{(\ell+1)} = {\gamma_i}^{(\ell)}$ y ${\Lambda_i}^{(\ell+1)} = {\Lambda_i}^{(\ell)}$ y se actualizan las demás parejas.

Por último, para mejorar la robustez del algoritmo, se introduce un parámetro de suavizado $\beta \in [0,1]$ en (5-29) y (5-30) que determina la rapidez del algoritmo para calcular los nuevos valores $({\gamma_i}^{(\ell+1)}, {\Lambda_i}^{(\ell+1)})$

$$ {\Lambda_i}^{(\ell+1)} = \beta\left(\frac{1}{{\sigma_{p_i}}^{2(\ell)}} - \frac{1}{{h_i}^{2(\ell)}}\right) + (1-\beta){\Lambda_i}^{(\ell)} \tag{5-31} $$

$$ {\gamma_i}^{(\ell+1)} = \beta\left(\frac{{\mu_{p_i}}^{(\ell)}}{{\sigma_{p_i}}^{2(\ell)}} - \frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}}\right) + (1-\beta){\gamma_i}^{(\ell)} \tag{5-32} $$

6 RESULTADOS DE SIMULACIÓN

En este capítulo se muestran los resultados de simulación en matlab del algoritmo GTA y EP para varios sistemas MIMO[11] y se compara con los algoritmos MMSE y ZF. La matriz de canal tiene una distribución normal de varianza unidad y media cero, cuyos elementos son independientes e idénticamente distribuidos. Se utilizan 100 realizaciones de la matriz de canal y cada matriz es usada para mandar 1000 mensajes, por lo que se envían un total de 10^5 mensajes. El factor de suavizado en las ecuaciones (5-31) y (5-32) del algoritmo EP se ha fijado a $\beta = 0.1$. El rendimiento del algoritmo se mostrará en función de la varianza del ruido aditivo σ^2 o, equivalentemente, de la relación señal a ruido (SNR) que se define como

$$SNR_{dB} = 10\log_{10}\frac{N \cdot E_s}{N_0} = 10\log_{10}\frac{N \cdot E_s}{\sigma^2} \tag{6-1}$$

Donde N es el número de antenas transmisoras, E_s es la energía de la señal y σ^2 la varianza del ruido aditivo gaussiano.

En la Figura 6-1 se representa un sistema MIMO 12x12 (a) y 16x16 (b) usando los símbolos de una constelación BPSK. Se muestran los algoritmos de detección EP[12], GTA, GTA-SIC, MMSE, MMSE-SIC, ZF, ZF-SIC y la solución óptima ML. Se puede observar como los algoritmos GTA-SIC y EP obtienen unos resultados cercanos a la solución óptima. La versión estándar de GTA no ofrece unos resultados tan cercanos a la solución óptima como su versión SIC pero consigue probabilidades de error más bajas que el algoritmo MMSE-SIC sobre todo para SNRs altas.

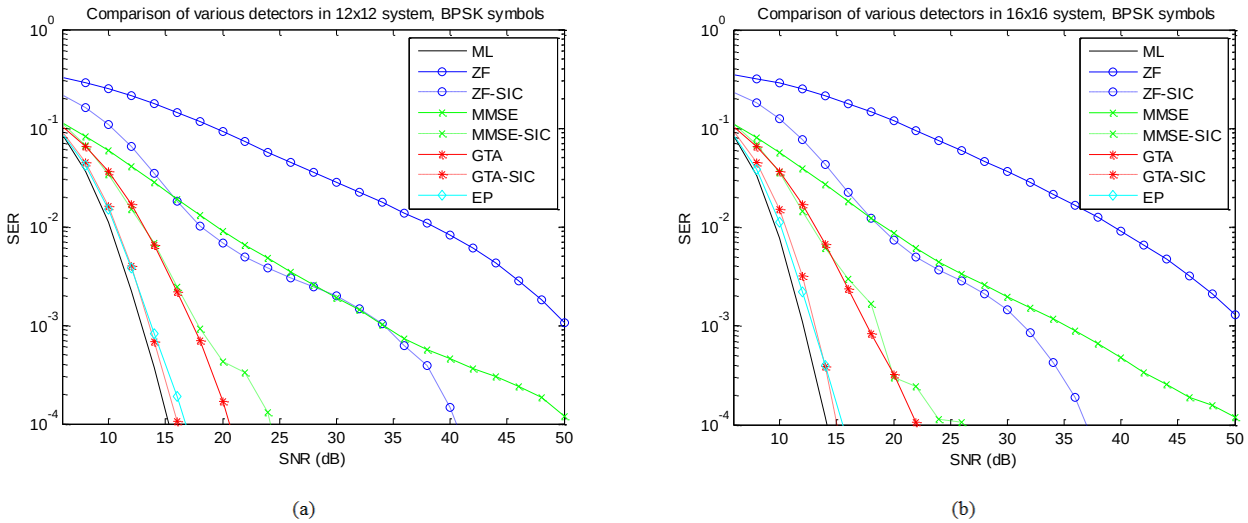

Figura 6-1. Probabilidad de error (SER) para símbolos BPSK y sistema MIMO: (a) 12x12, (b) 16x16.

[11] En concreto, se simulan la mayoría de los sistemas de las gráficas de los artículos [1], [2] y [13].
[12] Se toman 10 iteraciones para realizar la simulación de EP. Más adelante de este capítulo se explica por qué se toman 10 iteraciones.

En la Figura 6-2 se representa un sistema MIMO 20x20, usando símbolos de una constelación 4-PAM (a) y 4-QAM (b). Se muestran los algoritmos de detección EP, GTA, GTA-SIC, MMSE, MMSE-SIC, ZF y ZF-SIC. En este caso (y en los siguientes) no se muestra la solución óptima ML debido a la gran carga computacional que presenta este algoritmo para constelación mayores que la BPSK y/o sistemas de mayor orden que 16x16. Nuevamente se observa que los algoritmos GTA-SIC y EP son los que obtienen probabilidades de error menores. En concreto, para la constelación BPSK (a), EP ofrece unos resultados muy similares a los de GTA-SIC (aunque para SNRs bajas parece que el EP mejora en algo la probabilidad de error respecto a GTA-SIC), mientras que para la constelación 4-QAM (b), EP mejora a GTA-SIC en 2 dB para una SER=10^{-3}. Por otro lado, el algoritmo GTA (en su versión estándar) y MMSE-SIC ofrecen resultados similares, salvo para SNRs altas dónde el GTA le supera.

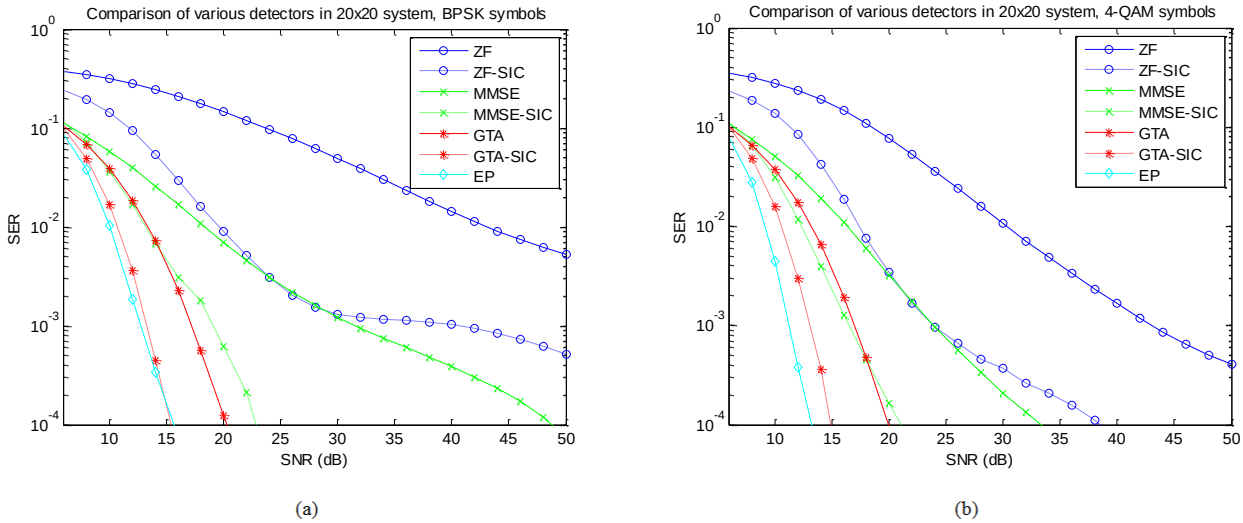

Figura 6-2. Probabilidad de error (SER) para un sistema MIMO 20x20 y símbolos: (a) 4-PAM, (b) 4-QAM.

En la Figura 6-3 se representan los algoritmos de detección EP, GTA, GTA-SIC, MMSE, MMSE-SIC, ZF y ZF-SIC para la constelación 16-QAM. En concreto, se muestra un sistema MIMO de 16x16 (a), 32x32 (b), 50x50 (c) y 64x64 (d). En ellas se muestra algo similar a las figuras anteriores, el algoritmo GTA-SIC y EP supera a todos los demás y el MMSE-SIC obtienen resultados parecidos, salvo para SNRs altas donde es mejor el GTA. Sin embargo, en estas figuras se observa también un comportamiento peculiar del EP para SNRs altas, aportando resultados peores que con el GTA-SIC sobre todo en sistemas de un orden no demasiado alto, como 16x16 y 32x32. Se observa también que, conforme aumenta el orden del sistema, los algoritmos EP y GTA-SIC se distancian, consiguiendo el algoritmo EP unas probabilidades de error menor que el GTA-SIC. En concreto, para una SER=10^{-3}, EP mejora a GTA-SIC en 5dB en sistemas de alto orden como 64x64.

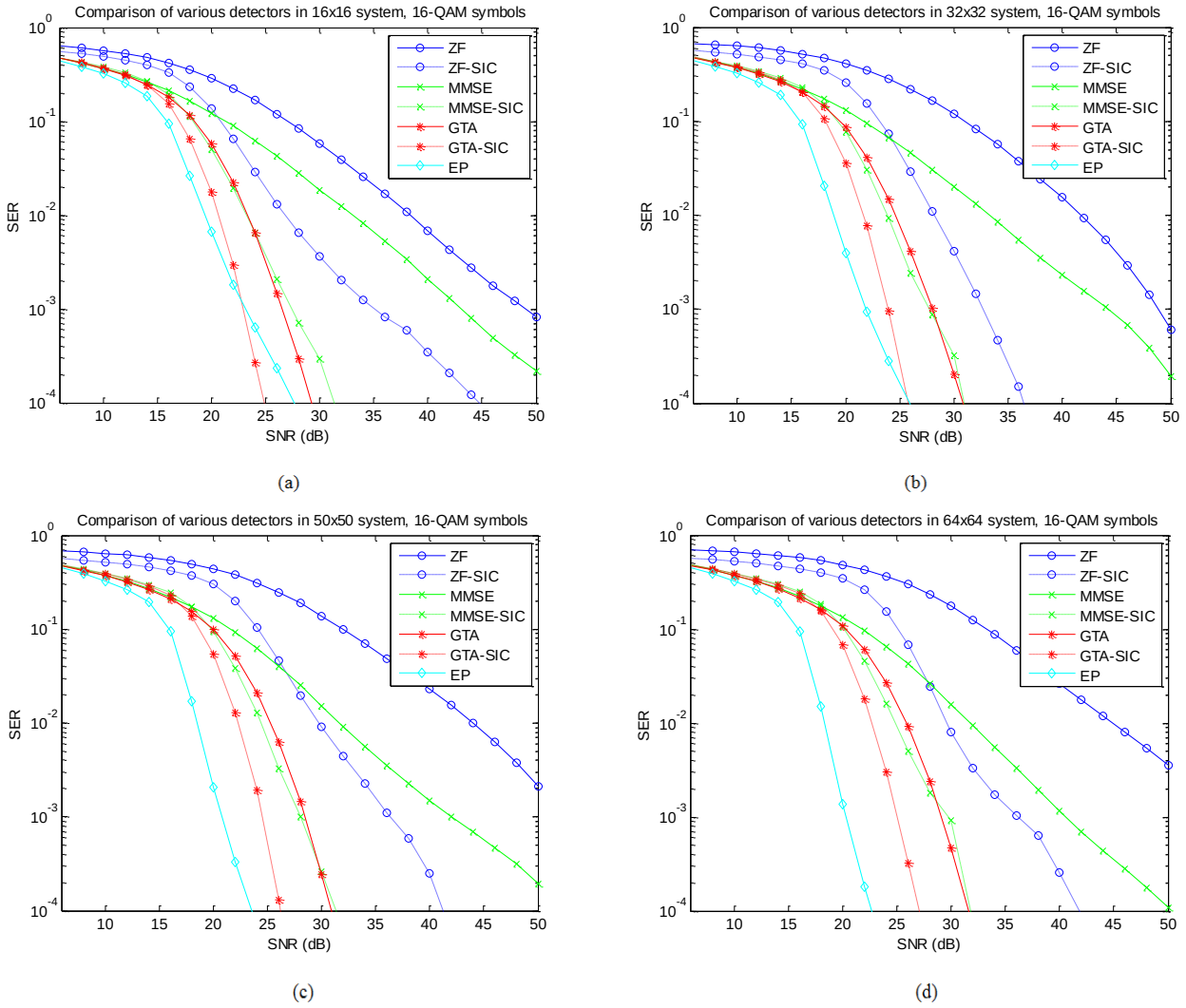

Figura 6-3. Probabilidad de error (SER) para símbolos 16-QAM y sistema MIMO: (a) 16x16, (b) 32x32, (c) 50x50, (d) 64x64.

La última constelación que se representa es una constelación 64-QAM, como se muestra en la Figura 6-4. En ella se comparan sistemas MIMO de 16x16 (a) y 32x32 (b) para los algoritmos de detección EP, GTA, GTA-SIC, MMSE, MMSE-SIC, ZF y ZF-SIC. Nuevamente se observa que los resultados del EP son los mejores para SNRs bajas, pero éstos empeoran con SNRs altas y que MMSE-SIC obtiene resultados similares a GTA, salvo en SNRs altas donde GTA es mejor y que los algoritmos que obtienen menor probabilidad de error son GTA-SIC y EP.

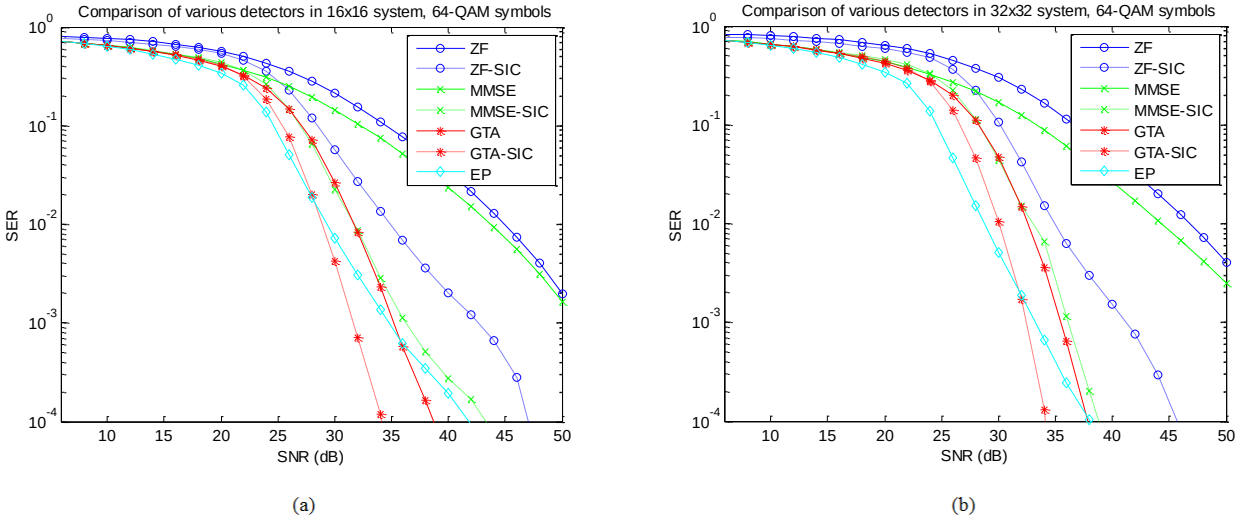

Figura 6-4. Probabilidad de error (SER) para símbolos 64-QAM y sistema MIMO: (a) 16x16, (b) 32x32.

En resumen, de las figuras anteriores se extrae que, para constelaciones y sistemas de alto orden el algoritmo que ofrece menor probabilidad de error es el EP (siempre y cuando la SNR no sea muy alta), seguido del algoritmo GTA-SIC. En cuanto a la versión estándar de GTA, su probabilidad de error es mucho menor que el MMSE y ZF (en su versión estándar y SIC) y ligeramente mejor que el MMSE-SIC para SNRs altas.

El motivo por el que se han tomado 10 iteraciones en el algoritmo EP reside en la Figura 6-5 donde se muestra, para un sistema 2x2, una constelación 16-PAM y una SNR=15dB, el valor exacto de la media (a) y varianza (b) de la distribución a posterir p(**x**|**y**) (4-1) en línea discontínua azul frente a la media (a) y varianza (b) en cada iteración del algoritmo EP, mostrado con círculos rojos. De esta Figura, se extrae que en unas 10 iteraciones, se alcanza una media y varianza bastante aproximada a la de la distribución a posteriori p(**x**|**y**) y por ello en todas las simulaciones se ha representado el algoritmo EP con 10 iteraciones.

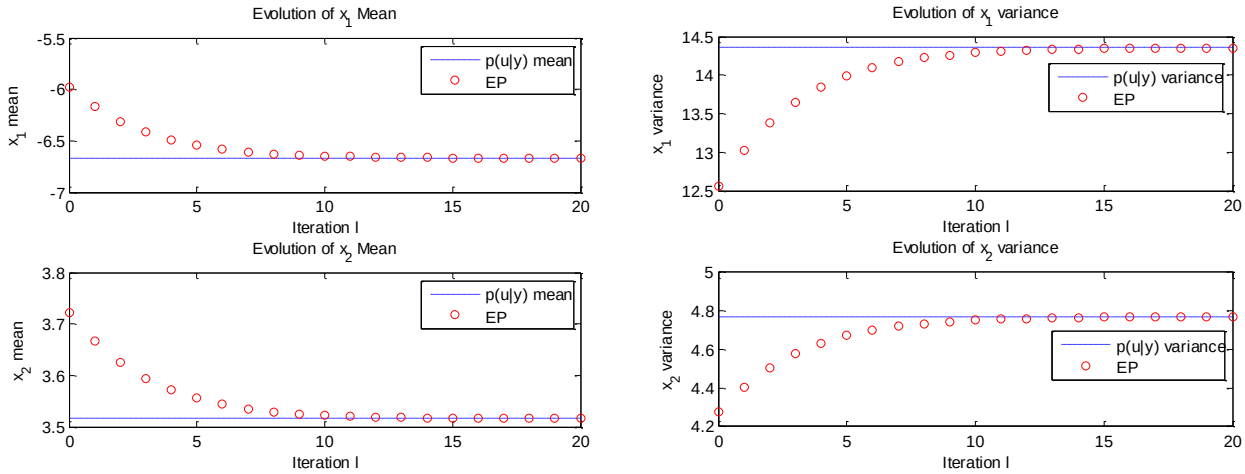

Figura 6-5. Evolución para un sistema 2x2, constelación 16-PAM y SNR=15 dB de cada componente de la media (a) y varianza (b) en cada iteración del algoritmo EP frente a su valor exacto.

Además, en la Figura 6-6 se muestra la evolución de gamma y GAMMA en cada iteración y se compara con los valores a los que fueron inicializados, con el objetivo de tener una idea de cómo se van desviando estos parámetros de sus valores iniciales. En esta figura se observa también que a partir de la iteración 10 los valores de γ_i y Λ_i permacen prácticamente constantes.

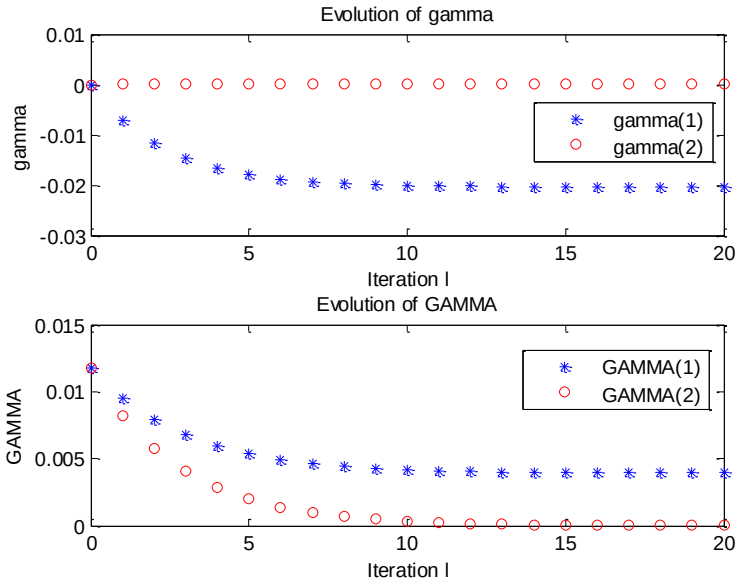

Figura 6-6. Evolución de $(\gamma_i{}^{(\ell+1)}, \Lambda_i{}^{(\ell+1)})$ para un sistema 2x2, constelación 16-PAM y SNR=15 dB.

Para corroborar aún más las 10 iteraciones tomadas en el algoritmo EP, en la Figura 6-7 se muestra una comparativa del algoritmo EP con distintos números de interaciones frente a los algoritmos GTA, GTA-SIC y MMSE para un sistema 32x32 y una constelación 16-QAM. Se observa que con dos únicas iteraciones del algoritmo EP ya se consigue superar al algoritmo GTA, pero no a GTA-SIC en SNRs altas. Si se sigue aumentando el número de iteraciones, llega un momento en que la probabilidad de error parece no mejorar, de hecho, no existe ninguna diferencia entre tomar 10 o 100 iteraciones del algoritmo EP.

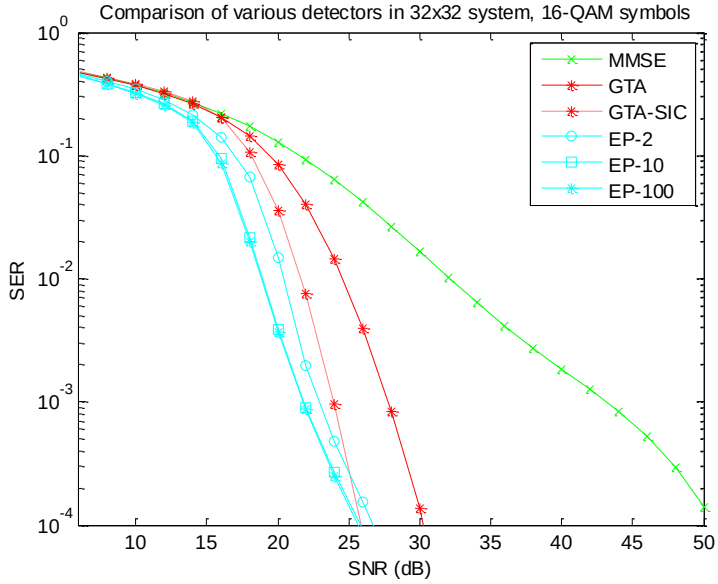

Figura 6-7. Probabilidad de error (SER) del algoritmo EP con distinto número de iteraciones para símbolos 16-QAM y sistema MIMO 32x32.

Por último, para tener una idea de la carga computaciones de los algoritmos de detección anteriores, en la Figura 6-8 se muestra una gráfica comparativa de los tiempos de simulación (para el simulador Matlab) necesarios para cada uno de los algoritmos anteriores y un sistema MIMO 10x10 con símbolos BPSK[13]. En la figura se observa que la solución ML es inabordable desde el punto de vista computacional y por ello se buscan otras técnicas que permitan alcanzar una probabilidad de error similar a la de la solución ML sin tanta carga computacional. Las versiones SIC de los algoritmos tardan más tiempo que su versión simple y se observa que los algoritmos ZF y MMSE (como ya se vio en el Capítulo 1) tienen una carga similar y menor a la del algoritmo GTA. El algoritmo EP tiene una carga computacional menor que el algoritmo GTA (tanto su versión estándar como SIC), pero mayor que los algoritmos ZF y MMSE.

[13] En este caso se han tomado 100 realizaciones de la matriz de canal y por cada matriz de canal se envían 10 mensajes, lo que supone un total de 1000 mensajes enviados.

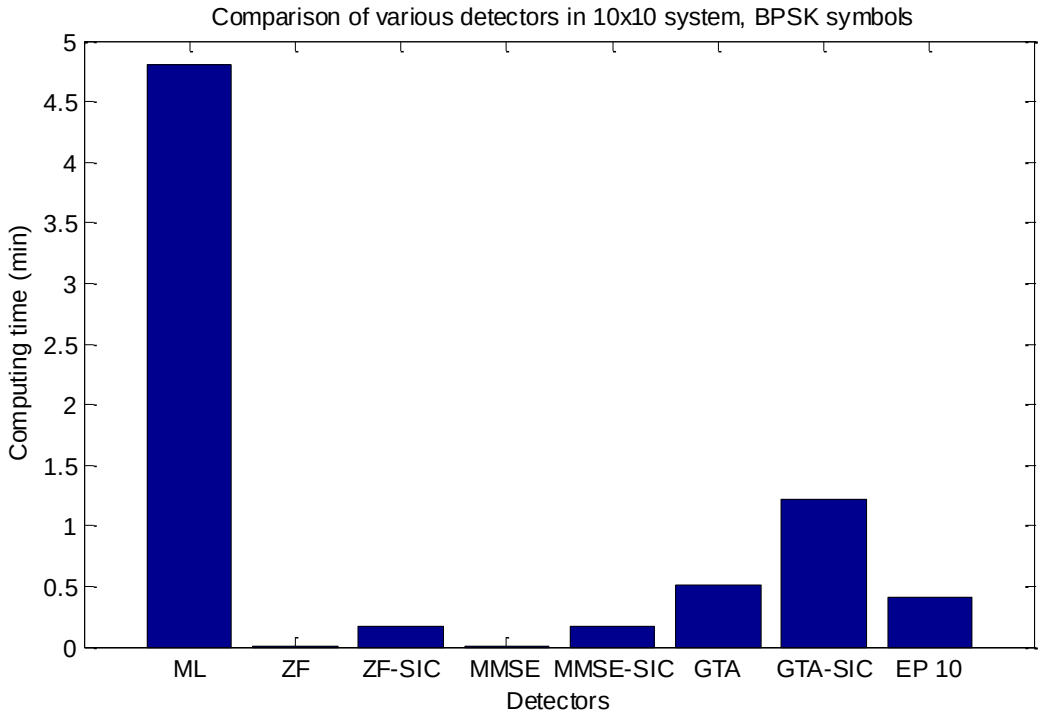

Figura 6-8. Resultados de tiempo de simulación en Matlab para un sistema MIMO 10x10 y símbolos BPSK.

7 ESTRUCTURA DEL CÓDIGO MATLAB

La programación en Matlab de este trabajo se estructura de la siguiente manera. Existe una función para cada algoritmo de detección que devuelve la probabilidad de error de dicho algoritmo para un sistema dado. Estas funciones requieren de unos parámetros de entrada obligatorios, aunque también pueden añadirse una serie de parámetros opcionales (Apartado 7.2.2). Gracias a un script principal (cuyo código se deja en este apartado) se llaman a cada una de estas funciones y se obtienen los gráficos del Capítulo 6.

En mathworks se ha publicado como toolbox todas las funciones realizadas en este trabajo que devuelven la probabilidad de error de un sistema según el algoritmo de detección elegido. Además, se ha dejado un ejemplo (example.m) que simula un sistema MIMO 12x12 con símbolos BPSK y 100 matrices de canal distintas (transmitiendo 1000 vectores por cada matriz de canal), llama a las funciones específicas de cada algoritmo de detección para obtener su probabilidad de error y dibuja estas probabilidades frente a la SNR (este código se deja en el Apartado 7.3). La dirección web donde se encuentra este toolbox es:

http://www.mathworks.com/matlabcentral/fileexchange/48212-detection-mimo-toolbox

Además de las funciones y ejemplo anterior, en el toolbox se incluye un pdf explicativo con un resumen de la teoría de cada algoritmo de detección y una explicación de las funciones. El contenido de este pdf se puede ver en el Anexo B.

En los Anexos C y D se encuentra el código Matlab de todas las funciones utilizadas en este trabajo (y por tanto publicadas en el toolbox de mathworks) y una explicación detallada dicho código, respectivamente.

7.1 Listado de funciones

- SER=ML(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)
  Devuelve la probabilidad de error para cada una de las SNRs (en dB) especificadas como parámetro de entrada (obligatorio) empleando la solución óptima maximum likelihood.

- SER=ZF(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)
  Devuelve la probabilidad de error para cada una de las SNRs (en dB) especificadas como parámetro de entrada (obligatorio) empleando el algoritmo zero-forcing.

- SER=ZF_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)
  Devuelve la probabilidad de error para cada una de las SNRs (en dB) especificadas como parámetro de entrada (obligatorio) empleando el algoritmo zero-forcing con cancelación sucesiva de interferencias.

- SER=MMSE(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)
  Devuelve la probabilidad de error para cada una de las SNRs (en dB) especificadas como parámetro de entrada (obligatorio) empleando el algoritmo minimum mean-square error.

- SER=MMSE_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)
  Devuelve la probabilidad de error para cada una de las SNRs (en dB) especificadas como parámetro de entrada (obligatorio) empleando el algoritmo minimum mean-square error con cancelación sucesiva de interferencias.

- SER=GTA(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)
  Devuelve la probabilidad de error para cada una de las SNRs (en dB) especificadas como parámetro de entrada (obligatorio) empleando el algoritmo gaussian tree approximation.

- SER=GTA_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)
  Devuelve la probabilidad de error para cada una de las SNRs (en dB) especificadas como parámetro de entrada (obligatorio) empleando el algoritmo gaussian tree approximation con cancelación sucesiva de interferencias.

- SER=EP(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP,x,n,H)
  Devuelve la probabilidad de error para cada una de las SNRs (en dB) especificadas como parámetro de entrada (obligatorio) empleando el algoritmo expectation propagation con L_EP iteraciones.

7.2 Explicación de las funciones

Todas las funciones de este trabajo devuelven un vector con las probabilidades de error de cada una de las SNRs dadas como parámetros de entrada. Estas funciones necesitan unos parámetros de entrada (obligatorios) que definen el sistema a simular. Estos parámetros se muestran en la Tabla 7–1.

Tabla 7–1. Parámetros de entrada obligatorios en las funciones del código Matlab.

Parámetros obligatorios	Nomenclatura
Tamaño de la constelación	L
Tipo de constelación (compleja o real)	complex_flag
Número de antenas transmisoras	N
Número de antenas receptoras	M
Número de simulaciones Montecarlo	n_iterations
Número de vectores transmitidos por canal	n_symbols
Vector fila con la SNRs (en dB) a simular	SNRdB
Número de iteraciones del algoritmo EP	L_EP

En el caso de querer comparar varios algoritmos de detección, el sistema debe ser exactamente el mismo en todos los algoritmos, es decir, la señal real transmitida (**x**), el ruido aleatorio (**n**) y las matrices de canal (**H**) no pueden diferir de un algoritmo a otro. Por este motivo, las funciones de este trabajo admiten como parámetros opcionales dichas variables. Los parámetros opcionales se muestran en la Tabla 7–2.

Tabla 7–2. Parámetros de entrada opcionales en las funciones del código Matlab.

Parámetros opcionales	Nomenclatura
Señal transmitida	x
Ruido (de varianza unidad)	n
Matrices de canal	H

Una vez dados todos los parámetros, cada función aplica un algoritmo de detección determinado para obtener una estimación de la señal transmitida, $\hat{\mathbf{x}}$. Comparando esta estimación $\hat{\mathbf{x}}$ con la señal real transmitida **x**, se obtiene el número de errores para cada SNR (que se denotará por $n\_errores$). Teniendo en cuenta que cada vector transmitido tiene tamaño N (es decir, el número de antenas transmisoras), que existen $n\_iterations$ canales diferentes y que por cada canal se transmiten $n\_symbols$ vectores, el número total de símbolos a decodificar son $N \cdot n\_iterations \cdot n\_symbols$. Por tanto, la probabilidad de error para una SNR dada se

calcula como

$$P_e = \frac{n\_errores}{N \cdot n\_iterations \cdot n\_symbols} \tag{7-1}$$

7.2.1 Limitaciones

Como se vio en el Apartado 1.1.1, este trabajo sólo considera sistemas *N*x*M* reales, es decir, resuelve el caso de constelaciones tipo PAM. Sin embargo, es posible simular constelaciones QAM duplicando las dimensiones del sistema y convirtiendo la costelación QAM en una PAM, i.e, para modelar un sistema *N*x*M* con una constelación *L*-QAM se consideraría que el sistema es de tamaño *2N*x*2M* y se procedería de igual forma que para una $2^{log_2 L/2}$-PAM.

7.2.2 Parámetros entrada/salida

Todas las funciones son del tipo[14]:

SER=algorithm_name(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)

donde los parámetros son los siguientes:

- Parámetros de entrada obligatorios:
  - L: Número positivo que indica el tamaño de la constelación.
  - c_flag: Número binario que indica si la constelación es real (valor 0) o compleja (valor 1).
  - n_iterations: Número positivo que indica el número de canales distintos a simular.
  - n_symbols: Número positivo que indica el número de vectores que se transmiten por cada canal.
  - N: Número positivo que indica el número de antenas transmisoras.
  - M: Número positivo que indica el número de antenas receptoras (en el caso del algoritmo ZF y ZF_SIC, M tiene que ser mayor o igual que N, ya que en caso contrario se obtienen matrices singulares).
  - SNRdB: Vector fila con cada una de las SNRs (en dB) a simular.
  - L_EP: Constante positiva que indica el número de iteraciones del algoritmo EP[15].
- Parámetros de entrada opcionales:
  - x: Matriz de dimensiones *N*x*n_symbols*x*n_iterations*, en la que cada columna contiene la señal de entrada transmitida por cada *n_symbols* y *n_iterations*. Cada elemento del vector será algún símbolo de la constelación dada.
  - n: Matriz de dimensiones *M*x*n_symbols*x*n_iterations*, en la que cada columna contiene el vector de ruido de varianza unidad por cada *n_symbols* y *n_iterations*.
  - H: Matriz de dimensiones *M*x*N*x*n_iterations*, que contiene las n_iterations matrices de canal de dimensión *M*x*N* cada una de ellas.
- Variable de salida:
  - SER: Vector columna con el mismo tamaño que el parámetro de entrada SNRdB, en el que cada elemento SER_i corresponde a la probabilidad de error para la $SNRdB_i$.

En caso de no especificar los parámetros opcionales, la llamada a la función sería:

SER=algorithm_name(L,c_flag,n_iterations,n_symbols,N,M,SNRdB)

[14] Excepto la del algoritmo EP, que necesita una entrada más para indicar el número de interaciones utilizadas en el algoritmo (L_EP). El tipo de esta función es: SER=algorithm_name(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP,x,n,H)

[15] Este parámetro sólo existe para la función EP (Ver nota 14).

7.3 Script principal

Como el objetivo de este trabajo es comparar distintos algoritmos para el sistema, es necesario generar la señal transmitida, vector de ruido y matrices de canal y pasárselos como parámetros de entrada opcionales a las funciones del Apartado 7.1. Estas funciones devuelven la probabilidad de error para cada una de las SNRs. Con las instrucciones 'tic' y 'toc' de matlab se calcula también el tiempo de cálculo de cada una de dichas funciones.

A estas funciones se les puede llamar de dos formas:

- Únicamente se les pasa los parámetros y dentro de las funciones se realizan todos los bucles (for) necesarios para repetir el algoritmo con todas las matrices de canal y vectores transmitidos por cada canal.
- La segunda opción aprovecha la ejecución en paralelo de los bucles for en Matlab mediante el comando parfor. En este caso, el bucle for destinado a las distintas matrices de canal se realiza fuera de las funciones, pasándole a éstas una sóla matriz de canal. De esta forma, se ejecutan en paralelo los algoritmos para distintas matrices de canal.

Finalmente, se dibujan las probabilidades de error (en escala logarítmica) frente a las SNRs y el tiempo de cálculo según el algoritmo utilizado, dando lugar a las gráficas del Capítulo 6.

El código utilizado para ello es:

```
%% Define a system
L=2;    % constalation size

complex_flag=0;  % 0 - real     1 - complex

n_iterations=100;     % number of different channels H
n_symbols=10;    % number transmitted vector for each channel H

N=12;    % number of transmitt antennas
M=12;    % number of receve antennas

SNRdB=6:2:50; % Signal to Noise Ratios

L_EP=10; % Number of EP iterations

% If system is complex, it is necessary to convert it in a real one
if complex_flag==1
    N=N*2;
    M=M*2;
    L=2^(log2(L)/2); % Converting L-QAM in a PAM
end

%% Create x, n and H in order to compare various algorithms
% To compare algorithms, it is necessary that x, n and H do not change
% between calls to a function of different algoritm.

% create message and noise (of unit variance)
x=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
n=randn(M,n_symbols,n_iterations);

% Channel matrix (normal random matrix with zero mean and unit variance)
if complex_flag==0
    H=randn(M,N,n_iterations);
else
    H=zeros(M,N,n_iterations);
    H(:,1:N/2,:)=randn(M,N/2,n_iterations);
    H(1:M/2,N/2+1:end,:)=-H(M/2+1:end,1:N/2,:);
    H(M/2+1:end,N/2+1:end,:)=H(1:M/2,1:N/2,:);
end
```

```
%% Calls to functions (that return SER)
% Two ways of calling functions
% 1.- For loop of n_iterations is inside functions
% tic; SER_ML=ML(L,complex_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H);
time_ML=toc;
% tic; SER_ZF=ZF(L,complex_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H);
time_ZF=toc;
% tic;
SER_ZF_SIC=ZF_SIC(L,complex_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H);
time_ZF_SIC=toc;
% tic; SER_MMSE=MMSE(L,complex_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H);
time_MMSE=toc;
% tic;
SER_MMSE_SIC=MMSE_SIC(L,complex_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H);
time_MMSE_SIC=toc;
% tic; SER_GTA=GTA(L,complex_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H);
time_GTA=toc;
% tic;
SER_GTA_SIC=GTA_SIC(L,complex_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H);
time_GTA_SIC=toc;
% tic; SER_EP=EP(L,complex_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP,x,n,H);
time_EP=toc;

% 2.- For loop of n_iterations is outside functions and use parloop
% Initialitation
SER_ML=zeros(length(SNRdB),1); SER_EP=zeros(length(SNRdB),1);
SER_ZF=zeros(length(SNRdB),1); SER_ZF_SIC=zeros(length(SNRdB),1);
SER_MMSE=zeros(length(SNRdB),1); SER_MMSE_SIC=zeros(length(SNRdB),1);
SER_GTA=zeros(length(SNRdB),1); SER_GTA_SIC=zeros(length(SNRdB),1);
time_ML=0; time_EP=0; time_ZF=0; time_ZF_SIC=0;
time_MMSE=0; time_MMSE_SIC=0; time_GTA=0; time_GTA_SIC=0;

parfor i=1:n_iterations
    tic;
SER_ML=SER_ML+ML(L,complex_flag,1,n_symbols,N,M,SNRdB,x(:,:,i),n(:,:,i),H(:,:
,i))/n_iterations; time_ML=time_ML+toc;
    tic;
SER_ZF=SER_ZF+ZF(L,complex_flag,1,n_symbols,N,M,SNRdB,x(:,:,i),n(:,:,i),H(:,:
,i))/n_iterations; time_ZF=time_ZF+toc;
    tic;
SER_ZF_SIC=SER_ZF_SIC+ZF_SIC(L,complex_flag,1,n_symbols,N,M,SNRdB,x(:,:,i),n(
:,:,i),H(:,:,i))/n_iterations; time_ZF_SIC=time_ZF_SIC+toc;
    tic;
SER_MMSE=SER_MMSE+MMSE(L,complex_flag,1,n_symbols,N,M,SNRdB,x(:,:,i),n(:,:,i)
,H(:,:,i))/n_iterations; time_MMSE=time_MMSE+toc;
    tic;
SER_MMSE_SIC=SER_MMSE_SIC+MMSE_SIC(L,complex_flag,1,n_symbols,N,M,SNRdB,x(:,:
,i),n(:,:,i),H(:,:,i))/n_iterations; time_MMSE_SIC=time_MMSE_SIC+toc;
    tic;
SER_GTA=SER_GTA+GTA(L,complex_flag,1,n_symbols,N,M,SNRdB,x(:,:,i),n(:,:,i),H(
:,:,i))/n_iterations; time_GTA=time_GTA+toc;
    tic;
SER_GTA_SIC=SER_GTA_SIC+GTA_SIC(L,complex_flag,1,n_symbols,N,M,SNRdB,x(:,:,i)
,n(:,:,i),H(:,:,i))/n_iterations; time_GTA_SIC=time_GTA_SIC+toc;
    tic;
SER_EP=SER_EP+EP(L,complex_flag,1,n_symbols,N,M,SNRdB,L_EP,x(:,:,i),n(:,:,i),
H(:,:,i))/n_iterations; time_EP=time_EP+toc;
end

%% Figures
% Figure one: SER versus SNR of various detectors
```

```
figure, semilogy(SNRdB,SER_ML,'-k',SNRdB,SER_ZF,'o-b',
SNRdB,SER_ZF_SIC,'o:b',SNRdB,SER_MMSE,'x-g',
SNRdB,SER_MMSE_SIC,'x:g',SNRdB,SER_GTA,'*-
r',SNRdB,SER_GTA_SIC,'*:r',SNRdB,SER_EP,'d-c')
xlim([SNRdB(1) SNRdB(end)]), xlabel('SNR (dB)'), ylabel('SER')
legend('ML','ZF','ZF-SIC','MMSE','MMSE-SIC','GTA','GTA-SIC',['EP
',num2str(L_EP)])

if (complex_flag==0 && L==2)
    constalation_name='BPSK';
elseif (complex_flag==0 && L>2)
    constalation_name=[num2str(L),'-PAM'];
elseif (complex_flag==1)
    constalation_name=[num2str(L^2),'-QAM'];
    N=N/2;
    M=M/2;
end

title(['Comparison of various detectors in ',num2str(N),'x',num2str(M),'
system, ', constalation_name, ' symbols']);
axis([SNRdB(1) SNRdB(end) 1e-4 1]);

% Figure two: computing time
figure, bar([time_ML/60 time_ZF/60 time_ZF_SIC/60 time_MMSE/60
time_MMSE_SIC/60 time_GTA/60 time_GTA_SIC/60 time_EP/60])
algorithms={'ML','ZF','ZF-SIC','MMSE','MMSE-SIC','GTA','GTA-SIC',['EP
',num2str(L_EP)]};
set(gca,'XTickLabel',algorithms);
xlabel('Detectors'), ylabel('Computing time (min)')

if (complex_flag==0 && L==2)
    constalation_name='BPSK';
elseif (complex_flag==0 && L>2)
    constalation_name=[num2str(L),'-PAM'];
elseif (complex_flag==1)
    constalation_name=[num2str(L^2),'-QAM'];
end

title(['Comparison of various detectors in ',num2str(N),'x',num2str(M),'
system, ', constalation_name, ' symbols']);
```

8 CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

Los sistemas de comunicaciones actuales utilizan múltiples antenas transmisoras y receptoras, además de constelaciones de alto orden con el objetivo de maximizar la eficiencia espectral. Sin embargo, a medida que aumenta el número de antenas y el tamaño de la constelación, el diseño de un receptor óptimo se hace más difícil. De hecho, la detección de los símbolos transmitidos mediante la solución de máxima verosimilitud (ML) es totalmente irrealizable, ya que la carga computacional aumenta exponencialmente con el número de antenas. Existen otros algoritmos, de menor carga computacional, como son el MMSE y ZF pero sus resultados se quedan bastante lejos de la solución óptima. En este trabajo, se han revisado otros algoritmos alternativos, en concreto, GTA y EP, cuyos resultados mejoran los obtenidos con los algoritmos estándar (a cambio de aumentar la carga computacional).

- Con el algoritmo GTA-SIC se alcanzan probabilidades de error bastante mejores que cualquiera de los algoritmos estándar antes mencionados (ZF y MMSE tanto en su versión normal como con cancelación sucesiva de interferencias), aunque es cierto que no llega a alcanzar los resultados de la solución óptima de máxima verosimilitud. Incluso la versión estándar del algoritmo GTA obtiene mejores probabilidades de error que MMSE-SIC, aunque esta mejora sólo se aprecia para SNRs altas (para SNRs bajas, ambos algoritmos coinciden prácticamente en la probabilidad de error). Sin embargo, la carga computacional de este algoritmo sigue siendo elevada. Aunque no llega a ser irrealizable cómo la solución óptima, GTA (y sobre todo su versión SIC) tiene una carga computacional demasiado elevada para llegar a un resultado que sigue estando lejos de la solución óptima.
- Por otro lado, el algoritmo EP alcanza probabilidad de error menores incluso que el algoritmo GTA-SIC en la mayoría de simulaciones (únicamente empeora la probabilidad de error con SNRs altas y sistemas del orden 16x16 o 32x32). En concreto, para sistemas de alto orden como 64x64 y una constelación 16-QAM puede llegar a mejorar en 5dB para una SER de 10^{-3} al algoritmo GTA-SIC. Además, aunque su carga computacional sigue siendo mayor que la de los algoritmos ZF y MMSE, este algoritmo presenta una carga computacional menor que el algoritmo GTA (tanto en su versión estándar como SIC), por lo que resulta bastante interesante seguir profundizando en el algoritmo EP.

El código Matlab elaborado para las simulaciones de este trabajo sólo admite constelaciones reales, es decir, tipo L-PAM. Para simular constelaciones L-QAM, se convierte previamente el sistema complejo en un sistema real (duplicando sus dimensiones), por lo que una posible línea de investigación sería la extensión de este código a números complejos, con el fin de poder simular cualquier tipo de constelación y sistema complejo.

También sería interesante abordar la versión SIC del algoritmo EP con el fin de intentar mejorar aún más la probabilidad de error que ofrece su versión estándar. Sin embargo, dado que este algoritmo ya es, por si mismo, iterativo es posible que la carga computacional aumente de forma considerable sin aportar demasiados beneficios. Como alternativa, podría pensarse alguna modificación sobre el algoritmo EP estándar que decodificara inicialmente aquellos símbolos que están poco afectados por el ruido, cancelara su efecto y

aplicara el algoritmo EP en el resto de símbolos, reduciendo así el tamaño de las ecuaciones del algoritmo EP y por tanto su carga computacional.

Por último, en este trabajo se ha demostrado la mejora que ofrece el algoritmo EP frente a otros algoritmos, por lo que se podría seguir aplicando en otros ámbitos que no fuera la decodificación MIMO, como por ejemplo, igualación o codificación.

ANEXO A: DEMOSTRACIONES

A.1 Desarrollo de la ecuación (3-7)

$$\exp\left(-\frac{1}{2\sigma^2}\|\mathbf{Hx}-\mathbf{y}\|^2\right)=\prod_i \exp\left(-\frac{1}{2\sigma^2}(\langle\mathbf{H}\rangle_{\mathrm{i}}\mathbf{x}-y_i)^2\right)$$

Demostración:

Primero se desarrolla el vector $\mathbf{Hx}-\mathbf{y}$ de la primera expresión, al que luego se le calculará la norma.

$$\mathbf{Hx}-\mathbf{y}=\begin{bmatrix}H_{11} & \cdots & H_{1N}\\ \vdots & \ddots & \vdots\\ H_{M1} & \cdots & H_{MN}\end{bmatrix}\begin{bmatrix}x_1\\ \vdots\\ x_N\end{bmatrix}-\begin{bmatrix}y_1\\ \vdots\\ y_M\end{bmatrix}=\begin{bmatrix}H_{11}x_1+\cdots+H_{1N}x_N-y_1\\ \vdots\\ H_{M1}x_1+\cdots+H_{MN}x_N-y_M\end{bmatrix}=\begin{bmatrix}\langle\mathbf{H}\rangle_1\mathbf{x}-y_1\\ \vdots\\ \langle\mathbf{H}\rangle_{\mathrm{M}}\mathbf{x}-y_M\end{bmatrix}$$

donde $\langle\mathbf{H}\rangle_{\mathrm{i}}$ denota la fila *i*-ésima de la matriz **H**.

La norma euclídea de un vector cualquiera $\mathbf{v}=(v_1,\ldots,v_M)^{\mathrm{T}}$ es

$$\|\mathbf{v}\|=\sqrt{v_1^2+\cdots+v_M^2}$$

Aplicando esta definición al vector $\mathbf{Hx}-\mathbf{y}$ se obtiene que

$$\|\mathbf{Hx}-\mathbf{y}\|^2=(\langle\mathbf{H}\rangle_1\mathbf{x}-y_1)^2+\cdots+(\langle\mathbf{H}\rangle_{\mathrm{M}}\mathbf{x}-y_M)^2=\sum_{i=1}^{M}(\langle\mathbf{H}\rangle_{\mathrm{i}}\mathbf{x}-y_i)^2$$

Finalmente, desarrollando la exponencial del primer término de la expresión del enunciado, se llega al producto de exponenciales del segundo término

$$\exp\left(-\frac{1}{2\sigma^2}\|\mathbf{Hx}-\mathbf{y}\|^2\right)=\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{M}(\langle\mathbf{H}\rangle_{\mathrm{i}}\mathbf{x}-y_i)^2\right)=\prod_{\mathrm{i}=1}^{\mathrm{M}}\exp\left(-\frac{1}{2\sigma^2}(\langle\mathbf{H}\rangle_{\mathrm{i}}\mathbf{x}-y_i)^2\right)$$

■

A.2 Desarrollo de la ecuación (4-3)

Lemma A.1 (Función densidad de probabilidad en zero forcing)

La expresión de la función densidad de probabilidad de la salida de un detector zero forcing, antes del decisor, toma la expresión:

$$\mathcal{N}(\mathbf{y}\colon \mathbf{Hx},\sigma^2\mathbf{I})\propto\mathcal{N}(\mathbf{x}\colon \mathbf{z},\mathbf{C})$$

donde

$$\mathbf{C}=\sigma^2(\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\qquad \text{y}\qquad \mathbf{z}=\mathbf{C}\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{y}$$

Demostración:

Primero se desarrolla el primer término.

$$\mathcal{N}(\mathbf{y}\colon \mathbf{Hx}, \sigma^2\mathbf{I}) \propto \exp\left(-\frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{Hx})^{\mathrm{T}}(\mathbf{y}-\mathbf{Hx})\right)$$
$$= \exp\left(-\frac{1}{2\sigma^2}\mathbf{y}^{\mathbf{T}}\mathbf{y} + \frac{1}{\sigma^2}(\mathbf{Hx})^{\mathrm{T}}\mathbf{y} - \frac{1}{2\sigma^2}(\mathbf{Hx})^{\mathrm{T}}(\mathbf{Hx})\right)$$
$$= \exp\left(-\frac{1}{2\sigma^2}\mathbf{x}^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathbf{Hx} + \frac{1}{\sigma^2}\mathbf{x}^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathbf{y} + cte_x\right)$$

Ahora el segundo término.

$$\mathcal{N}(\mathbf{x}\colon \mathbf{z}, \mathbf{C}) \propto \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{z})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{x}-\mathbf{z})\right) = \exp\left(-\frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{C}^{-1}\mathbf{x} + \mathbf{x}^{\mathbf{T}}\mathbf{C}^{-1}\mathbf{z} + cte\right)$$

Comparando ambas expresiones término a término, se obtienen el valor de **z** y **C** en función de **H**, **y**, σ

$$\frac{1}{\sigma^2}\mathbf{H}^{\mathrm{T}}\mathbf{H} = \mathbf{C}^{-1} \;\Leftrightarrow\; \mathbf{C} = \sigma^2(\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}$$

$$\frac{1}{\sigma^2}\mathbf{H}^{\mathrm{T}}\mathbf{y} = \mathbf{C}^{-1}\mathbf{z} \;\Leftrightarrow\; \mathbf{z} = \mathbf{C}\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{y}$$

■

A.3 Desarrollo de la ecuación (5-20)

$$\mathcal{N}(\mathbf{y}\colon \mathbf{Hx}, \sigma^2\mathbf{I})\prod_{i=1}^{N}\exp\left(\gamma_i x_i - \frac{1}{2}\Lambda_{\mathrm{i}}{x_i}^2\right) \propto \mathcal{N}(\mathbf{x}\colon \boldsymbol{\mu}_{\mathbf{EP}}, \sum_{\mathbf{EP}})$$

donde

$$\sum_{\mathbf{EP}} = \left(\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{H} + \mathrm{diag}(\boldsymbol{\Lambda})\right)^{-1}$$
$$\boldsymbol{\mu}_{\mathbf{EP}} = \sum_{\mathbf{EP}}\left(\sigma^{-2}\mathbf{H}^{\mathrm{T}} + \boldsymbol{\gamma}\right)$$

Demostración:

El producto de las exponenciales (gaussianas sin normalizar) de una sola variable puede expresarse como una única gaussiana multivariable de la forma

$$\prod_{i=1}^{N}\exp\left(\gamma_i x_i - \frac{1}{2}\Lambda_i{x_i}^2\right) = \exp\left(\gamma_1 x_1 - \frac{1}{2}\Lambda_1{x_1}^2 + \cdots + \gamma_N x_N - \frac{1}{2}\Lambda_N{x_N}^2\right)$$
$$= \exp\left(\mathbf{x}^{\mathbf{T}}\boldsymbol{\gamma} - \frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathrm{diag}(\boldsymbol{\Lambda})\mathbf{x}\right)$$

donde $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_N)^{\mathrm{T}}$ y $\boldsymbol{\Lambda} = (\Lambda_1, \ldots, \Lambda_N)^{\mathrm{T}}$. El operador $\mathrm{diag}(\boldsymbol{\Lambda})$ devuelve una matriz diagonal con la diagonal dada por **Λ**.

Si ahora se desarrolla la expresión de una gaussiana multivariable de media **μ** y covarianza $\sum$

$$\mathcal{N}(\mathbf{x}\colon \boldsymbol{\mu}, \sum) \propto \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathbf{T}}\sum^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) = \exp\left(-\frac{1}{2}\mathbf{x}^{\mathbf{T}}\sum^{-1}\mathbf{x} + \mathbf{x}^{\mathbf{T}}\sum^{-1}\boldsymbol{\mu} + cte\right) \qquad \text{(A-1)}$$

y se compara con la exponencial multivariable del principio de la demostración, se obtiene que su media ($\boldsymbol{\mu}_{\mathbf{exp}}$) y covarianza ($\sum_{\mathbf{exp}}$) es

$$\mathrm{diag}(\boldsymbol{\Lambda}) = \sum^{-1} \;\Leftrightarrow\; \sum_{\mathbf{exp}} = \mathrm{diag}(\boldsymbol{\Lambda})^{-1}$$

$$\boldsymbol{\gamma} = \sum^{-1}\boldsymbol{\mu} \;\Leftrightarrow\; \boldsymbol{\mu}_{\mathbf{exp}} = \sum_{\mathbf{exp}}\boldsymbol{\gamma}$$

y por tanto, se puede expresar como

$$\exp\left(\boldsymbol{\gamma}^{\mathbf{T}}\mathbf{x} - \frac{1}{2}\mathrm{diag}(\boldsymbol{\Lambda})\mathbf{x}\mathbf{x}^{\mathbf{T}}\right) \propto \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{exp}}\right)^{\mathbf{T}}\sum\nolimits_{\mathbf{exp}}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{exp}}\right)\right)$$

Por otro lado, la gaussiana $\mathcal{N}(\mathbf{y}\colon \mathbf{H}\mathbf{x}, \sigma^2\mathbf{I})$ es equivalente a una gaussiana de media $\mathbf{z} = \mathbf{C}\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{y}$ y covarianza $\mathbf{C} = \sigma^2(\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}$ (demostrado en el Apartado A.1).

Agrupando lo visto hasta ahora, (5-20) se puede escribir como un único producto de dos gaussianas que, nuevamente, es una gaussiana de la forma

$$\begin{aligned}\mathcal{N}(\mathbf{y}\colon \mathbf{H}\mathbf{x}, \sigma^2\mathbf{I}) \prod_{i=1}^{N} \exp\left(\gamma_i x_i - \frac{1}{2}\Lambda_i {x_i}^2\right) &\\ \propto \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{z})^{\mathbf{T}}\mathbf{C}^{-1}(\mathbf{x}-\mathbf{z})\right) &\exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{exp}}\right)^{\mathbf{T}}\sum\nolimits_{\mathbf{exp}}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{exp}}\right)\right)\\ = \exp\left(-\frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{C}^{-1}\mathbf{x} + \mathbf{x}^{\mathbf{T}}\mathbf{C}^{-1}\mathbf{z} + cte_1 - \frac{1}{2}\mathbf{x}^{\mathbf{T}}\sum\nolimits_{\mathbf{exp}}^{-1}\mathbf{x} + \mathbf{x}^{\mathbf{T}}\sum\nolimits_{\mathbf{exp}}^{-1}\boldsymbol{\mu}_{\mathbf{exp}} + cte_2\right)&\\ \propto \exp\left(-\frac{1}{2}\mathbf{x}^{\mathbf{T}}\left(\mathbf{C}^{-1} + \sum\nolimits_{\mathbf{exp}}^{-1}\right)\mathbf{x} + \mathbf{x}^{\mathbf{T}}\left(\mathbf{C}^{-1}\mathbf{z} + \sum\nolimits_{\mathbf{exp}}^{-1}\boldsymbol{\mu}_{\mathbf{exp}}\right)\right)&\end{aligned}$$

donde $\mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}$ y $\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} = \mathbf{C}\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{y}$.

Comparando nuevamente con la gaussiana multivariable (A-1), se obtienen la media (5-21) y covarianza (5-22)

$$\left(\mathbf{C}^{-1} + \sum\nolimits_{\mathbf{exp}}^{-1}\right) = \sum\nolimits_{\mathbf{EP}}^{-1} \;\Leftrightarrow\; \sum\nolimits_{\mathbf{EP}} = \left(\mathbf{C}^{-1} + \sum\nolimits_{\mathbf{exp}}^{-1}\right)^{-1} = \left(\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{H} + \mathrm{diag}(\boldsymbol{\Lambda})\right)^{-1}$$

$$\mathbf{C}^{-1}\mathbf{z} + \sum\nolimits_{\mathbf{exp}}^{-1}\boldsymbol{\mu}_{\mathbf{exp}} = \sum\nolimits_{\mathbf{EP}}^{-1}\boldsymbol{\mu}_{\mathbf{EP}} \;\Leftrightarrow\; \boldsymbol{\mu}_{\mathbf{EP}} = \sum\nolimits_{\mathbf{EP}}\left(\mathbf{C}^{-1}\mathbf{z} + \sum\nolimits_{\mathbf{exp}}^{-1}\boldsymbol{\mu}_{\mathbf{exp}}\right) = \sum\nolimits_{\mathbf{EP}}\left(\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{y} + \boldsymbol{\gamma}\right)$$

■

A.4 Desarrollo de la ecuación (5-24)

$$\frac{\mathcal{N}(x_i\colon {\mu_i}^{(\ell)}, {\sigma_i}^{2(\ell)})}{\exp\left(\gamma_i{}^{(\ell)}x_i - \frac{1}{2}\Lambda_{\mathrm{i}}{}^{(\ell)}{x_i}^2\right)} \sim \mathcal{N}(x_i\colon {t_i}^{(\ell)}, {h_i}^{2(\ell)})$$

donde

$${h_i}^{2(\ell)} = \frac{{\sigma_i}^{2(\ell)}}{1 - {\sigma_i}^{2(\ell)}\Lambda_{\mathrm{i}}{}^{(\ell)}}$$

$${t_i}^{(\ell)} = {h_i}^{2(\ell)}\left(\frac{{\mu_i}^{(\ell)}}{{\sigma_i}^{2(\ell)}} - \gamma_i{}^{(\ell)}\right)$$

Demostración:

El primer término es un producto de dos gaussianas, por lo que el resultado es otra gaussiana de la forma

$$\frac{\mathcal{N}(x_i: {\mu_i}^{(\ell)}, {\sigma_i}^{2(\ell)})}{\exp\left(\gamma_i{}^{(\ell)} x_i - \frac{1}{2}\Lambda_i{}^{(\ell)} {x_i}^2\right)} \propto \exp\left(-\frac{1}{2{\sigma_i}^{2(\ell)}}\left(x_i - {\mu_i}^{(\ell)}\right)^2\right)\exp\left(-\gamma_i{}^{(\ell)} x_i + \frac{1}{2}\Lambda_i{}^{(\ell)} {x_i}^2\right)$$

$$= \exp\left(-\frac{{x_i}^2}{2{\sigma_i}^{2(\ell)}} - \frac{{\mu_i}^{2(\ell)}}{2{\sigma_i}^{2(\ell)}} + \frac{x_i {\mu_i}^{(\ell)}}{{\sigma_i}^{2(\ell)}} - \gamma_i{}^{(\ell)} x_i + \frac{1}{2}\Lambda_i{}^{(\ell)} {x_i}^2\right)$$

$$= \exp\left(-\frac{1}{2}{x_i}^2\left(\frac{1}{{\sigma_i}^{2(\ell)}} - \Lambda_i{}^{(\ell)}\right) + x_i\left(\frac{{\mu_i}^{(\ell)}}{{\sigma_i}^{2(\ell)}} - \gamma_i{}^{(\ell)}\right) + cte\right)$$

Desarrollando ahora el segundo término

$$\mathcal{N}(x_i: {t_i}^{(\ell)}, {h_i}^{2(\ell)}) \propto \exp\left(-\frac{1}{2{h_i}^{2(\ell)}}\left(x_i - {t_i}^{(\ell)}\right)^2\right) = \exp\left(-\frac{1}{2}{x_i}^2\left(\frac{1}{{h_i}^{2(\ell)}}\right) + x_i\left(\frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}}\right) + cte\right)$$

Comparando ambas expresiones término a término, se obtienen el valor de ${t_i}^{(\ell)}$ (5-25) y ${h_i}^{2(\ell)}$ (5-26).

$$\frac{1}{{\sigma_i}^{2(\ell)}} - \Lambda_i{}^{(\ell)} = \frac{1}{{h_i}^{2(\ell)}} \;\Leftrightarrow\; {h_i}^{2(\ell)} = \frac{{\sigma_i}^{2(\ell)}}{1 - {\sigma_i}^{2(\ell)}\Lambda_i{}^{(\ell)}}$$

$$\frac{{\mu_i}^{(\ell)}}{{\sigma_i}^{2(\ell)}} - \gamma_i{}^{(\ell)} = \frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}} \;\Leftrightarrow\; {t_i}^{(\ell)} = {h_i}^{2(\ell)}\left(\frac{{\mu_i}^{(\ell)}}{{\sigma_i}^{2(\ell)}} - \gamma_i{}^{(\ell)}\right)$$

■

A.5 Desarrollo de las ecuaciones (5-29) y (5-30)

Imponer que ${\mu_{p_i}}^{(\ell)}$ y ${\sigma_{p_i}}^{2(\ell)}$ sean la media y la varianza de $\mathcal{N}(x_i: {t_i}^{(\ell)}, {h_i}^{2(\ell)})\exp\left(\gamma_i{}^{(\ell+1)} x_i -$ *12*Λi*(ℓ+1)xi2* resulta en una pareja (*γi(ℓ+1)*, Λi*(ℓ+1)*) tal que

$$\Lambda_i{}^{(\ell+1)} = \frac{1}{{\sigma_{p_i}}^{2(\ell)}} - \frac{1}{{h_i}^{2(\ell)}}$$

$$\gamma_i{}^{(\ell+1)} = \frac{{\mu_{p_i}}^{(\ell)}}{{\sigma_{p_i}}^{2(\ell)}} - \frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}}$$

Demostración:

El primer factor de la distribución es una gaussiana univariable que, como se demostró en el Apartado A.4, puede expresarse de la forma

$$\mathcal{N}(x_i: {t_i}^{(\ell)}, {h_i}^{2(\ell)}) \propto \exp\left(-\frac{1}{2}{x_i}^2\left(\frac{1}{{h_i}^{2(\ell)}}\right) + x_i\left(\frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}}\right) + cte\right)$$

Como la distribución del enunciado es un producto de dos gaussianas, el resultado es otra gaussiana

$$\mathcal{N}\left(x_i: {t_i}^{(\ell)}, {h_i}^{2(\ell)}\right)\exp\left(\gamma_i{}^{(\ell+1)} x_i - \frac{1}{2}\Lambda_i{}^{(\ell+1)} {x_i}^2\right)$$

$$\propto \exp\left(-\frac{1}{2}{x_i}^2\left(\frac{1}{{h_i}^{2(\ell)}}\right) + x_i\left(\frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}}\right) + cte + \gamma_i{}^{(\ell+1)} x_i - \frac{1}{2}\Lambda_i{}^{(\ell+1)} {x_i}^2\right)$$

$$= \exp\left(-\frac{1}{2}{x_i}^2\left(\frac{1}{{h_i}^{2(\ell)}} + \Lambda_i{}^{(\ell+1)}\right) + x_i\left(\frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}} + \gamma_i{}^{(\ell+1)}\right) + cte\right)$$

Desarrollando los términos que dependen de x_i en una distribución gaussiana genérica de una variable

$$\mathcal{N}(x_i: \mu, \sigma^2) \propto \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu)^2\right) = \exp\left(-\frac{1}{2}{x_i}^2\left(\frac{1}{\sigma^2}\right) + x_i\left(\frac{\mu}{\sigma^2}\right) + cte\right)$$

y comparándolo con la distribución anterior, se obtiene la media y varianza de la distribución del enunciado

$\mathcal{N}(x_i: {t_i}^{(\ell)}, {h_i}^{2(\ell)})\exp\left(\gamma_i{}^{(\ell+1)}x_i - \frac{1}{2}\Lambda_i{}^{(\ell+1)}{x_i}^2\right)$

$$\frac{1}{{h_i}^{2(\ell)}} + \Lambda_i{}^{(\ell+1)} = \frac{1}{\sigma^2} \;\Leftrightarrow\; \sigma^2 = \frac{{h_i}^{2(\ell)}}{1 + {h_i}^{2(\ell)}\Lambda_i{}^{(\ell+1)}}$$

$$\frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}} + \gamma_i{}^{(\ell+1)} = \frac{\mu}{\sigma^2} \;\Leftrightarrow\; \mu = \sigma^2\left(\frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}} + \gamma_i{}^{(\ell+1)}\right)$$

Esta media (μ) y varianza (σ^2) hay que igualarlas a ${\mu_{p_i}}^{(\ell)}$ y ${\sigma_{p_i}}^{2(\ell)}$, respectivamente y de ahí despejar los valores de la pareja ($\gamma_i{}^{(\ell+1)}$, $\Lambda_i{}^{(\ell+1)}$)

$$\sigma^2 = \frac{{h_i}^{2(\ell)}}{1 + {h_i}^{2(\ell)}\Lambda_i{}^{(\ell+1)}} = {\sigma_{p_i}}^{2(\ell)} \;\Leftrightarrow\; \Lambda_i{}^{(\ell+1)} = \frac{1}{{\sigma_{p_i}}^{2(\ell)}} - \frac{1}{{h_i}^{2(\ell)}}$$

$$\mu = {\sigma_{p_i}}^{2(\ell)}\left(\frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}} + \gamma_i{}^{(\ell+1)}\right) = {\mu_{p_i}}^{(\ell)} \;\Leftrightarrow\; \gamma_i{}^{(\ell+1)} = \frac{{\mu_{p_i}}^{(\ell)}}{{\sigma_{p_i}}^{2(\ell)}} - \frac{{t_i}^{(\ell)}}{{h_i}^{2(\ell)}}$$

■

ANEXO B: PDF TOOLBOX

B.1 Theory behind the functions

The aim of this chapter is to introduce the MIMO communication systems and described some detection algorithms used in these systems. This chapter only tries to provide a brief description of such algorithms (but do not explain the theory behind them) and it's recommended that users refer to the bibliography listed at the end of this document.

B.1.1 MIMO Systems

We consider a MIMO communication system with *N* transmit antennas and *M* receive antennas [1]. The transmitted symbol vector is a Nx1 i.i.d. complex vector $\mathbf{x} = (x_1, \dots, x_N)^{\mathrm{T}}$, where each component $x_i \in \mathcal{A}$. The symbols are transmitted over a complex MIMO channel defined by the *M*x*N* channel matrix **H**, which comprises i.i.d. elements drawn from a complex normal distribution of zero mean and unit variance. The received vector $\mathbf{y} = (y_1, \dots, y_M)^{\mathrm{T}}$ is given by

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v} \tag{B.0-1}$$

where $\mathbf{v} = (v_1, \dots, v_M)^{\mathrm{T}}$ is and additive Gaussian random vector with independent zero mean components and σ^2 variance.

The MIMO detection problem consists of finding the unknown transmitted vector **x** given **H** and **y**. It is convenient to reformulate the complex-valued MIMO system into a real-valued one. It can be translated into an equivalent double-size real-valued representation that is obtained by considering the real and imaginary parts separately

$$\begin{bmatrix} \mathrm{IRe}(\mathbf{y}) \\ \mathrm{IIm}(\mathbf{y}) \end{bmatrix} = \begin{bmatrix} \mathrm{IRe}(\mathbf{H}) & \mathrm{IIm}(\mathbf{H}) \\ -\mathrm{IIm}(\mathbf{H}) & \mathrm{IRe}(\mathbf{H}) \end{bmatrix} \begin{bmatrix} \mathrm{IRe}(\mathbf{x}) \\ \mathrm{IIm}(\mathbf{x}) \end{bmatrix} + \begin{bmatrix} \mathrm{IRe}(\mathbf{v}) \\ \mathrm{IIm}(\mathbf{v}) \end{bmatrix} \tag{B.0-2}$$

This rule will be applied in the functions of the Detection MIMO Toolbox, so that we only focus in real values and *L*-PAM constelations.

B.1.2 Maximum likelihood (ML) solution

The maximum likelihood (ML) solution [4] is

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathcal{A}^N} \|\mathbf{y} - \mathbf{H}\mathbf{x}\|^2 \tag{B.0-3}$$

Where $\mathcal{A}^N$ denotes the finite set of possible symbols in a *L*-PAM constelation.

However, going over all the $|\mathcal{A}|^N$ vectors is unfeasible when either *N* or *L* is large.

B.1.3 Zero-forcing (ZF) algorithm

This sub-optimal solution is base don a linear decisión that ignores the finite-set constraint of transmitted symbols [4]

$$\mathbf{z} = \mathbf{H}^{\dagger}\mathbf{y} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} \tag{B.0-4}$$

where $\mathbf{H}^{\dagger}$ denotes the pseudo-inverse of $\mathbf{H}$.

Then, it finds the closest point in $\mathcal{A}$ for each symbol independently

$$\hat{x}_i = \arg\min_{a\in\mathcal{A}} |z_i - a| \tag{B.0-5}$$

This algorithm performs poorly due to its inability to handle ill-conditioned realizations of the matrix $\mathbf{H}$ (if $M<N$ then $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ is even singular).

B.1.4 Minimum mean square error (MMSE) algorithm

A better perfonmance can be obtained by using a Bayesian estimation for the continuous linear system [4]

$$\mathbf{E}(\mathbf{x}|\mathbf{y}) = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} \tag{B.0-6}$$

where E_s is the mean symbol energy.

Then, it finds the closest point in $\mathcal{A}$ for each symbol independently

$$\hat{x}_i = \arg\min_{a\in\mathcal{A}} |\mathrm{E}(\mathbf{x}|y_i) - a| \tag{B.0-7}$$

B.1.5 Nulling and cancelling with optimal ordering

In this method, the estimation (B.0-5)-(B.0-7) is used for only one of the entries of $\mathbf{x}$, say k_1. Then this entry, x_{k_1}, is assumed to be known and its effect is cancelled out to obtain a reduced-order least-squeares problem with N-1 unknowns. The process is repeated to find x_{k_2}, etc. [4]-[6]

The k_i indexes are chosen to minimize the effects of error propagation, so we start from the "strongest" to the "weakest" signal[16].

This algorithm can be described as a recursive procedure, including determination of the optimal ordering, as follows

$$\mathbf{y_1} \coloneqq \mathbf{y}$$

$$\mathbf{G_1} = \begin{cases} \mathbf{H}^{\dagger} & \text{if ZF} \\ \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}} & \text{if MMSE} \end{cases}$$

$$\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ N$$

$$k_i = \arg\min_{\mathrm{j}\notin(\mathrm{k}_1,\ldots,\mathrm{k}_{\mathrm{i}-1})} \left\|\langle\mathbf{G_i}\rangle_{\mathrm{j}}\right\|^2$$

$$\boldsymbol{\omega}_{\mathbf{k_i}}^{\mathrm{T}} = \langle\mathbf{G_i}\rangle_{\mathrm{k_i}}$$

$$\hat{x}_{k_i} \coloneqq \mathrm{Q}\left(\boldsymbol{\omega}_{\mathbf{k_i}}^{\mathrm{T}}\mathbf{y_i}\right)$$

$$\mathbf{y_{i+1}} = \mathbf{y_i} - \mathbf{H}_{\mathrm{k_i}}\hat{x}_{k_i}$$

$$\mathbf{G_{i+1}} = \begin{cases} \mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}}^{\dagger} & \text{if ZF} \\ \left(\mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}}^{\mathrm{T}}\mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}}^{\mathrm{T}} & \text{if MMSE} \end{cases}$$

$$\boldsymbol{end}$$

where $\langle\mathbf{H}\rangle_{\mathrm{i}}$ is the ith row of $\mathbf{H}$, $\mathbf{H}_{\mathrm{i}}$ is the ith column of $\mathbf{H}$, $\mathbf{H}_{\bar{\mathbf{k}}_{\mathbf{i}}}$ denotes the matrix obtained by zeroing columns $x_{k_1}, \ldots, x_{k_i}$ of H and $\mathrm{Q}(\cdot)$ denotes the quantization operation appropriate to the constellation in use.

[16] The "strongest" signal is the one with the smallest variance.

B.1.6 Gaussian tree approximation (GTA) algorithm

Given the constrained linear system specified in paragraph B.1.1, the probability function of the discrete random vector **x** given **y** is

$$p(\mathbf{x}|\mathbf{y}) \propto \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{H}\mathbf{x}-\mathbf{y}\|^2\right) \qquad \mathbf{x} \in \mathcal{A}^N \tag{B.0-8}$$

Assuming a non-informative prior, (B.0-8) can be written as follows

$$p(\mathbf{x}|\mathbf{y}) \propto \exp\left(-\frac{1}{2}(\mathbf{z}-\mathbf{x})^T\mathbf{C}^{-1}(\mathbf{z}-\mathbf{x})\right) \qquad \mathbf{x} \in \mathcal{A}^N \tag{B.0-9}$$

where $\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y}$ and $\mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}$.

GTA algorithm [1] first ignores the discrete nature of the prior $p(\mathbf{x})$, so it rewrites (B.0-9) as

$$f(\mathbf{x}|\mathbf{y}) \propto \exp\left(-\frac{1}{2}(\mathbf{z}-\mathbf{x})^T\mathbf{C}^{-1}(\mathbf{z}-\mathbf{x})\right) \qquad \mathbf{x} \in \mathcal{R}^N \tag{B.0-10}$$

Then it searchs for a tree approximation of the distribution (B.0-10), which is also a Gaussian distribution

$$f_{GTA}(\mathbf{x}) = \prod_{i=1}^{N} f\left(x_i|x_{p(i)}\right) \qquad \mathbf{x} \in \mathcal{R}^N \tag{B.0-11}$$

Given the Gaussian tree approximation (B.0-11) of the Gaussian distribution (B.0-10), the next step is applying the finite-set constraint to form a discrete loop-free approximation of (B.0-8)

$$p_{GTA}(\mathbf{x}|\mathbf{y}) \propto \prod_{i=1}^{N} f\left(x_i|x_{p(i)}\right) \qquad \mathbf{x} \in \mathcal{A}^N \tag{B.0-12}$$

Using BP algorithm we can efficiently obtain the exact marginal distributions of (B.0-12)

$$p_{GTA}(x_i|\mathbf{y}) \qquad x_i \in \mathcal{A}, \quad i = 1, \ldots, N \tag{B.0-13}$$

That provides a soft decision result. Taking the most likely symbol, we obtain the GTA solution

$$\hat{x}_i = \arg\max_a p_{GTA}(a|\mathbf{y}) \qquad a \in \mathcal{A} \tag{B.0-14}$$

The GTA algorithm is summarized as

- Inputs:
  A constrained linear system $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v}$, a noise variance σ^2 and a finite symbol set $\mathcal{A}$ whose mean symbol energy is E_s

- Algorithm:
  - Compute
    $$\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y} \quad , \quad \mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}$$
  - Denote

$$f(x_i;\mathbf{z},\mathbf{C}) = \exp\left(-\frac{1}{2}\frac{(x_i - z_i)^2}{C_{ii}}\right)$$

$$f(x_i|x_j;\mathbf{z},\mathbf{C}) = \exp\left(-\frac{1}{2}\frac{\left((x_i - z_i) - \frac{C_{ij}}{C_{jj}}(x_j - z_j)\right)^2}{C_{ii} - \frac{C_{ij}^2}{C_{jj}}}\right)$$

- Compute the maximum spanning tree of the *N*-node graph where the weight of the *i-j* edge is the square of the correlation coeficiente

$$\rho_{ij}^2 = \frac{C_{ij}^2}{C_{ii}C_{jj}}$$

- Asumme that the tree is rooted at node '1' and denote the parent of node i by p(i). Apply BP on the loop free distribution

$$\hat{p}(x_1,\dots,x_N|\boldsymbol{y}) \propto f(x_1;\mathbf{z},\mathbf{C})\prod_{i=2}^{N} f(x_i|x_{p(i)};\mathbf{z},\mathbf{C})$$

to find the most likely configuration.

B.1.7 Gaussian tree approximation with successive interference cancelation (GTA-SIC) algorithm

In each iteration we apply the GTA algorithm and obtain a hard decision of the single (most reliable) symbol. We cancel its contribution and obtain a system with a smaller number of unknown variables. This process is iterated until all the message symbols are decoded. [2]

The GTA-SIC algorithm is summarized as

- Inputs:
  A constrained linear system $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v}$, a noise variance σ^2 and a finite symbol set $\mathcal{A}$ whose mean symbol energy is E_s

- Algorithm:
  $\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ N$
  - Compute
    $\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y}$, $\mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}$
  - Denote

$$f(x_k;\mathbf{z},\mathbf{C}) = \exp\left(-\frac{1}{2}\frac{(x_k - z_k)^2}{C_{kk}}\right)$$

$$f(x_k|x_j;\mathbf{z},\mathbf{C}) = \exp\left(-\frac{1}{2}\frac{\left((x_k - z_k) - \frac{C_{kj}}{C_{jj}}(x_j - z_j)\right)^2}{C_{kk} - \frac{C_{kj}^2}{C_{jj}}}\right)$$

  - Optimal ordering:

$$k_i = \arg\min_{j\notin(k_1,\dots,k_{i-1})} C_{k_i k_i}$$

  - Optimal tree: Compute the máximum spanning tree (rooted at node x_{k_i}) of the *N*-(*i*-1)-node graph where the weight of the k-j edge is the square of the correlation coefficient

$$\rho_{kj}^2 = \frac{C_{kj}^2}{C_{kk}C_{jj}}$$

  - BP algorithm: Send messages from leaves to root ('downward' messages). The message from *k* to its parent *j=p(k)* is

$$m_{k\to p(k)}\left(x_{p(k)}\right)=\sum_{x_k\in\mathcal{A}} f\left(x_k|x_{p(k)};\mathbf{z},\mathbf{C}\right)\prod_{j\,|\,p(j)=k} m_{j\to k}(x_k) \qquad x_{p(k)}\in\mathcal{A}$$

- Decision:

$$\hat{x}_{k_i}=\arg\max_{a}\left(f(a;\mathbf{z},\mathbf{C})\prod_{j\,|\,p(j)=k_i} m_{j\to k_i}(a)\right) \qquad a\in\mathcal{A}$$

- Interference cancelation:

$$\mathbf{y}=\mathbf{y}-\mathbf{H}_{\mathrm{k_i}}\hat{x}_{k_i}$$
$$\mathbf{H}=\mathbf{H}_{\overline{\mathbf{k_i}}}$$

end

B.1.8 Expectation propagation (EP) algorithm

The symbol posterior distribution (B.0-8) can be expressed as

$$p(\mathbf{x}|\mathbf{y})\propto\mathcal{N}(\mathbf{y}:\mathbf{H}\mathbf{x},\sigma^2\mathbf{I})\prod_{i=1}^{N}\mathbb{I}_{x_i\in\mathcal{A}} \tag{B.0-15}$$

where $\mathbb{I}_{x_i\in\mathcal{A}}$ is the indicator function that takes value one if $x_i\in\mathcal{A}$ and zero otherwise.

Given this factorization of (B.0-8), EP method approximates it by a Gaussian approximation $q(\mathbf{x})$ [14]-[16]. This approximation is obtained by replacing each one of the non-Gaussian factors in (B.0-15) by an unnormalized Gaussian [13]

$$q_{EP}(\mathbf{x})\propto\mathcal{N}(\mathbf{y}:\mathbf{H}\mathbf{x},\sigma^2\mathbf{I})\prod_{i=1}^{N}\exp\left(\gamma_i x_i-\frac{1}{2}\Lambda_{\mathrm{i}}{x_i}^2\right)\propto\mathcal{N}(\mathbf{x}:\boldsymbol{\mu}_{\mathbf{EP}},\boldsymbol{\Sigma}_{\mathbf{EP}}) \tag{B.0-16}$$

where

$$\boldsymbol{\Sigma}_{\mathbf{EP}}=\left(\sigma^{-2}\mathbf{H}^{\mathrm{T}}\mathbf{H}+\mathrm{diag}(\boldsymbol{\Lambda})\right)^{-1} \tag{B.0-17}$$

$$\boldsymbol{\mu}_{\mathbf{EP}}=\boldsymbol{\Sigma}_{\mathbf{EP}}\left(\sigma^{-2}\mathbf{H}^{\mathrm{T}}+\boldsymbol{\gamma}\right) \tag{B.0-18}$$

The EP purpose is update the pairs (γ_i, Λ_{i}) for *i*=1, …, *N* recursively and recompute mean vector and covariance matrix of (B.0-17) and (B.0-18).

The EP algorithm is summarized as

3. Initialize ${\gamma_i}^{(0)}=0$ and ${\Lambda_i}^{(0)}={E_s}^{-1}$ for all $i=1,\ldots,N$. If we do not continue with the next steps, it would give the MMSE solution.

   We denote with (ℓ) the iterations of the EP algorithm.

4. **for** $\ell=0$ **to** L_{EP}

   a. Compute the *i*-th marginal of the distribution $q^{(\ell)}(\mathbf{x})$ of (B.0-16) as ${q_i}^{(\ell)}(x_i)=\mathcal{N}(x_i:{\mu_i}^{(\ell)},{\sigma_i}^{2(\ell)})$ where ${\mu_i}^{(\ell)}$ and ${\sigma_i}^{2(\ell)}$ are obtained by replacing $\boldsymbol{\gamma}^{(\ell)}$ y $\boldsymbol{\Lambda}^{(\ell)}$ in (B.0-17) and (B.0-18).

   b. Remove exponentials independently (one by one) of the distribution $q^{(\ell)}(\mathbf{x})$, i.e, compute

$$q^{(\ell)\backslash i}(x_i)=\frac{{q_i}^{(\ell)}(x_i)}{\exp\left(\gamma_i{}^{(\ell)}x_i-\frac{1}{2}{\Lambda_{\mathrm{i}}}^{(\ell)}{x_i}^2\right)}\sim\mathcal{N}(x_i:{t_i}^{(\ell)},{h_i}^{2(\ell)}) \tag{B.0-19}$$

   where

$$h_i{}^{2(\ell)} = \frac{\sigma_i{}^{2(\ell)}}{1 - \sigma_i{}^{2(\ell)}\Lambda_i{}^{(\ell)}} \tag{B.0-20}$$

$$t_i{}^{(\ell)} = h_i{}^{2(\ell)}\left(\frac{\mu_i{}^{(\ell)}}{\sigma_i{}^{2(\ell)}} - \gamma_i{}^{(\ell)}\right) \tag{B.0-21}$$

c. Compute the mean $\mu_{p_i}{}^{(\ell)}$ and variance $\sigma_{p_i}{}^{2(\ell)}$ of the distribution

$$\hat{p}_i{}^{(\ell)}(x_i) \propto q^{(\ell)\setminus i}(x_i)\mathbb{I}_{x_i \in \mathcal{A}} = \begin{cases} q^{(\ell)\setminus i}(x_i) & si\ x_i \in \mathcal{A} \\ 0 & otherwise \end{cases} \tag{B.0-22}$$

d. Update the pair $(\gamma_i{}^{(\ell+1)}, \Lambda_i{}^{(\ell+1)})$ so that the following unnormalized Gaussian distribution

$$q^{(\ell)\setminus i}(x_i)\exp\left(\gamma_i{}^{(\ell+1)}x_i - \frac{1}{2}\Lambda_i{}^{(\ell+1)}x_i{}^2\right) \tag{B.0-23}$$

Has mean $\mu_{p_i}{}^{(\ell)}$ and variance $\sigma_{p_i}{}^{2(\ell)}$. The solution is given by

$$\Lambda_i{}^{(\ell+1)} = \frac{1}{\sigma_{p_i}{}^{2(\ell)}} - \frac{1}{h_i{}^{2(\ell)}} \tag{B.0-24}$$

$$\gamma_i{}^{(\ell+1)} = \frac{\mu_{p_i}{}^{(\ell)}}{\sigma_{p_i}{}^{2(\ell)}} - \frac{t_i{}^{(\ell)}}{h_i{}^{2(\ell)}} \tag{B.0-25}$$

end

B.2 Fuction reference

The aim of this chapter is to give a brief description of the functions used in Detection MIMO toolbox. Apart from the functions, we give an example script (‘example.m’) that plots SNRs vs SER for different dectection algorithms (i.e, using all the functions of Detection MIMO toolbox) for a 12x12 system, BPSK constalation, a frame length of 10 channel uses and 100 realizations of the channel matrix.

B.2.1 Function structure

All functions used in this toolbox have the same structure. They need various input parameters, some of them are obligatory and others are optional. Finally, the return a vector of SER. Typology of functions is

SER=algorithm_name(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x,n,H)

where parameters are:

- Obligatory inputs
  - *L*: A positive number with constalation size.
  - *c_flag*: A binary number, which indicates if constalation is real (value '0') or complex (value '1').
  - *n_iterations*: A positive number with the number of different channels to simulate.
  - *n_symbols*: A positive number with the number of vectors whose are transmitted by each channel **H**.
  - *N*: A positive number with the number of transmitt antennas.
  - *M*: A positive number with the number of receve antennas (for functions ZF and ZF_SIC, it must be greater than *N* because in other case, matrices are singular).

- **SNRdB**: Row-vector with each SNR (in dB) to simulate.

- Optional inputs
  - **x**: *N*x*n_symbols*x*n_iterations*-dimensional matrix, where each column has the transmitted signal by each *n_symbol* and *n_iterations*.
  - **n**: *M*x*n_symbols*x*n_iterations*-dimensional matrix, where each column has the noise vector (with unit variance) by each *n_symbol* and *n_iterations*.
  - **H**: *M*x*N*x*n_iterations*-dimensional matrix, with the *n_iterations* different channel matrices **H** (*M*x*N*-dimensional).

- Output
  - **SER**: Column-vector with the same size than **SNRdB**, that indicates error probability of each SNR, i.e, *SER(i)* has the signal error rate asociated with *SNRdB(i)*.

Optional inputs are only used in case we want to compare the same system (same **H** and **y**) in various functions of Detection MIMO toolbox.

B.2.2 Function description

B.2.2.1 ML

Syntax

```
SER=ML(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
```

Description

`ML` returns a vector of error probability (SER) (where each element correspond to each SNRdB element given as parameter: SER(i)--> SNRdB(i)) using Maximum Likelihood (ML) solution.

Obligatory parameters: constalation size (L>0), complex flag to indicate if the symbols of constalation are complex (c_flag={0,1}), number of different channels H (n_iterations>0), number of transmitted vector for each channel H (n_symbols>0), number of transmitt antennas (N>0), number of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).

Optional parameters: N x n_symbols x n_iterations matrix of sended messages in a PAM constalation (x_all), M x n_symbols x n_iterations matrix of noise generated (n_all) and MxNxn_iterations Channel matrices (n_iterations matrices, whose size is MxN by each one) (H_all).

`SER=ML(L,c_flag,n_iterations,n_symbols,N,M,SNRdB)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are unknown for the user.

`SER=ML(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are given by the user. This option is useful when we want to compare various algorithms for the same system (x_all, n_all and H_all)

Limitations: This code is only for real systems; i.e, L-PAM constalations. If we want to compute L-QAM constalation, c_flag must be set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM

Example

Compute SER of BPSK constalation of a 12x12 system, using a frame length of 1000, 100 realizations of the channel matrix and ML solution.

```
>> SER=ML(2,0,100,1000,12,12,6:2:50);
```

B.2.2.2 ZF

Syntax

```
SER=ZF(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
```

Description

`ZF` returns a vector of error probability (SER) (where each element correspond to each SNRdB element given as parameter: SER(i)--> SNRdB(i)) using Zero Forcing (ZF) algorithm.

Obligatory parameters: constalation size (L>0), complex flag to indicate if the symbols of constalation are complex (c_flag={0,1}), number of different channels H (n_iterations>0), number of transmitted vector for each channel H (n_symbols>0), number of transmitt antennas (N>0), number of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).

Optional parameters: N x n_symbols matrix x n_iterations of sended messages in a PAM constalation (x_all), M x n_symbols x n_iterations matrix of noise generated (n_all) and MxNxn_iterations Channel matrices (n_iterations matrices, whose size is MxN by each one) (H_all).

`SER=ZF(L,c_flag,n_iterations,n_symbols,N,M,SNRdB)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are unknown for the user.

`SER=ZF(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are given by the user. This option is useful when we want to compare various algorithms for the same system (x_all, n_all and H_all)

Limitations: This code is only for real systems; i.e, L-PAM constalations. If we want to compute L-QAM constalation, c_flag must be set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM

Example

Compute SER of BPSK constalation of a 12x12 system, using a frame length of 1000, 100 realizations of the channel matrix and ZF algorithm.

```
>> SER=ZF(2,0,100,1000,12,12,6:2:50);
```

B.2.2.3 ZF_SIC

Syntax

```
SER=ZF_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
```

Description

`ZF_SIC` returns a vector of error probability (SER) (where each element correspond to each SNRdB element given as parameter: SER(i)--> SNRdB(i)) using Zero Forcing with Successive Interference Cancelation (ZF-SIC) algorithm.

Obligatory parameters: constalation size (L>0), complex flag to indicate if the symbols of constalation are complex (c_flag={0,1}), number of different channels H (n_iterations>0), number of transmitted vector for each channel H (n_symbols>0), number of transmitt antennas (N>0), number of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).

Optional parameters: N x n_symbols x n_iterations matrix of sended messages in a PAM constalation (x_all), M x n_symbols x n_iterations matrix of noise generated (n_all) and MxNxn_iterations Channel matrices (n_iterations matrices, whose size is MxN by each one) (H_all).

`SER=ZF_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are unknown for the

user.

`SER=ZF_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are given by the user. This option is useful when we want to compare various algorithms for the same system (x_all, n_all and H_all)

Limitations: This code is only for real systems; i.e, L-PAM constalations. If we want to compute L-QAM constalation, c_flag must be set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM

Example

Compute SER of BPSK constalation of a 12x12 system, using a frame length of 1000, 100 realizations of the channel matrix and ZF-SIC algorithm.

```
>> SER=ZF_SIC(2,0,100,1000,12,12,6:2:50);
```

B.2.2.4 MMSE

Syntax

```
SER=MMSE(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
```

Description

`MMSE` returns a vector of error probability (SER) (where each element correspond to each SNRdB element given as parameter: SER(i)--> SNRdB(i)) using Minimum Mean Square Error (MMSE) algorithm.

Obligatory parameters: constalation size (L>0), complex flag to indicate if the symbols of constalation are complex (c_flag={0,1}), number of different channels H (n_iterations>0), number of transmitted vector for each channel H (n_symbols>0), number of transmitt antennas (N>0), number of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).

Optional parameters: N x n_symbols x n_iterations matrix of sended messages in a PAM constalation (x_all), M x n_symbols x n_iterations matrix of noise generated (n_all) and MxNxn_iterations Channel matrices (n_iterations matrices, whose size is MxN by each one) (H_all).

`SER=MMSE(L,c_flag,n_iterations,n_symbols,N,M,SNRdB)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are unknown for the user.

`SER=MMSE(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are given by the user. This option is useful when we want to compare various algorithms for the same system (x_all, n_all and H_all)

Limitations: This code is only for real systems; i.e, L-PAM constalations. If we want to compute L-QAM constalation, c_flag must be set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM

Example

Compute SER of BPSK constalation of a 12x12 system, using a frame length of 1000, 100 realizations of the channel matrix and MMSE algorithm.

```
>> SER=MMSE(2,0,100,1000,12,12,6:2:50);
```

B.2.2.5 MMSE_SIC

Syntax

```
SER=MMSE_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
```

Description

`MMSE_SIC` returns a vector of error probability (SER) (where each element correspond to each SNRdB element given as parameter: SER(i)--> SNRdB(i)) using Minimum Mean Square Error with Successive Interference Cancelation (MMSE-SIC) algorithm.

Obligatory parameters: constalation size (L>0), complex flag to indicate if the symbols of constalation are complex (c_flag={0,1}), number of different channels H (n_iterations>0), number of transmitted vector for each channel H (n_symbols>0), number of transmitt antennas (N>0), number of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).

Optional parameters: N x n_symbols x n_iterations matrix of sended messages in a PAM constalation (x_all), M x n_symbols x n_iterations matrix of noise generated (n_all) and MxNxn_iterations Channel matrices (n_iterations matrices, whose size is MxN by each one) (H_all).

`SER=MMSE_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are unknown for the user.

`SER=MMSE_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are given by the user. This option is useful when we want to compare various algorithms for the same system (x_all, n_all and H_all)

Limitations: This code is only for real systems; i.e, L-PAM constalations. If we want to compute L-QAM constalation, c_flag must be set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM

Example

Compute SER of BPSK constalation of a 12x12 system, using a frame length of 1000, 100 realizations of the channel matrix and MMSE-SIC algorithm.

```
>> SER=MMSE_SIC(2,0,100,1000,12,12,6:2:50);
```

B.2.2.6 GTA

Syntax

```
SER=GTA(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
```

Description

`GTA` returns a vector of error probability (SER) (where each element correspond to each SNRdB element given as parameter: SER(i)--> SNRdB(i)) using Gaussian Tree Aproximation (GTA) algorithm.

Obligatory parameters: constalation size (L>0), complex flag to indicate if the symbols of constalation are complex (c_flag={0,1}), number of different channels H (n_iterations>0), number of transmitted vector for each channel H (n_symbols>0), number of transmitt antennas (N>0), number of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).

Optional parameters: N x n_symbols x n_iterations matrix of sended messages in a PAM constalation (x_all), M x n_symbols x n_iterations matrix of noise generated (n_all) and MxNxn_iterations Channel matrices (n_iterations matrices, whose size is MxN by each one) (H_all).

`SER=GTA(L,c_flag,n_iterations,n_symbols,N,M,SNRdB)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are unknown for the user.

`SER=GTA(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all)

are given by the user. This option is useful when we want to compare various algorithms for the same system (x_all, n_all and H_all)

Limitations: This code is only for real systems; i.e, L-PAM constalations. If we want to compute L-QAM constalation, c_flag must be set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM

Example

Compute SER of BPSK constalation of a 12x12 system, using a frame length of 1000, 100 realizations of the channel matrix and GTA algorithm.

```
>> SER=GTA(2,0,100,1000,12,12,6:2:50);
```

B.2.2.7 GTA_SIC

Syntax

```
SER=GTA_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
```

Description

`GTA_SIC` returns a vector of error probability (SER) (where each element correspond to each SNRdB element given as parameter: SER(i)--> SNRdB(i)) using Gaussian Tree Aproximation with Successive Interference Cancelation (GTA-SIC) algorithm.

Obligatory parameters: constalation size (L>0), complex flag to indicate if the symbols of constalation are complex (c_flag={0,1}), number of different channels H (n_iterations>0), number of transmitted vector for each channel H (n_symbols>0), number of transmitt antennas (N>0), number of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).

Optional parameters: N x n_symbols x n_iterations matrix of sended messages in a PAM constalation (x_all), M x n_symbols x n_iterations matrix of noise generated (n_all) and MxNxn_iterations Channel matrices (n_iterations matrices, whose size is MxN by each one) (H_all).

`SER=GTA_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are unknown for the user.

`SER=GTA_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are given by the user. This option is useful when we want to compare various algorithms for the same system (x_all, n_all and H_all)

Limitations: This code is only for real systems; i.e, L-PAM constalations. If we want to compute L-QAM constalation, c_flag must be set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM

Example

Compute SER of BPSK constalation of a 12x12 system, using a frame length of 1000, 100 realizations of the channel matrix and GTA-SIC algorithm.

```
>> SER=GTA_SIC(2,0,100,1000,12,12,6:2:50);
```

B.2.2.8 EP

Syntax

```
SER=EP(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP,x_all,n_all,H_all)
```

Description

`EP` returns a vector of error probability (SER) (where each element correspond to each SNRdB element given as parameter: SER(i)--> SNRdB(i)) using Expectation Propagation (EP) algorithm.

Obligatory parameters: constalation size (L>0), complex flag to indicate if the symbols of constalation are complex (c_flag={0,1}), number of different channels H (n_iterations>0), number of transmitted vector for each channel H (n_symbols>0), number of transmitt antennas (N>0), number of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB), number of EP iterations (L_EP>0).

Optional parameters: N x n_symbols x n_iterations matrix of sended messages in a PAM constalation (x_all), M x n_symbols x n_iterations matrix of noise generated (n_all) and MxNxn_iterations Channel matrices (n_iterations matrices, whose size is MxN by each one) (H_all).

`SER=EP(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are unknown for the user.

`SER=EP(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP,x_all,n_all,H_all)` computes signal error rate for a system whose sended message (x_all), noise vector (n_all) and Channel matrices (H_all) are given by the user. This option is useful when we want to compare various algorithms for the same system (x_all, n_all and H_all)

Limitations: This code is only for real systems; i.e, L-PAM constalations. If we want to compute L-QAM constalation, c_flag must be set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM

Example

Compute SER of BPSK constalation of a 12x12 system, using a frame length of 1000, 100 realizations of the channel matrix and EP algorithm with 10 iterations.

```
>> SER=EP(2,0,100,1000,12,12,6:2:50,10);
```

ANEXO C: CÓDIGO MATLAB

C.1 Función ML

```
function SER=ML(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)

% Function: ML
% Author:   Irene Santos Velázquez, irenesantos@us.es
% Created:  01/08/2014
%
% Description: This function returns a vector of error probability (SER)
% (where each element correspond to each SNRdB element given as parameter:
% SER(i)--> SNRdB(i)) using Maximum Likelihood (ML) solution.
%
% Obligatory parameters: constalation size (L>0), complex flag to indicate
% if the symbols of constalation are complex (c_flag={0,1}), number of
% different channels H (n_iterations>0), number of transmitted vector for
% each channel H (n_symbols>0), number of transmitt antennas (N>0), number
% of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).
%
% Optional parameters: N x n_symbols matrix x n_iterations of sended
% messages in a PAM constalation (x_all), M x n_symbols x n_iterations
% matrix of noise generated (n_all) and MxNxn_iterations Channel matrices
% (n_iterations matrices, whose size is MxN by each one) (H_all).
%
% SER=ML(L,c_flag,n_iterations,n_symbols,N,M,SNRdB) computes signal error
% rate for a system whose sended message (x_all), noise vector (n_all) and
% Channel matrices (H_all) are unknown for the user.
%
% SER=ML(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
% computes signal error rate for a system whose sended message (x_all),
% noise vector (n_all) and Channel matrices (H_all) are given by the user.
% This option is useful when we want to compare various algorithms for the
% same system (x_all, n_all and H_all)
%
% Limitations: This code is only for real systems; i.e, L-PAM
% constalations. If we want to compute L-QAM constalation, c_flag must be
% set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM
% constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM
%
% Example: Compute SER of BPSK constalation of a 12x12 system, using a
% frame length of 1000, 100 realizations of the channel matrix and ML
% solution.
% SER=ML(2,0,100,1000,12,12,6:2:50);
%
% Copyright (c) 2014 Irene Santos
```

```
% Checking if x_all,n_all and H_all are given
if nargin<8
    if c_flag==1
        N=N*2;
        M=M*2;
        L=2^(log2(L)/2);
        H_all=zeros(M,N,n_iterations);
        H_all(:,1:N/2,:)=randn(M,N/2,n_iterations);
        H_all(1:M/2,N/2+1:end,:)=-H_all(M/2+1:end,1:N/2,:);
        H_all(M/2+1:end,N/2+1:end,:)=H_all(1:M/2,1:N/2,:);
    else
        H_all=randn(M,N,n_iterations);
    end
    x_all=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
    n_all=randn(M,n_symbols,n_iterations);
end

% Alphabet
A=-(L-1):2:(L-1);

energy= (L*L-1)/3; % PAM energy

n_errors=zeros(length(SNRdB),1); % number of errors get for each SNR

for loop1=1:n_iterations

    H=H_all(:,:,loop1); % normal random matrix with zero mean and unit
variance

    for ind_db=1:length(SNRdB)
        sigma=sqrt(N*energy*10^(-SNRdB(ind_db)/10));
        err=0;

        parfor loop2=1:n_symbols
            x=x_all(:,loop2,loop1);
            % apply noisy linear
            y=H*x+sigma*n_all(:,loop2,loop1);

            x_ml=A(1)*ones(N,1);

            % Initialization of ML
            min_norm=norm(H*x_ml-y);
            x_decod=x_ml;

            for k=1:L^N
                for kk=N:-1:1
                    if (mod(k-1,L^(N-kk))==0 && k~=1)
                        ind=find(A==x_ml(kk)); % Index of alphabet element
                        if ind==L
                            ind=0;
                        end
                        x_ml(kk)=A(ind+1);
                    end
                end

                % ML solution
                norm_act=norm(H*x_ml-y);
                if norm_act<min_norm
                    x_decod=x_ml;
                    min_norm=norm_act;
                end
            end
```

```
            err=err+sum(x~=x_decod);
        end
        n_errors(ind_db)=err;
    end
end

SER=n_errors/(N*n_iterations*n_symbols);
```

C.2 Función ZF

```
function SER=ZF(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)

% Function: ZF
% Author:   Irene Santos Velázquez, irenesantos@us.es
% Created:  01/07/2014
%
% Description: This function returns a vector of error probability (SER)
% (where each element correspond to each SNRdB element given as parameter:
% SER(i)--> SNRdB(i)) using Zero Forcing (ZF) algorithm.
%
% Obligatory parameters: constalation size (L>0), complex flag to indicate
% if the symbols of constalation are complex (c_flag={0,1}), number of
% different channels H (n_iterations>0), number of transmitted vector for
% each channel H (n_symbols>0), number of transmitt antennas (N>0), number
% of received antennas (M>=N), row-vector of Signal to Noise Ratios (SNRdB).
%
% Optional parameters: N x n_symbols x n_iterations matrix of sended
% messages in a PAM constalation (x_all), M x n_symbols x n_iterations
% matrix of noise generated (n_all) and MxNxn_iterations Channel matrices
% (n_iterations matrices, whose size is MxN by each one) (H_all).
%
% SER=ZF(L,c_flag,n_iterations,n_symbols,N,M,SNRdB) computes signal error
% rate for a system whose sended message (x_all), noise vector (n_all) and
% Channel matrices (H_all) are unknown for the user.
%
% SER=ZF(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
% computes signal error rate for a system whose sended message (x_all),
% noise vector (n_all) and Channel matrices (H_all) are given by the user.
% This option is useful when we want to compare various algorithms for the
% same system (x_all, n_all and H_all)
%
% Limitations: This code is only for real systems; i.e, L-PAM
% constalations. If we want to compute L-QAM constalation, c_flag must be
% set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM
% constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM
%
% Example: Compute SER of BPSK constalation of a 12x12 system, using a
% frame length of 1000, 100 realizations of the channel matrix and ZF
% algorithm.
% SER=ZF(2,0,100,1000,12,12,6:2:50);
%
% Copyright (c) 2014 Irene Santos

% Checking if x_all,n_all and H_all are given
if nargin<8
    if c_flag==1
        N=N*2;
        M=M*2;
        L=2^(log2(L)/2);
        H_all=zeros(M,N,n_iterations);
        H_all(:,1:N/2,:)=randn(M,N/2,n_iterations);
```

```
        H_all(1:M/2,N/2+1:end,:)=-H_all(M/2+1:end,1:N/2,:);
        H_all(M/2+1:end,N/2+1:end,:)=H_all(1:M/2,1:N/2,:);
    else
        H_all=randn(M,N,n_iterations);
    end
    x_all=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
    n_all=randn(M,n_symbols,n_iterations);
end

% Alphabet
A=-(L-1):2:(L-1);

energy= (L*L-1)/3; % PAM energy

n_errors=zeros(length(SNRdB),1); % number of errors get for each SNR

for loop1=1:n_iterations

    H=H_all(:,:,loop1); % normal random matrix with zero mean and unit
variance
    x=x_all(:,:,loop1);
    n=n_all(:,:,loop1);

    parfor ind_db=1:length(SNRdB)
        sigma=sqrt(N*energy*10^(-SNRdB(ind_db)/10));

        % apply noisy linear
        y=H*x+sigma*n;

        %z=inv(H'*H)*H'*y;
        z=pinv(H)*y;

        % Decoder
        belief_x=zeros(N*n_symbols,L);
        for i=1:L
            belief_x(:,i)=abs(z(:)-A(i));
        end

        [min_belief,ind]=min(belief_x,[],2);
        x_decod=A(ind)';

        n_errors(ind_db)=sum(x(:)~=x_decod);
    end
end

SER=n_errors/(N*n_iterations*n_symbols);
```

C.3 Función ZF_SIC

```
function
SER=ZF_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)

% Function: ZF_SIC
% Author:   Irene Santos Velázquez, irenesantos@us.es
% Created:  01/08/2014
%
% Description: This function returns a vector of error probability (SER)
% (where each element correspond to each SNRdB element given as parameter:
% SER(i)--> SNRdB(i)) using Zero Forcing with Successive Interference
```

```
% Cancelation (ZF-SIC) algorithm.
%
% Obligatory parameters: constalation size (L>0), complex flag to indicate
% if the symbols of constalation are complex (c_flag={0,1}), number of
% different channels H (n_iterations>0), number of transmitted vector for
% each channel H (n_symbols>0), number of transmitt antennas (N>0), number
% of received antennas (M>=N), row-vector of Signal to Noise Ratios (SNRdB).
%
% Optional parameters: N x n_symbols x n_iterations matrix of sended
% messages in a PAM constalation (x_all), M x n_symbols x n_iterations
% matrix of noise generated (n_all) and MxNxn_iterations Channel matrices
% (n_iterations matrices, whose size is MxN by each one) (H_all).
%
% SER=ZF_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB) computes signal
% error rate for a system whose sended message (x_all), noise vector
% (n_all) and Channel matrices (H_all) are unknown for the user.
%
% SER=ZF_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
% computes signal error rate for a system whose sended message (x_all),
% noise vector (n_all) and Channel matrices (H_all) are given by the user.
% This option is useful when we want to compare various algorithms for the
% same system (x_all, n_all and H_all)
%
% Limitations: This code is only for real systems; i.e, L-PAM
% constalations. If we want to compute L-QAM constalation, c_flag must be
% set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM
% constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM
%
% Example: Compute SER of BPSK constalation of a 12x12 system, using a
% frame length of 1000, 100 realizations of the channel matrix and ZF-SIC
% algorithm.
% SER=ZF_SIC(2,0,100,1000,12,12,6:2:50);
%
% Copyright (c) 2014 Irene Santos

% Checking if x_all,n_all and H_all are given
if nargin<8
    if c_flag==1
        N=N*2;
        M=M*2;
        L=2^(log2(L)/2);
        H_all=zeros(M,N,n_iterations);
        H_all(:,1:N/2,:)=randn(M,N/2,n_iterations);
        H_all(1:M/2,N/2+1:end,:)=-H_all(M/2+1:end,1:N/2,:);
        H_all(M/2+1:end,N/2+1:end,:)=H_all(1:M/2,1:N/2,:);
    else
        H_all=randn(M,N,n_iterations);
    end
    x_all=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
    n_all=randn(M,n_symbols,n_iterations);
end

% Alphabet
A=-(L-1):2:(L-1);

energy= (L*L-1)/3; % PAM energy

n_errors=zeros(length(SNRdB),1); % number of errors get for each SNR

for loop1=1:n_iterations

    H=H_all(:,:,loop1); % normal random matrix with zero mean and unit
variance
```

```
    for ind_db=1:length(SNRdB)
        sigma=sqrt(N*energy*10^(-SNRdB(ind_db)/10));
        err=0;

        parfor loop2=1:n_symbols
            x=x_all(:,loop2,loop1);
            % apply noisy linear
            yi=H*x+sigma*n_all(:,loop2,loop1);

            x_decod=zeros(N,1);

            % SIC (Successive Interference Cancelation)
            Gi=pinv(H);
            %Gi=inv(H'*H)*H';
            Hi=H;
            index=[];
            for i=1:N
                Gi_aux=Gi;
                Gi_aux(index,:)=NaN*Gi_aux(index,:); % We change '0s' of Gi
as 'NaN' (to not consider in calculation of minimal values)
                [norm_min,ki]=min(sum(Gi_aux.^2,2));
                %[norm_min,ki]=min(sum(Gi.^2,2));
                index=[index ki];

                wki=Gi(ki,:)';
                yki=wki'*yi;
                [min_s,ind]=min(abs(yki-A),[],2);
                ski=A(ind);
                x_decod(ki)=ski;
                % Update yi and Gi for next iteration
                yi=yi-ski*Hi(:,ki);
                Hi(:,ki)=0*Hi(:,ki); % We write as '0' the ki-column of H
                %Hi=Hi(:,[1:ki-1,ki+1:end]); % Delete ki-column of H
                Gi=pinv(Hi);
                %Gi=inv(Hi'*Hi)*Hi';
            end
            err=err+sum(x~=x_decod);
        end
        n_errors(ind_db)=err;
    end
end

SER=n_errors/(N*n_iterations*n_symbols);
```

C.4 Función MMSE

```
function
SER=MMSE(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)

% Function: MMSE
% Author:   Irene Santos Velázquez, irenesantos@us.es
% Created:  01/07/2014
%
% Description: This function returns a vector of error probability (SER)
% (where each element correspond to each SNRdB element given as parameter:
% SER(i)--> SNRdB(i)) using Minimum Mean Square Error (MMSE) algorithm.
%
% Obligatory parameters: constalation size (L>0), complex flag to indicate
% if the symbols of constalation are complex (c_flag={0,1}), number of
```

```
% different channels H (n_iterations>0), number of transmitted vector for
% each channel H (n_symbols>0), number of transmitt antennas (N>0), number
% of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).
%
% Optional parameters: N x n_symbols x n_iterations matrix of sended
% messages in a PAM constalation (x_all), M x n_symbols x n_iterations
% matrix of noise generated (n_all) and MxNxn_iterations Channel matrices
% (n_iterations matrices, whose size is MxN by each one) (H_all).
%
% SER=MMSE(L,c_flag,n_iterations,n_symbols,N,M,SNRdB) computes signal error
% rate for a system whose sended message (x_all), noise vector (n_all) and
% Channel matrices (H_all) are unknown for the user.
%
% SER=MMSE(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
% computes signal error rate for a system whose sended message (x_all),
% noise vector (n_all) and Channel matrices (H_all) are given by the user.
% This option is useful when we want to compare various algorithms for the
% same system (x_all, n_all and H_all)
%
% Limitations: This code is only for real systems; i.e, L-PAM
% constalations. If we want to compute L-QAM constalation, c_flag must be
% set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM
% constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM
%
% Example: Compute SER of BPSK constalation of a 12x12 system, using a
% frame length of 1000, 100 realizations of the channel matrix and MMSE
% algorithm.
% SER=MMSE(2,0,100,1000,12,12,6:2:50);
%
% Copyright (c) 2014 Irene Santos

% Checking if x_all, n_all and H_all are given
if nargin<8
    if c_flag==1
        N=N*2;
        M=M*2;
        L=2^(log2(L)/2);
        H_all=zeros(M,N,n_iterations);
        H_all(:,1:N/2,:)=randn(M,N/2,n_iterations);
        H_all(1:M/2,N/2+1:end,:)=-H_all(M/2+1:end,1:N/2,:);
        H_all(M/2+1:end,N/2+1:end,:)=H_all(1:M/2,1:N/2,:);
    else
        H_all=randn(M,N,n_iterations);
    end
    x_all=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
    n_all=randn(M,n_symbols,n_iterations);
end

% Alphabet
A=-(L-1):2:(L-1);

energy= (L*L-1)/3; % PAM energy

n_errors=zeros(length(SNRdB),1); % number of errors get for each SNR

for loop1=1:n_iterations

    H=H_all(:,:,loop1); % normal random matrix with zero mean and unit
variance
    x=x_all(:,:,loop1);
    n=n_all(:,:,loop1);
```

```
    parfor ind_db=1:length(SNRdB)
        sigma=sqrt(N*energy*10^(-SNRdB(ind_db)/10));

        % apply noisy linear
        y=H*x+sigma*n;

        z=inv(H'*H+(sigma^2/energy)*eye(N))*H'*y;

        % Decoder
        belief_x=zeros(N*n_symbols,L);
        for i=1:L
            belief_x(:,i)=abs(z(:)-A(i));
        end

        [min_belief,ind]=min(belief_x,[],2);
        x_decod=A(ind)';

        n_errors(ind_db)=sum(x(:)~=x_decod);
    end
end

SER=n_errors/(N*n_iterations*n_symbols);
```

C.5 Función MMSE_SIC

```
function
SER=MMSE_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)

% Function: MMSE_SIC
% Author:   Irene Santos Velázquez, irenesantos@us.es
% Created:  01/08/2014
%
% Description: This function returns a vector of error probability (SER)
% (where each element correspond to each SNRdB element given as parameter:
% SER(i)--> SNRdB(i)) using Minimum Mean Square Error with Successive
% Interference Cancelation (MMSE-SIC) algorithm.
%
% Obligatory parameters: constalation size (L>0), complex flag to indicate
% if the symbols of constalation are complex (c_flag={0,1}), number of
% different channels H (n_iterations>0), number of transmitted vector for
% each channel H (n_symbols>0), number of transmitt antennas (N>0), number
% of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).
%
% Optional parameters: N x n_symbols x n_iterations matrix of sended
% messages in a PAM constalation (x_all), M x n_symbols x n_iterations
% matrix of noise generated (n_all) and MxNxn_iterations Channel matrices
% (n_iterations matrices, whose size is MxN by each one) (H_all).
%
% SER=MMSE_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB) computes signal
% error rate for a system whose sended message (x_all), noise vector
% (n_all) and Channel matrices (H_all) are unknown for the user.
%
% SER=MMSE_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
% computes signal error rate for a system whose sended message (x_all),
% noise vector (n_all) and Channel matrices (H_all) are given by the user.
% This option is useful when we want to compare various algorithms for the
% same system (x_all, n_all and H_all)
%
% Limitations: This code is only for real systems; i.e, L-PAM
% constalations. If we want to compute L-QAM constalation, c_flag must be
```

```
% set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM
% constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM
%
% Example: Compute SER of BPSK constalation of a 12x12 system, using a
% frame length of 1000, 100 realizations of the channel matrix and MMSE-SIC
% algorithm.
% SER=MMSE_SIC(2,0,100,1000,12,12,6:2:50);
%
% Copyright (c) 2014 Irene Santos

% Checking if x_all,n_all and H_all are given
if nargin<8
    if c_flag==1
        N=N*2;
        M=M*2;
        L=2^(log2(L)/2);
        H_all=zeros(M,N,n_iterations);
        H_all(:,1:N/2,:)=randn(M,N/2,n_iterations);
        H_all(1:M/2,N/2+1:end,:)=-H_all(M/2+1:end,1:N/2,:);
        H_all(M/2+1:end,N/2+1:end,:)=H_all(1:M/2,1:N/2,:);
    else
        H_all=randn(M,N,n_iterations);
    end
    x_all=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
    n_all=randn(M,n_symbols,n_iterations);
end

% Alphabet
A=-(L-1):2:(L-1);

energy= (L*L-1)/3; % PAM energy

n_errors=zeros(length(SNRdB),1); % number of errors get for each SNR

for loop1=1:n_iterations

    H=H_all(:,:,loop1); % normal random matrix with zero mean and unit
variance

    for ind_db=1:length(SNRdB)
        sigma=sqrt(N*energy*10^(-SNRdB(ind_db)/10));
        err=0;

        parfor loop2=1:n_symbols
            x=x_all(:,loop2,loop1);
            % apply noisy linear
            yi=H*x+sigma*n_all(:,loop2,loop1);

            x_decod=zeros(N,1);

            % SIC (Successive Interference Cancelation)
            Gi=inv(H'*H+(sigma^2/energy)*eye(N))*H';
            Hi=H;
            index=[];
            for i=1:N
                Gi_aux=Gi;
                Gi_aux(index,:)=NaN*Gi_aux(index,:); % We change '0s' of Gi
as 'NaN' (to not consider in calculation of minimal values)
                [norm_min,ki]=min(sum(Gi_aux.^2,2));
                index=[index ki];

                wki=Gi(ki,:)';
```

```
                yki=wki'*yi;
                [min_s,ind]=min(abs(yki-A),[],2);
                ski=A(ind);
                x_decod(ki)=ski;
                % Update yi and Gi for next iteration
                yi=yi-ski*Hi(:,ki);
                Hi(:,ki)=0*Hi(:,ki); % We write as '0' the ki-column of H
                %Hi=Hi(:,[1:ki-1,ki+1:end]); % Delete ki-column of H
                Gi=inv(Hi'*Hi+(sigma^2/energy)*eye(N))*Hi';
            end

            err=err+sum(x~=x_decod);
        end
        n_errors(ind_db)=err;
    end
end

SER=n_errors/(N*n_iterations*n_symbols);
```

C.6 Función GTA

```
function SER=GTA(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)

% Function: GTA
% Author:   Irene Santos Velázquez, irenesantos@us.es
% Created:  01/07/2014
%
% Description: This function returns a vector of error probability (SER)
% (where each element correspond to each SNRdB element given as parameter:
% SER(i)--> SNRdB(i)) using Gaussian Tree Aproximation (GTA) algorithm.
%
% Obligatory parameters: constalation size (L>0), complex flag to indicate
% if the symbols of constalation are complex (c_flag={0,1}), number of
% different channels H (n_iterations>0), number of transmitted vector for
% each channel H (n_symbols>0), number of transmitt antennas (N>0), number
% of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).
%
% Optional parameters: N x n_symbols x n_iterations matrix of sended
% messages in a PAM constalation (x_all), M x n_symbols x n_iterations
% matrix of noise generated (n_all) and MxNxn_iterations Channel matrices
% (n_iterations matrices, whose size is MxN by each one) (H_all).
%
% SER=GTA(L,c_flag,n_iterations,n_symbols,N,M,SNRdB) computes signal error
% rate for a system whose sended message (x_all), noise vector (n_all) and
% Channel matrices (H_all) are unknown for the user.
%
% SER=GTA(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
% computes signal error rate for a system whose sended message (x_all),
% noise vector (n_all) and Channel matrices (H_all) are given by the user.
% This option is useful when we want to compare various algorithms for the
% same system (x_all, n_all and H_all)
%
% Limitations: This code is only for real systems; i.e, L-PAM
% constalations. If we want to compute L-QAM constalation, c_flag must be
% set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM
% constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM
%
% Example: Compute SER of BPSK constalation of a 12x12 system, using a
% frame length of 1000, 100 realizations of the channel matrix and GTA
% algorithm.
% SER=GTA(2,0,100,1000,12,12,6:2:50);
```

```
%
% Copyright (c) 2014 Irene Santos

% Checking if x_all,n_all and H_all are given
if nargin<8
    if c_flag==1
        N=N*2;
        M=M*2;
        L=2^(log2(L)/2);
        H_all=zeros(M,N,n_iterations);
        H_all(:,1:N/2,:)=randn(M,N/2,n_iterations);
        H_all(1:M/2,N/2+1:end,:)=-H_all(M/2+1:end,1:N/2,:);
        H_all(M/2+1:end,N/2+1:end,:)=H_all(1:M/2,1:N/2,:);
    else
        H_all=randn(M,N,n_iterations);
    end
    x_all=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
    n_all=randn(M,n_symbols,n_iterations);
end

% Alphabet
A=-(L-1):2:(L-1);

energy= (L*L-1)/3; % PAM energy
%energy=1; % example

n_errors=zeros(length(SNRdB),1); % number of errors get for each SNR

for loop1=1:n_iterations

    H=H_all(:,:,loop1); % normal random matrix with zero mean and unit
variance

    for ind_db=1:length(SNRdB)
        sigma=sqrt(N*energy*10^(-SNRdB(ind_db)/10));
        err=0;

        % ////////////////////////
        % /////// Tree BP ////////
        % ////////////////////////

        % Compute weights of total graph
        C=sigma^2*inv(H'*H+(sigma^2/energy)*eye(N)); % Covariance matrix
        %C=[1 .3 .5 .4;.3 1 .8 .6; .5 .8 1 .7; .4 .6 .7 1];%example
        w=C.^2./(diag(C)*diag(C)');

        % As we look for maximum weights, rewrite diagonals of w as -1
        % (we are not interested in weights of nodes with theirselves)

        w_aux=w-2*diag(diag(w));

        % Compute maximum spanning

        parent=zeros(N,1); % Keep father of each node, i.e. parent(i)
        % is the parent node of i

        % Assume that root node is node 1
        parent(1)=0;
        inTree=1; % Keep nodes added to the tree
        Tree=zeros(N); % Edges of tree are keep as '1'
```

```
        for k=1:N-1
           % We look for maximal weights of nodes inside the tree
           [max_ws,inds]=max(w_aux(inTree,:),[],2);
           % We only save the maximum weigth of previous weights
           [max_w,ind]=max(max_ws);
           % Write as -1 the edges weigth inside the tree (to not consider
           % them in future iterations)
           new_node=inds(ind);
           w_aux(inTree,new_node)=-1;
           w_aux(new_node,inTree)=-1;
           % Add new node to the tree
           parent(new_node)=inTree(ind);
           inTree=[inTree new_node];
           Tree(new_node,parent(new_node))=1;
           Tree(parent(new_node),new_node)=1;
        end

        parfor loop2=1:n_symbols
            x=x_all(:,loop2,loop1);
            % apply noisy linear
            y=H*x+sigma*n_all(:,loop2,loop1);
            %y=H*x; % example

            % MMSE estimator
            %z=inv(H'*H+(sigma^2/energy)*eye(N))*H'*y;
            z=sigma^(-2)*C*H'*y; % This way avoids to do an inverse again

            % f(x1; z, C)
            f_x1=@(x1)(exp(-0.5*((x1-z(1)).^2)/C(1,1)));

            % f(xi|xj; z, C) (where xj=x_p(i), i.e, the parent node of i)
            f_xi_xj=@(xi,zi,xj,zj,Cij,Cii,Cjj)(exp(-0.5*(((xi-zi)-
(Cij/Cjj)*(xj-zj)).^2)/(Cii-(Cij^2/Cjj))));

            % Look for the tree leaves and its edges
            n_edges=sum(Tree,1)'; % Number of edges of each node
            n_children=[n_edges(1);n_edges(2:N)-1]; % Number of chilgren of
each node
            leaf_node=n_children==0; % Leaves are kept as '1'

            % ////////////////////
            % /////// BP /////////
            % ////////////////////

            % ////////////////////////////////////////////
            % ///// DOWNWARD BP MESSAGES (m_i-->p(i)) //////
            % ////////////////////////////////////////////

            messages_down=ones(N,L); % Each column is associated with an
alphabet symbol
            % Initialized as '1' to merge (25) and (26). This means we
            % consider messages from leaves to children as '1'

            i_nodes=find(leaf_node==1); % We start from leaves

            n_messages_to_father=zeros(N,1); % Keep the number os messages
sended to each parent node

            while (isempty(i_nodes)==0)
                i_nodes_sig=[];
                for k=1:length(i_nodes)
                    i=i_nodes(k);
```

```
                    pi=parent(i);
                    xi=A;
                    children_i=find(parent==i);
                    messages_children=prod(messages_down(children_i,:),1); %
multiply messages from children to parent node i

                    for kk=1:L
                        xpi=A(kk);

messages_down(i,kk)=sum(f_xi_xj(xi,z(i),xpi,z(pi),C(i,pi),C(i,i),C(pi,pi)).*m
essages_children);
                    end

                    n_messages_to_father(pi)=n_messages_to_father(pi)+1;

                    if (n_edges(pi)-1==n_messages_to_father(pi) && pi~=1) %
If parent pi is the root node, it can not added as next node because downward
messages end in this node
                        i_nodes_sig=[i_nodes_sig; pi]; % pi is added as next
node because it has already received all messages from its children and it
can send a message to its parent
                    end
                end
                i_nodes=i_nodes_sig;
            end


            % ////////////////////////////////////////////
            % ///// UPWARD BP MESSAGES (m_p(i)-->i) //////
            % ////////////////////////////////////////////

            messages_up=ones(N,L); % Each column is associated with an
alphabet symbol
            % Initialized as '1' to merge (27) and (28). This means we
            % consider messages from parent of root node to root node as '1'

            i_nodes=find(parent==1); % We start from root node to its
children

            while(isempty(i_nodes)==0)
                i_nodes_sig=[];
                for k=1:length(i_nodes)
                    i=i_nodes(k);
                    pi=parent(i);
                    xpi=A;

                    children_pi=find(parent==pi);
                    ind_children=find(children_pi~=i);
                    children_pi=children_pi(ind_children);% We remove child
of pi, i.e, i, because we are doing it now
                    messages_children=prod(messages_down(children_pi,:),1); %
multiply messages from children of node p(i) to p(i) (less i)

                    ppi=parent(pi); % Parent of parent node of i p(p(i))
                    message_parent=messages_up(pi,:);

                    for kk=1:L
                        xi=A(kk);

messages_up(i,kk)=sum(f_xi_xj(xi,z(i),xpi,z(pi),C(i,pi),C(i,i),C(pi,pi)).*mes
sages_children.*message_parent);
                    end
```

```
                    children_i=find(parent==i);
                    i_nodes_sig=[i_nodes_sig; children_i];
                end
                i_nodes=i_nodes_sig;
            end



            % ////////////////////////////
            % ///// BELIEF VARIABLE //////
            % ////////////////////////////

            % After message passing, we compute the belief at each variable

            belief_x=zeros(N,L);
            for i=1:N
                xi=A;
                children_i=find(parent==i);
                if i==1

belief_x(i,:)=f_x1(xi).*prod(messages_down(children_i,:),1);
                else

belief_x(i,:)=messages_up(i,:).*prod(messages_down(children_i,:),1);
                end
            end


            % ////////////////////
            % ///// DECODER //////
            % ////////////////////

            % Decoding

            [max_belief,ind]=max(belief_x,[],2);
            x_decod=A(ind)';

            err=err+sum(x~=x_decod);
        end
        n_errors(ind_db)=err;
    end
end

SER=n_errors/(N*n_iterations*n_symbols);
```

C.7 Función GTA_SIC

```
function
SER=GTA_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)

% Function: GTA_SIC
% Author:   Irene Santos Velázquez, irenesantos@us.es
% Created:  01/08/2014
%
% Description: This function returns a vector of error probability (SER)
% (where each element correspond to each SNRdB element given as parameter:
% SER(i)--> SNRdB(i)) using Gaussian Tree Aproximation with Successive
% Interference Cancelation (GTA-SIC) algorithm.
```

```
%
% Obligatory parameters: constalation size (L>0), complex flag to indicate
% if the symbols of constalation are complex (c_flag={0,1}), number of
% different channels H (n_iterations>0), number of transmitted vector for
% each channel H (n_symbols>0), number of transmitt antennas (N>0), number
% of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB).
%
% Optional parameters: N x n_symbols x n_iterations matrix of sended
% messages in a PAM constalation (x_all), M x n_symbols x n_iterations
% matrix of noise generated (n_all) and MxNxn_iterations Channel matrices
% (n_iterations matrices, whose size is MxN by each one) (H_all).
%
% SER=GTA_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB) computes signal
% error rate for a system whose sended message (x_all), noise vector
% (n_all) and Channel matrices (H_all) are unknown for the user.
%
% SER=GTA_SIC(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,x_all,n_all,H_all)
% computes signal error rate for a system whose sended message (x_all),
% noise vector (n_all) and Channel matrices (H_all) are given by the user.
% This option is useful when we want to compare various algorithms for the
% same system (x_all, n_all and H_all)
%
% Limitations: This code is only for real systems; i.e, L-PAM
% constalations. If we want to compute L-QAM constalation, c_flag must be
% set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM
% constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM
%
% Example: Compute SER of BPSK constalation of a 12x12 system, using a
% frame length of 1000, 100 realizations of the channel matrix and GTA-SIC
% algorithm.
% SER=GTA_SIC(2,0,100,1000,12,12,6:2:50);
%
% Copyright (c) 2014 Irene Santos

% Checking if x_all,n_all and H_all are given
if nargin<8
    if c_flag==1
        N=N*2;
        M=M*2;
        L=2^(log2(L)/2);
        H_all=zeros(M,N,n_iterations);
        H_all(:,1:N/2,:)=randn(M,N/2,n_iterations);
        H_all(1:M/2,N/2+1:end,:)=-H_all(M/2+1:end,1:N/2,:);
        H_all(M/2+1:end,N/2+1:end,:)=H_all(1:M/2,1:N/2,:);
    else
        H_all=randn(M,N,n_iterations);
    end
    x_all=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
    n_all=randn(M,n_symbols,n_iterations);
end

% Alphabet
A=-(L-1):2:(L-1);

energy= (L*L-1)/3; % PAM energy
%energy=1; % example

n_errors=zeros(length(SNRdB),1); % number of errors get for each SNR

for loop1=1:n_iterations
    for ind_db=1:length(SNRdB)
        sigma=sqrt(N*energy*10^(-SNRdB(ind_db)/10));
        %sigma=1e-10; % example
```

```
        err=0;

        for loop2=1:n_symbols

            H=H_all(:,:,loop1); % normal random matrix with zero mean and
unit variance

            x=x_all(:,loop2,loop1);
            % apply noisy linear
            y=H*x+sigma*n_all(:,loop2,loop1);
            %y=H*x; % example

            % SIC (Successive Interference Cancelation)
            x_decod=zeros(N,1);
            original_ind=(1:N)'; % Keep original order of variables

            for k_sic=1:N

                % ////////////////////////
                % /////// Tree BP ////////
                % ////////////////////////

                % Compute weights of total graph
                C=sigma^2*inv(H'*H+(sigma^2/energy)*eye(N-k_sic+1)); %
Covariance matrix
                %C=[1 .3 .5 .4;.3 1 .8 .6; .5 .8 1 .7; .4 .6 .7 1];%example

                % MMSE estimator
                %z=inv(H'*H+(sigma^2/energy)*eye(N-k_sic+1))*H'*y;
                z=sigma^(-2)*C*H'*y; % This way avoids to do an inverse again

                if k_sic==N % Last iteration is special because there is only
one node (tree is not necessary)
                    xi=A;
                    belief_x=f_xi(xi,z(1),C(1,1));
                    % Decoder
                    [max_belief,ind]=max(belief_x,[],2);
                    x_decod(original_ind(1))=A(ind);
                    break
                end

                % We look for the smallest diagonal element of C. Root node
                % will be the index of this element
                [minC,root]=min(diag(C));

                w=C.^2./(diag(C)*diag(C)');

                % As we look for maximum weights, rewrite diagonals of w as -
1
                % (we are not interested in weights of nodes with
theirselves)

                w_aux=w-2*diag(diag(w));

                % Compute maximum spanning

                parent=zeros(N-k_sic+1,1); % Keep father of each node,
                % i.e. parent(i) is the parent node of i

                % Assume that root node is node root
                parent(root)=0;
```

```
                inTree=root; % Keep nodes added to the tree
                Tree=zeros(N-k_sic+1); % Edges of tree are keep as '1'

                for k=1:N-k_sic
                    % We look for maximal weights of nodes inside the tree
                    [max_ws,inds]=max(w_aux(inTree,:),[],2);
                    % We only save the maximum weigth of previous weights
                    [max_w,ind]=max(max_ws);
                    % Write as -1 the edges weigth inside the tree (to not
consider
                    % them in future iterations)
                    new_node=inds(ind);
                    w_aux(inTree,new_node)=-1;
                    w_aux(new_node,inTree)=-1;
                    % Add new node to the tree
                    parent(new_node)=inTree(ind);
                    inTree=[inTree new_node];
                    Tree(new_node,parent(new_node))=1;
                    Tree(parent(new_node),new_node)=1;
                end



                % f(xi; z, C)
                f_xi=@(xi,zi,Cii)(exp(-0.5*((xi-zi).^2)/Cii));

                % f(xi|xj; z, C) (where xj=x_p(i), i.e, the parent node of
                % i)
                f_xi_xj=@(xi,zi,xj,zj,Cij,Cii,Cjj)(exp(-0.5*(((xi-zi)-
(Cij/Cjj)*(xj-zj)).^2)/(Cii-(Cij^2/Cjj))));

                % Look for the tree leaves and its edges
                n_edges=sum(Tree,1)'; % Number of edges of each node
                n_children=n_edges-1; % Number of chilgren of each node
                n_children(root)=n_children(root)+1;
                leaf_node=n_children==0; % Leaves are kept as '1'

                % ////////////////////
                % /////// BP /////////
                % ////////////////////

                % ////////////////////////////////////////////
                % ///// DOWNWARD BP MESSAGES (m_i-->p(i)) //////
                % ////////////////////////////////////////////

                messages_down=ones(N-k_sic+1,L); % Each column is associated
with an alphabet symbol
                % Initialized as '1' to merge (25) and (26). This means we
                % consider messages from leaves to children as '1'

                i_nodes=find(leaf_node==1); % We start from leaves

                n_messages_to_father=zeros(N-k_sic+1,1); % Keep the number os
messages sended to each parent node

                while (isempty(i_nodes)==0)
                    i_nodes_sig=[];
                    for k=1:length(i_nodes)
                        i=i_nodes(k);
                        pi=parent(i);
                        xi=A;
                        children_i=find(parent==i);
```

```
messages_children=prod(messages_down(children_i,:),1); % multiply messages
from children to parent node i

                        for kk=1:L
                            xpi=A(kk);

messages_down(i,kk)=sum(f_xi_xj(xi,z(i),xpi,z(pi),C(i,pi),C(i,i),C(pi,pi)).*m
essages_children);
                        end

                        n_messages_to_father(pi)=n_messages_to_father(pi)+1;

                        if (n_edges(pi)-1==n_messages_to_father(pi) &&
pi~=root) % If parent pi is the root node, it can not added as next node
because downward messages end in this node
                            i_nodes_sig=[i_nodes_sig; pi]; % pi is added as
next node because it has already received all messages from its children and
it can send a message to its parent
                        end
                    end
                    i_nodes=i_nodes_sig;
                end


                % ////////////////////////////
                % ///// BELIEF VARIABLE //////
                % ////////////////////////////

                % After message passing, we compute the belief root
                % variable

                xi=A;
                children_i=find(parent==root);

belief_x=f_xi(xi,z(root),C(root,root)).*prod(messages_down(children_i,:),1);


                % ////////////////////
                % ///// DECODER //////
                % ////////////////////

                % Decoding

                [max_belief,ind]=max(belief_x,[],2);
                x_decod(original_ind(root))=A(ind);

                % Interference Cancelation
                y=y-H(:,root)*x_decod(original_ind(root));
                % We copy last column of H to root column and delete the
                % last column
                H(:,root)=H(:,end);
                H=H(:,1:N-k_sic);

                original_ind(root)=original_ind(N-k_sic+1);
            end
            err=err+sum(x~=x_decod);
        end
        n_errors(ind_db)=err;
    end
end
```

```
SER=n_errors/(N*n_iterations*n_symbols);
```

C.8 Función EP

```
function
SER=EP(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP,x_all,n_all,H_all)

% Function: EP
% Author:   Irene Santos Velázquez, irenesantos@us.es
% Created:  02/10/2014
%
% Description: This function returns a vector of error probability (SER)
% (where each element correspond to each SNRdB element given as parameter:
% SER(i)--> SNRdB(i)) using Expectation Propagation (EP) algorithm.
%
% Obligatory parameters: constalation size (L>0), complex flag to indicate
% if the symbols of constalation are complex (c_flag={0,1}), number of
% different channels H (n_iterations>0), number of transmitted vector for
% each channel H (n_symbols>0), number of transmitt antennas (N>0), number
% of received antennas (M>0), row-vector of Signal to Noise Ratios (SNRdB),
% number of EP iterations (L_EP>0).
%
% Optional parameters: N x n_symbols x n_iterations matrix of sended
% messages in a PAM constalation (x_all), M x n_symbols x n_iterations
% matrix of noise generated (n_all) and MxNxn_iterations Channel matrices
% (n_iterations matrices, whose size is MxN by each one) (H_all).
%
% SER=EP(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP) computes signal
% error rate for a system whose sended message (x_all), noise vector
% (n_all) and Channel matrices (H_all) are unknown for the user.
%
% SER=EP(L,c_flag,n_iterations,n_symbols,N,M,SNRdB,L_EP,x_all,n_all,H_all)
% computes signal error rate for a system whose sended message (x_all),
% noise vector (n_all) and Channel matrices (H_all) are given by the user.
% This option is useful when we want to compare various algorithms for the
% same system (x_all, n_all and H_all)
%
% Limitations: This code is only for real systems; i.e, L-PAM
% constalations. If we want to compute L-QAM constalation, c_flag must be
% set as '1', NxM system multiply by two its size --> 2Nx2M system and QAM
% constalation converts to PAM constalation --> 2^(log2(L)/2))-PAM
%
% Example: Compute SER of BPSK constalation of a 12x12 system, using a
% frame length of 10, 100 realizations of the channel matrix and 10
% iterations for EP algorithm.
% SER=EP(2,0,100,10,12,12,6:2:50,10);
%
% Copyright (c) 2014 Irene Santos

% Checking if x_all,n_all and H_all are given
if nargin<8
    if c_flag==1
        N=N*2;
        M=M*2;
        L=2^(log2(L)/2);
        H_all=zeros(M,N,n_iterations);
        H_all(:,1:N/2,:)=randn(M,N/2,n_iterations);
        H_all(1:M/2,N/2+1:end,:)=-H_all(M/2+1:end,1:N/2,:);
        H_all(M/2+1:end,N/2+1:end,:)=H_all(1:M/2,1:N/2,:);
    else
        H_all=randn(M,N,n_iterations);
```

```
    end
    x_all=floor(rand(N,n_symbols,n_iterations)*L)*2-(L-1);
    n_all=randn(M,n_symbols,n_iterations);
end

% Alphabet
A=-(L-1):2:(L-1);

energy= (L*L-1)/3; % PAM energy

n_errors=zeros(length(SNRdB),1); % number of errors get for each SNR

for loop1=1:n_iterations

    H=H_all(:,:,loop1); % normal random matrix with zero mean and unit
variance

    for ind_db=1:length(SNRdB)
        sigma=sqrt(N*energy*10^(-SNRdB(ind_db)/10));
        err=0;

        parfor loop2=1:n_symbols
            x=x_all(:,loop2,loop1);
            % apply noisy linear
            y=H*x+sigma*n_all(:,loop2,loop1);

            % Initialization
            gamma=zeros(N,1);
            GAMMA=ones(N,1)/energy;
            cavity_i=zeros(N,L);

            % Constants
            beta=0.2;
            epsilon=5e-7;

            % meand and variance vectors of all marginal qi(x) at iteration
            % l=0
            C_q=inv(sigma^(-2)*H'*H+diag(GAMMA));
            var_qi=diag(C_q);
            nu_qi=C_q*(sigma^(-2)*H'*y+gamma);

            for l=1:L_EP

                % Mean (ti) and variance (hi_2) of each cavity marginal
                hi_2=var_qi./(1-var_qi.*GAMMA);
                ti=hi_2.*(nu_qi./var_qi-gamma);

                % Compute cavity marginal for each xi
                % Note: Both 'for' do the same thing, but the first one by
                % columns and the other by rows. Depend on the number of
                % columns (N) or rows (L), we will interest in one or
                % another for.
                if L<=N
                    for k=1:L
                        cavity_i(:,k)=(1./sqrt(2*pi*hi_2)).*exp(-0.5*(A(k)-
ti).^2./hi_2);
                    end
                else
                    for k=1:N
                        cavity_i(k,:)=(1/sqrt(2*pi*hi_2(k)))*exp(-0.5*(A-
ti(k)).^2/hi_2(k));
```

```
                    end
                end

                % Normalize distribution (33)
                K=sum(cavity_i,2);
                cavity_i=diag(1./K)*cavity_i;

                % Mean (nu_pi) and variance (var_pi) of distribution (33)
                nu_pi=sum((diag(A)*cavity_i')',2);
                var_pi=sum(((diag(A)*ones(L,N))'-
diag(nu_pi)*ones(N,L)).^2.*cavity_i,2);
                var_pi=max([var_pi,epsilon*ones(N,1)],[],2);

                % Compute new values for gamma and GAMMA
                GAMMA_aux=beta*((1./var_pi)-(1./hi_2))+(1-beta)*GAMMA;
                gamma_aux=beta*((nu_pi./var_pi)-(ti./hi_2))+(1-beta)*gamma;

                for k=1:N
                    if GAMMA_aux(k)>=0
                        GAMMA(k)=GAMMA_aux(k);
                        gamma(k)=gamma_aux(k);
                    end
                end

                % meand and variance vectors of all marginal qi(x) at next
                % iteration l
                C_q=inv(sigma^(-2)*H'*H+diag(GAMMA));
                var_qi=diag(C_q);
                nu_qi=C_q*(sigma^(-2)*H'*y+gamma);
            end

            % Decoder EP
            belief_x=zeros(N,L);
            for i=1:L
                belief_x(:,i)=abs(nu_qi-A(i));
            end

            [min_belief,ind]=min(belief_x,[],2);
            x_decod=A(ind)';

            err=err+sum(x~=x_decod);
        end
        n_errors(ind_db)=err;
    end
end

SER=n_errors/(N*n_iterations*n_symbols);
```

ANEXO D: EXPLICACIÓN DEL CÓDIGO

D.1 Función ML

Como se vio en la ecuación (1-3), la solución de máxima verosimilitud consiste es buscar el vector de entrada $\hat{\mathbf{x}}$ que hace mínima la norma $\|\mathbf{y} - \mathbf{H}\mathbf{x}\|^2$, lo que supone recorrer todos vectores $\hat{\mathbf{x}}$ con todas las $|\mathcal{A}|^N$ posibles combinaciones de símbolos. Para ello se ha seguido la regla mostrada en la Figura D-1. Según el número de iteración en la que se encuentre, cambiará el elemento de una determinada posición. En concreto, el último elemento del vector $\mathbf{x}$ que se busca cambia en cada iteración, el penúltimo elemento cambia cada L iteraciones, el antepenúltimo elemento cada L^2 y así hasta llegar al primer elemento. Siguiendo este método se obtienen todos los posibles vectores $\mathbf{x}$ con todas las combinaciones de símbolos, es decir, con L^N combinaciones siendo L el tamaño de la constelación y N el número de antenas transmisoras.

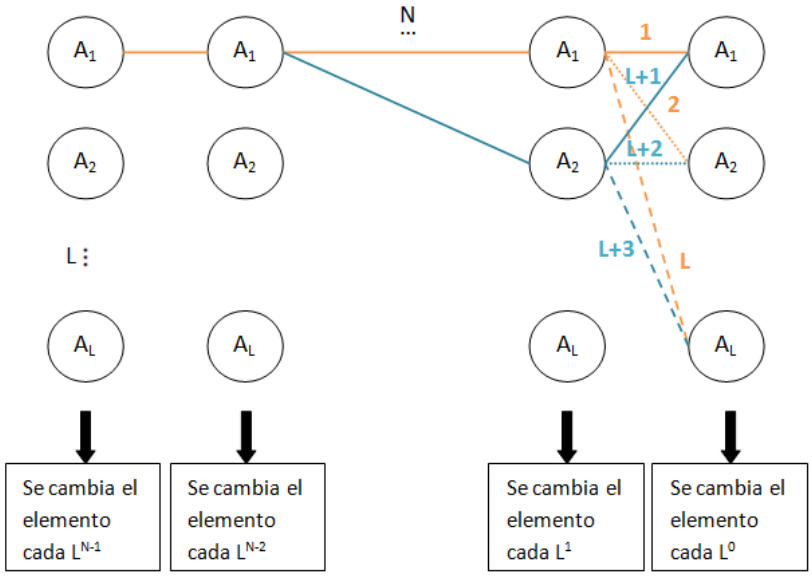

Figura D-1. Regla para realizar todas las combinaciones de símbolos.

Con cada uno de los vectores $\mathbf{x}$ que se obtienen se calcula la norma de la ecuación (1-3) y el vector $\mathbf{x}$ de menor norma será vector decodificado buscado $\hat{\mathbf{x}}$.

D.2 Función ZF_SIC

Para cada matriz de canal, vector transmitido por dicho canal y varianza de ruido dada se aplica la recursividad vista en el Apartado 1.2.4 y que se resume a continuación.

$$\mathbf{y_1} \coloneqq \mathbf{y}$$
$$\mathbf{G_1} = \mathbf{H}^{\dagger}$$

$$
\begin{aligned}
&\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ N\\
&\quad k_i = \arg\min_{\mathrm{j}\notin(\mathrm{k}_1,\ldots,\mathrm{k}_{\mathrm{i}-1})} \left\|\langle \mathbf{G_i}\rangle_{\mathrm{j}}\right\|^2\\
&\quad \boldsymbol{\omega}_{\mathbf{k_i}}^{\mathrm{T}} = \langle \mathbf{G_i}\rangle_{\mathrm{k_i}}\\
&\quad \hat{x}_{k_i} = \arg\min_{a\in\mathcal{A}} \left|\boldsymbol{\omega}_{\mathbf{k_i}}^{\mathrm{T}}\mathbf{y_i} - a\right|\\
&\quad \mathbf{y_{i+1}} = \mathbf{y_i} - \mathbf{H}_{\mathrm{k_i}}\hat{x}_{k_i}\\
&\quad \mathbf{G_{i+1}} = \mathbf{H}_{\overline{\mathbf{k_i}}}^{\dagger}\\
&\boldsymbol{end}
\end{aligned}
$$

La función que en matlab calcula el mínimo de un vector o matriz se denomina min. Como al final de cada iteración se pone a cero la columna k_i de $\mathbf{H}^{\dagger}$, es decir, $\mathbf{G_{i+1}}$ es una matriz con filas nulas, no se puede aplicar directamente la función min de matlab porque siempre devolvería el índice de algunas de las filas nulas. Por este motivo, se utiliza una matriz **G** auxiliar en el código que sustituye los ceros de las filas por NaN (valor que ignora la función min de matlab).

D.3 Función MMSE_SIC

Para cada matriz de canal, vector transmitido por dicho canal y varianza de ruido dada se aplica la recursividad vista en el Apartado 1.2.4 y que se resume a continuación.

$$
\begin{aligned}
&\mathbf{y_1} \coloneqq \mathbf{y}\\
&\mathbf{G_1} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\\
&\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ N\\
&\quad k_i = \arg\min_{\mathrm{j}\notin(\mathrm{k}_1,\ldots,\mathrm{k}_{\mathrm{i}-1})} \left\|\langle \mathbf{G_i}\rangle_{\mathrm{j}}\right\|^2\\
&\quad \boldsymbol{\omega}_{\mathbf{k_i}}^{\mathrm{T}} = \langle \mathbf{G_i}\rangle_{\mathrm{k_i}}\\
&\quad \hat{x}_{k_i} = \arg\min_{a\in\mathcal{A}} \left|\boldsymbol{\omega}_{\mathbf{k_i}}^{\mathrm{T}}\mathbf{y_i} - a\right|\\
&\quad \mathbf{y_{i+1}} = \mathbf{y_i} - \mathbf{H}_{\mathrm{k_i}}\hat{x}_{k_i}\\
&\quad \mathbf{G_{i+1}} = \left(\mathbf{H}_{\overline{\mathbf{k_i}}}^{\mathrm{T}}\mathbf{H}_{\overline{\mathbf{k_i}}} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}_{\overline{\mathbf{k_i}}}^{\mathrm{T}}\\
&\boldsymbol{end}
\end{aligned}
$$

La función que en matlab calcula el mínimo de un vector o matriz se denomina min. Como al final de cada iteración se pone a cero la columna k_i de $\mathbf{H}^{\dagger}$, es decir, $\mathbf{G_{i+1}}$ es una matriz con filas nulas, no se puede aplicar directamente la función min de matlab porque siempre devolvería el índice de algunas de las filas nulas. Por este motivo, se utiliza una matriz **G** auxiliar en el código que sustituye los ceros de las filas por NaN (valor que ignora la función min de matlab).

D.4 Función GTA

Lo primero que hace es calcular el árbol de expansión máximo para una matriz de canal y SNR concretas. Los pasos a seguir son los mostrados en el ejemplo del Apartado 4.3.2, que se detallan a continuación.

Primero se calcula la matriz de covarianzas (**C**) y la matriz de pesos que está formada por los coeficientes de correlación (**w**). Se necesitará una matriz de pesos auxiliar $\mathbf{w_{aux}}$ (inicializada a **w**) y en la que se irán sustituyendo por -1 los pesos de las ramas entre los nodos que se vayan añadiendo al árbol. De esta manera, se evitará volver a tener en cuenta esos nodos en iteraciones posteriores para la obtención del árbol. Como no tienen sentido ramas de un nodo a sí mismo, los pesos de la diagonal de $\mathbf{w_{aux}}$ también se sustituyen por -1.

En resumen, la matriz auxiliar de pesos se inicializará con los siguientes valores:

$$\mathbf{w_{aux}} = \begin{bmatrix} -1 & w_{12} & \cdots & w_{1N} \\ w_{21} & -1 & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \cdots & -1 \end{bmatrix}$$

Además de la matriz anterior, se van a utilizar las variables mostradas en la Tabla D–1 para el cálculo del árbol.

Tabla D–1. Variables necesarias para el cálculo del árbol en Matlab.

Variables	Breve descripción	Inicialización
$\mathbf{w_{aux}}$	Matriz auxiliar de pesos que tiene a -1 los pesos de las ramas de los nodos ya agregados al árbol.	$\mathbf{w_{aux}} = \mathbf{w}$ $w_{aux}(i,i) = -1$
parent	Vector que en cada elemento i contiene al padre del elemento i (es decir, p(i)). Vale 0 en el elemento raíz.	$parent(1) = 0$
inTree	Vector que en cada iteración añade el nodo agregado al árbol.	$inTree = 1$
Tree	Matriz cuyos elementos a 1 simbolizan las ramas agregadas al árbol y a 0 las ramas inexistentes en el árbol.	$\mathbf{Tree} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$

Como se muestra en la tabla anterior, el vector **parent** se inicializa indicando el nodo raíz, es decir, poniendo a 0 el nodo 1. Una vez elegido el nodo raíz, hay que añadirlo al árbol y por eso se añade el elemento '1' al vector **inTree**. Como el nodo '1' aún no está conectado a ningún otro nodo, la matriz **Tree** deberá ser la matriz nula (ya que no existen todavía ramas en el árbol).

Una vez inicializada, se comienza el proceso recursivo para el cálculo del árbol. En cada iteración, se busca la rama de mayor peso de los nodos que están en el árbol, es decir, se busca el elemento mayor de entre todas las filas de la matriz $\mathbf{w_{aux}}$ correspondientes a nodos que están ya en el árbol (es decir, a los nodos que están en el vector **inTree**). Ese elemento, denotado por $w_{aux}(i,j)$, indica el peso de la nueva rama que será añadida al árbol y que conecta los nodos i (nodo padre) y j (nodo hijo del padre i). Como j es un nodo nuevo del árbol, se ponen a -1 todos los pesos de la matriz $\mathbf{w_{aux}}$ que conecta este nuevo nodo j con todos los nodos que se tenían antes en el árbol, es decir, en el vector **inTree** (que aún no está actualizado con el nuevo nodo j). Como i es el padre de j es necesario añadir al vector **parent** esta información, es decir, se impone $parent(j) = i$. Antes de acabar la iteración, hay que añadir el nuevo nodo j al vector **inTree** y se ponen a 1 los dos elementos que simbolizan las ramas entre i y j en la matriz **Tree**, es decir, $Tree(i,j) = 1$ y $Tree(j,i) = 1$.

Este procedimiento se repite hasta que los *N* nodos se hayan incluido en el árbol.

Una vez que se tiene el árbol de expansión máximo para una matriz de canal y SNR concreta, se aplica el algoritmo de paso de mensajes BP para cada uno de los vectores que se transmite por canal.

El vector recibido se obtiene mediante

$$\mathbf{y} = \mathbf{Hx} + \mathbf{v} = \mathbf{Hx} + \sigma\mathbf{n} \tag{C-1}$$

En este caso, el estimador utilizado es el de MMSE, es decir, se calcula (1-6). A continuación se definen las funciones vistas en el Apartado 4.2, que se recuerdan a continuación.

$$f(x_i; \mathbf{z}, \mathbf{C}) = \exp\left(-\frac{1}{2}\frac{(x_i - z_i)^2}{C_{ii}}\right)$$

$$f(x_i | x_j; \mathbf{z}, \mathbf{C}) = \exp\left(-\frac{1}{2}\frac{\left((x_i - z_i) - \frac{C_{ij}}{C_{jj}}(x_j - z_j)\right)^2}{C_{ii} - \frac{C_{ij}^2}{C_{jj}}}\right)$$

Como puede observarse, la primera función depende de las variables x_i, z_i y C_{ii}, mientras que la segunda depende de x_i, z_i, x_j, z_j, C_{ij}, C_{ii} y C_{jj}. En realidad, la primera función sólo se utiliza en el cálculo de la

variable predicha (y no en el cálculo de los mensajes como la segunda) y únicamente para el nodo raíz, por lo que tomando como nodo raíz el nodo 1, sólo se necesita definir la función siguiente

$$f(x_i) = exp\left(-\frac{1}{2}\frac{(x_i - z_1)^2}{C_{11}}\right)$$

Ya definidas estas funciones, es necesario calcular las constantes mostradas en la Tabla D–2, que se utilizarán durante todo el paso de mensajes.

Tabla D–2. Constantes necesarias para el algoritmo BP en Matlab.

Constantes	Breve descripción
$\mathbf{n}_{\mathbf{edges}}$	Vector que contiene el número de ramas de cada nodo.
$\mathbf{n}_{\mathbf{children}}$	Vector que contiene el número de hijos de cada nodo.
$\mathbf{leaf}_{\mathbf{node}}$	Vector que indica con un 1 los nodos que son nodos extremos (y con 0 los que no lo son).

Para calcular el número de ramas ($\mathbf{n}_{\mathbf{edges}}$) que salen de cada nodo basta con sumar (por filas) los unos de la matriz **Tree**. El número de hijos de cada nodo ($\mathbf{n}_{\mathbf{children}}$) es el número de ramas de dicho nodo menos uno (ya que una de las ramas lo conecta con su nodo padre y las demás con sus hijos), a excepción del nodo raíz al que todas sus ramas lo conectan con sus hijos y no hay que restarle uno. Los nodos extremos (**leaf_node**) se localizan fácilmente ya que son los únicos que no tienen hijos, es decir, son aquellos i en los $n_{children}\ (i)$ es cero.

Tras definir estas constantes se comienza con el paso de mensajes "hacia abajo" en el que serán necesarias las variables mostradas en la Tabla D–3.

Tabla D–3. Variables necesarias para el paso de mensajes "hacia abajo" en matlab.

Variables	Breve descripción	Inicialización
$\mathbf{messages}_{\mathbf{down}}$	Matriz (de tamaño $N\ x\ L$ siendo L el tamaño de la constelación) que contiene los mensajes "hacia abajo" para todos los posibles valores del alfabeto de $x_{p(i)}$. Simboliza a $m_{i\to p(i)}(x_{p(i)})$ de las ecuaciones (4-17)-(4-18). Se inicializa a uno para unificar las ecuaciones (4-17)-(4-18).	$\mathbf{messages}_{\mathbf{down}} = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$
n_messages_to_father	Vector que indica el número de mensajes que le han llegado a cada nodo padre.	$\mathbf{n}_{\mathbf{messages\_to\_father}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$
$\mathbf{messages}_{\mathbf{children}}$	Matriz que contiene el producto de los mensajes enviados por los nodos hijos a su padre para todos los posibles valores del alfabeto de x_i. Simboliza a $\prod_{j\,\vert\,p(j)=i} m_{j\to i}(x_i)$ de la ecuación (4-17).	-
$\mathbf{i}_{\mathbf{nodes}}$	Vector que contiene los nodos que pueden realizar el paso de mensajes en la siguiente iteración (dado que ya tienen los mensajes de todos sus hijos).	Se inicializa con los nodos extremos
$\mathbf{i}_{\mathbf{nodes\_sig}}$	Vector auxiliar que almacena el nodo padre que ya ha recibido todos los mensajes de los hijos y que puede realizar el paso de mensajes en las próximas iteraciones.	(vacío)
i	Nodo hijo que envía el mensaje en una iteración dada.	-
pi	Nodo padre del nodo *i*.	-

El paso de mensajes "hacia abajo" se comienza por los nodos extremos y por ello se inicializa $\mathbf{i}_{\mathbf{nodes}}$ con estos nodos. Se van cogiendo los nodos de $\mathbf{i}_{\mathbf{nodes}}$ uno a uno. Sea i el nodo actual y pi el padre de dicho nodo. Lo primero es buscar si el nodo i tiene hijos. Si los tiene, es necesario realizar el producto de los mensajes de los hijos de i a i y se almacena dicho producto en la variable $\mathbf{messages}_{\mathbf{children}}$. En caso de que i no tenga hijos

y gracias a la inicialización de $\mathbf{messages_{down}}$ a todo uno, la variable $\mathbf{messages_{children}}$ valdrá uno, con lo cual se están uniendo las ecuaciones (4-17)-(4-18) en una sola expresión. A continuación se multiplica esta variable $\mathbf{messages_{children}}$ con la correspondiente función $f(x_i|x_{pi};\mathbf{z},\mathbf{C})$ (habiendo sustituido en esta última los correspondientes valores de z_i, z_{pi}, C_{ipi}, C_{ii} y C_{pipi}) y se suma para todos los valores de x_i. El resultado dependería de x_{pi}, por lo que se debe hacer esto último para todos los posibles valores del alfabeto de x_{pi} y por ello se necesita el for con índice *kk* del código matlab. Una vez completado el mensaje, se suma uno al elemento *pi* del vector que cuenta los mensajes pasados al nodo padre, es decir, a $\mathbf{n_{messages\_to\_father}}$ y se comprueba si este nodo padre *pi* ha recibido los mensajes de todos sus hijos ya que, en ese caso, se añade este nodo *pi* al vector $\mathbf{i_{nodes\_sig}}$ que almacena los siguientes nodos que pueden proceder al paso de mensajes.

Una vez que se ha completado este proceso para todos los nodos que había en el vector $\mathbf{i_{nodes}}$ se reinicializa dicho vector con los nodos añadidos al vector auxiliar $\mathbf{i_{nodes\_sig}}$ y se comienza todo el procedimiento.

Este proceso iterativo termina cuando el vector $\mathbf{i_{nodes\_sig}}$ se queda vacío, es decir, cuando ya no quedan más nodos por enviar mensajes.

Ahora se comienza con el paso de mensajes "hacia arriba" en el que serán necesarias las variables mostradas en la Tabla D–4.

Tabla D–4. Variables necesarias para el paso de mensajes "hacia arriba" en matlab.

Variables	Breve descripción	Inicialización
$\mathbf{messages_{up}}$	Matriz (de tamaño $N \times L$ siendo L el tamaño de la constelación) que contiene los mensajes "hacia arriba" para todos los posibles valores del alfabeto de x_i. Simboliza a $m_{p(i)\to i}(x_i)$ de las ecuaciones (4-19)-(4-20). Se inicializa a uno para unificar las ecuaciones (4-19)-(4-20).	$\mathbf{messages_{up}} = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$
$\mathbf{children_{pi}}$	Vector que contiene los hijos de *pi* (distintos al nodo *i*), es decir, contiene los nodos hermanos del nodo *i*.	-
$\mathbf{messages_{children}}$	Matriz que contiene el producto de los mensajes enviados por los nodos hijos a su padre para todos los posibles valores del alfabeto de x_i. Simboliza a $\prod_{\{j\,\mid j\neq i, p(j)=p(i)\}} m_{j\to p(i)}(x_{p(i)})$ de las ecuaciones (4-19)-(4-20).	-
$\mathbf{message_{parent}}$	Vector que contiene el mensaje de *ppi* a *pi* para todos los posibles valores del alfabeto de $x_{p(i)}$. Simboliza a $m_{p(p(i))\to p(i)}(x_{p(i)})$ de la ecuación (4-19).	-
$\mathbf{i_{nodes}}$	Vector que contiene los nodos que pueden realizar el paso de mensajes en la siguiente iteración (dada que ya tienen el mensaje de su padre).	Se inicializa con los nodos hijos del nodo raíz
$\mathbf{i_{nodes\_sig}}$	Vector auxiliar que almacena los nodos hijos del nodo *i* y que pueden realizar el paso de mensajes en las próximas iteraciones.	(vacío)
i	Nodo hijo que envía el mensaje en una iteración dada.	-
pi	Nodo padre del nodo *i*.	-
ppi	Nodo padre del nodo *pi*.	-

El paso de mensajes "hacia arriba" se comienza por los hijos del nodo raíz y por ello se inicializa $\mathbf{i_{nodes}}$ con estos nodos. Se van cogiendo los nodos de $\mathbf{i_{nodes}}$ uno a uno. Sea i el nodo actual y pi el padre de dicho nodo. Lo primero es buscar si el nodo pi tiene hijos distintos al nodo i, es decir, hay que buscar a los hermanos de i. Si los tiene, es necesario realizar el producto de los mensajes de los hermanos de i a pi (que están almacenas en la matriz creada en el paso de mensajes anterior $\mathbf{messages_{down}}$) y se almacena dicho producto en la variable $\mathbf{messages_{children}}$. En caso de que i no tenga hermanos la variable $\mathbf{messages_{children}}$ valdrá uno. Seguidamente se busca al padre de *pi*, es decir, a *ppi* y se almacena el mensaje de *ppi* a *pi* (que es el mensaje guardado en la fila *pi* de $\mathbf{messages_{up}}$) en la variable $\mathbf{messages_{parent}}$. En el caso de que *pi* fuera el nodo raíz, *ppi* valdría cero y $\mathbf{message_{parent}}$ valdría uno ya que $\mathbf{messages_{up}}$ se ha inicializado a uno, con lo cual se están uniendo las ecuaciones (4-19)-(4-20) en una sola expresión.

A continuación se multiplican las dos variables $\mathbf{messages_{children}}$ y $\mathbf{message_{parent}}$ con la correspondiente función $f(x_i|x_{pi};\mathbf{z},\mathbf{C})$ (habiendo sustituido en esta última los correspondientes valores de z_i, z_{pi}, C_{ipi}, C_{ii} y C_{pipi}) y se suma para todos los valores de x_{pi}. El resultado dependería de x_i, por lo que se debe hacer esto último para todos los posibles valores del alfabeto de x_i y por ello se necesita el for con índice *kk* del código matlab.

Una vez completado el mensaje, se añaden los nodos hijos de *i* al vector $\mathbf{i_{nodes\_sig}}$ que almacena los siguientes nodos que pueden proceder al paso de mensajes.

Una vez que se ha completado este proceso para todos los nodos que había en el vector $\mathbf{i_{nodes}}$ se reinicializa dicho vector con los nodos añadidos al vector auxiliar $\mathbf{i_{nodes\_sig}}$ y se comienza todo el procedimiento.

Este proceso iterativo termina cuando el vector $\mathbf{i_{nodes\_sig}}$ se queda vacío, es decir, cuando ya no quedan más nodos por enviar mensajes.

D.5 Función GTA_SIC

Para cada matriz de canal, vector transmitido por dicho canal y varianza de ruido dada se aplica la recursividad vista en el Apartado 4.4.1y que se resume a continuación.

$\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ N$

$$\mathbf{z} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y}$$

$$\mathbf{C} = \sigma^2\left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \frac{\sigma^2}{E_s}\mathbf{I}\right)^{-1}$$

$$k_i = \arg\min_{\mathrm{j}\notin(\mathrm{k}_1,\ldots,\mathrm{k}_{\mathrm{i}-1})} C_{jj}$$

Se calcula el árbol de expansión máximo (con nodo raíz x_{k_i})

Se realiza el intercambio de mensajes hacia abajo (desde los extremos al nodo raíz).

Se calcula la variable predicha según (4-23)

$$\hat{x}_{k_i} = \arg\max_{a} belief_i(a) \qquad a \in \mathcal{A}$$

Se realiza la cancelación de interferencias

$$\mathbf{y} = \mathbf{y} - \mathbf{H}_{\mathrm{k_i}}\hat{x}_{k_i}$$

$$\mathbf{H} = \mathbf{H}_{\bar{\mathbf{k}}_\mathbf{i}}$$

$\boldsymbol{end}$

A diferencia de como se hacía para ZF-SIC y MMSE-SIC, las columnas de **H** no se ponen a cero, sino que se eliminan (para no tener estas ramas en cuenta en el árbol calculado en la siguiente iteración).

Como el índice que corresponde al menor elemento diagonal C_{jj} puede ser cualquiera y al final de cada iteración hay que eliminar dicha columna de **H**, se pierde el índice original al que corresponde cada columna de **H**. Por ello, es necesaria una variable auxiliar que se vaya actualizando con el índice al que corresponde cada columna de **H** después de cada iteración. A esta variable se la ha denominado en el código **original_ind**.

D.6 Función EP

Para cada matriz de canal, vector transmitido por dicho canal y varianza de ruido dada se aplica la recursividad vista en el Apartado 5.1 y que se resume a continuación.

1. Inicializar ${\gamma_i}^{(0)} = 0$ y ${\Lambda_i}^{(0)} = {E_s}^{-1}$ para todo $i = 1, \ldots, N$.

2. **for** $\ell = 0$ **to** L_{EP}

   a. Calcular $q_i{}^{(\ell)}(x_i) = \mathcal{N}(x_i : \mu_i{}^{(\ell)}, \sigma_i{}^{2(\ell)})$

   donde $\mu_i{}^{(\ell)} = \mu_{EP}(i)$ y $\sigma_i{}^{2(\ell)} = \sum_{EP}(i,i)$

   b. Calcular $q_i{}^{(\ell)\setminus i}(x_i) = \mathcal{N}(x_i : t_i{}^{(\ell)}, h_i{}^{2(\ell)})$

   donde

   $$h_i{}^{2(\ell)} = \frac{\sigma_i{}^{2(\ell)}}{1 - \sigma_i{}^{2(\ell)}\Lambda_i{}^{(\ell)}}$$

   $$t_i{}^{(\ell)} = h_i{}^{2(\ell)}\left(\frac{\mu_i{}^{(\ell)}}{\sigma_i{}^{2(\ell)}} - \gamma_i{}^{(\ell)}\right)$$

   c. Calcular la media $\mu_{p_i}{}^{(\ell)}$ y la varianza $\sigma_{p_i}{}^{2(\ell)}$ de la distribución $\hat{p}_i{}^{(\ell)}(x_i)$[17].

   Estas medias se calculan de la siguente manera:

   $$\mu_{p_i}{}^{(\ell)} = \sum_{x_i \in \mathcal{A}} x_i \, \hat{p}_i{}^{(\ell)}(x_i)$$

   $$\sigma_{p_i}{}^{2(\ell)} = \sum_{x_i \in \mathcal{A}} \left(x_i - \mu_{p_i}{}^{(\ell)}\right)^2 \hat{p}_i{}^{(\ell)}(x_i)$$

   d. Actualizar la pareja $(\gamma_i{}^{(\ell+1)}, \Lambda_i{}^{(\ell+1)})$ según las ecuaciones

   $$\Lambda_i{}^{(\ell+1)} = \frac{1}{\sigma_{p_i}{}^{2(\ell)}} - \frac{1}{h_i{}^{2(\ell)}}$$

   $$\gamma_i{}^{(\ell+1)} = \frac{\mu_{p_i}{}^{(\ell)}}{\sigma_{p_i}{}^{2(\ell)}} - \frac{t_i{}^{(\ell)}}{h_i{}^{2(\ell)}}$$

end

[17] No se puede olvidar el factor de normalización de $\hat{p}_i{}^{(\ell)}(x_i)$.

REFERENCIAS

[1] Goldberger, J. & Leshem, A. 2011, "MIMO Detection for High-Order QAM Based on a Gaussian Tree Approximation", Information Theory, IEEE Transactions on, vol. 57, no. 8, pp. 4973-4982.

[2] Goldberger, J. 2013, "Improved MIMO detection based on successive tree approximations", Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on, pp. 2004.

[3] Kschischang, F.R., Frey, B.J. & Loeliger, H.-. 2001, "Factor graphs and the sum-product algorithm", Information Theory, IEEE Transactions on, vol. 47, no. 2, pp. 498-519.

[4] Kailath, T., Vikalo, H. & Hassibi, B. 2003, "MIMO Receive Algorithms", California Institute of Technology, Stanford University.

[5] Foschini, G.J. 1996, "Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas", Bell Labs Technical Journal, vol. 1, no. 2, pp. 41-59.

[6] Golden, G.D., Foschini, C.J., Valenzuela, R. & Wolniansky, P.W. 1999, "Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture", Electronics Letters, vol. 35, no. 1, pp. 14-16.

[7] Bindu, E. & Reddy, B.V.R. 2013, "Performance analysis of multiple detector algorithms with optimal ordering for V-BLAST architecture in MIMO system", Advance Computing Conference (IACC), 2013 IEEE 3rd International, pp. 319.

[8] Chow, C. & Liu, C. 1968, "Approximating discrete probability distributions with dependence trees", Information Theory, IEEE Transactions on, vol. 14, no. 3, pp. 462-467.

[9] Prim, R.C. 1957, "Shortest connection networks and some generalizations", Bell System Technical Journal, The, vol. 36, no. 6, pp. 1389-1401.

[10] Yedidia, J.S., Freeman, W.T. & Weiss, Y. 2001, "Understanding belief propagation and its generalizations", Proc. Int. Joint Conf. on Artificial Intelligence.

[11] Grotschel, M., Lov'asz, L. & Schriver, A. 1993, "Geometric Algorithms and Combinatorial Optimization", Springer Verlag, 2nd edition.

[12] Goldberger, J. 2014, "MIMO detection based on averaging Gaussian projections", Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on, pp. 1916.

[13] Céspedes, J., M.Olmos, P., Sánchez-Fernández, M. & Perez-Cruz, F. 2014, "Expectation Propagation Detection for High-Orden High-Dimensional MIMO Systems", Communications, IEEE Transactions on, vol. 62, no. 8.

[14] Minka, T. 2001, "Expectation propagation for approximate Bayesian inference", in Proc. 17th Conf. Uncertainty Artif. Intell, pp. 362-369.

[15] Minka, T. 2001, "A family of algorithms for approximate Bayesian Inference", Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., MIT, Cambridge, MA, USA.

[16] Bishop, C. 2006, "Pattern Recognition and Machine Learning", Springer Science, 1st edition.