CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN DE ELECTRÓNICA DEL ESTADO SÓLIDO

"Sistemas de Procesamiento de Señales para el Análisis de Información Multidimensional"

Tesis que presenta

M. en C. LUZ NOÉ OLIVA MORENO

Para obtener el grado de
DOCTOR EN CIENCIAS

En la Especialidad de
INGENIERÍA ELÉCTRICA

ASESORES

Dr. José Antonio Moreno Cadenas Dr. Felipe Gómez Castañeda

México, D.F. **2008**

OBJETIVO

El objetivo de este trabajo doctoral es la investigación y la aplicación de las técnicas de procesamiento de señales mediante modelos neuronales artificiales, con las cuales, será posible reducir las componentes de datos estadísticos con el propósito de identificar a aquellos que pertenezcan a una misma clase; así como para desarrollar e implementar un sistema automatizado con estas características, cuyas propiedades permitan tener la posibilidad de visualización para tomar decisiones.

Este trabajo esta dedicado a mi madre

Gilberta Moreno Oliva

(1952-2007)

Madre te convertiste en lo más grande para mi..

Pusiste en mi el fervor, la paciencia y la comprensión

Fuiste tu quien dio su vida por mi..

Y siempre te llevo en mi mente y en mi corazón

Tus enseñanzas fueron muchas

Pero más lo fueron los bellos recuerdos

Y en aquel niño al que un día regañabas

Para aprender de ti tus consejos..

Hoy te doy las gracias y también te digo

Madre mi labor es la tuya..

AGRADECIMIENTOS

Además agradezco:

- A mis asesores de tesis de doctorado los Doctores Felipe Gómez Castañeda y José Antonio Moreno Cadenas por todo el apoyo y las enseñanzas brindadas.
- Quiero agradecer a los Doctores: Alfredo Reyes Barranca, Rodolfo Quintero Romo, Juan Carlos Sánchez García y Heron Molina lozano que fungieron como mis sinodales en mi examen de doctorado.

- Agradecimientos especiales a: Dr. Miguel A. Alemán, M. en C. Luis Martín Flores Nava, Dr. Jesús Ezequiel Molinar Solís, Dra. Brenda Aline García Lozano, M. en C. Jesús de la Cruz Alejo, M. en C. José A. Medina, M. en C. Claudia A. López Rodríguez, Yesenia Cervantes Aguirre y M. en C. Jacqueline Arzate Gordillo, por sus sabios consejos e incondicional ayuda.
- A mis amigos y compañeros del CINVESTAV: ....
  (M. en C. Sergio Garduza, M. en C. José Luis Ochoa, al Dr. Víctor Cabrera)
- A Graciela Meza Castellanos, Raúl Montaño Molina

- Y quiero manifestar mi agradecimiento al CONACYT por la beca otorgada.

RESUMEN

El Análisis de Componentes Independientes (ICA *Independent Component Analysis*) es una técnica estadística utilizada para encontrar las componentes independientes dentro de un espacio multidimensional, en el cual no se conoce información alguna de los datos. En este trabajo se muestran algunas de las técnicas de ICA. Comenzando con ICA por descorrelación no lineal, siendo este trabajo uno de los pioneros de ICA. La siguiente técnica trata sobre el algoritmo INFOMAX, el cual se basa en la maximización de la información por medio de la entropía relativa. La última metodología de ICA, se basa en la maximización de la no gaussianidad. Al aplicar la transformación de las componentes obtenidas por ICA y los datos de entrada, se obtiene la solución al problema de separación ciega de fuentes (BSS *Blind Source Separation*).

El presente trabajo explica cada una de estas técnicas de ICA, las cuales se consideraron por su viabilidad para ser implementadas en dispositivos VLSI programables, mostrando los resultados relevantes obtenidos.

ABSTRACT

The Independent Component Analysis (ICA) is a statistical technique employed to find the Independent Component in a multidimensional space, without knowledge about information data. In this work some techniques of ICA are shown. First of all we had begun with ICA by Nonlinear Decorrelation, a pioneering work in the ICA field. The next part goes on to deal with INFOMAX Algorithm, which is based on the maximization of Information deal with relative entropy and the last ICA methodology is based on the non-gaussianity maximization. Applying the transformation of the components obtained by ICA and input data, the Blind Source Separation problem solution is obtained.

This work explains every technique for the ICA and considers their viability to be implemented on a VLSI embedded system, showing the relevant results.

ACRÓNIMOS Y SÍMBOLOS

ICA *Independent Component Análisis – Análisis de Componentes Independientes*
BSS *Blind Source Separation – Separación Ciega de Fuentes*
PCA *Principal Component Analysis – Análisis de Componentes Principales*
IC *Independent Component – Componente Independiente*
INFOMAX *information maximization – Maximización de la Información*
MLE *Maximum Likelihood Estimator – Máxima Verosimilitud*
FPGA *Field Programmable Gate Array - Arreglo de Compuertas Programables de Campo*
PLD *Programable Logic Devices*
HDL *Hardware Description Language – Lenguaje de descripción de Hardware*
VHDL *Very High Speed Integrated Circuit Hardware Description Language*
DSP *Digital Signal Processor- Procesador Digital de Señales*
H-J *Jutten-Hérault*
EEG *Electroencephalograph - electroencefalografía*
fdp *función de densidad de probabilidad*
kurt *kurtosis - Curtosis*

A	Matriz de mezcla
s*(t)*	Vector de señales o fuentes independientes
e*(t)*	Vector de señales o fuentes mezcladas
n*(t)*	Vector de ruido
$\hat{\mathbf{s}}_i(t)$	Vector de señales o fuentes estimadas
V	Matriz de transformación de blanqueo
z*(t)*	Vector de señales blanqueadas

W	Matriz de pesos de una red neuronal, o matriz que estima a la matriz de mezcla **A**, dependiendo del contexto
P	Matriz de Permutación
I	Matriz Identidad
E	Matriz de vectores propios
D	Matriz de valores propios
R	Matriz de correlación
J	Matriz Jacobiana
U	Matriz de vectores ortogonales
H(x)	Entropía de "x"
$E\{x\}$	Esperanza o valor esperado de "x"
I(X;Y)	Información mutua entre X e Y
λ_j	Valores propios

i	Número de fuentes
a_{ij}	Coeficiente de mezcla
$s_i(t)$	Señal de fuente independiente
$e_i(t)$	Señal de mezclada
$\hat{s}_i(t)$	Señal estimada de la fuente independiente
$n(t)$	Señal de ruido
w_{ij}	Coeficiente de pesos
$p(x)$	Probabilidad de "x"
η	Razón de aprendizaje
σ	Desviación estándar
σ^2	Varianza
μ	Valor medio
cov()	Covarianza
corr()	Correlación
$D(f_1 \| \| f_2)$	Divergencia de Kullback-Leibler entre las funciones de distribución f_1 y f_2
$\mathbf{N}(\mu,\sigma^2)$	Distribución gaussiana de media μ y varianza σ^2
$\boldsymbol{y}$	Vector de señales recuperadas o estimadas

CONTENIDO

INTRODUCCIÓN

Las redes neuronales artificiales permiten emular las características propias de los humanos: la capacidad de memorizar y asociar hechos. Son un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos de un sistema que es capaz de adquirir conocimiento a través de la experiencia. Una red neuronal artificial es un sistema para el tratamiento de la información cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano, la neurona.

ICA se encuentra muy relacionado al problema de separación ciega de fuentes (BSS) el cual consiste en determinar a través de un arreglo de transductores, las señales de las fuentes originales que intervienen en una mezcla, sin información alguna de las señales originales ni de las ponderaciones de la mezcla. Este es un problema clásico de procesamiento de señales. Basándonos en la observación biológica del cerebro, en donde los impulsos son generados simultáneamente en las terminales nerviosas y se transmiten en forma de mezclas de señales a través de las fibras nerviosas, observando que el sistema nervioso central es capaz de separar las múltiples señales generadas en cada terminal nerviosa por medio de las neuronas. El concepto de la separación ciega de fuentes ha abierto un capítulo de gran relevancia en la Teoría de Señales, con aplicaciones en una gran diversidad de campos como: radiocomunicaciones, procesamiento de voz y de imágenes, biomedicina, transductores inteligentes, etc.

Un problema clásico (enunciado por primera vez en 1985 por Christian Jutten y Jeanny Hérault [JUT91a]) es conocido como *Cocktail Party* y consiste en obtener las conversaciones individuales de las personas, a través de un arreglo de micrófonos (transductores) ubicados estratégicamente dentro de una habitación, asumiendo que las mezclas de señales se generan en el medio en que se propagan y en los transductores (se considera un modelo de mezcla lineal, es decir, que el medio y los transductores se comportan de forma lineal). ICA se basa en la distribución estadística de los datos de mezcla para encontrar las componentes y el BSS se basa en la transformación lineal que independice las señales recibidas a través de los transductores.

ICA tiene un enfoque estadístico y parte de la hipótesis de que las señales originales (fuentes) son estadísticamente independientes y los procedimientos que lo siguen, se basan en propiedades de las fuentes y de las observaciones (mezclas), caracterizadas por sus distribuciones de probabilidad. Se pretende que si las señales originales son estadísticamente independientes, las señales recuperadas también deben de serlo. Por lo tanto, los métodos relacionados con ICA consideran como condición primordial la independencia estadística, buscando dentro de espacios multidimensionales las componentes principales (en la dirección de máxima varianza) de las variables a determinar.

Existen varias técnicas estadísticas para la determinación de las componentes independientes y se mencionarán en este trabajo de tesis.

La tesis está dividida en 5 capítulos, cuyo contenido se menciona brevemente:

- **Capítulo 1**. Se efectúa la introducción al problema de ICA, presentando algunas restricciones e indeterminaciones de ICA. También se muestran las arquitecturas y sus reglas de aprendizaje de la red, desarrollados bajo enfoques estadísticos. Algunos de los conceptos son mencionados en los apéndices.

- **Capítulo 2**. Se muestran los resultados del desempeño de las redes mostradas en el capítulo 1 empleando MATLAB como lenguaje de programación.

- **Capítulo 3**. Se muestran las arquitecturas y los resultados obtenidos de las implementaciones de la redes en DSP y FPGA.

- **Capítulo 4**. Finalmente en este capítulo se presentan las conclusiones generales de este trabajo de tesis, así como los trabajos a futuro que se pueden realizar.

- **Apéndice**. Se incluyen algunos conceptos básicos como probabilidad y estadística, álgebra lineal y teoría de la información, que ayudan a comprender mejor el contenido de la presente tesis. También, se muestra el código correspondiente de la implementación del DSP.

CAPÍTULO 1

Introducción al problema del Análisis de Componentes Independientes (ICA)

1.1- Introducción

El análisis de componentes independientes (**ICA** *Independent Component Analysis*) es un método para encontrar los factores o componentes de datos estadísticos multivariados (multidimensional). Este método es utilizado frecuentemente en el problema de separación ciega de fuentes (**BSS** *Blind Source Separation*)[HYV00, WON00]; el cual consiste en recuperar las señales de las fuentes independientes generadas por ***s*** fuentes, teniendo tan solo acceso a las señales mezcladas ***e***, recibidas por los transductores. La ponderación de la mezcla ***A*** es totalmente desconocida, y representa al medio en donde se propagan dichas señales.

ICA es un método que encuentra en un sistema multidimensional la dirección de los ejes, con el objetivo de obtener una transformación lineal en la mezcla, la cual resulte en variables estadísticamente independientes, tánto como sea posible [JUT91, COM94, BEL95, CAR96]. **ICA** generaliza la técnica de análisis de componentes principales (**PCA** *Principal Component Analysis*), en la que suele basarse.

En este capítulo se dará una breve introducción al modelo de mezcla y algunas de las soluciones hasta ahora desarrolladas. En los apéndices A y B se incluyen las bases de estadística y de teoría de la información necesaria para entender adecuadamente este capítulo.

1.2 Formulación del problema

El problema abordado en esta tesis consiste en separar y estimar las múltiples fuentes de señal de una mezcla, a través de un arreglo discreto de transductores, sin un previo conocimiento de las

características de las fuentes originales y de los canales de transmisión. Como punto de partida, se supone un número igual de fuentes desconocidas y de transductores, en donde cada transductor por sus características y su posición, reciben una combinación lineal diferente de cada una de las señales. En la figura1.2.1 se muestra un esquema del arreglo de transductores y de fuentes originales de tamaño $n \times n$.

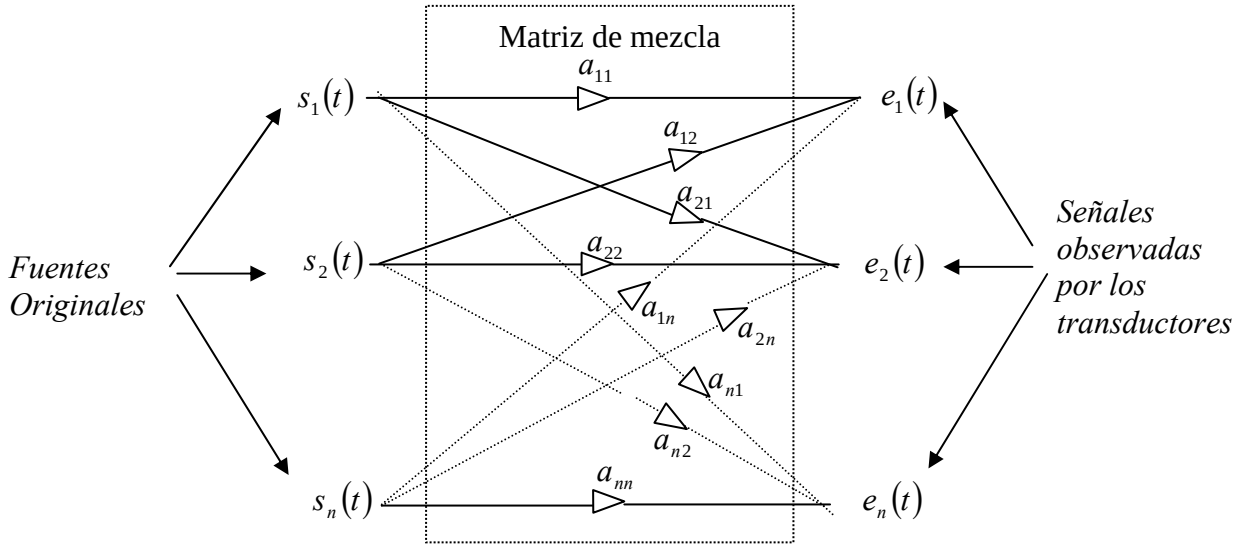


Fig. 1.2.1 Arreglo de transductores y fuentes originales.

El sistema mostrado es descrito por la siguiente ecuación:

$$e_i(t) = \sum_{i,j=1}^{n} a_{ij} s_j(t) \qquad (1.2.1)$$

$$(i, j = 1,2,.....,n)$$

En donde: $e_i(t)$ Es la señal continua en el tiempo, de la salida del i-ésimo transductor.

$s_j(t)$ Es la señal desconocida, emitida por la j-ésima fuente.

a_{ij} Es un número escalar de la combinación lineal.

Escrito en forma matricial, la ecuación 1.2.1 se expresa como:

$$\mathbf{e}(t) = \mathbf{A}\mathbf{s}(t) \qquad (1.2.2)$$

En donde A representa la matriz de mezcla de tamaño $n \times n$. Un modelo más general considera que ingresa ruido en los sensores; por lo tanto la expresión 1.2.2 se expresa como:

$$\mathbf{e}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t) \qquad (1.2.3)$$

donde $\mathbf{n}(t)$ es un vector que representa las variables del ruido.

En este modelo se desprecian los retardos de tiempo que ocurren durante la mezcla, por lo tanto se dice que es un modelo de mezclado instantáneo.

En la figura 1.2.2 se muestran dos ejemplos de señales de fuentes independientes (originales) con distribución uniforme. En la parte superior derecha de la imagen se observan los datos en un espacio bidimensional (sin la variable tiempo) mientras que en la izquierda y en la parte inferior, se muestra la distribución de cada una de las variables independientes (histogramas), así como su representación en el tiempo.

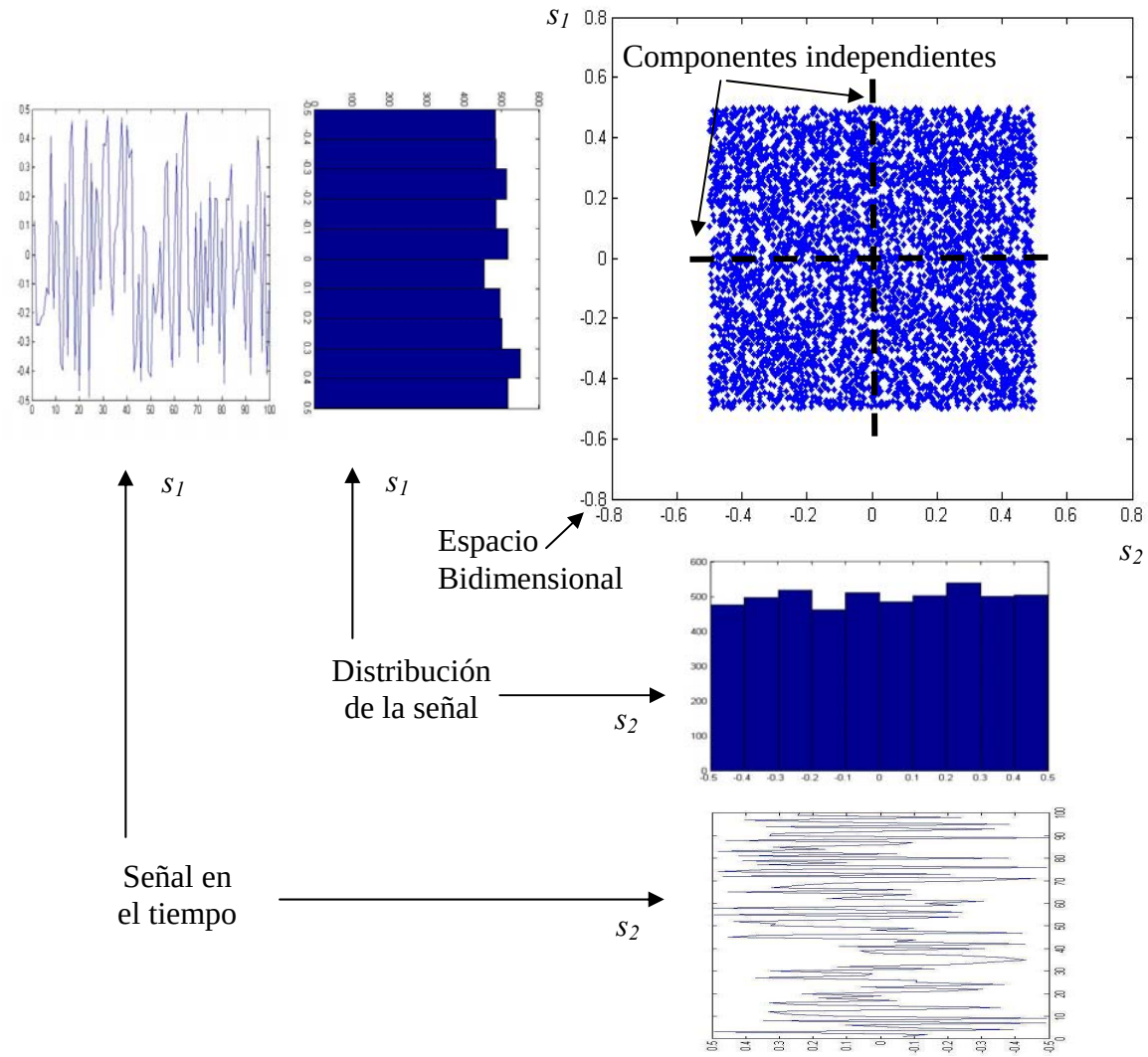


Fig. 1.2.2 Muestras de componentes independientes s_1 y s_2 con distribución uniforme.

Considerando la expresión 1.2.2 y asumiendo una matriz de mezcla ***A***, el resultado de la variable $\mathbf{e}(t)$ se muestra en la figura 1.2.3. En la parte superior derecha de la imagen son mostrados los datos en un plano bidimensional. A la izquierda y en la parte inferior se muestran sus distribuciones correspondientes.

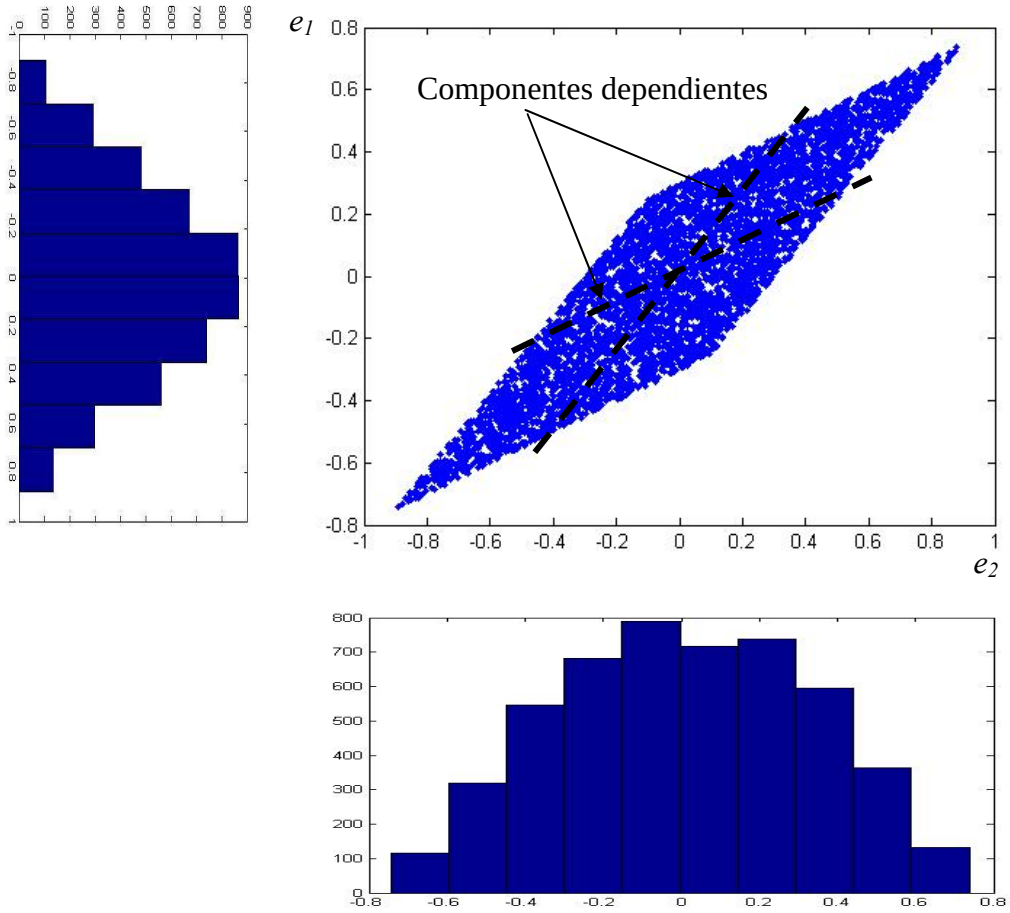


Fig. 1.2.3 Mezcla de las señales independientes s_1 y s_2.

El problema cosiste en encontrar una transformación que encuentre la inversa de la matriz con la cual se mezcló. Para obtener una solución al problema, se debe cumplir lo siguiente: $\mathbf{A} \in \mathbf{R}^{n \times n}$ y Det. $\mathbf{A} \neq 0$ (***A*** no debe ser singular o casi singular). En este sistema solamente las señales de salida $\mathbf{e}(t)$ de cada transductor están disponibles. La matriz de mezcla ***A*** y las señales de fuentes primarias $\mathbf{s}(t)$ son desconocidas.

Aplicar un método convencional (como el de Gauss [STE90]) para encontrar la matriz de mezcla en este problema es imposible, ya que no se cuenta con más información que las señales adquiridas por los transductores. Por otra parte, una aproximación para resolver este problema es utilizar las propiedades estadísticas de las señales (existiendo otros métodos como el geométrico [PUN95], pero en este trabajo no se tratará). Existen cuatro modelos de mezcla posibles que se mencionarán a continuación:

Modelo de mezcla no lineal (Modelo General). Para mezclas instantáneas, el modelo de datos no lineal tiene la forma general:

$$\mathbf{e}(t) = f(\mathbf{s}(t)) \tag{1.2.4}$$

en donde f es una función de n variables del espacio **s** hacia el espacio **e**. Una característica del problema ICA no lineal es que en el caso general, la solución siempre existe. Una razón de esto, es que si dos variables aleatorias son independientes, cualquiera de sus funciones no lineales también lo serán.

Modelo de mezcla post-no lineal. Donde el medio es lineal y la no linealidad es introducida en los sensores.

$$\mathbf{e}(t) = f(\mathbf{A}\mathbf{s}(t)) \tag{1.2.5}$$

Taleb y Jutten han desarrollado métodos de BSS para este caso. Sus principales resultados pueden encontrarse en [JUT00] y una descripción breve de sus estudios puede encontrarse en [TAL99]

Modelo de mezcla lineal. En este caso se considera que tanto el medio como los sensores se comportan de forma lineal. Siendo éste el caso considerado en este trabajo.

$$\mathbf{e}(t) = \mathbf{A}\mathbf{s}(t) \tag{1.2.6}$$

Modelo convolutivo. Este modelo considera que el medio de propagación actúa como un filtro con p entradas y q salidas. Usualmente se admite que es introducido por el medio y los sensores, en un medio lineal y estacionario.

El modelo considerado en este trabajo, es el modelo de mezcla lineal en el cual las señales son mezcladas instantáneamente y los algoritmos de separación de fuentes tratan de explotar la diversidad espacial producida por la mezcla percibida por los distintos sensores. Por otra parte, los modelos convolutivos se proponen con objeto de aprovechar la diversidad espectral, aunque el enfoque fundamental en este trabajo es esencialmente espacial y busca la estructura a través de los sensores, no a lo largo del tiempo.

1.3 Restricciones en ICA.

En el presente trabajo se va a considerar únicamente el caso de mezclas lineales e instantáneas modeladas con las expresión 1.2.6, como ya fue mencionado en la formulación del problema. La solución consiste en aprovechar las propiedades estadísticas, donde cada componente representará una señal independiente.

Empleando una transformación de datos, se logrará hacer una estructura de mezcla, una más visible, donde se pueda obtener cada conjunto de datos que pertenezcan a un mismo grupo. Al agregar a cada dato su variable de tiempo correspondiente, ésta proporcionará cada unas de las señales de fuentes independientes (***s****(t)*).

Para asegurar que los modelos ICA den un buen estimado de cada una de las componentes, se deben hacer las siguientes consideraciones:

1.- Las componentes independientes deben ser estadísticamente independientes. Es decir, que las variables $s_1, s_2, ..., s_n$ son independientes, sí y solo sí, el valor de una variable no da información acerca del valor de la otra. Matemáticamente, la independencia estadística se puede definir como la igualdad entre la función de densidad de probabilidad (fdp) de juntura y el producto de las fdp's marginales.

$$p(s_1, s_2, ..., s_n) = p(s_1)p(s_2)....p(s_n) = \prod_{i=1}^{n} p(s_i) \tag{1.3.1}$$

2.- Las componentes independientes deben tener distribuciones no gaussianas.

Para la estimación del modelo de ICA, las componentes con distribución gaussiana son difíciles de encontrar, ya que presentan una estructura de densidad rotacionalmente simétrica. Proponiendo dos variables s_1 y s_2 con densidades de probabilidad gaussiana, varianza igual a la unidad y media cero ($\mu = 0$, $\sigma^2 = 1$), su probabilidad de juntura se define como:

$$p(s_1, s_2) = \frac{1}{2\pi} \exp(-\frac{s_1^2 + s_2^2}{2}) = \frac{1}{2\pi} \exp(-\frac{\|\mathbf{s}\|^2}{2}) \tag{1.3.2}$$

Consideremos que la matriz de mezcla ***A*** es ortogonal por efecto del blanqueado (es decir, es una transformación que efectúa la descorrelación y normalización de la varianza de los datos de entrada **e**, como se mostrará en el capítulo 1.7.1). Por lo tanto, la transformación **e** = **As** se expresa como $\mathbf{s} = \mathbf{A}^T \mathbf{e}$ y se substituye en 1.3.2:

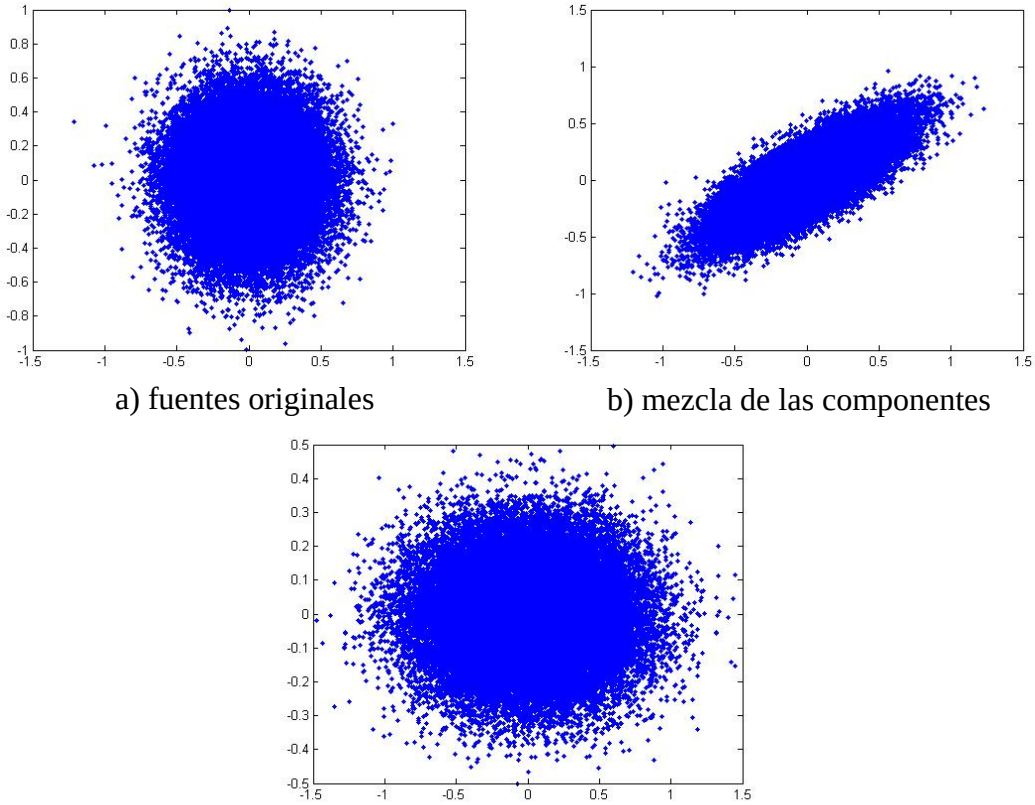
$$p(s_1, s_2) = \frac{1}{2\pi} \exp(-\frac{\|\mathbf{A}^T\mathbf{e}\|^2}{2}) \tag{1.3.3}$$

Debido a la ortogonalidad de **A**, se tiene que $\|\mathbf{A}^T\mathbf{e}\|^2 = \|\mathbf{e}\|^2$. Entonces se tiene que:

$$p(s_1, s_2) = \frac{1}{2\pi} \exp(-\frac{\|\mathbf{e}\|^2}{2}) \tag{1.3.4}$$

Se observa en la ecuación 1.3.4 que la distribución no depende de la matriz de mezcla **A**, por lo tanto no hay forma de inferir en dicha matriz de mezcla. La figura 1.3.1a muestra la distribución de los datos de las fuentes originales para dos distribuciones gaussianas. La figura1.3.1b muestra la mezcla de las componentes y la figura 1.3.1c muestra las señales descorrelacionadas.

En la figura 1.3.1c se puede observar que las densidades son rotacionalmente simétricas. Por lo tanto no se puede obtener información de la dirección de sus vectores. Esto implica que no pueden estimarse las columnas de la matriz de mezcla ***A***.



a) fuentes originales b) mezcla de las componentes

c) Señales descorrelacionadas

Fig 1.3.1 distribución de dos variables gaussianas independientes.

3.- Por simplicidad asumimos una matriz de mezcla **A**, como una matriz cuadrada.

En otras palabras, el número de mezclas observadas es igual al número de componentes a encontrar. Esto simplifica la estimación de la matriz de mezcla **A**.

$$\mathbf{e}(t) = \mathbf{A}\mathbf{s}(t) \tag{1.3.5}$$

También se asume que la matriz de mezcla **A** es no singular.

1.4 Indeterminaciones de ICA

En los métodos de ICA, existen algunas indeterminaciones que no son identificables, como son la amplitud o potencia y el orden de las fuentes con respecto a las señales originales, pero se considera que las señales recuperadas son válidas. Es decir, que las señales pueden ser multiplicados o divididos por algún escalar (α_i).

$$e_i = \sum_{i,j=1}^{n} (\frac{1}{\alpha_i}\mathbf{a}_{ij})(s_j \alpha_i) \tag{1.4.1}$$

Una forma habitual para resolver el problema de la amplitud, es ajustar solo su distribución a una varianza unitaria $E\{\mathbf{s}_i^2\} = 1$, sin importar su amplitud.

Otra indeterminación, concierne al orden de las señales recuperadas. Las componentes se pueden permutar, sin que por ello varíe el orden de las componentes de las observaciones **e***(t)*.

$$\mathbf{e}(t) = \mathbf{A}\mathbf{s}(t) = \mathbf{A}\mathbf{P}^{-1}\mathbf{P}\mathbf{s}(t) \tag{1.4.2}$$

En donde **Ps***(t)* son las fuentes originales en un orden diferente y $\mathbf{AP}^{-1}$ es la matriz a estimar, resultando imposible obtener la estimación de las fuentes en el mismo orden de las originales ($\mathbf{P}^{-1}$).

1.5 ICA por descorrelación no lineal.

En esta sección se describirá brevemente una técnica basada en la descorrelación no lineal, empleada por Jutten y Hérault[JUT91a]. Esta fue la primera vez que se empleó el término *"Análisis de Componentes Independientes"*. Este modelo se explica con una arquitectura de una red neuronal y consiste de un filtro adaptable lineal recursivo, como se muestra en la figura 1.5.1. Cada círculo negro representa una operación de suma y cada triángulo representa la función de activación de la neurona. Esta arquitectura es similar a la red neuronal de Hopfield [HAY99].

El sistema es descrito por la siguiente ecuación:

$$\hat{s}_i(t) = e_i(t) - \sum_{i=1}^{n} w_{ij}\hat{s}_j(t) \tag{1.5.1}$$

$$(i, j = 1,2,.....,n) \qquad i \neq j$$

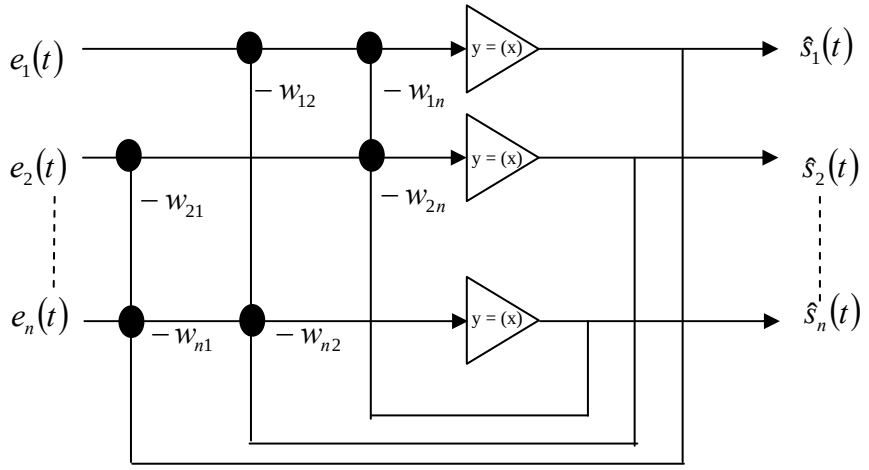


Fig. 1.5.1 Arquitectura neuronal de un filtro adaptable lineal recursivo

En donde: $e_i(t)$ Es la señal continua en el tiempo, de la salida del i-ésimo transductor.

$\hat{s}_i(t)$ Es la señal continua en el tiempo, de la i-ésima salida de la red neurona (valor estimado de la fuente original).

w_{ij} Es el valor del peso sináptico de la red.

La salida $\hat{s}_i(t)$ del i-ésimo operador es el resultado de una suma de la señal de entrada $e_i(t)$ y del producto parcial $-w_{ij}\hat{s}_j(t)$, en donde $j \neq i$. Expresando la ecuación 1.5.1 en notación matricial:

$$\hat{\mathbf{s}}(t) = \mathbf{e}(t) - \mathbf{W}\hat{\mathbf{s}}(t) \tag{1.5.2}$$

De esta ecuación se obtiene:

$$\hat{\mathbf{s}}(t) = (\mathbf{I} + \mathbf{W})^{-1}\mathbf{e}(t) \tag{1.5.3}$$

Siendo ***W*** la matriz de los coeficientes de pesos inhibitorios de la red e **I** la matriz identidad.

Para el análisis de este problema se tomará el caso de *n = 2*. Las ecuaciones 1.2.6 y 1.5.1 son escritas como:

$$e_1(t) = a_{11}s_1(t) + a_{12}s_2(t) \tag{1.5.4}$$

$$e_2(t) = a_{21}s_1(t) + a_{22}s_2(t) \tag{1.5.5}$$

$$\hat{s}_1(t) = e_1(t) - w_{12}\hat{s}_2(t) \tag{1.5.6}$$

$$\hat{s}_2(t) = e_2(t) - w_{21}\hat{s}_1(t) \tag{1.5.7}$$

Sustituyendo $e_1(t)$ y $e_2(t)$ en las primeras dos ecuaciones y despejando a $\hat{s}_1(t)$ y $\hat{s}_2(t)$ se obtiene:

$$\hat{s}_1(t) = \frac{(a_{11} - w_{12}a_{21})s_1 + (a_{12} - w_{12}a_{22})s_2}{1 - w_{12}w_{21}} \tag{1.5.8}$$

$$\hat{s}_2(t) = \frac{(a_{21} - w_{21}a_{11})s_1 + (a_{22} - w_{21}a_{12})s_2}{1 - w_{12}w_{21}} \tag{1.5.9}$$

Se pueden proponer dos pares particulares de coeficientes w_{ij}, dando las soluciones correspondientes:

caso i) si $w_{12} = a_{12}/a_{22}$ y $w_{21} = a_{21}/a_{11}$

Las señales de salida son: $\hat{s}_1(t) = a_{11}s_1(t)$ y $\hat{s}_2(t) = a_{22}s_2(t)$ (1.5.10)

caso ii) si $w_{12} = a_{11}/a_{21}$ y $w_{21} = a_{22}/a_{12}$

De manera que las señales de salida son: $\hat{s}_1(t) = a_{12}s_2(t)$ y $\hat{s}_2(t) = a_{21}s_1(t)$ (1.5.11)

Una condición de estabilidad en el sistema es la siguiente:

$$w_{12}w_{21} < 1 \tag{1.5.12}$$

Debido a que si el producto es igual a la unidad, las ecuaciones 1.5.8 y 1.5.9 se indeterminan o no convergen.

Teóricamente, la arquitectura de la red permite separar fuentes desconocidas. No obstante, el valor de w_{ij} es un coeficiente desconocido.

Jutten y Hérault desarrollaron un algoritmo matemático para la regla de aprendizaje basado en el método del *gradiente descendiente*.

$$\Delta w_{12} = \eta \cdot \frac{dw_{12}(t)}{dt} = \eta \cdot f[\hat{s}_1(t)] \cdot g[\hat{s}_2(t)] \tag{1.5.13}$$

Donde η es la razón de aprendizaje ($\eta > 0$).

Este algoritmo, ajusta los pesos sinápticos (***W***) de la red neuronal en forma continua de acuerdo a la siguiente expresión, conocida como la regla de Hebb o regla de adaptación, dada por:

$$w_{12}(t+\Delta t) = w_{12}(t) + \Delta w_{12} \tag{1.5.14}$$

Para lograr una independencia estadística, Jutten y Hérault propusieron el uso de dos funciones no lineales e impares, preservando la condición de media cero. Expresando las funciones *f* y *g* en series de Taylor de los términos impares:

$$f(x) = \sum_{j=1}^{\infty} f_{2j+1} x^{2j+1} \qquad g(x) = \sum_{m=1}^{\infty} g_{2m+1} x^{2m+1} \tag{1.5.15}$$

Ahora, substituyendo esta expansiones en la ecuación 1.5.13, se obtiene:

$$\frac{dw_{12}(t)}{dt} = \eta \cdot \sum_{j=1}^{\infty}\sum_{m=1}^{\infty} f_{2j+1} g_{2m+1} \hat{s}_1^{2j+1}(t) \hat{s}_2^{2m+1}(t) \tag{1.5.16}$$

Con el propósito de obtener el valor esperado del *gradiente descendiente*, se define el n-ésimo momento de una señal estacionaria, como:

$$E\{X^n\} = \sum_t x^n(t) p(t) \tag{1.5.17}$$

Por lo tanto, la convergencia del algoritmo tomando en cuenta 1.5.17 está sujeta a la siguiente condición:

$$E\left\{\frac{dw_{12}}{dt}\right\} = \eta \cdot \sum_{j=1}^{\infty}\sum_{m=1}^{\infty} f_{2j+1} g_{2m+1} \left\{\hat{s}_1^{2j+1}(t) \hat{s}_2^{2m+1}(t)\right\} = 0 \tag{1.5.18}$$

La convergencia se alcanza cuando los momentos de orden superior $\left\{\hat{s}_i^{2j+1}(t)\hat{s}_j^{2i+1}(t)\right\}$ son cero, esto implica que $\left\{\hat{s}_i^{2j+1}(t)\hat{s}_j^{2i+1}(t)\right\} = \left\{\hat{s}_i^{2j+1}(t)\right\}\left\{\hat{s}_j^{2i+1}(t)\right\} = 0$, obteniendo la independencia estadística (Apéndice A, sección A9) entre $s_i(t)$ y $s_j(t)$. Un gran número de funciones impares pueden ser utilizadas para la separación. Experimentalmente fueron consideradas las siguientes restricciones en *f* y *g* asegurando una convergencia rápida y estable de los pesos [JUT91b]:

a) Para una convergencia estable, *f* debe tener una curvatura positiva y *g* una curvatura negativa.

b) Para una convergencia rápida, *f* y *g* deben ser ortogonales en el origen.

Las funciones *f* y *g* propuestas por Jutten y Hérault son:

$$\text{Par 1: } f(x) = arctg(x) \text{ y } g(x) = sign(x) \tag{1.5.19}$$

$$\text{Par 2: } f(x) = x \qquad y \qquad g(x) = x^3 \tag{1.5.20}$$

A pesar de que: $f(x) = x$ es una función lineal, la combinación de ambas funciones consiguen la separación de fuentes independientes. Un estudio más detallado del trabajo de Hérault y Jutten puede encontrarse en [JUT91a, JUT91b].

1.6 ICA basado en la maximización de la información (INFOMAX).

El principio de maximización de la información también llamado INFOMAX, se encuentra muy relacionado al de máxima verosimilitud [HYV03, GAE90], este método se basa en la medida de la entropía relativa de dos o más distribuciones de probabilidad, como se mencionará a continuación. En la figura 1.6.1 se muestra la arquitectura de la red neuronal, en la que se efectúa la maximización de la información transferencia y la minimización de la información mutua entre sus salidas. Cada entrada $e_i(t)$ está formada por la mezcla de las señales y cada salida $u_i(t)$ es la señal estimada de la fuente independiente. El objetivo es encontrar una matriz $\boldsymbol{W}$ inversa a la matriz de mezcla $\boldsymbol{A}$, que recupere las señales de las fuentes independientes.

Algunos conceptos como la entropía, serán tratados en esta sección (y son definidos en el Apéndice C), con el fin de conocer la razón de información, del cual se basa INFOMAX.

La entropía se encuentra definida como:

$$H(X) = \sum_{i=1}^{n} P(x_i) \log \frac{1}{P(x_i)} \tag{1.6.1}$$

Donde X es el conjunto de variables aleatorias de x_i. La regla de aprendizaje para obtener la matriz $\boldsymbol{W}$ se deriva de la maximización de la entropía ($H(\hat{S})$) de las salidas de la red. Esto fue propuesto por Bell y Sejnowski [BEL95].

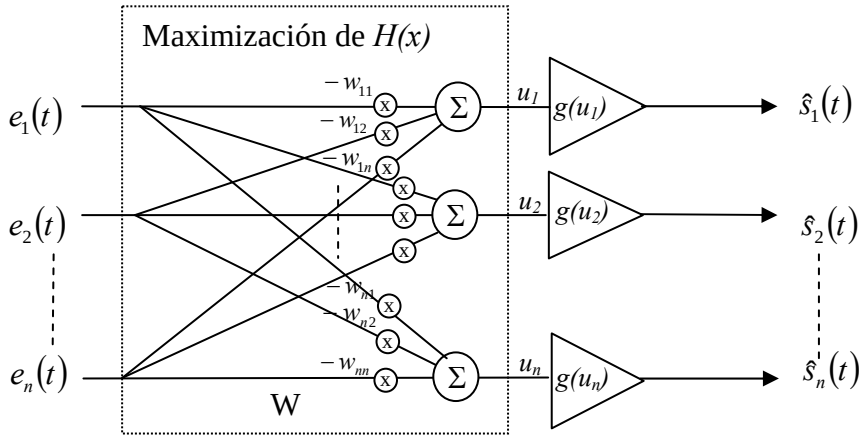


Fig. 1.6.1 Arquitectura de la red neuronal.

La entropía conjunta de la salida de la red, está dada por:

$$H(\hat{s}_1, \hat{s}_2, ...\hat{s}_n) = H(\hat{s}_1) + H(\hat{s}_2) + ... + H(\hat{s}_n) - I(\hat{s}_1, \hat{s}_2, ...\hat{s}_n) \tag{1.6.2}$$

Donde $H(\hat{s}_i)$ son las entropías marginales de la *i-esima* salida, e $I(\hat{s}_1, \hat{s}_2, ..., \hat{s}_n)$ es la información mutua entre las salidas. Maximizar la entropía de juntura $H(\hat{s}_1, \hat{s}_2, ..., \hat{s}_n)$ de la salida de la red, equivale a maximizar las entropías marginales $H(\hat{s}_i)$ y minimizar la información mutua ($I(\hat{s}_i)$).

Por lo tanto, el valor máximo que se puede lograr alcanzar, es cuando no existe información mutua entre sus salidas, es decir que no contienen información adicional acerca de otra salida. En consecuencia, el término de $I(\hat{s}_1, \hat{s}_2, ..., \hat{s}_n)$ será igual a cero y su entropía marginal será igual a su entropía de juntura (entropía máxima).

$$H(\hat{s}_1, \hat{s}_2, ...\hat{s}_n) = H(\hat{s}_1) + H(\hat{s}_2) + ... + H(\hat{s}_n) \tag{1.6.3}$$

Bell y Sejnowski eligieron una función no lineal *g(u)*(función logística), de la cual su derivada es una función de distribución de probabilidad similar a una función super-gaussiana (gaussiana con

distribución más concentrada en la media y más dispersa en los extremos), asumiendo de antemano que las fuentes originales tienen distribuciones super-gaussianas, como se verá más adelante.

Un parámetro de ajuste es la matriz **W**. Esta matriz obtiene su valor óptimo a través de maximizar la entropía $H(\hat{\mathbf{s}})$ con relación a **W**. Por lo tanto, al derivar la ecuación 1.6.2 con respecto a **W**, se obtiene la igualdad con la divergencia de Kullback-Leibler (Ver apéndice C, sección C5) entre el estimado de la distribución de la fuente $p(\hat{\mathbf{s}})$ y la fuente $p(\mathbf{s})$.

$$\frac{\partial H(\hat{\mathbf{s}})}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}(D(p(\mathbf{s}) \,\|\, p(\hat{\mathbf{s}}))) \tag{1.6.4}$$

Considerando el caso del valor **W** óptimo cuando la entropía es máxima (y la información mutua es igual a cero), el estimado de la distribución de la fuente $p(\hat{\mathbf{s}})$ y la fuente $p(\mathbf{s})$ serán iguales. Puesto que, la función no lineal $g_i(u_i)$ a la salida de la red (ver Fig. 1.6.1) no introduce dependencia entre las señales, la información mutua seguirá siendo igual a cero ($I(\hat{s}_1, \hat{s}_2, ..., \hat{s}_n) = 0$).

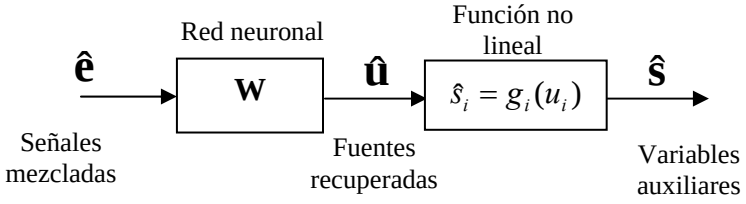


Fig. 1.6.2 Método de máxima entropía para la separación de fuentes.

De acuerdo a la figura 1.6.2, la función de transferencia entre $\hat{s}_i$, u_i se denota como:

$$\frac{p(u_i)}{p(\hat{s}_i)} = \left| \frac{\partial g_i(u_i)}{\partial u_i} \right| \tag{1.6.5}$$

La relación entre $\hat{s}_i$, u_i y la función de transferencia no lineal es (PAP91):

$$p(\hat{s}_i) = \frac{p(u_i)}{\left| \frac{\partial g_i(u_i)}{\partial u_i} \right|} \tag{1.6.6}$$

Para una distribución uniforme de $\hat{s}_i$, se tiene que:

$$p(u_i) = \left| \frac{\partial g_i(u_i)}{\partial u_i} \right| \tag{1.6.7}$$

Esto asume que u_i es una variable independiente con una distribución de la forma de la derivada de la no linealidad. En este caso la función empleada es la función logística (Ver figura 1.6.2).

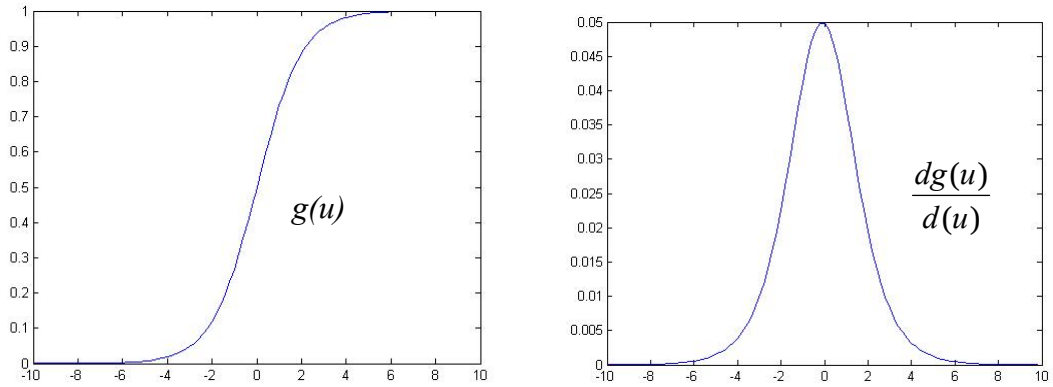


Fig. 1.6.2 Función logística y su derivada.

La función logística (o curva logística, modeliza a una función sigmoidea) es ampliamente utilizada para estimar distribuciones como las de música y señales de voz. Esta función de activación fue propuesta por Bell y Sejnowski, demostrando su convergencia para señales de audio como la voz y música [WON00].

La función no lineal ($g_i(u_i)$) es empleada principalmente para minimizar la información mutua y así obtener salidas independientes. Otra interpretación para el uso de la no-linealidad es la minimización de la correlación de orden superior (por su expansión en series de Taylor).

Como se mencionó anteriormente de la ecuación 1.6.2, la independencia estadística se obtiene cuando la información mutua de la red es cero y la entropía de juntura es igual a la suma de sus entropías marginales, donde la expresión de la entropía marginal se representa como:

$$H(\hat{s}_i) = -E\{\log p(\hat{s}_i)\} \tag{1.6.8}$$

La no linealidad realiza una transformación de la variable u_i a $\hat{s}_i$. Por lo tanto la ecuación 1.6.6 se puede expresar como:

$$p(\hat{s}_i) = \frac{p(u_i)}{\left|\frac{\partial \hat{s}_i}{\partial u_i}\right|} \tag{1.6.9}$$

substituyendo en 1.6.8 se obtiene:

$$H(\hat{s}_i) = -E\{\log(\frac{p(u_i)}{\left|\frac{\partial \hat{s}_i}{\partial u_i}\right|})\} \tag{1.6.10}$$

Rescribiendo en la ecuación 1.6.2.

$$H(\hat{\mathbf{s}}) = H(\hat{s}_1, \hat{s}_2, ...\hat{s}_n) = -E\{\log(\frac{p(u_1)}{\left|\frac{\partial \hat{s}_1}{\partial u_1}\right|}) - .... - E\{\log(\frac{p(u_n)}{\left|\frac{\partial \hat{s}_n}{\partial u_n}\right|}) - I(\hat{s}_1, \hat{s}_2, ...\hat{s}_n) \tag{1.6.11}$$

$$H(\hat{\mathbf{s}}) = -\sum_{i=1}^{n}\{E\{\log(\frac{p(u_i)}{\left|\frac{\partial \hat{s}_i}{\partial u_i}\right|})\} - I(\hat{\mathbf{s}}) \tag{1.6.12}$$

Realizando la derivada de la ecuación anterior respecto a $\boldsymbol{W}$ se obtiene:

$$\frac{\partial H(\hat{\mathbf{s}})}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}\left(-I(\hat{\mathbf{s}})\right) - \frac{\partial}{\partial \mathbf{W}}\sum_{i=1}^{n}\{E\{\log(\frac{p(u_i)}{\left|\frac{\partial \hat{s}_i}{\partial u_i}\right|})\} \tag{1.6.13}$$

En esta ecuación se observa la relación de la entropía de juntura con la información mutua. La minimización directa de la información mutua genera la maximización de la entropía de juntura. Dicho de otra forma, la información mutua será minimizada ($I(\hat{\mathbf{s}}) \cong 0$) cuando la no linealidad $\hat{s}_i = g_i(u_i)$ es aproximada a la función de densidad acumulativa (f.d.a) de la fuente estimada (u_i).

En el caso en el que la fuente estimada y la función no lineal resulten en un valor $I(\hat{\mathbf{s}})$ diferente de cero, existirá un error. Por lo tanto, en la ecuación 1.6.13 se plantea que el proceso de minimización de $I(\hat{\mathbf{s}})$ depende de las fuentes estimadas y de la función no lineal empleada. No obstante, el término de error para las aplicaciones propuestas (empleando una función logística y señales con distribución super-gausiana) se puede considerar como despreciable. En este caso, el término del error desaparece y el máximo de la entropía de juntura se obtiene derivando la entropía $H(\hat{\mathbf{s}})$ con respecto a **W**. Esto es calculando el gradiente de $H(\hat{\mathbf{s}})$.

$$\frac{\partial H(\hat{\mathbf{s}})}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}\left(-E\{\log|\mathbf{J}|\}\right) = \frac{\partial}{\partial \mathbf{W}}\log|\mathbf{J}| \tag{1.6.14}$$

El término $|\mathbf{J}|$ es el valor absoluto del Jacobiano. Donde la relación entre la densidad de salida $p(\hat{\mathbf{s}})$ y la densidad de entrada $p(\mathbf{e})$ puede ser definida como $p(\hat{\mathbf{s}}) = p(\mathbf{e})/|\mathbf{J}(\mathbf{e})|$ [PAP91]. Por lo tanto $|\mathbf{J}|$ puede ser definido como una transformación de $p(\mathbf{e})$ a $p(\hat{\mathbf{s}})$:

$$\mathbf{J}(\mathbf{e}) = \det\begin{bmatrix} \frac{\partial \hat{s}_1}{\partial e_1} & \cdots & \frac{\partial \hat{s}_1}{\partial e_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{s}_n}{\partial e_1} & \cdots & \frac{\partial \hat{s}_n}{\partial e_n} \end{bmatrix} \tag{1.6.15}$$

De donde cada elemento de la matriz tiene la forma:

$$\frac{\partial \hat{s}_i}{\partial e_j} = w_{ij}\frac{\partial \hat{s}_i}{\partial u_j} \tag{1.6.16}$$

Substituyendo en la ecuación 1.6.15 se tiene que:

$$\mathbf{J}(\mathbf{e}) = \det\begin{bmatrix} w_{11}\frac{\partial \hat{s}_1}{\partial u_1} & \cdots & w_{1n}\frac{\partial \hat{s}_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ w_{1n}\frac{\partial \hat{s}_n}{\partial u_1} & \cdots & w_{nn}\frac{\partial \hat{s}_n}{\partial u_n} \end{bmatrix} \tag{1.6.17}$$

Como no existe una conexión entre las salidas después de las neuronas, las derivadas parciales de $\partial \hat{s}_i/\partial u_j$, tomarán un valor diferente de cero únicamente en los casos donde $i = j$. Por lo tanto, puede definirse la ecuación 1.6.15 como:

$$\mathbf{J}(\mathbf{e}) = \det(\mathbf{W})\prod_{i=1}^{n}\left|\frac{\partial \hat{s}_i}{\partial u_i}\right| \tag{1.6.18}$$

Substituyendo la ecuación 1.6.18 en la ecuación 1.6.14

$$\frac{\partial H(\hat{\mathbf{s}})}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}\log\left(|\det(\mathbf{W})|\prod_{i=1}^{n}\left|\frac{\partial \hat{s}_i}{\partial u_i}\right|\right)$$

$$= \frac{\partial}{\partial \mathbf{W}} \log|\det(\mathbf{W})| + \sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{W}} \log\left|\frac{\partial \hat{s}_i}{\partial u_i}\right| \tag{1.6.19}$$

El primer término de la ecuación se expresa como:

$$\frac{\partial}{\partial \mathbf{W}} \log|\det(\mathbf{W})| = \frac{(adj\mathbf{W})^T}{\det \mathbf{W}} = \left(\mathbf{W}^T\right)^{-1} \tag{1.6.20}$$

debido a que el determinante de **W** puede ser obtenido de:

$$\det(\mathbf{W}) = \sum_{j=1}^{n} w_{ij} cof(w_{ij}) \tag{1.6.21}$$

El segundo término de la ecuación 1.6.19 puede ser determinado como:

$$\frac{\partial}{\partial w_{ij}} \sum_{i=1}^{n} \log\left|\frac{\partial \hat{s}_i}{\partial u_i}\right| = \frac{1}{\frac{\partial \hat{s}_i}{\partial u_i}} \frac{\partial^2 \hat{s}_i}{\partial u_i^2} e_j^T \tag{1.6.22}$$

Definiendo la derivada de la no linealidad $\hat{s}_i$ con respecto a u_i como una aproximación de la densidad de la fuente $p(u_i)$, como sigue:

$$p(u_i) = \frac{\partial \hat{s}_i}{\partial u_i} \tag{1.6.23}$$

entonces, de las ecuaciones 1.6.22 y 1.6.23, se obtiene que:

$$\frac{\partial}{\partial \mathbf{W}} \sum_{i=1}^{n} \log\left|\frac{\partial \hat{s}_i}{\partial u_i}\right| = \frac{\frac{\partial p(\mathbf{u})}{\partial \mathbf{u}}}{p(\mathbf{u})} \mathbf{e}^T \tag{1.6.24}$$

Substituyendo los dos términos obtenidos, en la ecuación 1.6.19 se tiene que:

$$\frac{\partial H(\hat{\mathbf{s}})}{\partial \mathbf{W}} = (\mathbf{W}^T)^{-1} + \left(\frac{\frac{\partial p(\mathbf{u})}{\partial \mathbf{u}}}{p(\mathbf{u})}\right) \mathbf{e}^T \tag{1.6.25}$$

Esta ecuación es resultado del gradiente de la función de la entropía. Una forma de maximizar la entropía es por medio del gradiente "natural", el cual se logra multiplicando por $\mathbf{W}^T\mathbf{W}$. Por lo tanto, la ecuación 1.6.25 se expresa como:

$$\Delta \mathbf{W} \propto \frac{\partial H(\hat{\mathbf{s}})}{\partial \mathbf{W}} \mathbf{W}^T \mathbf{W} = \left[\mathbf{I} + \left(\frac{\frac{\partial p(\mathbf{u})}{\partial \mathbf{u}}}{p(\mathbf{u})}\right) \mathbf{u}^T\right] \mathbf{W} \tag{1.6.26}$$

Donde **I** es la matriz identidad y el segundo factor denota la no linealidad.

$$\varphi(\mathbf{u}) = -\frac{\frac{\partial p(\mathbf{u})}{\partial \mathbf{u}}}{p(\mathbf{u})} \tag{1.6.27}$$

la ecuación 1.6.26 resultante es:

$$\Delta \mathbf{W} \propto \left[\mathbf{I} - \varphi(\mathbf{u})\mathbf{u}^T\right]\mathbf{W} \tag{1.6.28}$$

En donde el término $\varphi(\mathbf{u})$ es la función no lineal y es determinante para la separación de fuentes sub y súper-gaussianas. Algunas funciones no lineales fueron propuestas por Bell y Sejnowski [BEL95]. La ecuación 1.6.28 se conoce con el nombre de *regla de aprendizaje* y se emplea en un modelo neuronal recursivo para el ajuste de los pesos sinápticos en donde en cada iteración se realizará una actualización hasta encontrar su valor óptimo. Este modelo neuronal se conoce como la regla de Hebb y se expresa como:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \Delta \mathbf{W} \tag{1.6.29}$$

Una forma alternativa de generar la regla de aprendizaje antes mencionada, se deriva de la estimación de máxima verosimilitud (*"Maximum Likelihood Estimator"* MLE) propuesta por Pearlmutter, Parra y Cardoso [PEA97, CAR97]. Esta consiste en expresar la función de densidad de probabilidad como:

$$p(\mathbf{e}) = \left|\det(\mathbf{W})\right| p(\mathbf{u}) \tag{1.6.30}$$

Donde $p(\mathbf{u}) = (p_1(u_1)) \cdot (p_2(u_2)) \cdot \ldots \cdot (p_n(u_n)) = \prod_{i=1}^{n} p_i(u_i)$ es el producto de las distribuciones de las fuentes estimadas. Retomando el logaritmo de la ecuación 1.6.30 se obtiene la representación del logaritmo de la verosimilitud:

$$L(\mathbf{u}, \mathbf{W}) = \log\left|\det(\mathbf{W})\right| + \sum_{i=1}^{n} \log p_i(u_i) \tag{1.6.31}$$

Retomando la ecuación 1.6.28 y maximizándola (derivándola respecto a **W**). Se obtiene el siguiente algoritmo para **W**.

$$\Delta \mathbf{W} \propto [(\mathbf{W}^T)^{-1} - \varphi(\mathbf{u})\mathbf{e}^T] \tag{1.6.32}$$

Una forma de maximizar el logaritmo de la verosimilitud es por medio del gradiente natural, el cual consiste en efectuar el producto de $\mathbf{W}^{\mathrm{T}}\mathbf{W}$ simplificando los cálculos para obtención de $(\mathbf{W}^{T})^{-1}$, resultando en:

$$\Delta\mathbf{W} \propto [\mathbf{I} - \varphi(\mathbf{u})\mathbf{u}^{T}]\mathbf{W} \tag{1.6.33}$$

La función no lineal empleada, tiene un papel importante en la separación de las fuentes. Como se mencionó anteriormente, la función no lineal propuesta fue una función logística ($g_i(u_i) = \tanh(u_i)$), así la regla de aprendizaje se rescribe como:

$$\Delta\mathbf{W} \propto [\mathbf{I} - \tanh(\mathbf{u})\mathbf{u}^{T}]\mathbf{W} \tag{1.6.34}$$

A este algoritmo de aprendizaje se le conoce como INFOMAX original, ya que solo puede separar señales con distribuciones súper gaussianas. Una extensión del algoritmo INFOMAX fue desarrollado por Girolami, el algoritmo es capaz de separar señales con distribuciones sub y súper-gaussianas. Esta técnica considera el modelado de mezcla de Pearson, el cual se expresa como:

$$p(u) = \frac{1}{2}\left(N(\mu,\sigma^2) + N(-\mu,\sigma^2)\right) \tag{1.6.35}$$

Donde $N(\mu,\sigma^2)$ es la densidad de una distribución normal con media μ y varianza σ^2. Se asumirá que se tiene para ambas funciones una varianza = 1 y una media en un intervalo de [0,…,2]. La distribución de *p(u)* que se obtendrá para un valor de μ=0 es una distribución gaussiana y para cualquier valor mayor de cero (μ>0) presentará una distribución sub-gaussiana.

Definiendo $a = \mu/\sigma^2$ y substituyendo en la ecuación 1.6.27 la distribución de una función normal se obtiene que:

$$\varphi(\mathbf{u}) = -\frac{\frac{\partial p(\mathbf{u})}{\partial \mathbf{u}}}{p(\mathbf{u})} = \frac{u}{\sigma^2} - a\left(\frac{\exp(au) - \exp(-au)}{\exp(au) + \exp(-au)}\right) \tag{1.6.36}$$

Usando la definición de la tangente hiperbólica y considerando que se tiene σ^2 =1, la función se reduce en:

$$\varphi(u) = u - \tanh(u) \tag{1.6.37}$$

Substituyendo 1.6.37 en 1.6.33 se obtiene la regla de aprendizaje para una función estrictamente sub-gaussiana

$$\Delta\mathbf{W} \propto [\mathbf{I} + \tanh(\mathbf{u})\mathbf{u}^T - \mathbf{u}\mathbf{u}^T]\mathbf{W} \tag{1.6.38}$$

En el caso de una fuente con distribución súper-gausiana se obtiene que:

$$\varphi(u) = u + \tanh(u) \tag{1.6.39}$$

Por lo tanto la regla de aprendizaje para fuentes súper-gaussianas será:

$$\Delta\mathbf{W} \propto [\mathbf{I} - \tanh(\mathbf{u})\mathbf{u}^T - \mathbf{u}\mathbf{u}^T]\mathbf{W} \tag{1.6.40}$$

La ecuación 1.6.40 muestra el caso general para distribuciones sub y súper gaussianas:

$$\Delta\mathbf{W} \propto [\mathbf{I} - \mathbf{K}\tanh(\mathbf{u})\mathbf{u}^T - \mathbf{u}\mathbf{u}^T]\mathbf{W} \tag{1.6.41}$$

En donde **K** representa una matriz con elementos sólo en su diagonal principal, los cuales efectúan una conmutación de signo para cada una de las distribuciones [WON99].

k_i = 1 : distribución súper-gaussiana.

k_i = -1 : distribución sub-gaussiana.

Girolami empleó el signo de la curtosis como criterio para definir la conmutación. Sin embargo, un análisis más detallado sobre la estabilidad para la conmutación fue determinado por Pham, Garrat y Amari [WON00]. El análisis de la estabilidad considera un promedio del gradiente en el punto óptimo. Una condición suficiente que garantice la estabilidad es:

$$k_i > 0 \qquad 1 \le i \le N \tag{1.6.42}$$

Donde k_i es:

$$k_i = E\{\varphi_i'(u_i)\}E\{u_i^2\} - E\{\varphi_i'(u_i)u_i\} \tag{1.6.43}$$

y

$$\varphi_i(u_i) = u_i + k_i \tanh(u_i) \tag{1.6.44}$$

Substituyendo 1.6.44 en 1.6.43, resulta en:

$$k_i = E\{k_i \sec h^2(u_i) + 1\}E\{u_i^2\} - E\{[k_i \tanh(u_i) + (u_i)](u_i)\}$$

$$= k_i(E\{\sec h^2(u_i)\}E\{u_i^2\} - E\{[\tanh(u_i)]u_i\} \tag{1.6.45}$$

Para asegurar que $k_i > 0$, el signo de k_i debe ser el mismo como el de $E\{\sec h^2(u_i)\}E\{u_i^2\} - E\{[\tanh(u_i)]u_i\}$. Por lo tanto, la regla de aprendizaje para la conmutación de la ecuación 1.6.41 es:

$$k_i = sign\left(E\{\sec h^2(u_i)\}E\{u_i^2\} - E\{[\tanh(u_i)](u_i)\}\right) \tag{1.6.46}$$

Finalmente se substituye la ecuación 1.6.46 en la ecuación 1.6.41, donde k_i es el i-esimo elemento.

1.7 ICA por maximización de la no-gaussianidad.

En este trabajo de tesis, al igual que en la mayoría de los trabajos desarrollados para el Análisis de Componentes Independientes, se divide el problema en dos etapas. En la primera se realiza una transformación **V** de las señales de entrada **e***(t)*, de forma que resultan en la señal $\mathbf{z}(t)$, con media cero, con varianza unitaria y descorrelacionadas, como se indica a continuación:

$$\mathbf{z}(t) = \mathbf{V}\mathbf{e}(t) \tag{1.7.1}$$

A este tratamiento de la señal, se le conoce como "*blanqueado espacial"* y simplifica el problema de ICA. En la figura 1.7.1 se muestran las etapas de ICA empleando este método.

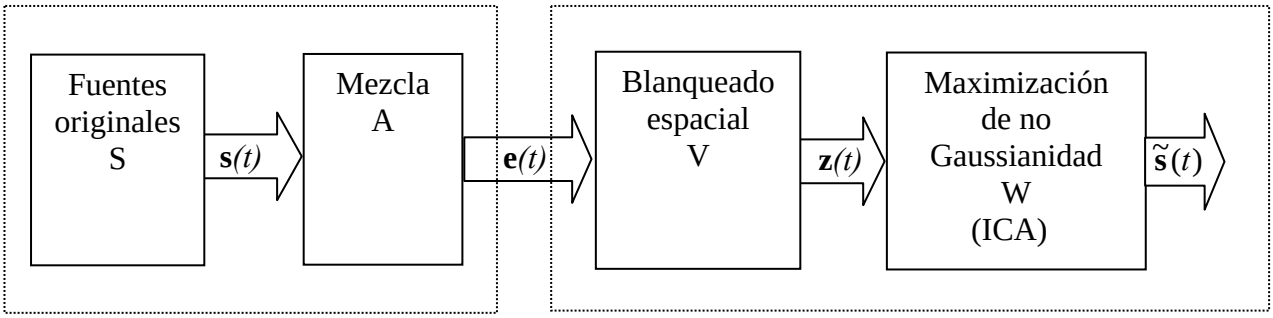


Fig. 1.7.1. Etapa del blanqueado y el Análisis de Componentes Independientes.

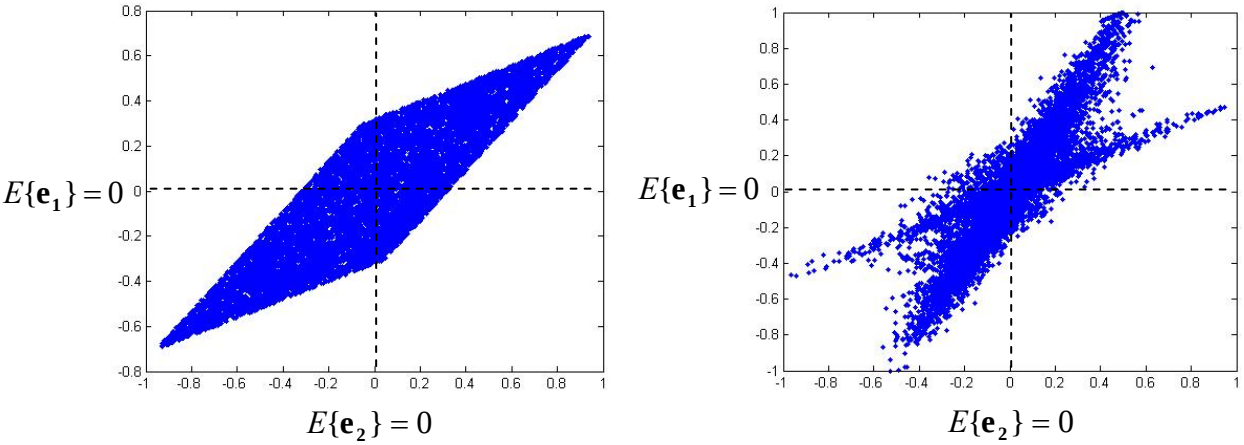
Con esta transformación las señales se descorrelacionan, pero dicha condición es necesaria, aunque no suficiente para la independencia estadística, por lo tanto; se realizará una transformación adicional que maximice la no-gaussianidad para obtener la separación, la cual se verá más adelante. A continuación, se muestra el proceso de blanqueado de las señales.

1.7.1 Blanqueado espacial (*V*).

El primer paso a realizar en el blanqueado, es hacer que los vectores de entrada **e***(t)* tengan media cero (a esto también se le conoce como el centrado de las variables). Esto se logra restando su media, es decir:

$$\mathbf{e}(t + \Delta t) \leftarrow \mathbf{e}(t) - E\{\mathbf{e}(t)\} \tag{1.7.2}$$

El símbolo "$\leftarrow$" es un término computacional que indica que se hará una substitución del valor actual al valor siguiente. De esta forma los datos son normalizados con respecto a la estadística de primer orden (la media). En la figura 1.7.2 se muestran dos distribuciones de datos con media cero.



a) Variables aleatorias uniformes b) Señales de audio

Fig. 1.7.2. Distribución con media cero de dos variables.

Las componentes de los vectores blanqueados **z***(t)* resultantes tienen la característica de estar descorreacionadas y con su varianza normalizadas [HYV03:140]. Por lo tanto se tiene que:

$$E\left\{\mathbf{z}\cdot\mathbf{z}^T\right\}=\mathbf{I} \tag{1.7.3}$$

La medida de la independencia lineal de los vectores se denominada *correlación*. Por lo tanto la descorrelación o correlación cero, indica que se tiene una independencia lineal entre vectores y, como se mencionó, es una condición necesaria pero no suficiente para la independencia estadística. Esto es, dos variables y_1 y y_2 se dice que están descorrelacionadas si su covarianza es cero.

$$\text{cov}(y_1 y_2)=E\left\{y_1 {y_2}^T\right\}-E\{y_1\}E\left\{{y_2}^T\right\}=0 \tag{1.7.4}$$

Asumiendo que las variables tienen media de cero, su covarianza es igual a su correlación $corr(y_1 y_2) = E\{y_1 y_2\}$.

Como se muestra en la ecuación 1.7.1, la descorrelación de los datos de entrada **e***(t)* esta dada por la transformación de la matriz **V**. Para encontrar esta matriz, cuyos vectores son linealmente independientes, se parte de la matriz de covarianza $\mathbf{C}_e = E\{\mathbf{e}\cdot\mathbf{e}^{\mathbf{T}}\}$, de los vectores de entrada **e***(t)*. Posteriormente, la matriz **V** es estimada a partir de los valores y los vectores propios (normalizados) de la matriz de covarianza ($\mathbf{C}_e$). Los vectores propios ($\mathbf{E}$) proporcionan la dirección de las componentes, mientras que los valores propios ($\mathbf{D}$) proporcionan la amplitud de las distribuciones (varianza). Por lo tanto la transformación **V** está dada por:

$$\mathbf{V} = \mathbf{D}^{-1/2}\mathbf{E}^T \tag{1.7.5}$$

Esta matriz existe siempre y cuando los valores propios sean positivos. Escribiendo la correlación en términos de sus vectores y valores propios, se denota como:

$$\mathbf{R}_e = \mathbf{C}_e = \mathbf{E}\mathbf{D}\mathbf{E}^T \tag{1.7.6}$$

Donde **E** representa los vectores propios, cuya característica es la de ortogonalidad, cuyos vectores satisfacen la siguiente condición:

$$\mathbf{E}^T\mathbf{E} = \mathbf{E}\mathbf{E}^T = \mathbf{I} \tag{1.7.7}$$

Esto demuestra que:

$$E\{\mathbf{z}\cdot\mathbf{z}^T\} = \mathbf{V}E\{\mathbf{e}\cdot\mathbf{e}^T\}\mathbf{V}^T = \mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{E}\mathbf{D}\mathbf{E}^T\mathbf{E}\mathbf{D}^{-1/2} = \mathbf{I} \tag{1.7.8}$$

Donde **D** es una matriz de los valores propios (**D**=*diag*($d_1,\ldots,d_n$)), que representan la varianza. Considerando la normalización de las varianzas, se demuestra que la covarianza de **z** es una matriz unitaria, tomando el término de *blanqueado*.

De acuerdo con las expresiones anteriores se puede deducir que:

a) Los vectores propios de la matriz de correlación, **R** correspondiente a los vectores de entrada **e***(t)*, de media cero, definen vectores unitarios representando las direcciones principales a lo largo de las cuales las varianzas toman sus valores máximos.
b) Los valores propios asociados, definen los valores máximos de esas varianzas.

c) Proporciona datos decorrelacionados.

El operador lineal **V** puede ser obtenido de diversas formas, una de ellas es considerando una matriz **U** cuyos vectores sean ortogonales, sin que con esto afecte la descorrelación. Esto es, si $\mathbf{z} = \mathbf{UVe}$, entonces se obtiene que:

$$E\{\mathbf{z}\cdot\mathbf{z}^T\} = \mathbf{UV}E\{\mathbf{e}\cdot\mathbf{e}^T\}\mathbf{V}^T\mathbf{U}^T = \mathbf{UIU}^T = \mathbf{I} \tag{1.7.9}$$

Otra posible técnica para efectuar el blanqueado, es utilizando los métodos que se basan en la ejecución de una regla de aprendizaje en línea del análisis de componentes principales (PCA) [HYV03][SAB91][JOL86].

$$\Delta\mathbf{V} = \gamma\left(\mathbf{I} - \mathbf{Ve}\cdot\mathbf{e}^T\mathbf{V}^T\right)\mathbf{V} = \gamma(\mathbf{I} - \mathbf{z}\cdot\mathbf{z}^T)\mathbf{V} \tag{1.7.10}$$

Donde $\Delta\mathbf{V}$ indica el incremento en cada iteración y γ indica una razón de aprendizaje. Este método se basa en modelos neuronales (regla de Hebb [HAG96]), en donde cada iteración efectuará un ajuste de **V**. Esta expresión es:

$$\mathbf{V} \leftarrow \mathbf{V} + \Delta\mathbf{V} \tag{1.7.11}$$

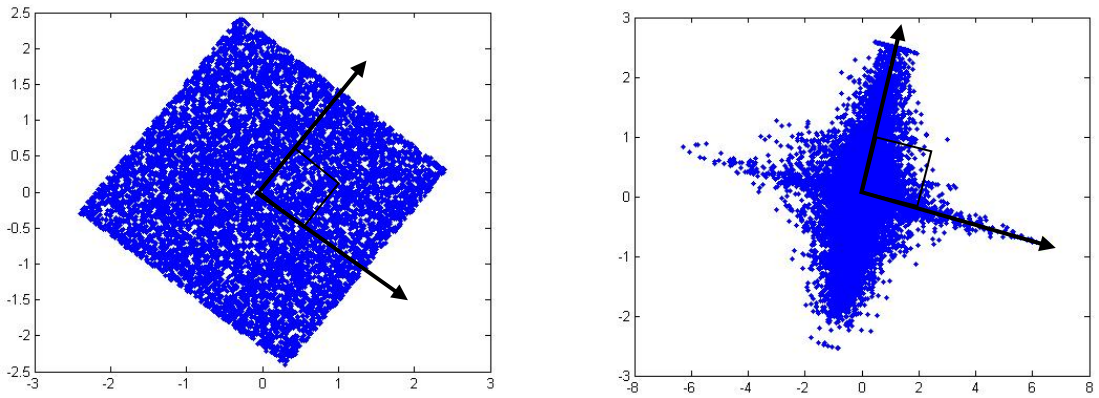
En la expresión 1.7.9 se puede observar que el término $\mathbf{Ve}\cdot\mathbf{e}^T\mathbf{V}^T$, que es igual a $\mathbf{z}\cdot\mathbf{z}^T$, efectúa la medida de la correlación de **z**. Como se mostró en la ecuación (1.7.7), el vector blanqueado es aquel en el que su correlación sea igual a la matriz identidad, en este caso $\Delta\mathbf{V} = \gamma(\mathbf{I} - \mathbf{z}\cdot\mathbf{z}^T)\mathbf{V}$ será igual a cero, y la transformación lineal **V** converge en una matriz de blanqueado.

Este algoritmo es muy sencillo de implementar pero a diferencia del 1.7.4, no pueden comprimir datos y a veces presenta problemas de estabilidad [OJA97]. Por último, Bell y Sejnowski [BEL95] proponen la siguiente matriz de blanqueo en su procedimiento de separación de fuentes basado en la maximización de la entropía:

$$\mathbf{V} = 2\sqrt{E^{-1}\left\{\mathbf{e}\cdot\mathbf{e}^{T}\right\}} \tag{1.7.12}$$

con la que se obtienen resultados adecuados en las aplicaciones que consideran.

En la figura 1.7.3 se muestran en planos bidimensionales dos señales descorrelacionadas, cuyos vectores principales se encuentran en la dirección de máxima varianza y muestran ortogonalidad.



a) Variables aleatorias uniformes b) Señales de audio

Fig. 1.7.3 distribución descorrelacionada de dos variables.

En el capítulo 2 se muestran algunos resultados con Matlab empleando los métodos antes mencionados.

1.7.2 Maximización de la no-gaussianidad.

La siguiente etapa después del blanqueado es la maximización de la no-gaussianidad [HYV99] [HYV03:160]. El modelo de mezcla lineal que fue mencionado al inicio y se expresa como **e** = **As**, en donde **e** es un vector de las señales de entrada, **A** es la matriz de mezcla y **s** son las fuentes originales. El teorema central del límite (ver apéndice A, sección A10) indica que, bajo condiciones muy generales, la distribución de la suma de variables aleatorias tiende a una Distribución Normal (también llamada distribución gaussiana) cuando la cantidad de variables(datos) es muy grande. Es decir, que la mezcla lineal de las fuentes (**e**) presentará una

distribución más cercana a una gaussiana, con respecto a las fuentes originales (s), de ahí la necesidad de determinar la no-gaussianidad.

La solución busca encontrar una trasformación que invierta a la matriz de mezcla, estimando cada una de las fuentes independientes **s**, esto es, $\hat{\mathbf{s}} = \mathbf{A}^{-1}\mathbf{e}$. Si se propone que $y = \mathbf{b}^{\mathrm{T}}\mathbf{e}$, resulta en $y = \mathbf{b}^{\mathrm{T}}\mathbf{As}$, donde **b** será un vector a ser determinado. Por lo tanto, se tendrá que y es una cierta combinación lineal de **s** dados los coeficientes de $\mathbf{b}^{\mathrm{T}}\mathbf{A}$. Expresando el producto de $\mathbf{b}^{\mathrm{T}}\mathbf{A}$ con el vector **q**, se obtiene que:

$$y = \mathbf{b}^{T}\mathbf{e} = \mathbf{q}^{T}\mathbf{s} = \sum_{i} q_i s_i \tag{1.7.13}$$

En el caso de que **b** sea una de las filas de la inversa de **A**, se obtendrá una de las componentes independientes. Por lo tanto, el vector **q** tendrá uno de sus valores igual a 1 y el resto a cero obteniendo solo a $\mathbf{s}_i$.

Ahora, el objetivo es encontrar un vector **b** que maximice la no-gaussianidad de y por medio del producto $\mathbf{b}^{\mathrm{T}}\ \mathbf{e}$. Tal vector corresponde a $\mathbf{q} = \mathbf{b}^{\mathrm{T}}\mathbf{A}$, para la estimación de las componentes independientes. Como se mencionó anteriormente, los datos tienen un procesamiento de blanqueado el cual se expresa como:

$$\mathbf{z} = \mathbf{Ve} = \mathbf{VAs} \tag{1.7.14}$$

Por lo tanto, la correlación de **z** será igual a la matriz identidad y las componentes de las variables serán ortogonales. En tal caso de encontrar un máximo (máxima no-gaussianidad), se podrá efectuar una rotación para acoplar cada una de las componentes sobre los ejes coordenados, obteniendo la característica de independencia estadística.

En este trabajo se mencionan dos métodos existentes. El primero que será abordado, mide la no-gaussianidad a través de su cumulante de cuarto orden conocido también como *Curtosis*. El siguiente método mide la no-gaussianidad a través de la entropía relativa referente a la entropía de una distribución gaussiana, y es conocida como *Negentropía*.

1.7.3 Medida de la no gaussianidad por curtosis.

La curtosis, como se mencionó anteriormente, es el cumulante de cuarto orden (ver Apéndice A) y mide la agudeza de una distribución tipo gaussiana. La curtosis de una variable (y) se denota como *Kurt(y)* y se expresa como:

$$Kurt(y) = E\{y^4\} - 3(E\{y^2\})^2 \tag{1.7.15}$$

Considerando que la variable (y) tiene varianza normalizada y media cero, como resultado del proceso de blanqueado, entonces la expresión 1.7.15, resulta en:

$$Kurt(y) = E\{y^4\} - 3 \tag{1.7.16}$$

Esto muestra que la curtosis es una versión normalizada del momento de cuarto orden $E\{y^4\}$. Para una distribución gaussiana, su momento de cuarto orden $E\{y^4\}$ es $3(E\{y^2\})^2$, por lo tanto la curtosis para este caso es cero. La curtosis puede tomar valores positivos o negativos dependiendo del tipo de distribución empleada. Para el caso de distribuciones súper-gaussianas (también llamadas leptocúrticas) tomará valores positivos y para el caso de distribuciones sub-gaussianas (también llamadas platicúrtica) tomará valores negativos.

Comúnmente, el caso general de no-gaussianidad se mide con el cuadrado de la curtosis. Sin embargo, también es utilizado su valor absoluto, siendo este último el más utilizado en ICA por su simplicidad computacional. Algunas propiedades importantes de la curtosis son:

$$Kurt(x_1 + x_2) = Kurt(x_1) + Kurt(x_2) \tag{1.7.17}$$

$$Kurt(\alpha x) = \alpha^4 Kurt(x) \tag{1.7.18}$$

Donde α es una constante.

Considerando un modelo de mezcla bidimensional formado por s_1 y s_2, cada una de estas variables tendrá su valor respectivo de curtosis diferente de cero (para el caso de fuentes independientes). El modelo de mezcla se representa como $\mathbf{e} = \mathbf{As}$. Por lo tanto, el algoritmo de

optimización busca las componentes independientes, empleando la transformación lineal $y = \mathbf{b}^T\mathbf{e}$.

El vector transformado se representa como $\mathbf{q} = \mathbf{A}^T\mathbf{b}$. Substituyendo cada una de las ecuaciones anteriores se obtiene que $y = \mathbf{b}^T\mathbf{e} = \mathbf{b}^T\mathbf{A}\mathbf{s} = \mathbf{q}^T\mathbf{s} = q_1 s_1 + q_2 s_2$, donde al aplicar la propiedad aditiva de la curtosis se obtienen:

$$Kurt(y) = Kurt(q_1 s_1) + Kurt(q_2 s_2) = q_1^4 Kurt(s_1) + q_2^4 Kurt(s_2) \qquad (1.7.19)$$

El problema es optimizar la curtosis, esto es: $\left|Kurt(y)\right| = \left|q_1^4 Kurt(s_1) + q_2^4 Kurt(s_2)\right|$. Al suponer las curtosis = 1, la ecuación 1.7.19 se simplificará en:

$$F(\mathbf{q}) = q_1^4 + q_2^4 \qquad (1.7.20)$$

Considerando que la varianza de las funciones es igual a la unidad, esto implica una restricción en $\mathbf{q}$: $E\{y^2\} = q_1^2 + q_2^2 = 1$. Esto restringe a la ecuación en un plano bidimensional a un círculo unitario. La figura 1.7.4 muestra la trayectoria del vector **q** en los cuatro cuadrantes y se observa que su valor óptimo se encuentra en los puntos donde uno de los elementos de **q** es cero, el cual ocurre sobre los ejes coordenados. En ese momento las componentes serán independientes.

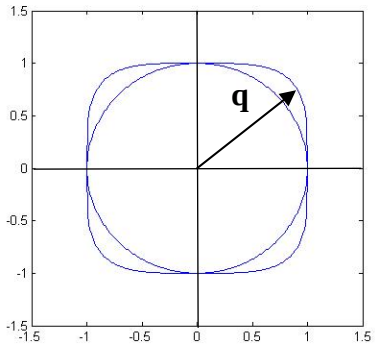


Fig. 1.7.4 Trayectoria del vector **q** considerando una curtosis igual a 1

El vector **e** se encuentra formado por la transformación de la matriz de mezcla de los datos de las fuentes originales, esto es **e** = **As**. Y el blanqueado efectúa la transformación **z** = **Ve** = **VAs**. Por lo tanto, la expresión $\mathbf{q} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$ puede escribirse como $\mathbf{q} = (\mathbf{VA})^{\mathrm{T}}\,\mathbf{w}$, del cual se obtiene que:

$$\|\mathbf{q}\|^2 = (\mathbf{w}^T\mathbf{VA})(\mathbf{A}^T\mathbf{V}^T\mathbf{w}) = \|\mathbf{w}\|^2 \tag{1.7.21}$$

en el que se observa que **w**, que es el vector a ser determinado, estará restringido a un círculo unitario, maximizando la curtosis en la dirección de las componentes independientes.

1.7.3.A.- Algoritmo del gradiente de la curtosis

Este método busca maximizar el valor absoluto de la curtosis, el cual consiste en derivar a la curtosis respecto a **w**.

$$\frac{\partial\left|kur(\mathbf{w}^T\mathbf{z})\right|}{\partial\mathbf{w}} = 4sign(kurt(\mathbf{w}^T\mathbf{z}))\left[E\left\{z(\mathbf{w}^T\mathbf{z})^3\right\} - 3\mathbf{w}\|\mathbf{w}\|^2\right] \tag{1.7.22}$$

Obteniendo la dirección del vector **w** en donde el valor absoluto de la curtosis de $y = \mathbf{w}^T\mathbf{z}$ es máximo, la proyección de **w** se efectuará dentro de un círculo unitario en donde en cada iteración efectuará una normalización de **w**. Por lo tanto, el algoritmo del gradiente se expresa como:

$$\Delta\mathbf{w}\,\alpha\, sign(kurt(\mathbf{w}^T\mathbf{z}))E\left\{\mathbf{z}(\mathbf{w}^T\mathbf{z})^3\right\} \tag{1.7.23}$$

$$\mathbf{w} \leftarrow \mathbf{w}/\|\mathbf{w}\| \tag{1.7.24}$$

Una versión en línea de este algoritmo se obtiene omitiendo el valor esperado de la ecuación 1.7.23 resultando en:

$$\Delta\mathbf{w}\,\alpha\, sign(kurt(\mathbf{w}^T\mathbf{z}))\mathbf{z}(\mathbf{w}^T\mathbf{z})^3 \tag{1.7.25}$$

$$\mathbf{w} \leftarrow \mathbf{w}/\|\mathbf{w}\| \tag{1.7.26}$$

También la curtosis puede ser estimada en línea utilizando la siguiente expresión iterativa:

$$\Delta\gamma \propto \left((\mathbf{w}^T\mathbf{z})^4 - 3\right) - \gamma \tag{1.7.27}$$

Denotando a γ como el estimado de la curtosis, que puede ser evaluado en línea.

1.7.3.B.- Algoritmo de punto fijo utilizando la curtosis

El algoritmo de punto fijo es una versión mejorada del gradiente y consiste en realizar la iteración en la dirección del gradiente, multiplicada por un escalar para encontrar el valor óptimo de **w**, resultando en una versión más eficiente, alcanzando una convergencia más rápida y más estable, como lo indica la siguiente expresión:

$$\mathbf{w} \propto \left[E\left\{ \mathbf{z}(\mathbf{w}^T\mathbf{z})^3 \right\} - 3\|\mathbf{w}\|^2 \mathbf{w} \right] \tag{1.7.28}$$

El algoritmo de punto fijo calcula el nuevo valor de **w** en cada iteración asignado un nuevo valor, de la siguiente manera:

$$\mathbf{w} \leftarrow E\left\{ \mathbf{z}(\mathbf{w}^T\mathbf{z})^3 \right\} - 3\mathbf{w} \tag{1.7.29}$$

En cada iteración y actualización de **w** se efectuará la división entre su norma conservando su magnitud. El vector resultante, **w** ,da la dirección de una de las componentes independientes de la señal blanqueada. Esto es, cuando la red converja, el vector actual y el nuevo apuntará en la misma dirección (**w** ó **-w**) y por lo tanto la actualización será la misma. Este algoritmo también es conocido como FastICA. El algoritmo FastICA tiene la ventaja sobre el del gradiente, en que su comportamiento es cúbico, lo cual hace su convergencia más rápida.

1.7.4 Medida de la no-gaussianidad por negentropía.

Una de las principales ventajas de la medida de no-gaussianidad por curtosis, es su simplicidad computacional. No obstante, este método tiene la desventaja de ser muy sensible a los valores aislados conocidos como *outliers*. Otra forma de determinar la no-gaussianidad es empleando la negentropía. La negentropía se deriva del término entropía, y se basa en la teoría de la información para obtener la medida o grado de incertidumbre que existe sobre un conjunto de datos, el cual esta dado por:

$$H(y) = -\int p_y(n)\log(p_y(n))dn \tag{1.7.30}$$

Un concepto de teoría de la información menciona que la máxima entropía se alcanza por una variable, cuya distribución sea la de una gaussiana (distribución normal), indicando que se tiene la máxima aleatoriedad. Por lo tanto, cualquier otra variable con una distribución diferente obtendrá un valor menor que el de una función tipo gaussiana.

La medida de la no-gaussianidad se obtiene aplicando la entropía diferencial entre una distribución gaussiana y la distribución de salida, es decir:

$$J(\hat{\mathbf{s}}) = H(s_{gauss}) - H(\hat{\mathbf{s}}) \tag{1.7.31}$$

A esta ecuación se le conoce como negentropía, y es no negativa o cero para una distribución gaussiana.

Un método para el cálculo de la negentropía es utilizando cumulantes de orden superior, el cual se expresa como:

$$J(\hat{s}) \approx \frac{1}{12}E\{\hat{s}^3\}^2 + \frac{1}{48}kurt(\hat{s})^2 \tag{1.7.32}$$

de donde se asume que la variable $\hat{s}$ tiene media cero y varianza unitaria. Por lo tanto, el término $E\{\hat{s}^3\}$ es igual a cero para distribuciones simétricas y la medida de la no-gaussianidad se determina por el cuadrado de la curtosis. La maximización del cuadrado de la curtosis será equivalente a la maximización de su valor absoluto.

Una aproximación útil es generalizar los cumulantes de orden superior mediante el valor esperado de la función no cuadrática general, reemplazando las funciones de $\hat{s}^3$ y $\hat{s}^4$ por otras funciones G^i donde el término *i* indica un índice, resultando en un método aproximado de la negentropía basado en las esperanzas $E\{G^i(\hat{s})\}$, reemplazando a las funciones impares por G^1 y las funciones pares por G^2. Obteniendo la siguiente aproximación:

$$J(\hat{s}) \approx k_1\left(E\left\{G^1(\hat{s})\right\}\right)^2 + k_2\left(E\left\{G^2(\hat{s})\right\} - E\left\{G^2(v)\right\}\right)^2 \tag{1.7.33}$$

donde k_1 y k_2 son constantes positivas y v es una variable con distribución gaussiana de media cero y varianza unitaria. Una aproximación obteniendo el cuadrado de la función es:

$$J(\hat{s}) \propto \left[E\left\{G(\hat{s})\right\} - E\left\{G(v)\right\}\right]^2 \tag{1.7.34}$$

Donde $G(\hat{s}) = \hat{s}^4$, es una aproximación de la curtosis basada en los momentos. Dos de las funciones de G que han demostraron rapidez y estabilidad son:

$$G_1(\hat{s}) = \frac{1}{a_1}\log\cosh a_1\hat{s}, \tag{1.7.35}$$

$$G_2(\hat{s}) = -\exp(-\hat{s}^2/2) \tag{1.7.36}$$

donde a es una constante entre 1 y 2, considerando frecuentemente el valor de 1. Esta técnica para la medición de la no-gaussianidad resulta más precisa aunque más robusta, respecto a la medición de la no-gaussianidad por curtosis.

Al igual que fue desarrollado para la curtosis, en la negentropía se desarrolló un algoritmo del gradiente y un algoritmo de punto fijo, el cual se menciona a continuación.

1.7.4.A.- Algoritmo del gradiente de la negentropía

El algoritmo del gradiente de la negentropía, al igual que la curtosis, consiste en derivar la aproximación de la curtosis (1.7.34) con respecto a **w**, y considerando la normalización $E\left\{\left(\mathbf{w}^T\mathbf{z}\right)^2\right\} = \|\mathbf{w}\|^2 = 1$, se obtiene el siguiente algoritmo:

$$\Delta\mathbf{w} \propto \gamma E\left\{\mathbf{z}g(\mathbf{w}^T\mathbf{z})\right\} \tag{1.7.37}$$

$$\mathbf{w} \leftarrow \mathbf{w}/\|\mathbf{w}\| \tag{1.7.38}$$

donde γ representa $\gamma = E\left\{G(\mathbf{w}^T\mathbf{z})\right\} - E\left\{G(v)\right\}$ y v es una variable aleatoria gaussiana estandarizada, donde la normalización es necesaria para efectuar la proyección del vector **w** en un círculo unitario manteniendo la varianza de $\mathbf{w}^T$ **z** constante. La función g es la derivada de G

utilizada para la aproximación de la negentropía. El valor esperado es omitido para obtener un algoritmo del gradiente estocástico en línea.

La constante γ da un algoritmo de auto-adaptación que puede ser estimado en línea como se muestra a continuación:

$$\Delta\gamma \propto \left(G(\mathbf{w}^T\mathbf{z}) - E\{G(v)\}\right) - \gamma \tag{1.7.39}$$

Esta constante corresponde al signo de la curtosis en (1.7.22).

Al efectuar la derivada de las funciones (1.7.35) y (1.7.36) resulta en una aproximación robusta de la negentropía. Alternativamente, puede utilizarse la derivada de la cuarta potencia, como en la curtosis lo cual conduce a las siguientes funciones:

$$g_1(\hat{s}) = \tanh(a_1\hat{s}) \tag{1.7.40}$$

$$g_2(\hat{s}) = \hat{s}\exp(-\hat{s}^2/2) \tag{1.7.41}$$

$$g_3(\hat{s}) = \hat{s}^3 \tag{1.7.42}$$

Donde a_1 es una constante entre $1 \le a_1 \le 2$ y frecuentemente toma el valor de 1. Este algoritmo puede ser simplificado. Nótese que la constante γ no cambia el punto estacionario de la regla de aprendizaje. Su signo no afecta su estabilidad. Por lo tanto, se reemplaza la γ por su signo sin afectar esencialmente el comportamiento de la regla de aprendizaje.

1.7.4. B - Algoritmo de punto fijo utilizando la negentropía

Como se mencionó con la curtosis, el método de punto fijo es un método mucho más rápido que el del gradiente, resultando en el algoritmo FastICA [OJA05] que encuentra la dirección del vector unitario **w**, tal proyección maximiza la no-gaussianidad. La no-gaussianidad es medida por la aproximación de la negentropía $J(\mathbf{w}^T\mathbf{z})$.

FastICA está basado en la iteración de punto fijo para encontrar el máximo de la no-gaussianidad de $\mathbf{w}^T\mathbf{z}$. Este algoritmo usa la negentropía, el cual es precedido por la normalización de **w**.

Retomando la ecuación (1.7.37) del algoritmo del gradiente, para la negentropía se propone la siguiente iteración de punto fijo:

$$\mathbf{w} \leftarrow E\{\mathbf{z}g(\mathbf{w}^T\mathbf{z})\} \tag{1.7.43}$$

La expresión (1.7.43) se puede modificar en una versión iterativa como se muestra a continuación:

$$(1+\alpha)\mathbf{w} = E\{\mathbf{z}g(\mathbf{w}^T\mathbf{z})\} + \alpha\mathbf{w} \tag{1.7.44}$$

Donde α es una constante multiplicada por ambos factores de la ecuación. La constante α puede ser determinada por el método de Newton [HYV03:66]. Este método, aplicado al del gradiente, da una convergencia más rápida, en pocas iteraciones. No obstante, requiere de la inversión de la matriz en cada iteración, haciéndolo un método muy robusto.

Otra forma de calcular la aproximación del método de Newton, consiste en un método iterativo para determinar el valor óptimo, resultando en:

$$\frac{\partial F}{\partial \mathbf{w}} = E\{\mathbf{z}\mathbf{z}^T g'(\mathbf{w}^T\mathbf{z})\} + \beta\mathbf{I} \tag{1.7.45}$$

Por lo tanto, se obtiene el siguiente método iterativo de Newton:

$$\mathbf{w} \leftarrow \mathbf{w} - \left[E\{\mathbf{z}g(\mathbf{w}^T\mathbf{z})\} + \beta\mathbf{w}\right] / \left[E\{g'(\mathbf{w}^T\mathbf{z})\} + \beta\right] \tag{1.7.46}$$

Este algoritmo pude ser simplificado efectuando la multiplicación de $\beta + E\{g'(\mathbf{w}^T\mathbf{z})\}$ resultando:

$$\mathbf{w} \leftarrow \mathbf{w} - E\{\mathbf{z}g(\mathbf{w}^T\mathbf{z}) - E\{g'(\mathbf{w}^T\mathbf{z})\}\mathbf{w}\} \tag{1.7.47}$$

Esta es la iteración de punto fijo básica en FastICA.

Las funciones de *g'* pueden ser calculadas como:

$$g'_1(\hat{s}) = a_1(1 - \tanh^2(a_1\hat{s})) \tag{1.7.48}$$

$$g'_2(\hat{s}) = (1 - \hat{s}^2)\exp(-\hat{s}^2/2) \tag{1.7.49}$$

$$g'_3(\hat{s}) = 3\hat{s}^2 \tag{1.7.50}$$

1.8 Conclusiones

En este capítulo se mostraron tres diferentes metodologías para el Análisis de Componentes Independientes (ICA). Históricamente, el primer algoritmo de ICA fue desarrollado por Jutten y Herault [JUT91a], en el cual se emplearon los momentos de orden superior para obtener una descorrelación no lineal, ofreciendo una baja complejidad en su modelo neuronal. Este algoritmo es adecuado para la implementación en hardware. No obstante, presenta algunas desventajas como el número limitado de entradas (de 2 y hasta 6 entradas) y las características específicas de sus señales (como su amplitud normalizada en ambas entradas). La arquitectura de la red crece proporcional al número de entradas de la red, mientras que el cálculo de sus pesos sinápticos crece en una razón de (nxn-n), donde n seria cada neurona o cada una de las entradas a separar, es decir que para un caso de 3 entradas, se emplearían 3 neuronas y 6 pesos. El siguiente algoritmo de ICA que se mostró, fue INFOMAX y fue desarrollado en mayor parte por Tee Won, Bell y Sejnowski [WON00], el cual se basa en el gradiente de la maximización de la información y minimización de la información mutua de sus salidas. Este algoritmo presenta buenas características para la implementación en línea y cuyo modelo teórico, permite un número grande de entradas (por ejemplo 100 entradas), el cual se encuentra limitado en hardware ya sea por el intervalo dinámico del voltaje (diseño analógico) ó el número de bits utilizado (diseño digital) y la capacidad del dispositivo en donde se desea implementar. Su arquitectura, crece en forma proporcional, mientras que el cálculo de sus pesos crece en una razón (nxn).

El último método analizado de ICA se basa en la maximización de la no-gaussianidad, efectuando previamente un proceso de blanqueado de las señales para simplificación del análisis, este preprocesamiento consiste en centrar los datos respecto a su media, normalizar su varianza y descorrelacionarlos. La siguiente fase del método es la maximización de la no gaussianidad, cuyo objetivo fue la de encontrar un vector **w** en la dirección que maximice la no-gaussianidad, mostrando dos técnicas: la primera consiste en determinar el grado de no-gaussianidad por medio de su cumulante de cuarto orden (mejor conocido como curtosis). El segundo método consiste en determinar el grado de no-gaussianidad por negentropía. Este último método (ICA por maximización de la no-gaussianidad), es ideal para la implementación en software ya que efectúa el procesamiento de los datos por lotes (batch). En el capítulo 2 se muestran las pruebas correspondientes a cada uno de los algoritmos mostrados en este capítulo.

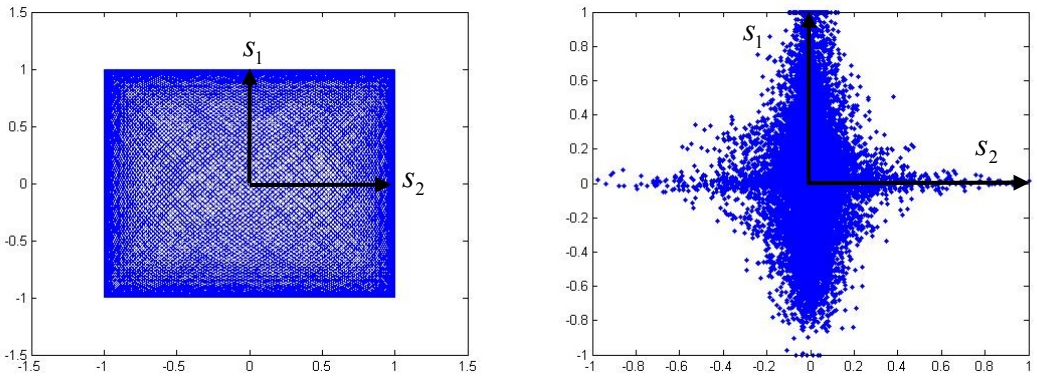
CAPÍTULO 2

Simulaciones con MATLAB de los algoritmos ICA

2.1 Introducción

En esta sección, se analiza y desarrolla la implementación de las redes en el entorno de simulación de MATLAB. También, se presentan los diagramas a bloques de los algoritmos que pueden ser representados por una arquitectura neuronal. Estos algoritmos fueron desarrollados y probados con la herramienta de simulación Simulink, mostrando sus resultados.

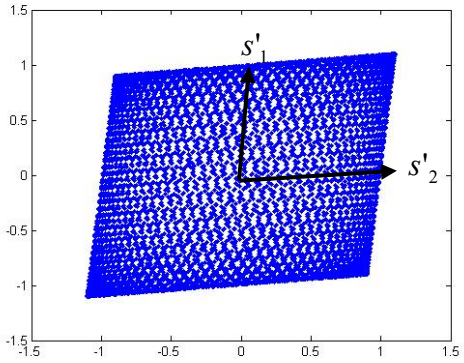
Para verificar el funcionamiento, se utilizaron datos de señales senoidales, de música y voz (con una razón de muestreo de 44100Hz durante 1 segundos). La figura 2.1.1 muestra el plano bidimensional de dichas señales, las cuales cumplen con la condición de independencia estadística.

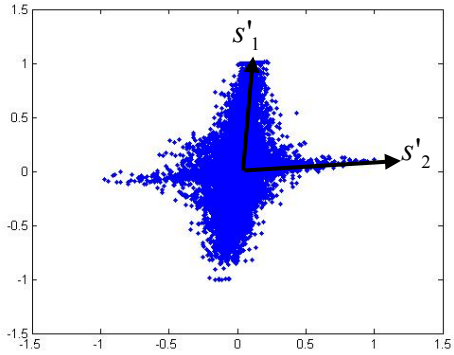


a) Dos señales senoidales b) Señal de música y voz.

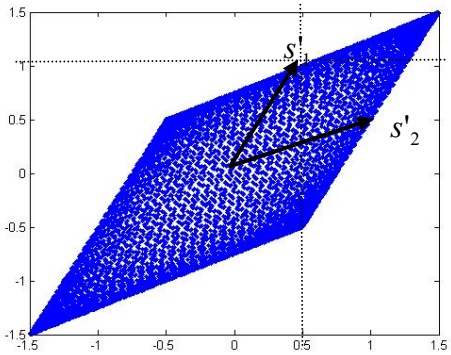
Fig. 2.1.1 Plano bidimensional de dos señales independientes.

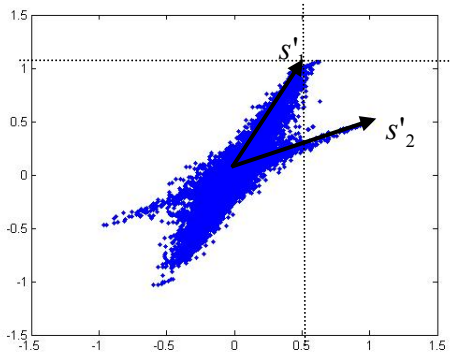
La característica de estas señales, como se mencionó anteriormente, tienen media cero y son independientes (fuentes originales). El primer paso, es efectuar la mezcla de las señales (combinación lineal), que se lleva a cabo en el medio en donde se propagan y son detectadas por los transductores. El modelo de mezcla considera que dichas señales se propagan en un medio lineal y homogéneo, sin retardos de tiempo. La Figura 2.1.2 muestra el plano bidimensional de las dos señales utilizadas y sus respectivas matrices de mezclas.



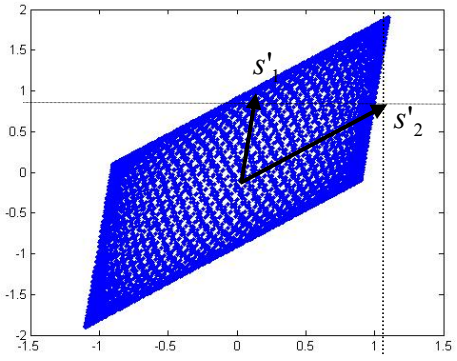


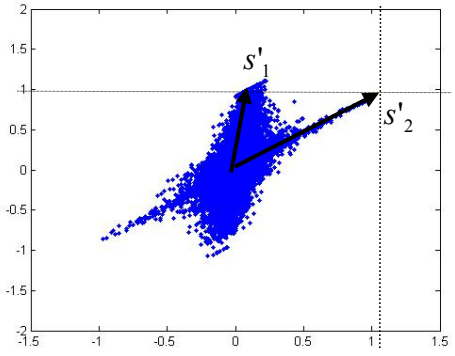
$$\begin{matrix} e_1(t) \\ e_2(t) \end{matrix} = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix}$$



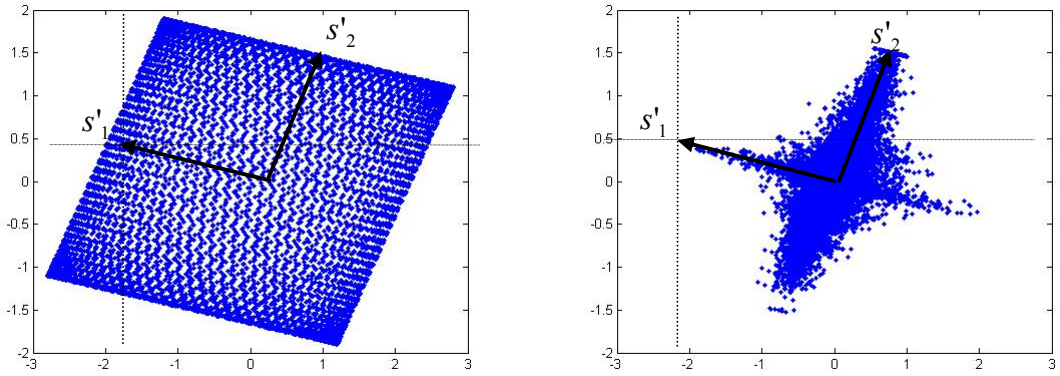


$$\begin{matrix} e_1(t) \\ e_2(t) \end{matrix} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix}$$





$$\begin{matrix} e_1(t) \\ e_2(t) \end{matrix} = \begin{bmatrix} 1 & 0.1 \\ 0.9 & 1 \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix}$$



$$\begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix} = \begin{bmatrix} -2 & 0.8 \\ 0.4 & 1.5 \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix}$$

a) Fuentes senoidales b) fuentes de señales de música

Fig. 2.1.2 Plano bidimensional de dos señales mezcladas.

Así mismo, se denota con s' a los vectores cuya dirección corresponde a la distribución de datos de cada fuente, llamada componente (en la dirección de máxima varianza). El método de ICA, encuentra las componentes y efectúa una transformación lineal con el objeto de independizar los datos de salida (es decir, ubicar los vectores de las componentes sobre los ejes coordenados).

Por otro lado, el objetivo de la red neuronal sólo es efectuar ésta transformación, empleando el algoritmo de ICA como su regla de aprendizaje. A continuación, se muestra la simulación de la primera regla de aprendizaje analizada en el capítulo 1.

2.2 Simulación de ICA por descorrelación no lineal (Red H-J).

En esta sección, se describe la simulación basada en la descorrelación no lineal desarrollada por Jutten y Hérault. Esta se divide en dos partes, La primera trata con la implementación de una red neuronal (2x2) y la segunda, su regla de aprendizaje. La arquitectura de la red se muestra en la Figura 2.2.1

2.2.1 Arquitectura y regla de aprendizaje de la Red Neuronal

Como fue mostrado en el capítulo 1, la ecuación se representa con la siguiente expresión:

$$\hat{s}_i(t) = e_i(t) - \sum_{i=1}^{n} w_{ij}\hat{s}_j(t) \qquad (2.2.1)$$

$$(i, j = 1,2,.....,n) \quad i \neq j$$

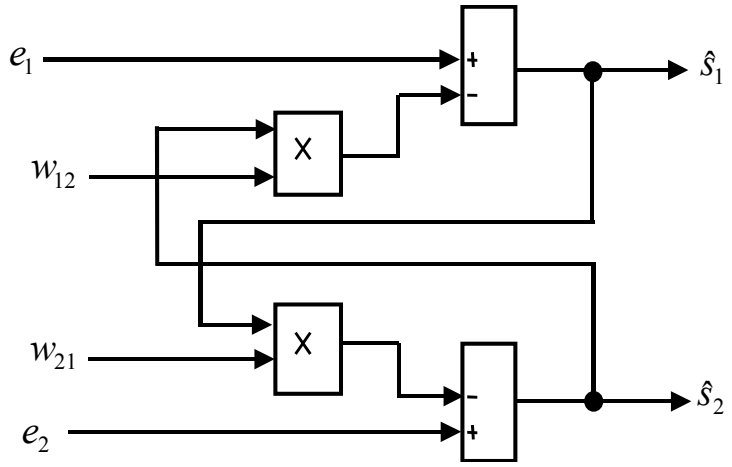


Fig. 2.2.1 Arquitectura de la red H-J.

Donde e_i es la entrada de la red, *w* son los pesos sinápticos y $\hat{s}_i$ sus salidas. La función de activación (ó función de transferencia) que se emplea es una función lineal.

De esta manera, la arquitectura de la red efectúa la transformación lineal de las señales de entrada (**e**), mientras que la regla de aprendizaje realiza el cálculo de los valores de los coeficientes de la matriz de pesos **W**. Calculando sólo los valores que se encuentran fuera de la diagonal principal, esto implica que la arquitectura de la red crece linealmente con el número de entradas, mientras que la arquitectura de la regla, crece en un factor $[(i)^2 - i]$, donde *i* indica el número de entradas. El cálculo de la matriz de pesos se genera iterativamente por medio de la regla de Hebb, cuya expresión es:

$$w_{ij}(t + \Delta t) = w_{ij}(t) + \eta \cdot f\left[\hat{s}_i(t)\right] \cdot g\left[\hat{s}_j(t)\right] \qquad (2.2.2)$$

Esta expresión determina el valor de w_{ij} en forma iterativa. El término (t+Δt) hace referencia al valor siguiente, mientras que el valor de (t) al valor actual. El término restante

de la ecuación, al producto de las dos funciones no lineales ($f[\hat{s}_1(t)]\cdot g[\hat{s}_2(t)]$). Este término, corresponde al incremento Δw_{12} multiplicado por un factor η designado como la razón de aprendizaje, así mismo, el valor Δw_{12} deberá ser diferente de cero cuando se encuentre en proceso de actualización de los pesos, y mínimo o cero, cuando la red converja. Las dos funciones no lineales propuestas son:

$$f(x)=\tanh(x) \qquad y \qquad g(x)=x^3 \tag{2.2.3}$$

Substituyendo la ecuación (2.2.3) en la (2.2.2), la regla de aprendizaje resultante es:

$$w_{12}(t+\Delta t)=w_{12}(t)+\eta\cdot\tanh(\hat{s}_1(t))\cdot(\hat{s}_2(t))^3 \tag{2.2.4}$$

En la Figura 2.2.2 se muestra el diagrama a bloques para la expresión (2.2.4).

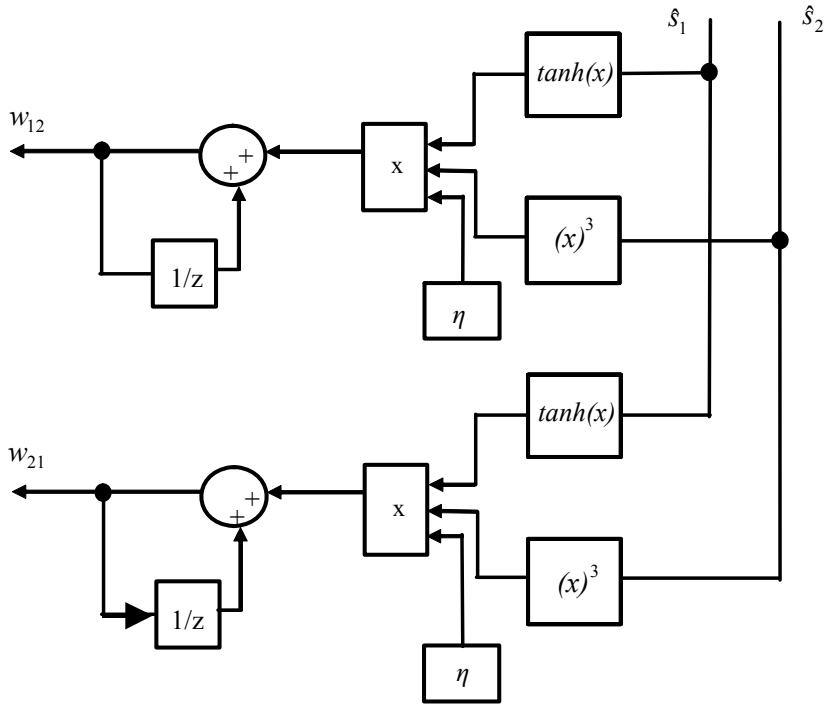


Fig. 2.2.2 Arquitectura de la regla de aprendizaje de la red (H-J).

De esta manera, usando las expresiones 2.2.1 y 2.2.4 se diseñó la arquitectura y el algoritmo de aprendizaje de la red en MATLAB. El valor inicial de los coeficientes de **W** fueron, para w_{11} y w_{22} =1 y para w_{12} y w_{21} = 0, la razón de aprendizaje η fue de 0.0004.

2.2.2 Código empleado en MATLAB.

La primera parte del código corresponde a los comentarios, e indica la manipulación de la función, posteriormente se hace referencia a los valores iniciales de la red. La función desarrollada para el algoritmo requiere un mínimo de dos columnas, una para cada variable. Posteriormente, se define la arquitectura en forma modular de la red con base a la ecuación (2.2.1). Por último, se define el algoritmo de aprendizaje expresado en la ecuación (2.2.4).

```
function [U,W]=ICAHJ(E,Alfa)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% esta función encuentra las componentes independientes empleando la red
% de Jutten Y Herault de NXM variables
% la descripción de la función es:
%
% function [var_out,weight]=ICA-HJ(var_in,Alfa)
%
% var_out= variables de salida
% Weight = matriz de pesos
% var_in = variables de entrada
% Alfa = razón de aprendizaje
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%% Valor de los pesos iniciales %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin < 2, Alfa = 0.0004; end
j=(max(size (E)));
k=(min(size (E)));
W=eye(k); dw=zeros(k);
U=zeros(k,j);

for i=1:j;

%%%%%%%%%% Arquitectura de la Red %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for e=1:k;
           for f=1:k;
            if ( e ~= f)
                H=f;
        U(e,i)=(E(e,i))-(E(H,i))*W(e,H);
            end
        end
    end

%%%%%%%%%% Regla de Aprendizaje %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    for e=1:k;
           for f=1:k;
            if ( e ~= f)
                H=f;
        dw(e,H)=(U(e,i)^3*tanh(U(H,i)));
            end
        end
    end

%%%%%%%%%% Actualizacion de los pesos %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    W=W+dw*Alfa;
end

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

En la siguiente sección se muestran los resultados de la simulación de la red H-J.

2.2.3 Resultados de MATLAB.

Para obtener el desempeño de la red en código de MATLAB, se empleó la combinación lineal de dos señales senoidales propuestas de 800Hz y 1KHz. La tabla 2.2 muestra las matrices de pesos obtenidas a partir de algunos valores propuestos de mezcla.

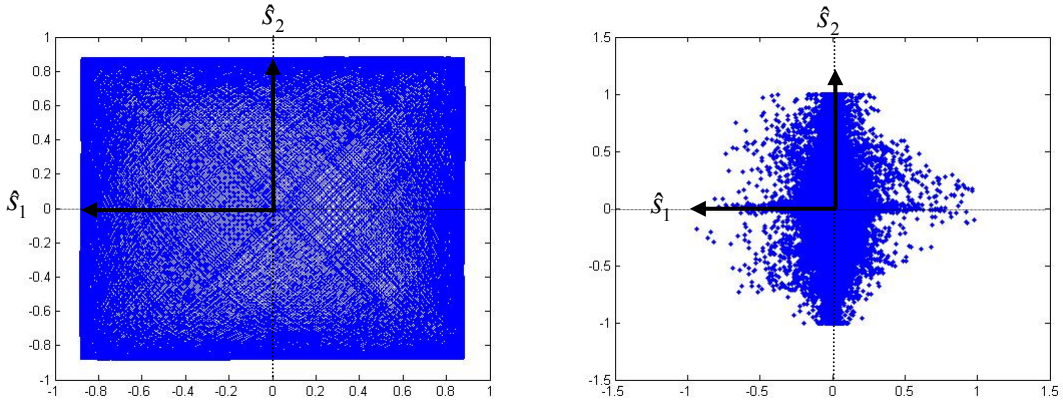
Tabla 2.2 Resultados de simulación obtenidos por la función **`ICAHJ.m`** , así como su error relativo.

No.	Matriz de Mezcla	Matriz de Pesos	Error relativo
1	$\begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.488 \\ 0.488 & 1.0 \end{bmatrix}$	(w_{12}) **2.4%** (w_{21}) **2.4%**
2	$\begin{bmatrix} 1.0 & 0.9 \\ 0.1 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.89 \\ 0.097 & 1.0 \end{bmatrix}$	(w_{12}) **1.1%** (w_{21}) **3.0%**
3	$\begin{bmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.821 \\ 0.821 & 1.0 \end{bmatrix}$	(w_{12}) **8.7%** (w_{21}) **8.7%**
4	$\begin{bmatrix} 1.5 & 0.9 \\ 0.9 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.93 \\ 0.58 & 1.0 \end{bmatrix} \rightarrow \begin{bmatrix} 1.5 & 0.93 \\ 0.87 & 1.0 \end{bmatrix}$	(w_{12}) **3.3%** (w_{21}) **3.3%**
5	$\begin{bmatrix} 2.0 & 1.0 \\ 0.5 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.987 \\ 0.247 & 1.0 \end{bmatrix} \rightarrow \begin{bmatrix} 2.0 & 0.987 \\ 0.494 & 1.0 \end{bmatrix}$	(w_{12}) **1.3%** (w_{21}) **1.2%**

En los casos 4 y 5 de la tabla, se observa que los valores sobre la diagonal principal de la matriz de mezcla (es decir los coeficientes a_{11} y a_{22} , específicamente el coeficiente a_{22}), tienen valores diferentes de la unidad. No obstante, durante el proceso la red encuentra los valores óptimos, proporcionales a la mezcla. Cabe mencionar, que la arquitectura de la red solo puede ajustar los coeficientes a_{12} y a_{21} (coeficientes fuera de la diagonal), considerando los elementos sobre la diagonal normalizados.

Así también, la tabla 2.2 muestra los valores de las matrices obtenidas por la red y su equivalencia proporcional de dichos valores de mezcla. El objetivo, es efectuar una comparación entre los valores propuestos (matriz de mezcla) y los generados (matriz de pesos), observando la correspondencia de cada una de las matrices de mezclas utilizadas ($\mathbf{W} = \mathbf{A}^{-1}$), en donde cada columna de la matriz representa uno de los vectores. Así mismo, se muestra el error relativo de cada vector.

Por otro lado, al aplicar la transformación lineal $\hat{\mathbf{s}}(t) = \mathbf{W}\mathbf{e}(t)$, se obtiene la solución al problema de separación ciega de fuentes (BSS). En la Figura 2.2.3, se muestra el resultado de simulación en un plano bidimensional de las señales de salida de la red.



a) Fuentes senoidales b) Fuentes de señal de música

Fig. 2.2.3 Resultado de simulación en un plano bidimensional de la red (H-J).

La Figura 2.2.4 muestra las señales de entrada representadas en el espacio y en el tiempo, utilizando la matriz de mezcla del caso No. 5 (de la tabla 2.2).

Para este ejemplo, se emplearon dos fuentes de señales senoidales (de 800Hz y 1KHz). La Figura 2.2.5 muestra el resultado de simulación de la salida de la red.

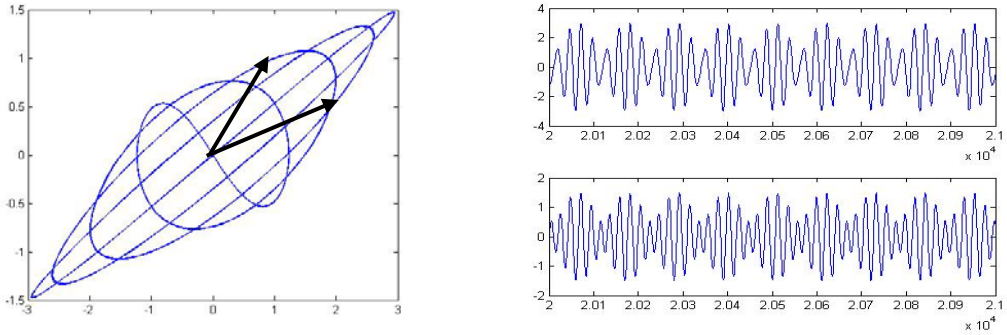


Fig. 2.2.4 Mezcla de señales en el plano bidimensional y en el tiempo.

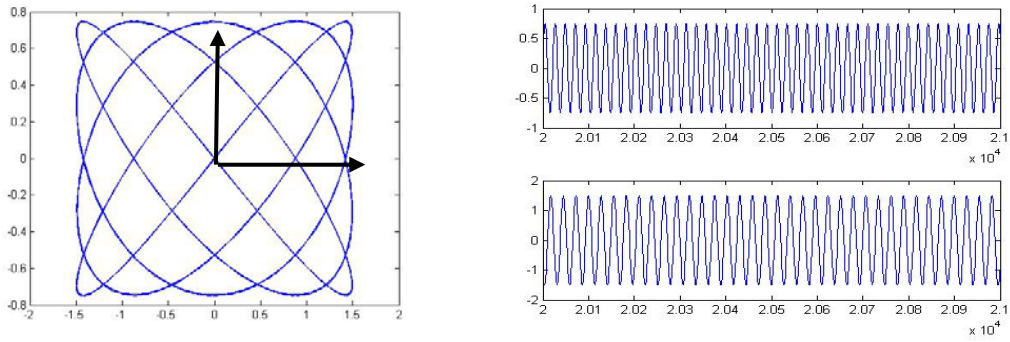


Fig. 2.2.5 Resultado de simulación de la salida de la red.

El resultado de simulación de la salida de la red, se obtiene una vez que ésta ha convergido al valor óptimo. El tiempo de convergencia depende del valor de la razón de aprendizaje (η) utilizada (para este ejemplo se utilizo 4x10-4, el cual puede ser ajustado para proporcionar una convergencia más rápida).

Algunas desventajas que presenta la red H-J, es el aumento en el error relativo que se presenta al operar un gran número de entradas, o al aproximar los valores de los coeficientes de la matriz de mezcla fuera de la diagonal hacia la unidad. No obstante, este algoritmo es muy simple de implementar con circuitos analógicos y digitales [COH92, LNO06], operando en tiempo real con buenos resultados para algunas aplicaciones. Algunos resultados de simulación digital en FPGA de este algoritmo serán mostrados en el capítulo 3.

El siguiente punto, muestra la implementación en MATLAB del algoritmo ICA basado en la maximización de la información o también conocido como INFOMAX.

2.3 Simulación de ICA basado en (INFOMAX)

En este punto, se describe la implementación de ICA basada en la maximización de la información (INFOMAX), desarrollado por Te Won Lee, Bell y Sejnowski [TEW00a]. La primera parte, muestra el desarrollo de la implementación de una red neuronal (2x2), mientras que en la segunda, su algoritmo de aprendizaje.

2.3.1 Arquitectura y regla de aprendizaje de la Red Neuronal

La arquitectura de la red neuronal consiste en un arreglo paralelo de perceptrones, en donde cada una de las salidas define la señal independiente de cada fuente. A este arreglo se le denomina de una sola capa y efectúa la transformación lineal de la forma $\mathbf{u} = \mathbf{We}$, en donde **u** representa el vector de salida de la red, **W** la matriz de pesos y **e** es el vector de datos de entrada. La arquitectura de la red se muestra en la Figura 2.3.1 y se expresa matemáticamente como:

$$u_j = \sum_i^n w_{ij} e_i \qquad i, j = 1,2,3....n \tag{2.3.1}$$

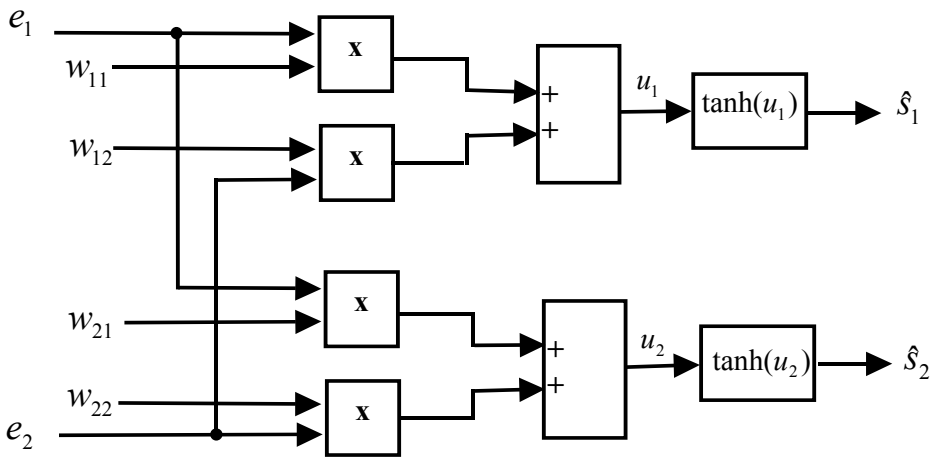


Fig. 2.3.1 Arquitectura de la red para INFOMAX.

Donde e_i indica la señal de entrada (de una sola variable del vector **e**), $\hat{s}_i$ el valor estimado de la fuente independiente, u_i es una aproximación de la salida estimada y w son los coeficientes de la matriz de pesos sinápticos **W**. La expresión resultante, considerando la $\tanh(u_i)$ como la función de activación, se expresa como:

$$\hat{s} = \tanh(\mathbf{u}) = \tanh(\mathbf{We}) \tag{2.3.2}$$

La siguiente etapa, muestra el algoritmo de la regla de aprendizaje para determinar los coeficientes de la matriz **W**. Cabe mencionar, que este método determina cada uno de los coeficientes de la matriz **W**. La regla de aprendizaje se basa en el método iterativo de la regla de Hebb:

$$\mathbf{W}(t + \Delta t) = \mathbf{W}(t) + \Delta\mathbf{W} \tag{2.3.3}$$

Donde **W** indica respectivamente, la matriz de pesos sinápticos actual (t) y siguiente ($t+\Delta t$), cuyos coeficientes son actualizados en cada iteración por $\Delta\mathbf{W}$. Éste es el factor de incremento que ajusta hacia el valor óptimo a los pesos sinápticos, el cual será igual a cero cuando la red converja. Consecuentemente, la matriz actual $\mathbf{W}(t)$ será igual a la siguiente $\mathbf{W}(t+\Delta t)$. El incremento $\Delta\mathbf{W}$ se determina por la siguiente expresión (ver capítulo 1):

$$\Delta\mathbf{W} \propto [\mathbf{I} - \tanh(\mathbf{u})\mathbf{u}^T]\mathbf{W} \tag{2.3.4}$$

En donde **I** representa la matriz Identidad de dimensión *nxn*, **u** es un vector con los datos de la salida estimada y **W** representa la matriz de pesos. La Figura 2.3.2 muestra una arquitectura 2x2 de la regla de aprendizaje usando las ecuaciones (2.3.3) y (2.3.4).

En este ejemplo, la arquitectura de la red determina el cálculo de los cuatro coeficientes de los pesos (w_{11}, w_{12}, w_{21} y w_{22}), que implica a su vez, que la red se incremente de forma cuadrática, con respecto al incremento del número de entradas.

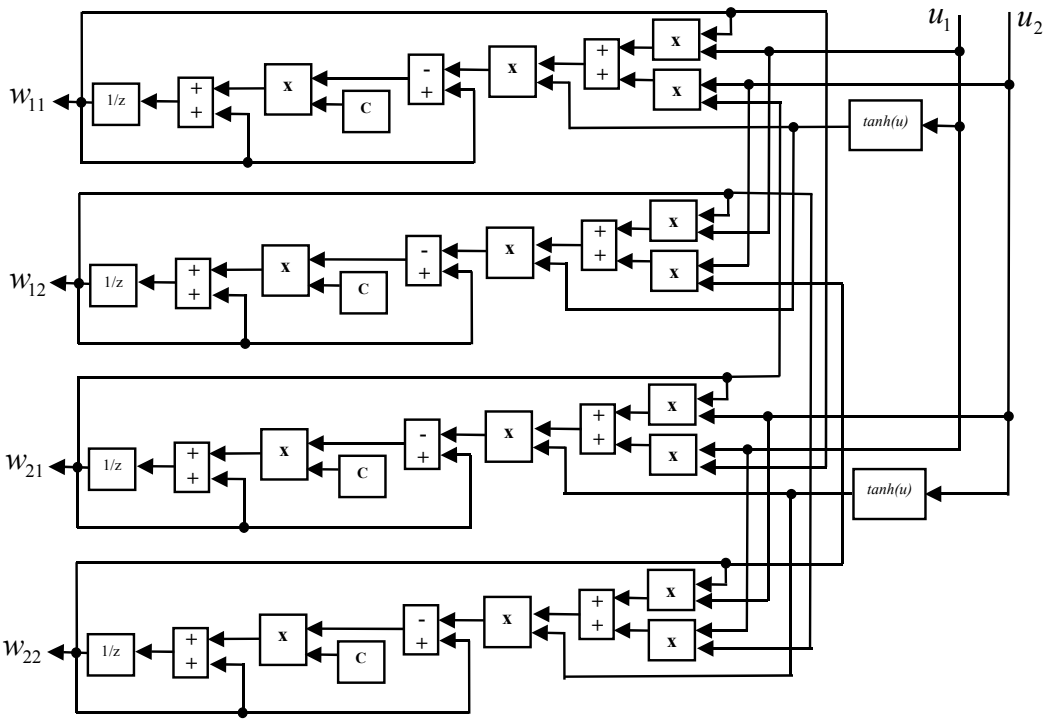


Fig.2.3.2 Algoritmo de la regla de aprendizaje de INFOMAX.

En la siguiente subsección, se analizará la implementación del algoritmo en programación de MATLAB, mostrando algunos resultados de simulación.

2.3.2 Código empleado en MATLAB.

La primera sección del código define los comentarios para la manipulación de la función. Esta función requiere un mínimo de dos columnas, y cada columna representa los datos de cada variable a separar. Otro parámetro que se define en la función es Alfa, que corresponde a la razón de aprendizaje (con un valor predefinido de 0.0004). Los datos de salida generados, corresponden a las variables separadas y a su matriz de pesos óptima.

Posteriormente, la siguiente etapa define la arquitectura en forma modular de la red, de acuerdo con la ecuación (2.3.3). Por último se define el algoritmo de aprendizaje para la ecuación (2.3.4).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INFOMAX
%   regresa los componentes independientes empleando INFOMAX original
%   de cada vector de entrada
%   [salida,W]=ICAinfo(E,alfa)
%     alfa = es la razón de aprendizaje, valor predeterminado de 4e-4
%        E = datos de entrada
%   salida = datos de salida
%        W = Matriz de pesos
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [salida,W] = ICAinfo(E,alfa)
%%%%%%%%% valores predefinidos %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin < 2, alfa =0.0004; end %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[j,k]=(size (E'));
P=(max(size (E)));  %filas
N=(min(size (E))); % columnas
if j == N;
    E=E;
else
    E=E'; %intercambio de filas a columnas
end
Uo=zeros(N,P);   % vector of ceros
W=eye(N);   % condición inicial

for i=1:P;
    e = (E(i,:));
    %%%%%%%%%%% Arquitectura de la Red %%%%%%%%%%%%
    Uo(:,i)=(e*W');
    %%%%%%%%%%% Regla de Aprendizaje %%%%%%%%%%%%
     DW = (eye(N) - (tanh(Uo(:,i))*Uo(:,i)'))*W;
    %%%%%%%%%%% Actualizacion de los pesos %%%%%%%%%
     W = W + (DW')*alfa;
end
[salida] = (Uo);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

En el siguiente punto, se muestran los resultados del la implementación de INFOMAX.

2.3.3 Resultados de MATLAB.

Para obtener el desempeño de la red implementada, se utilizó la combinación lineal de dos señales senoidales (800Hz y 1KHz), aplicando los mismos valores de las matrices de mezclas utilizadas en la tabla 2.2, con el objetivo de efectuar una comparación.

Tabla 2.3 Resultados de simulación obtenidos por la función **`ICAinfo.m`** , así como su error relativo.

No.	**Matriz de Mezcla**	**Matriz de Pesos**	**Error relativo**
1	$\begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 2.446 & -1.223 \\ -1.223 & 2.446 \end{bmatrix} \rightarrow \begin{bmatrix} 1.0 & -0.5 \\ -0.5 & 1.0 \end{bmatrix}$	(w_{12}) **0.0%** (w_{21}) **0.0%**
2	$\begin{bmatrix} 1.0 & 0.9 \\ 0.1 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 2.013 & -1.798 \\ -0.218 & 2.036 \end{bmatrix} \rightarrow \begin{bmatrix} 1.0 & -0.883 \\ -0.102 & 1.0 \end{bmatrix}$	(w_{12}) **1.8%** (w_{21}) **2.0%**
3	$\begin{bmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 9.6504 & -8.684 \\ -8.684 & 9.6504 \end{bmatrix} \rightarrow \begin{bmatrix} 1.0 & -0.899 \\ -0.899 & 1.0 \end{bmatrix}$	(w_{12}) **0.1%** (w_{21}) **0.1%**
4	$\begin{bmatrix} 1.5 & 0.9 \\ 0.9 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 3.962 & -2.452 \\ -2.369 & 2.680 \end{bmatrix} \rightarrow \begin{bmatrix} 1.5 & 0.914 \\ 0.896 & 1.0 \end{bmatrix}$	(w_{12}) **1.5%** (w_{21}) **0.4%**
5	$\begin{bmatrix} 2.0 & 1.0 \\ 0.5 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 2.462 & 1.193 \\ 0.625 & 1.214 \end{bmatrix} \rightarrow \begin{bmatrix} 2.0 & 0.982 \\ 0.507 & 1.0 \end{bmatrix}$	(w_{12}) **1.8%** (w_{21}) **1.4%**

En esta tabla se observa que el error relativo es menor que el obtenido por la red H-J. No obstante, la complejidad es mayor en este algoritmo. La implementación a nivel de sistemas digitales de este algoritmo se presentará en el capítulo 3.

La Figura 2.3.3, muestra las señales de entrada de la red considerando la matriz de mezcla del caso No. 5 (de la tabla 2.3).

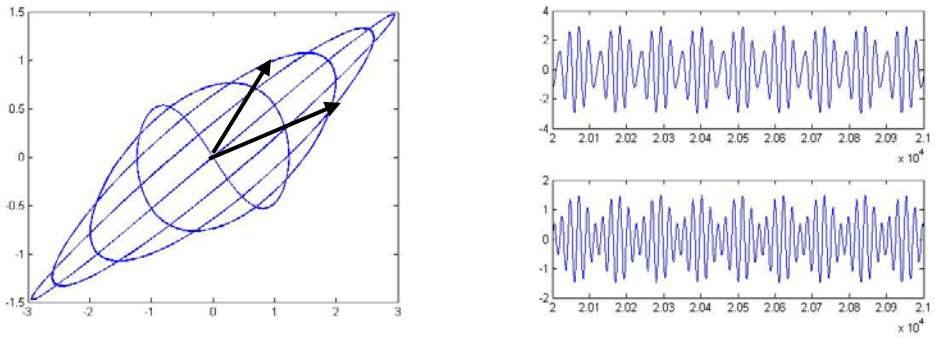


Fig. 2.3.3 Gráfica de entrada de la red en el plano bidimensional y su correspondiente en el tiempo.

La Figura 2.3.4, muestra las señales de salida de la red, una vez que ésta ha convergido. Se puede observan que sus componentes se encuentran en la dirección de los ejes coordenados, cumpliendo con la condición de independencia estadística.

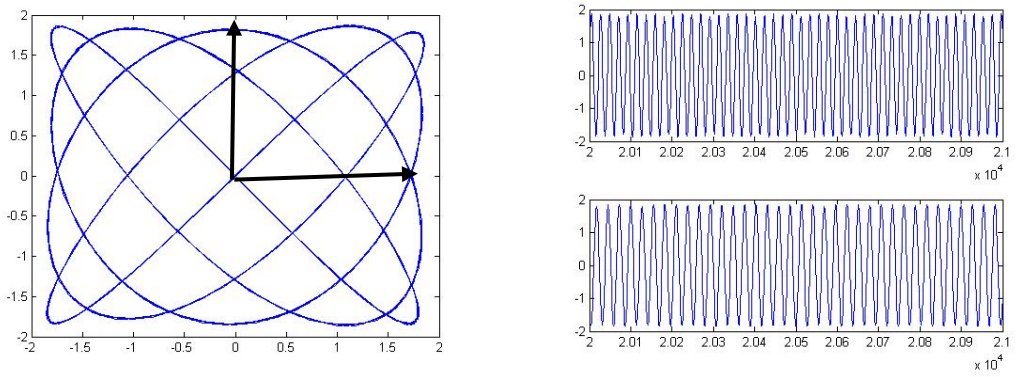


Fig. 2.3.4 Gráfica de salida de la red en el plano bidimensional y su correspondiente en el tiempo.

2.4 Simulación del blanqueado espacial

El blanqueado espacial consiste básicamente de tres procesos, que son: el centrado de las variables, normalización de la varianza y la descorrelación. Estos procesos son necesarios para los algoritmos de ICA, basados en la maximización de la no-gaussianidad, los cuales, se presentarán más adelante.

La descorrelación es una condición necesaria pero no suficiente para la independencia estadística. Por lo tanto, al encontrar una de las direcciones de las componentes independientes (blanqueada), se efectúa una rotación con el propósito de que cada una de las componentes se localice en la dirección de los ejes coordenados, cuya característica es la de independencia. A continuación, se presentan las etapas para la implementación del algoritmo del blanqueado.

1. Centrado de las variables, el cual se logra restando su media:

$$\mathbf{e}(t) \leftarrow \mathbf{e}(t) - E\{\mathbf{e}(t)\} \qquad (2.4.1)$$

2. Transformación lineal **V**, donde $E\{\mathbf{z}\cdot\mathbf{z}^T\} = \mathbf{I}$ (normalización).

$$\mathbf{z}(t) = \mathbf{V}\mathbf{e}(t) \qquad (2.4.2)$$

3. Estimación de la matriz **V**, que es estimada a partir de la siguiente expresión (descorrelación):

$$\mathbf{V} = \mathbf{D}^{-1/2}\mathbf{E}^T \tag{2.4.3}$$

donde **D** es una matriz de los valores propios (D=*diag*($d_1,...,d_n$)) y **E** representa los vectores propios.

2.4.1 Arquitectura y regla de aprendizaje de la Red Neuronal

Como se mencionó en el capítulo 1, existe otro método para efectuar el blanqueado con base en la ejecución de una regla de aprendizaje en línea del análisis de componentes principales, la cual se expresa como:

$$\Delta\mathbf{V} = \gamma\left(\mathbf{I} - \mathbf{V}\mathbf{e}\cdot\mathbf{e}^T\mathbf{V}^T\right)\mathbf{V} = \gamma(\mathbf{I} - \mathbf{z}\cdot\mathbf{z}^T)\mathbf{V} \tag{2.4.4}$$

Donde $\Delta\mathbf{V}$ representa el incremento en cada iteración y γ indica una razón de aprendizaje con base a la siguiente expresión:

$$\mathbf{V} \leftarrow \mathbf{V} + \Delta\mathbf{V} \tag{2.4.5}$$

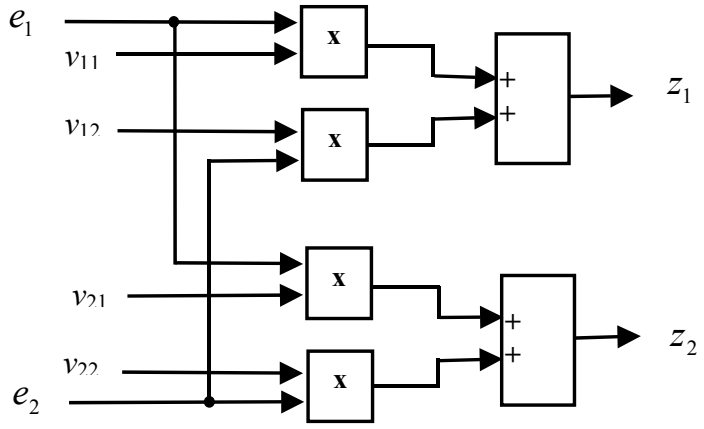


Fig. 2.4.1 Arquitectura para efectuar la transformación lineal de **e**(t).

La arquitectura (para el caso particular de tamaño 2x2) utilizada para efectuar la transformación ($\mathbf{z}(t) = \mathbf{V}\mathbf{e}(t)$), se muestra en la Figura (2.4.1), en donde los coeficientes v_{11}, v_{12}, v_{21} y v_{22} de la matriz (**V**), se definen por las ecuaciones (1.7.9) y (1.7.10). La Figura 2.4.2, muestra la arquitectura de la regla de aprendizaje (para obtener la matriz **V**). La siguiente parte trata con la implementación del algoritmo en MATLAB.

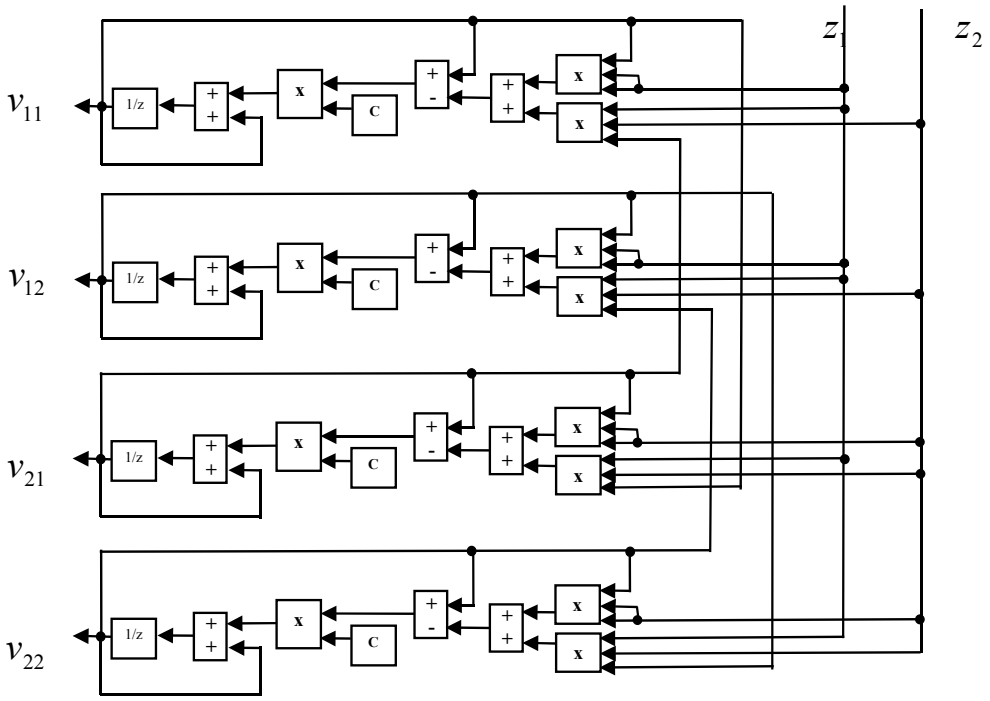


Fig. 2.4.2 Arquitectura para el cálculo de la matriz **V**.

2.4.2 Código en MATLAB.

Se muestra a continuación, el código en MATLAB para realizar el blanqueado de las señales, describiendo primero la manipulación de la función y posteriormente definiendo la implementación de la ecuación (2.4.4), obteniendo como salida la matriz de blanqueado **V** y los datos **Z**.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   funcion de blanqueado empleando los valores y vectores propios
%   function [Z,V] = wtn_DE(E)
%
%   E = vectores de datos de entrada
%   gama = razon de actualizacion
%   V = matriz de blanqueado
%   Z = datos de salida blanqueados
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Z,V] = wtn_DE(E)
%%%%%%%%% Default values %%%%%%%%%%
[j,k]=(size (E'));
P=(max(size (E)));
N=(min(size (E))); % P=17408, N=2, for example
if j == N;
    E=E';
else
```

```
    E=E;
end
%%%%%%%%%%%% Arquitectura %%%%%%%%%%%%%%%%%
x = cov(E',1);
[Ev,D] = eig(x); %Eigenvectores y eigenvalores
D=inv(sqrt(D));
V=D*Ev';
Z=V*E;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Una vez que se calculo la matriz **V**, se efectúa la transformación lineal **z** = **Ve**, obteniendo el blanqueado espacial de los datos de entrada **e**, en la salida **z**. En el siguiente punto, se muestran algunos resultados de simulación del blanqueado espacial.

2.4.3 Resultados de MATLAB.

La tabla 2.4, muestra los resultados obtenidos por la arquitectura mostrada en el punto 2.4.1, utilizando señales senoidales. Una columna muestra las matrices de mezclas utilizadas, la siguiente muestra las matrices **V** obtenidas por la función y la ultima columna muestra la correlación de los datos de salida.

Tabla 2.4 Resultados obtenidos por la arquitectura de blanqueado (utilizando la ec. 2.4.4).

No.	Matriz de Mezcla	Matriz V	$E\{\mathbf{z}\cdot\mathbf{z}^T\}\cong\mathbf{I}$
1	$\begin{bmatrix}1.0 & 0.5\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}0.8945 & -0.4471\\ -0.4471 & 0.8945\end{bmatrix}$	$\begin{bmatrix}1.0 & 0.0004\\ 0.0004 & 1.0\end{bmatrix}$
2	$\begin{bmatrix}1.0 & 0.9\\ 0.1 & 1.0\end{bmatrix}$	$\begin{bmatrix}0.8870 & -0.3450\\ -0.4619 & 0.9386\end{bmatrix}$	$\begin{bmatrix}1.0 & 0.0003\\ 0.0003 & 1.0\end{bmatrix}$
3	$\begin{bmatrix}1.0 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}0.7438 & -0.6684\\ -0.6684 & 0.7438\end{bmatrix}$	$\begin{bmatrix}1.0 & 0.0142\\ 0.0142 & 1.0\end{bmatrix}$
4	$\begin{bmatrix}1.5 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}0.735 & -0.5144\\ -0.6687 & 0.8575\end{bmatrix}$	$\begin{bmatrix}1.0 & -0.0013\\ -0.0013 & 1.0\end{bmatrix}$
5	$\begin{bmatrix}2.0 & 1.0\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}0.8088 & -0.2941\\ -0.588 & 0.9557\end{bmatrix}$	$\begin{bmatrix}1.0 & -0.0002\\ -0.0002 & 1.0\end{bmatrix}$

La tabla 2.5, muestra los resultados obtenidos por la función de blanqueado (función wtn_DE), utilizando las mismas condiciones anteriores.

Tabla 2.5 Resultados obtenidos por la función de blanqueado wtn_DE (utilizando la ec. 2.4.3).

No.	Matriz de Mezcla	Matriz V	$E\{\mathbf{z}\cdot\mathbf{z}^T\}\cong\mathbf{I}$
1	$\begin{bmatrix}1.0 & 0.5\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}-2.0 & 2.0\\ 0.6667 & 0.6667\end{bmatrix}$	$\begin{bmatrix}1.0 & 0\\ 0 & 1.0\end{bmatrix}$
2	$\begin{bmatrix}1.0 & 0.9\\ 0.1 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.374 & -2.0295\\ -0.7426 & -0.5027\end{bmatrix}$	$\begin{bmatrix}1.0 & 0\\ 0 & 1.0\end{bmatrix}$
3	$\begin{bmatrix}1.0 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}-10 & 10\\ 0.5263 & -0.5027\end{bmatrix}$	$\begin{bmatrix}1.0 & 0\\ 0 & 1.0\end{bmatrix}$
4	$\begin{bmatrix}1.5 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}2.708 & -3.563\\ -0.5155 & -0.3918\end{bmatrix}$	$\begin{bmatrix}1.0 & 0\\ 0 & 1.0\end{bmatrix}$
5	$\begin{bmatrix}2.0 & 1.0\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}0.9078 & -2.095\\ -0.5358 & -0.2321\end{bmatrix}$	$\begin{bmatrix}1.0 & 0\\ 0 & 1.0\end{bmatrix}$

De las dos tablas, se observa que los mejores resultados se obtienen en la tabla 2.4.3, ya que el resultado del producto, se obtiene la matriz identidad. La figura 2.4.1, muestra la mezcla de dos señales senoidales en el plano bidimensional, considerando la matriz de mezcla del caso No. 3 (de la tabla 2.4).

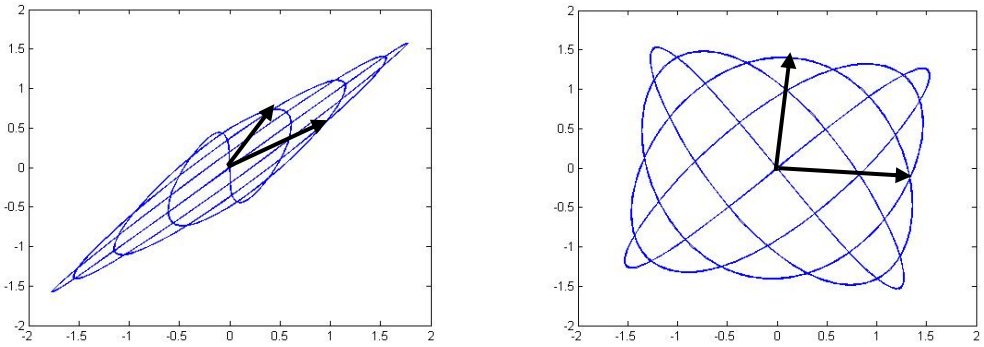


Fig. 2.4.1 Señal mezclada y señal blanqueada en el plano bidimensional.

La figura 2.4.2, muestra el blanqueado de las señales usando la función de blanqueado por valores y vectores propios, considerando la matriz de mezcla del caso No. 3 (de la tabla 2.5).

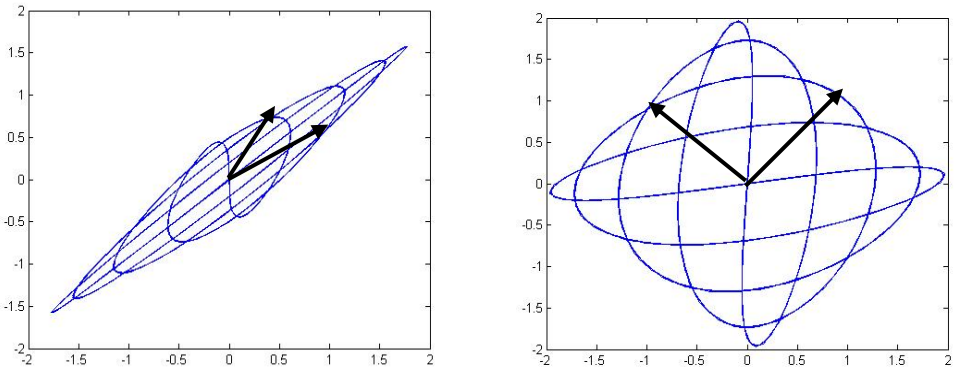


Fig. 2.4.2 Señal mezclada y señal blanqueada en el plano bidimensional.

La siguiente etapa después del blanqueado, es la maximización de la no-gaussianidad, que consiste en medir la gaussianidad de una distribución mediante un vector unitario ubicado en el origen. De esta forma, se efectúa una rotación en el plano bidimensional, para determinar la dirección del valor mínimo de la gaussianidad. Esta dirección corresponde a una de las componentes independientes (con base al teorema central del límite).

Como se mencionó en el capítulo 1, el blanqueado corresponde a la mitad del procesamiento de ICA. El siguiente punto muestra los métodos para obtener la medida de la no-gaussianidad, por curtosis y por negentropía.

2.5. Simulación de ICA basada en la curtosis

En esta sección, se muestra la implementación del algoritmo ICA basado en la maximización de la no-gaussianidad por curtosis. Este método busca determinar al menos una de las componentes (fuentes originales), considerando que se parten de señales blanqueadas. Y consecuentemente las señales (de las fuentes originales) cumplieron la condición de independencia estadística y no-gaussianidad.

Para encontrar una de las "2n" componentes, donde "n" es el número de fuentes independientes, se utilizará la transformación lineal $y = \mathbf{w}^T \mathbf{z}$. Hallando la solución, en la dirección del vector **w** que incrementa (maximiza) el valor absoluto de la curtosis.

Como fue mostrado en el capítulo 1, existen dos metodologías para encontrar la maximización de la curtosis. La primera se basa en el gradiente, que se utiliza generalmente en sistemas de redes neuronales. La segunda se basa en algoritmos de punto fijo, los cuales son más rápidos y más eficientes en cuanto a su convergencia, aplicado generalmente a métodos por lotes (*batch*). En ambos casos, estos algoritmos pueden encontrar múltiples componentes bajo la restricción de descorrelación. A continuación, se muestra la arquitectura de la red empleada en algoritmos basados en la curtosis por gradiente.

2.5.1 Arquitectura y regla de aprendizaje de la Red Neuronal

Como se mencionó anteriormente, ICA está constituido por dos etapas. La primera, fue descrita en el punto 2.4, y efectúa el blanqueado de las señales cuya arquitectura se muestra en las Figuras 2.4.1 y 2.4.2. A continuación, se muestra la segunda etapa de la arquitectura y su regla de aprendizaje, para generar la medida de la no gaussianidad, completando con esto, el procesamiento de ICA. En la Figura 2.5.1, se muestra el diagrama a bloques de esta arquitectura.

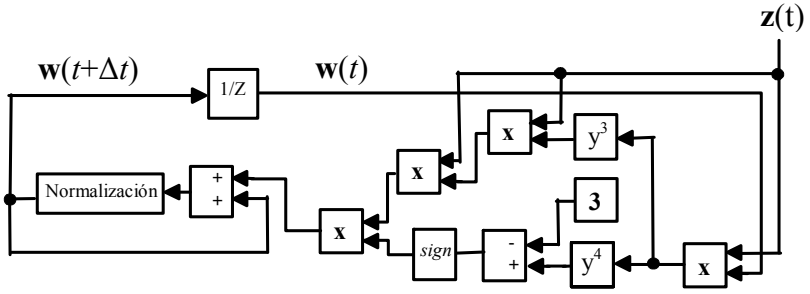


Fig. 2.5.1 Arquitectura de la regla de aprendizaje, basada en el gradiente de la curtosis.

Este diagrama a bloques, corresponde a la regla de aprendizaje y representa las expresiones (1.7.26) y (1.7.27). Estas expresiones, generan la normalización y el cálculo del vector **w**. A su vez, el bloque *1/Z* genera un retardo de una muestra para obtener el valor nuevo a partir del los valores actuales. Se considera que **z(t)** es un vector con datos blanqueados de entrada y **w(t)**, como el valor actual de los pesos.

La arquitectura de la red que genera la transformación lineal, es similar a la que se mostró en la Figura 2.4.1, por lo que su explicación será omitida.

2.5.2 Código en MATLAB.

A continuación se muestra el código en MATLAB del Análisis de Componentes Independientes por curtosis de punto fijo. La función se invoca con **`[ica]=ICAcur_fp(d)`**, para generar todo el procesamiento de ICA (blanqueado y máxima no gaussianidad). El formato de los datos de entrada, pueden ser representados en "n" columna o en "n" filas, siempre y cuando la cantidad de datos sea mayor que el número de variables. Los datos de entrada se designan con la variable "d" de la función, mientras que su salida se obtiene en la variable "ica".

De esta manera, la primera parte del programa corresponde al ajuste de las filas y columnas para su posterior procesamiento. Posteriormente, se efectúa el blanqueado de las señales mediante sus valores y vectores propios. La última parte, muestra la implementación del algoritmo ICA por curtosis de punto fijo, con la ecuación (1.7.29) analizada en el capítulo 1.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% efectúa el Análisis de Componentes Independientes
% por Curtosis de punto fijo
% [ica] = ICAcur_fp(d)
% d = variable de entrada
% icas = variable de salida
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%Ajuste de filas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ica] = ICAcur_fp(d);
disp('Leyendo matriz...');
[j,k]=((size (d)));  % Ajuste de las columnas
P=(max(size (d)));      % y las filas
N=(min(size (d))); % por ejemplo P=17408, N=2.
if j == N;
    d=d;
else
    d=d';  % en caso de que estén invertidas se efectúa su transpuesta
end
[canales,muestras] = size(d); % define la dimensión de los datos
A=rand(canales,canales); % matriz con datos iniciales aleatorios
s=d; % cambio de variable
disp('Etapa de centrado...');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Centrado %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:canales;
    s(i,:)=s(i,:)-mean(s(i,:));  % se efectúa la resta de su media
end
%%%%%%%%%%%%%%%%%%%%%%%% Blanqueado %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Etapa de blanqueado...');
```

```
x = cov(s',1);
[E,D] = eig(x); %Eigenvectores y eigenvalores
D=inv(sqrt(D));
V=D*E';
matrizB=V*s;

%%%%%%%%%% Algoritmo de Punto Fijo por Curtosis %%%%%%%%%%%%%%%%%%%%%%%%
disp('Algoritmo de punto fijo...');
z=matrizB; % cambio de variable
itera_max=1000;
mix_A=zeros(canales); % matriz de nxn
for comp=1:canales;
    w_ant=rand(canales,1);
    w_ant=w_ant./norm(w_ant,2);
    B=mix_A;
    for iter=1:itera_max;
        w_nueva=((z*(z'*w_ant).^3)/(muestras))-3*(w_ant);
        w_nueva=w_nueva./norm(w_nueva,2);
        w_nueva=w_nueva-B*B'*w_nueva;
        w_nueva=w_nueva./norm(w_nueva,2);
        if 1-abs(w_nueva'*w_ant) <= 0.0000001;
            break;
        else
            w_ant=w_nueva;
        end
    end
    mix_A(:,comp)=w_nueva;
end
disp('Imprimiendo ICA...');
ica=mix_A'*z;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

A continuación, se muestran algunos resultados de las simulaciones en MATLAB.

2.5.3 Resultados de MATLAB.

En este punto, se muestran los resultados de simulación empleando las mismas matrices de mezcla de la tabla 2.2. Al igual que en las simulaciones anteriores, se utilizó la combinación lineal de dos señales senoidales (de 800Hz y 1KHz).

Empleando la arquitectura del punto 2.5.1, que hace uso del algoritmo de ICA por gradiente de curtosis, se obtuvieron los valores de los coeficientes de la matriz de pesos **W**. La tabla 2.6, muestra los resultados de simulación.

En la tabla 2.7, se muestran los resultados de simulación utilizando el código de MATLAB del algoritmo de punto fijo.

Tabla 2.6. Resultados de simulación obtenidos por la arquitectura de ICA por gradiente, así como su error relativo.

No.	Matriz de Mezcla	Matriz de Pesos	Error relativo
1	$\begin{bmatrix}1.0 & 0.5\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.8874 & 0.9394\\ 0.9463 & 1.8838\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.4986\\ 0.5013 & 1.0\end{bmatrix}$	(w_{12}) **0.28%** (w_{21}) **0.26%**
2	$\begin{bmatrix}1.0 & 0.9\\ 0.1 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.7894 & 1.3510\\ 0.1788 & 1.4971\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.9024\\ 0.0996 & 1.0\end{bmatrix}$	(w_{12}) **0.26%** (w_{21}) **0.4%**
3	$\begin{bmatrix}1.0 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}7.4558 & 6.6849\\ 6.7128 & 7.4307\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.8996\\ 0.9003 & 1.0\end{bmatrix}$	(w_{12}) **0.004%** (w_{21}) **0.33%**
4	$\begin{bmatrix}1.5 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}2.0792 & 2.0493\\ 1.2457 & 2.2734\end{bmatrix}\rightarrow\begin{bmatrix}1.5 & 0.9014\\ 0.8986 & 1.0\end{bmatrix}$	(w_{12}) **0.15%** (w_{21}) **0.14%**
5	$\begin{bmatrix}2.0 & 1.0\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.6427 & 1.2014\\ 0.4101 & 1.2022\end{bmatrix}\rightarrow\begin{bmatrix}2.0 & 0.999\\ 0.4992 & 1.0\end{bmatrix}$	(w_{12}) **0.1%** (w_{21}) **0.14%**

Tabla 2.7. Resultados de simulación obtenidos por la función **`ICAcur_fp`** , así como su error relativo.

No.	Matriz de Mezcla	Matriz de Pesos	Error relativo
1	$\begin{bmatrix}1.0 & 0.5\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.8856 & 0.9428\\ .9428 & 1.8856\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.5\\ 0.5 & 1.0\end{bmatrix}$	(w_{12}) **0%** (w_{21}) **0%**
2	$\begin{bmatrix}1.0 & 0.9\\ 0.1 & 1.0\end{bmatrix}$	$\begin{bmatrix}0.1554 & 1.5541\\ 1.5541 & 1.3987\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.9\\ 0.1 & 1.0\end{bmatrix}$	(w_{12}) **0%** (w_{21}) **0%**
3	$\begin{bmatrix}1.0 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}0.7443 & 0.6698\\ 0.6698 & 0.7443\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	(w_{12}) **0%** (w_{21}) **0%**
4	$\begin{bmatrix}1.5 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}2.4685 & 2.6\\ 1.4488 & 2.4685\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.9494\\ 0.5869 & 1.0\end{bmatrix}$	(w_{12}) **0.05%** (w_{21}) **0.02%**
5	$\begin{bmatrix}2.0 & 1.0\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.8856 & 0.9428\\ 0.4714 & 0.9428\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 1.0\\ 0.25 & 1.0\end{bmatrix}$	(w_{12}) **0%** (w_{21}) **0%**

En la primera tabla, se observa que los valores obtenidos por el método del gradiente, presentan un error relativo mayor que con el método de punto fijo. Este error se genera, debido a que en cada iteración, se produce una actualización de los coeficientes y por lo tanto, tienden al valor óptimo aunque no llegan a ser exactos, obteniendo solo un aproximado de ellos. El tiempo de convergencia que le toma a los coeficientes en obtener el valor óptimo, se encuentra relacionado con los valores iniciales asignados; Es decir, que cuanto mas alejado se encuentre el valor inicial del óptimo, mayor será el tiempo de convergencia, con respecto a aquellos en donde el valor inicial y el final se encuentren mas próximos. Cabe mencionar, que el error relativo se obtuvo a partir de las matrices de pesos normalizadas.

Como se mencionó anteriormente, la solución de la matriz de pesos corresponde a la inversa de la matriz de mezcla ($\mathbf{A}^{-1} = \mathbf{W}$). Por lo tanto, al aplicar la transformación lineal $\hat{\mathbf{s}}(t) = \mathbf{We}(t)$ se obtendrán las soluciones al problema de separación ciega de fuentes (BSS). En la figura 2.5.2 se muestra en un plano bidimensional las señales resultantes de salida de la red.

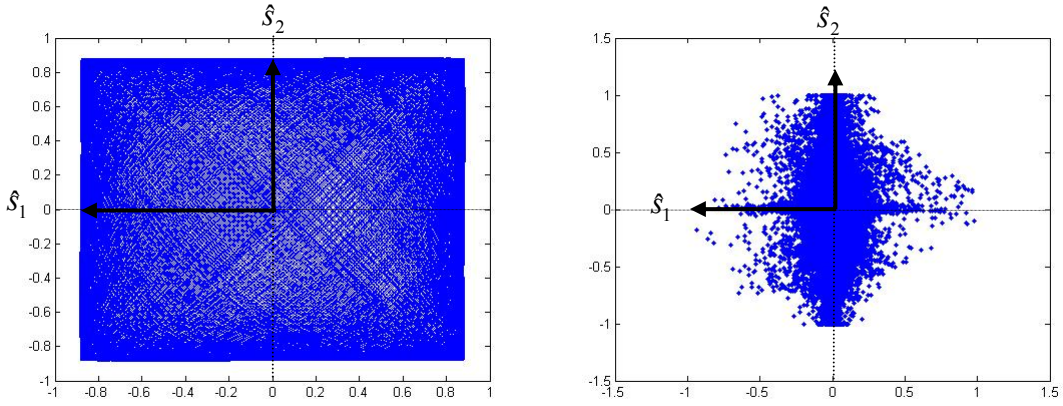


Fig. 2.5.2 Resultado de simulación en un plano bidimensional de la red (H-J).

Empleando las señales senoidales antes mencionadas y la matriz de mezcla del caso No. 5 de la tabla 2.7, el resultado de simulación se muestra en la figura 2.5.3.

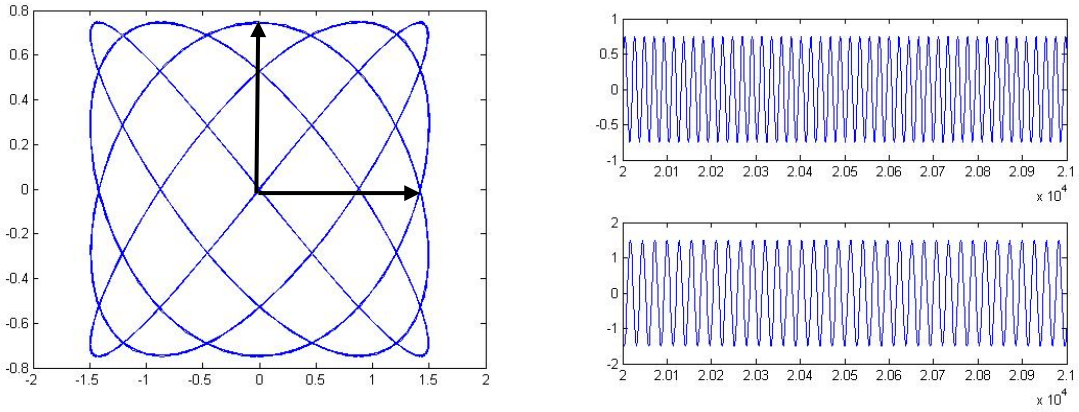


Fig. 2.5.3 Resultado de simulación empleando la función **`ICAcur_fp`**.

En este punto, se mostraron algunos resultados obtenidos empleando la curtosis como medida de no gaussianidad. Sin embargo, en la práctica este método presenta algunas desventajas cuando los valores obtenidos muestran valores muy alejados de la varianza (outliers). Por ejemplo una muestra de 1000 valores de una variable aleatoria (con media cero y varianza unitaria) contienen un solo valor igual a 10. Por lo tanto, la curtosis de este valor, será igual a $10^4/1000-3=7$, el cual ocasionará el aumento de la curtosis general, dependiendo solo de aquellas muestras con valores muy alejados de la varianza, ocasionando que el error aumente en las matrices de pesos. Por lo tanto, se dice que la curtosis no es un método robusto en la obtención de la no gaussianidad.

Otra medida de la no gaussianidad con mejores resultados es la negentropía. El siguiente punto muestra la implementación en MATLAB del algoritmo ICA basado en la negentropía.

2.6 Simulación de ICA basada en la negentropía

En este punto, se muestra la implementación del ICA basada en la medida de la no gaussianidad por negentropía. La negentropía se deriva del término entropía, y se basa en la teoría de la información para obtener la medida o grado de incertidumbre que existe sobre un conjunto de datos.

En el capítulo 1 se mostraron las ecuaciones de la negentropía, las cuales mencionan que una variable gaussiana obtiene la máxima entropía. Así mismo, al efectuar la diferencia con respecto a otra variable, se obtiene un valor mayor o igual a cero. De esta forma se determina la medida de la no gaussianidad. El término "*neg*" hace referencia a la negación o complemento de la entropía.

Al igual que en la curtosis, se derivó un algoritmo de la negentropía por gradiente y por punto fijo. A continuación, se muestra la implementación de la arquitectura de la Negentropía por gradiente.

2.6.1 Arquitectura y regla de aprendizaje de la Red Neuronal

En este punto, se muestra la implementación del algoritmo del gradiente de la negentropía como medida de la no gaussianidad. Esta, se encuentra precedida de la etapa de blanqueado, cuya arquitectura fue mostrada en las Figuras 2.4.1 y 2.4.2. La implementación de la regla de aprendizaje del gradiente, se muestra mediante el diagrama a bloques de la Figura 2.6.1.

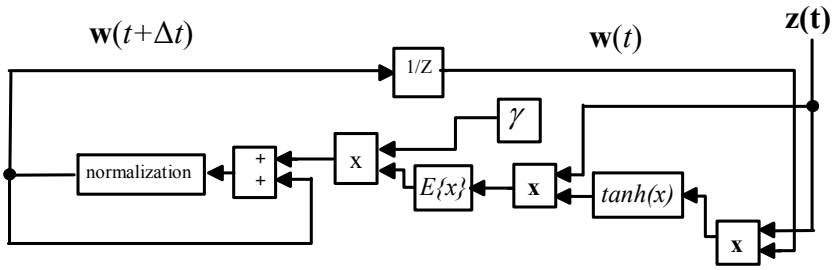


Fig. 2.6.1 Arquitectura de la regla de aprendizaje, basada en el gradiente de la negentropía.

Este diagrama a bloques, corresponde a la regla de aprendizaje y representa las expresiones (1.7.37) y (1.7.39), generando la normalización y el cálculo del vector **w**. El bloque *1/Z* genera un retardo de una muestra, para obtener el valor nuevo a partir de los valores actuales. Se considera que **z(t)** es un vector con datos blanqueados de entrada y **w(t)**, es el valor actual de los pesos.

2.6.2 Código en MATLAB.

En esta sección se muestra el código en MATLAB de ICA empleando la negentropía como medida de la no gaussianidad. La primera parte, después de los comentarios, muestra el ajuste de filas y columnas, posteriormente se realiza el proceso de blanqueado de las señales y por último, se efectúa el cálculo de las Componentes Independientes empleando la negentropía como la medida de la no gaussianidad.

En el capítulo 1 se definieron tres funciones y son asignadas con base al valor que toma "tipo". Para el valor de "1", efectúa la función (1.7.40); para "2", la (1.7.41) y para el tipo "3", la (1.7.42).

```
% efectua el Analisis de Componetes Independientes
% por Negentropia de punto fijo
% [icas] = procICA(d,tipo)
% donde tipo = 1 ; tanh (x)
%       tipo = 2 ; exp (x)
%       tipo = 3 ; pow (x)
% y d son los datos de entrada
%*******Generacion de Canales y
muestras*********************************
function [icas] = ICAneg_fp(d,tipo);
disp('Leyendo matriz...');
[j,k]=((size (d)));  % Ajuste de las columnas
P=(max(size (d)));      % y las filas
N=(min(size (d))); % por ejemplo P=17408, N=2.
if j == N;
    d=d;
else
    d=d';           % en caso de que esten invertidas se efectua su
transpuesta
end
[canales,muestras] = size(d); % define los datos de entrada, el número de
canales y de muestras
A=rand(canales,canales); % matriz con datos iniciales aleatorios
s=d; % cambio de variable
disp('Etapa de centrado...');
%*******Centrado***********************************************************
**
for i=1:canales
    s(i,:)=s(i,:)-mean(s(i,:));  % se efectua la resta de su media
end
%*******Blanqueado********************************************************
**
disp('Etapa de blanqueado...');
x = cov(s',1);
[E,D] = eig(x); %Eigenvectores y eigenvalores
```

```
D=inv(sqrt(D));
V=D*E';
matrizB=V*s;

%*******Algoritmo de Punto Fijo con Negentropia************************
disp('Algoritmo de punto fijo...');
z=matrizB; % cambio de variable
itera_max=1000;
mix_A=zeros(canales); % matriz de nxn
for comp=1:canales
    w_ant=rand(canales,1);
    w_ant=w_ant./norm(w_ant,2);
    B=mix_A;
    for iter=1:itera_max
        switch (tipo)
            case 1 %funcion tanh
                a1=1;
                G=tanh(a1.*(z'*w_ant));
                g=a1.*(1-tanh(a1.*(z'*w_ant)).^2); %funcion no lineal
            case 2 %funcion exp
                a2=1;
                G=(z'*w_ant).*exp(-a2.*((z'*w_ant).^2)/2);
                g=(1-a2.*(z'*w_ant).^2).*exp(-a2.*((z'*w_ant).^2)/2);
%funcion no lineal
            case 3 %funcion pow
                G=(z'*w_ant).^3;
                g=3*(z'*w_ant).^2; %funcion no lineal
        end
        w_nueva=(z*G-sum(g)'*w_ant)/(muestras);
        w_nueva=w_nueva./norm(w_nueva,2);
        w_nueva=w_nueva-B*B'*w_nueva;
        w_nueva=w_nueva./norm(w_nueva,2);
        if 1-abs(w_nueva'*w_ant) <= 0.0000001
            break;
        else
            w_ant=w_nueva;
        end
    end
    mix_A(:,comp)=w_nueva;
end
icas=mix_A'*z;
```

A continuación, se muestran algunos resultados de las simulaciones en MATLAB del programa antes mencionado.

2.6.3 Resultados de MATLAB.

Al igual que en los casos anteriores, se empleó la combinación lineal de señales senoidales (800 y 1KHz) para su caracterización. Las tablas 2.8 y 2.9 muestran las matrices de pesos obtenidas a partir de los algoritmos del gradiente y de punto fijo por negentropía.

Tabla 2.8 Resultados de simulación obtenidos por la arquitectura de ICA por gradiente, así como su error relativo.

No.	Matriz de Mezcla	Matriz de Pesos	Error relativo
1	$\begin{bmatrix}1.0 & 0.5\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.8874 & 0.9428\\ 0.9428 & 1.8874\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.4995\\ 0.4995 & 1.0\end{bmatrix}$	(w_{12}) **0.1%** (w_{21}) **0.1%**
2	$\begin{bmatrix}1.0 & 0.9\\ 0.1 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.7880 & 1.3391\\ 0.1801 & 1.4980\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.8939\\ 0.1007 & 1.0\end{bmatrix}$	(w_{12}) **0.6 %** (w_{21})**0.7%**
3	$\begin{bmatrix}1.0 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}7.5 & 6.6989\\ 6.6989 & 7.4432\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.9\\ 0.8931 & 1.0\end{bmatrix}$	(w_{12}) **0.0 %** (w_{21}) **0.7%**
4	$\begin{bmatrix}1.5 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}2.0545 & 2.0132\\ 1.2822 & 2.2420\end{bmatrix}\rightarrow\begin{bmatrix}1.5 & 0.8979\\ 0.936 & 1.0\end{bmatrix}$	(w_{12}) **0.23%** (w_{21}) **0.4 %**
5	$\begin{bmatrix}2.0 & 1.0\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.6314 & 1.2122\\ 0.4202 & 1.2122\end{bmatrix}\rightarrow\begin{bmatrix}2.0 & 1.0\\ 0.51 & 1.0\end{bmatrix}$	(w_{12}) **2.0%** (w_{21}) **0.0%**

Tabla 2.9 Resultados de simulación obtenidos por la función **`ICAneg_fp`** , así como su error relativo.

No.	Matriz de Mezcla	Matriz de Pesos	Error relativo
1	$\begin{bmatrix}1.0 & 0.5\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.8856 & 0.9428\\ 0.9428 & 1.8856\end{bmatrix}\rightarrow\begin{bmatrix}-1.0 & 0.5\\ -0.5 & 1.0\end{bmatrix}$	(w_{12}) **0%** (w_{21}) **0%**
2	$\begin{bmatrix}1.0 & 0.9\\ 0.1 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.5541 & 1.3987\\ 0.1554 & 1.5541\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.9\\ 0.1 & 1.0\end{bmatrix}$	(w_{12}) **0 %** (w_{21}) **0%**
3	$\begin{bmatrix}1.0 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}7.4430 & 6.6989\\ 6.6989 & 7.4430\end{bmatrix}\rightarrow\begin{bmatrix}1.0 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	(w_{12}) **0%** (w_{21}) **0 %**
4	$\begin{bmatrix}1.5 & 0.9\\ 0.9 & 1.0\end{bmatrix}$	$\begin{bmatrix}2.4685 & 2.4386\\ 1.4817 & 2.7002\end{bmatrix}\rightarrow\begin{bmatrix}1.5 & 0.9031\\ 0.9003 & 1.0\end{bmatrix}$	(w_{12}) **0.34%** (w_{21}) **0.03 %**
5	$\begin{bmatrix}2.0 & 1.0\\ 0.5 & 1.0\end{bmatrix}$	$\begin{bmatrix}1.0532 & 1.6336\\ 0.2633 & 1.6326\end{bmatrix}\rightarrow\begin{bmatrix}2.0 & 1.0\\ 0.5 & 1.0\end{bmatrix}$	(w_{12})**0 %** (w_{21}) **0 %**

Estas tablas, muestran el desempeño del algoritmo de gradiente y de punto fijo por negentropía, donde se observa que el error se reduce en el algoritmo de punto fijo, debido a que el cálculo de la matriz de pesos se efectúa mediante el promedio de todos los datos, mientras que el del gradiente lo realiza dato a dato.

Como se mencionó anteriormente, la solución de la matriz de pesos debe corresponder a la inversa de la matriz de mezcla ($\mathbf{A}^{-1} = \mathbf{W}$). Por lo tanto, al aplicar la transformación lineal $\hat{\mathbf{s}}(t) = \mathbf{W}\mathbf{e}(t)$ se obtendrán las soluciones del problema de separación ciega de fuentes (BSS). En la Figura 2.6.2 se muestra en un plano bidimensional las señales resultantes de salida de la red.

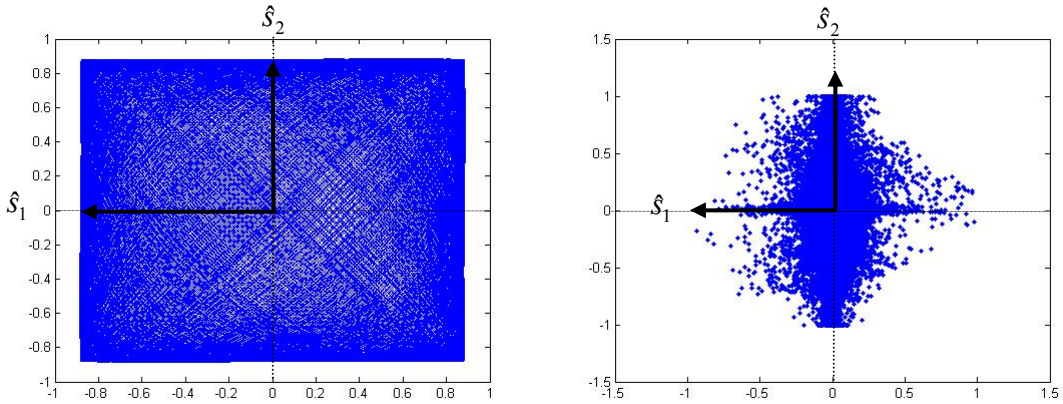


Fig. 2.6.2 Resultado de simulación del algoritmo de máxima no gaussianidad por negentropía de punto fijo.

La Figura 2.6.3, muestra el resultado de simulación de la salida de la red, una vez que ésta ha convergido al valor óptimo.

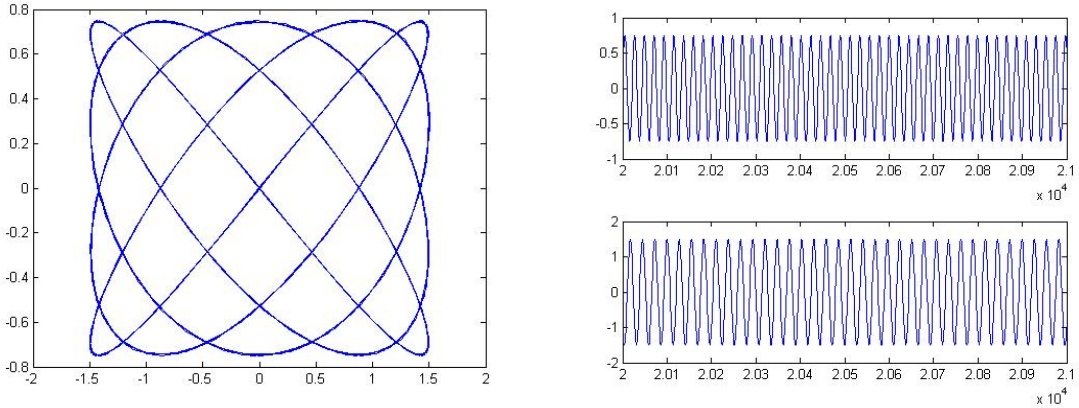


Fig. 2.6.3 Resultado de simulación de la salida de la red.

2.7 Conclusiones

En este capítulo, se mostraron las simulaciones en MATLAB de los algoritmos analizados en el capítulo 1. El primer algoritmo analizado fue el de H-J (ICA por descorrelación no lineal) [JUT91a], Este algoritmo es de tipo recursivo diseñado para el funcionamiento en línea y de manera paralela, en adecuado para la implementación en circuitos eléctricos (Analógicos o digitales). Sin embargo, presenta muy poca precisión en el cálculo de los coeficientes de la matriz de pesos. El error puede llegar a tomar valores del 20%, bajo ciertas características de algunas señales reales. No obstante, para algunas aplicaciones es adecuando.

Otro algoritmo simulado fue ICA por INFOMAX [WON00a]. Al igual que la red H-J, está diseñado para operar en línea y de forma recursiva, presentando una mayor complejidad para el cálculo de los coeficientes. Este algoritmo presenta muy buenos resultados al presentarle señales reales. No obstante, la implementación en programa computacional, requiere de muchos recursos para el cálculo de cada componente [HYV98]. El ajuste de este algoritmo, se realiza mediante la razón de aprendizaje (η), y se ve afectado, de acuerdo al número de entradas y a las características de las señales utilizadas. El valor utilizado para dos fuentes senoidales, fue de $\eta = 0.0004$ y en algunos casos para señales reales fue de $1x10^{-5}$, al aumentar el número de entradas al doble, se observaba que el factor η tenia que reducirse aproximadamente a la mitad, aunque depende de la cantidad de datos y de las señales utilizadas.

Los siguientes algoritmos analizados, fueron los de maximización de la no gaussianidad, divididos en dos etapas. La primera, efectúa el blanqueado de las señales. En este punto se mostró la implementación en arquitectura y en programación de los algoritmos de blanqueado, así como los resultados obtenidos, los cuales mostraron buen desempeño para ambas implementaciones. En la segunda parte, se realizó la medida de la no gaussianidad mediante la programación y la implementación de las arquitecturas. Los resultados obtenidos, muestran que los algoritmos basados en el gradiente, presentan mayor error que los algoritmos de punto fijo (implementado en programación de MATLAB y Simulink). No obstante, aquellos basados en la curtosis presentan algunas desventajas [HYV03:182], con

respecto al de negentropía, como la sensibilidad a los *outliers*, por lo que será este algoritmo con el que se analizara ICA en el siguiente capítulo.

Cabe mencionar que las pruebas realizadas, se llevaron a cabo utilizando dos señales senoidales. En el capítulo 3, se muestran algunas aplicaciones realizando pruebas con señales reales.

CAPÍTULO 3

Aplicaciones e implementaciones de los algoritmos ICA

3.1 Introducción

En la actualidad, la implementación de arquitecturas de redes neuronales ha tenido un gran auge debido a su buen desempeño. En este trabajo de tesis, se muestra la implementación en hardware (operando en tiempo real) de los algoritmos ICA en dispositivos programables como FPGA (*Field Programmable Gate Array*) y DSP (*Digital Signal Processor*). También, se muestra la implementación en software (operando fuera de línea) mediante Matlab del algoritmo ICA por negentropía de punto fijo, aplicado a la separación de señales EEG (*electroencefalográficas*). Cada una de estas implementaciones permite la visualización de los datos por diferentes medios, identificando aquellos que provienen de una sóla fuente (IC- Componente Independiente). En el caso de las implementaciones en hardware, los datos pueden observarse mediante el uso del osciloscopio (en algunos experimentos, solo se muestra la visualización con el simulador del hardware). La implementación en software, permite la visualización de los datos mediante la graficación en el monitor de la computadora. Esta última aplicación también permite la impresión en papel para ayudar al especialista en la toma de decisiones.

En este capítulo, se da una explicación breve de los dos dispositivos utilizados (FPGA y DPS). Posteriormente, se desarrolla la implementación de los algoritmos, cuyas arquitecturas, fueron mostradas en el capítulo 2. Y finalmente, se muestra la respuesta obtenida en tiempo real de la salida de la red. Para esto, se utilizaron dos archivos de audio con una razón de muestreo de 44100Hz durante 3 segundos.

Los algoritmos implementados en hardware, fueron el H-J e INFOMAX, y el algoritmo implementado por software, fue el de ICA de punto fijo por negentropía. Las pruebas realizadas a este último algoritmo, muestran doce señales mezcladas y logrando separar hasta 18 señales de entrada. A diferencia de las anteriores implementaciones, la separación no se obtiene en tiempo real.

3.2 Descripción general del FPGA

Una herramienta utilizada en este trabajo, es el FPGA *(Field Programable Gate Array)*. Un FPGA es un a arreglo de bloques lógicos programables colocados en una infraestructura de interconexiones programable; es posible programar la funcionalidad de los bloques lógicos, las interconexiones entre bloques y las conexiones entre entradas y salidas. Un FPGA es programable a nivel de hardware, por lo que proporciona las ventajas de un procesador de propósito general y un circuito especializado. Los sistemas basados en FPGA proporcionan un mejor desempeño que sus correspondientes implementaciones en software. Para ello el fabricante proporciona las herramientas de diseño adecuadas.

Los elementos básicos constituyentes de un FPGA, específicamente los de la familia de Xilinx, se pueden ver en la Figura 3.2.1.

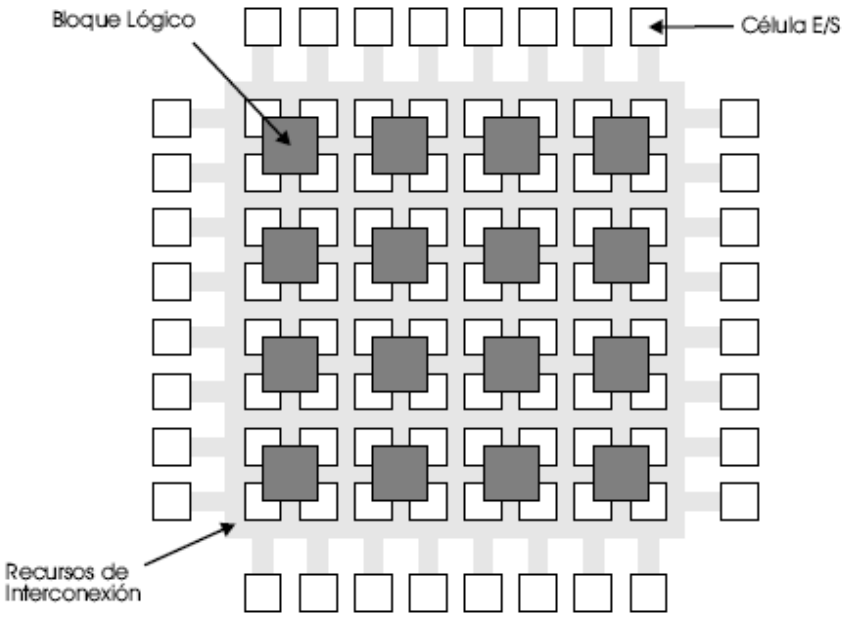


Fig. 3.2.1 Estructura general de un FPGA.

Esta constituido de los *bloques lógicos*, cuya estructura y contenido se denomina arquitectura. Hay muchos tipos de arquitecturas, que varían principalmente en complejidad (desde una simple compuerta hasta módulos más complejos o estructuras tipo PLD). Suelen

incluir compuertas biestables para facilitar la implementación de circuitos secuenciales. Otros módulos de importancia son los bloques de *entrada/salida*, que permiten la conexión del exterior con los bloques programables mediante canales de conexión vertical y horizontal. Y por último, los r*ecursos de interconexión*, cuya estructura y contenido se denomina arquitectura de ruteado.

Entre las numerosas ventajas que proporciona el uso de un FPGA, dos destacan principalmente: el bajo costo del prototipo y el corto tiempo de producción.

El HDL (*Hardware Description Language*), es un lenguaje de programación usado para modelar en software el comportamiento de un dispositivo programable (hardware), como el FPGA. Existen dos aspectos en la descripción de hardware, que el HDL facilita: una abstracción del comportamiento del modelo y un modelado estructurado del hardware.

Abstracción del comportamiento del modelo. En este caso, el HDL es declarativo, para facilitar la abstracción de la descripción del comportamiento del hardware para propósitos específicos. Este comportamiento no está relacionado por aspectos de diseño o estructurales del hardware.

Modelado estructurado del hardware. El modelado estructurado, consiste en el diseño a bloques de la estructura interna de los dispositivos digitales del hardware, el cual puede ser modelado en HDL, facilitando la comprensión superficial de algunos aspectos del lenguaje, pero dificulta el entendimiento de la dimensión temporal del mismo. Este tipo de modelado, es independiente del comportamiento del diseño.

El comportamiento del hardware puede ser modelado y representado en varios niveles de abstracción durante el diseño del proceso. Los modelos de alto nivel describen la operación de hardware de una forma abstracta, mientras que los de bajo nivel incluyen mayor detalle como las estructuras de hardware empleadas [SMI01, PER02, CHA03].

Una vez descrita en forma general la estructura del FPGA, así como el lenguaje de programación HDL, se procederá a mostrar la implementación de los algoritmos de ICA.

3.2.1 Implementación del algoritmo H-J en FPGA

Esta implementación, separa las señales en tiempo real de dos fuentes audibles utilizando el algoritmo H-J (mostrado en el capítulo anterior). Esta implementación, fue dividida en tres etapas; la primera que constituye un convertidor Analógico/Digital (ADCS7476MSPS 12bit A/D), el cual recibe las señales analógicas de entrada. Posteriormente, se encuentra el FPGA (Spartan 3 de la familia de Xilinx) que implementa el algoritmo ICA y finalmente de un convertidor Digital/Analógico (DAC121S101 12bit A/D), que devuelva la respuesta en forma analógica de los datos procesados.

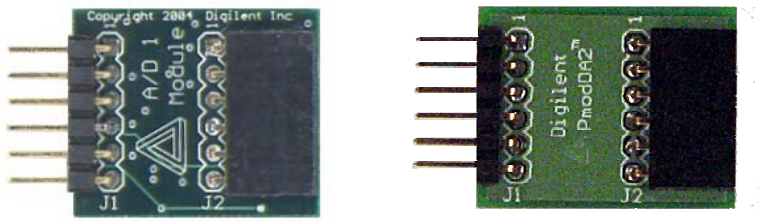


Fig. 3.2.2 Convertidor A/D y D/A, respectivamente.

Es importante señalar que el tamaño de palabra que se utilizó en el diseño de esta implementación, fue de 10 bits, es decir, 9 bits de datos mas 1 bit de signo (despreciando los bits menos significativos para facilitar el diseño de las funciones no lineales y evitar ruido). Los valores, se estandarizaron al rango de ± 0.5V, obteniendo un intervalo de 1V, al dividir entre 1024 se obtuvo una resolución de una milésima por cada bit. De modo que un número puede ser representado de la forma

$$z \times 10^{-3}$$

Al multiplicar dos números (a y b), de la forma $z \times 10^{-3}$, se obtiene el resultado de la forma $ab \times 10^{-6}$. Por lo tanto, si se divide entre 1000, se obtiene un número de la forma de z. Una forma fácil de realizar esta división es haciendo un corrimiento de 10 bits.

$$\frac{(ab \times 10^{-6})}{10^{3}} 10^{3} = \frac{(ab \times 10^{-3})}{10^{3}}$$

Algunos números normalizados con esta técnica, se presentan en la función cúbica de la tabla 3.1. La función tangente hiperbólica, fue asignada mediante una tabla de búsqueda.

Tabla 3.1 Resultados obtenidos de las funciones no lineales.

Normalizado	Cúbico	Tanh	Decimal
0,490	0,118	0,454	501
0,489	0,117	0,454	500
0,488	0,116	0,453	499
0,487	0,116	0,452	498
0,004	0,000	0,004	4
0,003	0,000	0,003	3
0,002	0,000	0,002	2
0,001	0,000	0,001	1
0,000	0,000	0,000	0
0,000	0,000	0,000	1023
-0,001	0,000	-0,001	1022
-0,002	0,000	-0,002	1021
-0,003	0,000	-0,003	1020
-0,004	0,000	-0,004	1019
-0,005	0,000	-0,005	1018
-0,006	0,000	-0,006	1017
-0,007	0,000	-0,007	1016
-0,008	0,000	-0,008	1015
-0,485	-0,114	-0,451	527

Una vez presentado el diseño de las funciones no lineales, se muestra el código en VHDL de la red neuronal y su regla de aprendizaje. La implementación de la arquitectura de la red está constituida de un sumador, un multiplicador y su función de transferencia (el multiplicador se incluye en la etapa del actualizador).

Sumador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Sumador is
   Port ( eta : in std_logic_vector(9 downto 0);
          clk : in std_logic;
          Wn : out std_logic_vector(9 downto 0));
end Sumador;

architecture Behavioral of Sumador is
signal auxiliar : std_logic_vector (9 downto 0) := (others => '0');
begin
   Wn <= auxiliar;
```

```
   process (clk) begin
              if clk'event and clk = '1' then
                        auxiliar <= auxiliar + eta;
                       end if;
            end process;
end Behavioral;
```

Función de Transferencia

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Transferencia is
   Port ( a : out std_logic_vector(9 downto 0);
        n : in std_logic_vector(9 downto 0);
        clk : in std_logic);
end Transferencia;

architecture Behavioral of Transferencia is
signal aux : std_logic_vector(9 downto 0) := (others => '0');
begin
  a <= aux;
  process (clk) begin
                        if clk = '1' and clk'event then
                          aux <= n;
                        end if;
            end process;
end Behavioral;
```

El siguiente código de VHDL, muestra la implementación de la regla de aprendizaje, de la red H-J.

Función f(x) y g(x)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Funcion_F is Port(
           a : in STD_LOGIC_VECTOR (9 DOWNTO 0);
           f : out  STD_LOGIC_VECTOR (9  DOWNTO 0));
end Funcion_F;

architecture Behavioral of Funcion_F is
type ram_type is array (1023 downto 0) of std_logic_vector (9  downto 0);
signal memory: ram_type;
signal read_a : integer range 0 to 1023 := 0;
begin
           memory(0)   <= "0000000000";    memory(1) <= "0000000000";
           memory(2)   <= "0000000000";    memory(3) <= "0000000000";
           memory(4)   <= "0000000000";    memory(5) <= "0000000000";
```

```
        memory(6)   <= "0000000000";    memory(7) <= "0000000000";
        memory(8)   <= "0000000000";    memory(9) <= "0000000000";
        read_a <= CONV_INTEGER(a);
  f <= memory(read_a);
end Behavioral;
```

Actualizador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Actualizacion is
   Port ( f : in std_logic_vector(9 downto 0);
          g : in std_logic_vector(9 downto 0);
          eta : out std_logic_vector(9 downto 0));
end Actualizacion;

architecture Behavioral of Actualizacion is
signal signo : std_logic;
signal aux1, aux2,aux3 : std_logic_vector (9 downto 0);
signal resultado1 : std_logic_vector(19 downto 0);
constant divisor : std_logic_vector(2 downto 0):= "101";

begin
aux1 <= (not(f) + 1) when f(9) = '1' else f;
aux2 <= (not(g) + 1) when g(9) = '1' else g;
signo <= f(9) xor g(9);

resultado1 <= aux1 * aux2;

aux3 <= (resultado1(19 downto 10)) when signo = '0' else
            (not(resultado1(19 downto 10)) + 1) when signo = '1' else
            (others => '0');

eta(9 downto 5) <= (others => signo);
eta(4 downto 0) <= aux3(9 downto 5);
end Behavioral;
```

Algunas implementaciones de su contraparte analógica, se pueden encontrar en [LNO06] [MAR92]. El siguiente punto, muestra los resultados experimentales obtenidos de esta implementación.

3.2.2 Resultados de la red H-J (FPGA)

Los siguientes resultados de simulación, fueron obtenidos por la implementación del código VHDL de la red H-J. Cabe mencionar que sólo se muestra el desempeño de la red con señales reales y no una caracterización de éstos.

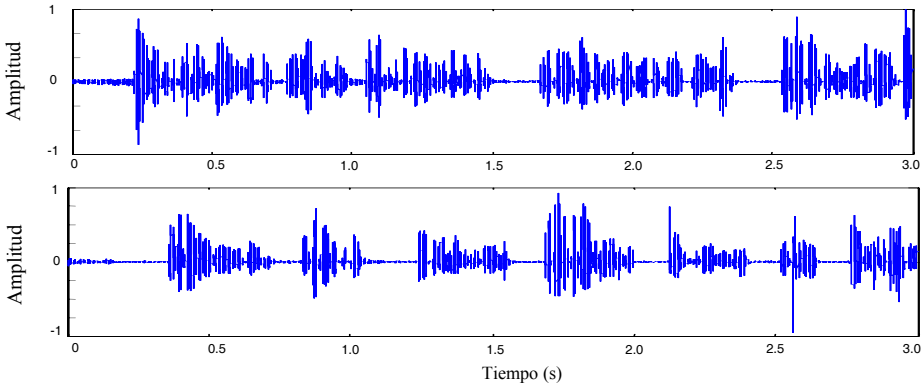


Fig. 3.2.3 Señales de audio de dos fuentes independientes.

Como se enunció, se utilizaron dos señales de audio (figura 3.2.3), las cuales se mezclaron y se introdujeron al sistema. La Figura 3.2.4 muestra las señales de mezcla.

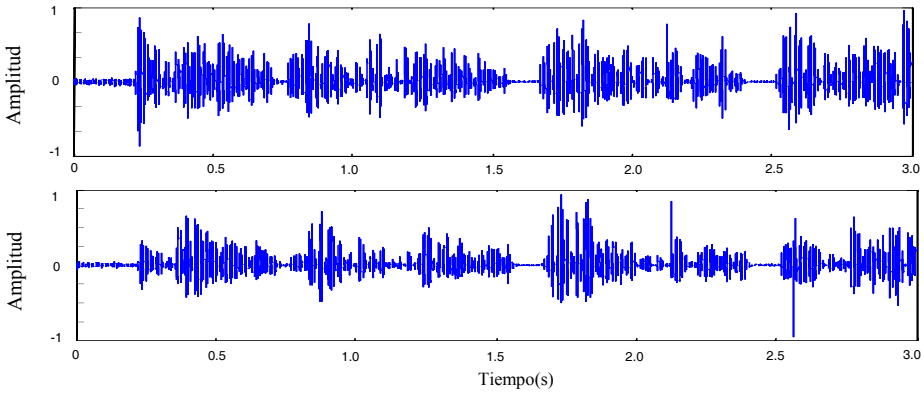


Fig. 3.2.4 Señales mezcladas de las fuentes originales.

La matriz de mezcla utilizada fue: $\mathbf{A} = \begin{bmatrix} 1.0 & 0.8 \\ 0.3 & 1.0 \end{bmatrix}$

Por último, la Figura 3.2.5 muestra la señal de salida generada por la red.

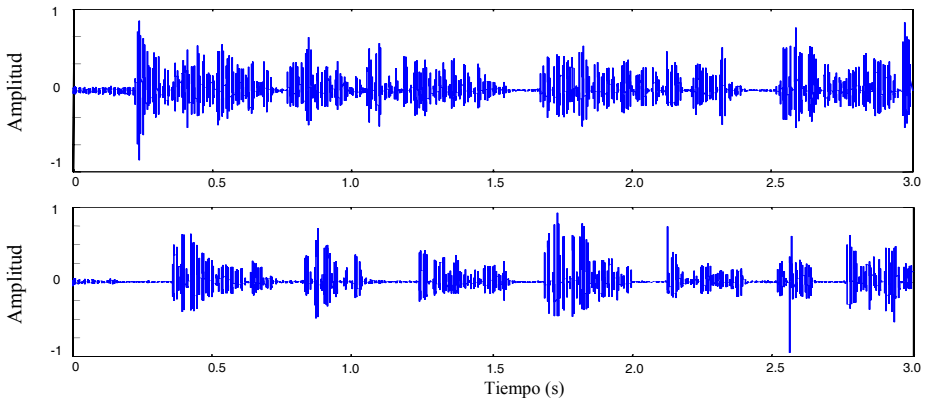


Fig. 3.2.5 Simulación de la salida de la red H-J implementada en FPGA.

3.2.3 Implementación del algoritmo INFOMAX en FPGA

Retomando la arquitectura de la red neuronal antes presentada, la implementación de la red en el FPGA se presenta de la siguiente forma.

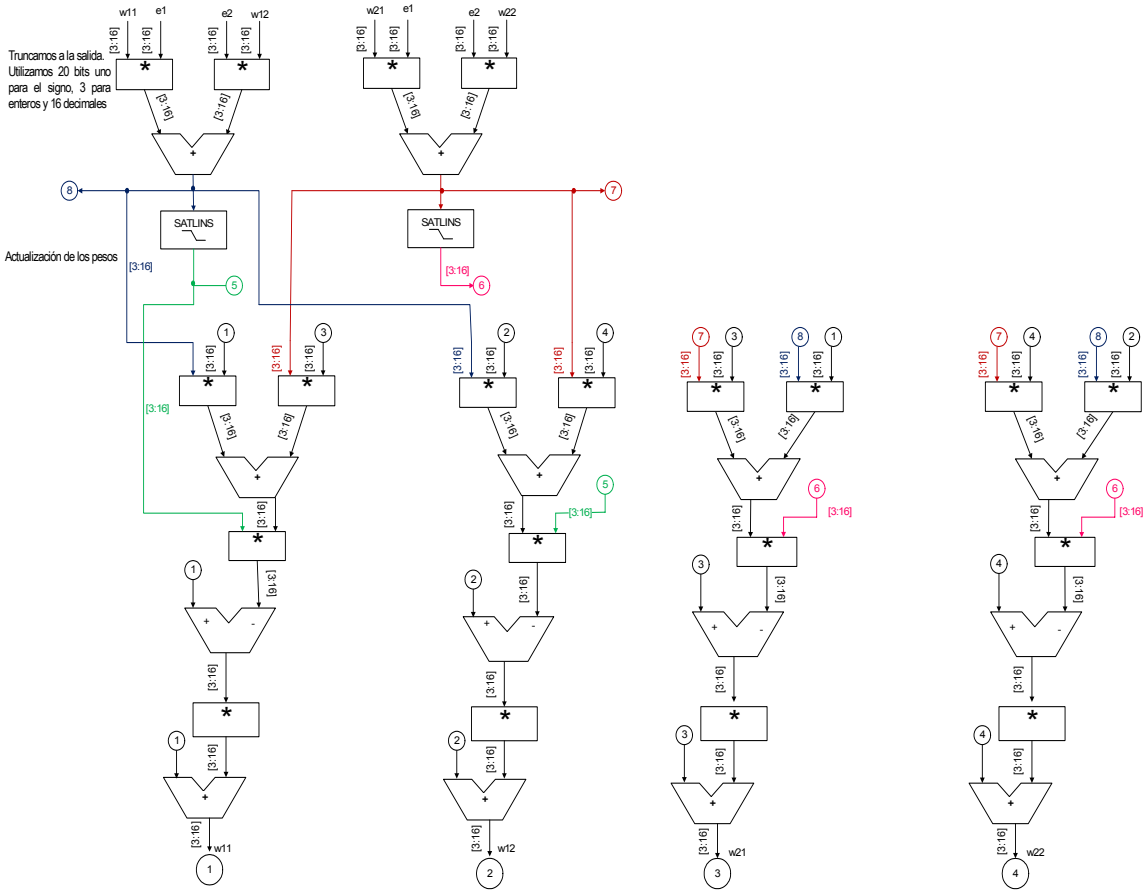
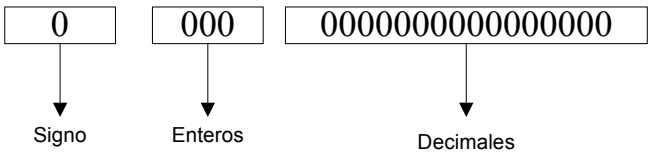


Fig. 3.2.6 Diagrama a Bloques de la Red INFOMAX

En la Figura 3.2.6 se muestran los bloques que son implementados en la Red INFOMAX en código VHDL. Los datos que se utilizan para la implementación provienen del convertidor AD de 12 bits utilizado para la implementación del algoritmo H-J. No obstante, la arquitectura fue diseñada para operar hasta 14 bits de resolución, mientras que internamente la red opera con 20bits. El formato de los bits tomándolos de izquierda a derecha es:



Los cuales tienen un formato en punto fijo.

Por su complejidad, la función tangente hiperbólica que se utiliza en el algoritmo se remplaza por una función satlins. Esta función puede considerarse como una aproximación de la tanh, como se muestran en las siguientes gráficas.

$$a = \frac{e^{n} - e^{-e}}{e^{n} + e^{-n}}$$

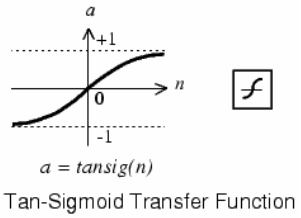


Fig. 3.2.7 Función tangente hiperbólica (tanh)

$$a = -1 \qquad n < -1$$
$$a = n \qquad -1 \le n \le 1$$
$$a = +1 \qquad n > 1$$

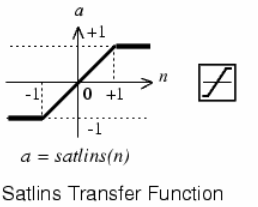


Fig. 3.2.8 Lineal saturado simétrico (satlins)

A continuación se muestra el código utilizado en la implementación.

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------------------------
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
----------------------------------------------------------------------------------
entity Bloque is Port (
```

```
  E1 : in sfixed (3 DOWNTO -16); --
  E2 : in sfixed (3 DOWNTO -16); --
  OSC_CLK : in std_logic;
  RST : in std_logic;
  r11, r12, r21, r22, Salida_1, Salida_2 : out real);
--  Salida_1, Salida_2 : out sfixed (3 DOWNTO -16));
end Bloque;
---------------------------------------------------------------------------------
architecture RTL of Bloque is

  Signal E1pf, E2pf : sfixed (3 DOWNTO -16) := "0000000000000000000";
  Signal U1, U2, Sat_01, Sat_02 : sfixed (3 DOWNTO -18) := "000000000000000000000";
  Signal Cpf : sfixed (3 DOWNTO -19) := "0000000000000000000000";
  Signal Sum_X1, Sum_X2: real := 0.0;
--  Signal Sum_X1, Sum_X2, r11, r12, r21, r22: real := 0.0;
  --Signal w11, w21, w12, w22 : sfixed (3 DOWNTO -16):= "0000000000000000000";  -- Pesos
  Signal aw11, aw21, aw12, aw22 : sfixed (6 DOWNTO -18):= "000000000000000000000000";  -- Pesos
Siguientes
--  Constant Const : real := 0.0000152587890625; -- 32e-5 = "0000000000000010101" = X"00015"
  Constant Const : real := 0.0000057220458984375;
  --Constant Cpf : sfixed (3 DOWNTO -16) := "0000000000000010101"; -- 32e-5
  --Constant Cpf : sfixed (3 DOWNTO -16) := "0000000000001010100"; -- 0.00128
  --Constant Cpf : sfixed (3 DOWNTO -16) := "0000000000010000000"; -- 0.03125
  Signal w11 : sfixed (6 DOWNTO -18) := "000000100000000000000000000"; -- 1.0 = X"10000"
  Signal w12 : sfixed (6 DOWNTO -18) := "000000000000000000000000000"; -- 0.0 = X"00000"
  Signal w21 : sfixed (6 DOWNTO -18) := "000000000000000000000000000"; -- 0.0
  Signal w22 : sfixed (6 DOWNTO -18) := "000000100000000000000000000"; -- 1.0
  SIGNAL CLK : STD_LOGIC := '0';
  SIGNAL CONT : INTEGER RANGE 0 TO 1;
  SIGNAL enb: STD_LOGIC := '0';

begin
  -- Coversion a Punto Fijo del
  -- Valor de la Constante
  Cpf  <= to_sfixed (Const, Cpf);                    -- Constante
-------------------------------------------------------------------------------------
  DIVISOR : PROCESS ( OSC_CLK, RST )
  BEGIN
    IF ( RST = '1' ) THEN
                        CONT <= 0;
--        w11 <= dw11; -- 1.0
--        w21 <= dw21; -- 0.0
--        w12 <= dw12; -- 1.0
--        w22 <= dw22; -- 0.0
            ELSIF ( OSC_CLK'EVENT AND OSC_CLK = '1' ) THEN
                                CONT <= CONT + 1;
                                IF ( CONT = 1 ) THEN
                                                CLK <= NOT CLK;
                                                CONT <= 0;
                                END IF;
            END IF;
  END PROCESS DIVISOR;
-------------------------------------------------------------------------------------
  ADQUISICION : PROCESS ( CLK, RST )
  BEGIN
    IF ( RST = '1' ) THEN
```

```
                E1pf <= "0000000000000000000";
                E2pf <= "0000000000000000000";
        ELSIF ( CLK'EVENT AND CLK = '1' ) THEN
      -- Coversion a Punto Fijo de los
      -- Valores de las Entradas
      E1pf <= E1;                       -- Entrada 1
      E2pf <= E2;                       -- Entrada 2
      --enb <= NOT enb;
        END IF;
 END PROCESS ADQUISICION;
-------------------------------------------------------------------------------
 CALCULO_US : PROCESS ( E1pf, E2pf )
 BEGIN
        U1 <= resize (3 * ((E1pf * w11) + (E2pf * w12)), U1);
        U2 <= resize (3 * ((E1pf * w21) + (E2pf * w22)), U2);
 END PROCESS CALCULO_US;
-------------------------------------------------------------------------------
 SATLINS : PROCESS ( U1, U2 )
 BEGIN
        -- Implementacion de la Funcion Satlins 1

        Sum_X1 <= to_real (U1);
        if (U1 >= 1.0) then -- >= 1.0
                Sat_01 <= "00010000000000000000";
        elsif (U1 <= -1.0) then -- <= -1.0
                Sat_01 <= "11110000000000000000";
        else
                Sat_01 <= U1;
        end if;

   Sum_X2 <= to_real (U2);
        if (U2 >= 1.0) then -- >= 1.0
                Sat_02 <= "00010000000000000000";
        elsif (U2 <= -1.0) then -- <= -1.0
                Sat_02 <= "11110000000000000000";
        else
                Sat_02 <= U2;
        end if;

--      Sum_X1 <= to_real (U1);
--      if (Sum_X1 >= 1.0) then -- >= 1.0
--              Sat_01 <= "00010000000000000000";
--      elsif (Sum_X1 <= -1.0) then -- <= -1.0
--              Sat_01 <= "11110000000000000000";
--      else
--              Sat_01 <= U1;
--      end if;
--      -- Implementacion de la Funcion Satlins 2
--      Sum_X2 <= to_real (U2);
--      if (Sum_X2 >= 1.0) then -- >= 1.0
--              Sat_02 <= "00010000000000000000";
--      elsif (Sum_X2 <= -1.0) then -- <= -1.0
--              Sat_02 <= "11110000000000000000";
--      else
--              Sat_02 <= U2;
--      end if;
```

```
  END PROCESS SATLINS;
-------------------------------------------------------------------------------
  CALC_PESOS: PROCESS (Sat_01, Sat_02) is
  BEGIN
    -- W11
          aw11 <= resize (w11 + ((w11 - (((w11 * U1) + (w12 * U2)) * Sat_01)) * Cpf), w11);

    -- W21
          aw21 <= resize (w21 + ((w21 - (((w21 * U1) + (w22 * U2)) * Sat_01)) * Cpf), w21);

    -- W12
          aw12 <= resize (w12 + ((w12 - (((w12 * U2) + (w11 * U1)) * Sat_02)) * Cpf), w12);

    -- W22
          aw22 <= resize (w22 + ((w22 - (((w22 * U2) + (w21 * U1)) * Sat_02)) * Cpf), w22);
  END PROCESS CALC_PESOS;
-------------------------------------------------------------------------------
  ACTUALIZACION : PROCESS (clk, rst)
  BEGIN
    IF rst = '1' THEN
      w11 <= "00000010000000000000000000";
      w21 <= "00000000000000000000000000";
      w12 <= "00000000000000000000000000";
      w22 <= "00000010000000000000000000";
    ELSIF clk'event AND clk = '1' THEN
      --IF enb = '1' THEN
        w11 <= aw11;
        w21 <= aw21;
        w12 <= aw12;
        w22 <= aw22;
      --END IF;
    END IF;
  END PROCESS ACTUALIZACION;
-------------------------------------------------------------------------------
  -- Salidas del Sistema
  Salida_1 <= to_real (U1);
  Salida_2 <= to_real (U2);
  r11 <= to_real (w11);
  r12 <= to_real (w12);
  r21 <= to_real (w21);
  r22 <= to_real (w22);
--  Salida_1 <= Sat_01;
--  Salida_2 <= Sat_02;
end RTL;
```

La implementación del algoritmo INFOMAX también se ha realizado en forma analógica como se muestra en [SEO01]

3.2.4 Resultados del algoritmo INFOMAX (FPGA)

Los siguientes resultados de simulación, fueron obtenidos por la implementación del código VHDL de la red INFOMAX. Cabe mencionar que solo se muestra el desempeño de la red con señales reales y no una caracterización de éstos.

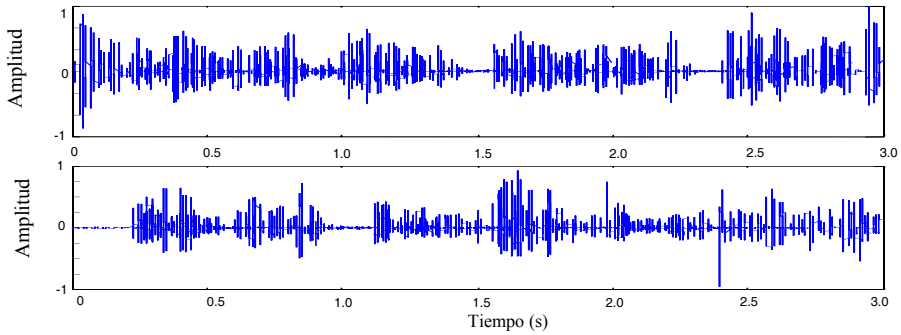


Fig. 3.2.9 Señales de audio de dos fuentes independientes.

Como se enunció, se utilizaron dos señales de audio (figura 3.2.9), las cuales se mezclaron y se introdujeron al sistema. La Figura 3.2.10 muestra las señales de mezcla.

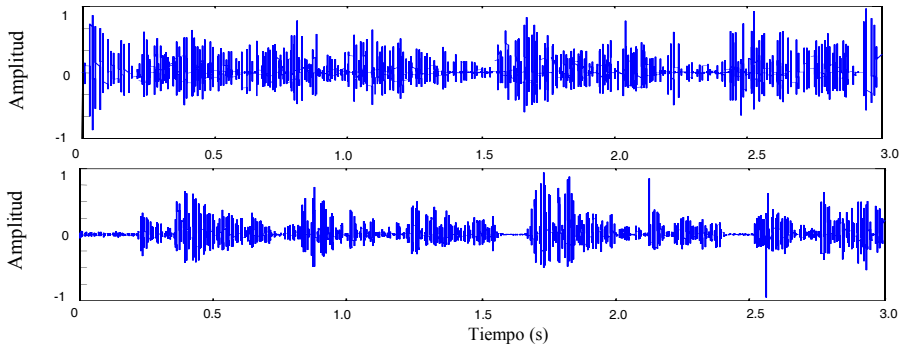


Fig. 3.2.10 Señales mezcladas de las fuentes originales.

Por último, la Figura 3.2.11 muestra la señal de salida generada por la red.

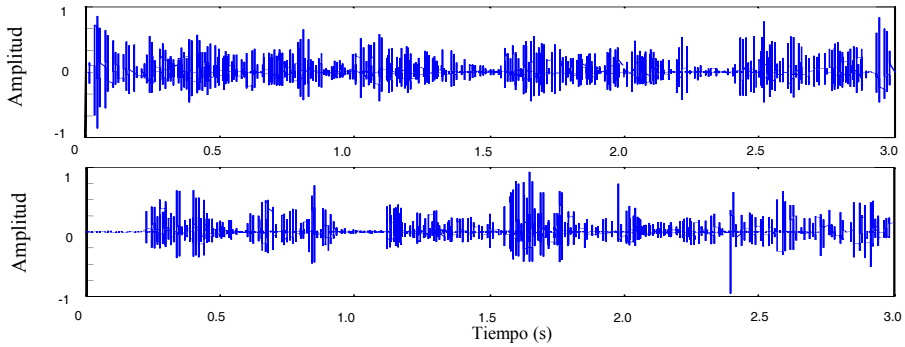


Fig. 3.2.11. Simulación de la salida de la red INFOMAX implementada en FPGA.

3.3. Descripción general del DSP

Un DSP (*Digital Signal Processor*), es un procesador de señales por medios digitales y su principal aplicación, es la de medir y filtrar señales analógicas del mundo real. Su primer paso es, convertir una señal analógica a una forma digital para efectuar el procesamiento. Una señal digital consiste de una cadena de números, usualmente (pero no necesariamente) en forma binaria. El procesamiento de la señal digital, se realiza mediante cálculos numéricos. Al igual que un procesador de propósito general, un DSP es un dispositivo programable, con su propio conjunto de instrucciones. Por último, su salida generalmente es convertida a una forma analógica.

Los chips DSP son capaces de realizar millones de operaciones de punto flotante por segundo, y como los microprocesadores normales, están siendo introducidas versiones más rápidas y poderosas continuamente. Los DSP pueden ser embebidos en complejos sistemas en un chip (SOC- *System on Chip*) con circuitería tanto analógica como digital.

Una de las aplicaciones principales del DSP es el filtrado y acondicionado de señales analógicas. Como ejemplo muy simple, un DSP se puede programar para: a) tomar una forma de onda analógica; b) como la salida de un preamplificador de audio y; c) pasar a la salida sólo los componentes de la frecuencia que estén debajo de una frecuencia determinada. Todas las frecuencias mayores las atenúa el filtro.

En la Figura 3.3.1, se muestra la arquitectura básica de un DSP. La sección de multiplicar y acumular es central para todos los DSP y se usa en la mayoría de las aplicaciones. La unidad aritmética lógica y el registro de desplazamiento proporcionan el soporte necesario para trabajar con el sistema binario mientras se procesan señales.

Otra aplicación útil del DSP es el filtrado por interpolación. La forma de onda reconstruida siempre es una aproximación de la original debido al error de cuantización. Los cambios de escalón repentinos de un punto de datos al siguiente también introducen ruido de alta frecuencia en la señal reconstruida.

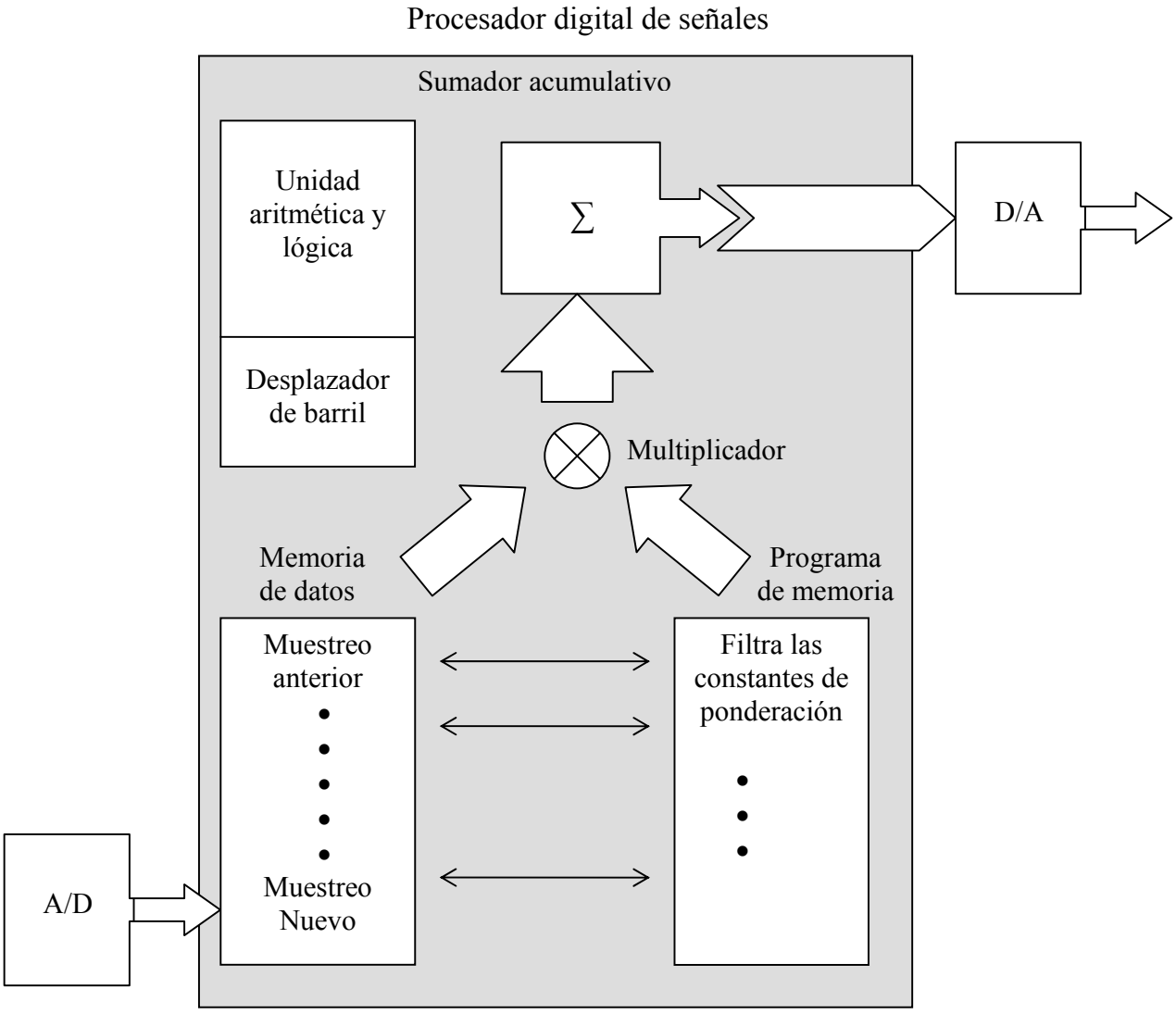


Fig. 3.3.1 Arquitectura básica de un DSP.

Los DSP's están integrados en muchos sistemas comunes, por ejemplo, reproductores de CD para filtrar la señal digital que está siendo leída a partir del disco, para minimizar el ruido de cuantización que es inevitablemente provocado por la digitalización de la música. Los sistemas de teléfonos usan el DSP para cancelar ecos en las líneas telefónicas. Los MODEM de alta velocidad son dispositivos estándar en la PC, que es posible obtener a un bajo costo mediante un DSP. Las cajas de efectos especiales para guitarras y otros instrumentos que realizan eco, reverberación, enfasamiento y otros efectos utilizan DSP. Aún cuando la teoría que enmarca las técnicas DSP, como transformadas de Hilbert y

Fourier, para el diseño de filtros digitales y la compresión de señales, que pueden ser extremadamente complejos. Las operaciones numéricas requeridas para implementar estas técnicas de hecho son muy simples y consisten principalmente de operaciones que podrían ser hechas por una calculadora de cuatro operaciones básicas. La arquitectura de un DSP está diseñada para resolver estas operaciones increíblemente rápidas, procesando cientos de millones de muestras cada segundo, para proveer un desempeño en tiempo real: esto es, la habilidad de procesar una señal "viva" al mismo tiempo que es muestreada para después sacar la señal procesada, por ejemplo a una bocina o pantalla de video.[TI05].

3.3.1 Implementación del algoritmo H-J en DSP

Esta implementación, utiliza el algoritmo H-J mostrado en los capítulos anteriores y hace uso del DSP TMS320C6713-DSK de Texas Instrument (Figura 3.3.2), el cual cuenta con convertidor Analógico/Digital y Digital/Analógico de dos canales. Las herramientas utilizadas para la programación, fueron el Code Composer Studio y Simulink de Matlab.

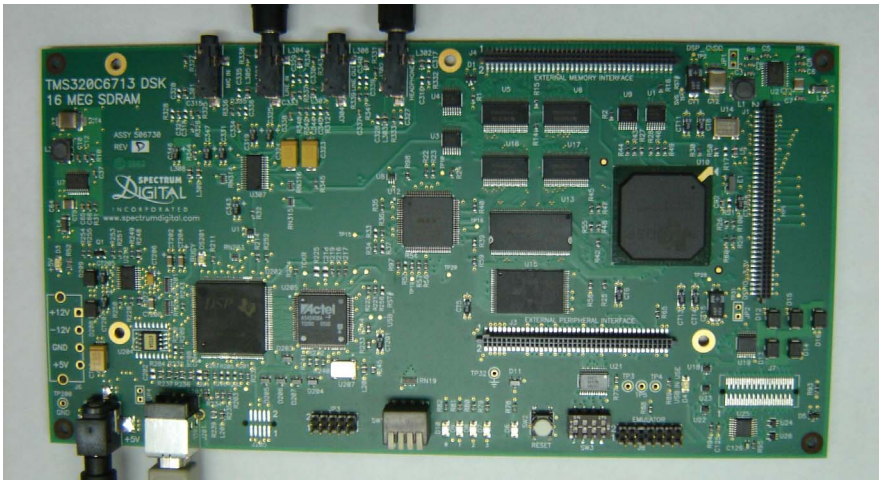
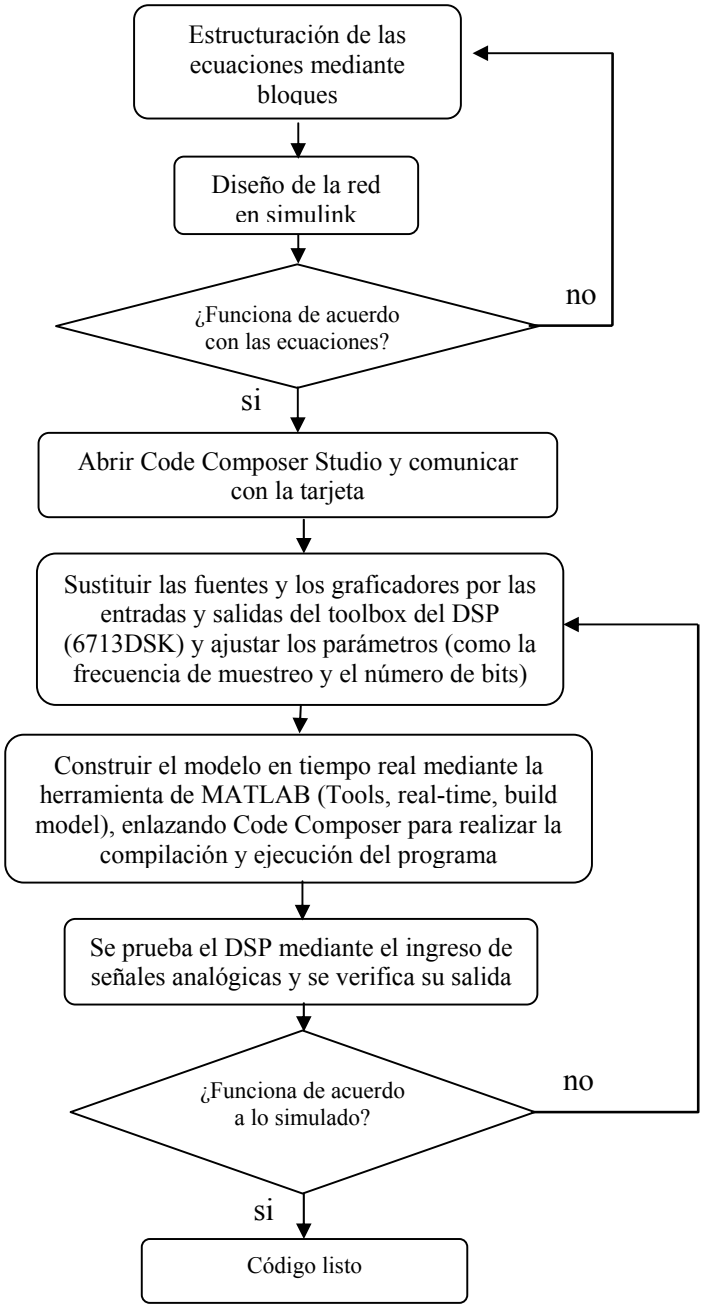


Fig. 3.3.2 DSP TMS320C6713-DSK de Texas Instrument.

El siguiente diagrama de flujo muestra la secuencia para la implementación del algoritmo en el DSP, empleando Simulink.



Para la implementación, fue necesario verificar el funcionamiento en Simulink mediante la construcción de la arquitectura de la red y su regla de aprendizaje por medio de bloques. Una vez que fue comprobado su funcionamiento, se utiliza la herramienta de Matlab para, generar el código correspondiente de cada bloque y exportarlo al Code Composer.

Una vez exportado al Code Composer, se compila y se descarga el programa hacia la tarjeta, posteriormente se efectúa la ejecución del mismo. El siguiente punto muestra los resultados obtenidos por la implementación del algoritmo.

3.3.2 Resultados de la red H-J (DSP)

La figura 3.3.3, muestran dos señales de audio (a una razón de muestreo de 44100Hz durante 3 segundos)

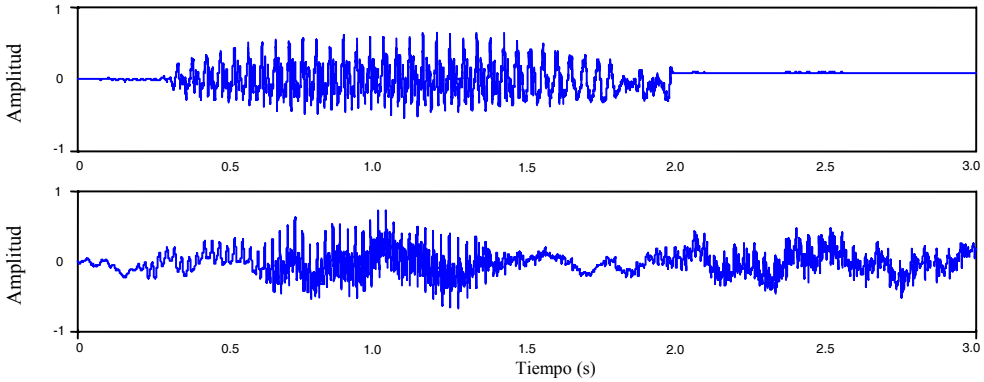


Fig. 3.3.3 Señales de dos fuentes de audio independientes.

La matriz de mezcla utilizada es: $\mathbf{A} = \begin{bmatrix} 1.0 & 0.8 \\ 0.3 & 1.0 \end{bmatrix}$

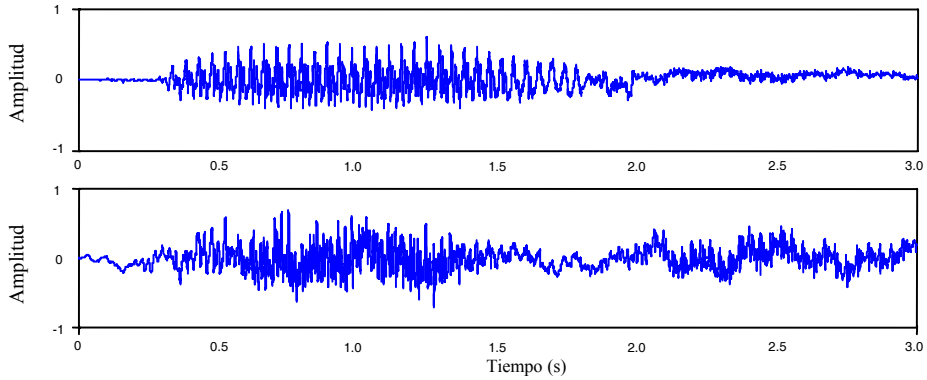


Fig. 3.3.4 Señales mezcladas de las fuentes originales.

y los resultados de simulación obtenidos por la implementación del algoritmo H-J.

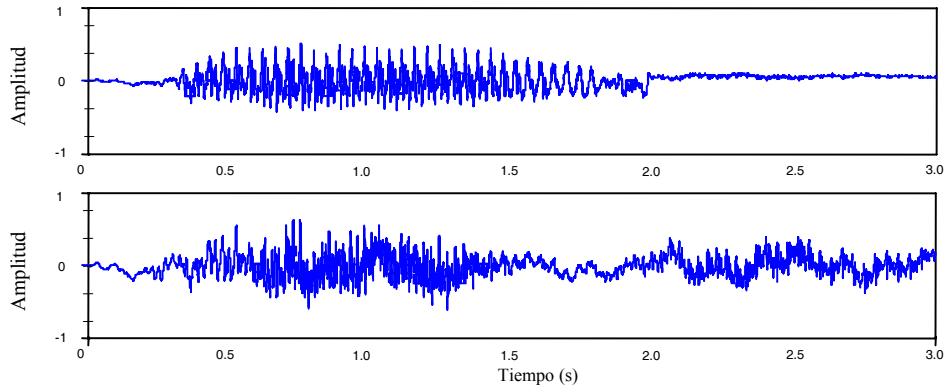


Fig.3.3.5 Simulación de las señales de salida de la red H-J, implementada en DSP.

3.3.3 Implementación del algoritmo INFOMAX en DSP

Esta implementación, separa la señal de dos fuentes audibles, utilizando el algoritmo INFOMAX (mostrado en los capítulos anteriores). Para la implementación, fue necesario verificar el funcionamiento en Simulink, mediante la construcción de la arquitectura por medio de bloques. Una vez que fue comprobado su funcionamiento, se utilizó MATLAB para hacer la compilación y generar el código correspondiente.

3.3.4 Resultados de INFOMAX (DSP)

Se utilizaron las señales de audio mostradas en la Figura 3.3.6 Mostrando los resultados obtenidos por el osciloscopio en la Figura 3.3.8.

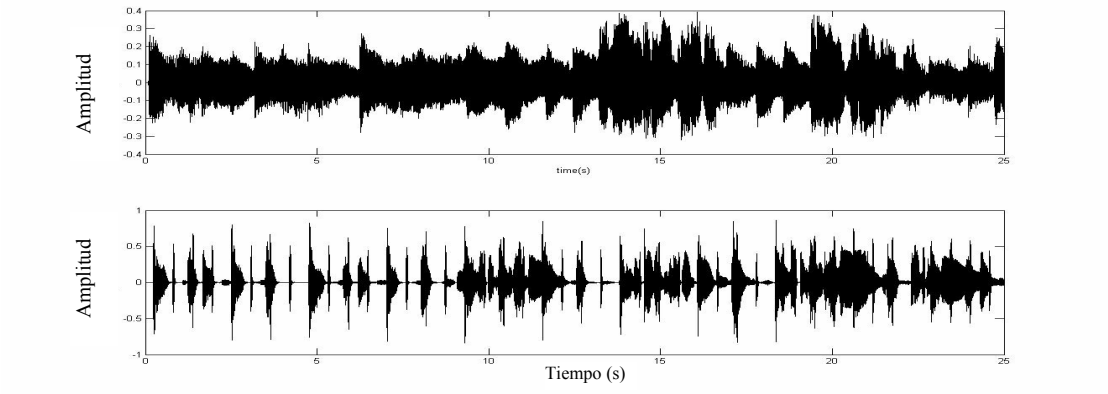


Fig. 3.3.6 Señales de dos fuentes independientes

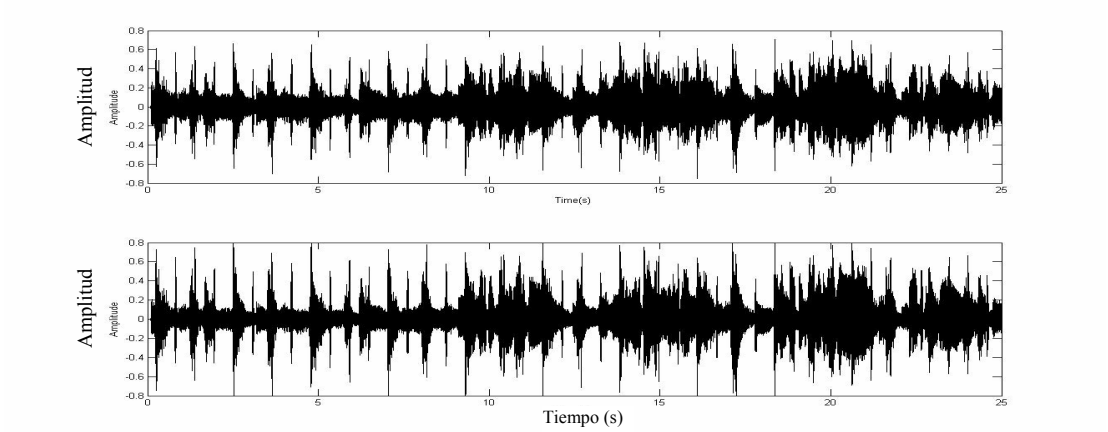


Fig. 3.3.7 Señales mezcladas de las originales.

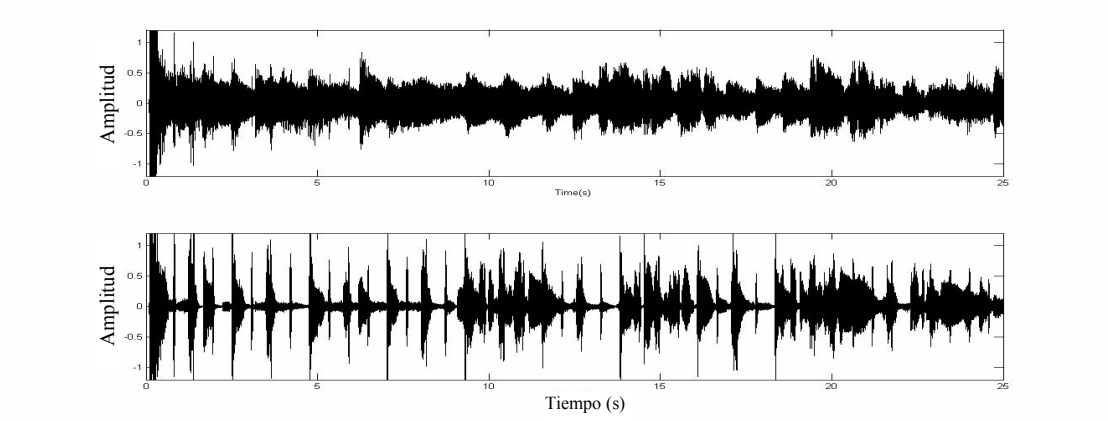


Fig.3.3.8 Señales de salida del algoritmo INFOMAX, implementado en DSP.

En la gráfica anterior, se muestra la respuesta de salida de la red, en la cual se puede observar que corresponden con las señales originales mostradas en la figura 3.3.6.

3.4 Implementación de ICA por el método de la negentropía

En este punto, se propuso el diseño de un navegador de señales electroencefalográficas (EEG - *electroencephalograph*) que permita la visualización de las señales electroencefalográficas para el diagnostico de algunas enfermedades. Algunos de estos navegadores (Harmonie Signal File Browser), hacen uso de herramientas como el filtrado de las señales, aislando cada una de éstas. FastICA de punto fijo se propuso como método de separación de estas señales, desarrollando un software prototipo de un navegador EEG, el cual permite separar cada una de las componentes independientes de archivos clínicos de

EEG. La figura 3.4.1, muestra la pantalla principal del sistema desarrollado. El código de este navegador, se anexan en el archivo *navegadorEEG.fig*.

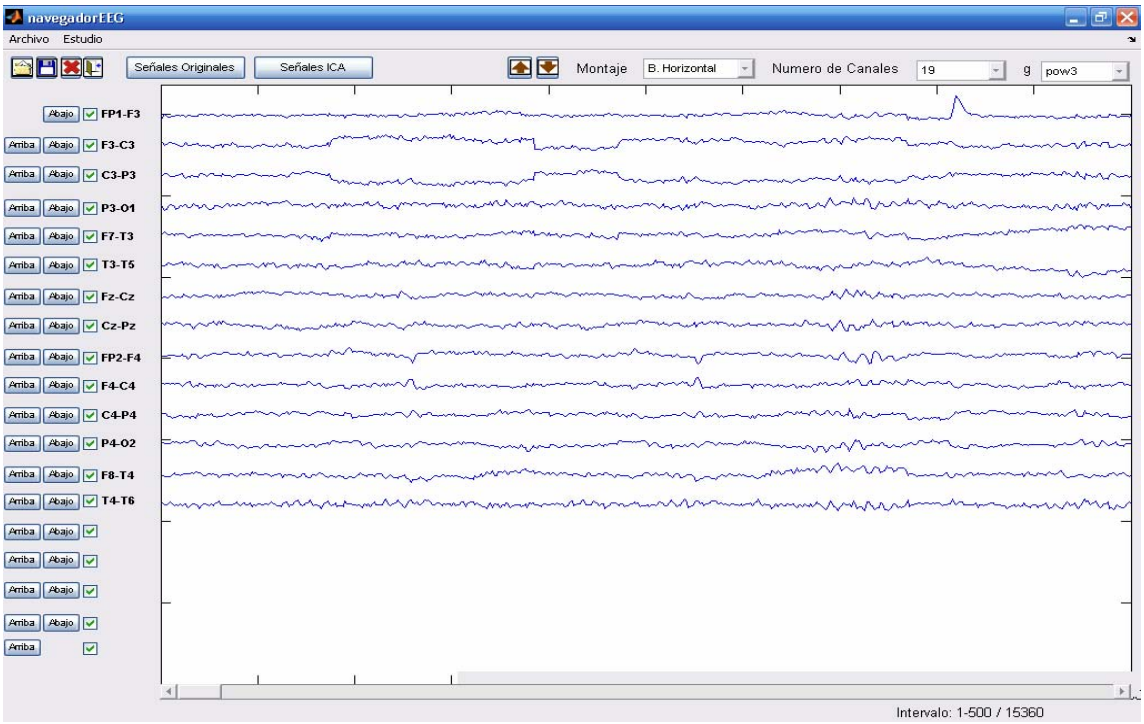


Fig. 3.4.1 Pantalla principal del sistema.

Para comprobar el funcionamiento del algoritmo, se aplicaron diferentes tipos de señales, las cuales se muestran en el siguiente punto.

3.4.1 Resultados de simulación

Para verificar el desempeño de la red, se generaron doce señales independientes; las cuales se muestras en la Figura 3.4.2. La matriz de mezcla utilizada, fue generada aleatoriamente y se muestra a continuación:

-0.6034	-1.7179	0.0253	0.9900	0.1701	0.6065	-0.2211	-0.7412	-0.8578	1.1773	-0.5918	3.4128
1.8336	-0.0539	0.2615	-1.7817	-0.4958	0.0742	-1.6758	1.0341	-0.7716	0.2257	2.2496	-0.3947
-0.9653	0.2179	0.7435	-0.0440	1.2027	1.0805	-1.6981	-0.6182	1.4643	0.7576	-0.0163	1.2059
-0.3970	2.1732	-0.1620	0.8902	-0.1121	-0.6624	-0.1085	0.4238	1.0918	-0.8484	0.7358	0.3078
0.0898	0.5724	0.4357	-0.4561	0.5628	0.4754	-0.3008	0.8949	-0.2168	1.8595	-0.6409	-0.4864
0.2640	0.8150	0.8613	-1.9037	-0.0307	1.2443	-1.3683	-0.2375	1.4199	-0.0360	1.4443	-0.3310
0.2922	-1.0789	0.0641	-0.3921	-1.3228	0.0296	0.7377	-0.1279	0.6269	2.5915	-1.1590	0.7767
0.7818	-0.5799	-1.7273	-1.1070	-1.0830	0.6917	-0.4043	1.0195	2.2215	-0.6913	0.6863	-0.3327
0.4040	-1.8757	0.7160	1.7575	0.1575	-0.6856	0.8568	1.7484	-1.2924	-1.5765	0.7304	2.0963
-0.3254	0.9175	0.0366	-0.7192	-1.4664	-0.0431	3.3437	0.9875	1.1703	-0.6101	0.5145	0.3888
-0.3739	-0.5469	-0.1849	-0.2199	0.6736	-1.5477	0.6265	-0.4201	-1.1789	0.3767	1.6958	-0.6525
-0.2943	-0.6051	-0.8147	0.5750	0.3625	-1.0718	1.2796	-0.3337	-0.5679	1.2728	-0.6763	-0.0568

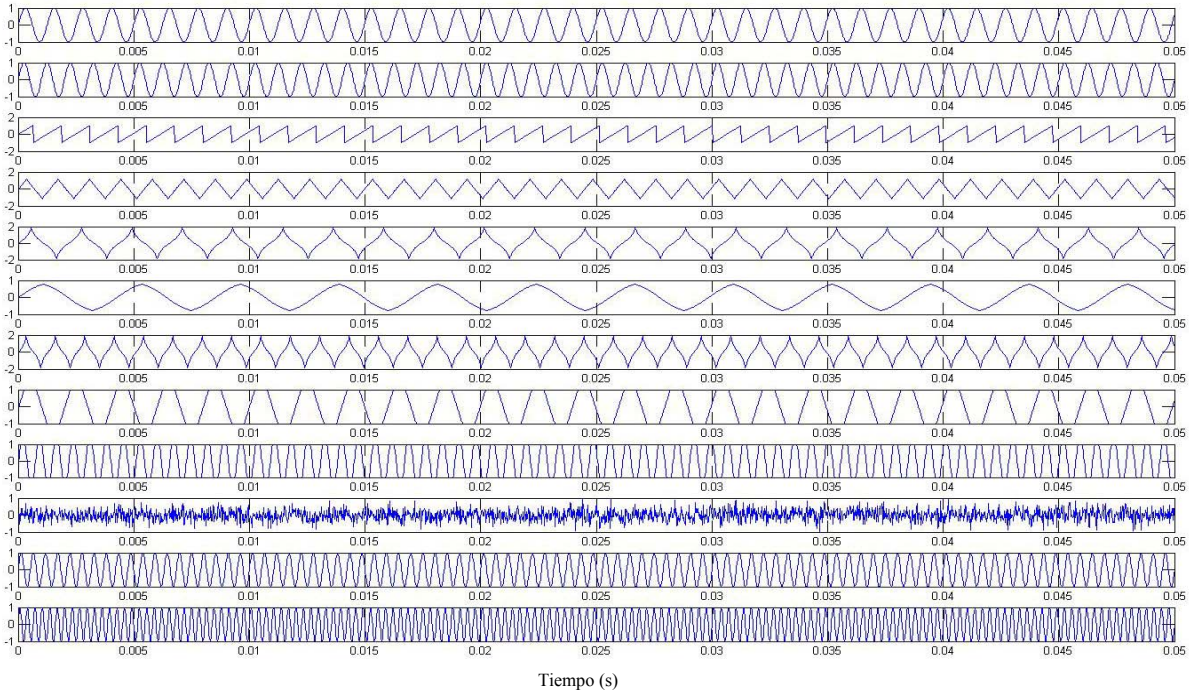


Fig. 3.4.2 Doce señales de fuentes independientes.

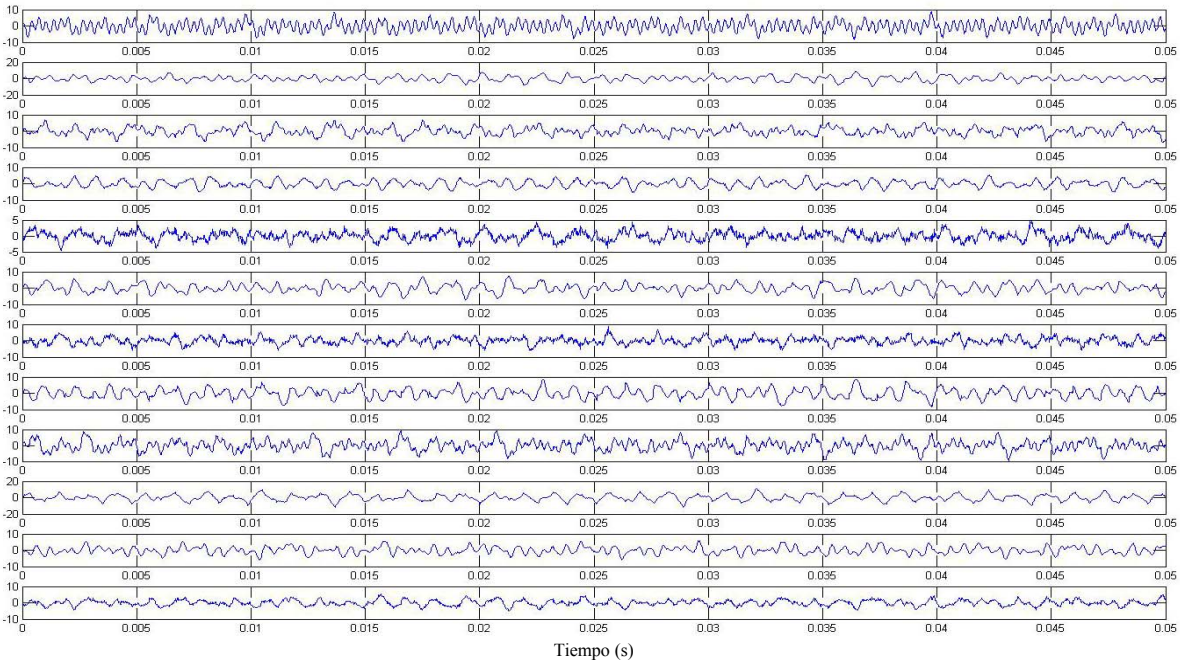


Fig. 3.4.3 Señales mezcladas de las fuentes originales.

La figura 3.4.3, muestra los resultados de la combinación lineal de las señales originales utilizando la matriz antes mencionada.

Los datos de la mezcla fueron guardados en un archivo ".txt" y posteriormente se ingresaron al navegador. La Figura 3.3.4, muestra los resultados obtenidos por el navegador de estas señales.

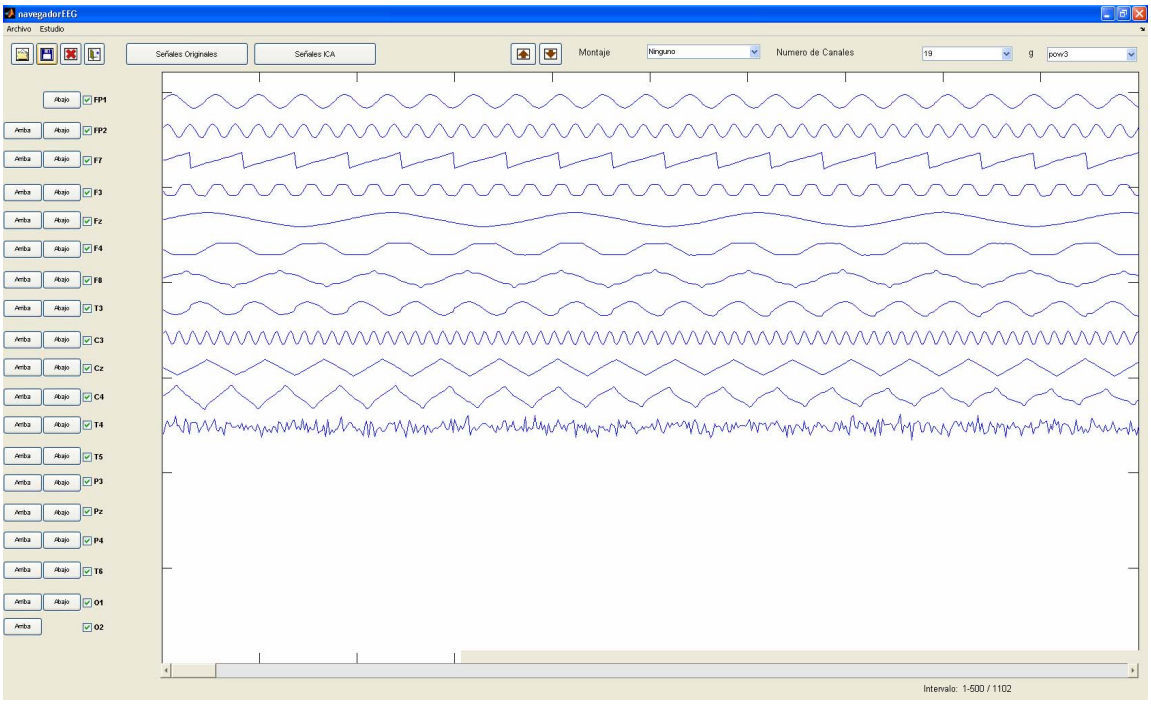


Fig. 3.4.4 Resultados finales del algoritmo ICA

En la ventana del navegador, se puede observar la estimación de las fuentes originales observando el orden intercambiado de las señales recuperadas, por efecto de la matriz de permutación.

3.5 Conclusiones

En este capítulo, se mostró el desempeño de los algoritmos ICA, implementados en hardware y software. La primera implementación realizada en este trabajo, fue en un FPGA, mostrando el desempeño de la red utilizando los algoritmos H-J e INFOMAX. La aplicación se desarrolló para solucionar el problema del "*cocktail party"*, el cual, consistió de 2 entradas y dos salidas analógicas (con posibilidad a expandirse), por donde se ingresaban y extraían las señales de audio. Como se mostró en el capítulo 2, el mayor error se obtiene en la red H-J.

La implementación en software, fue desarrollada para el diseño de un explorador EEG, utilizando el algoritmo de FastICA por negentropía de punto fijo. Verificando el desempeño mediante doce señales generadas aleatoriamente. Esta aplicación, no se restringe únicamente al uso de señales EGG, si no también, se puede verificar su funcionamiento aplicando señales de audio previamente convertidas a en un listado de datos.

Los resultados de los algoritmos implementados, únicamente mostraron las señales de salida de la red, utilizando señales reales; Quedando como trabajo futuro el análisis por lo que no fue mostrado un análisis sobre el valor de la matriz de pesos sinápticos. No obstante, se observa su funcionamiento en los resultados de la separación de las señales.

Una de las ventajas de efectuar las implementaciones en la tarjeta del DSP, es que permite verificar de una manera muy rápida los algoritmos operando en tiempo real y con muy buena aproximación ya que realiza operaciones de punto flotante.

CAPÍTULO 4

Conclusiones y trabajos futuros

4.1 Conclusiones

El objetivo general de esta tesis fue alcanzado satisfactoriamente, ya que se realizó la investigación de las técnicas de procesamiento mediante modelos neuronales artificiales, con las cuales, fue posible reducir las componentes de datos, con el propósito de identificar a aquellos que pertenecían a una misma clase.

En esta tesis, se abordaron las técnicas relacionadas con la exploración multidimensional, basadas en el Análisis de Componentes Independientes. En una primera etapa, se analizó y aplicó el método de ICA por descorrelación no lineal, el cual se baso en un modelo neuronal de baja complejidad para el cálculo de los pesos, facilitando su implementación en hardware. Posteriormente, se analizó y aplicó el método de INFOMAX cuyo algoritmo es más robusto, ya que realiza el cálculo de todos los coeficientes de la matriz de pesos **W** obteniendo mejores resultados (que el método anterior) en la separación.

Finalmente, se analizó y aplicó el método basado en la maximización de la no-gaussianidad. Este método esta constituido de dos etapas, en la primera se realizó el blanqueado de las señales y en la segunda, su independencia estadística. Este a su vez, se dividió en dos, los cuales fueron: el método de la medición de la curtosis basada en los cumulantes y el método de medición por negentropía. Ambos métodos fueron analizados con respecto al gradiente y al punto fijo.

Para la verificación de todos los métodos antes mencionados, se utilizó la herramienta MATLAB. Realizando la implementación del código correspondiente al algoritmo y probando con diferentes matrices de mezcla. De esta manera, se comprobó que el mínimo error se obtiene usando los algoritmos de maximización de la no-gaussianidad.

Otro de los puntos del objetivo es el desarrollo e implementación de estas técnicas. Las implementaciones, se realizaron en hardware y software. En el caso del hardware, se realizo en un DSP (mostrando sus resultados experimentales) y en el FPGA (solo se mostraron los resultados de simulación, aun se encuentran en proceso de la implementación), mostrando sus características y comportamiento a través de resultados de simulación y experimentales. Cabe mencionar, que aunque los métodos implementados fueron para dos señales de entrada, estos puede incrementarse.

La última parte muestra una aplicación desarrollada en software y consistió de un navegador EEG, el cual utilizó el algoritmo de negentropía de punto fijo como método de separación de señales electroencefalográficas. Este algoritmo presentó mejores resultados en la separación de las señales, solo que para la manipulación de los datos, es difícil efectuarlos en tiempo real. Por lo tanto, la elección del algoritmo adecuado dependerá del tipo de aplicación.

Algunas aportaciones que considero de este trabajo es la creación de un vínculo entre el desarrollo matemático particularmente en los algoritmos de ICA y el desarrollo de hardware (en FPGA y DSP), y la posible aplicación de estas técnicas a nuevos sistemas para mejorar su procesamiento.

4.2 Trabajos futuros

Como trabajos futuros a corto plazo de esta tesis, se mencionan:

- La implementación en hardware del algoritmo ICA utilizando el gradiente de la negentropía.
- El desarrollo de nuevos algoritmos orientados a las implementaciones en hardware.
- La síntesis y simplificación de las arquitecturas digitales.

Como consecuencia de los puntos anteriores, se menciona el desarrollo de un sistema en hardware con un mayor número de entradas de los actuales desarrollados (12 entradas) y una separación con el mínimo de error, ya que esto facilitará el procesamiento de etapas subsecuentes en sistemas más complejos.

Como trabajos futuros a largo plazo de esta tesis, se mencionan:

- El desarrollo de quipo médico con aplicaciones en señales EEG, en la predicción de algunas enfermedades referentes al cerebro.
- El desarrollo de equipo ambiental para la medición de la cantidad de ozono presente en el ambiente.
- La implementación de estos algoritmos en sensores inteligentes que permitan obtener, procesar y enviar la información adquirida.

Entre otras posibles aplicaciones.

APENDICE "A"

CONCEPTOS DE ESTADÍSTICA

A1.- Introducción

En este apéndice, se darán los conceptos básicos de teoría de la probabilidad, estadística y procesos aleatorios. Estos conceptos son utilizados en los métodos de ICA, aquí mencionados. Estos temas pueden consultarse en [PAP91, COO71, HAM91, WIL68]

La teoría de la probabilidad trata con el promedio del fenómeno de las masas ocurriendo secuencialmente o simultáneamente. Se ha observado que ciertos promedios aproximan a un valor constante tanto como el número de observaciones incrementa. El objetivo de este promedio es describir y predecir los eventos sucesivos en términos de sus probabilidades.

En el análisis de una señal aleatoria, es de interés el estudio de su Función de Densidad de probabilidad y su función de distribución de probabilidad, ya que proporcionan información sobre la ocurrencia de algún valor determinado.

Se dice que una variable aleatoria que puede asumir cualquier valor dentro de algún intervalo especificado, es una variable continua. Sin embargo, una variable aleatoria discreta que solo pueda asumir ciertos valores, puede ser tratada con el mismo método.

A2.- Función de Distribución de Probabilidad o Función de Distribución.

Sea X una variable aleatoria y x cualquier valor permitido (o evento) de esta variable. La función de distribución de probabilidad es definida como la probabilidad acumulada de que el evento x de la variable aleatoria observada X ocurra, por lo tanto la probabilidad es menor o igual que el valor permitido x, esto es:

$$P_X(x) = \Pr(X \leq x) \qquad \text{(A2.1)}$$

Puesto que la función de distribución de probabilidad es una probabilidad, debe satisfacer los siguientes axiomas básicos:

1.- $0 \leq P_X(x) \leq 1 \qquad -\infty < x < \infty$

2.- $P_X(-\infty) = 0 \qquad P_X(\infty) = 1$

3.- $P_X(x)$ es no decreciente tanto como x incrementa

4.- $P_r(x_1 < X < x_2) = P_X(x_2) - P_X(x_1)$

Algunas funciones de distribución son mostradas en la figura A1. Indicando una variable aleatoria continua con valores posibles en el intervalo de - ∞ a + ∞.

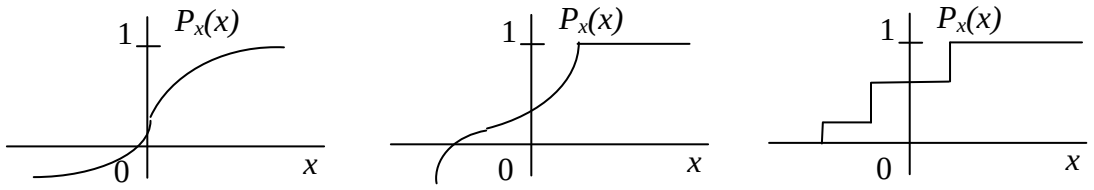


Fig. A1 Ejemplos de posibles funciones de distribución de probabilidad.

A3.- Función de Densidad de probabilidad

También conocida como la distribución de probabilidad. La cual como su nombre lo indica, define como se distribuye la probabilidad a través de la densidad de repetición de una variable aleatoria. Esta forma es más conveniente para muchos cálculos de interés.
Esta función de densidad se puede obtener a través de tomar la derivada de la función de distribución de probabilidad ($P_x(x)$).

$$px(x) = \lim_{\varepsilon \longrightarrow 0} \frac{P_x(x+\varepsilon) - P_x(x)}{\varepsilon} = \frac{dP_x(x)}{dx} \qquad \text{(A3.1)}$$

Rescribiendo la ecuación (A3.1), la función de densidad de probabilidad puede ser interpretada como:

$$px(x)dx = P_r(x < X \leq x + dx) \qquad \text{(A3.2)}$$

La ecuación (A3.2) establece que la probabilidad del elemento, $px(x)dx$, es la probabilidad de que el evento de la variable aleatoria X se encuentre en el intervalo de posibles valores de x y $x + dx$.

Puesto que $px(x)$ es una función de densidad de probabilidad y no una probabilidad, no es necesario que sus valores sean menor que 1; pero no negativos. Sus propiedades se pueden resumir en los siguientes axiomas:

1.- $px(x) \geq 0 \qquad -\infty < x < +\infty$

2.- $\int_{-\infty}^{\infty} px(x)dx = 1$

3.- $P_x(x) = \int_{-\infty}^{x} px(u)du$

4.- $\int_{x1}^{x2} px(x)dx = P_r(x_1 < X \leq x_2)$

Algunas funciones de densidad de probabilidad son mostradas en la figura A2

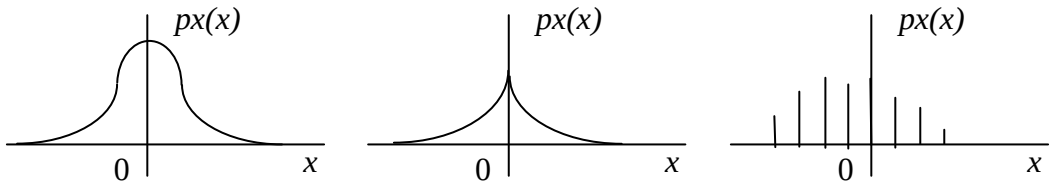


Fig. A2 Ejemplos de posibles funciones de densidad de probabilidad.

Particularmente la función de densidad para una variable aleatoria discreta consiste de un conjunto de funciones delta, cada una teniendo un área igual a su magnitud.

Existen muchas posibles funciones de densidad pero algunas cuantas de estas, tienen un significado en el análisis de procesamiento de señales.

A4. Valor medio y Valor cuadrático medio

Uno de los conceptos más importantes asociados en estadística es el valor medio o promedio de una variable o variables aleatorias. El valor medio o promedio de una variable que esta en función del tiempo, se obtiene integrando sobre todo el intervalo del tiempo que se toma la variable; otra forma de tomar el promedio de una variable, es considerar el número de muestras y promediarlas sobre el número de éstas.

Diferentes notaciones son utilizadas para encontrar el valor medio. En ingeniería el símbolo más empleado es $E\{X\}$ que usualmente se lee el valor esperado de "X" o la esperanza matemática de "X".

$$\overline{X} = E\{X\} = \int_{-\infty}^{\infty} xp(x)dx \tag{A4.1}$$

El valor esperado de cualquier función puede obtenerse de forma similar:

$$E\{F(X)\} = \int_{-\infty}^{\infty} f(x)p(x)dx \tag{A4.2}$$

Una función de particular importancia es $f(x) = x^n$, puesto que esta función nos permite generar **los momentos** de una variable aleatoria.

$$\overline{X^n} = E\{X^n\} = \int_{-\infty}^{\infty} x^n p(x)dx \tag{A4.3}$$

Los momentos más importantes son para $n = 1$ y $n = 2$ el cual genera el valor medio (A4.1) y el valor cuadrático medio (A4.4), respectivamente.

$$\overline{X^2} = E\left\{X^2\right\} = \int_{-\infty}^{\infty} x^2 p(x) dx \qquad \text{(A4.4)}$$

A5. Varianza y Desviación estándar.

Otro de los conceptos más importantes en estadística es el **momento central**, el cual se define como la diferencia entre una variable aleatoria y su valor medio. Por lo tanto el "*n*"-esímo momento central es:

$$\overline{(X - \overline{X})^n} = E\left\{\left\{X - \overline{X}\right\}^n\right\} = \int_{-\infty}^{\infty} (x - \overline{X})^n \, p(x) dx \qquad \text{(A5.1)}$$

El momento central para n =1 es cero, mientras que para n =2 se le denomina la varianza, usualmente simbolizada por σ^2.

$$\sigma^2 = \overline{(X - \overline{X})^2} = \overline{X^2} - \overline{X}^2 \qquad \text{(A5.2)}$$

Por esta razón se define la varianza, como la diferencia entre el valor cuadrático medio y el cuadrado del valor medio. La raíz cuadrada de la varianza se conoce como la desviación estándar σ.

Algunas de las desventajas de los momentos, es que existen distribuciones para el cual los momentos no son finitos y que el conocimiento de estos no necesariamente especifica la función de densidad de probabilidad. Sin embargo, para muchas de las distribuciones los momentos no cumplen con estas condiciones.

A6. Cumulantes.

Los cumulantes son otras constantes que describen y especifican las propiedades y características de una distribución y son más útiles, desde el punto de vista teórico, que los momentos. Los cumulantes de orden "*k* " pueden calcularse a partir de los momentos de igual orden o inferior a k. Los cumulantes del orden 1 al 4, respecto al origen y en función de los momentos son:

$$K_1 = \overline{X} \text{ (Media)} \tag{A6.1}$$

$$K_2 = \overline{X^2} - \overline{X}^2 \text{ (Varianza)} \tag{A6.2}$$

$$K_3 = \overline{X^3} - 3\overline{X^2}\,\overline{X} + 2\overline{X}^3 \tag{A6.3}$$

$$K_4 = \overline{X^4} - 4\overline{X^3}\,\overline{X} - 3\overline{X^2}^2 + 12\overline{X^2}\,\overline{X}^2 - 6\overline{X}^4 \tag{A6.4}$$

Si las variables aleatorias están centradas en su media, es decir $\overline{X} = 0$, las expresiones de los cumulantes se simplifican en:

$$K_1 = 0 \tag{A6.5}$$

$$K_2 = \overline{X^2} \tag{A6.6}$$

$$K_3 = \overline{X^3} \text{ (skewness)} \tag{A6.7}$$

$$K_4 = \overline{X^4} - 3\overline{X^2}^2 \text{ (Kurtosis)} \tag{A6.8}$$

En donde el cumulante normalizado de 3er orden determina la asimetría (skewness) de una distribución y el de 4to orden determina la agudeza o curtosis (Kurtosis). Para una variable con distribución normal (gaussiana) los cumulantes de orden superior a 2 son cero; es decir $K_K = 0 \;\; (\forall K > 2)$. Se vera más a delante la utilidad de cada uno de los cumulantes de orden superior para una distribución normal.

A7. Variable aleatoria Gaussiana o Normal.

Una de las funciones de densidad más estudiadas es la Gaussiana o normal. Alguna de las razones para su importancia, son:

1.- Este proporciona un buen modelo matemático para muchos fenómenos aleatorios observados físicamente.
2.- La combinación lineal de variables aleatorias Gaussianas da como resultado una nueva variable aleatoria Gaussiana.
3.- Los procesos aleatorios del cual una variable aleatoria Gaussiana es derivado, puede ser completamente especificado, en el sentido estadístico, esto es, el conocimiento de su primero y segundo momento.
4.- En el análisis del sistema, el proceso Gaussiano es frecuentemente el único para el cual un análisis estadístico completo puede ser realizado en una situación lineal y no lineal.

La representación matemática de la función de densidad Gaussiana es:

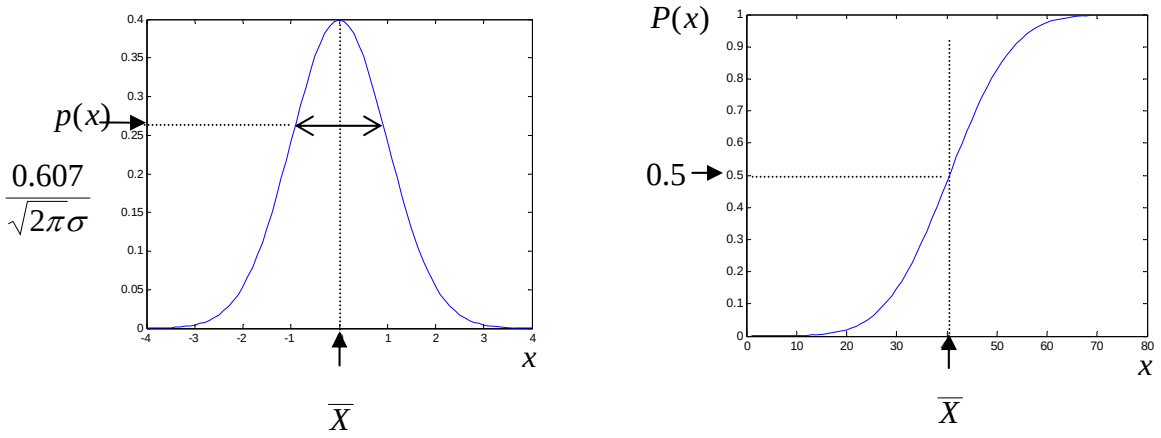
$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[\frac{-(x-\overline{X})^2}{2\sigma^2}\right] \qquad -\infty < x < \infty \qquad \text{(A7.1)}$$

Donde $\overline{X}$ y σ^2 son la media y la varianza, respectivamente. La figura A3 muestra la función de densidad y su correspondiente función de distribución de una variable aleatoria con media cero ($\overline{X}$ =0) y varianza unitaria ($\sigma = \sigma^2 = 1$)). Algunas características de este tipo de distribución, son las siguientes:

1.- Tienen un único máximo y este ocurre en el valor de la media.
2.- La función de densidad de probabilidad es simétrica alrededor de su media.
3.- El ancho de la distribución, es directamente proporcional a la desviación estándar σ .
4.- El valor máximo de la función de densidad es inversamente proporcional a la desviación estándar σ .

Usualmente la función de densidad de probabilidad para este tipo de distribución se considera normalizada, es decir, con media cero y varianza unitaria ($\overline{X}=0$, $\sigma^2=1$) y define como:

$$\Phi(x)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}\exp\left[-\frac{x^2}{2}\right]dx \qquad \text{(A7.2)}$$



a) b)

Fig. A3 Variable aleatoria Gaussiana: a) función de densidad y b) función de probabilidad.

Donde $\Phi(x)$ representa esta función de distribución normalizada. Efectuando un cambio de variable, de la ecuación (A7.1) y tomando la 11, se obtiene que:

$$p(x)=\Phi\left[\frac{x-\overline{X}}{\sigma}\right] \qquad \text{(A7.3)}$$

El "*n*"-esimo momento central de una variable aleatoria gaussiana con media cero y varianza unitaria, puede expresarse como:

$$\begin{aligned}\overline{(X-\overline{X})^n}&=0 && n \text{ impar}\\ &=(\sigma^2=1),(3\sigma^4=3),(5\sigma^6=5),....(n-1)\sigma^n && n \text{ par}\end{aligned} \qquad \text{(A7.4)}$$

Bajo ciertas condiciones, que suelen cumplir la mayoría de las funciones de distribución, si dos variables aleatorias, tienen sus respectivos momentos iguales (con que tengan los 3 o 4 primeros momentos iguales), se puede considerar que sus funciones son idénticas.

Considerando el centrando de la media en cero y normalizando la varianza de la señal, los cumulantes, efectuando ($\overline{X}=0$, $\sigma^2=1$), es decir, efectuando el siguiente cambio de variable.

$$s=\frac{x-\overline{X}}{\sigma} \tag{A7.5}$$

Donde *s* es la señal normalizada; es decir que tienen media cero y varianza unitaria. Los cumulantes normalizados $\tilde{K}_K$, son los cumulantes de las señales estandarizadas. El cumulante normalizado de orden 3 ($\tilde{K}_3$), mide la asimetría (skewness) de la función de distribución ya que es cero si la función de densidad de probabilidad de *x* posee un eje de simetría. Sí x esta centrada, el factor de asimetría (γ_1) resulta ser:

$$\gamma_1=\tilde{\chi}_3=\frac{E\{s^3\}}{E\{s^2\}^{3/2}}=\frac{\mu_3}{\mu_2^{3/2}}=\frac{\kappa_3}{\kappa_2^{3/2}} \tag{A7.6}$$

Si la distribución es simétrica: $\gamma_1=0$ (ya que toda distribución de probabilidad simétrica tiene los momentos impares nulos, y en particular $\mu_3=0$)

El cumulante estandarizado de orden 4, $\tilde{\kappa}_4$, es el *factor de agudeza* o c*urtosis* (γ_2). Si la señal está centrada en la media:

$$\gamma_2=\tilde{\kappa}_4=\frac{E\{s^4\}}{E\{s^2\}^2}-3=\frac{\mu_4}{\mu_2^2}-3=\frac{\kappa_4}{\kappa_2^2} \tag{A7.7}$$

Si $\gamma_2=0$ se dice que la distribución es mesoaguda (gaussiana). Si la agudeza es negativa ($\gamma_2<0$) se dice que la distribución es *sub-gaussiana* (o *platicúrtica*) , ya que tiende a cero en

el infinito más rápidamente que la distribución gaussiana. Por contrario, si la agudeza es positiva ($\gamma_2>0$), la función de distribución tiende a cero en el infinito más lentamente que la distribución gaussiana se dice que es *super-gaussiana* (o *leptocúrtica*).

A8. Múltiples variables aleatorias

El término múltiples, considerara que se trabaja con dos o mas variables aleatorias, pero por simplicidad se consideran solo dos variables (extendiendo el mismo estudio a un grupo mayor). Definiendo su función de distribución de probabilidad de x y y como:

$$P(X) = P\left[X \le x\right] \tag{A8.1}$$

$$P(Y) = P\left[Y \le y\right] \tag{A8.2}$$

Estas distribuciones de probabilidad, también se le conoce como probabilidades marginales. Expresando la densidad de probabilidad conjunta (de juntura) de las dos variables se define como:

$$P(X,Y) = P\left[X \le x, Y \le y\right] \tag{A8.3}$$

Algunas de las propiedades análogas de la probabilidad de juntura con respecto a la distribución de una sola variable se muestran a continuación:

1.- $0 \le P_X(x,y) \le 1 \qquad -\infty < x < \infty \quad -\infty < y < \infty$

2.- $P(-\infty, y) = P(x,-\infty) = P(-\infty,-\infty) = 0$

3.- $P(\infty,\infty) = 1$

4.- $P(x,y)$ es no decreciente tanto como x ó y incrementa

5.- $P(\infty, y) = P(y) \qquad P(x,\infty) = P(x)$

El valor esperado de la función de densidad de probabilidad de juntura de dos variables aleatorias puede ser determinado por la siguiente expresión:

$$E\left[f(X,Y)\right] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\,p(x,y)\,dxdy \tag{A8.4}$$

Considerando el caso general de $f(X,Y) = XY$. La ecuación A8.4 se puede representar como:

$$E\{XY\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyp(x,y)dxdy \qquad \text{(A8.5)}$$

Esta expresión determina la correlación de dos variables aleatorias.

Las propiedades de la función de distribución de probabilidad de juntura son análogas a aquellas de una sola variable, y se mencionan a continuación:

1.- $p(x,y) \geq 0 \quad -\infty < x < +\infty \quad -\infty < y < +\infty$

2.- $\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x,y)dxdy = 1$

3.- $P(x,y) = \int_{-\infty}^{y}\int_{-\infty}^{x} p(u,v)dudv$

4.- $p(x) = \int_{-\infty}^{\infty} p(x,y)dy \qquad p(y) = \int_{-\infty}^{\infty} p(x,y)dx$

5.- $P[x_1 < X \leq x_2, y_1 < X \leq y_2] = \int_{x_1}^{x_2}\int_{y_1}^{y_2} p(x,y)dydx$

También es posible determinar la función de densidad de probabilidad de juntura derivando la función de distribución. Puesto que son dos variables independientes la derivada obtendrá un resultado parcial.

Otro concepto es la probabilidad condicional. Esta es la probabilidad de un evento *X*, dado que otro evento *Y* ha ocurrido y se expresa como *P(X|Y)*. Entonces la probabilidad de un evento *X*, dado un evento *Y*, se define como:

$$P(X \mid Y) = \frac{P(X,Y)}{P(Y)} \qquad \text{(A8.6)}$$

Análogamente *P(Y|X)*.

$$P(Y \mid X) = \frac{P(X,Y)}{P(X)} \tag{A8.7}$$

donde $P(X,Y) = P(Y,X)$ es la probabilidad de juntura del evento X y Y. Y $P(X)$ ó $P(Y)$ es la probabilidad marginal de los eventos particulares de X y Y, respectivamente. Si el evento X esta contenido en Y $(X \subset Y)$ entonces $(X,Y)=X$ y

$$P(X \mid Y) = \frac{P(X)}{P(Y)} \geq P(X) \tag{A8.8}$$

Sí $(Y \subset X)$, entonces $(X,Y)=Y$ y

$$P(X \mid Y) = \frac{P(Y)}{P(Y)} = 1 \tag{A8.9}$$

La figura A4 muestra la distribución de densidad de probabilidad de x y y, en la cual se denota la densidad marginal, la densidad condicional $(y|x)$ y de juntura (y,x)

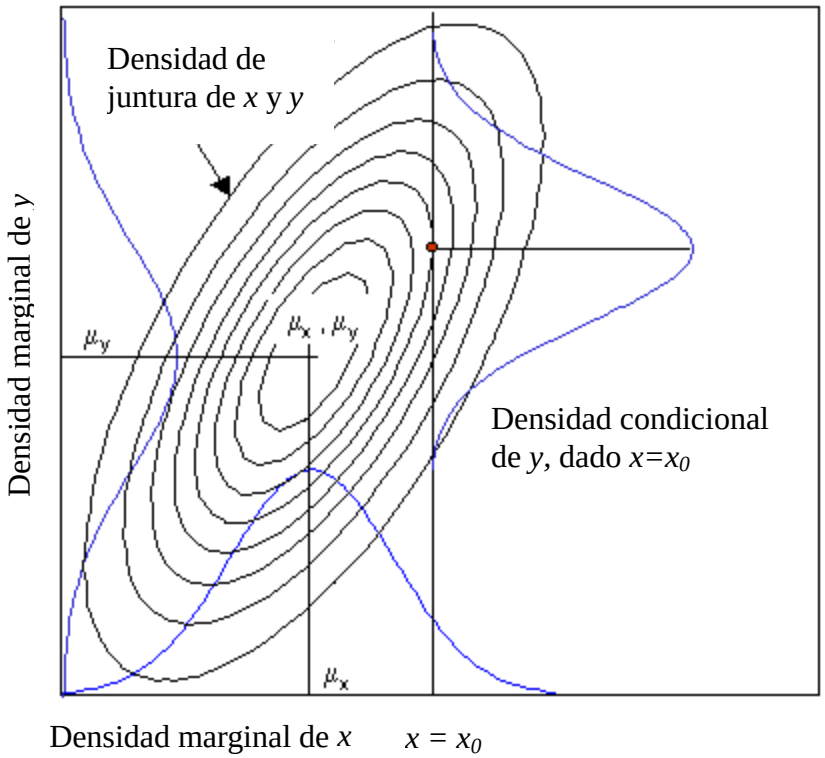


Fig. A4 Distribución de densidades de probabilidad, marginales, condicionales y de juntura

A9 Independencia estadística

Considerando el caso de dos variables aleatorias X y Y . Cuando dos variables aleatorias son estadísticamente independientes, el conocimiento de una variable aleatoria no da información acerca del valor de la otra y matemáticamente en términos de probabilidad, la independencia estadística se define como:

$$P(X,Y) = P(X)P(Y) \tag{A9.1}$$

En general si existe n eventos, la expresión (A9.1) se representa como:

$$P(X,Y,...,n) = p(X)\,p(Y),...,\,p(n) \tag{A9.2}$$

En el caso que se tome cualquier función, se satisface la misma propiedad básica.

$$E\{g(X)h(Y)\} = E\{g(X)\}E\{h(Y)\} \tag{A9.3}$$

Donde $g(X)$ y $h(Y)$ son cualquier función integrable.

$$E\{g(X)h(Y)\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x)h(y)p_{XY}(x,y)dydx$$

$$= \int_{-\infty}^{\infty} g(x)\,p_X(x)dx \int_{-\infty}^{\infty} h(y)\,p_Y(y)dy = E\{g(X)\}E\{h(Y)\} \tag{A9.4}$$

La misma condición de independencia estadística se cumple para el caso multidimensional.

$$E\{g(X)g(Y)g(Z)...\} = E\{g(X)\}E\{g(Y)\}E\{g(Z)\}... \tag{A9.5}$$

Donde *g( )* son funciones arbitrarias.

Como se mencionó anteriormente la probabilidad condicional se representa como:

$$P(Y,X) = P(X,Y) = P(X \mid Y)P(Y) = P(Y \mid X)P(X) \qquad \text{(A9.6)}$$

Despejando de los dos últimos términos de la ecuación (A9.6), el término *P(X |Y)*:

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)} \quad P(Y) \neq 0 \qquad \text{(A9.7)}$$

El termino *P(Y)* se puede representar como:

$$P(Y) = P(Y \mid X_1)P(X_1) + P(Y \mid X_2)P(X_2) + ... + P(Y \mid X_n)P(X_n) \qquad \text{(A9.8)}$$

Substituyendo la ecuación (A9.8) en (A9.7), se obtiene:

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y) = P(Y \mid X_1)P(X_1) + P(Y \mid X_2)P(X_2) + ... + P(Y \mid X_n)P(X_n)} \qquad \text{(A9.9)}$$

La ecuación (A9.7) o la (A9.9) son referidas como el Teorema de Bayes. Por lo tanto, la expresión de *P(Y |X)*, se deduce como:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} \quad P(X) \neq 0 \qquad \text{(A9.10)}$$

Sí las dos variables (*X* y *Y*) son independientes, se obtiene su probabilidad marginal, esto es:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} = \frac{P(X,Y)}{P(X)} = \frac{P(X)(Y)}{P(X)} = P(Y) \qquad \text{(A9.11)}$$

Equivalentemente:

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)} = P(X) \qquad \text{(A9.12)}$$

A10 Teorema de límite central

El teorema del límite central o teorema central del límite concierne a la suma de un gran número de variables aleatorias independientes con mismas funciones de densidad de probabilidad. Del cual el resultado se aproximará a una función de densidad Gaussiana, tanto como el número de variables independientes se incremente. Esta suma queda definida como:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_n \tag{A10.1}$$

En donde X_n, son las variables aleatorias independientes. El teorema del límite central tiene una importante consecuencia en el análisis de componentes independientes.

A11 Correlación

La correlación indica la fuerza y la dirección de una relación lineal entre dos variables aleatorias. Se considera que dos variables aleatorias cuantitativas están correlacionadas cuando los valores de una de ellas varían sistemáticamente con respecto a los valores homónimos de la otra.

Si dos variables aleatorias *X* y *Y* tienen los valores posibles *x* y *y*, entonces el valor esperado de su producto es conocido como la correlación, definido como:

$$E\{X,Y\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyp(x,y)dxdy = \overline{XY} \tag{A11.1}$$

La correlación de dos variables aleatorias independientes continuas es igual a sus probabilidades marginales.

$$\begin{aligned} E\{X,Y\} &= \int_{-\infty}^{\infty} xp(x)dx\int_{-\infty}^{\infty} yp(y)dy \\ &= E\{X\}E\{Y\} = \overline{X}\,\overline{Y} \end{aligned} \tag{A11.2}$$

Si dos variables aleatorias tienen su media diferente de cero, es conveniente restarle este valor.

$$E\{(X-\overline{X})(Y-\overline{Y})\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x-\overline{X})(y-\overline{Y})\,p(x,y)dxdy \qquad \text{(A11.3)}$$

Esta ecuación es conocida como la covarianza, análogamente a la varianza de una sola variable.

El coeficiente de correlación o covarianza normalizada [COO71] es un índice estadístico (entre -1 y +1) que indica la relación lineal entre dos variables aleatorias. El cálculo del coeficiente de correlación lineal se realiza dividiendo la covarianza por el producto de las desviaciones estándar de ambas variables:

$$r = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \qquad \text{(A11.4)}$$

El coeficiente de correlación es una forma simple de la covarianza normalizada. Note que si dos variables aleatorias de media cero no están correlacionadas, entonces su covarianza es igual a 0. Sí r =1 se tiene una total dependencia de las dos variables. Sí r = -1 existe una dependencia negativa, es decir, que cuando una de ellas aumenta, la otra disminuye en idéntica proporción.

APENDICE "B"

CONCEPTOS DE ÁLGEBRA LINEAL

B1.- Introducción

El álgebra lineal, es la rama de la matemática que concierne al estudio de vectores, espacios vectoriales, transformaciones lineales y sistemas de ecuaciones lineales. Los espacios vectoriales son un tema central en la matemática moderna; por lo que el álgebra lineal es usada ampliamente en álgebra abstracta y análisis funcional. El álgebra lineal considera espacios de dimensión arbitraria o incluso de dimensión infinita. Un espacio vectorial de dimensión *n* se dice que es *n*-dimensional. La mayoría de los resultados encontrados en 2 y 3 dimensiones pueden extenderse al caso *n*-dimensional.

En este apéndice, se darán algunos conceptos básicos del álgebra lineal. Estos conceptos son utilizados en los métodos de ICA, aquí mencionados. Algunas otras referencias pueden consultarse en [STE90, GOL93, STR82].

B2. Sistema de ecuaciones lineales

Un sistema lineal, es un sistema que obedece las propiedades de escalado y de superposición y se describe con la siguiente ecuación.

$$a_1 x_1 + a_2 x_2 + ... + a_n x_n = b \qquad \text{(B2.1)}$$

Donde $a_1, a_2, ...a_n$ y b_n son números reales y $x_1, x_2, ..., x_n$ son variables. Un sistema lineal de *m* ecuaciones y *n* incógnitas es un sistema de la forma:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned} \tag{B2.2}$$

A este sistema se le conoce como sistema de ecuaciones lineales. Una forma de representar a este conjunto de ecuaciones es mediante la notación de Matrices y Vectores.

B3. Matrices y vectores

Denotando la ecuación (B2.2) en forma matricial y vectorial, se tiene de la forma:

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{B3.1}$$

Simplificando la notación anterior, se tiene que:

$$\mathbf{Ax} = \mathbf{b} \tag{B3.2}$$

Donde **A** es una matriz de dimensión *mxn*, donde *n* son las columnas y *m* las filas, **x** es una vector fila de dimensión *n* y **b** es el vector columna de dimensión *m*. Cuando una matriz es de dimensión *nxn*, se dice que es una matriz cuadrada de dimensión *n*. Cabe hacer notar que las matrices son denotadas con mayúscula y en negrita, a diferencia de los vectores que son denotadas en minúscula.

Un vector fila de dimensión *m* es una matriz de dimensión *1xm*, cuyos elementos se representan como una sucesión de datos en forma horizontal. Como por ejemplo, el vector fila (*1x8*).

$$\mathbf{b} = (a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8)$$

Un vector columna de dimensión *n* es una matriz de dimensión nx1, cuyos elementos se representan como una sucesión de datos en forma vertical. Como por ejemplo, el vector columna (*4x1*).

$$\mathbf{b} = \begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{matrix}$$

B4. Matriz Identidad

La matriz de identidad *I*, algunas veces denominada matriz unidad, es una matriz cuadrada en la cual los elementos situados sobre la diagonal principal son iguales a 1 y el resto de los elementos son iguales a 0.

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{B4.1}$$

La matriz identidad es una matriz que cumple la propiedad de ser el elemento neutro del producto de matrices. Esto quiere decir que el producto de cualquier matriz por la matriz identidad no tiene ningún efecto

B5. Producto punto de dos vectores

En matemáticas el producto escalar, también conocido como producto interno o producto punto, es una función definida sobre un espacio vectorial cuyo resultado es una magnitud escalar. El nombre espacio escalar se utiliza para denominar un espacio vectorial real sobre el que se ha definido una operación de producto interior que tiene como resultado un número real. El producto punto de **a** y **b** se define como:

$$< \mathbf{a}, \mathbf{b} >= \sum_{i=1}^{n} a_i b_i \tag{B5.1}$$

De manera similar se define el producto punto de dos vectores columnas o de un vector fila y un vector columna. En todos los casos, los vectores deben tener el mismo número de componentes. El producto escalar en el caso particular de dos vectores en el plano, o en un espacio euclídeo N-dimensional, se define como el producto de sus módulos multiplicado por el coseno del ángulo θ que forman.

B6. Norma euclidiana de un vector

La norma euclidiana es un espacio vectorial normado de dimensión finita en el que la norma es heredada de un producto escalar. El espacio euclídeo es el espacio matemático *n*-dimensional. Sean $\mathbf{a} = [a_1, a_2, a_3,..., a_n]$. La norma euclidiana de **a** se define como:

$$\|\mathbf{a}\| = \sqrt{<\mathbf{a}, \mathbf{a}>} = \sqrt{\sum_{i=1}^{n} a_i^2} \qquad \text{(B6.1)}$$

B7. Producto entre matrices

Dada una matriz **A** y una matriz **B**, el producto de estas dos matrices se puede realizar sólo si el número de columnas de la matriz **A** es el mismo que el número de filas de la matriz **B**, es decir sí **A** es una matriz *m*-por-*n* y **B** es una matriz *n*-por-*p*, entonces su producto matricial **AB** es la matriz *m*-por-*p* (*m* filas, *p* columnas) dada por:

$$c_{ij} = \sum_{K=1}^{m} \sum_{S=1}^{m} a_{ik} a_{sj} \qquad \text{(B7.1)}$$

Esta expresión es el producto punto del i-esima fila de **A** y la j-esima columna de **B**.

B8. Matriz invertible

Una matriz **A** de dimensiones $n \times n$ se dice que es invertible, inversible, inversa o no singular si y solo existe una matriz **B** de dimensiones $n \times n$(cuadrada) tal que **AB** = **BA** = **I**. En tal caso, **B** suele denominarse con $\mathbf{A}^{-1}$ y se denomina como la inversa de A.

Algunas propiedades de la matriz inversa, son:

$$(\mathbf{BA})^{-1} = \mathbf{A}^{-1}\mathbf{B}^{-1} \tag{B8.1}$$

Si la matriz es invertible, también lo es su transpuesta, y el inverso de su transpuesta es la transpuesta de su inversa, es decir:

$$(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T \tag{B8.2}$$

Y, evidentemente:

$$(\mathbf{A}^{-1})^{-1} = \mathbf{A} \tag{B8.3}$$

B9. Determinante

El determinante de una matriz **A**, es un escalar o polinomio, que resulta de obtener todos los productos posibles de una matriz de acuerdo a una serie de restricciones, siendo denotado como |**A**| o det(**A**). El valor numérico es conocido también como módulo de la matriz. Por ejemplo:

$$\text{Para } \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ el } \det(\mathbf{A}) = a_{11}a_{22} - a_{12}a_{21}$$

$$\text{Para } \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\text{el } \det(\mathbf{A}) = a_{11}[a_{22}a_{33} - a_{23}a_{32}] - a_{12}[a_{21}a_{33} - a_{23}a_{31}] + a_{13}[a_{21}a_{32} - a_{22}a_{31}]$$

Teorema B9.1. Sea **A** una matriz cuadrada. **A** es no singular si y solo si el det(**A**) es distinto de cero

Teorema B9.2. Sea **A** una matriz cuadrada de dimensión *n* y sea **b** un vector columna de dimensión *n*. El sistema **Ax** = **b** tiene un único vector solución x si y solo si A es no singular.

Corolario B9.3. El sistema **Ax** = 0 tiene una solución distinta de cero si y solo si el determinante de **A** es igual a cero.

B10. Independencia lineal

Un conjunto de vectores es linealmente independiente si ninguno de ellos puede ser escrito con una combinación lineal de los restantes. Sea $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}$,....,$\mathbf{x}_n$ vectores en el espacio vectorial V son linealmente independientes sí

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + a_3\mathbf{x}_3 + ... + a_n\mathbf{x}_n = 0 \qquad \text{(B10.1)}$$

Se dice que son linealmente independientes si existen números a_1, a_2, a_3, y a_n , no todos igual a cero.

Entre las propiedades de los vectores linealmente dependientes e independientes encontramos:

1. Un conjunto de vectores es linealmente dependiente si y solamente si alguno de los vectores es combinación lineal de los demás.
2. Si un conjunto de vectores es linealmente independiente cualquier subconjunto suyo también lo es.

Sean $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}$,....,$\mathbf{x}_n$ vectores en $\Re^n$ se dice que tales vectores generan a $\Re^n$ si y solo si para cada vector en $\Re^n$ existen constantes a_1, a_2, a_3, y a_n tales que:

$$a = a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + a_3\mathbf{x}_3 + ... + a_n\mathbf{x}_n \qquad \text{(B10.2)}$$

Una base de $\Re^n$ es un conjunto de vectores en $\Re^n$ que son linealmente independientes y que generan a $\Re^n$.

Los vectores $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}$,....,$\mathbf{x}_n$ son una base de $\Re^n$ si y solo si el determinante de la matriz cuyas columnas son $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}$,....,$\mathbf{x}_n$ es distinto de cero.

B11. Valores y Vectores propios (eigenvalores, eigenvectores)

Sea **A** una matriz cuadrada de dimensión *n*. Un vector **x** se dice que es un eigenvector o valor característico de la matriz **A** si y solo si existe una constante λ tal que $\mathbf{Ax} = \lambda \mathbf{x}$. La constante λ se denomina el eigenvalor o valor característico de **A** asociado a un vector **x**.

Sea A una matriz cuadrada de dimensión n. Los eigenvalores de λ de A se obtienen de resolver la ecuación polinomial.

$$|\mathbf{A} - \lambda \mathbf{I}| = 0 \qquad \text{(B11.1)}$$

La cual se denomina la ecuación característica asociada a la matriz cuadrada A. Los vectores propios se obtienen de resolver el sistema $\mathbf{Ax} = \lambda \mathbf{x}$, donde λ es una de las raíces de la ecuación característica.

B12. Matriz Diagonalizable

Una matriz cuadrada **A** es diagonalizable si y solo si existe una matriz no singular **P** tal que la matriz $\mathbf{D} = \mathbf{P}^{-1}\mathbf{AP}$ es diagonal. Esta definición equivale a decir que A es diagonalizable si y solo si **A** es similar a una matriz diagonal.

Teorema B12.1: Una matriz cuadrada **A** de dimensión *nxn* es diagonalizable si y solo si **A** tiene *n* vectores propios linealmente independientes.

Corolario B12.2: Sea **A** diagonalizable .Se tiene que $\mathbf{D} = \mathbf{P}^{-1}\mathbf{AP}$ es diagonal si y solo si la matriz **P** tiene como columnas los vectores propios de **A**, mientras que **D** tiene como elementos diagonales los valores propios correspondientes.

Sí **A** es semejante a una matriz diagonal $\mathbf{D} = \mathbf{P}^{-1}\mathbf{AP}$, donde $\mathbf{D} = \text{diag}(\lambda_1,.....,\lambda_n)$, entonces A tiene la factorización $\mathbf{A} = \mathbf{PDP}^{-1}$ y para cualquier potencia natural *m* de **A** se tiene que

$$\mathbf{A}^m = \left(\mathbf{PDP}^{-1}\right)^m = \mathbf{PD}^m\mathbf{P}^{-1} = \mathbf{P}diag(\lambda_1^m,....,\lambda_n^m)\mathbf{P}^{-1} \qquad \text{(B12.1)}$$

Si las entradas de la matriz diagonal **D** son no negativas, se puede definir la matriz raíz cuadrada de **A**

$$\mathbf{A}^{\frac{1}{2}} = \mathbf{PD}^{\frac{1}{2}}\mathbf{P}^{-1} \qquad \text{(B12.2)}$$

Donde $\mathbf{D}^{\frac{1}{2}} = diag(\lambda_1^{\frac{1}{2}},...,\lambda_n^{\frac{1}{2}})$, así como la matriz $\mathbf{A}^{-\frac{1}{2}}$ mediante la igualdad

$$\mathbf{A}^{-\frac{1}{2}} = \mathbf{PD}^{-\frac{1}{2}}\mathbf{P}^{-1} \qquad \text{(B12.3)}$$

Donde $\mathbf{D}^{-\frac{1}{2}} = diag(\lambda_1^{-\frac{1}{2}},...,\lambda_n^{-\frac{1}{2}})$, ($\lambda_i > 0 \quad i = 1,2,3,...,n$)

Teorema B12.3: Suponga que **A** tiene los vectores propios $\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_n$ no nulos pertenecientes a los valores propios distintos $\lambda_1,.....,\lambda_n$. Entonces $\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_n$ son linealmente independientes.

Corolario B12.4: Sea **A** una matriz cuadrada de dimensiones *nxn* cuyo polinomio característico es el producto de n factores distintos ($\lambda - \lambda_i$). Entonces A es similar a una matriz diagonal.

Una **matriz ortogonal** es necesariamente cuadrada e invertible*:* $A^{-1} = \mathbf{A}^T$. La inversa de una matriz ortogonal es una matriz ortogonal. El producto de dos matrices ortogonales es una matriz ortogonal. El determinante de una matriz ortogonal vale +1 ó -1. Cumpliéndose que $\mathbf{AA}^T$ es igual a la matriz identidad **I**.

Los teoremas anteriores constituyen la base del siguiente algoritmo.

B13. Algoritmo para determinar si una matriz es diagonalizable

Dada la matriz cuadrada **A** de dimensión *nxn*.

1.- Hallar el polinomio característico $p(\lambda)$ de **A**.

2.- Encontrar las raíces distintas $\{\lambda_i\}_{i=1}^{m}$ de la ecuación $p(\lambda)=0$.

3.- Para cada valor propio λ_i construir la matriz $\mathbf{M_i} = \mathbf{A} - \lambda_i \mathbf{I}$ y encontrar una base B_i para el espacio solución del sistema $\mathbf{M_i X} = 0$, la cual estará formada por vectores linealmente independientes.

4.- Determinar la dimensión *k* del conjunto $S = U_{i=1}^{m} B_i$. Se sabe que si $i \neq j$ entonces cualquier vector de B_i es linealmente independiente de cualquier vector contenido en B_j. El conjunto S contiene el mayor número posible de vectores propios linealmente independientes asociados a la matriz **A**.

5.- **A** es diagonalizable si y solo si k es igual a n, en cuyo caso la matriz **P** que contiene como columnas los vectores de S es tal que $\mathbf{P}^{-1}\,\mathbf{AP} = \mathbf{D} = \text{diag}(\lambda_1,....,\lambda_n)$.

B14. Matrices simétricas

Sea **A** una matriz de dimensión *nxm* con entradas complejas.

i) Su **matriz conjugada** $\overline{\mathbf{A}}$ se obtiene tomando el conjugado complejo de cada elemento de $\mathbf{A}$.

ii) La **matriz transpuesta conjugada** $\mathbf{A}^H$ de $\mathbf{A}$ se obtiene de transponer su conjugada $\overline{\mathbf{A}}$. Observando que $\mathbf{A}^H = (\overline{\mathbf{A}})^T = \overline{\mathbf{A}^T}$.

iii) Una **matriz cuadrada compleja A** es Hermética si y solo si $\mathbf{A}^H = \mathbf{A}$.

iv) Si **u** y **v** son vectores columna complejos de dimensión *n*, su producto interno se define como $<\mathbf{u},\mathbf{v}>=\mathbf{u}^T\overline{\mathbf{v}}$.

v) Una matriz cuadrada compleja es **unitaria** si y solo si $\mathbf{A}\mathbf{A}^H = \mathbf{I}$. Cuando **A** es real, unitaria equivalente a ortonormal.

vi) Una matriz cuadrada compleja **A** es **normal** si y solo si $\mathbf{A}\mathbf{A}^H = \mathbf{A}^H\mathbf{A}$.

Teorema B14.1. Sea **A** una matriz simétrica real. Cada raíz de su polinomio característico es real.

Teorema B14.2. Sea **A** una matriz simétrica real con dos valores propios no nulos (o cero) distintos λ_1 y λ_2 que tienen asociados los vectores característicos **u** y **v**. Entonces **u** y **v** son ortogonales.

Teorema B14.3. Sea **A** una matriz real simétrica de dimensión *nxn*. Entonces **A** tiene *n* vectores propios ortonormales.

Corolario B14.4 Sea **A** una matriz real simétrica de dimensión *nxn*. Existe una matriz ortonormal **P** tal que $\mathbf{P}^{-1}\,\mathbf{AP}$ es diagonal.

APENDICE "C"

CONCEPTOS DE TEORÍA DE LA INFORMACIÓN

C1. Introducción

La Teoría de la Información es una teoría matemática creada por Claude Shannon en el año 1948 y que forma la piedra angular sobre la que se ha desarrollado toda la teoría actual de la comunicación y la codificación. Esta teoría establece los límites de cuánto se puede comprimir la información y de cuál es la máxima velocidad a la que se puede transmitir información. La Teoría de la Información es, por tanto una teoría de límites alcanzables: máxima compresión de datos y máxima tasa de transmisión de información transmitida sin errores. Las aplicaciones de esta teoría son enormes y abarcan desde las ciencias de la computación (criptografía, aprendizaje), la ingeniería eléctrica (Teoría de la comunicación y teoría de la codificación), la estadística o la biología (secuencias de ADN, código genético).

Una gran parte de los métodos de ICA se basan en conceptos de la Teoría de la información que se presentan en este apéndice. Pueden encontrarse más detalles en las referencias [COV91, MAC93 y WON00].

C2. Entropía

La información se encuentra muy relacionada a la aleatoriedad o "sorpresa" de un resultado. Por ejemplo considere el caso de lanzar una moneda al aire. La probabilidad que caiga cara o sol es del 50%, se dice que este experimento es equiprobable. Y conocer previamente el resultado, no resulta posible si no se cuenta con mas información.

Shannon propuso la entropía como una medida apropiada de la incertidumbre. La entropía es definida como:

$$H(X) = \sum_{x \in A_x} P(x) \log \frac{1}{P(x)} \qquad \text{(C2.1)}$$

Donde *X* es el conjunto de la variable aleatoria *x*, donde éste puede tomar posibles valores. Cuando se usa el logaritmo natural o neperiano, la unidad de información es el *nat*, y cuando se usa el logaritmo en base 2 la unidad es el *bit*. La Figura 1 muestra la grafica de la entropía en donde se observa, que la máxima entropía se alcanza cuando la probabilidad es del 50% y de cero cuando se conoce certeramente su valor. Es decir, que cuando en la situación no hay "sorpresas" el evento no lleva "información" ya que se sabe a priori cual será el mensaje. Mientras que, al aumentar el nivel de "incertidumbre" entonces habría mas "sorpresa" y por lo tanto mas información. En consecuencia, las palabras "incertidumbre", "sorpresa" e "información" están relacionadas.

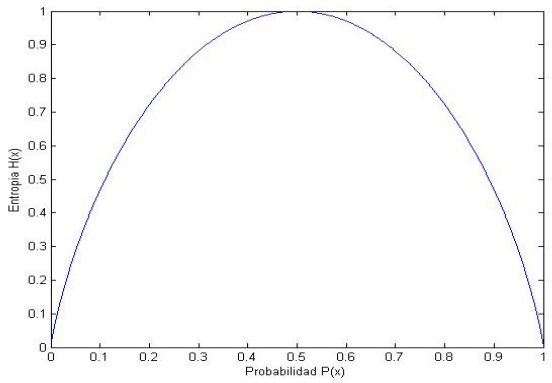


Fig. 1. Grafica de la entropía para el lanzamiento de una moneda.

Note que la Figura es una curva simétrica y alcanza su valor máximo cuando se obtiene la máxima incertidumbre.

C3.- Entropía de juntura, marginal y condicional

La *entropía de juntura* es una medida de la entropía y mide la información contenida en un sistema de dos o mas variables. La entropía de juntura se define como:

$$H(X,Y) = \sum_{x,y \in A_x A_y} P(x,y) \log \frac{1}{P(x,y)} \qquad \text{(C3.1)}$$

Donde *P(x,y)* es la función de probabilidad de juntura de las variables *x* y *y*. Sí estas dos variables aleatorias fueran independientes, se cumpliría la siguiente condición $P(x,y) = P(x)P(y)$. Entonces se obtendría que la entropía resultaría en:

$$\begin{aligned} H(X,Y) &= \sum_{x,y \in A_x A_y} P(x)P(y) \log \frac{1}{P(x)P(y)} \\ &= \sum_{x,y \in A_x A_y} P(x)P(y) \log \frac{1}{P(x)} + \sum_{x,y \in A_x A_y} P(x)P(y) \log \frac{1}{P(y)} \\ &= \sum_{x \in A_x} P(x)(\sum_{y \in A_y} P(y)) \log \frac{1}{P(x)} + \sum_{y \in A_y} P(y)(\sum_{x \in A_x} P(x)) \log \frac{1}{P(y)} \\ &= \sum_{x \in A_x} P(x) \log \frac{1}{P(x)} + \sum_{y \in A_y} P(y) \log \frac{1}{P(y)} \end{aligned}$$

$$H(X,Y) = H(X) + H(Y) \qquad \text{(C3.2)}$$

Esto demuestra que la *entropía de juntura* para dos variables independientes, es la suma de sus *entropías marginales* o individuales.

Para medir la incertidumbre de una variable *X* dada otra variable *Y* , se define la *entropía condicional* de *X* dado *Y* , escribiéndose como *H(X\Y)*. La *entropía condicional H(X\Y)* representa la cantidad de incertidumbre que existe en *X* después de haber observado a *Y* , y se define como:

$$H(X\backslash Y) = H(X,Y) - H(Y) \qquad \text{(C3.3)}$$

con:

$$0 \leq H(X\backslash Y) \leq H(X)$$

La diferencia *H(X) - H(X\Y)* representa la incertidumbre de la entrada que se va a obtener observando la salida. Esta cantidad se denomina *información mutua* entre las variables aleatorias *X* e *Y*. Denotándola como *I(X;Y).*

C4.- Información mutua

La *información mutua* es la medida de información que tiene una variable con respecto a otra. Para dos variables aleatorias, se tiene que:

$$I(X;Y) = H(X) - H(X \mid Y) \qquad \text{(C4.1)}$$

La *información mutua I(X;Y)* es simétrica y no negativa. donde *H(Y\X)* es la entropía condicional. El primer miembro de la (C4.1) es la media de la información dada por la salida del sistema *X*, menos la media de la información dada por *Y* suponiendo que conocemos la entrada *X*.

La Figura 2 muestra una interpretación visual de la entropía de la entrada *X*, representada con el círculo de la izquierda, la entropía de la salida *Y* se representa con el círculo de la derecha y la información mutua entre *X* e *Y* se representa con la intersección de ambos círculos.

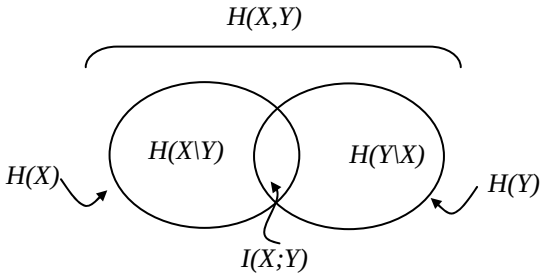


Fig. 2 Relación entre la información mutua *I(X;Y)* y las entropías *H(X)* y *H(Y).*

C5 Divergencia de Kullback-Leibler

La divergencia Kullback-Leibler o entropía relativa, determina la distancia entre dos densidades de probabilidad. Tomando valores positivos, e iguales que cero cuando las dos distribuciones son semejantes. Esta es una consecuencia directa de la estricta convexidad de el logaritmo negativo, y la aplicación de la clásica desigualdad de Jensen.

$$E\{f(y)\} \geq f(E\{y\}) \tag{C5.1}$$

Asignando a *f(y) = - log (y)* y asumiendo que $y = p^2(x)/p^1(x)$, se tendrá que:

$$E\{f(y)\} = E\left\{-\log\frac{p^2(x)}{p^1(x)}\right\} = \int p^1(x)\left\{-\log\frac{p^2(x)}{p^1(x)}\right\}dx \tag{C5.2}$$

$$f(E\{y\}) = -\log\int p^1(x)\left\{\frac{p^2(x)}{p^1(x)}\right\}dx = -\log\int p^2(x)dx = 0 \tag{C5.3}$$

Obteniendo la igualdad si las dos distribuciones son semejantes. La divergencia de Kullback-Leibler, no es propiamente una medida de la distancia.

La divergencia de Kullback-Leibler tiene las siguientes propiedades importantes: La información Mutua es positiva y cero si y solo si las variables son independientes. Esto es una consecuencia directa de la divergencia de Kullback-Leibler.

$$D(P \,||\, Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)} \tag{C5.4}$$

La información mutua es una forma especial de la entropía relativa.

$$I(X,Y) = D(P(x,y) \,||\, P(x)P(y)) \tag{C5.5}$$

APENDICE "D"

Código del algoritmo INFOMAX para el DSP

El código que se implemento en el DSP mediante el Code Componer, se muestra a continuación

```
/*
 * ECG.c
 * Real-Time Workshop code generation for Simulink model "ECG.mdl".
 *
 * Model Version              : 1.448
 * Real-Time Workshop version : 6.0  (R14)  05-May-2004
 * C source code generated on : Fri Jul 29 16:33:38 2005
 */

#include "ECG.h"
#include "ECG_private.h"

/* Block signals (auto storage) */
#pragma DATA_ALIGN(ECG_B,8)
BlockIO_ECG ECG_B;

/* Block states (auto storage) */
#pragma DATA_ALIGN(ECG_DWork,8)
D_Work_ECG ECG_DWork;

/* Real-time model */
rtModel_ECG ECG_M_;
rtModel_ECG *ECG_M = &ECG_M_;

/* Model output function */
static void ECG_output(int_T tid)
{

  /* local block i/o variables */

  real_T rtb_DelayMix[64];
  real_T rtb_Gain[64];
  real_T rtb_Sum_d[64];
  real_T rtb_Sum1[64];
  real_T rtb_DelayLine2[64];
  real_T rtb_MathFunction[64];
  real_T rtb_DelayLine1[64];
  real_T rtb_MathFunction1[64];
  real_T rtb_Switch1[64];
  real_T rtb_Switch2[64];
  real_T rtb_temp32[64];
  boolean_T rtb_Switch[3];
```

```
  /* S-Function Block: <Root>/ADC (c6711dsk_adc) */
  {
    const real_T ADCScaleFactor = 1.0 / 32768.0;
    const short *restrict i_buf = (const short*) adcBuffPtr;
    real_T *restrict o_buf = ECG_B.ADC;
    int_T i=0;

#pragma UNROLL(2)
#pragma MUST_ITERATE(64,64,64)
     for (i=0; i<64; i++) {
       o_buf[i] = (real_T)(int) i_buf[i] * ADCScaleFactor;
     }
  }

  /* Gain: '<Root>/Delay Mix' incorporates:
   *  Sum: '<S3>/Sum'
   *  Constant: '<S3>/Constant1'
   *
   * Regarding '<Root>/Delay Mix':
   *   Gain value: ECG_P.DelayMix_Gain
   */
  {
    int_T i1;

/*
 * Abstract:
 *   Scheduler-related functions and main() function.
 *   This file is auto-generated by Embedded Target for TI C6000(tm) DSP.
 *
 * Compiler-specified defines:
 *        RT             - Required.
 *  MODEL=modelname - Required.
 *        NUMST=#        - Required. Number of sample times.
 *        NCSTATES=#     - Required. Number of continuous states.
 *  TID01EQ=1 or 0  - Optional. Only define to 1 if sample time task
 *                    id's 0 and 1 have equal rates.
 *  MULTITASKING    - Optional. (use MT for a synonym).
 */

/*=========*
 * Headers *
 *=========*/

#include <float.h>
#include <stdlib.h>
#include <string.h>
#include "rtwtypes.h"
# include "rtmodel.h"
#include "rt_sim.h"
#include "rt_nonfinite.h"
```

```
#include "c6000_main.h"
#include "ECGcfg.h"

/*====================*
 * External functions *
 *====================*/
extern RT_MODEL *MODEL(void);

extern void MdlInitializeSizes(void);
extern void MdlInitializeSampleTimes(void);
extern void MdlStart(void);
extern void MdlOutputs(int_T tid);
extern void MdlUpdate(int_T tid);
extern void MdlTerminate(void);

/*===================================*
 * Global data                       *
 *===================================*/

struct {
  int_T stopExecutionFlag;
  volatile int_T isrOverrun;
  int_T overrunFlags[NUMST];
  const char_T *errmsg;
} GBLbuf = {NULL,0,0,0};

RT_MODEL *S;

/*===================================*
 * Extern variables                  *
 *===================================*/
extern volatile int pendingInterrupt;

/* Function: rt_OneStep --------------------------------------------
 *
 * Abstract:
 *      Perform one step of the model.  Single-tasking implementation.
 */
static void rt_OneStep(void)
{
  real_T tnext;

  /*********************************************
   * Check and see if base step time is too fast *
   *********************************************/
```

```
  if (GBLbuf.overrunFlags[0]++) {
    GBLbuf.stopExecutionFlag = 1;
    return;
  }
  /*********************************************
   * Check and see if error status has been set  *
   *********************************************/

  if (rtmGetErrorStatus(S) != NULL) {
    GBLbuf.stopExecutionFlag = 1;
    return;
  }

  tnext = rt_SimGetNextSampleHit();
  rtsiSetSolverStopTime(rtmGetRTWSolverInfo(S),tnext);

  MdlOutputs(0);

  MdlUpdate(0);
  rt_SimUpdateDiscreteTaskSampleHits(rtmGetNumSampleTimes(S),
    rtmGetTimingData(S),
    rtmGetSampleHitPtr(S),
    rtmGetTPtr(S));

  GBLbuf.overrunFlags[0]--;
}                                /* end rtOneStep */

/* Function: exitprocessing ---------------------------------
 *
 * Abstract:
 *      Perform various tasks at program exit.
 */

void exitprocessing()
{
  disable_DMA();
  disable_interrupts();
  UTL_halt();
}

/* Function: singleRateStep ---------------------------------
* Abstract: This function is called from the software interrupt (SWI)
 *          in DSP/BIOS and serves as a wrapper for rt_OneStep.
 */

int_T singleRateStep(void)
{
```

```
pendingInterrupt = FALSE;

if (!GBLbuf.stopExecutionFlag &&
 (rtmGetT(S) <= rtInf)) {

  if (rtmGetStopRequestedFlag(S)) {
    LOG_printf(&LOG_MW1, "Stop requested.");

    exitprocessing();
  }

  rt_OneStep();

  if (pendingInterrupt) {
    GBLbuf.isrOverrun = 1;
  }

  if (GBLbuf.isrOverrun) {

    /* turn on board LED(s) */
    *(volatile unsigned int *)(0x90080000) = 0;

    /* Halt */
    exitprocessing();
  }

  return(EXIT_SUCCESS);
}

/*********************
 * Clean up and exit *
 *********************/

LOG_printf(&LOG_MW1, "**stopping the model**");

if (rtmGetT(S) >= rtInf) {
  LOG_printf(&LOG_MW1, "Requested stop time has been reached.");
}

if (rtmGetErrorStatus(S) != NULL) {
  LOG_printf(&LOG_MW1, "Error detected.");

  exitprocessing();
  exit(EXIT_FAILURE);
}
```

```
  MdlTerminate();
  exitprocessing();
  return(EXIT_SUCCESS);
}

/* Function: main -------------------------------------------
 *
 * Abstract:
 *      Set up board and execute model code.
 */
int_T main(void)
{
  const char *status;

  /************************
   * Initialize the model *
   ************************/
  rt_InitInfAndNaN(sizeof(real_T));

  S = MODEL();
  if (rtmGetErrorStatus(S) != NULL) {
    LOG_printf(&LOG_MW1, "Error during model registration");

    exitprocessing();
    exit(EXIT_FAILURE);
  }

  rtmSetTFinal(S,rtInf);

  MdlInitializeSizes();
  MdlInitializeSampleTimes();

  status = rt_SimInitTimingEngine(rtmGetNumSampleTimes(S),
   rtmGetStepSize(S),
   rtmGetSampleTimePtr(S),
   rtmGetOffsetTimePtr(S),
   rtmGetSampleHitPtr(S),
   rtmGetSampleTimeTaskIDPtr(S),
   rtmGetTStart(S),
   &rtmGetSimTimeStep(S),
   &rtmGetTimingData(S));

  if (status != NULL) {
    LOG_printf(&LOG_MW1, "Failed to initialize sample time engine");

    exitprocessing();
    exit(EXIT_FAILURE);
```

```
  }

  MdlStart();
  if (rtmGetErrorStatus(S) != NULL) {
    GBLbuf.stopExecutionFlag = 1;
  }

  /* turn off board LED(s) */
  *(volatile unsigned int *)(0x90080000) = 0x07000000;

  relocate_ISVT();

  turnOn_L2Cache();

  LOG_printf(&LOG_MW1, "**starting the model**");

  enable_interrupts();

  /* Drop out of main() and enter DSP/BIOS Kernel */
}                                  /* end main() */

/* [EOF] ECG_main.c */

/*
 * ECG_data.c
 *
 * Real-Time Workshop code generation for Simulink model "ECG.mdl".
 *
 * Model Version              : 1.448
 * Real-Time Workshop version : 6.0  (R14)  05-May-2004
 * C source code generated on : Fri Jul 29 16:33:38 2005
 */

#include "ECG.h"
#include "ECG_private.h"

/* Block parameters (auto storage) */
#pragma DATA_ALIGN(ECG_P,8)
Parameters_ECG ECG_P = {
  0.01 ,                           /* Constant1_Value : '<S3>/Constant1' */
  5.0 ,                           /* DelayMix_Gain : '<Root>/Delay Mix' */
  1.0 ,                           /* Gain1_Gain : '<Root>/Gain1' */
  0.18 ,                           /* SineWave_Amplitude : '<Root>/Sine Wave' */
  0.0075 ,                           /* SineWave_Frequency : '<Root>/Sine Wave' */
  0.0 ,                           /* SineWave_Phase : '<Root>/Sine Wave' */
  4.0 ,                           /* Gain_Gain : '<Root>/Gain' */
  0.8 ,                           /* Gain2_Gain : '<Root>/Gain2' */
```

```
  0.5 ,                    /* Gain3_Gain : '<Root>/Gain3' */
  1.0 ,                    /* Gain4_Gain : '<Root>/Gain4' */
  0.008 ,                    /* Constant1_Value_e : '<Root>/Constant1' */
  0.0 ,                    /* DelayLine2_IC : '<Root>/Delay Line2' */
  3.0 ,                    /* Constant_Value : '<S6>/Constant' */
  0.0 ,                    /* DelayLine1_IC : '<Root>/Delay Line1' */
  0.0 ,                    /* DelayLine_IC : '<Root>/Delay Line' */
  0.00008 ,                    /* Constant_Value_f : '<Root>/Constant' */
  3.0 ,                    /* Constant1_Value_f : '<S5>/Constant1' */
  0.0 ,                    /* DelayLine3_IC : '<Root>/Delay Line3' */
  0.8 ,                    /* Gain6_Gain : '<Root>/Gain6' */
  0.8 ,                    /* Gain5_Gain : '<Root>/Gain5' */
  0.5                      /* Gain7_Gain : '<Root>/Gain7' */
};

#include <std.h>
#include <csl.h>
#include <csl_edma.h>
#include <csl_mcbsp.h>
#include <csl_irq.h>
#include "MW_c6xxx_bsl.h"
#include "ECGcfg.h"

/* Function: Read_Codec_Control ----------------------------------------
 *
 * Abstract:
 *      This function reads Control Word from Codec
 */

unsigned int Read_Codec_Control(unsigned int Register)
{
  unsigned int Register_temp = 0;

  Register_temp = ( ((Register & 0x001F) << 8) | 0x2000);

  mcbsp0_write(0);
  mcbsp0_read ( );
  mcbsp0_write(1);
  mcbsp0_read ( );
  mcbsp0_write(Register_temp);
  Register_temp = mcbsp0_read ( );
  mcbsp0_write(0);
  mcbsp0_read ( );

  return Register_temp;
}
```

```
/* Function: Write_Codec_Control ---------------------------------------
 *
 * Abstract:
 *      This function writes Control Word from Codec
 */

void Write_Codec_Control(unsigned int Register, unsigned int Data)
{
  unsigned int Register_temp = 0;

  Register_temp = ( (Register & 0x001F) << 8) | (Data & 0x00ff);

  mcbsp0_write(0);
  mcbsp0_read ( );
  mcbsp0_write(1);
  mcbsp0_read ( );
  mcbsp0_write(Register_temp);
  mcbsp0_read ( );
  mcbsp0_write(0);
  mcbsp0_read ( );
}

/* Function: codec_error ----------------------------------------------
 *
 * Abstract:
 *      This function traps Codec Errors
 */

void codec_error (int id)
{
  /* Codec Initialization Error - Exit and Reset DSK */
  //for (;;);                                          /* loop forever */
}

/* Function: config_codec ---------------------------------------------
* Abstract:
 *      This function initally configures the codec
 */

void config_codec(void)
{
  unsigned int temp_variable;

  /* Performs Voice Channel Initialisation of TLC320AD535 Codec        */
  /* AD535 has 2 serial port channels - Data, Voice                    */
  /* Data  Channel Controlled by Registers 1,2        (Reg 0 = NOP)    */
  /* Voice Channel Controlled by Registers 3,4,5,6                     */
```

```
  /* Only Voice channel used on DSK 6211                                    */

  /* Set-Up Register 0 (NOP) - Dummy Read/Write Codec */
  Write_Codec_Control(0, 0);
  temp_variable = Read_Codec_Control(0);

  /* Set-Up Register 1 / 2   - Only Used by Data Serial Port -NA    */

  /* Set-Up Register 3       - S/W Reset + Power Down + No loop / gain=0dB */
  Write_Codec_Control(3, 0x00C6);       // + With    Reset
  Write_Codec_Control(3, 0x0006);       // + Without Reset
  temp_variable = Read_Codec_Control(3);
  if ( (temp_variable & 0x00ff) != 0x0006)
  codec_error(3);

  /* Set-Up Register 4       - Voice ADC gain = 0dB */
  Write_Codec_Control(4, 0x0040);
  temp_variable = Read_Codec_Control(4);
  if ( (temp_variable & 0x00ff) != 0x0040)
  codec_error(4);

  /* Set-Up Register 5       - Spkr L/R gain = 0dB */
  Write_Codec_Control(5, 0x0002);
  temp_variable = Read_Codec_Control(5);
  if ( (temp_variable & 0x00fe) != 0x0002)
  codec_error(5);

  /* Set-Up Register 6       - Handset gain  = 0dB */
  Write_Codec_Control(6, 0x0000);
  temp_variable = Read_Codec_Control(6);
  if ( (temp_variable & 0x0080) != 0x0000)
  codec_error(6);

  /* Set-Up Register - (NOP)            */
  Write_Codec_Control(0, 0);
  temp_variable = Read_Codec_Control(0);
}

/* Function: codec_init --------------------------------
 *
 * Abstract:
 *      Initialize the codec
 */

void codec_init()
{
  initDMABuffers();
```

```
  config_McBSP();
  config_codec();
}

/* Function: config_codec_input ----------------------------------------
 *      This function configures the codec input characteristics
 *      based on model characteristics
 */

void config_codec_input(void)
{
  unsigned int temp_variable;
  /* Set-Up Register 4 */
  Write_Codec_Control(4, 0x0000 | CODEC_MIC_GAIN_OFF | CODEC_ADC_GAIN);
  temp_variable = Read_Codec_Control(4);
  if ( (temp_variable & 0x00ff) != (0x0000 | CODEC_MIC_GAIN_OFF |
    CODEC_ADC_GAIN) )
  codec_error(4);
}

/* Function: config_codec_output ---------------------------------------
 *      This function configures the codec output characteristics
 *      based on model characteristics
 */

void config_codec_output(void)
{
  unsigned int temp_variable;
  /* Set-Up Register 5 - Spkr L/R gain */
  Write_Codec_Control(5, 0x0000 | CODEC_SPK_ON | CODEC_DAC_ATTN);
  temp_variable = Read_Codec_Control(5);
  if ( (temp_variable & 0x00fe) != (0x0000 | CODEC_SPK_ON | CODEC_DAC_ATTN) )
  codec_error(5);
}
```

APENDICE "E"

Apéndice de los algoritmos ICA

En este apéndice, se muestran los algoritmos de ICA tratados en la tesis y son mostrados en pseudolenguaje, para su programación.

ICA por descorrelación no lineal
1.- Centrado de los datos, esto es hacer su media cero. 2.- Elegir una matriz inicial **W** = **I** (matriz identidad) 3.- Calcular $\hat{s}_i(t) = e_i(t) - \sum_{i=1}^{n} w_{ij}(t)\hat{s}_j(t)$ $(i, j = 1,2,...,n) \qquad i \neq j$ 4.- Calcular $\Delta w_{ij}(t) = \eta \cdot \tanh(\hat{s}_i(t)) \cdot (\hat{s}_j(t))^3$, donde η es la razón de aprendizaje y se propuso de 0.0004. 5.- Calcular $w_{ij}(t + \Delta t) = w_{ij}(t) + \Delta w_{ij}(t)$. 6.- Actualizar $w_{ij}(t) = w_{ij}(t + \Delta t)$ e ir al paso 3.

ICA basado en la maximización de la información (INFOMAX)
1.- Centrado de los datos, esto es hacer su media cero. 2.- Elegir una matriz inicial **W** = **I** (matriz identidad) 3.- Calcular $\hat{\mathbf{s}}(t) = \mathbf{W}(t)\mathbf{e}(t)$ $(i, j = 1,2,...,n) \qquad i \neq j$ 4.- Calcular $\Delta\mathbf{W}(t) \propto \left[\mathbf{I} - \varphi(\mathbf{u}(t))\mathbf{u}^T(t)\right]$, donde $\varphi(\mathbf{u})$ = *tanh(u)*. 5.- Calcular $\mathbf{W}(t + \Delta t) = \mathbf{W}(t) + \Delta\mathbf{W}(t)$. 6.- Actualizar $\mathbf{W}(t) = \mathbf{W}(t + \Delta t)$ e ir al paso 3.

ICA basado en la maximización de la no-gaussianidad por curtosis usando el gradiente
1.- Centrado de los datos, esto es hacer su media cero. 2.- Blanquear los datos y asignarlos como **z**. 3.- Elegir un vector inicial **w** de norma unitaria 4.- $\Delta \mathbf{w} \propto sign(Kurt(\mathbf{w}^T\mathbf{z}))E\{\mathbf{z}(\mathbf{w}^T\mathbf{z})^3\}$ 5.- Normalizar $\mathbf{w} \leftarrow \mathbf{w}/\|\mathbf{w}\|$. 6.- Si no converge ir al paso 4.

ICA basado en la maximización de la no-gaussianidad por curtosis de punto fijo
1.- Centrado de los datos, esto es hacer su media cero. 2.- Blanquear los datos y asignarlos como **z**. 3.- Elegir un vector inicial **w** de norma unitaria 4.- $\mathbf{w} \leftarrow E\{\mathbf{z}(\mathbf{w}^T\mathbf{z})^3\} - 3\mathbf{w}$ 5.- Normalizar $w \leftarrow \mathbf{w}/\|\mathbf{w}\|$. 6.- Si no converge ir al paso 4.

ICA basado en la maximización de la no-gaussianidad por negentropía usando el gradiente
1.- Centrado de los datos, esto es hacer su media cero. 2.- Blanquear los datos y asignarlos como **z**. 3.- Elegir un vector inicial **w** de norma unitaria 4.- $\mathbf{w} \leftarrow E\{\mathbf{z}g(\mathbf{w}^T\mathbf{z})\} - E\{g'(\mathbf{w}^T\mathbf{z})\}\mathbf{w}$, donde g es definido en 1.7.40 – 1.7.42 y g' en 1.7.48 y 1.7.50 5.- Normalizar w$\leftarrow \mathbf{w}/\|\mathbf{w}\|$. 6.- Si no converge ir al paso 4.

ICA basado en la maximización de la no-gaussianidad por negentropía de punto fijo
1.- Centrado de los datos, esto es hacer su media cero. 2.- Blanquear los datos y asignarlos como **z**. 3.- Elegir un vector inicial **w** de norma unitaria y un valor inicial para γ. 4.- Actualizar $\Delta\mathbf{w} \propto \gamma\mathbf{z}g(\mathbf{w}^T\mathbf{z})$, donde g es definido en 1.7.40 – 1.7.42. 5.- Normalizar w$\leftarrow \mathbf{w}/\|\mathbf{w}\|$. 6.- Si el sign de γ no se conoce, actualizar $\Delta\gamma \propto (G(\mathbf{w}^T\mathbf{z}) - E\{G(v)\}) - \gamma$ 7.- Si no converge ir al paso 4.

Referencias bibliografícas

[BEL95] Bell, A. J. and Sejnowski, T.J. *"An Information-Maximization Approach to Blind Separation and Blind Deconvolution"*. Neural Computation, 7:1129-1159. 1995.

[CAR97] Cardoso, J-F. *"Infomax and maximum likelihood for blind source separation"*. IEEE Signal processing Letters. pp(4): 112-114 1997.

[CHA03] Chang-Min Kim, H. Park, T. Kim, Y. Choi, S Lee, *"FPGA Implementation of ICA Algorithm for Blind Signal Separation and Adaptive Noise Canceling"*, IEEE Transactions on Neural Network, Vol 14, No. 5, pp 1038-1046, 2003.

[COH92] M. H. Cohen, A. G. Andreou, *"Current-mode subthreshold MOS implementation of the Hérault-Jutten autoadaptive network"*. IEEE journal of solid-state circuits, vol. 27, nº 5, May 1992.

[COO71] G. R. Cooper. "Probabilistic Methods of Signal and System Analysis" **Holt, Rinehart and Winston Inc**, 1971.

[COV91] Thomas M. Cover "Elements of Information Theory" **A Wiley Interscience Publication**. 1991.

[DIA96] K.I. Diamataras, S. Y. Kung "Principal Components Neural Networks theory and applications ", **John wiley & sons, Inc**. 1996.

[GAE90] Gaeta, M. & Lacoume, J-L. *"Source Separation Without priori Knowledge: tha Maximum likelihood solucion"*. pp(621-624). Proceedings of EUSIPO. 1990.

[GOL93] Gene H. Golub, Charles F. Van Loan, "Matrix Computations", **The johns Hopkins University Press**, 1993.

[HAG96] M. T. Hagan, H. B. Demuth, M. H. Beale ."Neural Network Design" **PWS Publishing Company**. 1996.

[HAM91] Richar W. Hamming ."The Art of Probability for scientists and engineers ". **Addison-Wesley Publishing Company, inc**. 1991.

[HAY99] S. Haykin. "Neural Networks: a comprehensive foundation". **Prentice Hall, inc**. 1999 pp 680-690.

[HYV00] A. Hyvärinen, E. Oja, *"Independent Component Analysis: algorithms and applications"*. Elsevier Science Ltd. Neural Networks. pp(411-430).2000.

[HYV03] A. Hyvärinen, J. Karhunen, E. Oja. "Independent Component Analysis**" John Wiley & Sons**, Inc. 2003.

[HYV98] A. Hyvärinen, *"The FastICA MATLAB toolbox."* Helsinki Univ. of Technology.

[HYV99] A. Hyvärinen, *"Fast and robust fixed-point algorithms for independent component analysis"*. IEEE transactions on Neural Networks. pp(626-634).1999.

[JOL86] I.T. Jollife, "Principal Component Analysis", **Springer – Verlag New York Inc**. 1986.

[JON03] John S Ebersole,Timothy A Pedley. "Current Practice of Clinical Electroencephalography (Hardcover)". **Lippincott Williams & Wilkins**. 2003.

[JUT00] C. Jutten. *"Source Separation: from dusk till dawn"*. In Proc. 2da Int. Workshop on Independent Component Analysis and Blind Source Separation (ICA'2000), pp 15-26, Helsinki, Finland, 2000.

[JUT91a] C. Jutten and J. Herault. *"blind separation of sources, Part I: An adaptive algorithm based on neuromimetic architecture"*. Signal processing Vol. 24 (pp 1-10). New York: Elsevier Science. 1991.

[JUT91b] C. Jutten and Herault, *"Blind separation of source, Part II : Problems statement"*. Signal processing, Vol. 24. (pp. 11-20). New York: Elsevier Science. 1991.

[KULL51] S. Kullback and R. A. Leibler. *"On information and sufficiency. Ann. Math. Stat"*., 22: 79-86, 1951.

[LNO06] L.N.Oliva, O. A.Cárdenas, J.A. Moreno, L.M. Flores ,F.Gomez. *"Implementación de un Algoritmo ICA con circuitos CMOS Analógicos para el problema BSS"*.XII International Workshop. 2006.

[MAC03] David J.C. Mackay. "Information Theory, Inference and Learning Algorithms", **Cambridge University Press**, 2003.

[MAR92]Marc H. Cohen & Andreas G. Andreou. (1992), *"CurrentMode Subthreshold MOS Implementation of the HeraultJutten Autoadaptive Network"*, IEEE Journal of Solid-State Circuits 26 (5):714-727. 73

[OJA05] http://www.cis.hut.fi/oja/, Sitio web by Erkki Oja.

[PAP91] A. Papoulis. "Probability, Random Variables, and Stochastic Processes" **Mc Graw Hill, 3er edition**, 1991.

[PEA97] Pearlmutter, B. and Parra, L. *"Maximum likelihood blind source separation: A context-sensitive generalization of ICA"*. In advances in Neural Information Processing Systems 9.pp 613-619.1997.

[PER02] Perry, Douglas L. "VHDL: Programming by Example **" McGraw-Hill**, 2002.

[PUN95] C.G. Puntonet, A. Prieto, C. Jutten, M. Rodríguez-Alvarez and J. Ortega. *"Separation of sources: a geometry-based procedure for reconstruction of n-valued signal"*. Signal Processing. Vol.46, No. 3. pp. 267-284.1995.

[SAB91] Sabine Van Huffel, Joos Vandewalle "The Total Least Squares Problem Computational Aspects and Analysis". **Katholieke Universiteit Leuven. Philadelphia**, 1991.

[SEO01]Ki-seok Cho,Soo-young Lee *"Implementation of Infomax ICA Algorithm with Analog CMOS Circuits"*. CiteSeerPSU:267579. 2001

[SMI01] Smith, D. "HDL Chip Design." **Doone Publications**, USA, 2001

[STE90] Steven J. Leon "Linear Algebra With Applications" **Prentice Hall**. 1990.

[STR82] Gilbert Strang "Algebra lineal y sus aplicaciones", **Massachusetts Institute of Technology, Fondo Educativo Interamericano**, 1982.

[TAL99] A. Taleb And C. Jutten. *"Source Separation in Post-nonlinear mixture"*. IEEE trans. on Signal Processing, 47(10): 2807-2820, 1999.

[TI05] www.ti.com. Sitio web de Texas Instruments

[WIL68] William Feller . "An introduction to probability theory and its applications". **Wiley International Edition**. 1968.

[WON99] Te-Won Lee, M. Girolami and T. Sejnowski, *"Independent Component Analysis Using an Extended Infomax Algorithm for Mixed Sub-Gaussian and Super- Gaussian Sources"*, Neural Computation, Vol: 11(2), 409-433.1999.
http:// www.cis.hut.fi/proyects/ica/fastica/.

[WON00a] Te-Won Lee "Independent Component Analysis Theory and applications " **Kluwer Academic Publishers**. 2000.

[WON00b] Te-Won Lee, M. Girolami and T. Sejnowski, "*A Unifying Information-Theoretic Framework for Independent Component Analysis*", Computers and Mathematics with Applications", Vol: 39, pp 1-21.2000.

[WON00c] http://www.cnl.salk.edu/~tewon/ica_cnl.html , Sitio web by by Te-Won Lee, Mark Girolami and Terry Sejnowski.