Ingeniería en Computación

Proyecto Tecnológico

Estudio experimental de los momentos de desvanecimiento de funciones wavelet generadas con filtros digitales

Diego Alejandro Alcántara Díaz

209332294

Dr. Oscar Herrera Alcántara

Trimestre 15 Invierno

Fecha: 24 de Abril del 2015

Declaratoria

Yo, Oscar Herrera Alcántara, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Yo, Diego Alejandro Alcántara Díaz, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Tabla de contenido (Índice)

Resumen

En este proyecto se realizaron experimentos con los momentos de desvanecimiento de los wavelets generados con filtros paramétricos de reconstrucción perfecta de tamaño 12. Se hicieron las pruebas con un algoritmo de enjambre de partículas, empezando con un momento de desvanecimiento hasta seis. Se encontraron valores de parámetros con los cuales se satisfacen en gran medida las restricciones de las ecuaciones de momentos de desvanecimiento, en la medida de lo posible, y que se aproximan a los valores encontrados por Ingrid Daubechies.

Introducción

Una de las técnicas frecuentemente usadas en aplicaciones de ingeniería es la de realizar una transformación de un problema de un espacio a otro, en el que el problema pueda tener una mejor representación, y permita su manipulación. Ejemplo de estas transformaciones son la transformada de Laplace y la transformada de Fourier. La primera es usada para analizar circuitos eléctricos que dan lugar a ecuaciones integro-diferenciales, y que al transformarlas se les puede dar un tratamiento algebraico, en tanto que la segunda permite analizar una función desde el punto de vista de sus componentes de frecuencia. Recientemente, la transformada wavelet ha mostrado ser otra herramienta de análisis para señales temporales al llevarlas al dominio tiempo-frecuencia, que la hace atractiva para analizar señales no estacionarias, en las que su distribución de probabilidades cambia al paso del tiempo. A diferencia de las transformadas de Laplace y Fourier, mencionadas anteriormente, la transformada wavelet no predefine una función base única sino un conjunto de familias de funciones, o condiciones (de admisibilidad), que dan lugar a una infinidad de opciones. Se han publicado trabajos en donde la idea es maximizar la suavidad de los wavelets tanto como sea posible [6], y también se ha publicado criterios paramétricos para generar familias de wavelets con propiedades de ortogonalidad (bases no redundantes).

Los criterios paramétricos incluyen el uso de ecuaciones trigonométricas, que varían sus parámetros en el intervalo [0, 2π), y que dan lugar a coeficientes de filtros de reconstrucción perfecta, los cuales a su vez dan lugar a wavelets, que son funciones base a partir de las cuales se pueden expresar otras funciones matemáticas. Cabe mencionar que no todos los valores de los parámetros generan funciones wavelet "suaves".

En el presente trabajo se aproximaron funciones wavelets conocidas como "Daubechies N", en donde N=4, 6, 8, 10 y 12 haciendo uso de ecuaciones paramétricas de filtros de reconstrucción perfecta.

Antecedentes

Proyectos Terminales

-Santiago Santiago, Teresa Dalia y Hernández Trejo, Antonio de Jesús. Sistema de identificación de personas a través del iris mediante análisis de texturas por filtrado de wavelets. Arturo Zúñiga López. [1]

Relación encontrada: En este proyecto terminal, se aplican técnicas de filtrado usando la transformada wavelet para procesar imágenes del iris y en base a ello realizar identificación de personas.

Diferencia: En el proyecto propuesto no se aplica la transformada wavelet, sino que se trabajará aproximando varias funciones wavelet a partir de filtros paramétricos de reconstrucción perfecta.

-Garduño Martínez, Javier Adrián. Estudio experimental de la aproximación de filtros digitales paramétricos para implementar la transformada wavelet discreta con polinomios evolutivos. Oscar Herrera Alcántara. [2]

Relación encontrada: En este proyecto lo que se propone es aproximar las ecuaciones paramétricas de filtros de reconstrucción perfecta usando polinomios.

Diferencia: En el proyecto propuesto se trabajará aproximando funciones wavelet a partir de filtros paramétricos de reconstrucción perfecta.

-Hernández Nieves, María Sara. Clasificación de llantos de bebé con wavelets. Oscar Herrera Alcántara. [3]

Relación encontrada: En este proyecto terminal se aplica la transformada wavelet discreta para procesar y clasificar señales de audio con llantos de bebé.

Diferencia: En el proyecto propuesto no se aplica la transformada wavelet, sino que se trabajará aproximando funciones wavelet que cumplan condiciones de desvanecimiento.

Tesis:
-Application of Wavelets to Filtering and Analysis of Self-Similar Signals de Karlton Edward Wirsing. [4]

Relación encontrada: En esta tesis se aplican métodos basados en funciones wavelets para detectar similitudes entre señales.

Diferencia: En el proyecto propuesto se pretende aproximar un conjunto de funciones wavelets a partir de filtros paramétricos de reconstrucción perfecta.

Artículos:
-Aplicación de Algoritmos Genéticos a la Compresión de Imágenes con Evolets de Herrera, O. y Mora, R. [5]

Relación encontrada: En éste artículo, se utilizan algoritmos junto con wavelets para la compresión de imágenes y éstas pierdan la menor definición posible.

Diferencia: En el proyecto propuesto, se propone aplicar un algoritmo de partículas junto con wavelets para el análisis de filtros digitales y sus momentos de desvanecimiento.

Patente:
- Particle swarm optimization based orthogonal wavelet blind equalization method, publicada en Google Patents. [6]

Relación encontrada: En esta patente se expone un método basado en wavelets ortogonales para reconstruir una señal usando un algoritmo de partículas para reconstruir una señal más rápido y con menor margen de error.

Diferencia: En el presente proyecto, se usa el algoritmo de partículas para experimentar con los momentos de desvanecimiento.

Software:
-LIFTPACK: A Software Package for Wavelet Transforms using Lifting, G. Fernández, S. Periaswamy, W. Sweldens. [7]

Relación encontrada: El software implementa el esquema Lifting. El esquema Lifting consta de tres pasos. 1.-Separacion de muestras, 2.-Predicción y 3.- Actualización. El tercer paso garantiza la construcción de funciones wavelet biortogonales con momentos de desvanecimiento.

Diferencia: En el presente proyecto se utilizan wavelets ortogonales en lugar de biortogonales.

Justificación

No existe una única parametrización para todas las funciones wavelet, y una técnica consiste en parametrizar las respuestas al impulso de filtros de reconstrucción perfecta que las generan. Al proponer una parametrización es de interés calcular los parámetros que generen funciones wavelet con propiedades de ortogonalidad y suavidad al maximizar los momentos de desvanecimiento, ver [7]. El cálculo de los parámetros no es tarea sencilla toda vez que se tienen múltiples soluciones, y en muchos casos no es posible aplicar métodos de optimización por gradiente.

Considerando lo anterior, se trabajó con parametrizaciones de filtros de reconstrucción perfecta que generan wavelets que maximizan sus momentos de desvanecimiento, y cuyas aplicaciones están presentes en múltiples tesis, artículos, y programas de cómputo.

Objetivos

Objetivo General: Aplicar un algoritmo evolutivo para optimizar los parámetros de filtros digitales de reconstrucción perfecta que generen funciones wavelet sujetas a condiciones de momentos de desvanecimiento.

Objetivos Particulares: Aplicar las formulaciones matemáticas de las condiciones de momentos de desvanecimiento en filtros de reconstrucción perfecta de longitud doce.

-Calcular parámetros de filtros de reconstrucción perfecta de longitud doce, que cumplan con el mayor número de condiciones de momentos de desvanecimiento.

-Comparar la similitud entre parámetros y el error de aproximación entre las funciones wavelet generadas con los “filtros Daubechies” y con los filtros aproximados con un algoritmo de optimización evolutiva.

Marco teórico

El análisis con wavelets es un área de estudio en el área de las matemáticas aplicadas. Los wavelets son funciones que fueron diseñadas para tratar con datos no estacionarios. Y gracias a su generalidad y a sus buenos resultados, se han vuelto muy útiles en un gran número de disciplinas.

Uno de los usos más comunes de los wavelets es en el campo del procesamiento de imágenes y señales, y también en la animación por computadora. Aunque su uso se puede extender más allá de estas áreas. Actualmente son usados por el FBI para codificar su base de datos que contiene millones de huellas digitales. En el futuro, los científicos podrían aplicar wavelets en el procesamiento y codificación de imágenes concernientes al área de salud, como la detección de un tumor cancerígeno o incluso en el estudio del clima. [8]

Uno de los resultados más importantes de la teoría de wavelets es que la transformada wavelet discreta se puede implementar usando un par de filtros digitales, cuyos coeficientes deben cumplir las siguientes condiciones de reconstrucción perfecta, que el caso particular de los filtros digitales de longitud 12 (con coeficientes hi donde i=1, 2,…,12) son:

$$h_0^2 + h_1^2 + h_2^2 + h_3^2 + h_4^2 + h_5^2 + h_6^2 + h_7^2 + h_8^2 + h_9^2 + h_{10}^2 + h_{11}^2 = \frac{1}{2} \quad (1)$$

$$h_0h_2 + h_1h_3 + h_2h_4 + h_3h_5 + h_4h_6 + h_5h_7 + h_6h_8 + h_7h_9 + h_8h_{10} + h_9h_{11} = 0 \quad (2)$$

$$h_0h_4 + h_1h_5 + h_2h_6 + h_3h_7 + h_4h_8 + h_5h_9 + h_6h_{10} + h_7h_{11} = 0 \quad (3)$$

$$h_0h_6 + h_1h_7 + h_2h_8 + h_3h_9 + h_4h_{10} + h_5h_{11} = 0 \quad (4)$$

$$h_0h_8 + h_1h_9 + h_2h_{10} + h_3h_{11} = 0 \quad (5)$$

$$h_0h_{10} + h_1h_{11} = 0 \quad (6)$$

Como ya se mencionó, es relevante que las funciones wavelet generadas a partir de los filtros de reconstrucción perfecta tengan el mayor número posible de momentos de desvanecimiento [6]. Matemáticamente los momentos de desvanecimiento se definen así:

$$\int_{-\infty}^{+\infty} t^p \, f(t) \, dt \quad (7)$$

La ecuación (7) es llamada el p-ésimo momento de desvanecimiento de la función $f(t)$. Está integral marca las condiciones de los momentos de desvanecimiento. Para el caso discreto se puede verificar que, en general, no existen expresiones analíticas para las funciones wavelet obtenidas desde filtros digitales ortogonales, y que son usadas en el

presente proyecto. Las ecuaciones que determinan los coeficientes de los filtros ortogonales para el caso de longitud 12, con coeficientes hi con i=1,2,…,12, son:

$$p = \frac{1}{4}\left(\cos\beta - \cos(2\alpha+\gamma) + \sin\beta + \sin(2\alpha+\gamma)\right) \tag{8}$$

$$q = \frac{1}{4}\left(\cos\beta + \cos(2\alpha+\gamma) + \sin\beta - \sin(2\alpha+\gamma)\right) \tag{9}$$

$$h_0 = \frac{p}{2}\left(\cos\beta + \cos\gamma\right) \tag{10}$$

$$h_1 = \frac{p}{2}\left(\sin\beta + \sin\gamma\right) \tag{11}$$

$$h_2 = \frac{q}{2}\left(\cos\beta + \cos\gamma\right) \tag{12}$$

$$h_3 = \frac{q}{2}\left(\sin\beta + \sin\gamma\right) \tag{13}$$

$$h_4 = \frac{1}{4}(1 - \cos 2\alpha + \sin 2\alpha) - \frac{p}{2}(\cos\beta + \cos\gamma) - \frac{q}{2}(\cos\beta - \cos\gamma) \tag{14}$$

$$h_5 = \frac{1}{4}(1 + \cos 2\alpha + \sin 2\alpha) - \frac{p}{2}(\sin\beta + \sin\gamma) - \frac{q}{2}(\sin\beta - \sin\gamma) \tag{15}$$

$$h_6 = \frac{1}{4}(1 + \cos 2\alpha - \sin 2\alpha) - \frac{p}{2}(\cos\beta - \cos\gamma) - \frac{q}{2}(\cos\beta + \cos\gamma) \tag{16}$$

$$h_7 = \frac{1}{4}(1 - \cos 2\alpha - \sin 2\alpha) - \frac{p}{2}(\sin\beta - \sin\gamma) - \frac{q}{2}(\sin\beta + \sin\gamma) \tag{17}$$

$$h_8 = \frac{q}{2}\left(\cos\beta - \cos\gamma\right) \tag{18}$$

$$h_9 = \frac{q}{2}\left(\sin\beta - \sin\gamma\right) \tag{19}$$

$$h_{10} = \frac{p}{2}\left(\cos\beta - \cos\gamma\right) \tag{20}$$

$$h_{11} = \frac{p}{2}\left(\sin\beta - \sin\gamma\right) \tag{21}$$

Como se puede apreciar, los filtros de longitud 12 de las ecuaciones X a Y involucran los parámetros p, q, alfa, beta y gamma, y donde $\alpha, \beta,$ y γ varían en el intervalo [0, 2PI).

Para el caso discreto, dado un filtro de reconstrucción perfecta en la forma de polinomio trigonométrico H(z) = h0 + h1z+ h2 z^2 +…+ h11z^11, las condiciones de momentos de desvanecimiento que corresponden a la ecuación (1) se obtienen derivando H(z) tantas veces como momentos de desvanecimiento se requieran. A continuación, en las ecuaciones W a T, se muestran cinco derivadas del polinomio trigonométrico:

Primera Derivada:

$$H'(z) = 0 + h_1 + 2h_2z + 3h_3z^2 + 4h_4z^3 + 5h_5z^4 + 6h_6z^5 + 7h_7z^6 + 8h_8z^7 + 9h_9z^8 + 10h_{10}z^9 + 11h_{11}z^{10} \quad (22)$$

Segunda Derivada:

$$H''(z) = 0 + 0 + 2h_2 + 6h_3z + 12h_4z^2 + 20h_5z^3 + 30h_6z^4 + 42h_7z^5 + 56h_8z^6 + 72h_9z^7 + 90h_{10}z^8 + 110h_{11}z^9 \quad (23)$$

Tercera Derivada:

$$H'''(z) = 0 + 0 + 0 + 6h_3 + 24h_4z + 60h_5z^2 + 120h_6z^3 + 210h_7z^4 + 336h_8z^5 + 504h_9z^6 + 720h_{10}z^7 + 990h_{11}z^8 \quad (24)$$

Cuarta Derivada:

$$H''''(z) = 0 + 0 + 0 + 0 + 24h_4 + 120h_5z + 360h_6z^2 + 840h_7z^3 + 1680h_8z^4 + 3024h_9z^5 + 5040h_{10}z^6 + 7920h_{11}z^7 \quad (25)$$

Quinta Derivada:

$$H'''''(z) = 0 + 0 + 0 + 0 + 0 + 120h_5 + 720h_6z + 2520h_7z^2 + 6720h_8z^3 + 15120h_9z^4 + 30240h_{10}z^5 + 55440h_{11}z^6 \quad (26)$$

La idea principal del proyecto propuesto fué identificar el conjunto de parámetros p, q, $\alpha, \beta,$ y γ involucrados en cada coeficiente hi (a, b) tales que hacen que cada una de esas derivadas sea igual a cero en la medida de lo posible.

Para ello se usó el algoritmo de cúmulo de partículas (PSO, por sus siglas en inglés) que usa una función de evaluación que incremente el número de condiciones satisfechas (ecuaciones 22 a 26), con lo cual se logra identificar los valores de los coeficientes

(ecuaciones 8 a 21) que mejor empaten con los coeficientes de los filtros publicados por Ingrid Daubechies. También se deben cumplir las ecuaciones 1 a 7.

La técnica de optimización empleada se llama "penalización" que en el caso de minimización del error de aproximación se inicializa con un valor relativamente grande, y por cada condición satisfecha se reduce en una parte proporcional hasta que se cumplan todas las restricciones y se obtenga un problema sin restricciones.

Momentos de desvanecimiento: El número de momentos de desvanecimiento determina lo que la wavelet "no puede ver". Una wavelet con un momento de desvanecimiento no ve funciones lineales y el coeficiente de la wavelet es cero. Dos momentos de desvanecimiento hacen que la wavelet "no vea" funciones cuadráticas y así sucesivamente. Con una wavelet con muchos momentos de desvanecimiento se obtienen coeficientes con valores pequeños cuando se analizan bajas frecuencias; los coeficientes miden la semejanza entre la wavelet y la señal a analizar, y una wavelet que oscila rápidamente no se asemeja a ondas de baja frecuencia.

Una wavelet tiene m momentos de desvanecimiento si y sólo si puede generar polinomios de grado menor o igual a m. Mientras esta propiedad es utilizada para describir el poder de aproximación de las funciones, en el caso de las wavelets se usa también en análisis funcional para caracterizar singularidades (discontinuidades, puntos de inflexión, etc).

El número de momentos de desvanecimiento está determinado por los coeficientes $\mathrm{h}(n)$ del filtro h que se encuentra caracterizado en la función de escala. El efecto práctico de los momentos de desvanecimiento es concentrar la información de la señal en un número relativamente pequeño de coeficientes de transformación wavelet. La Tabla 1 muestra en su primera columna la longitud de filtros de reconstrucción perfecta ortogonales con un número par de coeficientes, la segunda columna muestra el número máximo de veces que se derivado el polinomio trigonométrico H(z), y la tercera columna indica el número máximo de momentos de desvanecimiento que el wavelet (o filtro) puede alcanzar.

n	D'	M	D
2	0	1	Haar
4	1	2	D4
6	2	3	D6
8	3	4	D8

10	4	5	D10
12	5	6	D12

Tabla 1. Relación entre tamaño del filtro, derivadas y momentos de desvanecimiento.

El siguiente diagrama de bloques ilustra cada uno de los bloques que se implementaron para cumplir con los objetivos planteados en esta propuesta.

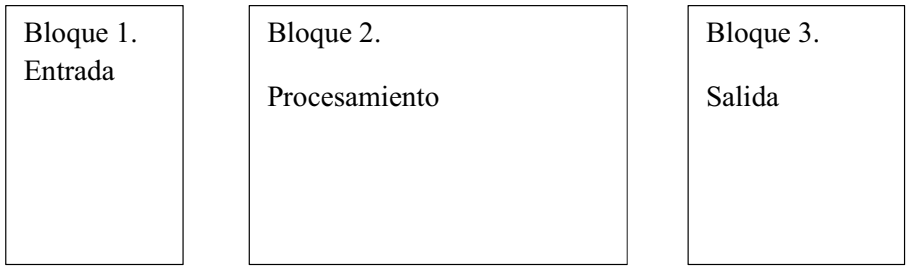

Figura 1. Diagrama de bloques de nivel 0.

El Bloque 1 representó la entrada que es un polinomio trigonométrico $H(z) = a_0 + b_0 z + a_1 z^2 + b_1 z^3 + a_2 z^4 + b_2 z^5 + a_3 z^6 + b_3 z^7 + a_4 z^8 + b_4 z^9 + a_5 z^{10} + b_5 z^{11}$ de longitud 12.

El Bloque 2 realizó el procesamiento para calcular el número máximo de momentos de desvanecimiento que se pueden alcanzar con las parametrizaciones de los coeficientes del filtro H(z).

El Bloque 3 comparó los resultados del Bloque 2 con las funciones wavelet fijas, con un máximo número de momentos de desvanecimiento.

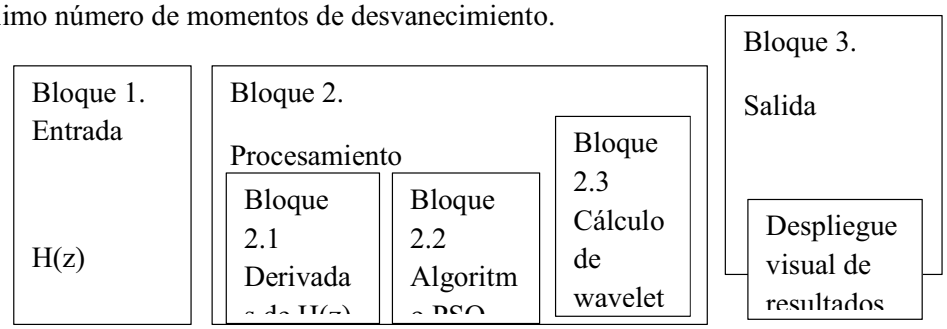

Figura 2. Diagrama de bloques de nivel 0.

En el Bloque 2.1 se calculó la derivada de los polinomios trigonométricos H(z), y se realizó manualmente.

En el Bloque 2.2 con las derivadas de orden n, $H^n(z)$, se aplicó un algoritmo de optimización por cúmulo de partículas que calculó los parámetros que maximizan el mayor número de momentos de desvanecimiento, con cierta precisión y prioridad. La prioridad se da porque debe primero cumplirse el primer momento de desvanecimiento antes del segundo, y antes del tercero y así sucesivamente. La salida es el conjunto de parámetros óptimos para las condiciones de momentos de desvanecimiento dadas.

Parte del trabajo que se desarrolló consistió en diseñar un algoritmo que tenga en cuenta estas consideraciones. Es importante mencionar que no existe garantía alguna de que las parametrizaciones propuestas puedan dar lugar a soluciones válidas para las condiciones de momentos de desvanecimiento, y que en su caso se calculó la mejor aproximación.

En el bloque 2.3 se aproximó la función wavelet con los coeficientes codificados como valores de doble precisión, usando el algoritmo de cascada, misma que debe compararse con las funciones wavelet obtenidas con los coeficientes de la Tabla 2, que maximizaron los momentos de desvanecimiento sin hacer uso de las ecuaciones paramétricas X aY.

En el Bloque 3 se realizaron comparaciones entre las aproximaciones de los wavelets con el mayor número posible de momentos de desvanecimiento y filtros fijos de longitud 4, 6, 8, 10 y 12. Incluye un bloque de desplegado visual de las aproximaciones.

Se trabajaró con los filtros de tamaño 12 con experimentos con orden ascendente de momentos de desvanecimiento, es decir, al principio se trabajaró con un momento de desvanecimiento, posteriormente con dos momentos de desvanecimiento y así sucesivamente) hasta que se realizaron experimentos con los 6 momentos de desvanecimiento que tienen los filtros de tamaño 12, cumpliendo cada vez con las restricciones de los momentos de desvanecimiento anteriores conforme sea posible cumplirlos.

Cabe mencionar que son conocidos los valores de filtros que cumplen con los momentos de desvanecimiento para el caso un momento hasta seis momentos (Filtros Daubechies). Y en este trabajo se determinaron los valores de los parámetros que aproximan los filtros correspondientes mostrados en las columnas de la tabla 1.

Se diseñaró el algoritmo que priorice el cumplimiento de las condiciones de momento de desvanecimiento en orden ascendente conforme sea posible y bajo un criterio de error.

D2 (Haar)	**D4**	**D6**	**D8**	**D10**	**D12**
1	0.6830127	0.47046721	0.32580343	0.22641898	0.15774243
1	1.1830127	1.14111692	1.01094572	0.85394354	0.69950381
	0.3169873	0.650365	0.8922014	1.02432694	1.06226376

-0.1830127	-0.19093442	-0.03957503	0.19576696	0.44583132
	-0.12083221	-0.26450717	-0.34265671	-0.31998660
	0.0498175	0.0436163	-0.04560113	-0.18351806
		0.0465036	0.10970265	0.13788809
		-0.01498699	-0.00882680	0.03892321
			-0.01779187	-0.04466375
			4.71742793e-3	7.83251152e-4
				6.75606236e-3
				-1.52353381e-3

Tabla 2. Coeficientes de filtros de reconstrucción perfecta para filtros de longitud 4 a 12.

Desarrollo del proyecto

Para el desarrollo de este proyecto se aplicó un algoritmo de cúmulo de partículas para encontrar valores aproximados lo más posible a los filtros daubechies que cumplieran las restricciones tanto inherentes a los wavelets de tamaño 12 y a las restricciones inherentes a cada momento de desvanecimiento. Después de identificar las ecuaciones pertenecientes a los momentos de desvanecimiento 2 a 6 se utilizó el algoritmo de enjambre de partículas en cada ecuación de diferente número de momentos de desvanecimiento (i.e. se inició por la ecuación de dos momentos de desvanecimiento (ecuación 22), después se utilizó la ecuación de 3 momentos de desvanecimiento (ecuación 23) y así sucesivamente.) asegurando que se cumplieran las restricciones y ecuaciones anteriores. De esta forma se obtienen los valores de los ángulos mejor aproximados en la medida de lo posible y con el menor margen de error posible para cada ecuación.

Resultados

En la siguiente sección se muestran los valores de los parámetros y los valores de los filtros obtenidos de éstos que mejor aproximan los valores de los filtros obtenidos por la matemática Ingrid Daubechies, utilizando la ecuación de momentos de desvanecimiento acorde a los momentos de desvanecimiento requeridos para el experimento y calculando un margen de error utilizando los filtros obtenidos en el experimento con los valores de los filtros daubechies. Los resultados se mostrarán por etapas.

Primer Experimento: Tres momentos de desvanecimiento (ecuación 23) comparada con los filtros Daubechies 6 (D6).

En este experimento se tomaron los valores del Artículo [10] para comparar los valores obtenidos en este experimento y los valores de los filtros daubechies.

Procedencia	Valor de α	Valor de β	Valor de γ
Valores Obtenidos	1.021731262	2.111773531	2.236504713
Artículo	1.3	1.979327721	1.958037288
Ingrid Daubechies	No se conocen en forma precisa	No se conocen en forma precisa	No se conocen en forma precisa

Tabla 3: Valores de los parámetros conocidos en el primer experimento con 3 momentos de desvanecimiento.

Filtros Obtenidos	Filtros del Artículo	Filtros Daubechies D6
0.020761898	0.030407801	0.235233605
-0.030130955	-0.072347202	0.57055846
-0.117664048	-0.13123029	0.3251825
0.170761364	0.312227255	-0.09546721
0.554979491	0.566013582	-0.060416105
0.381557204	0.238397384	0.02490875
0.033140771	0.037362578	0
-0.028238821	0.022795881	0
0.010663462	-0.003323852	0
0.007347717	-0.001397028	0
-0.001881575	7.70E-04	0
-0.001296509	3.24E-04	0

Tabla 4: Valores de los Filtros en el Primer Experimento con 3 Momentos de Desvanecimiento.

Error Artículo contra Experimento	Error Daubechies contra Experimento
0.009645903	0.020761898

-0.042216247	-0.030130955
-0.013566242	-0.117664048
0.141465891	-0.064472241
0.011034091	-0.015578969
-0.14315982	0.056374704
0.004221807	0.128607981
0.051034702	0.032177284
-0.013987314	-0.014245288
-0.008744744	0.007347717
0.002651756	-0.001881575
0.001620219	-0.001296509

Tabla 5: Errores de los filtros obtenidos en el primer experimento con 3 Momentos de Desvanecimiento.

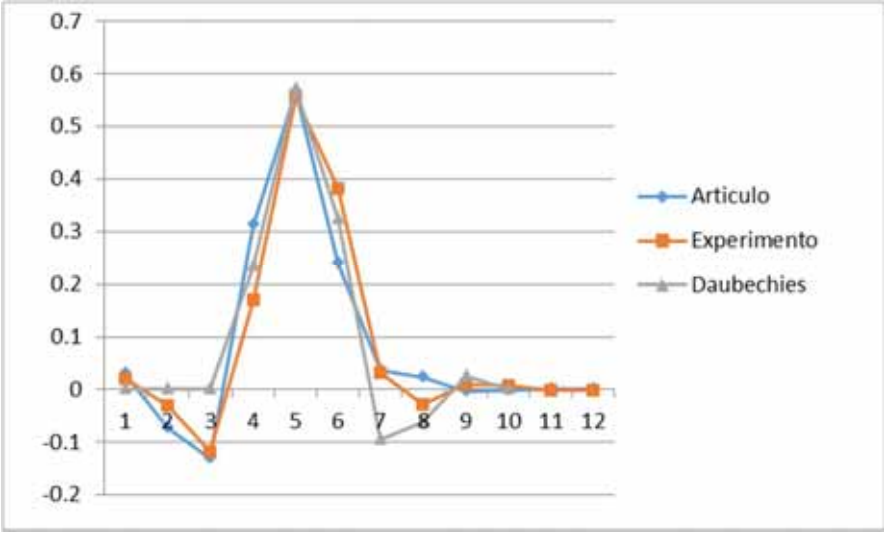

Figura 3: Gráfica de los filtros daubechies y los obtenidos en el primer experimento y el artículo con 3 Momentos de Desvanecimiento.

En los experimentos se usó el error cuadrático medio para calcular el valor de error, y en lo sucesivo se hará referencia a este valor como “error”.

La suma del error entre los valores obtenidos en el primer experimento y los valores obtenidos en el Artículo es: 0.061639083

La suma del error entre los valores obtenidos en el primer experimento y los valores obtenidos por Daubechies (D6) es: 0.058165546

La suma del error entre el valor de los parámetros obtenidos en el primer experimento y el obtenido en el Artículo [10] es: 0.347596632

No se calcula el valor de la suma del error entre los ángulos obtenidos en el primer experimento y los ángulos utilizados por Ingrid Daubechies ya que el valor de estos es desconocido.

Segundo Experimento: Cuatro momentos de desvanecimiento (Ecuación 24) comparada con los filtros Daubechies 8 (D8).

En este experimento solo se usaron los valores calculados en este experimento y los obtenidos por Ingrid Daubechies.

Procedencia	Valor de α	Valor de β	Valor de γ
Experimento	-3.14159265358979	-1.07355157466211	-2.82097915309342
Daubechies D8	No se conoce en forma precisa	No se conoce en forma precisa	No se conoce en forma precisa

Tabla 6: Valores de los ángulos obtenidos en el segundo experimento con 4 Momentos de Desvanecimiento.

Filtros Obtenidos	Filtros Daubechies 8 (D8)
-0.013688998800884717	0.32580343
-0.03462727151785501	1.01094572
0.06111620422227803	0.8922014
0.15459767573423433	-0.03957503
0.19832439036359875	-0.26450717

0.46163643097762896	0.0436163
0.39752858260855134	0.0465036
-0.1382489587066395	-0.01498699
-0.18463539156271397	0
0.07299084054022621	0
0.04135521316917059	0
-0.0163487170275949	0

Tabla 7: Valores de los Filtros Obtenidos en el segundo Experimento con 4 Momentos de Desvanecimiento.

Error Experimento contra Daubechies
-0.176590714
-0.540100132
-0.384984496
0.174385191
0.330577975
0.439828281
0.374276783
-0.130755464
-0.184635392
0.072990841
0.041355213
-0.016348717

Tabla 8: Errores obtenidos en el segundo experimento con 4 Momentos de Desvanecimiento.

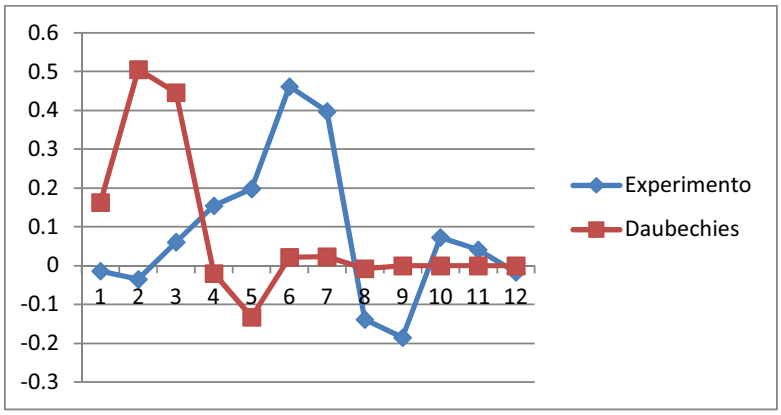

Figura 4: Gráfica de los filtros Daubechies 8 y los filtros obtenidos en el segundo experimento con 4 Momentos de Desvanecimiento.

La suma del error entre los valores obtenidos en el segundo experimento y los valores obtenidos por Daubechies (D6): 0.289082158

No se calcula el valor de la suma del error entre los parámetros w obtenidos en el segundo experimento y los parámetros que corresponden a los filtros daubechies ya que el valor de estos es desconocido.

Tercer Experimento: Cinco momentos de desvanecimiento (Ecuación 25) comparada con los filtros Daubechies 10 (D10).

En este experimento solo se usaron los valores calculados en este experimento y los obtenidos por Ingrid Daubechies.

Procedencia	Valor de α	Valor de β	Valor de γ
Experimento	-3.10747228585442	2.6996445067579335	1.2024788675648184
Daubechies	No se conoce en forma precisa	No se conoce en forma precisa	No se conoce en forma precisa

Tabla 9: Valores de los ángulos del tercer experimento con 5 Momentos de Desvanecimiento.

Filtros Obtenidos	Filtros Daubechies 10 (D10)
-0.01247520308003674	0.22641898
0.031209819656297894	0.85394354

0.07722596341487348	1.02432694
-0.19319993233769164	0.19576696
-0.1493696022763253	-0.34265671
0.41351597464102485	-0.04560113
0.4341376672247665	0.10970265
0.18832373553925175	-0.00882680
0.17947362522395724	-0.01779187
0.07173927779257336	4.71742793e-3
-0.02899245050723514	0
-0.011588875291456132	0

Tabla 10: Valores de los filtros obtenidos en el tercer experimento con 5 Momentos de Desvanecimiento.

Error experimento contra Daubechies
-0.125684693
-0.39576195
-0.434937507
-0.291083412
0.021958753
0.43631654
0.379286342
0.192737136
0.18836956
0.069380564

-0.028992451
-0.011588875

Tabla 11: Errores obtenidos en el tercer experimento con 5 Momentos de Desvanecimiento.

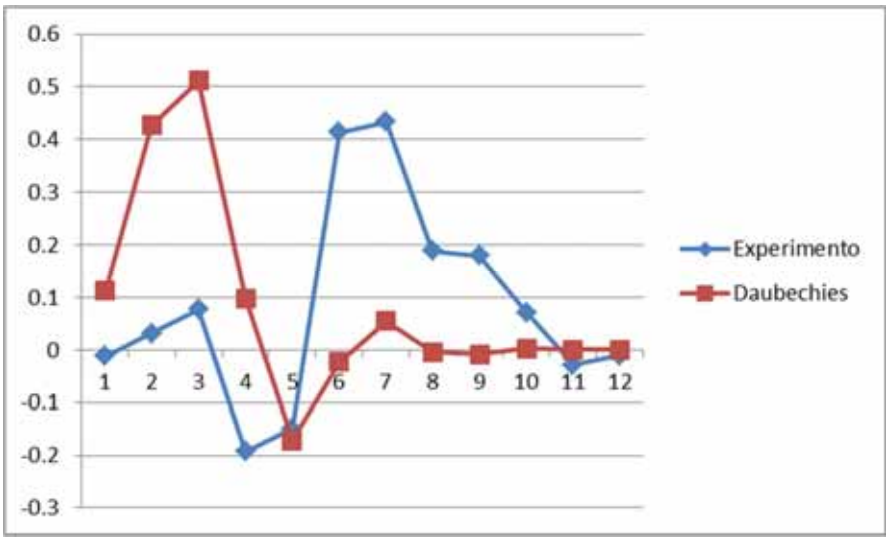

Figura 5: Gráfica de los filtros Daubechies 10 y los filtros obtenidos en el Tercer Experimento con 5 momentos de Desvanecimiento.

La suma del error entre los valores obtenidos en el tercer experimento y los valores obtenidos por Daubechies (D6) es: 0.267621625

No se calcula el valor de la suma del error entre los ángulos obtenidos en el tercer experimento y los parámetros que corresponden a los filtros daubechies ya que el valor de estos es desconocido.

Cuarto experimento: Seis momentos de desvanecimiento (Ecuación 26) comparada con los filtros Daubechies 12 (D12).

No existen parámetros que cumplan las ecuaciones de momentos de desvanecimiento y se aproximen a los valores calculados por Ingrid Daubechies.

Análisis y discusión de resultados

Se observa que cuando se incrementa el número de condiciones de momentos de desvanecimiento es más difícil realizar las aproximaciones.

En la Figura 3 se observa que para 3 momentos de desvanecimiento la aproximación es satisfactoria. Los valores de los parámetros obtenidos permiten calcular los valores de los coeficientes de los filtros, con un error de 0.06 comparado con los filtros publicados en el artículo [10] y un error de 0.05 comparado con los filtros daubechies.

Cuando se incrementa una condición más para tener 4 momentos de desvanecimiento el error se incrementa a 0.28 al compararlo con los filtros daubechies. Esto refleja que es un problema más complicado. La comparación entre los filtros daubechies y los obtenidos en este trabajo se visualizan en la Figura 4. Nótese que siguen la misma forma, con un desplazamiento entre ellas.

Cuando se incrementa otra condición más para tener 5 momentos de desvanecimiento el error es 0.26, pero teniendo en cuenta que el criterio para cumplir las condiciones de momento de desvanecimiento (ecuaciones 23 a 25) no es exigiendo que sea exactamente igual a cero, sino que la condición de un valor absoluto menor a 0.1, y para los casos de 3 y 4 momentos se requirió un valor de 0.01. Esto se hace porque se trabajó con aproximaciones. La comparación entre los filtros daubechies y los obtenidos en este trabajo se visualizan en la Figura 5 en donde se aprecia que siguen la misma forma.

Finalmente cuando se incluyó la condición de 6 momentos de desvanecimiento aun cuando se probó con valores relativamente grandes (menor a 1.0) lo cual indica que no es posible obtener una solución para este caso, y es precisamente uno de los resultados experimentales de interés toda vez que no se conoce si existen parámetros precisos que cumplan con estas condiciones.

Conclusiones

Se puede apreciar que mientras más momentos de desvanecimiento se tiene y más restricciones se deben cumplir hay una mayor dificultad para encontrar valores que satisfagan la ecuación. Se comprueba que no hay solución para una ecuación que tiene 6 momentos de desvanecimiento o que tiene mucho margen de error, lo cual significa que no existen valores que satisfagan esta ecuación y sus restricciones.

Se aplicó un algoritmo de optimización con manejo de restricciones para optimizar los parámetros de filtros digitales de reconstrucción perfecta que generan funciones wavelet sujetas a condiciones de momentos de desvanecimiento.

Conforme se incrementaron las condiciones de momentos de desvanecimiento se observó experimentalmente que es cada vez más difícil obtener soluciones, a pesar de que se amplía el margen de error para considerar que una condición se cumple. Esta técnica se aplica

cuando se trabaja con aproximaciones, porque es difícil obtener valores para los cuales la condiciones de momentos de desvanecimiento sean exactamente cero.

Se aplicaron las fórmulas matemáticas de las condiciones de momentos de desvanecimiento en filtros de reconstrucción perfecta de longitud doce, y fue posible compararlas con filtros ya publicados. Se observó que al incluir más condiciones de momentos de desvanecimiento las aproximaciones difieren cada vez más. Y para el caso de 6 momentos de desvanecimiento se observó experimentalmente que no fue posible obtener una solución a ese problema.

Se calcularon parámetros optimizados de filtros de reconstrucción perfecta de longitud doce, que hicieran cumplir con el mayor número de condiciones de momentos de desvanecimiento. Como ya se indicó, conforme se ponen más restricciones es más difícil hacer las aproximaciones.

Se compararon los valores de los parámetros en el caso de 3 momentos de desvanecimiento, con los valores publicados en [10], y se obtuvo un error de 0.34, que para tener una idea de que tan grande es este error considérese que la suma de los filtros es 1, entonces se está hablando de una estimación del 34%, por dar un criterio de comparación.

NOTA.

Existen varios algoritmos evolutivos particle swarm optimization entre los cuales se pueden mencionar algoritmos genéticos, estrategias evolutivas, evolución diferencial y programación genética.

Posibles trabajos futuros podrían enfocarse en comparar los desempeños de estos algoritmos sobre el problema de aproximación de parámetros de filtros de reconstrucción perfecta.

En este trabajo, los objetivos hacen referencia al uso de un algoritmo evolutivo, sin embargo en la propuesta sometida y aprobada también se indica el uso del método PSO (de las siglas en inglés de Particle Swarm Optimization) que hace referencia a los enjambres de abejas (bioinspirado), entonces para ser consistentes con la propuesta aprobada se usó PSO.

Referencias bibliográficas

[1] T. D. Santiago Santiago y A. de J. Hernández Trejo, " Sistema de identificación de personas a través del iris mediante análisis de texturas por filtrado de wavelets", proyecto

terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2010.
[2] J. A. Garduño Martínez, " Estudio experimental de la aproximación de filtros digitales paramétricos para implementar la transformada wavelet discreta con polinomios evolutivos", proyecto terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2011.
[3] M. S. Hernández Nieves, " Clasificación de llantos de bebé con wavelets", proyecto terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2012.

[4] K. E. Wirsing, "Application of Wavelets to Filtering and Analysis of Self-Similar Signals," M.S. thesis, Virginia Polytechnic Institute and State University, Falls Church, Virginia, 2014
[5] O. Herrera y R. Mora, "Aplicación de Algoritmos Genéticos a la Compresión de Imágenes con Evolets", *Avances Recientes en Sistemas Inteligentes*, O. Herrera, M.Gonzalez, Ed. Sociedad Mexicana de Inteligencia Artificial, 2011, pp. 157-166.

[6] Google Patents, "Particle swarm optimization based orthogonal wavelet blind equalization method." CN. Patent CN102123115 B, 16 Abr 2014.

[7] G. Fernández, S. Periaswamy, W. Sweldens, "LIFTPACK: A Software Package for Wavelet Transforms using Lifting", 1996. Disponible: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1021578.

[8] I. Daubechies, Ten lectures on wavelets, SIAM, CBMS Lecture Series, 1992.

[9] Wavelet-Based Image Compression, "Image Compression Theory", 2000. [En Línea]. Disponible: https://www.clear.rice.edu/elec301/Projects00/wavelet_image_comp/img-compression-theory.html

[10] D. W. Roach, "A Subclass of the Length 12 Parametrized Wavelets", Murray State University, Murray, KY 42071, USA, 2012

[11] http://jswarm-pso.sourceforge.net. Consultada el 22 de abril del 2015

Apéndice A

A continuación de muestra el código fuente utilizado para el desarrollo de este proyecto. Se utilizaron dos programas, uno es el programa que ejecuta el algoritmo de PSO, y el otro es un programa que se utilizó para calcular los filtros y comprobar si los parámetros calculados por el algoritmo PSO eran suficientemente satisfactorios para los experimentos llevados a cabo en el presente proyecto. Se utilizó la biblioteca jswarm_pso disponible en [11].

Programa PSO

Clase Example

```
package pso;



import net.sourceforge.jswarm_pso.Neighborhood;

import net.sourceforge.jswarm_pso.Neighborhood1D;

import net.sourceforge.jswarm_pso.Swarm;

import net.sourceforge.jswarm_pso.example_2.SwarmShow2D;


/**

 * Función de oprimización de enjambre simple.

 */

public class Example {


	 //Principal

	 public static void main(String[] args) {

	Swarm swarm;

			// Create a swarm (using 'MyParticle' as sample particle and 'MyFitnessFunction' as fitness function)

	//Se crea un enjambre (usando MyParticle como partícula de muestra y 'MyFitnessFunction'

	// como la función fitness)

			swarm = new Swarm(Swarm.DEFAULT_NUMBER_OF_PARTICLES, new MyParticle(), new MyFitnessFunction());


			// Creación de Vecindario

			Neighborhood neigh = new Neighborhood1D(Swarm.DEFAULT_NUMBER_OF_PARTICLES / 5, true);

			swarm.setNeighborhood(neigh);

			swarm.setNeighborhoodIncrement(0.9);
```

```
                // Fijar ocnstantes de psoición y velocidad. I.e.: Lugar para buscar soluciones.
                swarm.setInertia(1);
        swarm.setParticleIncrement(0.9);
        swarm.setGlobalIncrement(0.9);

        //Fijar Posición del enjambre (Buscar por soluciones.)
        swarm.setMaxPosition(Math.PI);
                swarm.setMinPosition(-1*Math.PI);

        //Número de iteraciones antes de detenerse.
                int numberOfIterations = 1700000;
                boolean showGraphics = false;

                if (showGraphics) {
                        int displayEvery = numberOfIterations / 100 + 1;
                        SwarmShow2D ss2d = new SwarmShow2D(swarm, numberOfIterations, displayEvery, true);
                        ss2d.run();
                } else {
            //Optimizar
                        for (int i = 0; i < numberOfIterations; i++)
            {swarm.evolve();}
             System.out.println(swarm.getBestFitness());

                }
                // Imprimir Resultados.
                System.out.println(swarm.toStringStats());
         }
}
```

Clase MyFitnessFunction

```
package pso;

import net.sourceforge.jswarm_pso.FitnessFunction;

public class MyFitnessFunction extends FitnessFunction {

   public MyFitnessFunction() {
      super(false); // Minimizar la función.
   }

   /**
    * Evalua las par´ticulas en una función.
    *
    * @param position : Posición de la partícula.
    * @return Función fitness para una partícula.
    */

   @Override
   public double evaluate(double position[]) {
      //Declaración de variables y sustracción de los ángulos.
      long fitness = 40000;
      double sumaError = 0;
      double alfa = position[0];
      double beta = position[1];
      double gamma = position[2];

      //Cálculo de los filtros para contunuar la evaluación.
      double a[] = new double[6];
      double b[] = new double[6];
      double p = (Math.cos(beta) - Math.cos(2 * alfa + gamma) + Math.sin(beta) + Math.sin(2 * alfa + gamma)) / 4;
```

```
double q = (Math.cos(beta) + Math.cos(2 * alfa + gamma) + Math.sin(beta) - Math.sin(2 * alfa + gamma)) / 4;

a[0] = (p * (Math.cos(beta) + Math.cos(gamma))) / 2;

b[0] = (p * (Math.sin(beta) + Math.sin(gamma))) / 2;

a[1] = (q * (Math.cos(beta) + Math.cos(gamma))) / 2;

b[1] = (q * (Math.sin(beta) + Math.sin(gamma))) / 2;

a[2] = ((1 - Math.cos(2 * alfa) + Math.sin(2 * alfa)) / 4) - ((p * (Math.cos(beta) + Math.cos(gamma))) / 2) - ((q * (Math.cos(beta) - Math.cos(gamma))) / 2);

b[2] = ((1 + Math.cos(2 * alfa) + Math.sin(2 * alfa)) / 4) - ((p * (Math.sin(beta) + Math.sin(gamma))) / 2) - ((q * (Math.sin(beta) - Math.sin(gamma))) / 2);

a[3] = ((1 + Math.cos(2 * alfa) - Math.sin(2 * alfa)) / 4) - ((p * (Math.cos(beta) - Math.cos(gamma))) / 2) - ((q * (Math.cos(beta) + Math.cos(gamma))) / 2);

b[3] = ((1 - Math.cos(2 * alfa) - Math.sin(2 * alfa)) / 4) - ((p * (Math.sin(beta) - Math.sin(gamma))) / 2) - ((q * (Math.sin(beta) + Math.sin(gamma))) / 2); //Corregida

a[4] = (q * (Math.cos(beta) - Math.cos(gamma))) / 2;

b[4] = (q * (Math.sin(beta) - Math.sin(gamma))) / 2;

a[5] = (p * (Math.cos(beta) - Math.cos(gamma))) / 2;

b[5] = (p * (Math.sin(beta) - Math.sin(gamma))) / 2;



//Valores daubechies 12

/*double valoresDaubechies12 [] = new double[12];


valoresDaubechies12[0] = 0.15774243 / 2;

valoresDaubechies12[1] = 0.69950381 / 2;

valoresDaubechies12[2] = 1.06226376 / 2;

valoresDaubechies12[3] = 0.44583132 / 2;

valoresDaubechies12[4] = -0.31998660 /2;

valoresDaubechies12[5] = -0.18351806 /2;

valoresDaubechies12[6] = 0.13788809 /2;

valoresDaubechies12[7] = 0.03892321 /2;

valoresDaubechies12[8] = -0.04466375 /2;
```

```
valoresDaubechies12[9] = 7.83251152e-4 / 2;

valoresDaubechies12[10] = 6.75606236e-3 /2;

valoresDaubechies12[11] = -1.52353381e-3 /2;*/


//Valores Daubechies 6

/*double valoresDaubechies12 [] = new double[12];

valoresDaubechies12[0] = 0.47046721 / 2;

valoresDaubechies12[1] = 1.14111692 / 2;

valoresDaubechies12[2] = 0.650365/ 2;

valoresDaubechies12[3] = -0.19093442 / 2;

valoresDaubechies12[4] = -0.12083221 /2;

valoresDaubechies12[5] = 0.0498175 /2;

valoresDaubechies12[6] = 0;

valoresDaubechies12[7] = 0;

valoresDaubechies12[8] = 0;

valoresDaubechies12[9] = 0;

valoresDaubechies12[10] = 0;

valoresDaubechies12[11] = 0;*/


//Valores Daubechies 8

/*double valoresDaubechies12 [] = new double[12];

valoresDaubechies12[0] = 0.32580343 / 2;

valoresDaubechies12[1] = 1.01094572/ 2;

valoresDaubechies12[2] = 0.8922014 / 2;

valoresDaubechies12[3] = -0.03957503 / 2;

valoresDaubechies12[4] = -0.26450717 /2;

valoresDaubechies12[5] = 0.0436163 /2;

valoresDaubechies12[6] = 0.0465036 /2;

valoresDaubechies12[7] = -0.01498699 /2;

valoresDaubechies12[8] = 0;
```

```
valoresDaubechies12[9] = 0;

valoresDaubechies12[10] = 0;

valoresDaubechies12[11] = 0;*/


//Valores Daubechies 10


double valoresDaubechies12 [] = new double[12];

valoresDaubechies12[0] = 0.22641898 / 2;

valoresDaubechies12[1] = 0.85394354/ 2;

valoresDaubechies12[2] = 1.02432694 / 2;

valoresDaubechies12[3] = 0.19576696 / 2;

valoresDaubechies12[4] = -0.34265671 /2;

valoresDaubechies12[5] = -0.04560113 /2;

valoresDaubechies12[6] = 0.10970265 /2;

valoresDaubechies12[7] = -0.00882680 /2;

valoresDaubechies12[8] = -0.01779187 / 2;

valoresDaubechies12[9] = 4.71742793e-3 / 2;

valoresDaubechies12[10] = 0;

valoresDaubechies12[11] = 0;



//Almacenamiento de diferencia de error

double errores[] = new double[12];

errores[0] = a[0] - valoresDaubechies12[0];

errores[1] = b[0] - valoresDaubechies12[1];

errores[2] = a[1] - valoresDaubechies12[2];

errores[3] = b[1] - valoresDaubechies12[3];

errores[4] = a[2] - valoresDaubechies12[4];

errores[5] = b[2] - valoresDaubechies12[5];

errores[6] = a[3] - valoresDaubechies12[6];
```

```
errores[7] = b[3] - valoresDaubechies12[7];

errores[8] = a[4] - valoresDaubechies12[8];

errores[9] = b[4] - valoresDaubechies12[9];

errores[10] = a[5] - valoresDaubechies12[10];

errores[11] = b[5] - valoresDaubechies12[11];


//Cálculo de error cuadrático medio.

for (double errore : errores) {

   sumaError += errore*errore;

}


//Ecuaciónes de momentos de desvanecimiento

//double hz = a[0] + a[1] + a[2] + a[3] + a[4] + a[5] + b[0] + b[1] + b[2] + b[3] + b[4] + b[5] ;

double hk = 0 + b[0] + (-2 * a[1]) + (3 * b[1]) + (-4 * a[2]) + (5 * b[2]) + (-6 * a[3]) + (7 * b[3]) +

      (-8 * a[4]) + (9 * b[4]) + (-10 * a[5]) + (11 * b[5]);//Primera derivada, dos momentos de desvanecimiento

double hz = 0 + 0 + (2 * a[1]) + (-6 * b[1]) + (12 * a[2]) + (-20 * b[2]) + (30 * a[3]) + (-42 * b[3]) +

      (56 * a[4]) + (-72 * b[4]) + (90 * a[5]) + (-110 * b[5]); //Segunda derivada, tres momentos de desvanecimiento

double hx = 0 + 0 + 0 + (6 * b[1]) + (-24 * a[2]) + (60 * b[2]) + (-120 * a[3]) + (210 * b[3]) +

      (-336 * a[4]) + (504 * b[4]) + (-720 * a[5]) + (990 * b[5]); //Tercera derivada, cuatro momentos de desvanecimiento

double ha = 0 + 0 + 0 + 0 + (24 * a[2]) + (-60 * b[2]) + (360 * a[3]) + (-840 * b[3]) +

      (1680 * a[4]) + (-3024 * b[4]) + (5040 * a[5]) + (-7920 * b[5]); //Cuarta derivada, cinco momentos de desvanecimiento

/*double hb = 0 + 0 + 0 + 0 + 0 + (60 * b[2]) + (-720 * a[3]) + (2520 * b[3]) +

      (-6720 * a[4]) + (15120 * b[4]) + (-30240 * a[5]) + (55440 * b[5]);*///Quinta derivada, seis momentos de desvanecimiento


//Cálculo y comprobación de valores y de si

//estos cumplen las condiciones para ser considerados

//satisfactorios.

hk = Math.pow(hk, 2);

hk = Math.sqrt(hk);

hz = Math.pow(hz, 2);
```

```
        hz = Math.sqrt(hz);
        hx = Math.pow(hx, 2);
        hx = Math.sqrt(hx);
        ha = Math.pow(ha, 2);
        ha = Math.sqrt(ha);
        //hb = Math.pow(hb, 2);
        //hb = Math.sqrt(hb);
        //return hz + hk + hx + ha + hb;
        if (hk < 0.1) {
            fitness -= 10000;
        }
        if (hz < 0.1) {
            fitness -= 10000;
        }
        if (hx < 0.1) {
            fitness -= 10000;
        }
        if (ha < 0.1) {
            fitness -= 10000;
        }
        /*if (hx < 0.1) {
            fitness -= 10000;
        }*/
        return fitness + Math.sqrt(sumaError);
    }
}
```

Clase My Particle

```
package pso;
```

```
import net.sourceforge.jswarm_pso.Particle;


/**
 * Partícula simple.
 */


public class MyParticle extends Particle {


		/** Constructor */
		public MyParticle() {
			super(3); // Se crea una partícula de dos dimensiones.
		}
}
```

Programa Comprobador

Clase Comprobador

```
package comprobador;


import java.util.Scanner;
//Clase Comprobador.
public class Comprobador {


   //Declaración de Variables.
   public static Scanner sc = new Scanner(System.in);
   public static double alfa,beta,gamma;
   public static double a[] = new double[6];
   public static double b[] = new double[6];
   public static double valores[] = new double[12];
   public static double p;
```

```
    public static double q;

    public static double sumaError= 0;

    public static double sumaDaubechies = 0;

    public static double sumaValores = 0;


    public static void main(String[] args) {

        //Recepción de parámetros.

        System.out.println("Valor de alfa:");

        alfa = sc.nextDouble();

        System.out.println("Valor de beta:");

        beta = sc.nextDouble();

        System.out.println("Valor de gamma:");

        gamma = sc.nextDouble();

        System.out.println();


        //Cálculo de constantes.

        p = (Math.cos(beta) - Math.cos(2 * alfa + gamma) + Math.sin(beta) + Math.sin(2 * alfa + gamma)) / 4;

        q = (Math.cos(beta) + Math.cos(2 * alfa + gamma) + Math.sin(beta) - Math.sin(2 * alfa + gamma)) / 4;

        valores[0] = (p * (Math.cos(beta) + Math.cos(gamma))) / 2;

        valores[1] = (p * (Math.sin(beta) + Math.sin(gamma))) / 2;

        valores[2] = (q * (Math.cos(beta) + Math.cos(gamma))) / 2;

        valores[3] = (q * (Math.sin(beta) + Math.sin(gamma))) / 2;

        valores[4] = ((1 - Math.cos(2 * alfa) + Math.sin(2 * alfa)) / 4) - ((p * (Math.cos(beta) + Math.cos(gamma))) / 2) - ((q * (Math.cos(beta) - Math.cos(gamma))) / 2);

        valores[5] = ((1 + Math.cos(2 * alfa) + Math.sin(2 * alfa)) / 4) - ((p * (Math.sin(beta) + Math.sin(gamma))) / 2) - ((q * (Math.sin(beta) - Math.sin(gamma))) / 2);

        valores[6] = ((1 + Math.cos(2 * alfa) - Math.sin(2 * alfa)) / 4) - ((p * (Math.cos(beta) - Math.cos(gamma))) / 2) - ((q * (Math.cos(beta) + Math.cos(gamma))) / 2);

        valores[7] = ((1 - Math.cos(2 * alfa) - Math.sin(2 * alfa)) / 4) - ((p * (Math.sin(beta) - Math.sin(gamma))) / 2) - ((q * (Math.sin(beta) + Math.sin(gamma))) / 2); //Corregida

        valores[8] = (q * (Math.cos(beta) - Math.cos(gamma))) / 2;
```

```
valores[9] = (q * (Math.sin(beta) - Math.sin(gamma))) / 2;

valores[10] = (p * (Math.cos(beta) - Math.cos(gamma))) / 2;

valores[11] = (p * (Math.sin(beta) - Math.sin(gamma))) / 2;


//Valores daubechies 12.

double valoresDaubechies12 [] = new double[12];


valoresDaubechies12[0] = 0.15774243 / 2;

valoresDaubechies12[1] = 0.69950381 / 2;

valoresDaubechies12[2] = 1.06226376 / 2;

valoresDaubechies12[3] = 0.44583132 / 2;

valoresDaubechies12[4] = -0.31998660 /2;

valoresDaubechies12[5] = -0.18351806 /2;

valoresDaubechies12[6] = 0.13788809 /2;

valoresDaubechies12[7] = 0.03892321 /2;

valoresDaubechies12[8] = -0.04466375 /2;

valoresDaubechies12[9] = 7.83251152e-4 / 2;

valoresDaubechies12[10] = 6.75606236e-3 /2;

valoresDaubechies12[11] = -1.52353381e-3 /2;




//Almacenamiento de diferencia de error.

double errores[] = new double[12];

errores[0] = valores[0] - valoresDaubechies12[0];

errores[1] = valores[1] - valoresDaubechies12[1];

errores[2] = valores[2] - valoresDaubechies12[2];

errores[3] = valores[3] - valoresDaubechies12[3];

errores[4] = valores[4] - valoresDaubechies12[4];

errores[5] = valores[5] - valoresDaubechies12[5];
```

```
errores[6] = valores[6] - valoresDaubechies12[6];

errores[7] = valores[7] - valoresDaubechies12[7];

errores[8] = valores[8] - valoresDaubechies12[8];

errores[9] = valores[9] - valoresDaubechies12[9];

errores[10] = valores[10] - valoresDaubechies12[10];

errores[11] = valores[11] - valoresDaubechies12[11];


//Impresión de margen de error.

System.out.println("Margen de error");

System.out.println();


//Cálculo de error cuadrático medio.

for (double errore : errores) {

   sumaError += errore*errore;

}

//Cálculo suma filtros Daubechies.

for (double errore : valoresDaubechies12) {

   sumaDaubechies += errore;

}

//Cálculo de los valores de los filtros.

//calculados de los parámetros.

for (double errore : valores) {

   sumaValores += errore;

}


System.out.println();

//Comprobación de las ecuaciones de momentos de desvanecimiento.

//Clase CompEcuación.

CompEcuacion ce = new CompEcuacion();

ce.comprobarDerivadas(alfa, beta, gamma);
```

```
        //Impresión de los valores de los filtros.
        System.out.println("Valores de los filtros");
        System.out.println();
        for (double valor : valores) {
            System.out.println(valor);
        }
    }
}
```

Clase CompEcuacion

```
package comprobador;


//Clase CompEcuacion.
public class CompEcuacion {
    //Metodo de comprobación de ecuaciones de desvanecimiento.
    public void comprobarDerivadas(double alfa, double beta, double gamma){


        //Cálculo de los filtros a partir de los parámetros.
        double a[] = new double[6];
        double b[] = new double[6];
        double p = (Math.cos(beta) - Math.cos(2 * alfa + gamma) + Math.sin(beta) + Math.sin(2 * alfa + gamma)) / 4;
        double q = (Math.cos(beta) + Math.cos(2 * alfa + gamma) + Math.sin(beta) - Math.sin(2 * alfa + gamma)) / 4;
        a[0] = (p * (Math.cos(beta) + Math.cos(gamma))) / 2;
        b[0] = (p * (Math.sin(beta) + Math.sin(gamma))) / 2;
        a[1] = (q * (Math.cos(beta) + Math.cos(gamma))) / 2;
        b[1] = (q * (Math.sin(beta) + Math.sin(gamma))) / 2;
        a[2] = ((1 - Math.cos(2 * alfa) + Math.sin(2 * alfa)) / 4) - ((p * (Math.cos(beta) + Math.cos(gamma))) / 2) - ((q * (Math.cos(beta) -
Math.cos(gamma))) / 2);
        b[2] = ((1 + Math.cos(2 * alfa) + Math.sin(2 * alfa)) / 4) - ((p * (Math.sin(beta) + Math.sin(gamma))) / 2) - ((q * (Math.sin(beta) -
Math.sin(gamma))) / 2);
```

```
        a[3] = ((1 + Math.cos(2 * alfa) - Math.sin(2 * alfa)) / 4) - ((p * (Math.cos(beta) - Math.cos(gamma))) / 2) - ((q * (Math.cos(beta) + Math.cos(gamma))) / 2);

        b[3] = ((1 - Math.cos(2 * alfa) - Math.sin(2 * alfa)) / 4) - ((p * (Math.sin(beta) - Math.sin(gamma))) / 2) - ((q * (Math.sin(beta) + Math.sin(gamma))) / 2); //Corregida

        a[4] = (q * (Math.cos(beta) - Math.cos(gamma))) / 2;

        b[4] = (q * (Math.sin(beta) - Math.sin(gamma))) / 2;

        a[5] = (p * (Math.cos(beta) - Math.cos(gamma))) / 2;

        b[5] = (p * (Math.sin(beta) - Math.sin(gamma))) / 2;


        //Cálculo de los valores de cada ecuación de los momentos te desvanecimiento.

        double hk = 0 + b[0] + (-2 * a[1]) + (3 * b[1]) + (-4 * a[2]) + (5 * b[2]) + (-6 * a[3]) + (7 * b[3]) +

                (-8 * a[4]) + (9 * b[4]) + (-10 * a[5]) + (11 * b[5]);//Primera derivada, dos momentos de desvanecimiento

        double hz = 0 + 0 + (2 * a[1]) + (-6 * b[1]) + (12 * a[2]) + (-20 * b[2]) + (30 * a[3]) + (-42 * b[3]) +

                (56 * a[4]) + (-72 * b[4]) + (90 * a[5]) + (-110 * b[5]); //Segunda derivada, tres momentos de desvanecimiento

        double hx = 0 + 0 + 0 + (6 * b[1]) + (-24 * a[2]) + (60 * b[2]) + (-120 * a[3]) + (210 * b[3]) +

                (-336 * a[4]) + (504 * b[4]) + (-720 * a[5]) + (990 * b[5]); //Tercera derivada, cuatro momentos de desvanecimiento

        double ha = 0 + 0 + 0 + 0 + (24 * a[2]) + (-60 * b[2]) + (360 * a[3]) + (-840 * b[3]) +

                (1680 * a[4]) + (-3024 * b[4]) + (5040 * a[5]) + (-7920 * b[5]); //Cuarta derivada, cinco momentos de desvanecimiento


        //IMpresión de los valores de cada escuación.

        System.out.println();

        System.out.println("Valor de la primera derivada: " + hk);

        System.out.println("Valor de la segunda derivada: " + hz);

        System.out.println("Valor de la tercera derivada: " + hx);

        System.out.println("Valor de la cuarta derivada: " + ha);

        System.out.println();

    }

}
```