Universidad
Politécnica
de Cartagena

industriales
etsii UPCT

MULTIRRESOLUCIÓN VECTORIAL DE HERMITE. APLICACIONES.

Titulación: Ingeniería Industrial
Alumno: Jesús Llamas Munguía
Director: Juan Carlos Trillo Moya

Cartagena, 19 de Noviembre de 2013

A toda mi familia, en especial a mis padres y a mi hermana, por su apoyo incondicional.

A esas personas que han estado a mi lado en estos años y me han ayudado cuando lo he necesitado.

A mi director Juan Carlos, por su ayuda y predisposición,

A todos vosotros,

¡Muchas gracias!

Autor	Jesús Llamas Munguía
E-mail del Autor	chus_llamas@gmail.com
Directores	Juan Carlos Trillo Moya
E-mail del Director	jctrillo@upct.es
Título del PFC	Multirresolución Vectorial de Hermite. Aplicaciones.
Descriptores	Compresión de datos, reconstrucción de Hermite , multirresolución
Resumen	Los algoritmos de multirresolución se están utilizando actualmente para la compresión de datos y en particular para la compresión de imágenes digitales. En dichos algoritmos es crucial disponer de un operador de predicción preciso. La posibilidad que investigamos en este proyecto es la utilización de la interpolación de Hermite como operador de predicción, dentro de la multirresolución vectorial.
Titulación	Ingeniero Industrial
Departamento	Matemática Aplicada y Estadística
Fecha de Presentación	Noviembre de 2013

Índice de cuadros

Índice de figuras

Índice general

Capítulo 1

Introducción.

Las transformaciones de multirresolución son una de las herramientas más eficaces para la compresión de imágenes, de video, y de datos en general. Proporcionan algoritmos rápidos y buenos resultados en comparación con otras aproximaciones clásicas como los métodos basados en la transformada de Fourier.

En este proyecto se lleva a cabo el estudio de los algoritmos de Multirresolución de Harten para el caso de discretización por valores puntuales en 1D. En particular se estudia su generalización al caso vectorial, donde a parte de disponer de los valores de la función se dispone también de los valores puntuales de las primeras derivadas de la función.

La Multiresolución de Harten fue desarrollada para permitir el uso de reconstrucciones no lineales. En [5],[15] tenemos algunos ejemplos donde el uso de reconstrucciones no lineales mejora los resultados de las transformaciones wavelet standard, en el sentido de que no generan tantos coeficientes de detalle altos.

El objetivo del enfoque propuesto por Harten es la construcción de esquemas de multirresolución adaptados a cada proceso de discretización. El paso fundamental en la construcción de un esquema de multirresolución es la definición de un operador reconstrucción apropiado para la discretización que se está considerando.

Al utilizar técnicas de interpolación independientes de los datos, es decir, lineales, la capacidad de compresión del esquema de multirresolución se verá reducida. Por otra parte, al utilizar técnicas de interpolación que dependan de los datos, es decir, no lineales, la capacidad de compresión del esquema de multirresolución mejora ([6],[5], [11]).

La reducción en las capacidades de compresión de los esquemas lineales se debe en gran medida a que para conseguir mayor orden de aproximación en el operador de reconstrucción se requiere la utilización de más puntos

para su construcción, aumentando así el soporte de la reconstrucción. Esto hace que los casos en que se cruza una discontinuidad aumenten, y por tanto se empeore mucho la aproximación. Para evitar este hecho, una posibilidad es utilizar un soporte más compacto, y justamente esta idea hace que considerar como operador de reconstrucción la interpolación de Hermite pueda dar buenos resultados. Ahora bien, para utilizar la interpolación de Hermite necesitamos derivadas, y por esto nos vamos a centrar en el caso de multirresolución vectorial.

La multirresolución vectorial en el entorno de Harten fue estudiada en [14], [13], y es en estos trabajos donde nos hemos basado para la elaboración de este proyecto fin de carrera.

La memoria está organizada como sigue. En la Sección 2 se desarrollan los contenidos acerca de la Multirresolución de Harten en el entorno de valores puntuales. También estudiamos los métodos de reconstrucción de Lagrange, PPH y ENO. En la Sección 3 introducimos la Multirresolución Vectorial, así como la reconstrucción basada en interpolación de Hermite. Consideramos de manera más detallada el caso especial de utilizar sólo la primera derivada. En la Sección 4 se detallan los códigos programados en Matlab que han sido desarrollados. En la Sección 5 se elabora un tutorial de la Interfaz Gráfica, así como un ejemplo acerca del uso de la misma. En la Sección 6 se llevan a cabo experimentos numéricos. Y por último, en la Sección 7 expondremos las conclusiones obtenidas en este proyecto.

Capítulo 2

Multirresolución de Harten.

El objetivo de la multirresolución es obtener una reordenación multiescala de la información contenida en un conjunto de datos discretos a una cierta resolución. Por ejemplo, esta información puede ser el resultado de discretizar una función, denotada f, en un cierto espacio vectorial V^k, en el que k indica el nivel de resolución. Un mayor valor de k indica una mayor resolución. Para realizar la transición entre distintos niveles de resolución se utilizan dos operadores llamados decimación y predicción. El operador decimación proporciona información discreta a un nivel de resolución $k-1$ a partir de la información contenida en el nivel k:

$$D_k^{k-1} : V^k \to V^{k-1},$$

y debe ser lineal y sobreyectivo.

El operador predicción actúa en sentido opuesto, dando una aproximación a la información discreta en el nivel k a partir de la información contenida en el nivel $k-1$:

$$P_{k-1}^k : V^{k-1} \to V^k.$$

Además, al operador predicción no se le exige que sea lineal.

Los datos discretos se obtienen a partir de la discretización de una función f, para lo cual existen distintos operadores. Dependiendo del operador discretización utilizado, la secuencia de datos f^k que resulta es diferente. El objetivo del enfoque propuesto por Harten es la construcción de esquemas de multirresolución adaptados a cada proceso de discretización. Esto se consigue definiendo un operador reconstrucción apropiado. Estos operadores, discretización y reconstrucción, son los elementos a partir de los cuales se construyen los operadores decimación y predicción del esquema de multirresolución.

Formalmente, sea F un espacio de funciones:

$$F \subset \{f | f : \Omega \subset \mathbb{R}^m \longrightarrow \mathbb{R}\}.$$

El operador discretización asigna a cada elemento de este espacio, $f \in F$, una secuencia f^k de datos discretos perteneciente al espacio V^k. De este modo se define el operador discretización:

$$D_k : F \to V^k.$$

que ha de ser lineal y sobreyectivo y que a cada $f \in F$ le asocia:

$$f^k = D_k(f).$$

La reconstrucción opera en sentido inverso, tomando una secuencia de datos discretos para reconstruir, a partir de la información proporcionada por dichos datos, la función de la que provienen:

$$R_k : V^k \to F.$$

La principal novedad introducida por Harten consiste en que a este operador reconstrucción no se le exige que sea lineal.

Por motivos de consistencia, se requiere que los operadores discretización y reconstrucción satisfagan la siguiente condición:

$$D_k R_k f^k = f^k, \, \forall f^k \in V^k,$$

o expresado de otro modo:

$$D_k R_k = I_{V^k},$$

es decir, si tomamos la información reconstruida a partir de unos datos discretos con una cierta resolución y la discretizamos a ese mismo nivel de resolución, la información discreta obtenida coincide con la original.

En la Figura 2.1 se muestran las relaciones existentes entre los operadores discretización y reconstrucción, y los operadores decimación y predicción. Según estas relaciones, el operador decimación se define del siguiente modo:

$$D_k^{k-1} := D_{k-1} R_k.$$

Aunque aparentemente el operador decimación depende de la elección del operador reconstrucción, en realidad no es así si (y sólo si) la sucesión de operadores discretización $\{D_k\}$ es anidada, es decir, si se tiene:

$$D_k f = 0 \Longrightarrow D_{k-1} f = 0, \;\; \forall f \in F.$$

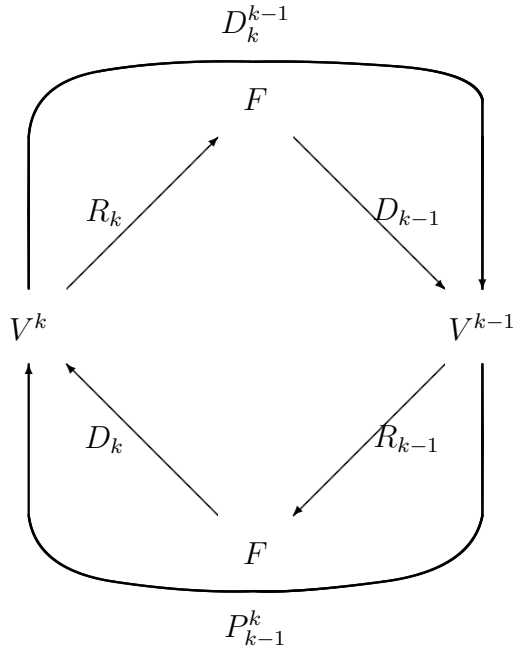

Figura 2.1: Definición de operadores.

La propiedad de anidamiento significa que la información contenida en los datos a un cierto nivel de resolución k no será nunca mayor que la información contenida en un nivel de resolución superior.
De forma similar, el operador predicción se construye según la expresión:

$$P_{k-1}^k := D_k R_{k-1}.$$

A partir de estas definiciones se obtiene la siguiente relación de consistencia para los operadores decimación y predicción:

$$D_k^{k-1} P_{k-1}^k = D_{k-1} R_k D_k R_{k-1} = D_{k-1} R_{k-1} = I_{V^{k-1}}.$$

Esta última relación lo que significa es que cuando utilizamos estos operadores no inventamos información, es decir, si decimamos la información obtenida a partir de la predicción realizada sobre una información con resolución dada por V^{k-1}, obtenemos exactamente la misma información de partida, sin introducir ningún elemento nuevo.
Consideremos ahora f^k, la información discreta en el nivel de resolución k. Si aplicamos el operador decimación sobre f^k obtenemos f^{k-1}, es decir, la información contenida en el nivel de resolución $k-1$:

$$f^{k-1} = D_k^{k-1} f^k.$$

En este caso, podemos interpretar que $P_{k-1}^k f^{k-1}$ es una aproximación a f^k, con un error:

$$e^k := \left(I_{V^k} - P_{k-1}^k D_k^{k-1}\right) f^k =: Q_k f^k \in V^k.$$

De esta forma podemos representar la información contenida en f^k en la forma ya descrita, y recíprocamente, conociendo f^{k-1} y e^k se puede calcular f^k mediante la expresión $P_{k-1}^k f^{k-1} + e^k = f^k$.

El problema es que haciendo esto incluimos información redundante, ya que, si suponemos que V^k es un espacio de dimensión finita (como lo es habitualmente en la práctica), $dim\ V^k = N_k$, tenemos por un lado f^k, que contiene la información codificada en N_k elementos, mientras que $\left\{f^{k-1}, e^k\right\}$ contiene la misma información codificada en $N_{k-1} + N_k$ elementos. Esta información redundante puede ser eliminada, como consecuencia del siguiente resultado:

$$\begin{aligned} D_k^{k-1} e^k &= D_k^{k-1}\left(I_{V^k} - P_{k-1}^k D_k^{k-1}\right) v^k && (2.1)\\ &= D_k^{k-1} v^k - D_k^{k-1} P_{k-1}^k D_k^{k-1} v^k && (2.2)\\ &= D_k^{k-1} v^k - D_k^{k-1} v^k = 0, && (2.3) \end{aligned}$$

es decir, $e^k \in N\left(D_k^{k-1}\right) = \left\{f^k \in V_k : D_k^{k-1} f^k = 0\right\}$, cuya dimensión es $dimN\left(D_k^{k-1}\right) = dimV^k - dimV^{k-1} = N_k - N_{k-1}$.

Sea $\left\{\mu_i^k\right\}$ el conjunto de elementos que generan el espacio $N\left(D_k^{k-1}\right)$. Podemos expresar el error e^k como:

$$e^k = \sum d_i^k \mu_i^k.$$

Si definimos un operador G_k que asocie a cada elemento $e^k \in N\left(D_k^{k-1}\right)$ su correspondiente conjunto de coeficientes $\left\{d_i^k\right\}$, podemos establecer la siguiente equivalencia:

$$f^k \equiv \left\{f^{k-1}, d^k\right\},$$

mediante las relaciones:

$$\begin{aligned} f^{k-1} &= D_k^{k-1} f^k \\ d^k &= G_k\left(I - P_{k-1}^k D_k^{k-1}\right) f^k, \end{aligned}$$

para obtener $\left\{f^{k-1}, d^k\right\}$ a partir de f^k, y :

$$P_{k-1}^k f^{k-1} + E_k d^k = f^k,$$

en sentido inverso.

En este caso la equivalencia de información lo es también en cuanto a número de elementos utilizados para codificar dicha información, ya que en $\left\{f^{k-1}, d^k\right\}$ tenemos $N_{k-1}+(N_k - N_{k-1}) = N_k$ elementos, los mismos que hay en f^k. Decimos entonces que los coeficientes $\left\{d_i^k\right\}$ contienen la información no redundante del error de predicción, y serán llamados detalles.

Iterando este procedimiento en cada nivel de resolución, se consigue la descomposición multiescala que se muestra en la Figura 2.2, y que permite establecer la siguiente equivalencia:

$$f^L \equiv \left\{f^0, d^L, \ldots, d^1\right\},$$

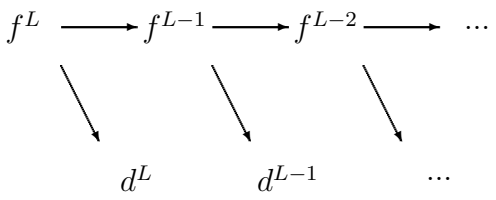

Figura 2.2: Descomposición multiescala.

Por tanto, y para resumir, los algoritmos para las transformaciones directa e inversa de la multirresolución son los siguientes:

(Directa)

$$f^L \to Mf^L = \left\{f^0, d^1, \ldots, d^L\right\} \begin{cases} \text{Do} \quad k = L, \ldots, 1 \\ f^{k-1} = D_k^{k-1} f^k \\ d^k = G_k(f^k - P_{k-1}^k f^{k-1}) \end{cases} \tag{2.4}$$

e

(Inversa)

$$Mf^L \to M^{-1}Mf^L \begin{cases} \text{Do} \quad k = 1, \ldots, L \\ f^k = P_{k-1}^k f^{k-1} + E_k d^k \end{cases} \tag{2.5}$$

El paso fundamental en la construcción de un esquema de multirresolución es la definición de un operador reconstrucción apropiado para la discretización que se está considerando. Habitualmente se utilizan dos tipos de

discretización: la discretización por valores puntuales y la discretización por medias en celda. Para este proyecto nos centraremos en la discretización por valores puntuales que pasamos a describir a continuación.

2.1. MR para la Discretización por Valores Puntuales en [0,1].

Se considera el conjunto de redes anidadas en el intervalo [0,1] dado por:

$$X^k = \{x_j^k\}_{j=0}^{J_k}, \quad x_j^k = jh_k, \quad h_k = 2^{-k}/J_0, \quad J_k = 2^k J_0,$$

donde J_0 es un entero fijo y X^k una partición uniforme en el intervalo unidad cerrado. La discretización por valores puntuales viene dada por:

$$D_k : \begin{cases} C([0,1]) & \to V^k \\ f & \mapsto f^k = (f_j^k)_{j=0}^{J_k} = (f(x_j^k))_{j=0}^{J_k} \end{cases} \tag{2.6}$$

donde V^k es el espacio de las secuencias reales de dimensión $J_k + 1$. Un operador de reconstrucción para esta discretización es cualquier operador R_k tal que:

$$R_k : V^k \to C([0,1]); \quad \text{y satisface} \quad D_k R_k f^k = f^k, \tag{2.7}$$

lo cual significa que:

$$(R_k f^k)(x_j^k) = f_j^k = f(x_j^k). \tag{2.8}$$

En otras palabras, $(R_k f^k)(x)$ es una función continua que interpola los datos f^k en X^k.

Si se escribe $(R_k f^k)(x) = I_k(x; f^k)$, entonces uno puede definir las transformadas directa (2.4) e inversa (2.5) de la multirresolución como:

$$f^L \to Mf^L \begin{cases} \text{Do} \quad k = L, \ldots, 1 & \\ \quad f_j^{k-1} = f_{2j}^k & 0 \le j \le J_{k-1}, \\ d_j^k \quad = f_{2j-1}^k - I_{k-1}(x_{2j-1}^k; f^{k-1}) & 1 \le j \le J_{k-1}. \end{cases} \tag{2.9}$$

y

$$Mf^L \to M^{-1}Mf^L \begin{cases} \text{Do} \quad k = 1, \ldots, L \\ f_{2j}^k = f_j^{k-1} & 1 \le j \le J_{k-1}, \\ f_{2j-1}^k = I_{k-1}(x_{2j-1}^k; f^{k-1}) + d_j^k & 0 \le j \le J_{k-1}. \end{cases} \tag{2.10}$$

Podemos pensar en el análisis de multirresolución como un análisis de la regularidad de una función. Los mayores coeficientes d_j^k van asociados a las singularidades de la función, lo que significa que no podemos predecir la información contenida en esas regiones. Más importante aún es el hecho de que si utilizamos una técnica de interpolación independiente de los datos, el conjunto de los intervalos afectados por una singularidad no se reduce únicamente a aquel intervalo en el que se localiza dicha singularidad, sino a todos los intervalos cuyo stencil contenga el intervalo donde se encuentra la singularidad, por lo que habrá una mayor cantidad de coeficientes con valores significativos, y la capacidad de compresión del esquema de multirresolución se verá reducida. Esto es lo que ocurre cuando se utiliza interpolación lineal centrada.

Por otra parte, al utilizar técnicas de interpolación que dependan de los datos, es decir, no lineales, se minimiza la zona afectada por cada singularidad, y la capacidad de compresión del esquema de multirresolución mejora. Esto ocurre cuando se utilizan las técnicas no lineales.

Las técnicas de interpolación más usuales son los polinomios.

2.2. Reconstrucciones Lineales y No Lineales.

En esta sección vamos a presentar algunas reconstrucciones lineales y no lineales usadas en el entorno de multirresolución de Harten. Las presentamos dentro del entorno de valores puntuales, pero se pueden utilizar también mediante una estrategia basada en la función primitiva en el entorno de medias en celda. Es en este marco de valores puntuales donde la definición de la reconstrucción es más clara y por eso hemos preferido esta presentación.
En primer lugar presentamos un operador de reconstrucción lineal basado en interpolación de Lagrange, para seguidamente presentar otras versiones no

lineales cuyo objetivo es la mejora de los puntos débiles de esta reconstrucción.
Las reconstrucciones las presentamos también para órdenes altos de aproximación, aunque se utilizan más las de cuarto orden (tercer orden en medias en celda).

2.2.1. Reconstrucción Lineal: Lagrange.

Consideremos los valores de la función $f_{j-1}, f_j, f_{j+1}, f_{j+2}$ que se corresponden con las abscisas $x_{j-1}, x_j, x_{j+1}, x_{j+2}$ de una malla regular X y vamos a describir cómo construir un trozo polinómico del operador de reconstrucción y la predicción $\hat{f}_{2j+1}$ en el punto medio $\frac{x_j+x_{j+1}}{2} = x_{j+\frac{1}{2}}$. La reconstrucción en el intervalo $[x_j, x_{j+1}]$ viene dada por

$$P_j(x) = f_{j-1}L_{-1}(x) + f_jL_0(x) + f_{j+1}L_1(x) + f_{j+2}L_2(x), \tag{2.11}$$

donde $L_i(x)$ $i = -1, 0, 1, 2$ son los polinomios de Lagrange dados por

$$L_i(x) = \prod_{l=-1, l\neq i}^{2} \frac{(x - x_{j+l})}{(x_{j+i} - x_{j+l})}. \tag{2.12}$$

$\hat{f}_{2j+1}$ se define como la evaluación en $x_{j+\frac{1}{2}}$ del polinomio de Lagrange $P_j(x)$, es decir

$$P_j(x_{j+\frac{1}{2}}) = f_{j-1}L_{-1}(x_{j+\frac{1}{2}}) + f_jL_0(x_{j+\frac{1}{2}}) + f_{j+1}L_1(x_{j+\frac{1}{2}}) + f_{j+2}L_2(x_{j+\frac{1}{2}}), \tag{2.13}$$

y haciendo algunos cálculos

$$P_j(x_{j+\frac{1}{2}}) = -\frac{1}{16}f_{j-1} + \frac{9}{16}f_j + \frac{9}{16}f_{j+1} - \frac{1}{16}f_{j+2}. \tag{2.14}$$

Al conjunto de valores $\{-\frac{1}{16}, \frac{9}{16}, \frac{9}{16}, -\frac{1}{16}\}$ se le denomina máscara del operador de predicción basado en interpolación segmentaria de Lagrange centrada.

En el caso general, podemos llegar fácilmente a una expresión similar a (2.11). Para ello, vamos a considerar el polinomio interpolador de Lagrange de grado $r = 2s - 1$ basado en el conjunto de $2s$ puntos dado por $\{x_{j-s+1}, \ldots, x_j, x_{j+1}, \ldots, x_{j+s}\}$ y sus correspondientes valores de función $\{f_{j-s+1}, \ldots, f_j, f_{j+1}, \ldots, f_{j+s}\}$. En este caso el operador de reconstrucción estará formado por la unión de trozos polinómicos del tipo

$$P_j(x) = \sum_{i=-s+1}^{s} f_{j+i}L_i(x_{j+\frac{1}{2}}), \quad x \in [x_j, x_{j+1}], \tag{2.15}$$

donde los polinomios de Lagrange $L_i(x)$ se definen como en (2.12) por

$$L_i(x) = \prod_{l=-s+1, l\neq i}^{s} \frac{(x - x_{j+l})}{(x_{j+i} - x_{j+l})}. \tag{2.16}$$

Tendremos que el operador de predicción en el punto medio toma la forma

$$P_j(x_{j+\frac{1}{2}}) = \sum_{i=-s+1}^{s} f_{j+i} L_i(x_{j+\frac{1}{2}}). \tag{2.17}$$

La máscara del operador de predicción en este caso es $\{L_i(x_{j+\frac{1}{2}})\}_{i=-s+1}^{i=s}$. En la Tabla 2.1 podemos ver los valores de las máscaras para $s = 2, 3, 4$.

	Máscaras
$s = 2$	$\{\frac{-1}{16}, \frac{9}{16}, \frac{9}{16}, \frac{-1}{16}\}$
$s = 3$	$\{\frac{3}{256}, \frac{-25}{256}, \frac{150}{256}, \frac{150}{256}, \frac{-25}{256}, \frac{3}{256}\}$
$s = 4$	$\{\frac{-5}{2048}, \frac{49}{2048}, \frac{-245}{2048}, \frac{1225}{2048}, \frac{1225}{2048}, \frac{-245}{2048}, \frac{49}{2048}, \frac{-5}{2048}\}$

Cuadro 2.1: Máscaras del operador de predicción basado en interpolación segmentaria de Lagrange centrada con $2s$ puntos para $s = 2, 3, 4$.

Para más detalles sobre esta reconstrucción y sobre las propiedades de los esquemas de subdivisión y multirresolución asociados se puede consultar [10].

2.2.2. Reconstrucción No Lineal: PPH.

Vamos a definir como construir un trozo polinómico PPH de orden $2s$ para el intervalo $[x_j, x_{j+1}]$. Partimos de los $2s$ datos

$$\{f_{j-s+1}, \ldots, f_{j-3}, f_{j-2}, f_{j-1}, f_j, f_{j+1}, f_{j+2}, f_{j+3} \ldots, f_{j+s}\},$$

y lo que vamos a hacer es modificar algunos de ellos para suavizar la función de manera que luego podamos aplicar interpolación de Lagrange sobre datos sin discontinuidades apreciables. Esta modificación es hecha también teniendo en cuenta que nuestro objetivo es dar una aproximación al valor de la función en el punto medio.

Definimos los coeficientes B_{s-1}^i por medio de la siguiente recurrencia

$$B_{s-1}^{s-1} = 2, \tag{2.18}$$

$$B_{s-1}^{q} = \sum_{j=1}^{s-1-q} (-1)^j \left(\binom{2(q+j)}{(j-1)} - \binom{2(q+j)}{j} \right) B_{s-1}^{q+j}, \tag{2.19}$$

para $q = s-2, \ldots, 1$.

En la Tabla 2.2.2 podemos ver el valor de estos coeficientes para $s = 2, 3, 4$.

	B^i_{s-1}
$s = 2$	$\{2\}$
$s = 3$	$\{2, 6\}$
$s = 4$	$\{2, 10, 12\}$

Cuadro 2.2: Coeficientes B^i_{s-1}, $s = 2, 3, 4$.

También vamos a necesitar las medias p-power $power_p(x, y)$, que se introdujeron en [25] para cualquier número entero $p \geq 1$, y cualquier pareja (x, y) como:

$$power_p(x, y) = \frac{sign(x) + sign(y)}{2} \frac{x+y}{2} \left(1 - \left|\frac{x-y}{x+y}\right|^p\right). \quad (2.20)$$

Igualmente necesitaremos usar las diferencias divididas, que es conocido que actúan como indicadores de zonas de suavidad de la función. Las diferencias divididas en el caso de nodos igualmente espaciados pueden calcularse haciendo uso del famoso triángulo de Tartaglia. En nuestro caso, donde nos interesa comparar el tamaño absoluto de dichas diferencias para detectar las zonas de suavidad, podremos prescindir de los denominadores. Y por tanto, su cálculo se reduce al de las diferencias finitas del mismo orden.

Llevaremos a cabo una modificación progresiva de los datos de la siguiente manera (ver [11] y [9] para más detalles). Observamos que esta modificación está diseñada para mantener el orden de interpolación en las regiones convexas suaves en el momento de aplicar la interpolación de Lagrange.

Modificación de datos de entrada $f \to \tilde{f}$

- Paso 1

  Consideramos $\{f_{j-1}, f_j, f_{j+1}, f_{j+2}\}$.

Si $|\Delta^2_{j-1}f| \leq |\Delta^2_j f|$ entonces

$$\tilde{f}_{j+2} := f_{j+1} + f_j - f_{j-1} + B^1_1 pow_{2s-2}(\Delta^2_{j-1}f, \Delta^2_j f),$$

en otro caso

$$\tilde{f}_{j-1} := f_{j+1} + f_j - f_{j+2} + B^1_1 pow_{2s-2}(\Delta^2_{j-1}f, \Delta^2_j f).$$

Así, definimos

$$\tilde{f} := \begin{cases} (\ldots, f_{j-2}, f_{j-1}, f_j, f_{j+1}, \tilde{f}_{j+2}, f_{j+3}, \ldots) & \text{si } |\Delta^2_{j-1}f| \leq |\Delta^2_j f|, \\ (\ldots, f_{j-2}, \tilde{f}_{j-1}, f_j, f_{j+1}, f_{j+2}, f_{j+3}, \ldots) & \text{en otro caso.} \end{cases} \tag{2.21}$$

- Paso 2

  Consideramos $\{f_{j-2}, \tilde{f}_{j-1}, \tilde{f}_j, \tilde{f}_{j+1}, \tilde{f}_{j+2}, f_{j+3}\}$.
  Si $|\Delta^4_{j-1}\tilde{f}| \leq |\Delta^4_j \tilde{f}|$ entonces

  $$\tilde{f}_{j+3} := f_{j+1} + f_j - f_{j-2} + B^1_2 pow_{2s-2}(\Delta^2_{j-1}f, \Delta^2_j f) + B^2_2 pow_{2s-4}(\Delta^4_{j-2}\tilde{f}, \Delta^4_{j-1}\tilde{f}),$$

  en otro caso

  $$\tilde{f}_{j-2} := f_{j+1} + f_j - f_{j+3} + B^1_2 pow_{2s-2}(\Delta^2_{j-1}f, \Delta^2_j f) + B^2_2 pow_{2s-4}(\Delta^4_{j-2}\tilde{f}, \Delta^4_{j-1}\tilde{f}).$$

  Así, definimos

  $$f = \begin{cases} (\ldots, f_{j-2}, f_{j-1}, f_j, f_{j+1}, \tilde{f}_{j+2}, f_{j+3}, \ldots) & \text{si } |\Delta^2_{j-1}f| \leq |\Delta^2_j f|, \\ (\ldots, f_{j-2}, \tilde{f}_{j-1}, f_j, f_{j+1}, f_{j+2}, f_{j+3}, \ldots) & \text{en otro caso.} \end{cases} \tag{2.22}$$

  y definimos

  $$\tilde{f} := \begin{cases} (\ldots, f_{j-3}, f_{j-2}, \tilde{f}_{j-1}, \tilde{f}_j, \tilde{f}_{j+1}, \tilde{f}_{j+2}, \tilde{f}_{j+3}, f_{j+4} \ldots) & \text{si } |\Delta^4_{j-2}\tilde{f}| \leq |\Delta^4_{j-1}\tilde{f}|, \\ (\ldots, f_{j-3}, \tilde{f}_{j-2}, \tilde{f}_{j-1}, \tilde{f}_j, \tilde{f}_{j+1}, \tilde{f}_{j+2}, f_{j+3}, f_{j+4} \ldots) & \text{en otro caso.} \end{cases} \tag{2.23}$$

  Y así se realizarán sucesivos pasos hasta completar la modificación de los $2s$ puntos.

Aplicando la interpolación de Lagrange de orden $2s$ con $\tilde{f}$, a los datos de entrada modificados en el apartado de arriba, obtenemos la interpolación no lineal deseada.

Por contrucción, esta técnica de interpolación no lineal nos lleva a un operador de reconstrucción con muchas características deseables.Primero, cada

pieza polinómica se construye con un stencil de $2s$ puntos. Segundo, la reconstrucción es tan precisa como su equivalente lineal en las regiones convexas suaves. Tercero, la precisión se reduce según se acerca a las singularidades, pero no se pierde totalmente como ocurre en su contraparte lineal. En particular, las reconstrucciones están libres de los efectos de Gibbs.

Por claridad vamos a estudiar unos casos particulares. Supongamos que tenemos cuatro puntos dispuestos como en la Figura 2.3. En el primer paso miraremos qué diferencia dividida de segundo orden es mayor, y a continuación cambiaremos el valor correspondiente. Sólo nos quedará entonces llevar a cabo una interpolación de Lagrange con estos datos como queda representado en la Figura 2.4.

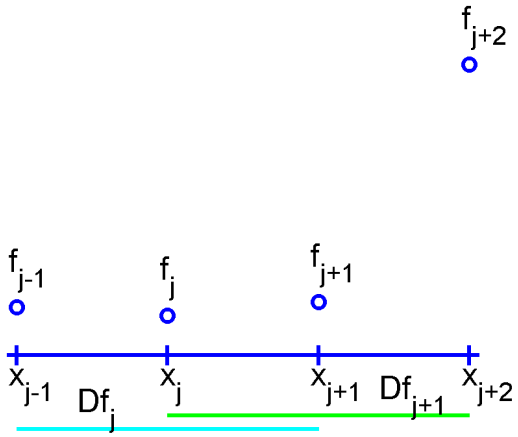

Figura 2.3: Primer paso de la reconstrucción PPH con 4 puntos.

En el siguiente ejemplo vamos a ilustrar de manera gráfica como se realizará la modificación de los datos en el caso de tener los valores de la Figura 2.5. En primer lugar se mirarán las diferencias divididas de segundo orden, que nos servirán para decidir que el valor a cambiar es f_{j-1} como aparece en la Figura 2.6. A continuación se considerarán las diferencias divididas de tercer orden indicadas en la Figura 2.7 y por tanto se cambiará el valor f_{j-2} tal y como se ve en la Figura 2.8.

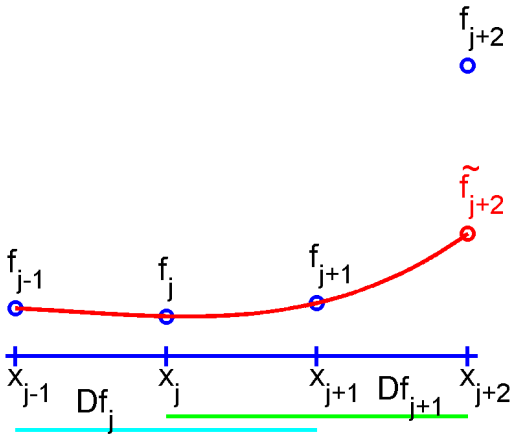

Figura 2.4: Construcción de la reconstrucción PPH con 4 puntos con los valores modificados.

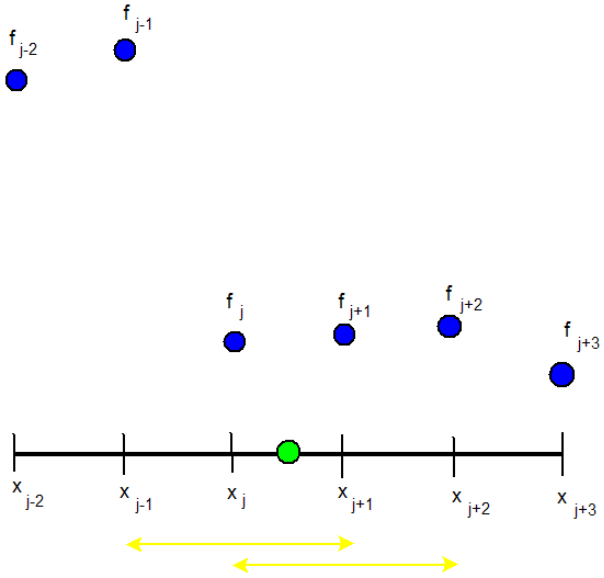

Figura 2.5: Primer paso de la reconstrucción PPH con 6 puntos.

2.2.3. Reconstrucción No Lineal: ENO.

La idea de las reconstrucciones ENO es construir trozos o partes polinomiales usando datos pertenecientes a las regiones suaves de la función en la

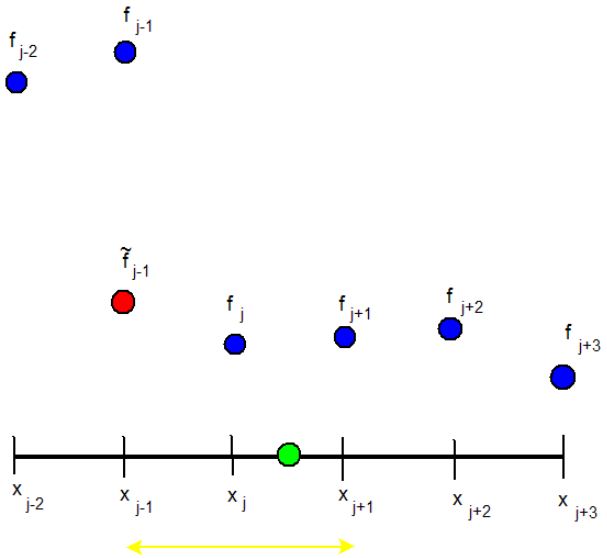

Figura 2.6: Modificación del primer valor en la construcción de la reconstrucción PPH con 6 puntos.

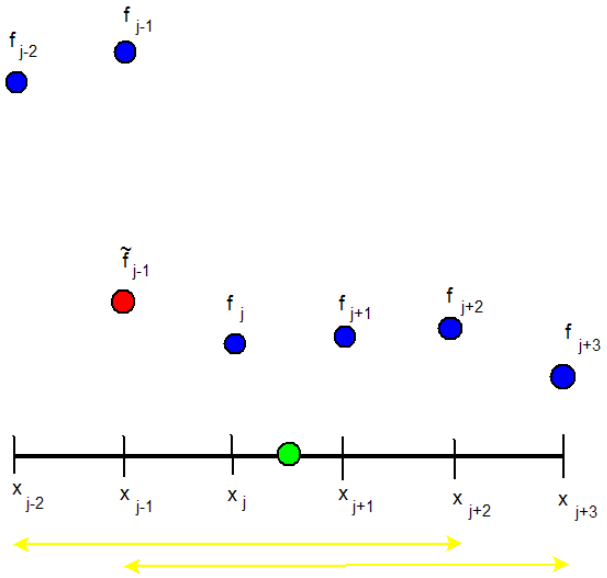

Figura 2.7: Segundo paso de la reconstrucción PPH con 6 puntos.

medida de lo posible. El punto clave en la interpolación ENO es el proceso de selección del stencil S (conjunto de datos usados para construir el polinomio interpolador), el cual se intenta elegir dentro de una región suave de la función $f(x)$. De forma más precisa, este proceso de selección trabaja de

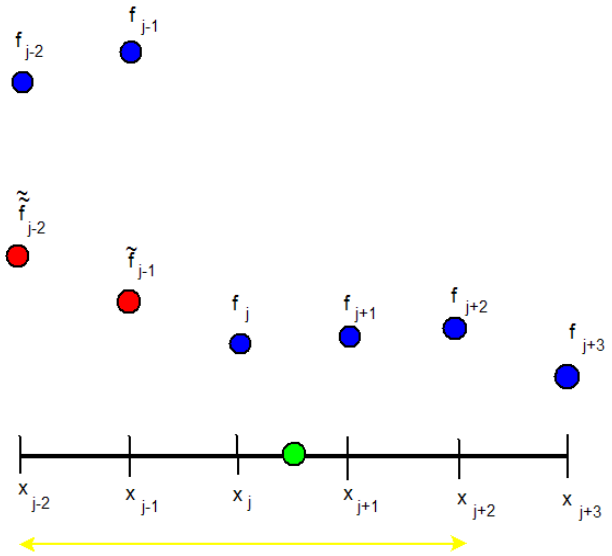

Figura 2.8: Modificación del segundo valor en la construcción de la reconstrucción PPH con 6 puntos.

la siguiente manera: para cada intervalo, $\left[x_{j-1}^k, x_j^k\right]$, se consideran todos los posibles conjuntos S con $r+1 \geq 2$ puntos incluyendo los puntos x_{j-1}^k y x_j^k.

Denominemos el stencil ENO para el jth intervalo:

$$S_j^{ENO} = \left\{ x_{s_{j-1}}^k, x_{s_j}^k, ..., x_{s_j+r-1}^k \right\},$$

siendo $r+1$ el orden de la interpolación.

Existen dos estrategias para realizar esta selección: la estrategia jerárquica y la no jerárquica.

La selección de S_j^{ENO} se realiza como sigue:

(1) La selección jerárquica del stencil consiste en, partiendo de los extremos del intervalo $\left[x_{j-1}^k, x_j^k\right]$, ir añadiendo puntos a derecha o izquierda, comparando las diferencias divididas correspondientes a los conjuntos formados por los extremos del intervalo más los puntos añadidos, y escogiendo la de menor valor absoluto. El algoritmo para este procedimiento es el siguiente:

Set $s_0 = j$
for $l = 0, \ldots, r-2$
  if $|f[x^k_{s_l-2}, \ldots, x^k_{s_l+l}]| < |f[x^k_{s_l-1}, \ldots, x^k_{s_l+l+1}]|$
    $s_{l+1} = s_l - 1$
  else
    $s_{l+1} = s_l$
  end
end
$s_j = s_{r-1}$

y,

(2) La selección no jerárquica del stencil considera las diferencias divididas de mayor orden correspondientes a todos los stencils posibles, y calcula el mínimo de entre todos los valores absolutos de dichas diferencias. El algoritmo es el siguiente:

Se elige un s_j tal que

Choose s_j such that
$|f[x^k_{s_j-1}, \ldots, x^k_{s_j+r-1}])| = \text{mín}_{j-r+1 \le l \le j}\{|f[x^k_{l-1}, \ldots, x^k_{l+r-1}]|\}$

Ambos algoritmos, (1) y (2) conducen asintóticamente a conjuntos de puntos de interpolación que se mueven lejos de la discontinuidad. Consecuentemente, el orden de aproximación del operador de predicción ENO sigue siendo $r+1$ siempre que sea posible evitar la discontinuidad.

Notar además, que no hay diferencia entre ambos algoritmos cuando $r = 2$, pero los stencils obtenidos pueden variar cuando $r > 2$. En cualquier caso, se observa que el stencil siempre contiene los extremos del subintervalo en el que se realiza la interpolación, evitando siempre que sea posible los intervalos que contienen singularidades.

En el caso en que $r = 3$ tenemos tres posibles stencils (ver Figura 2.9), es decir,

$S^1_j = \{x^k_{j-3}, x^k_{j-2}, x^k_{j-1}, x^k_j\}$,

$S^2_j = \{x^k_{j-2}, x^k_{j-1}, x^k_j, x^k_{j+1}\}$,

$S_j^3 = \left\{x_{j-1}^k, x_j^k, x_{j+1}^k, x_{j+2}^k\right\}$.

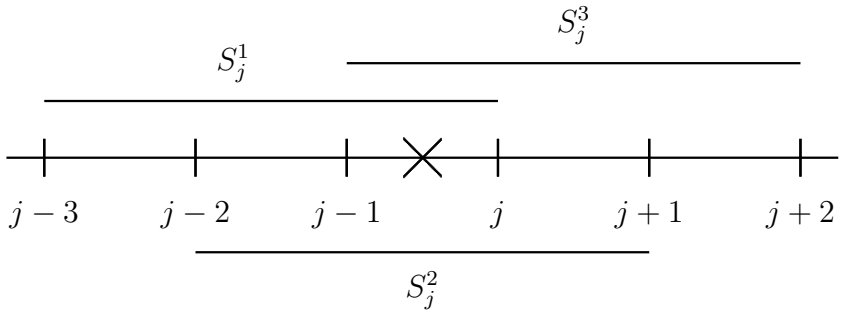

Figura 2.9: Stencil ENO.

El algoritmo de multirresolución queda,

$$\begin{cases} \text{Do} \quad k = L, \ldots, 1 \\ \quad f_j^{k-1} = f_{2j}^k \\ d_j^k = \begin{cases} f_{2j-1}^k - (5f_{j-3}^{k-1} + 15f_{j-2}^{k-1} - 5f_{j-1}^{k-1} + f_j^{k-1}), & si\ S_j^1 \\ f_{2j-1}^k - (\frac{-f_{j-2}^{k-1} + 9f_{j-1}^{k-1} + 9f_j^{k-1} - f_{j+1}^{k-1}}{16}), & si\ S_j^2 \\ f_{2j-1}^k - (5f_{j-1}^{k-1} + 15f_j^{k-1} - 5f_{j+1}^{k-1} + f_{j+2}^{k-1}), & si\ S_j^3 \end{cases} \end{cases} \tag{2.24}$$

$$\begin{cases} \text{Do} \quad k = 1, \ldots, L \\ \quad f_{2j}^k = f_j^{k-1} \\ f_{2j-1}^k = \begin{cases} d_j^k + (5f_{j-3}^{k-1} + 15f_{j-2}^{k-1} - 5f_{j-1}^{k-1} + f_j^{k-1}), & si\ S_j^1 \\ d_j^k + (\frac{-f_{j-2}^{k-1} + 9f_{j-1}^{k-1} + 9f_j^{k-1} - f_{j+1}^{k-1}}{16}), & si\ S_j^2 \\ d_j^k + (5f_{j-1}^{k-1} + 15f_j^{k-1} - 5f_{j+1}^{k-1} + f_{j+2}^{k-1}), & si\ S_j^3 \end{cases} \end{cases} \tag{2.25}$$

Capítulo 3

Multirresolución de Harten Vectorial

En este capítulo vamos a introducir la multirresolución de Harten vectorial para valores puntuales. En este contexto partimos de los valores puntuales de una cierta función, así como de los valores puntuales de su primera, segunda, ..., n-ésima derivada. Computaremos la versión de multirresolución de los datos de manera vectorial, considerando como entrada del algoritmo los datos dispuestos en forma de matriz, donde en la primera fila tendremos los valores de la función, en la segunda los valores de la primera derivada y así sucesivamente.

Como operador de reconstrucción utilizaremos la interpolación de Hermite de manera local. La utilización de esta interpolación nos permite aumentar el orden de reconstrucción sin aumentar mucho el soporte de la reconstrucción. Pero también es verdad que el orden de aproximación para las derivadas disminuirá en una unidad por derivada. Por esto veremos en los experimentos numéricos que a la hora de guardarnos detalles tendremos que hacerlo en mayor parte en los coeficientes correspondientes a las derivadas de orden superior.

3.1. Interpolación de Hermite

Nuestro objetivo es aplicar la interpolación de Hermite como operador de reconstrucción dentro de la multirresolución de Harten vectorial. Para llegar a entender bien cómo construir este operador de reconstrucción es aconsejable introducir primero la interpolación de Lagrange en la forma de Newton.

3.1.1. Interpolación de Lagrange en la forma de Newton

La polinomio interpolador de Lagrange en la forma de Newton se construye por medio de las diferencias divididas con la idea de que el polinomio así construido tenga propiedad de permanencia. Esta propiedad de permanencia ahorra cálculos posteriores si se da el caso de añadir un punto al conjunto de puntos usados para interpolar.

Dados los $n+1$ puntos (x_0, f_0),(x_1, f_1),...,(x_n, f_n), queremos construir un polinomio de grado menor o igual que n que pase por los puntos dados, es decir, que satisfaga $p(x_i) = f_i,\ \forall i = 1, \ldots, n$.

Sabemos que este problema de interpolación de Lagrange tiene solución única, y que la solución puede escribirse por medio de los polinomios de Lagrange asociados a cada nodo. Sin embargo ahora estamos interesados en escribir dicho polinomio de otra forma, usando diferencias divididas.

Resulta que el polinomio interpolador usando diferencias divididas toma la forma

$$p(x) = f_0 + f[x_0, x_1](x - x_0) + \ldots + f[x_0, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}).$$

Las diferencias divididas se calculan según la regla

$$f[x_0] = f(x_0),$$

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0},$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0},$$

$$\ldots\ldots\ldots,$$

$$f[x_0, x_1, \ldots\ldots, x_n] = \frac{f[x_1, x_2, \ldots\ldots, x_n] - f[x_0, x_1, \ldots\ldots, x_{n-1}]}{x_n - x_0}.$$

Y la fórmula del error como

$$|f(x) - p(x)| \leq |\frac{f^{(n+1)}(\theta)}{(n+1)!}(x - x_0) \cdots (x - x_n)|,$$

donde $\theta \in (x_0, x_n)$.

Como ejemplo vamos a construir el polinomio de tercer grado que pasa por los puntos $(0,1)$,$(2,3)$,$(4,6)$ y $(5,10)$. La expresión de dicho polinomio será

$$p_3(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) +$$

$$+ f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2).$$

Los tres primeros sumandos darán el mismo polinomio de Lagrange que interpola los tres primeros puntos, y añadiendo el cuarto sumando dará el polinomio interpolador de Lagrange que interpola los 4 puntos.

Una forma adecuada de calcular las diferencias divididas que necesitamos es a través de una tabla de diferencias divididas, tal y como la calculamos en este ejemplo. Notar que para calcular el valor de una celda $(i, j$ calculamos un cociente. El numerador se obtiene de restar el elemento $(i+1, j-1)$ con el $(i, j-1)$. El denominador de substraer el último y el primer nodo involucrado, es decir, de restar la celda $(i + j - 2, 0)$ con la $(i, 1)$.

x	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
0	1	$\frac{3-1}{2-0} = 1$	$\frac{3/2-1}{4-0} = \frac{1}{8}$	$\frac{5/6-1/8}{5-0} = \frac{34}{240} = \frac{17}{120}$
2	3	$\frac{6-3}{4-2} = \frac{3}{2}$	$\frac{4-3/2}{5-2} = \frac{5}{6}$	
4	6	$\frac{10-6}{5-4} = 4$		
5	10			

Cuadro 3.1: Tabla de diferencias divididas para el ejemplo de interpolación de Newton

De esta tabla nos interesa coger la primera fila para formar el polinomio interpolador de Lagrange.

$$p_3(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) +$$

$$+ f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) =$$

$$= 1 + 1(x - 0) + \frac{1}{8}(x - 0)(x - 2) + \frac{17}{120}(x - 0)(x - 2)(x - 4) =$$

$$= 1 + x + \frac{1}{8}x(x-2) + \frac{17}{120}x(x-2)(x-4).$$

Fórmula de error de la interpolación de Newton:

$$|f(x) - p_3(x)| = |\frac{f^{IV}(\theta)}{4!}(x-0)(x-2)(x-4)(x-5)|.$$

El denominador es 4! pues 4 es el número de puntos.

$\theta \in (min\{x,0,2,4,5\}, max\{x,0,2,4,5\}) \Rightarrow (min\{x,0\}, max\{x,5\})$, siendo x el punto donde calculo el polinomio interpolador. Por ejemplo, si quiero saber el error que da el polinomio de interpolación para $x = 3$,

$$|f(3) - p_3(3)| \leq |\frac{f^{IV}(\theta)}{4!}||3-0||3-2||3-4||3-5| = |\frac{f^{IV}(\theta)}{4!}|6 =$$

$$= |\frac{f^{IV}(\theta)}{4}|,$$

con

$$\theta \in (min\{3,0\}, max\{3,5\}) = (0,5),$$

$$f(3) \approx p_3(3).$$

En el siguiente apartado vamos a pasar a definir cómo llevar a cabo la interpolación de Hermite haciendo uso de las diferencias divididas generalizadas. Para construir dichas diferencias divididas también es posible crear una tabla del estilo de la vista en este apartado.

3.1.2. Interpolación de Hermite

La interpolación de Hermite consiste en encontrar un polinomio del grado requerido que satisfaga sobre cada punto x_i del stencil las condiciones $p^{(k)}(x_i) = f^{(k)}(x_i),\ \forall\ k = 1, \ldots, n_i$, donde el número de derivadas consecutivas consideradas puede depender del punto x_i, es decir, se puede considerar un número diferente de derivadas en cada punto.

De manera resumida podemos plantear el problema de la siguiente manera. Encontrar un polinomio $p(x)$ que satisfaga,

$$\begin{array}{llll} x_0 & x_1 & \dots & x_m \\ p(x_0) = f(x_0) & p(x_1) = f(x_1) & \dots & p(x_m) = f(x_m) \\ p'(x_0) = f'(x_0) & p'(x_1) = f'(x_1) & \dots & p'(x_m) = f'(x_m) \\ p''(x_0) = f''(x_0) & p''(x_1) = f''(x_1) & \dots & p''(x_m) = f''(x_m) \\ \dots & & & \\ p^{(n_0)}(x_0) = f^{(n_0)}(x_0) & p^{(n_1)}(x_1) = f^{(n_1)}(x_1) \dots & & p^{(n_m)}(x_m) = f^{(n_m)}(x_m) \end{array}$$

Reiteramos que las derivadas deben ser consecutivas, se tiene que tener de la primera a la última para cada nodo (cada columna). Las n_i, $\forall\, i = 1, \dots, m$ de cada derivada no tienen que ser las mismas.

Si contamos el número de condiciones que imponemos en cada punto

$$\begin{array}{c} \text{Condiciones en } x_0 \Rightarrow n_0 + 1 \\ \text{Condiciones en } x_1 \Rightarrow n_1 + 1 \\ \vdots \\ \text{Condiciones en } x_i \Rightarrow n_i + 1 \end{array}$$

El número de condiciones totales es por tanto $N = n_0 + n_1 + n_2 + ... + n_n + n + 1$.

El grado del polinomio es entonces igual al número de condiciones totales menos 1, es decir, $N - 1 = n_0 + n_1 + n_2 + ... + n_n + n$.

Diferencias divididas generalizadas:

$$f[x_0, x_0] = f'(x_0),$$

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \Rightarrow \text{cuando } x_1 \to x_0,\ f[x_0, x_0] = f'(x_0),$$

$$f[x_0, x_0, x_0] = \frac{f''(x_0)}{2!} \text{ porque } f[x_0, x_1, x_2] = \frac{f''(\xi)}{2!},$$
$$\text{siendo } \xi \in (min(x_0, x_1, x_2), max(x_0, x_1, x_2)).$$

Generalizando a $n + 1$ valores iguales de x_0,

$$f[x_0, x_0, ..., x_0] = \frac{f^{(n)}(x_0)}{n!}.$$

Si el primero y el último son distintos (los demás pueden ser iguales),

$$f[x_0, ..., x_n] = \frac{f[x_1, ..., x_n] - f[x_0, ..., x_{n-1}]}{x_n - x_0}.$$

El error en la interpolación de Hermite viene dado por

$$f(x) - p(x) = \frac{f^{(N)}(\theta)}{N!}(x - x_0)^{n_0} \cdots (x - x_i)^{n_i} \cdots (x - x_m)^{n_m},$$

$$\theta \in (min\{x, x_i\}, max\{x, x_i\}).$$

Cada factor $(x - x_i)$ va elevado al número de condiciones sobre x_i, y el número de la derivada y el factorial coinciden con el número de condiciones totales.

Debemos notar que la interpolación de Hermite es una generalización por un lado de la interpolación de Taylor, y por otro de la interpolación de Lagrange.

Ejemplo Hermite 1: Vamos a construir el polinomio de Hermite correspondiente a la siguiente tabla

x_0	x_1	x_2
$p(x_0) = f(x_0)$	$p(x_1) = f(x_1)$	$p(x_2) = f(x_2)$
	$p'(x_1) = f'(x_1)$	$p'(x_2) = f'(x_2)$
		$P''(x_2) = f''(x_2)$

En este caso tenemos los nodos $\{x_0, x_1, x_1, x_2, x_2, x_2\}$. Debemos observar que ponemos tantos nodos x_i como condiciones totales. Un nodo x_i se repite tantas veces como condiciones tengo sobre él. En este caso tenemos 6 condiciones, y por tanto el polinomio de Hermite será de grado 5.

$$\begin{aligned} p_5(x) &= f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_1](x - x_0)(x - x_1) \\ &+ f[x_0, x_1, x_1, x_2](x - x_0)(x - x_1)^2 \\ &+ f[x_0, x_1, x_1, x_2, x_2](x - x_0)(x - x_1)^2(x - x_2) \\ &+ f[x_0, x_1, x_1, x_2, x_2, x_2](x - x_0)(x - x_1)^2(x - x_2)^2. \end{aligned}$$

Error de Hermite:

$$f(x) - p_5(x) = \frac{f^{(6)}(\theta)}{6!}(x - x_0)(x - x_1)^2(x - x_2)^3,$$

$$\theta \in (min\{x, x_0, x_1, x_2\}, max\{x, x_0, x_1, x_2\}).$$

Cada factor $(x - x_i)$ va elevado al número de condiciones sobre x_i, y el número de la derivada y el factorial coinciden con el número de condiciones totales que es 6.

Ejemplo Hermite 2:

Construir un polinomio de grado menor o igual de 3 que satisfaga:

$p(1) = 3, \qquad p(2) = 1,$
$p'(1) = -1, \qquad p'(2) = 0.$

Hay 2 nodos $x_0 = 1$ y $x_1 = 2$, y 4 condiciones. Por tanto el conjunto de nodos será $\{x_0, x_0, x_1, x_1\}$. Buscamos un polinomio de la forma

$$\begin{aligned} p_3(x) &= f(x_0) + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 \\ &+ f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1). \end{aligned}$$

Tabla de diferencias generalizadas:

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
$x_0 = 1$	3	$f[x_0, x_0] = f'(1) = -1$	$f[x_0, x_0, x_1] = -1$	$f[x_0, x_0, x_1, x_1] = 3$
$x_0 = 1$	3	$f[x_0, x_1] = -2$	$f[x_0, x_1, x_1] = 2$	
$x_1 = 2$	1	$f[x_1, x_1] = f'(2) = 0$		
$x_1 = 2$	1			

Cuadro 3.2: Tabla de diferencias divididas generalizadas para el Ejemplo 2 de interpolación de Hermite.

Donde:

$$f[x_0, x_0, x_1] = \frac{f[x_0, x_1] - f[x_0, x_0]}{x_1 - x_0},$$

$$f[x_0, x_0, x_1, x_1] = \frac{f[x_0, x_1, x_1] - f[x_0, x_0, x_1]}{x_1 - x_0},$$

$$f[x_0, x_1, x_1] = \frac{f[x_1, x_1] - f[x_0, x_1]}{x_1 - x_0}.$$

Entonces el polinomio queda:

$$\begin{aligned} p_3(x) &= f(x_0) + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 \\ &+ f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1) \\ &= 3 - (x - 1) - (x - 1)^2 + 3(x - 1)^2(x - 2), \end{aligned}$$

y la fórmula del error de Hermite:

$$f(x) - p_3(x) = \frac{f^4(\theta)}{4!}(x - x_1)^2(x - x_2)^2,$$
$$\theta \in (min\{x, 1\}, max\{x, 2\}).$$

3.2. MR Vectorial de hermite: Función y derivada

En esta sección generalizamos el algoritmo de multirresolución escalar para el caso especial donde la función f y su derivada f' están dadas sobre el mallado X^m. En muchas aplicaciones, solo se dan los valores puntuales y las derivadas requeridas podrían necesitar ser calculadas por diferentes fórmulas de diferencias finitas.

Se proyectan los valores puntuales y los valores de las derivadas en el mallado formado por los nodos con índice par desde el nivel k al $k-1$:

$$f_j^{k-1} = f_{2j}^k,\ j = 0, 1, ..., J_{k-1}, \tag{3.1}$$

$$(f')_j^{k-1} = (f')_{2j}^k,\ j = 0, 1, ..., J_{k-1}. \tag{3.2}$$

Es conveniente eliminar la dependencia explícita del espaciado h_m, y para ello introducimos la definición $v(x_j^m)$ como

$$v_j^m = v(x_j^m) = h_m f'(x_j^m).$$

Los espaciados de dos niveles sucesivos están relacionados por $h_{k-1} = 2h_k$, y por tanto se puede reescribir (3.2) como

$$v_j^{k-1} = 2v_{2j}^k, j = 0, 1, ..., J_{k-1}. \tag{3.3}$$

Sobre el mallado uniforme X^{k-1}, las fórmulas de interpolación en el punto medio son:

$$\tilde{u}_{j-1/2}^{k-1} = I_u^{k-1}(x_{j-1/2}^{k-1}; [u_j^{k-1}, v_j^{k-1}]) = \frac{1}{2}(u_j^{k-1} + u_{j-1}^{k-1}) - \frac{1}{8}(v_j^{k-1} + v_{j-1}^{k-1}), \tag{3.4}$$

$$\tilde{v}_{j-1/2}^{k-1} = I_v^{k-1}(x_{j-1/2}^{k-1}; [u_j^{k-1}, v_j^{k-1}]) = \frac{3}{2}(u_j^{k-1} + u_{j-1}^{k-1}) - \frac{1}{4}(v_j^{k-1} + v_{j-1}^{k-1}). \tag{3.5}$$

Las proyecciones (3.1) y (3.3) son usadas para reescribir (3.4) y (3.5) sobre la red X^k:

$$\tilde{u}_{2j-1}^k = I_u^k(x_{2j-1}^k; [u_{2j}^k, v_{2j}^k]) = \frac{1}{2}(u_{2j}^k + u_{2j-2}^k) - \frac{1}{4}(v_{2j}^k + v_{2j-2}^k), \tag{3.6}$$

$$\tilde{v}_{2j-1}^k = I_v^k(x_{2j-1}^k; [u_{2j}^k, v_{2j}^k]) = \frac{3}{4}(u_{2j}^k + u_{2j-2}^k) - \frac{1}{4}(v_{2j}^k + v_{2j-2}^k). \tag{3.7}$$

Aquí la función $I_u^k(x_{2j-1}^k;[u_{2j}^k,v_{2j}^k])$ interpola los valores en los puntos impares u_{2j-1}^k (los puntos impares del mallado x_{2j-1}^k) usando los valores en los puntos pares $[u_{2j}^k,v_{2j}^k]$. Del mismo modo $I_v^k(x_{2j-1}^k;[u_{2j}^k,v_{2j}^k]$ interpola valores de la derivada en los puntos impares v_{2j-1}^k (los puntos impares del mallado x_{2j-1}^k) usando los valores en los puntos pares $[u_{2j}^k,v_{2j}^k]$. Una generalización obvia del algoritmo de multirresolución escalar es:

descomposición

Algoritmo 1

for $k=m,m-1,\ldots,n+1$
  for $j=0,1,\ldots,J_{k-1}$
    $u_j^{k-1}=u_{2j}^k,$
    $v_j^{k-1}=2v_{2j}^k,$
  end
  for $j=1,2,\ldots,J_{k-1}$
    $(r_u)_j^{k-1}=u_{2j-1}^k-I_u^k(x_{2j-1}^k;[u_{2j}^k,v_{2j}^k]),$
    $(r_v)_j^{k-1}=v_{2j-1}^k-I_v^k(x_{2j-1}^k;[u_{2j}^k,v_{2j}^k]),$
  end
end

reconstrucción

Algoritmo 2

for $k=n+1,n+2,\ldots,m$
  for $j=0,1,\ldots,J_{k-1}$
    $u_{2j}^k=u_j^{k-1},$
    $v_{2j}^k=\frac{1}{2}v_j^{k-1},$
  end
  for $j=1,2,\ldots,J_{k-1}$
    $u_{2j-1}^k=I_u^k(x_{2j-1}^k;[u_{2j}^k,v_{2j}^k])+(r_u)_j^{k-1},$
    $v_{2j-1}^k=I_v^k(x_{2j-1}^k;[u_{2j}^k,v_{2j}^k])+(r_v)_j^{k-1},$
  end
end

Alternativamente, el algoritmo anterior de reconstrucción puede ser escrito como

reconstrucción

Algoritmo 3

for $k = n+1, n+2, \ldots, m$
  for $j = 0, 1, \ldots, J_{k-1}$
    $u_{2j}^k = u_j^{k-1}$,
    $v_{2j}^k = \frac{1}{2} v_j^{k-1}$,
  end
  for $j = 1, 1, \ldots, J_{k-1}$
    $u_{2j-1}^k = I_u^{k-1}(x_{j-1/2}^{k-1}; [u_j^{k-1}, v_j^{k-1}]) + (r_u)_j^{k-1}$,
    $v_{2j-1}^k = \frac{1}{2} I_v^{k-1}(x_{j-1/2}^{k-1}; [u_j^{k-1}, v_j^{k-1}]) + (r_v)_j^{k-1}$,
  end
end

3.3. Aproximando derivadas

En el caso de que sólo tengamos los valores puntuales de la función, para aplicar la Multirresolución Vectorial de Hermite para función y primeras derivadas, necesitaremos aproximar las derivadas.

Para ello, utilizaremos la siguiente expresión, extraída del articulo [14]

$$\begin{aligned} f'(x_{j-1}) + 4f'(x_j) + f'(x_{j+1}) &= -\frac{3}{h}[f(x_{j-1}) - f(x_{j+1})] \\ &+ \frac{1}{30} f^{(5)}(\eta) h^4, \quad \text{con } x_{j-1} < \eta < x_{j+1}, \end{aligned} \tag{3.8}$$

donde $h = x_{j+1} - x_j$, para un espaciado uniforme.

Podemos escribir la ecuación anterior para todos los valores intermedios desde $j = 1$ hasta $n - 1 = J_k - 1$

$$\begin{aligned} f'_0 + 4f'_1 + f'_2 &= -\frac{3}{h}[f_0 - f_2], \ j = 1, \\ f'_1 + 4f'_2 + f'_3 &= -\frac{3}{h}[f_1 - f_3], \ j = 2, \\ &\vdots \\ f'_{n-2} + 4f'_{n-1} + f'_n &= -\frac{3}{h}[f_{n-2} - f_n], \ j = n-1. \end{aligned} \tag{3.9}$$

Como podemos ver, es mayor el número de incógnitas que de ecuaciones, por lo que resulta necesario calcular las derivadas en los extremos f'_0 y f'_n, mediante alguna fórmula de orden 4 disctretizada.

Por ejemplo:

$$f'(x_0) = \frac{c_0 f_0 + c_1 f_1 + c_2 f_2 + c_3 f_3 + c_4 f_4}{h} + O(h^4). \tag{3.10}$$

Haciendo los desarrollos de Taylor centrados en x_0, quedaría

$$c_0 f_0$$

$$c_1(f(x_0+h)) = c_1(f_0 + f'_0 h + \frac{f''_0}{2}h^2 + \frac{f'''_0}{3!}h^3 + \frac{f^{IV}_0}{4!}h^4 + \frac{f^V(\theta)}{5!}h^5),$$

$$c_2(f(x_0+2h)) = c_2(f_0+f'_0(2h)+\frac{f''_0}{2}(2h)^2+\frac{f'''_0}{3!}(2h)^3+\frac{f^{IV}_0}{4!}(2h)^4+\frac{f^V(\theta)}{5!}(2h)^5),$$

$$c_3(f(x_0+3h)) = c_3(f_0+f'_0(3h)+\frac{f''_0}{2}(3h)^2+\frac{f'''_0}{3!}(3h)^3+\frac{f^{IV}_0}{4!}(3h)^4+\frac{f^V(\theta)}{5!}(3h)^5),$$

$$c_4(f(x_0+4h)) = c_4(f_0+f'_0(4h)+\frac{f''_0}{2}(4h)^2+\frac{f'''_0}{3!}(4h)^3+\frac{f^{IV}_0}{4!}(4h)^4+\frac{f^V(\theta)}{5!}(4h)^5).$$

con $\theta \in (x_0, x_4)$.

Sumando todas las ecuaciones y dividiendo entre h, queremos obtener la ecuación (3.10), por lo que elegiremos los coeficientes del siguiente modo

$$\begin{aligned}
&c_0 + c_1 + c_2 + c_3 + c_4 = 0,\\
&c_1 + 2c_2 + 3c_3 + 4c_4 = 1,\\
&\frac{c_1}{2} + 2c_2 + \frac{9}{2}c_3 + 8c_4 = 0,\\
&\frac{c_1}{6} + \frac{4}{3}c_2 + \frac{9}{2}c_3 + \frac{32}{3}c_4 = 0,\\
&\frac{c_1}{24} + \frac{2}{3}c_2 + \frac{27}{8}c_3 + \frac{32}{3}c_4 = 0.
\end{aligned}$$

de donde obtenemos:

$c_0 = -\frac{25}{12}$, $c_1 = 4$, $c_2 = -3$, $c_3 = \frac{4}{3}$, $c_4 = -\frac{1}{4}$.

Con estos coeficientes quedaría:

$$f'(x_0) \approx \frac{-\frac{25}{12}f_0 + 4f_1 - 3f_2 + \frac{4}{3}f_3 - \frac{1}{4}f_4}{h}.$$

Podemos también escribirla de forma exacta, teniendo en cuenta la contribución de los términos con h^5 divididos entre h de los desarrollos de Taylor

$$f'(x_0) = \frac{-\frac{25}{12}f_0 + 4f_1 - 3f_2 + \frac{4}{3}f_3 - \frac{1}{4}f_4}{h} - \frac{1}{5}f^{(V)}(\theta)h^4.$$

Por simetría:

$$f'(x_n) = \frac{\frac{25}{12}f_n - 4f_{n-1} + 3f_{n-2} - \frac{4}{3}f_{n-3} + \frac{1}{4}f_{n-4}}{h} + \frac{1}{5}f^{(V)}(\mu)h^4,$$

con $\mu \in (x_{n-4}, x_n)$.

Por último, podríamos resolver ya el sistema de ecuaciones (3.9), donde tendríamos

$$\text{A}= \begin{pmatrix} 4 & 1 & 0 & \dots & \\ 1 & 4 & 1 & 0 & \dots \\ 0 & 1 & 4 & 1 & 0 \quad \dots \\ \vdots & 0 & \ddots & \ddots & \ddots \\ & \vdots & & & \end{pmatrix}, b = -\frac{3}{h}\begin{pmatrix} f_0 - f_2 \\ f_1 - f_3 \\ \vdots \\ f_{n-2} - f_n \end{pmatrix} + \begin{pmatrix} -f'_0 \\ 0 \\ \vdots \\ -f'_n \end{pmatrix},$$

$$\text{Incógnitas} = \begin{pmatrix} f'_1 \\ f'_2 \\ \vdots \\ f'_{n-1} \end{pmatrix}.$$

El problema de utilizar la ecuación (3.8), reside en que estamos doblando el número de puntos. Para solucionar este error, debemos aproximar las derivadas solo en los puntos impares. La expresión a utilizar en este caso es la siguiente:

$$\begin{aligned} f'(x_{j-2}) + 22f'(x_j) + f'(x_{j+2}) &= -\frac{12}{h}[f(x_{j-1}) - f(x_{j+1})] \\ &- \frac{1}{30}f^{(5)}(\eta)h^4, \quad \text{con } x_{j-2} < \eta < x_{j+2}, \end{aligned} \tag{3.11}$$

donde $h = x_{j+1} - x_j$.

Escribiendo la ecuación anterior para $j = 2 : 2 : n-2$, queda:

$$\begin{aligned} v_0 + 22v_2 + v_4 &= -24[f_1 - f_3],\ j = 2, \\ v_2 + 22v_4 + v_6 &= -24[f_3 - f_5],\ j = 4, \\ v_4 + 22v_6 + v_8 &= -24[f_5 - f_7],\ j = 6, \\ &\vdots \\ v_{n-4} + 22v_{n-2} + v_n &= -24[f_{n-3} - f_{n-1}],\ j = n-2, \end{aligned} \tag{3.12}$$

donde $v_j = 2hf'_j$.

Como pasaba en el caso anterior, el número de incógnitas es mayor que el número de ecuaciones, por lo que vuelve a ser necesario calcular las derivadas en los extremos mediante alguna fórmula de orden 4 discretizada. Al usar la misma fórmula, dichas derivadas son iguales que en el caso anterior. Por tanto, podríamos resolver ya el sistema 3.12, donde tendríamos:

$$\text{A}= \begin{pmatrix} 22 & 1 & 0 & \dots & & \\ 1 & 22 & 1 & 0 & \dots & \\ 0 & 1 & 22 & 1 & 0 & \dots \\ \vdots & 0 & \ddots & \ddots & \ddots & \\ & \vdots & & & & \end{pmatrix}, b = -24[\begin{pmatrix} f_1 \\ f_3 \\ \vdots \\ f_{n-3} \end{pmatrix} - \begin{pmatrix} f_3 \\ f_5 \\ \vdots \\ f_{n-1} \end{pmatrix}] + \begin{pmatrix} -v_0 \\ 0 \\ 0 \\ \vdots \\ -v_n \end{pmatrix},$$

$$\text{Incógnitas} = \begin{pmatrix} v_2 \\ \vdots \\ v_{n-3} \end{pmatrix}.$$

Para volver a recuperar los valores de la función f_j, $j = 0, ..., n$ a partir de los valores de v_j, f_j, $j = 0, 2, 4, ..., n$ aproximamos:

$$f_1 = \frac{1}{2}(f_0 + f_2) - \frac{1}{8}(v_2 + v_0),$$

$$f_{j+1} = f_{j-1} + \frac{1}{24}[v_{j-2} + 22v_j + v_{j+2}], \quad j = 2, 4, ..., n-2.$$

Capítulo 4

Algoritmos codificados en Matlab.

A continuación se presenta una generalización de los algoritmos para la Multirresolución Vectorial de Hermite en una dimensión. Estos algoritmos han sido codificados en Matlab.

4.1. Algoritmos para la Multirresolución Vectorial de Hermite.

4.1.1. MR Hermite 1D.

El siguiente programa, realiza la multirresolución vectorial utilizando la interpolación de Hermite como operador de predicción.

```
function [A,Ar,Ap,mrA,mrAt]=MR_Hermite1D(l,trun,fron,ruido,met,pos,varargin)


% Este programa realiza la multirresolución vectorial
% basada en interpolación de Hermite
%
% [A,Ar,Ap,mrA,mrAt]=MR_Hermite1D(l,trun,fron,ruido,met,pos,varargin)
%
% Variables de entrada:
% l son los niveles de multiresolucion que desciendes
% trun es la tolerancia de truncamiento, introducida en forma
% de vector para permitir diferente truncamiento según se
% trate de valores de función o de valores de alguna derivada.
```

```
% La primera entrada indica el método de truncamiento.
% trun=[1,com] se queda con los %com detalles mayores
% trun=[0,tol] indica la tolerancia de truncamiento, se queda con los
% detalles mayores en valor absoluto
% fron tipo de tratamiento en la frontera:
% 'p' periódico 'i' hacia el interior usando Lagrange
% 's' simétrico, 'a' antisimétrico
% 'c' extendido con ceros
% ruido es el nivel de ruido que se suma a los datos, también en forma
% vectorial
% met es un vector que indica cuantos nodos se toman en la interpolación
% de Hermite y cuantas condiciones hay sobre cada nodo
% pos posición del extremo izquierdo del intervalo de aproximación
% [x_j,x_{j+1}] dentro del conjunto de nodos
% varargin puede tomar los siguientes valores:
% 1) f,[a,b],n,nd en este caso se le pasa una cadena de caracteres con
% la función a considerar, el intervalo donde discretizar,
% el número de puntos donde discretizar, y el número de derivadas
% 2) A matriz de datos, que contiene los valores de la función en la primera
% fila y los de las derivadas en las filas sucesivas
% h espaciado entre los puntos
%
% Variables de Salida
% A matriz original
% Ar matriz original con ruido
% Ap aproximación obtenida del algoritmo inverso de MR partiendo inicial-
mente de Ar
% mrA versión de multirresolución de los datos
% mrAt versión de multirresolución de los datos ya truncada
%
% Ejemplo de aplicación:
% syms x;
% f='sin(x)';
% x1=linspace(0,2*pi,129);
% for i=1:2
%     A(i,:)=subs(diff(f,i-1),x1);
% end
% [A,Ar,Ap,mrA,mrAt]=MR_Hermite1D(3,[1,10^(-6),10^(-6)],'p',[0,0],[2,2],1,A)
```

```
% definición de los datos, que se guardan en la matriz A

if nargin==8
    A=varargin{1};
    n=size(A,2);
    nd=size(A,1);
    h=varargin{2}
elseif nargin==10
    f=varargin{1};
    extremos=varargin{2};
    n=varargin{3};
    nd=varargin{4};
    h=(extremos(2)-extremos(1))/(n-1);
    x=linspace(extremos(1),extremos(2),n);
    for i=1:nd
        A(i,:)=subs(diff(f,i-1),x);
    end

else
    display('Número inadecuado de variables de entrada');
    return;
end


% introducimos ruido en los datos

for i=1:nd
    Ar(i,:)=A(i,:)+ruido(i)*rand(1,n);
end


% descendemos por la pirámide de multirresolución
[mrA]=descender(Ar,h,l,fron,met,pos);
```

```
    % truncamiento de los coeficientes de detalle
[mrAt,rat]=truncarvp(mrA,l,trun);


% ascendemos por la pirámide de multirresolución
Ap=ascender(mrAt,h,l,fron,met,pos);




%%%%%%%%%%%%%%%%%%%%%%%%%%
% SALIDA DE RESULTADOS
%%%%%%%%%%%%%%%%%%%%%%%%%%%


if any(ruido)==0 % no hay ruido


   % calculamos los errores


   E_AAp=abs(A-Ap);
   E2_AAp=E_AAp.^2;

   % entre A y Ap

   norma1_AAp=sum(E_AAp')/n;
   Eg_medio_AAp_norma1=sum(norma1_AAp)/nd
   Eg_maximo_AAp_norma1=max(norma1_AAp)


   norma2_AAp=sum(E2_AAp')/n;
   Eg_medio_AAp_norma2=sum(norma2_AAp)/nd;
   Eg_maximo_AAp_norma2=max(norma2_AAp);


   normaI_AAp=max(E_AAp');
   Eg_medio_AAp_normaI=sum(normaI_AAp)/nd;
   Eg_maximo_AAp_normaI=max(normaI_AAp);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SALIDA DE RESULTADOS A UN FICHERO
%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Se abre o crea un archivo y se escribe en él para los resultados de
valores interpolados

fid=fopen('result_MR_Hermite.txt','w');
fprintf(fid,'*******************************************************\n');
hora=clock;
fprintf(fid,'Resultado ejecutado el día %d- %d- %d a las %d : %d : %d\n',...
hora(3),hora(2),hora(1),hora(4),hora(5),fix(hora(6)));
fprintf(fid,'*******************************************************\n\n');
fprintf(fid,' %f \n',rat);
for i=1:nd
    fprintf(fid,' %e ',norma1_AAp(i));
end
fprintf(fid,' %e %e \n', Eg_medio_AAp_norma1,Eg_maximo_AAp_norma1);
for i=1:nd
    fprintf(fid,' %e ',norma2_AAp(i));
end
fprintf(fid,' %e %e \n',Eg_medio_AAp_norma2,Eg_maximo_AAp_norma2);
for i=1:nd
    fprintf(fid,' %e ',normaI_AAp(i));
end
fprintf(fid,' %e %e \n',Eg_medio_AAp_normaI,Eg_maximo_AAp_normaI);

fclose(fid);

% salida gráfica de los datos originales y la aproximación

for i=1:nd
    figure(i);
    hold on;
    plot(A(i,:),'k','LineWidth',1);
    plot(Ap(i,:),'r','LineWidth',1);
```

```
        legend('Datos Originales','Datos Aproximados');
        texto_titulo=['Derivada ',num2str(i-1)];
        title(texto_titulo);
    end



else % sí que hay ruido


    % calculamos los errores


    E_AAp=abs(A-Ap);
    E2_AAp=E_AAp.^2;
    E_ArAp=abs(Ar-Ap);
    E2_ArAp=E_ArAp.^2;

    % entre A y Ap

    norma1_AAp=sum(E_AAp')/n;
    Eg_medio_AAp_norma1=sum(norma1_AAp)/nd;
    Eg_maximo_AAp_norma1=max(norma1_AAp);


    norma2_AAp=sum(E2_AAp')/n;
    Eg_medio_AAp_norma2=sum(norma2_AAp)/nd;
    Eg_maximo_AAp_norma2=max(norma2_AAp);


    normaI_AAp=max(E_AAp');
    Eg_medio_AAp_normaI=sum(normaI_AAp)/nd;
    Eg_maximo_AAp_normaI=max(normaI_AAp);

    % entre Ar y Ap

    norma1_ArAp=sum(E_ArAp')/n;
    Eg_medio_ArAp_norma1=sum(norma1_ArAp)/nd;
    Eg_maximo_ArAp_norma1=max(norma1_ArAp);
```

```
norma2_ArAp=sum(E2_ArAp')/n;
Eg_medio_ArAp_norma2=sum(norma2_ArAp)/nd;
Eg_maximo_ArAp_norma2=max(norma2_ArAp);


normaI_ArAp=max(E_ArAp');
Eg_medio_ArAp_normaI=sum(normaI_ArAp)/nd;
Eg_maximo_ArAp_normaI=max(normaI_ArAp);



%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SALIDA DE RESULTADOS A UN FICHERO
%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Se abre o crea un archivo y se escribe en él para los resultados de
valores interpolados

fid=fopen('result_MR_Hermite.txt','w');
fprintf(fid,'****************************************************\n');
hora=clock;
fprintf(fid,'Resultado ejecutado el día %d- %d- %d a las %d : %d : %d\n',...
hora(3),hora(2),hora(1),hora(4),hora(5),fix(hora(6)));
fprintf(fid,'****************************************************\n\n');
fprintf(fid,' %f \n',rat);
for i=1:nd
    fprintf(fid,' %e ',norma1_AAp(i));
end
fprintf(fid,' %e %e \n',Eg_medio_AAp_norma1,Eg_maximo_AAp_norma1);
for i=1:nd
    fprintf(fid,' %e ',norma2_AAp(i));
end
fprintf(fid,' %e %e \n',Eg_medio_AAp_norma2,Eg_maximo_AAp_norma2);
for i=1:nd
    fprintf(fid,' %e ',normaI_AAp(i));
end
fprintf(fid,' %e %e \n',Eg_medio_AAp_normaI,Eg_maximo_AAp_normaI);
for i=1:nd
    fprintf(fid,' %e ',norma1_ArAp(i));
```

```
    end
    fprintf(fid,'%e %e \n', Eg_medio_ArAp_norma1,Eg_maximo_ArAp_norma1);
    for i=1:nd
        fprintf(fid,'%e ',norma2_ArAp(i));
    end
    fprintf(fid,'%e %e \n', Eg_medio_ArAp_norma2,Eg_maximo_ArAp_norma2);
    for i=1:nd
        fprintf(fid,'%e ',normaI_ArAp(i));
    end
    fprintf(fid,'%e %e \n', Eg_medio_ArAp_normaI,Eg_maximo_ArAp_normaI);

    fclose(fid);



    % salida gráfica de los datos originales, los datos con ruido
    % y los datos aproximados



    for i=1:nd
        figure(i);
        hold on;
        plot(A(i,:),'k','LineWidth',1);
        plot(Ar(i,:),'b','LineWidth',1);
        plot(Ap(i,:),'r','LineWidth',1);
        legend('Datos Originales','Datos con Ruido','Datos Aproximados');
        texto_titulo=['Derivada ',num2str(i-1)];
        title(texto_titulo);
    end


end
```

Se puede observar, que para poder realizar la interpolación de Hermite, ha sido preciso utilizar programas adicionales al visto anteriormente, y que se muestran a continuación.

4.1.2. Descender.

Este algoritmo utiliza la predicción para descender en la pirámide de multirresolución.

```
function [mrA]=descender(Ar,h,l,fron,met,pos)


% Este programa va desciendo en la pirámide de multirresolución mediante
% la predicción dada por la interpolación de Hermite
%
% [mrA]=descender(Ar,h,l,fron,met,pos);
%
% Variables de entrada:
% Ar datos iniciales en la escala superior. Cada fila corresponde a
% valores puntuales de la función y sus respectivas derivadas.
% h espaciado entre los nodos
% l niveles de multirresolución
% fron tipo de tratamiento en la frontera:
% 'p' periódico 'i' hacia el interior
% 's' simétrico, 'a' antisimétrico
% 'c' extendido con ceros
% met es un vector que indica cuantos nodos se toman en la interpolación
% de Hermite y cuantas condiciones hay sobre cada nodo
% pos posición del extremo izquierdo del intervalo de aproximación
% [x_j,x_{j+1}] dentro del conjunto de nodos
%
% Variables de salida:
% mrA versión de multirresolución de los datos
%
% Ejemplo de aplicación:
% syms x;
% f='sin(x)';
% x1=linspace(0,2*pi,129);
% h=x1(2)-x1(1);
% for i=1:2
%       A(i,:)=subs(diff(f,i-1),x1);
% end
% [mrA]=descender(A,h,3,'p',[2,2],1)
```

```
% definimos x como variable simbólica
syms x

% tamaño de los datos
[nd,n]=size(Ar);

% número de nodos en que está basada la predicción
num=length(met);


% número de intervalos que requieren datos extra en la frontera
nif=num-pos-1; % número de intervalos fuera

% bucle para los niveles de multirresolución

for k=1:l

    % actualizamos el espaciado h
    h=2*h;

    % nodos con que construir la aproximación de Hermite
    nodos=linspace(0,(num-1)*h,num);

    % valores significativos de la escala inferior

    f1=Ar(1:nd,1:2:n);

    nk1=(n-1)/2+1; % dimensión de una escala inferior


    % cálculo de los primeros detalles en la frontera izquierda
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



    if fron=='p'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=1:nif
            pf=nif-i+1; % puntos fuera del intervalo
```

```
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pf:nk1-1),f1(:,1:pd)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            f2(1,i)=Ar(1,2*i)-pred(1);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                f2(j,i)=Ar(j,2*i)-pred(j);
            end

        end

    elseif fron=='i'

        pf=0; % puntos fuera del intervalo
        pd=num; % puntos dentro
        datos=[f1(:,1:pd)];

        for i=1:nif

            xa=(i-0.5)*h; % abscisa intermedia del intervalo considerado
            % donde evaluar el polinomio y sus derivadas
            for j=1:num-pos+i
                for cf=1:met(j+pos-i)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            for j=num-pos+i+1:num
                for cf=1:met(pos-i)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end

            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
```

```
            f2(1,i)=Ar(1,2*i)-pred(1);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                f2(j,i)=Ar(j,2*i)-pred(j);
            end
        end

    elseif fron=='s'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=1:nif
            pf=nif-i+1; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
            datos=[f1(:,pf+1:-1:2),f1(:,1:pd)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            f2(1,i)=Ar(1,2*i)-pred(1);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                f2(j,i)=Ar(j,2*i)-pred(j);
            end

        end



    elseif fron=='a'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=1:nif
            pf=nif-i+1; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
```

```
        datos=[2*f1(:,1)-f1(:,pf+1:-1:2),f1(:,1:pd)];
        for j=1:num
            for cf=1:met(j)
                condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
            end
        end
        [dp,ya]=Hermite(nodos,condiciones,xa);
        pred(1)=ya;
        f2(1,i)=Ar(1,2*i)-pred(1);
        for j=2:nd
            dp=diff(dp,1);
            pred(j)=subs(dp,xa);
            f2(j,i)=Ar(j,2*i)-pred(j);
        end

    end



elseif fron=='c'

    xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
    % donde evaluar el polinomio y sus derivadas
    for i=1:nif
        pf=nif-i+1; % puntos fuera del intervalo
        pd=num-pf; % puntos dentro
        datos=[zeros(nd,pf),f1(:,1:pd)];
        for j=1:num
            for cf=1:met(j)
                condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
            end
        end
        [dp,ya]=Hermite(nodos,condiciones,xa);
        pred(1)=ya;
        f2(1,i)=Ar(1,2*i)-pred(1);
        for j=2:nd
            dp=diff(dp,1);
            pred(j)=subs(dp,xa);
            f2(j,i)=Ar(j,2*i)-pred(j);
        end
```

```
    end

end

% cálculo de los detalles intermedios
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
% donde evaluar el polinomio y sus derivadas

for i=nif+1:nk1-nif-1

    datos=f1(:,i-pos+1:i+num-pos);
    for j=1:num
        for cf=1:met(j)
            condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
        end
    end
    [dp,ya]=Hermite(nodos,condiciones,xa);
    pred(1)=ya;
    f2(1,i)=Ar(1,2*i)-pred(1);
    for j=2:nd
        dp=diff(dp,1);
        pred(j)=subs(dp,xa);
        f2(j,i)=Ar(j,2*i)-pred(j);
    end

end

% cálculo de los últimos detalles en la frontera derecha
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if fron=='p'

    xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
    % donde evaluar el polinomio y sus derivadas
    for i=nk1-1:-1:nk1-nif
```

```
            pf=i-nk1+num-pos; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pd+1:nk1),f1(:,2:pf+1)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            f2(1,i)=Ar(1,2*i)-pred(1);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                f2(j,i)=Ar(j,2*i)-pred(j);
            end

        end


    elseif fron=='i'

        pf=0; % puntos fuera del intervalo
        pd=num; % puntos dentro
        datos=[f1(:,nk1-pd+1:nk1)];

        for i=nk1-1:-1:nk1-nif

            xa=(num-nk1+i-0.5)*h; % abscisa intermedia del intervalo
            % considerado donde evaluar el polinomio y sus derivadas
            for j=1:num-pos-nk1+i
                for cf=1:met(num-j+1)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            for j=num-pos+i-nk1+1:num
                for cf=1:met(j-num+pos-i+nk1)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
```

```
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            f2(1,i)=Ar(1,2*i)-pred(1);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                f2(j,i)=Ar(j,2*i)-pred(j);
            end
        end

    elseif fron=='s'


        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=nk1-1:-1:nk1-nif
            pf=i-nk1+num-pos; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pd+1:nk1),f1(:,nk1-1:-1:nk1-pf)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            f2(1,i)=Ar(1,2*i)-pred(1);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                f2(j,i)=Ar(j,2*i)-pred(j);
            end

        end

    elseif fron=='a'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=nk1-1:-1:nk1-nif
            pf=i-nk1+num-pos; % puntos fuera del intervalo
```

```
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pd+1:nk1),2*f1(:,nk1)-f1(:,nk1-1:-1:nk1-pf)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            f2(1,i)=Ar(1,2*i)-pred(1);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                f2(j,i)=Ar(j,2*i)-pred(j);
            end

        end

    elseif fron=='c'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=nk1-1:-1:nk1-nif
            pf=i-nk1+num-pos; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pd+1:nk1),zeros(nd,pf)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            f2(1,i)=Ar(1,2*i)-pred(1);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                f2(j,i)=Ar(j,2*i)-pred(j);
            end
        end
    end
```

```
    % actualización de variables

    Ar(:,1:n)=[f1,f2]; clear f1; clear f2;
    n=nk1;

    % se cierra el bucle de los niveles de multirresolución
end

mrA=Ar;
```

4.1.3. Ascender.

Utilizamos esta función, para ir ascendiendo en la pirámide de multirresolución mediante la predicción dada por la interpolación de Hermite.

```
function Ap=ascender(mrAt,h,l,fron,met,pos)

% Este programa va ascendiendo en la pirámide de multirresolución mediante
% la predicción dada por la interpolación de Hermite
%
% Ap=ascender(mrAt,h,l,fron,met,pos)
%
% Variables de entrada:
% mrAt versión de multirresolución de los datos ya truncada. Cada fila corresponde a
% la multirresolución de la función y sus respectivas derivadas.
% h espaciado entre nodos
% l niveles de multirresolución
% fron tipo de tratamiento en la frontera:
% 'p' periódico 'i' hacia el interior
% 's' simétrico, 'a' antisimétrico
% 'c' extendido con ceros
% met es un vector que indica cuantos nodos se toman en la interpolación
% de Hermite y cuantas condiciones hay sobre cada nodo
% pos posición del extremo izquierdo del intervalo de aproximación
% [x_j,x_{j+1}] dentro del conjunto de nodos
%
% Variables de salida:
% Ap aproximación de los datos originales
%
% Ejemplo de aplicación:
% Ejemplo de aplicación:
% syms x;
% f='sin(x)';
% x1=linspace(0,2*pi,129);
% h=x1(2)-x1(1);
% for i=1:2
%      A(i,:)=subs(diff(f,i-1),x1);
% end
% [mrA]=descender(A,h,3,'p',[2,2],1)
```

```
% Ap=ascender(mrA,h,3,'p',[2,2],1)


% definimos x como variable simbólica
syms x

% tamaño de los datos
[nd,n]=size(mrAt);

% número de nodos en que está basada la predicción
num=length(met);


% número de intervalos que requieren datos extra en la frontera
nif=num-pos-1; % número de intervalos fuera


% número de coeficientes significativos en el nivel más bajo de
% multirresolución
nk1=(n-1)/(2^l)+1;


% construimos el paso h en la última escala
h=2^l*h;

% se trata del bucle para los niveles de multirresolución

for k=l:-1:1

    % nodos con que construir la aproximación de Hermite
    nodos=linspace(0,(num-1)*h,num);

    % hacemos el proceso inverso que en la codificación

    n=2*nk1-1; % tamaño de la escala superior

    f1=mrAt(1:nd,1:nk1);
    f2=mrAt(1:nd,nk1+1:n);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%
% valores significativos impares de la escala superior
%%%%%%%%%%%%%%%%%%%%%%%%

mrAt(1:nd,1:2:n)=f1(1:nd,1:nk1);


%%%%%%%%%%%%%%%%%%%%%%%%
% valores significativos pares de la escala superior
%%%%%%%%%%%%%%%%%%%%%%%%


% cálculo de los primeros detalles en la frontera izquierda
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if fron=='p'

    xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
     % donde evaluar el polinomio y sus derivadas
    for i=1:nif
        pf=nif-i+1; % puntos fuera del intervalo
        pd=num-pf; % puntos dentro
        datos=[f1(:,nk1-pf:nk1-1),f1(:,1:pd)];
        for j=1:num
            for cf=1:met(j)
                condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
            end
        end
        [dp,ya]=Hermite(nodos,condiciones,xa);
        pred(1)=ya;
        mrAt(1,2*i)=pred(1)+f2(1,i);
        for j=2:nd
            dp=diff(dp,1);
            pred(j)=subs(dp,xa);
            mrAt(j,2*i)=pred(j)+f2(j,i);
        end

    end

elseif fron=='i'
```

```
        pf=0; % puntos fuera del intervalo
        pd=num; % puntos dentro
        datos=[f1(:,1:pd)];

        for i=1:nif

            xa=(i-0.5)*h; % abscisa intermedia del intervalo considerado
            % donde evaluar el polinomio y sus derivadas
            for j=1:num-pos+i
                for cf=1:met(j+pos-i)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            for j=num-pos+i+1:num
                for cf=1:met(pos-i)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end

            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            mrAt(1,2*i)=pred(1)+f2(1,i);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                mrAt(j,2*i)=pred(j)+f2(j,i);
            end
        end

    elseif fron=='s'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=1:nif
            pf=nif-i+1; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
            datos=[f1(:,pf+1:-1:2),f1(:,1:pd)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
```

```
            end
        end
        [dp,ya]=Hermite(nodos,condiciones,xa);
        pred(1)=ya;
        mrAt(1,2*i)=pred(1)+f2(1,i);
        for j=2:nd
            dp=diff(dp,1);
            pred(j)=subs(dp,xa);
            mrAt(j,2*i)=pred(j)+f2(j,i);
        end

    end



elseif fron=='a'

    xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
    % donde evaluar el polinomio y sus derivadas
    for i=1:nif
        pf=nif-i+1; % puntos fuera del intervalo
        pd=num-pf; % puntos dentro
        datos=[2*f1(:,1)-f1(:,pf+1:-1:2),f1(:,1:pd)];
        for j=1:num
            for cf=1:met(j)
                condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
            end
        end
        [dp,ya]=Hermite(nodos,condiciones,xa);
        pred(1)=ya;
        mrAt(1,2*i)=pred(1)+f2(1,i);
        for j=2:nd
            dp=diff(dp,1);
            pred(j)=subs(dp,xa);
            mrAt(j,2*i)=pred(j)+f2(j,i);
        end

    end
```

```
elseif fron=='c'

    xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
    % donde evaluar el polinomio y sus derivadas
    for i=1:nif
        pf=nif-i+1; % puntos fuera del intervalo
        pd=num-pf; % puntos dentro
        datos=[zeros(nd,pf),f1(:,1:pd)];
        for j=1:num
            for cf=1:met(j)
                condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
            end
        end
        [dp,ya]=Hermite(nodos,condiciones,xa);
        pred(1)=ya;
        mrAt(1,2*i)=pred(1)+f2(1,i);
        for j=2:nd
            dp=diff(dp,1);
            pred(j)=subs(dp,xa);
            mrAt(j,2*i)=pred(j)+f2(j,i);
        end

    end


end


% cálculo de los detalles intermedios
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
% donde evaluar el polinomio y sus derivadas

for i=nif+1:nk1-nif-1

    datos=f1(:,i-pos+1:i+num-pos);
    for j=1:num
        for cf=1:met(j)
            condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
        end
```

```
        end
        [dp,ya]=Hermite(nodos,condiciones,xa);
        pred(1)=ya;
        mrAt(1,2*i)=pred(1)+f2(1,i);
        for j=2:nd
            dp=diff(dp,1);
            pred(j)=subs(dp,xa);
            mrAt(j,2*i)=pred(j)+f2(j,i);
        end


    end



    % cálculo de los últimos detalles en la frontera derecha
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if fron=='p'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
         % donde evaluar el polinomio y sus derivadas
        for i=nk1-1:-1:nk1-nif
            pf=i-nk1+num-pos; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pd+1:nk1),f1(:,2:pf+1)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            mrAt(1,2*i)=pred(1)+f2(1,i);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                mrAt(j,2*i)=pred(j)+f2(j,i);
            end

        end
```

```
elseif fron=='i'

    pf=0; % puntos fuera del intervalo
    pd=num; % puntos dentro
    datos=[f1(:,nk1-pd+1:nk1)];

    for i=nk1-1:-1:nk1-nif

        xa=(num-nk1+i-0.5)*h; % abscisa intermedia del intervalo
        % considerado donde evaluar el polinomio y sus derivadas

        for j=1:num-pos-nk1+i
            for cf=1:met(num-j+1)
                condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
            end
        end
        for j=num-pos+i-nk1+1:num
            for cf=1:met(j-num+pos-i+nk1)
                condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
            end
        end

        [dp,ya]=Hermite(nodos,condiciones,xa);
        pred(1)=ya;
        mrAt(1,2*i)=pred(1)+f2(1,i);
        for j=2:nd
            dp=diff(dp,1);
            pred(j)=subs(dp,xa);
            mrAt(j,2*i)=pred(j)+f2(j,i);
        end
    end

elseif fron=='s'


    xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
    % donde evaluar el polinomio y sus derivadas
    for i=nk1-1:-1:nk1-nif
        pf=i-nk1+num-pos; % puntos fuera del intervalo
```

```
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pd+1:nk1),f1(:,nk1-1:-1:nk1-pf)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            mrAt(1,2*i)=pred(1)+f2(1,i);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                mrAt(j,2*i)=pred(j)+f2(j,i);
            end

        end

    elseif fron=='a'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=nk1-1:-1:nk1-nif
            pf=i-nk1+num-pos; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pd+1:nk1),2*f1(:,nk1)-f1(:,nk1-1:-1:nk1-pf)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            mrAt(1,2*i)=pred(1)+f2(1,i);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                mrAt(j,2*i)=pred(j)+f2(j,i);
            end

        end
```

```
    elseif fron=='c'

        xa=(pos-0.5)*h; % abscisa intermedia del intervalo considerado
        % donde evaluar el polinomio y sus derivadas
        for i=nk1-1:-1:nk1-nif
            pf=i-nk1+num-pos; % puntos fuera del intervalo
            pd=num-pf; % puntos dentro
            datos=[f1(:,nk1-pd+1:nk1),zeros(nd,pf)];
            for j=1:num
                for cf=1:met(j)
                    condiciones{j}(cf)=datos(cf,j)/factorial(cf-1);
                end
            end
            [dp,ya]=Hermite(nodos,condiciones,xa);
            pred(1)=ya;
            mrAt(1,2*i)=pred(1)+f2(1,i);
            for j=2:nd
                dp=diff(dp,1);
                pred(j)=subs(dp,xa);
                mrAt(j,2*i)=pred(j)+f2(j,i);
            end

        end



    end


    % actualización de variables
    nk1=n;
    h=h/2;

    % se cierra el bucle de los niveles de multirresolución


end

Ap=mrAt;
```

4.1.4. Factorial.

Esta rutina se utiliza tanto en el programa descender como en el programa ascender. Realiza el factorial de un número entero n.

```
function nfac=factorial(n)

% Este programa calcula el factorial de un número entero dado
% nfac=factorial(n)
% Variables de entrada:
% n número natural del cual se calcula su factorial
% Variables de salida:
% nfac n*(n-1)*...*2*1

if n==0 — n==1
    nfac=1;
else
    nfac=n;
    for i=n-1:-1:2
        nfac=nfac*i;
    end

end
```

4.1.5. Diferencias divididas generalizadas.

```
function [fila1,tabla,nc]=diferencias_divididas_generalizadas(nodos,condiciones)

% Este programa calcula las diferencias divididas generalizadas
% [fila1,tabla,nc]=diferencias_divididas_generalizadas(nodos,condiciones);
% Variables de entrada:
% nodos son las abscisas donde se imponen las condiciones
% condiciones es una celda que contiene tantos vectores como nodos. Cada
% vector contiene los datos [f(xi),f'(xi),f"(xi)/2!,
% ...,f^(ni)(xi)/(ni)!]
% Variables de salida:
% fila1 contiene los coeficientes para formar el polinomio interpolador
% de Hermite
% tabla contiene el cuadro completo de las diferencias divididas
% nc número de condiciones que hay en cada nodo
%
% Ejemplo:
% Construir la tabla de diferencias divididas generalizadas con los datos
% x0=1; x1=2;
% f(x0)=1, f'(x0)=2
% f(x1)=-1, f'(x1)=3, f"(x1)=4
% [fila1,tabla,nc]=diferencias_divididas_generalizadas([1,2],{[1,2],[-1,3,2]})


% número de nodos

n=length(nodos);

% número de condiciones en cada nodo

nc_total=0;
for i=1:n
    nc(i)=length(condiciones{i});
    nc_total=nc_total+nc(i);
end


% inicializamos la tabla a cero

tabla=zeros(nc_total,nc_total+1);
```

```
% completamos las primeras dos columnas

posicion=1;
for i=1:n
    for j=1:nc(i)
        tabla(posición,1)=nodos(i);
        tabla(posición,2)=condiciones{i}(1);
        posicion=posicion+1;
    end
end

% completamos el resto de la tabla

for i=3:nc_total+1  % columna
    for j=1:nc_total-i+2  % fila

        % comprobamos si todos los nodos son iguales o no para aplicar
        % una regla de cálculo u otra

        if tabla(j,1)==tabla(j+i-2)  % son todos los nodos iguales
            tabla(j,i)=condiciones{find(tabla(j,1)==nodos)}(i-1);
        else
            tabla(j,i)=(tabla(j+1,i-1)-tabla(j,i-1))/(tabla(j+i-2)-tabla(j,1));

        end

    end
end

fila1=tabla(1,:);
```

4.1.6. truncarvp.

Por último, la siguiente función ha sido utilizada para truncar los detalles del vector multirresolucionado.

```
function [c,rat]=truncarvp(a,l,trun)

% funcion que trunca los detalles del vector multirresolucionado a
% discretización por valores puntuales
% c=truncarvp(a,l,tol)
% Variables de entrada:
% a matriz de multirresolución vectorial a la que truncarle los detalles
% l niveles de multirresolución
% trun es la tolerancia de truncamiento, introducida en forma
% de vector para permitir diferente truncamiento según se
% trate de valores de función o de valores de alguna derivada.
% La primera entrada indica el método de truncamiento.
% trun=[1,com] se queda con los %com detalles mayores
% trun=[0,tol] indica la tolerancia de truncamiento, se queda con los
% detalles mayores en valor absoluto
% Variables de salida:
% c vector con los detalles truncados
% rat porcentaje de detalles no cero


[nd,n]=size(a);
n1=(n-1)/2^l+1;

if trun(1)==0 % se truncan los detalles menores que una tolerancia

    tol=trun(2:end);
    ltol=length(tol);

    if ltol==1 % tolerancia fija independiente de la derivada que sea
        c=[a(:,1:n1),a(:,n1+1:n).*(abs(a(:,n1+1:n))>=trun(2))];
    elseif ltol==nd % tolerancia dependiente de la derivada considerada
        for i=1:nd
            c(i,1:n)=[a(i,1:n1),a(i,n1+1:n).*(abs(a(i,n1+1:n))>=tol(i))];
        end
    end
    nz=nnz(c(:,n1+1:n));
```

```
    rat=nz*100/(n-n1)/nd;

else % nos quedamos con un porcentaje de los detalles

    com=trun(2:end);
    lcom=length(com);

    if lcom==1 % porcentaje global independiente de la derivada que sea

        % nos quedamos con los num_det mayores coeficientes de detalle

        num_det=floor(com*(n-n1)*nd/100);

        c=a(:,n1+1:n);
        c=reshape(c,1,nd*(n-n1)); % interpreta la matriz c en forma de vec-
tor
        [y,ind]=sort(abs(c),'descend'); % ordenamos los coeficientes de mayor
a menor valor absoluto
        y=c(ind); % deja los coeficientes ordenados pero con signo
        y(num_det+1:end)=0; % eliminamos los menos significativos
        c(ind)=y; % pone en posición los coeficientes ya truncados
        c=reshape(c,nd,n-n1); % vuelve a forma de matriz
        a(:,n1+1:n)=c; % copia en a los coeficientes truncados


    elseif lcom==nd % porcentaje dependiente de la derivada considerada

        for i=1:nd

            % nos quedamos con los num_det mayores coeficientes de detalle
            num_det=floor(com(i)*(n-n1)/100);
            c=a(i,n1+1:n);
            [y,ind]=sort(abs(c),'descend'); % ordenamos los coeficientes de ma-
yor a menor valor absoluto
            y=c(ind); % deja los coeficientes ordenados pero con signo
            y(num_det+1:n-n1)=0; % eliminamos los menos significativos
            c(ind)=y; % pone en posición los coeficientes ya truncados
            a(i,n1+1:n)=c; % copia en a los coeficientes truncados


        end
```

```
    end

    c=a;
    nz=nnz(c(:,n1+1:n));
    rat=nz*100/(n-n1)/nd;


end
```

Capítulo 5

Interfaz Gráfica.

Se ha utilizado el entorno gráfico de Matlab (GUIDE) para codificar los algoritmos necesarios, con el fin de que el usuario pueda interactuar con el programa de una manera más sencilla y sin necesidad de tener amplios conocimientos del Matlab.

5.1. Ejecución de la Interfaz Gráfica

Una vez abierto Matlab, debemos situar el directorio de trabajo en el lugar adecuado. Esto es, en la ruta

"./PFC/Interfaz Gráfica".

Para ejecutar la interfaz gráfica, debemos introducir en la linea de comandos de Matlab el siguiente comando:

```
>> MR_Vectorial_De_Hermite.
```

Esta orden, nos lleva directamente a la interfaz principal del programa de usuario descrita en el siguiente apartado (Figura 5.1).

5.2. Documentación de la Interfaz Gráfica

A continuación procederemos a describir la interfaz principal de usuario (Figura 5.1), dividiéndola en dos partes: Menú y Cuadro Central.

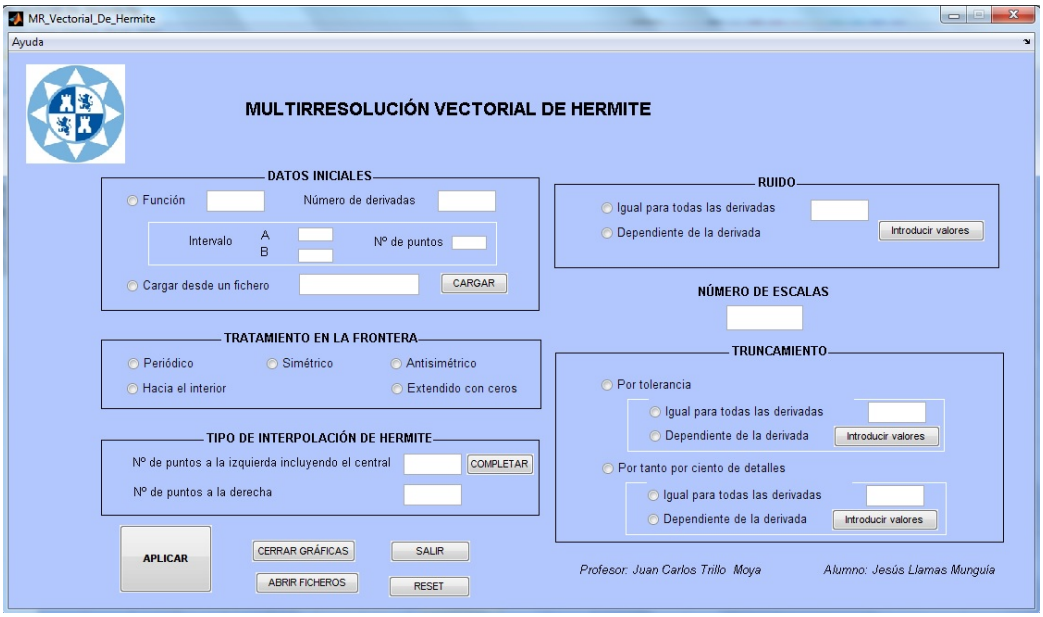

Figura 5.1: Interfaz gráfica principal de usuario.

5.2.1. Menú Principal

La interfaz principal de usuario cuenta con un menú en la parte superior izquierda (Figura 5.2) que consiste en una opción que puede desplegarse para llevar a cabo diferentes configuraciones.

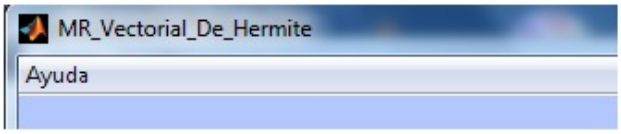

Figura 5.2: Menú de usuario.

Vamos a describir dicho menú y las diferentes opciones.

- **Ayuda**
  Este menú contiene un tutorial del manejo de la interfaz gráfica e información general acerca del proyecto (Figura 5.3).

  - **Tutorial**
    Seleccionando esta opción se abrirá un tutorial en formato pdf que contiene indicaciones sobre el manejo de la interfaz gráfica.

Figura 5.3: Submenús de ayuda.

- **Acerca de**
  Seleccionando esta opción recibiremos información general acerca del proyecto (Figura 5.4).

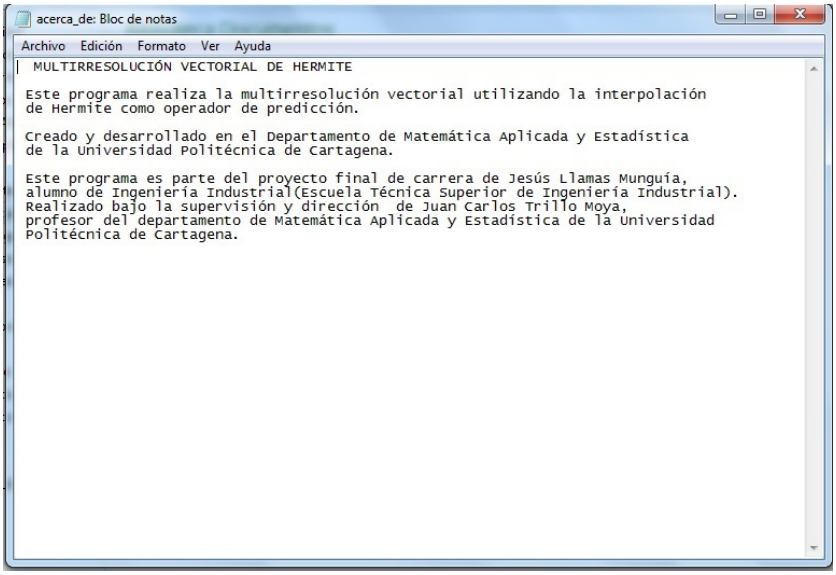

Figura 5.4: Información general acerca del proyecto.

5.2.2. Cuadro Central

Por "Cuadro Central" entendemos el resto de campos que forman la Interfaz Gráfica (Figura 5.1). Debemos configurar cada campo según los requerimientos. Iremos explicando campo a campo el significado de cada uno de ellos.

- **Datos iniciales**
  Debemos empezar introduciendo como datos iniciales: la función a con-

siderar, el número de derivadas, el intervalo donde discretizar y el número de puntos donde hacerlo. Se presenta la opción de cargar los datos mencionados desde un fichero. (Figura 5.5).

Figura 5.5: Cuadro de Datos Iniciales.

- **Ruido**
  Este cuadro hace referencia al nivel de ruido que se suma a los datos iniciales, teniendo la posibilidad de introducir el mismo valor de ruido para la función y sus derivadas, o un nivel de ruido distinto. (Figura 5.6). Si se elige la segunda opción, para introducir dicho ruido, se abre una nueva ventana para ello.(Figura 5.7).

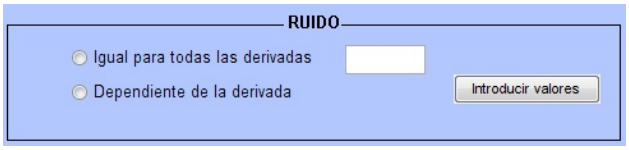

Figura 5.6: Ruido.

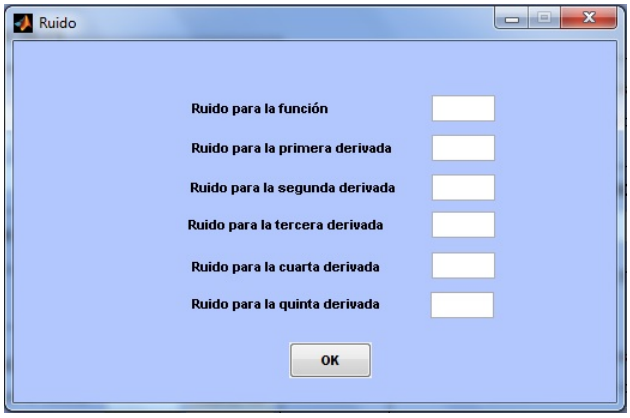

Figura 5.7: Ruido dependiente de la derivada.

- **Tratamiento en la frontera**
  Como el nombre del campo indica, se debe elegir el tipo de tratamiento que se quiere tomar en la frontera. (Figura 5.8).

Figura 5.8: Tratamiento en la frontera.

- **Número de escalas**
  Se trata de introducir los niveles de multirresolución que desciendes. (Figura 5.9).

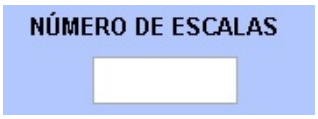

Figura 5.9: Número de escalas.

- **Truncamiento**
  En este apartado, tenemos que introducir el nivel de truncamiento. Hay dos opciones, introducir la tolerancia o el porcentaje de detalles guardados. Dentro de estas dos opciones, se ofrece la posibilidad de que el truncamiento sea igual o distinto según sean valores de función o valores de derivadas.(Figura 5.10). Pulsando el botón "Introducir valores", se abrirá una nueva ventana que permita rellenar el truncamiento, tanto por tolerancia como por porcentaje de detalles, en función de la derivada. (Figura 5.11).(Figura 5.12)

Figura 5.10: Truncamiento.

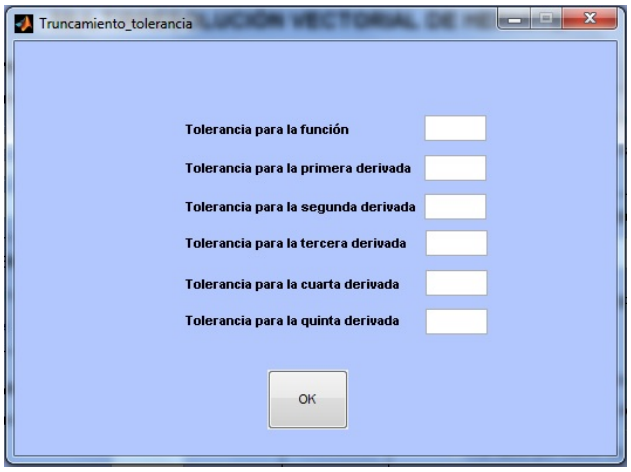

Figura 5.11: Truncamiento por tolerancia en función de la derivada.

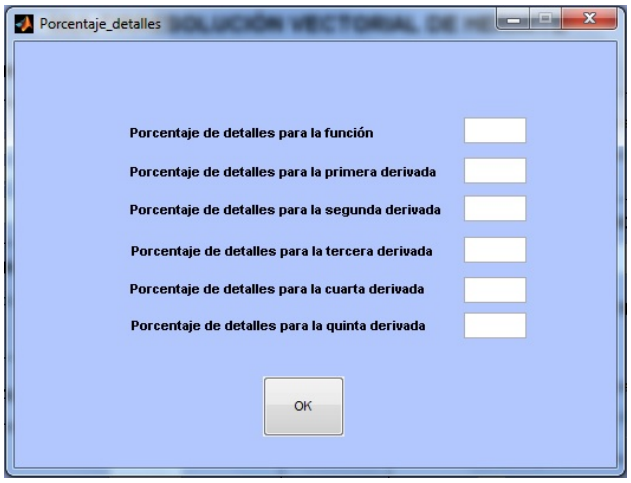

Figura 5.12: Truncamiento por porcentaje de detalles en función de la derivada.

- **Tipo de interpolación de Hermite**
  En este cuadro, se elige el conjunto de puntos usado para interpolar de manera local (Figura 5.13). Así mismo, con el botón "completar", se especifica cuantas derivadas se consideran sobre cada punto.(Figura 5.14).

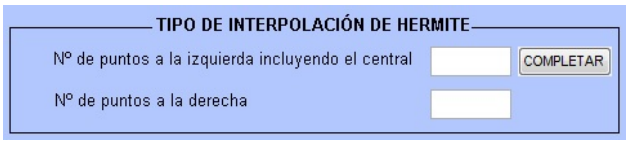

Figura 5.13: Tipo de interpolación de Hermite.

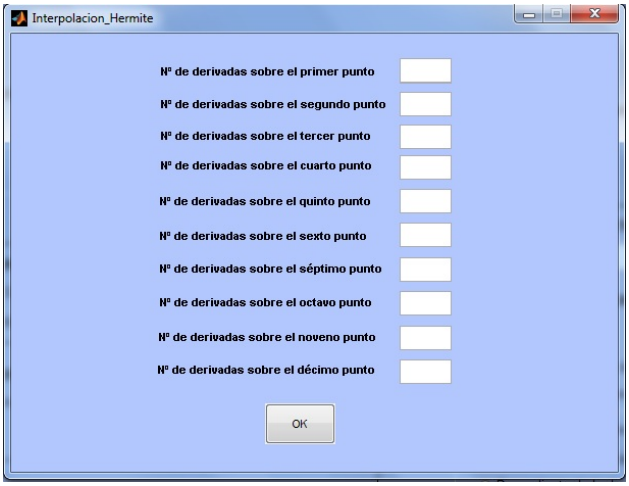

Figura 5.14: Número de derivadas sobre cada punto.

- **Botones de la Interfaz** Este conjunto de botones, ofrece la posibilidad de llevar a cabo una serie de operaciones que se describen a continuación.(Figura 5.15).

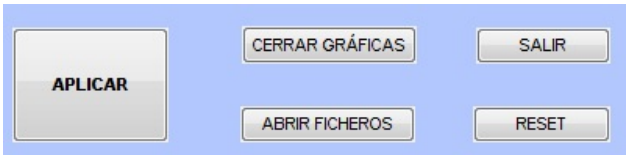

Figura 5.15: Botones de la interfaz.

  - **RESET:** resetea todos los campos de la interfaz a sus valores de inicio, además, se cierran todas las gráficas que estén abiertas.

  - **CERRAR GRÁFICAS:** se cierran todas las gráficas abiertas. A diferencia de "RESET" no afecta a los valores de los campos de la interfaz.

- **SALIR:** se sale del programa automáticamente.

- **APLICAR:** ejecuta el programa y muestra los resultados obtenidos.

- **ABRIR FICHEROS:** permite abrir en un fichero aparte, los resultados obtenidos al ejecutar el programa.

5.3. Ejemplo de uso de la Interfaz Gráfica

A continuación se explica un ejemplo práctico del uso de la interfaz. Vamos completando los campos secuencialmente como se ha descrito en el apartado anterior. De esta manera, una de las posibles configuraciones podría ser la que se contempla en la Figura 5.16.

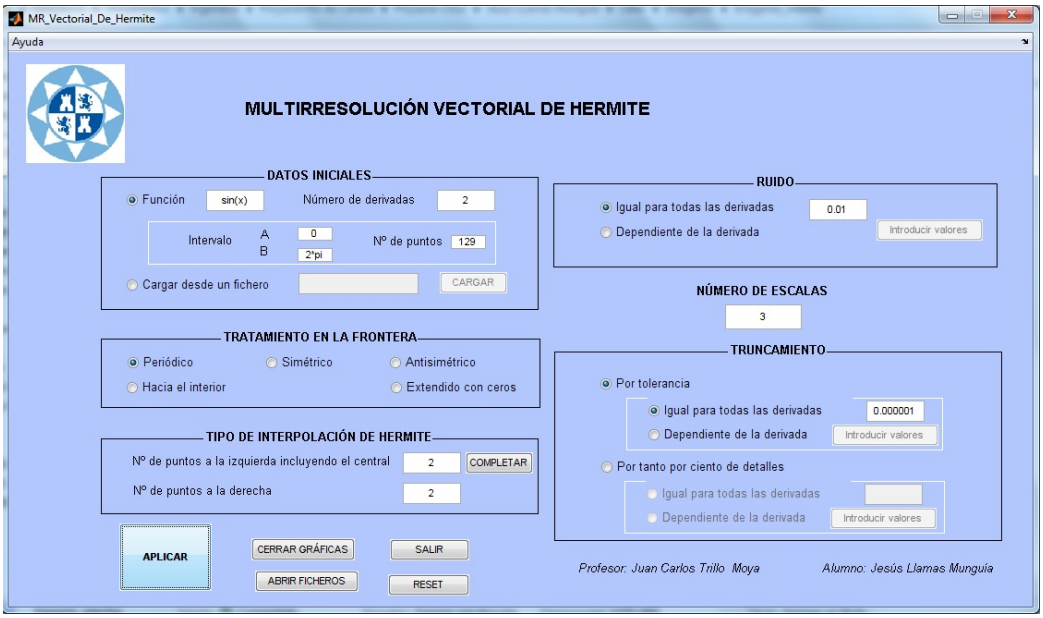

Figura 5.16: Ejemplo de configuración de la interfaz.

Una vez rellenados todos los campos, le damos al botón APLICAR y obtenemos una representación de todos los puntos interpolados, con los datos originales, los datos aproximados y los datos con ruido de la función y de las derivadas que hayamos puesto, en este caso 1 derivada (Figura 5.17), (Figura 5.18).

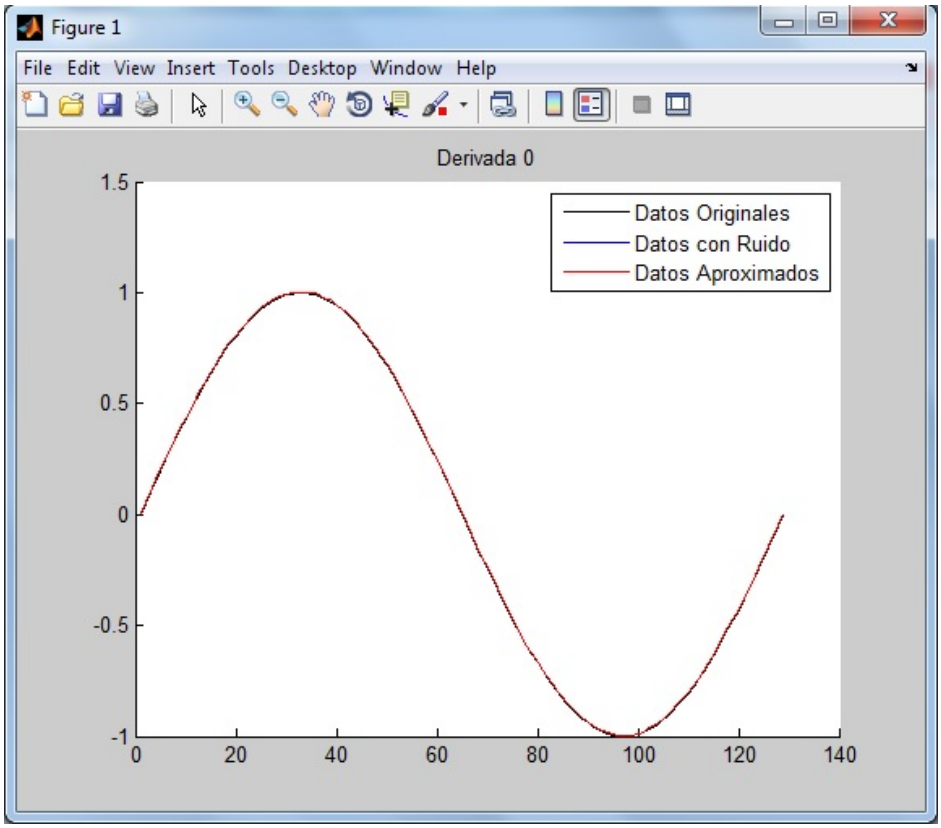

Figura 5.17: Gráfica de datos de la función.

Por último, presionando el botón ABRIR FICHEROS, obtenemos un fichero que contiene en primer lugar el porcentaje de detalles que no son cero y en segundo lugar, los errores en cada derivada y los errores medio y máximo, según la norma 1, norma 2 y norma Infinito (Figura 5.19).

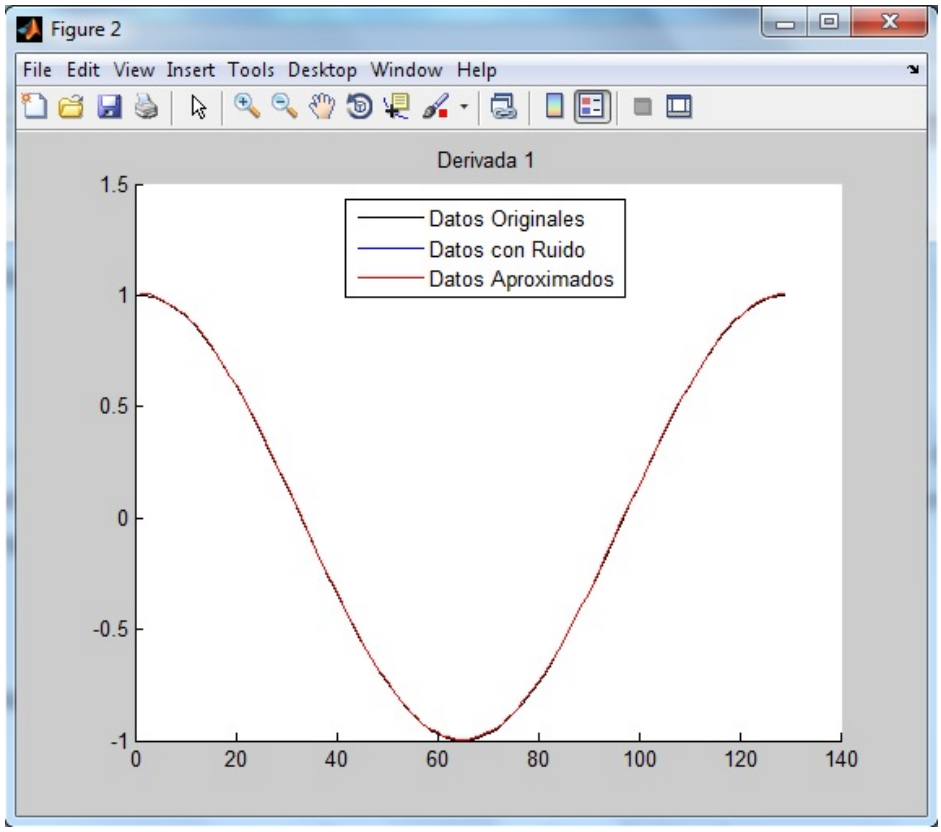

Figura 5.18: Gráfica de datos de la primera derivada.

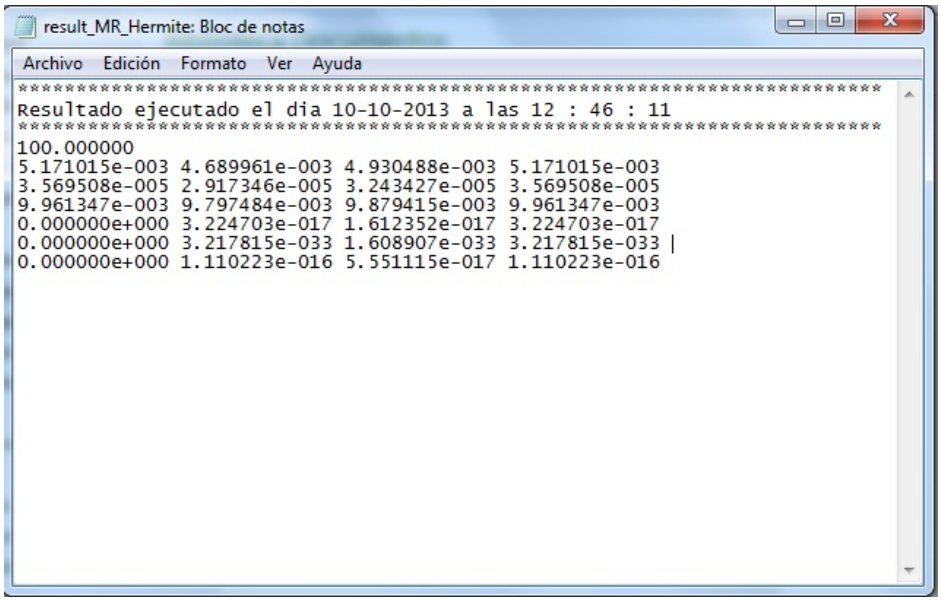

Figura 5.19: Fichero de datos obtenidos.

Capítulo 6

Experimentos Numéricos.

En este capítulo vamos a aplicar los algoritmos estudiados a algunos ejemplos numéricos.

6.1. Interfaz.

En el capítulo anterior hemos visto el funcionamiento de la interfaz gráfica desarrollada. Vamos a ver ahora varios ejemplos de su utilización. Queremos comprobar los diferentes errores que se producen a la hora de llevar a cabo la multirresolución vectorial, en función del número de puntos que se tomen, el numero de derivadas que usemos, el nivel de ruido introducido o el nivel de truncamiento utilizado.

En primer lugar, vamos a analizar las gráficas que nos proporciona el programa cuando introducimos los parámetros que podemos ver en la Figura 6.1. Trabajamos con la función $f(x) = \sin x$ en el intervalo $[0, 2\pi]$ discretizada con 129 puntos. Utilizamos como reconstrucción Hermite de dos puntos, con información sobre la función, la primera y la segunda derivada en cada punto. En este primer experimento no introducimos ruido. Utilizamos un truncamiento que guarda un 10 % de los detalles tanto para la función como para sus derivadas.

Obtenemos las gráficas de las Figuras 6.2, 6.3 y 6.4, que corresponden a la función, a la primera derivada y a la segunda derivada. Los resultados numéricos los podemos ver en la Figura 6.5. Se puede ver, como al no introducir ruido, guardando tan solo un 10 % de los detalles, la aproximación es buena para todas las derivadas.

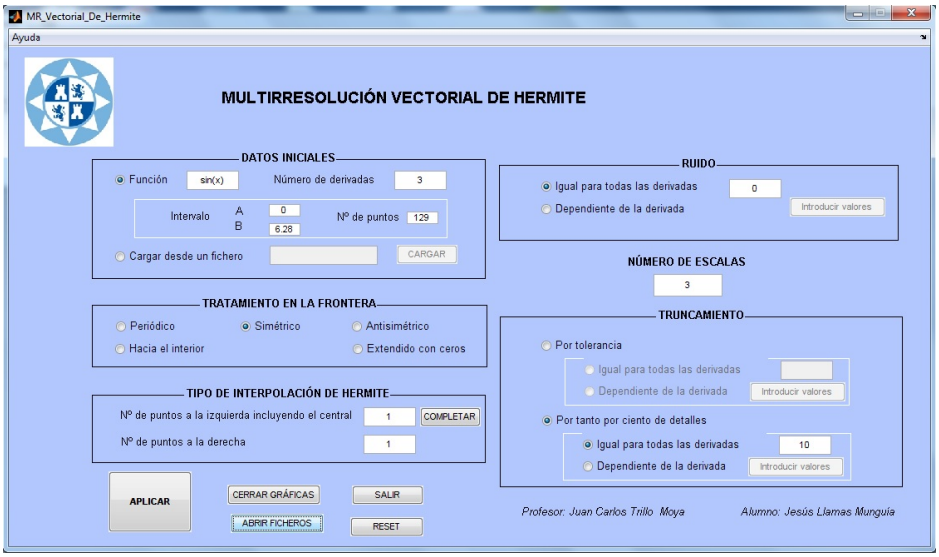

Figura 6.1: Datos interfaz gráfica. Experimento 1.

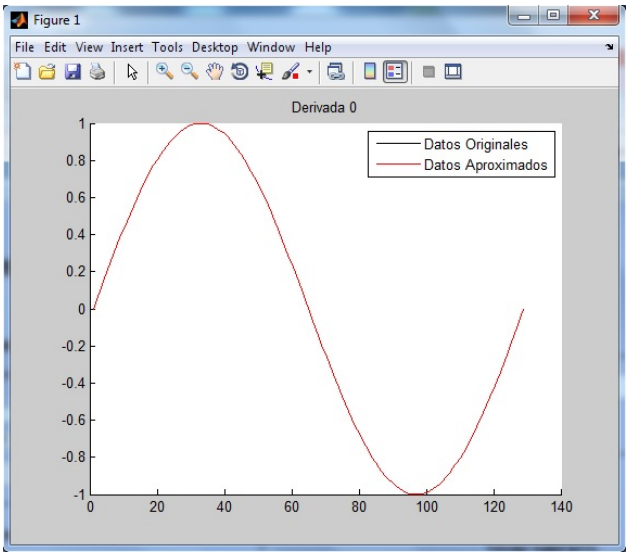

Figura 6.2: Función original y aproximada. Experimento 1: Hermite con 2 puntos, considerando función, primera y segunda derivada. Guardamos un 10 % de los detalles tanto en función como en cada derivada.

Ahora vamos a ver, como al introducir ruido, el porcentaje de detalles que nos debemos guardar para que los errores salgan aceptables, es mucho mayor. En el segundo experimento, se han introducido los datos que se muestran en la Figura 6.6, guardándonos en este caso un 70 % de detalles en la función y

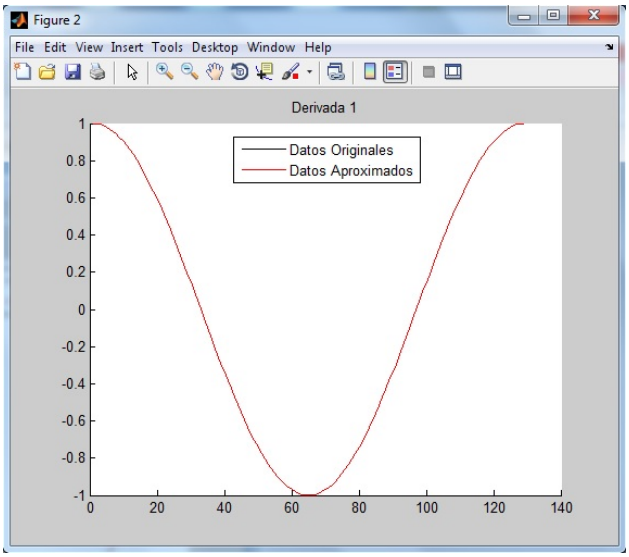

Figura 6.3: Primera derivada original y aproximada. Experimento 1: Hermite con 2 puntos, considerando función, primera y segunda derivada. Guardamos un 10 % de los detalles tanto en función como en cada derivada.

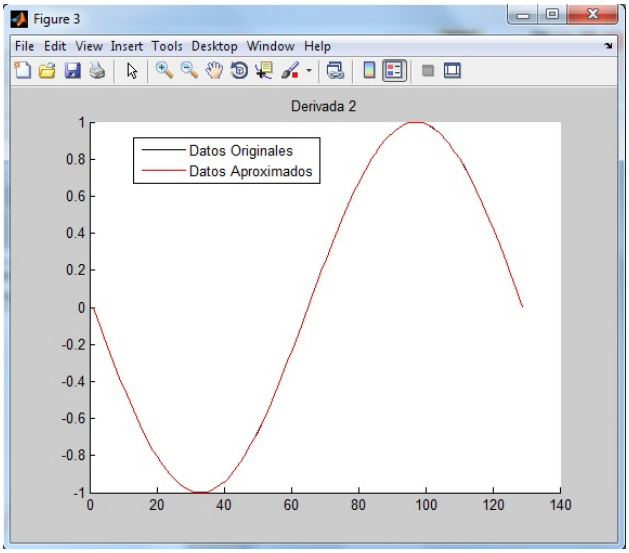

Figura 6.4: Segunda derivada original y aproximada. Experimento 1: Hermite con 2 puntos, considerando función, primera y segunda derivada. Guardamos un 10 % de los detalles tanto en función como en cada derivada.

un 100 % de detalles en la primera y segunda derivada.

Obtenemos las gráficas de las Figuras 6.7, 6.8 y 6.9, que corresponden

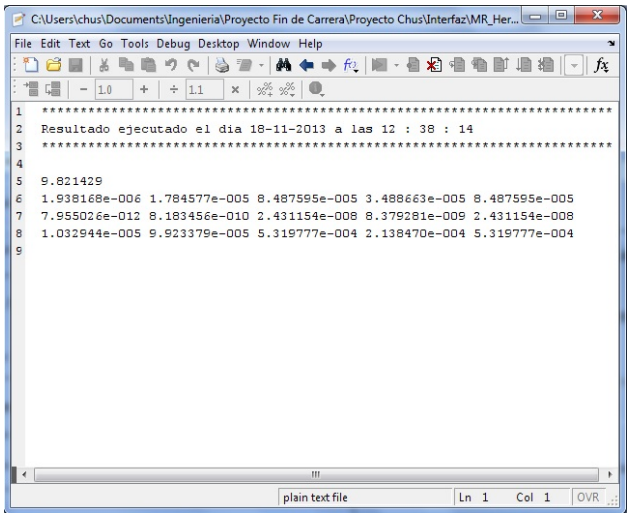

Figura 6.5: Resultados numéricos. Experimento 1: Hermite con 2 puntos, considerando función, primera y segunda derivada. Guardamos un 10 % de los detalles tanto en función como en cada derivada.

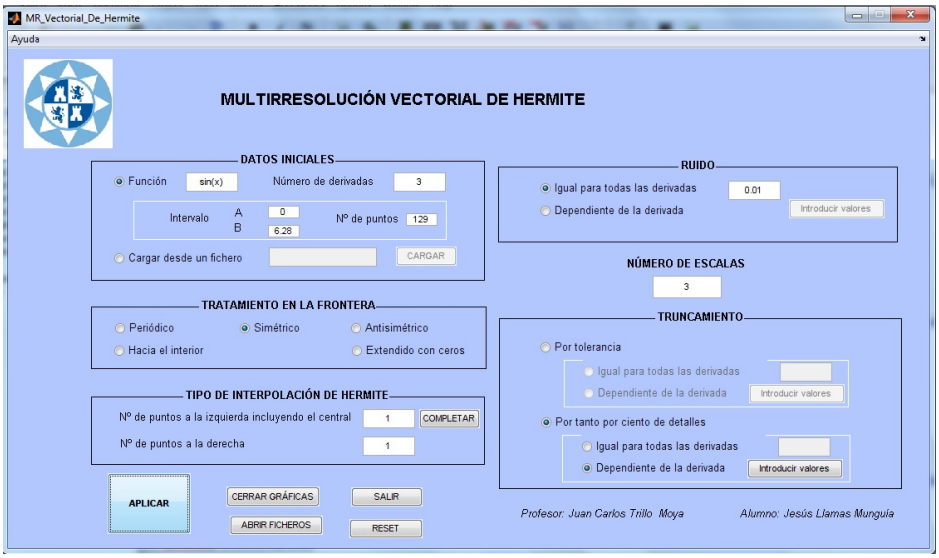

Figura 6.6: Datos interfaz gráfica. Experimento 2.

a la función, a la primera derivada y a la segunda derivada. Los resultados numéricos los podemos ver en la Figura 6.10. Observamos como un ruido pequeño de magnitud 0,01 afecta mucho en los errores cometidos y en la tasa de compresión. Los datos iniciales gráficamente parecen los mismos, al haber incluido ruido de poca intensidad, pero como ya hemos explicado, no lo son.

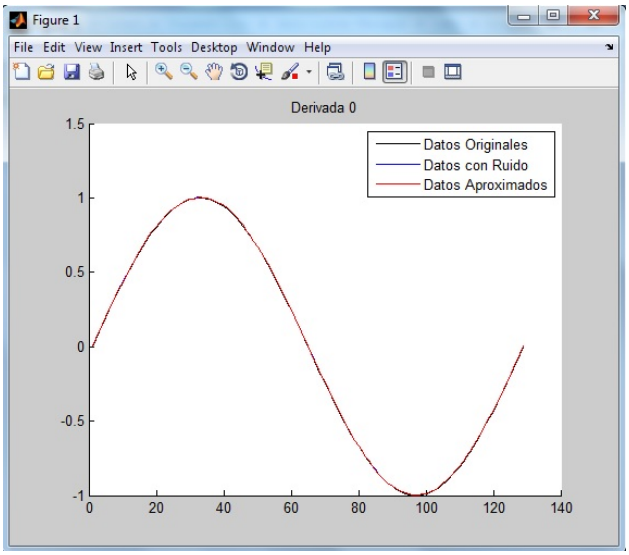

Figura 6.7: Función original,aproximada y con ruido. Experimento 2: Hermite con 2 puntos, considerando función, primera y segunda derivada. Guardamos un 70 % de los detalles en la función y un 100 % en cada derivada.

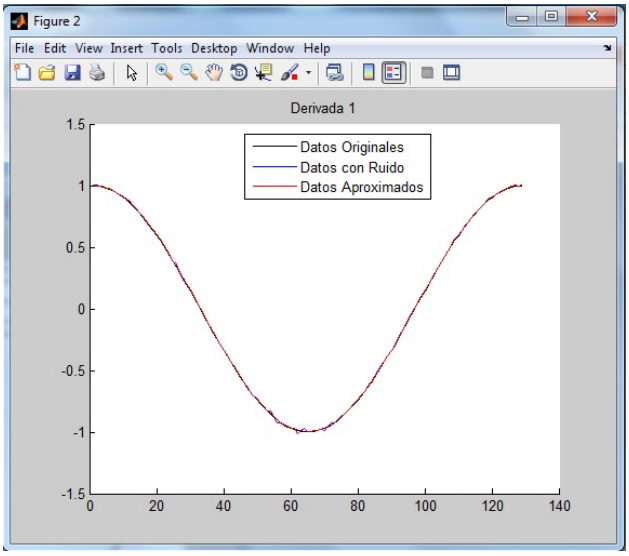

Figura 6.8: Primera derivada original, aproximada y con ruido. Experimento 2: Hermite con 2 puntos, considerando función, primera y segunda derivada. Guardamos un 70 % de los detalles en la función y un 100 % en cada derivada.

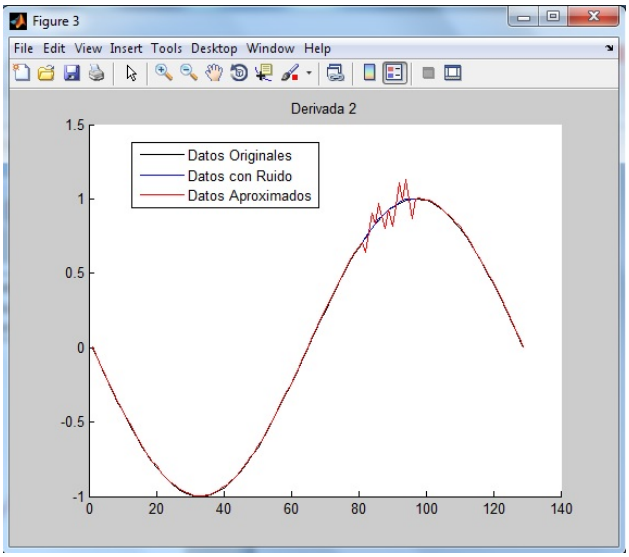

Figura 6.9: Segunda derivada original,aproximada y con ruido. Experimento 2: Hermite con 2 puntos, considerando función, primera y segunda derivada. Guardamos un 70 % de los detalles en la función y un 100 % en cada derivada.

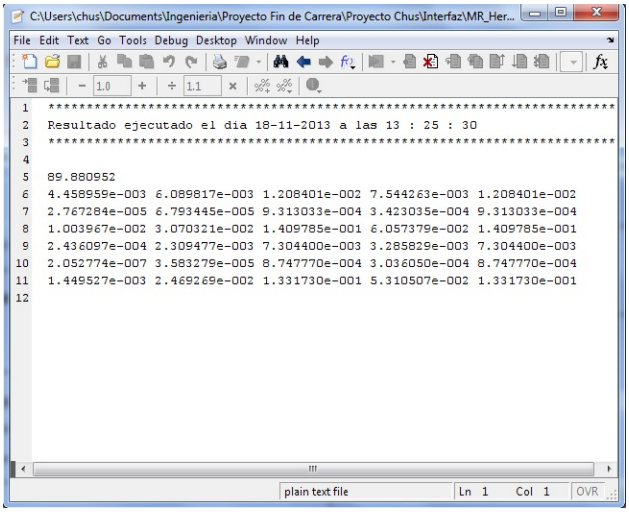

Figura 6.10: Resultados numéricos. Experimento 2: Hermite con 2 puntos, considerando función, primera y segunda derivada. Guardamos un 70 % de los detalles en la función y un 100 % en cada derivada.

A continuación, vamos a analizar las gráficas que nos proporciona el programa cuando introducimos los parámetros que podemos ver en la Figura

6.11. Trabajamos con la función $f(x) = \sin x$ en el intervalo $[0, 2\pi]$ discretizada con 129 puntos, exactamente igual que en los experimentos anteriores, pero esta vez utilizamos como reconstrucción Hermite de cuatro puntos, con información sobre la función y la primera en cada punto. Volvemos a hacerlo tanto sin ruido como con ruido. Como en los dos primeros experimentos, vuelve a ser necesario guardarse diferente porcentaje de detalles.

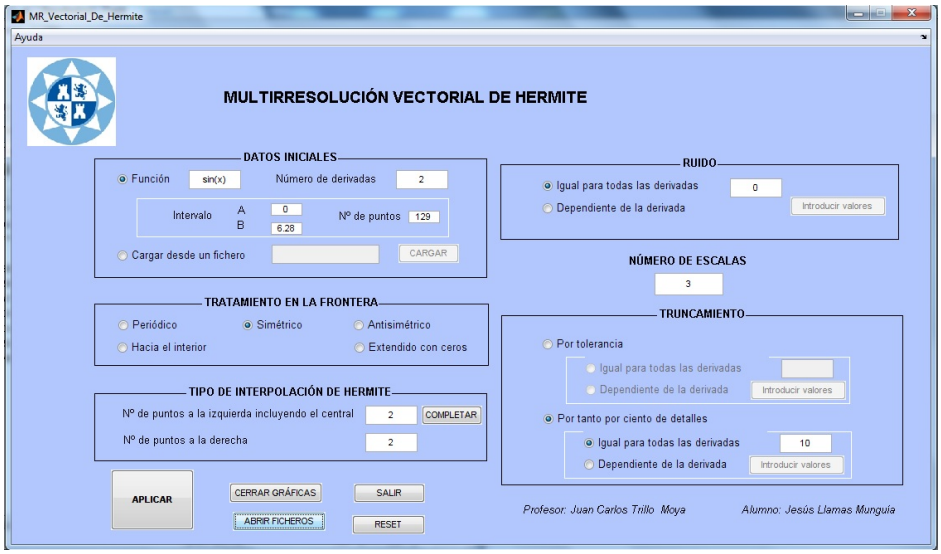

Figura 6.11: Datos interfaz gráfica. Experimento 3.

En este caso, las gráficas obtenidas son las de las Figuras 6.12 y 6.13, que corresponden a función y primera derivada. Los resultados numéricos los vemos en la Figura 6.14.

Por último, repetimos el experimento 3, añadiendo 0,01 de ruido. En este caso, guardando un 70 % de detalles en la función y en la primera derivada, los errores son aceptables. Cuantos más detalles nos guardamos, los resultados obtenidos son mejores.

Obtenemos las gráficas de las Figuras 6.15 y 6.16, correspondientes a la función y a la primera derivada. En la Figura 6.17 tenemos los resultados numéricos.

En el siguiente apartado, comentamos otra batería de experimentos.

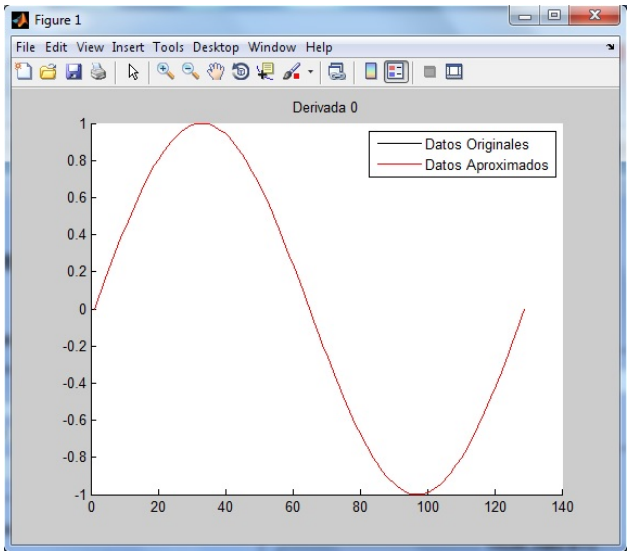

Figura 6.12: Función original y aproximada. Experimento 3: Hermite con 4 puntos, considerando función y primera derivada. Guardamos un 10 % de los detalles tanto en función como en la primera derivada.

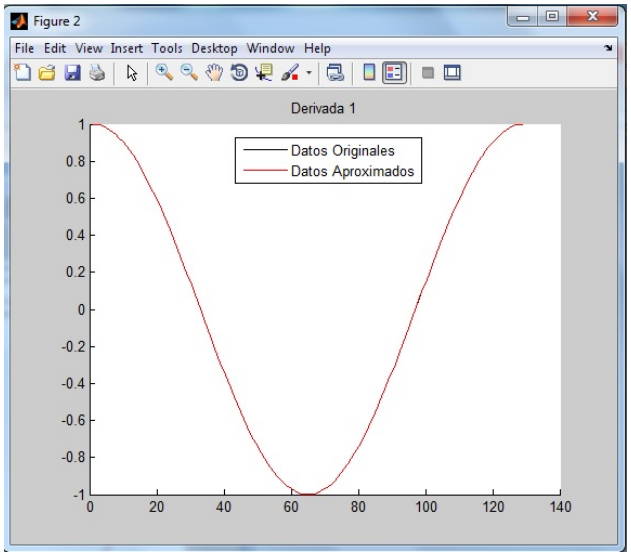

Figura 6.13: Primera derivada original y aproximada. Experimento 3: Hermite con 4 puntos, considerando función y primera derivada. Guardamos un 10 % de los detalles tanto en función como en la primera derivada.

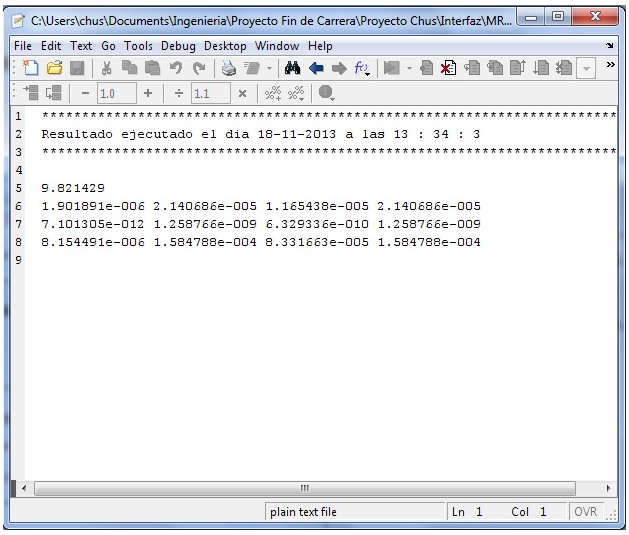

Figura 6.14: Resultados numéricos. Experimento 3: Hermite con 4 puntos, considerando función y primera derivada. Guardamos un 10 % de los detalles tanto en función como en la primera derivada.

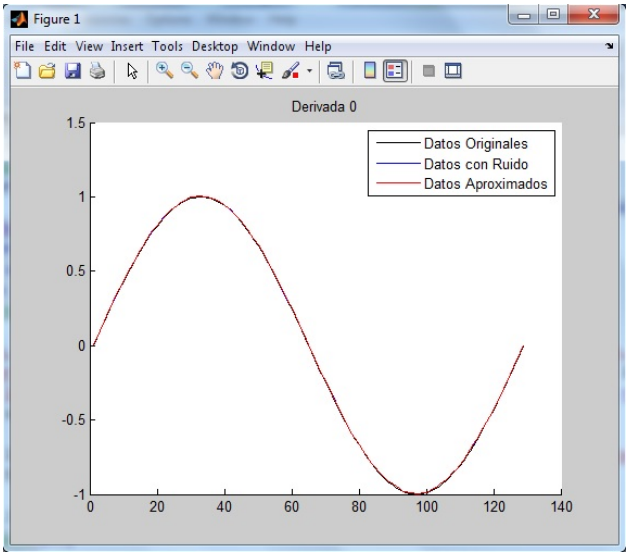

Figura 6.15: Función original y aproximada. Experimento 4: Hermite con 4 puntos, considerando función y primera derivada. Guardamos un 70 % de los detalles tanto en función como en la primera derivada.

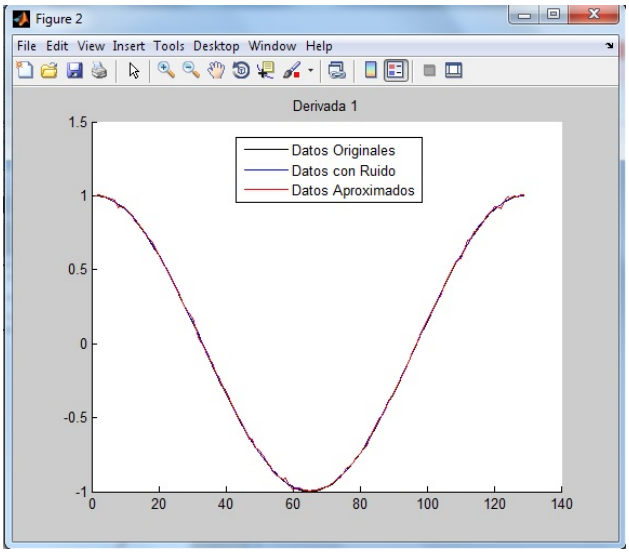

Figura 6.16: Primera derivada original y aproximada. Experimento 4: Hermite con 4 puntos, considerando función y primera derivada. Guardamos un 70 % de los detalles tanto en función como en la primera derivada.

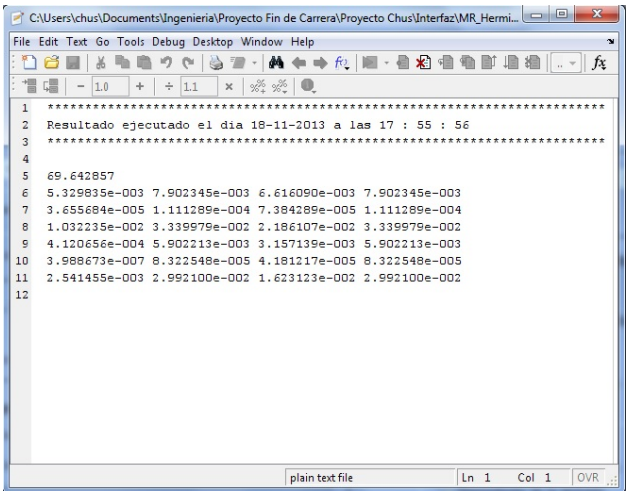

Figura 6.17: Resultados numéricos. Experimento 4: Hermite con 4 puntos, considerando función y primera derivada. Guardamos un 70 % de los detalles tanto en función como en la primera derivada.

6.2. Aproximando derivadas.

En esta sección vamos a analizar la diferencia de aplicar la multirresolución escalar a los valores puntuales de una función o la multirresolución vectorial a dichos valores de la función y a los valores aproximados de su derivada en los mismos puntos, tal y como se ha comentado en el apartado (3.3). Se comparan los dos métodos tanto para una función suave sinusoidal como para una función discontinua sinusoidal a trozos.

Los cuatro casos se han llevado a cabo con una compresión del 60 %, sin ruido y descendiendo 4 niveles de multirresolución. Se muestra para cada ejecución una imagen de la función original, su reconstrucción, los detalles guardados de la función, los detalles guardados de la primera derivada y un resumen de los resultados obtenidos.

6.2.1. Función suave sin aproximar la derivada.

Empezamos con una onda completa de la función seno como función test. Se trata de una función suave, con la cual ambos métodos deberían funcionar correctamente. Primeramente vamos a aplicar la multirresolución escalar a la discretización de la función con 129 puntos. La gráfica de la función puede verse en la Figura 6.18. En la Figura 6.19 puede verse la reconstrucción obtenida usando como operador de predicción la interpolación segmentaria de Lagrange de 4 puntos centrados. También puede verse la localización de los detalles guardados en la Figura 6.20. Y en la Figura 6.21 ofrecemos los resultados numéricos obtenidos en cuanto a los errores cometidos en la decodificación.

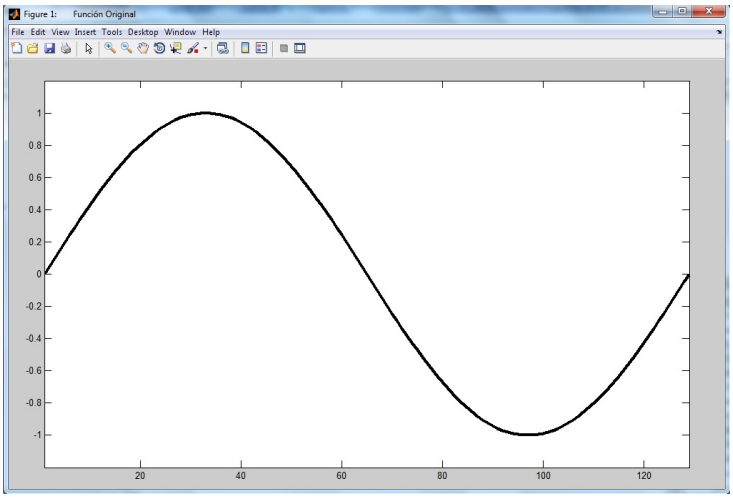

Figura 6.18: Función seno original.

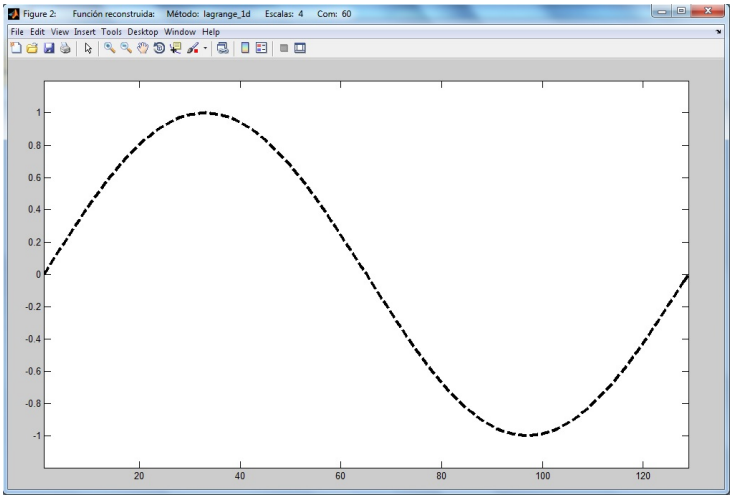

Figura 6.19: Reconstrucción de la función seno utilizando multirresolución basada en interpolación centrada de Lagrange de 4 puntos, guardando el 60 % de los detalles.

6.2.2. Función suave aproximando la primera derivada.

En este apartado vamos a aproximar los valores de la derivada en los mismos puntos en los que tenemos discretizada la función. Y posteriormente aplicaremos multirresolución vectorial de Hermite compacta con 2 puntos, y usando función y primera derivada.

En la Figura 6.22 vemos como la reconstrucción también es gráficamente buena en este caso. Los detalles guardados de la función y de la derivada,

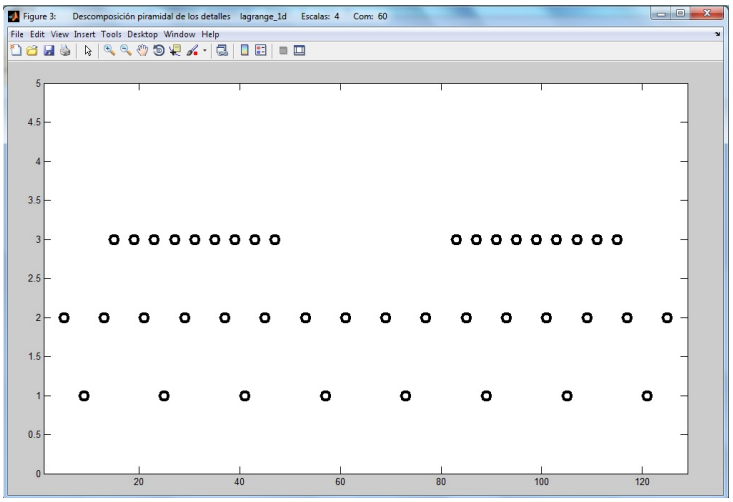

Figura 6.20: Detalles guardados de la función seno, utilizando multirresolución basada en interpolación centrada de Lagrange de 4 puntos, y guardando el 60 % de los detalles.

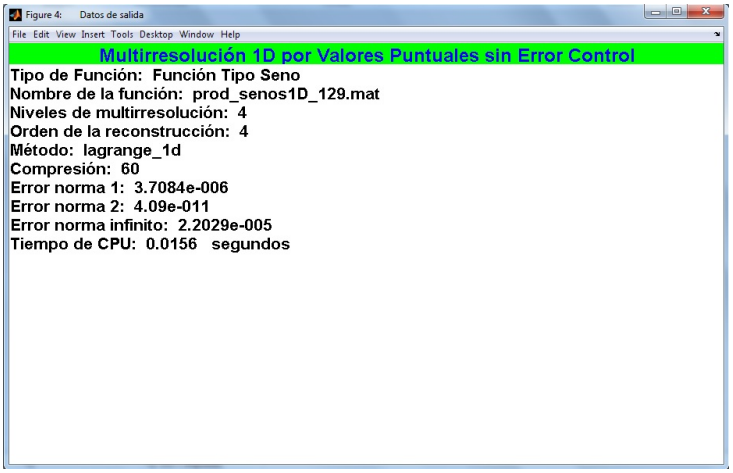

Figura 6.21: Resultados utilizando multirresolución basada en interpolación centrada de Lagrange de 4 puntos, guardando el 60 % de los detalles.

así como su localización, se ven en la Figura 6.23 y en la Figura 6.24 respectivamente. A continuación aparecen los errores cometidos en la Figura 6.25.

Se puede ver como los errores son ligeramente más pequeños en el caso de aproximar la primera derivada. Esto es debido a que, al guardarnos detalles

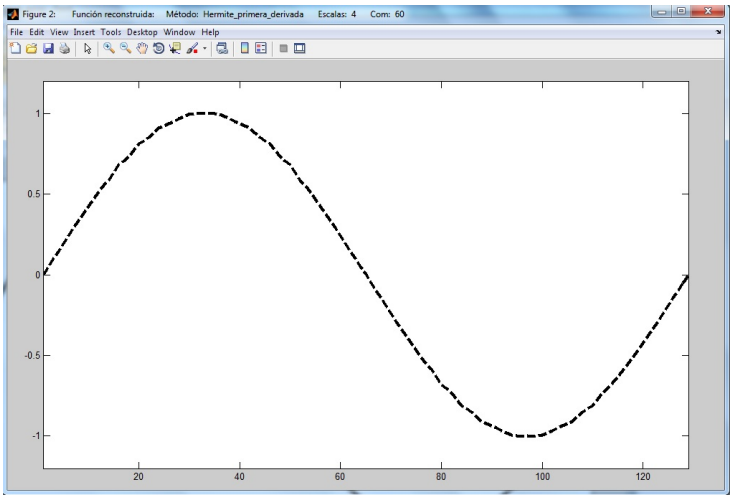

Figura 6.22: Reconstrucción de la función utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada, guardando el 60 % de los detalles totales.

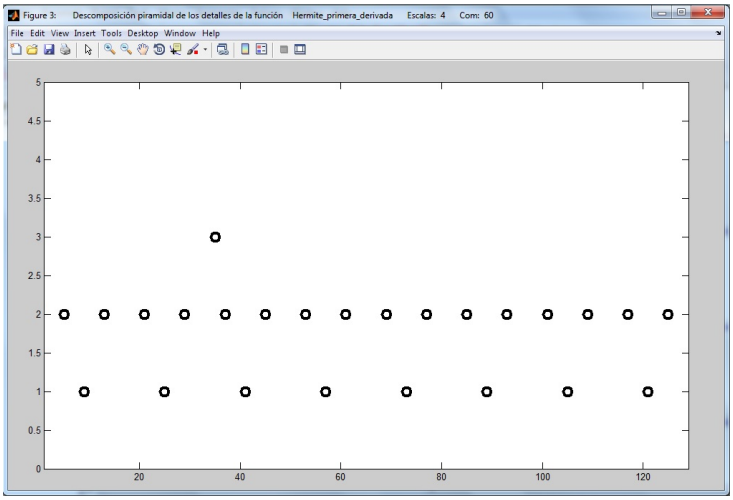

Figura 6.23: Detalles guardados de la función utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada, guardando el 60 % de los detalles totales.

en la función y en la derivada, nos quedamos con el doble de datos. Aún así, son prácticamente del mismo orden, porque al aproximar la derivada estamos introduciendo un error. Más adelante, vamos a ver los resultados que se obtienen guardándonos la mitad de los datos.

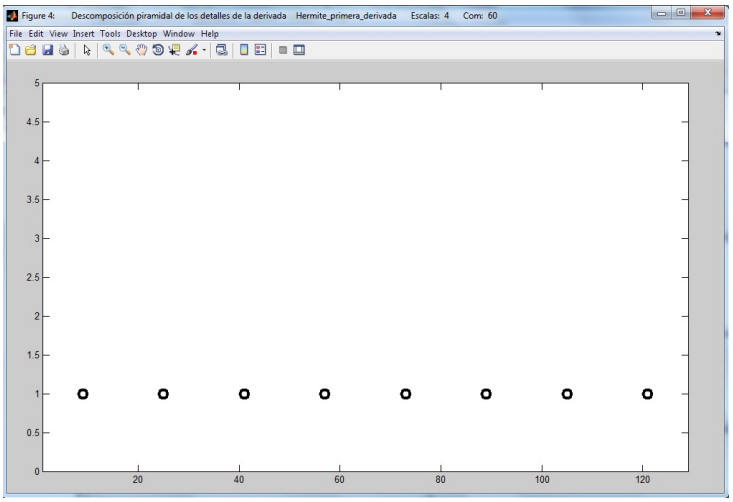

Figura 6.24: Detalles guardados de la primera derivada utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada, guardando el 60 % de los detalles totales.

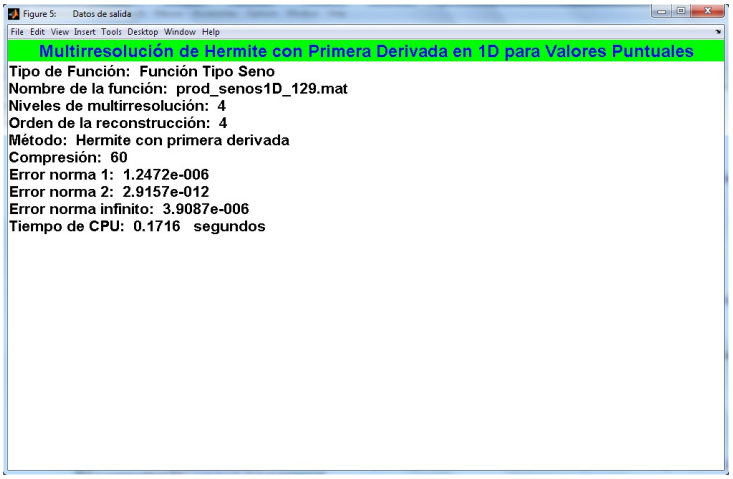

Figura 6.25: Resultados utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada, guardando el 60 % de los detalles totales.

6.2.3. Función discontinua sin aproximar la derivada.

A continuación analizamos el caso de una función discontinua, aunque sabemos que en este caso ambos algoritmos al ser inherentemente lineales, harán una mala aproximación. Por un lado la multirresolución basada en Lagrange con 4 puntos, tendrá 3 intervalos afectados por la discontinuidad por nivel de reconstrucción. Por otro lado la reconstrucción compacta de

Hermite usando 2 puntos sólo cogerá la discontinuidad en 1 intervalo. Lo que ocurre es que en este caso pretendemos aproximar las derivadas, y para hacerlo también cruzamos la discontinuidad en más de 1 intervalo.

En la Figura 6.26 vemos la función original utilizada. Se trata en este caso de una función sinusoidal a trozos. La reconstrucción aparece en la Figura 6.27. Los detalles guardados de la función se ven en la Figura 6.28, y en la Figura 6.29 aparecen los errores cometidos.

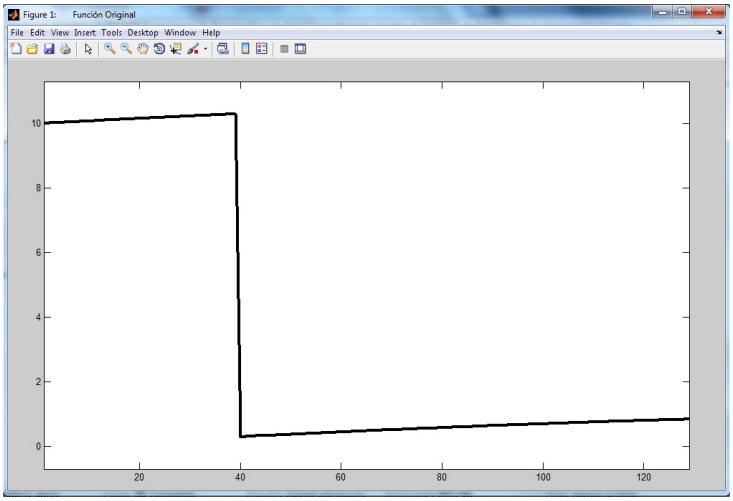

Figura 6.26: Función original sinusoidal a trozos, con discontinuidad de salto.

6.2.4. Función discontinua aproximando la primera derivada.

En este apartado vamos a ejecutar los algoritmos con la misma función discontinua, pero aproximando la primera derivada para comparar los resultados.

Las gráficas de la función reconstruida, de los detalles guardados tanto en la función como en la derivada, así como los errores cometidos en la aproximación aparecen en las Figuras 6.30, 6.31, 6.32 y 6.33 respectivamente.

En el caso de una función con discontinuidad, vemos que los errores son mayores aproximando la primera derivada que sin aproximarla. La diferencia entre estos errores, es bastante mayor que la que había en el caso de la función suave. Esto puede deberse, a que al aproximar la primera derivada en las zonas con discontinuidad, el error introducido es mayor.

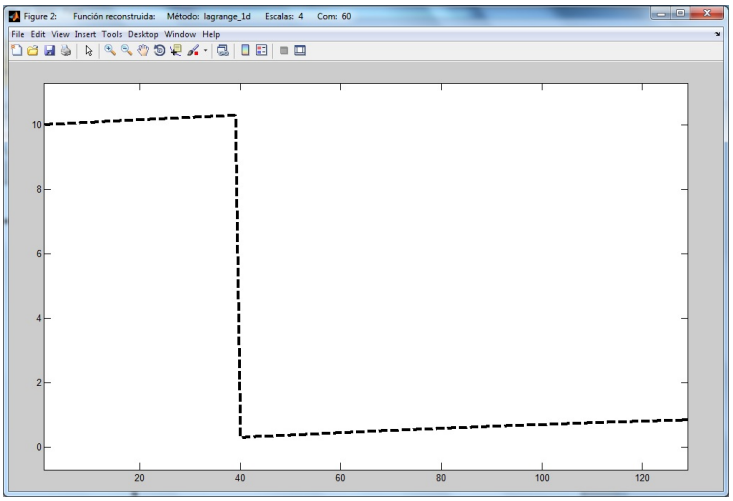

Figura 6.27: Reconstrucción de la función utilizando multirresolución basada en interpolación centrada de Lagrange de 4 puntos, guardando el 60 % de los detalles.

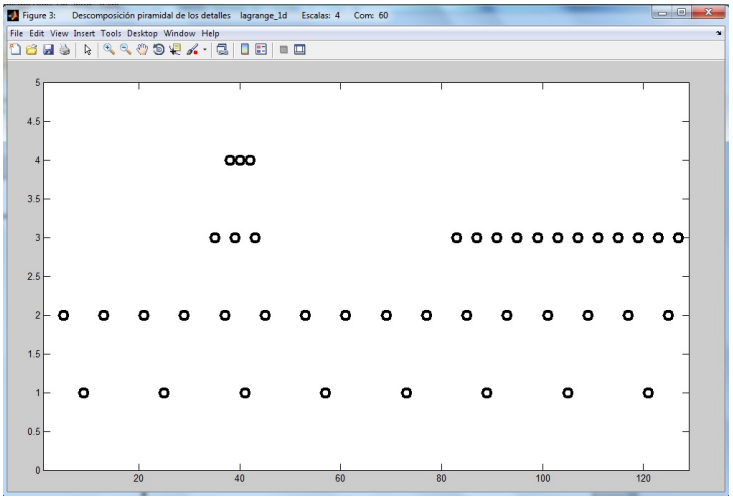

Figura 6.28: Detalles guardados de la función utilizando multirresolución basada en interpolación centrada de Lagrange de 4 puntos, guardando el 60 % de los detalles.

Mencionar también que los errores son del mismo orden en la función discontinua que en la suave. Esto ocurre debido a que la compresión es pequeña y por tanto se guarda los detalles alrededor de la discontinuidad y aproxima bien. Si comprimiéramos más, los errores en la función discontinua se dispararían, mientras que los de la función suave permanecerían dentro de unos

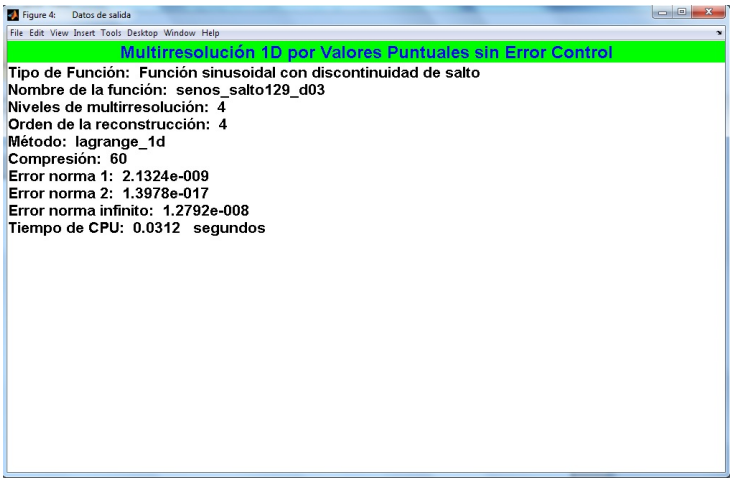

Figura 6.29: Resultados utilizando multirresolución basada en interpolación centrada de Lagrange de 4 puntos, guardando el 60 % de los detalles.

límites aceptables.

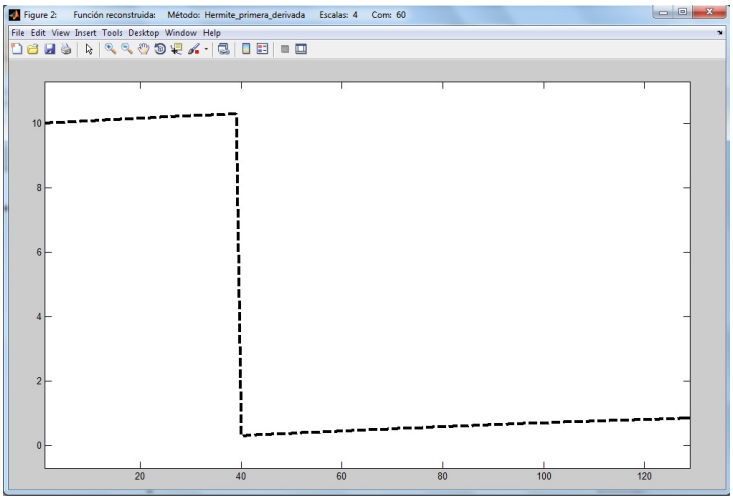

Figura 6.30: Reconstrucción de la función utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada, guardando el 60 % de los detalles totales.

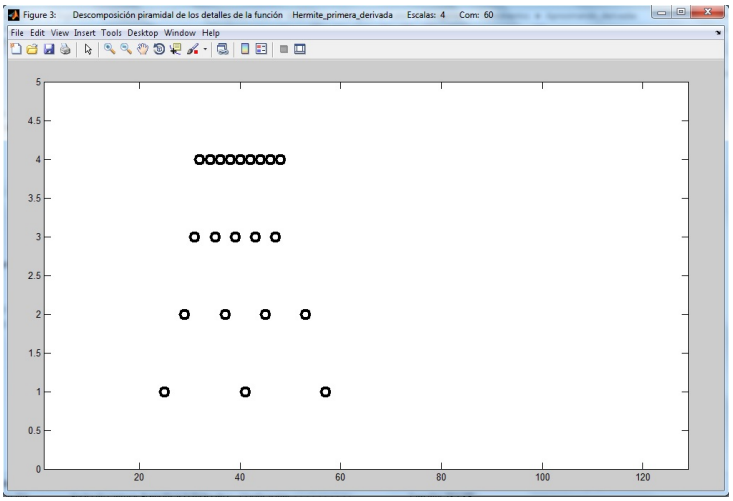

Figura 6.31: Detalles guardados de la función utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada, guardando el 60 % de los detalles totales.

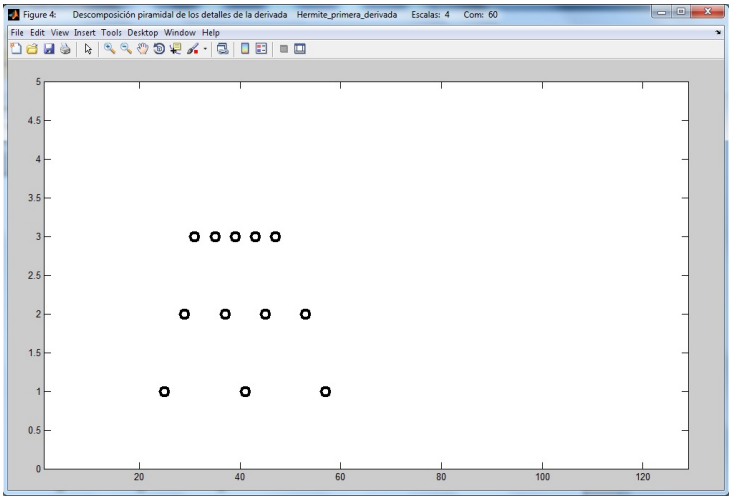

Figura 6.32: Detalles guardados de la derivada utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada, guardando el 60 % de los detalles totales.

6.2.5. Función suave aproximando la primera derivada en la mitad de los puntos.

Como se ha comentado en el punto 3.3, el problema reside en que estamos doblando el número de puntos. Aproximando la primera derivada sólo en los puntos pares, obtendríamos los siguientes resultados tanto para la función

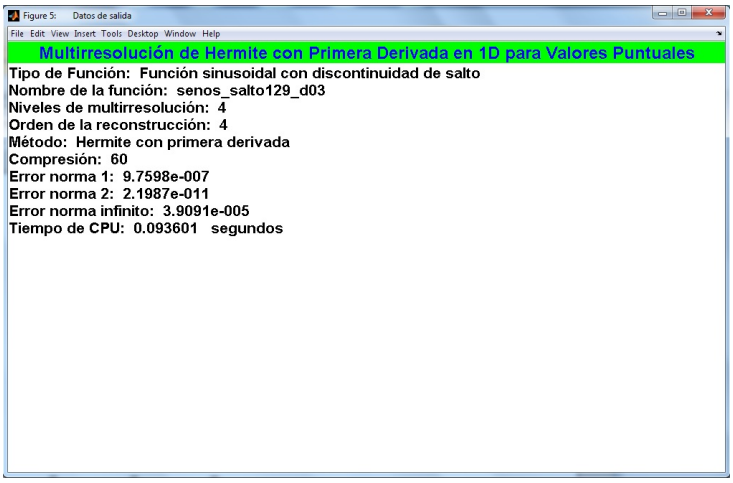

Figura 6.33: Resultados utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada, guardando el 60 % de los detalles totales.

suave como para la función discontinua. En el caso de la función suave, en la Figura 6.34 podemos ver la reconstrucción realizada. En las Figuras 6.35 y 6.36 se ven los detalles guardados. Y en la Figura 6.37, observamos los resultados numéricos de los errores cometidos.

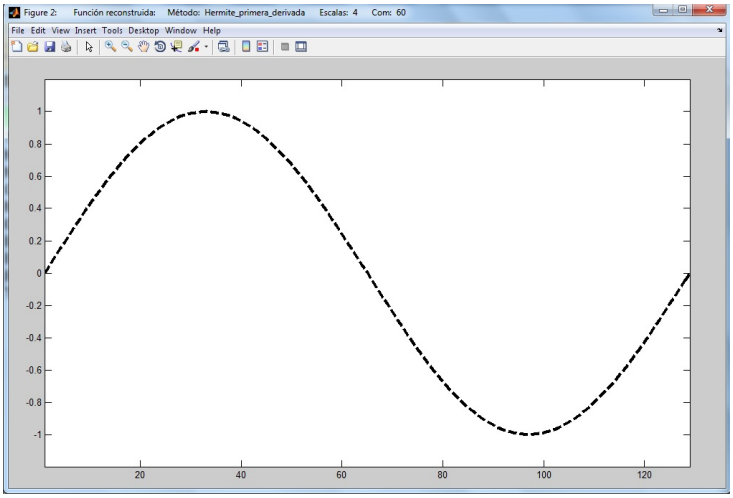

Figura 6.34: Reconstrucción de la función utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada aproximada en la mitad de puntos, guardando el 60 % de los detalles totales.

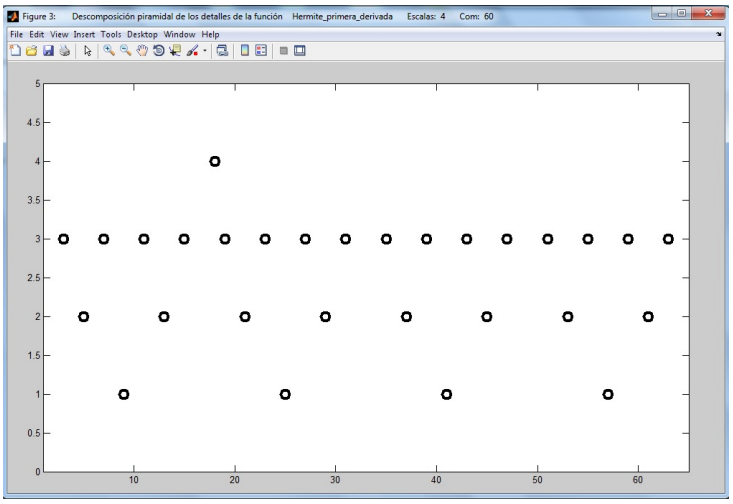

Figura 6.35: Detalles guardados de la función utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada aproximada en la mitad de puntos, guardando el 60 % de los detalles totales.

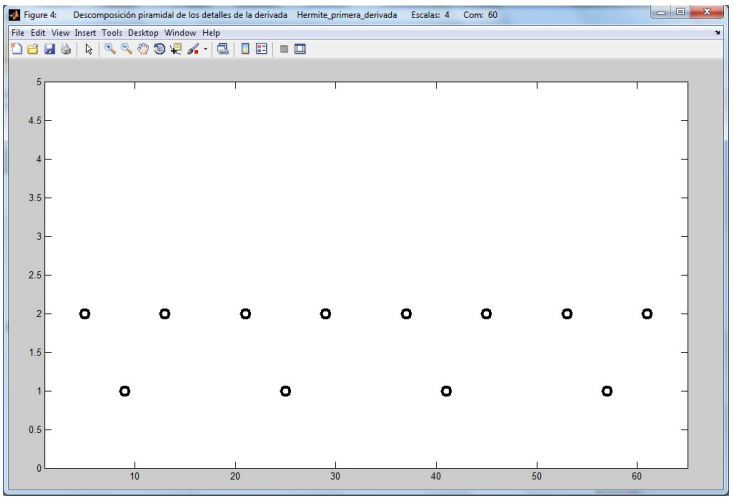

Figura 6.36: Detalles guardados de la primera derivada utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada aproximada en la mitad de puntos, guardando el 60 % de los detalles totales.

6.2.6. Función discontinua aproximando la primera derivada en la mitad de los puntos.

Igualmente en el caso de la función discontinua podemos visualizar la reconstrucción en la Figura 6.38, los detalles guardados en las Figuras 6.39

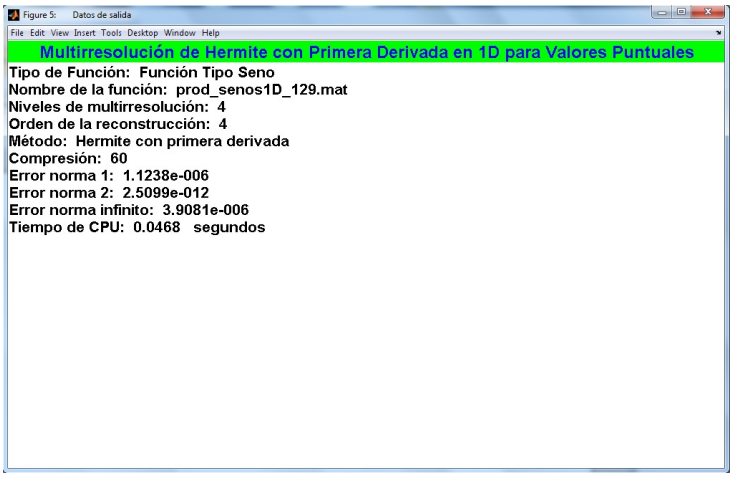

Figura 6.37: Resultados utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada aproximada en la mitad de puntos, guardando el 60 % de los detalles totales.

y 6.40, y los resultados numéricos en la Figura 6.41.

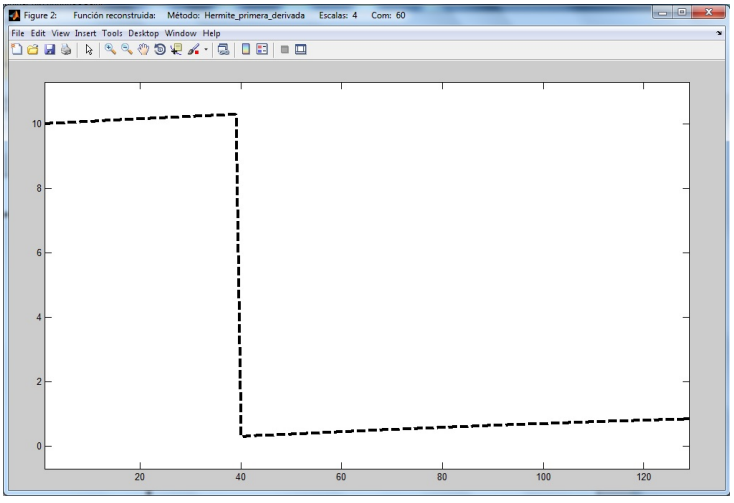

Figura 6.38: Reconstrucción de la función utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada aproximada en la mitad de puntos, guardando el 60 % de los detalles totales.

Vemos como para la función suave, los errores son del mismo orden aproximando la primera derivada en los puntos pares sólo o aproximándola para

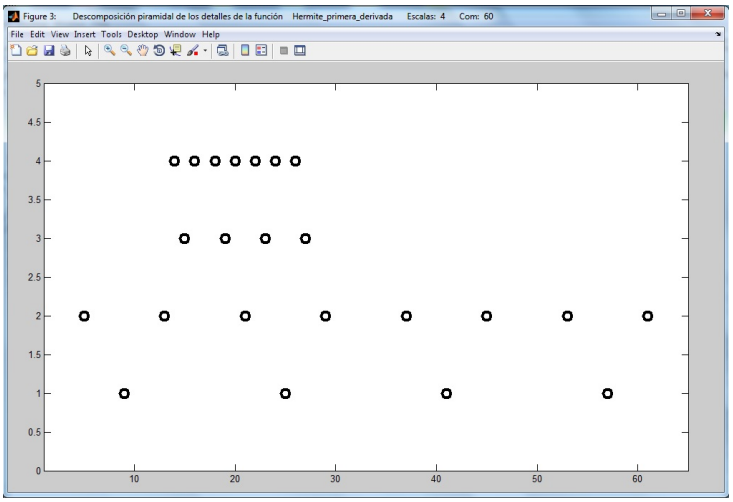

Figura 6.39: Detalles guardados de la función utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada aproximada en la mitad de puntos, guardando el 60 % de los detalles totales.

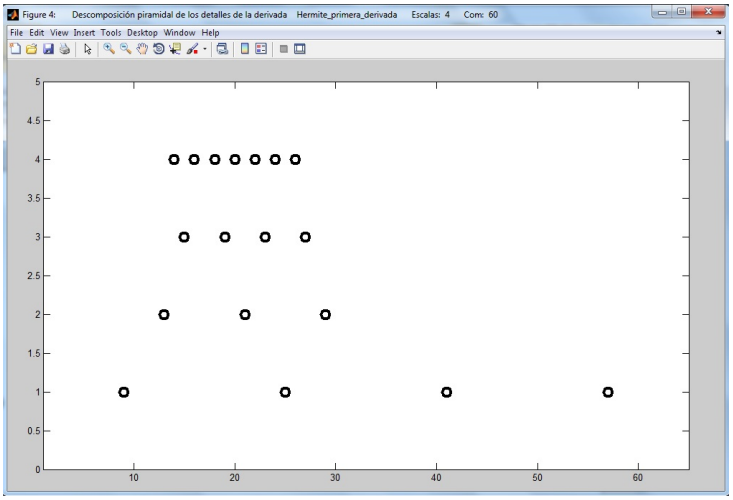

Figura 6.40: Detalles guardados de la derivada utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada aproximada en la mitad de puntos, guardando el 60 % de los detalles totales.

todos los puntos. Sin embargo, en el caso de una función discontinua, si que mejoran bastante los errores aproximando la primera derivada en los puntos pares sólo. Esto ocurre, porque debido a que nos guardamos menos valores significativos, nos podemos guardar más coeficientes de detalle con la misma compresión, y así obtener menor error. Recordamos que en el caso de discon-

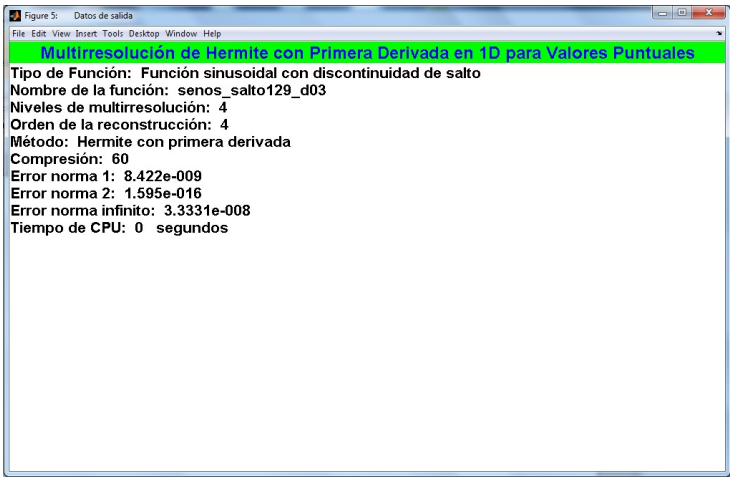

Figura 6.41: Resultados utilizando multirresolución de Hermite basada 2 puntos y en función y primera derivada aproximada en la mitad de puntos, guardando el 60 % de los detalles totales.

tinuidad es importante guardarse los detalles altos, ya que en caso contrario los errores se disparan.

Capítulo 7

Conclusiones.

En este proyecto hemos realizado un estudio teórico práctico de los algoritmos de Multirresolución de Harten en el caso vectorial, en el cual consideramos aparte de los valores puntuales de la función, los valores puntuales de las primeras derivadas. Se ha utilizado la interpolación de Hermite como operador de predicción. Para utilizar dicha interpolación, necesitamos derivadas, es por esto por lo que nos hemos centrado en el caso de multirresolución vectorial.

Se han desarrollado los programas Matlab necesararios para poner en práctica los algoritmos de multirresolución vectorial de Hermite, cualquiera que sea la elección del número de nodos utilizados para construir la reconstrucción de manera local, y cualquiera que sea el número de derivadas considerado.

Con el fin de facilitar el uso de estos programas por parte del usuario, se ha utilizado el entorno gráfico de Matlab (GUIDE) para codificar dichos algoritmos. De esta manera, no será necesario tener amplios conocimientos del Matlab para usarlos.

Por último, hemos aplicado los algoritmos estudiados a algunos ejemplos numéricos.

Las conclusiones principales las podríamos resumir diciendo que los algoritmos de multirresolución vectorial son útiles en el caso de que se disponga tanto de la función como de las primeras derivadas. Ahora bien, si solo se dispone de la función, como hemos visto en los experimentos numéricos, no resulta eficaz utilizar la multirresolución vectorial, a menos que se trate de aproximar una función con discontinuidad y se use un método de aproximación de derivadas que evite dichas discontinuidades y además se utilice una reconstrucción de Hermite lo más compacta posible (por ejemplo la basada en dos puntos). Puesto que las aproximaciones a las derivadas que hemos usado en este proyecto son lineales, y no dependientes de los datos, se ha

podido comprobar que los resultados aproximando las primeras derivadas no mejoran los resultados de la multirresolución aplicada sólo a la función.

Es decir, que en principio, se aconseja la utilización de multirresolución vectorial de Hermite sólo en el caso de ya disponer de las derivadas y no tener que aproximarlas. Siempre y cuando se quieran conservar también los valores de las derivadas.

Bibliografía

[1] I. Ali, J.C. Trillo and S. Amat, *Point values Hermite multiresolution for non-smooth noisy signals.*, Computing , **77**(3), 223-236, (2006).

[2] F. Aràndiga and R. Donat, *Stability through synchronization in nonlinear multiscale transformations*, SIAM J. Sci. Comput., **29**(1), 265-289,(2007).

[3] S. Amat, *Nonseparable multiresolution with error control. Appl. Math. Comput.*, **145**(1), 117-132, (2003).

[4] S.Amat, *A review on the piecewise polynomial harmonic interpolation*, Appl. Num. Math., **58** (8), 1168-1185, (2008).

[5] S.Amat, F.Aràndiga, A.Cohen and R.Donat, *Tensor product multiresolution analysis with error control for compact image representation. Signal Processing*, **82**(4), 587-608, (2002).

[6] S.Amat, F.Aràndiga, A.Cohen, R.Donat, G.García and M.von Oehsen, *Data compression with ENO schemes: A case study*, Appl. Comp. Harm. Anal., **11**, 273-288, (2001).

[7] S.Amat, S.Busquier and V.F.Candela, *A polynomial approach to Piecewise Hyperbolic Method*, Int. J. Comput. Fluid Dyn., **17** (3), 205-217, (2003).

[8] S. Amat, S. Busquier, and J.C. Trillo., *Stable Biorthogonal Multiresolution Transforms. Journal of Numerical Analysis, Industrial and Applied Mathematics*, **1**(3), 229-239, (2006).

[9] S.Amat, K.Dadourian, J.Liandrat and J.C. Trillo, *On a class of nonlinear interpolatory subdivision schemes*, submitted (2009).

[10] S.Amat, R.Donat and J.C.Trillo, *On specific stability bounds for linear multiresolution schemes based on piecewise polynomial Lagrange interpolation*, Journal of Math. Anal. and Appl., **1**, 18-27 , (2009).

[11] S.Amat, R.Donat, J.Liandrat and J.C.Trillo, *Analysis of a new nonlinear subdivision scheme. Applications in image processing*, Found. Comput. Math., **6** (2), 193-225, (2006).

[12] S.Amat and J.Liandrat. *On the stability of PPH nonlinear multiresolution*, Appl. Comp. Harm. Anal., **18** (2), 198-206, (2005).

[13] A. Baeza, F. Arándiga and R. Donat *Discrete multiresolution based on Hermite interpolation: Computing derivatives.*, Comm. Nonlinear Sci., Simulation,**9**, 263-273, (2004).

[14] R. Beam and R. Warming, *Discrete multiresolution analysing using Hermite interpolation: Biorthogonal multiwavelets.*, SIAM. J. Sci. Comp., **22**, 1269-1317, (2000).

[15] T.Chan and H.M.Zhou, *ENO-wavelet transforms for piecewise smooth functions. SIAM J. Numer. Anal*, **40** (4), 1369-1404, (2002).

[16] R.L. Claypoole, G. M. Davis, W. Sweldens, R.G. Baraniuk, *Nonlinear wavelet transforms for image coding via lifting.*, IEEE Trans. Image Process, **12**(12), 1449-1459, (2003).

[17] A. Cohen, N. Dyn and B. Matei, *Quasilinear subdivision schemes with applications to ENO interpolation*, Appl. Comp. Harm. Anal., **15**, 89-116, (2003).

[18] N.Dyn, *Subdivision schemes in computer aided geometric design*, Oxford University Press, **20**(4), 36-104, (1992).

[19] A.Harten, *ENO schemes with subcell resolution.*, J. Comput. Phys., **83**, 148-184, (1989).

[20] A.Harten, *Discrete multiresolution analysis and generalized wavelets.*, J. Appl. Numer. Math., **12**, 153-192, (1993).

[21] A.Harten, *Multiresolution representation of data II.*, SIAM J. Numer. Anal., **33** (3), 1205-1256, (1996).

[22] A. Harten, B. Engquist, S.J. Osher and S.R. Chakravarthy, *Uniformly high order accurate essentially non-oscillatory schemes III.*, J. Comput. Phys., **71**, 231-303, (1987).

[23] F.Kuijt, *Convexity Preserving Interpolation - Stationary Nonlinear Subdivision and Splines.*, PhD Thesis, University of Twenty, Faculty of Mathematical Sciences, (1998).

[24] A. Marquina, *Local piecewise hyperbolic reconstruction of numerical fluxes for nonlinear scalar conservation laws.*, SIAM J. Sci. Comput., **15** (4), 892-915, (1994).

[25] S.Serna and A.Marquina, *Power ENO methods: a fifth order accurate Weighted Power ENO method*, J. Comput. Phys., **194**, 632-658, (2004).

[26] W. Sweldens, *The lifting scheme: a custum-design construction of biorthogonal wavelets, Applied and Computational Harmonic Analysis*, **3** (2), 186-200, (1996).

[27] W. Sweldens, *The lifting scheme: a construction of second generation wavelets, SIAM Journal on Mathematical Analysis*, **29** (2), 511-546, (1998).

[28] W. Sweldens and P. Schröder, *Building your own wavelets at home, Wavelets in Computers Graphics, ACM SIGGRAPH Course notes*, 15-87, (1996).

[29] J.C. Trillo, *Nonlinear multiresolution and applications in image processing*, PhD in the University of Valencia, Spain, (2007).