Reconstrucciones Lineales Y No Lineales. Y Diferentes Aplicaciones

Autor: Pascual Madrid Díaz
Director: Juan Carlos Trillo Moya

Noviembre 2011

Autor	Pascual Madrid Díaz
E-mail del Autor	pasimadrid@hotmail.es
Director	Juan Carlos Trillo Moya
E-mail del Director	jctrillo@upct.es
Codirector	
Título del PFC	Reconstrucciones Lineales Y No Lineales. Y Diferentes Aplicaciones
Descriptores	Reconstrucciones, Multirresolución, Compresión, Zoom
Resúmen	El proyecto consiste en el estudio de los operadores de reconstrucción de Lagrange y PPH ([14], [6]) de orden n. También se lleva a cabo la implementación de los programas e interfaces en Matlab y la aplicación de dichos métodos a funciones en 1D y 2D, así como a imágenes digitales, mapas cartograficos, etc.
Titulación	Ingeniero Técnico de Telecomunicaciones, Especialidad Telemática
Departamento	Matemática Aplicada y Estadística
Fecha de Presentación	Noviembre de 2011

Desearía agradecer a toda mi familia y amigos, la paciencia que han demostrado tener conmigo durante todo este tiempo, en especial a mis padres. A Conchi por apoyarme en los momentos más duros. Agradecer especialmente a Juan Carlos que durante este tiempo se ha convertido en un gran amigo. Y a Carlos, Sonia y todo el Departamento de Matemática Aplicada y Estadística por sus consejos.

Índice general

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

El trabajo presentado en esta memoria está dedicado al estudio de los operadores de reconstrucción tanto lineales, *Lagrange*, como no lineales, *PPH*. También nos centraremos en la investigación de las posibles aplicaciones de dichos operadores y en la programación en Matlab de los algoritmos necesarios para la realización del estudio.

Un problema habitual en teoría de aproximación consiste en reconstruir una función a partir de un conjunto discreto de datos. En este proyecto abordaremos este problema desde el punto de vista de valores puntuales, considerando los datos discretos como los valores que toma una función en un conjunto de puntos. De este modo la función puede ser aproximada por otra función que posea los mismos valores en dicho conjunto de puntos.

Uno de los objetivos de este estudio será comprobar como se comporta la reconstrucción lineal, independiente de los datos, de *Lagrange* al operar con una función que contiene una singularidad. El aumentar el orden de la técnica interpolatoria no resuelve el problema, ya que para este caso se ven afectados un mayor número de *stencils*(conjunto de puntos usados para construir un polinomio interpolador) por la singularidad. También se aplicará la reconstrucción no lineal, dependiente de los datos, *PPH* para comprobar que se mejoran los resultados obtenidos con *Lagrange*.

El estudio de estos conceptos viene motivado por el interés suscitado en las últimas décadas por los métodos de multirresolución y su aplicación a la compresión de imágenes digitales. Resulta que un ingrediente fundamental de los esquemas de multirresolución es la reconstrucción utilizada. De ahí que nuestro objetivo sea el análisis de dos reconstrucciones de particular importancia en este entorno.

Para entender mejor estos conceptos en el capítulo 2 definimos con cierto detalle la multirresolución de Harten y en particular el caso correspondiente a una discretización de valores puntuales. En el capítulo 3 se presentan las definiciones de ambas reconstrucciones, así como algunas de sus propiedades y también mencionaremos brevemente algunas posibles aplicaciones. Entre estas aplicaciones hablaremos del zoom de datos geológicos, de la compresión de imágenes digitales y de fórmulas para la integración numérica adaptadas a las singularidades.. También cabe comentar que en el capítulo 4, podremos observar los resultados de los experimentos numéricos llevados a cabo con las reconstrucciones anteriormente mencionadas. En los capítulos 5 y 6 daremos una descripción de la *Interfaz Gráfica de Usuario* y de los códigos generados en Matlab, respectivamente. Para finalizar, en el capítulo 7 daremos algunas conclusiones extraídas del proyecto.

Capítulo 2

Algoritmos de multirresolución y operadores de reconstrucción

En este capítulo resumimos el análisis de *Multirresolución de Harten* recalcando especialmente las partes que van a ser utilizadas en este proyecto fin de carrera, como es el caso de la discretización por *valores puntuales*.

2.1. Multirresolución de Harten

El fin de la multirresolución es una reordenación multiescala del contenido de un conjunto de datos discretos a una resolución concreta. Por ejemplo, esta información puede ser el resultado de discretizar una función, denotada f, y es un elemento perteneciente a un espacio, 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 $k-1$:

$$P_{k-1}^{k} : V^{k-1} \rightarrow V^{k}$$

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:

$$D_k : F \rightarrow V^k$$

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:

$$R_k : V^k \rightarrow F$$

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.

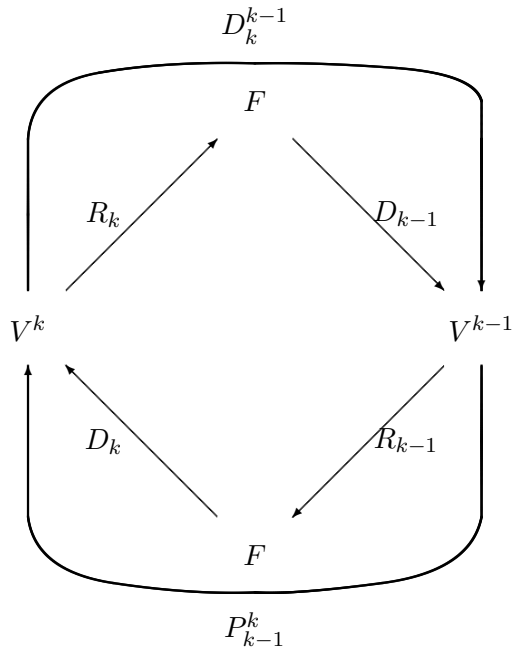

Figura 2.1: Definición de operadores.

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:

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

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:

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

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:

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

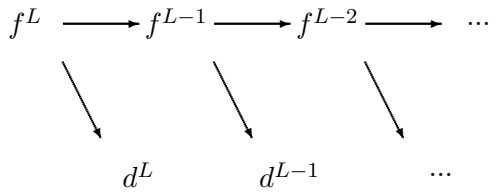

Figura 2.2: Descomposición multiescala.

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 en cuestión, se utilizará la discretización por valores puntuales, que pasamos a describir en el siguiente apartado.

2.2. Multirresolución 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

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

El valor de los coeficientes d_j^k varía en función de la singularidad, por lo que podemos pensar en el análisis de multirresolución como un análisis de la regularidad de una función. Los mayores coeficientes 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 ocurre cuando se utiliza interpolación lineal centrada.

Al usar técnicas de interpolación que dependan de los datos, esto es, no lineales, se minimiza la zona afectada por cada singularidad, y la capacidad de compresión del esquema de multirresolución mejora.
Las técnicas de interpolación más usuales son los polinomios.

Capítulo 3

Operadores de reconstrucción lineales y no lineales

Lo que tratan de conseguir las reconstrucciones es, a partir de unos datos discretizados de una función, aproximar la función original. Para ello, se intenta conseguir una función lo más sencilla posible (normalmente se usarán polinomios) que aproxime a la función original usando los datos disponibles. Para ello es posible usar técnicas lineales, independientes de los datos, que son relativamente sencillas de aplicar o técnicas no lineales, que se adaptan a los datos, las cuales proporcionan un mejor resultado.

3.1. Algunos ejemplos en 1D

En esta sección nos centraremos en describir los operadores con los que hemos trabajado a lo largo de este proyecto fin de carrera. En el caso de las reconstrucciones lineales hemos usado el operador de *Lagrange* y para las reconstrucciones no lineales hemos optado por *PPH*. A continuación los describiremos con más detalle en su subsección correspondiente.

3.1.1. *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

$$P_j(x) = f_{j-1}L_{-1}(x) + f_jL_0(x) + f_{j+1}L_1(x) + f_{j+2}L_2(x) \tag{3.1}$$

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

$$L_i(x) = \prod_{l=-1, l\neq i}^{2} \frac{(x - x_{j+l})}{(x_{j+i} - x_{j+l})}. \tag{3.2}$$

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

$$P_j(x_{j+\frac{1}{2}}) = f_{j-1}L_{-1}(x_{j+\frac{1}{2}}) + f_jL_0(x_{j+\frac{1}{2}}) + f_{j+1}L_1(x_{j+\frac{1}{2}}) + f_{j+2}L_2(x_{j+\frac{1}{2}}), \tag{3.3}$$

y haciendo algunos cálculos

$$P_j(x_{j+\frac{1}{2}}) = -\frac{1}{16}f_{j-1} + \frac{9}{16}f_j + \frac{9}{16}f_{j+1} - \frac{1}{16}f_{j+2}. \tag{3.4}$$

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 (3.1). 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

$$P_j(x) = \sum_{i=-s+1}^{s} f_{j+i}L_i(x_{j+\frac{1}{2}}), \quad x \in [x_j, x_{j+1}], \tag{3.5}$$

donde los polinomios de *Lagrange* $L_i(x)$ se definen como en (3.2) por

$$L_i(x) = \prod_{l=-s+1, l\neq i}^{s} \frac{(x - x_{j+l})}{(x_{j+i} - x_{j+l})}. \tag{3.6}$$

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

$$P_j(x_{j+\frac{1}{2}}) = \sum_{i=-s+1}^{s} f_{j+i} L_i(x_{j+\frac{1}{2}}). \tag{3.7}$$

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 3.1 podemos ver los valores de las máscaras para $s = 2, 3, 4$.

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

Tabla 3.1: 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 [7].

3.1.2. *PPH*

Esta subsección trata de la descripción del operador *PPH*, cuyas principales características son tres. Primero, cada parte esta centrada con un stencil fijo centrado de $2s$ puntos. Segundo, su reconstrucción para las partes suaves es tan exacta como su equivalente lineal. Tercero, se reduce la exactitud cerca de las zonas con singularidades, pero no se pierde por completo como en su equivalente lineal.

Una vez hecha esta pequeña introducción sobre *PPH*, vamos a definir como construir un trozo polinómico 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 realizar es una modificación de algunos de ellos para suavizar la función de manera que luego podamos aplicar interpolación de *Lagrange* sobre datos sin discontinuidades apreciables. Dicha modificación tiene 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

$$B_{s-1}^{s-1} = 2, \tag{3.8}$$

$$B_{s-1}^{q} = \sum_{j=1}^{s-1-q} (-1)^j \left(\binom{2(q+j)}{(j-1)} - \binom{2(q+j)}{j} \right) B_{s-1}^{q+j}, \tag{3.9}$$

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

En la Tabla 3.1.2 podemos ver el valor de estos coeficientes para $s = 2, 3, 4$.

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

Tabla 3.2: Coeficientes B_{s-1}^i, $s = 2, 3, 4$.

También vamos a necesitar las medias p-power $power_p(x, y)$, que se introdujeron en [26] para cualquier número entero $p \geq 1$, y cualquier pareja (x, y) como:

$$power_p(x, y) = \frac{sign(x) + sign(y)}{2} \frac{x+y}{2} \left(1 - \left| \frac{x-y}{x+y} \right|^p \right). \tag{3.10}$$

Igualmente necesitaremos usar las diferencias divididas, que es conocido que actúan como indicadores de zonas de suavidad de la función. Es conocido que 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 [6] y [5] 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

  $$\tilde{f} := \begin{cases} (\ldots, f_{j-2}, f_{j-1}, f_j, f_{j+1}, \tilde{f}_{j+2}, f_{j+3}, \ldots) & \text{si } |\Delta^2_{j-1} f| \leq |\Delta^2_j f|, \\ (\ldots, f_{j-2}, \tilde{f}_{j-1}, f_j, f_{j+1}, f_{j+2}, f_{j+3}, \ldots) & \text{en otro caso.} \end{cases} \tag{3.11}$$

- 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

  $$f = \begin{cases} (\ldots, f_{j-2}, f_{j-1}, f_j, f_{j+1}, \tilde{f}_{j+2}, f_{j+3}, \ldots) & \text{si } |\Delta^2_{j-1} f| \leq |\Delta^2_j f|, \\ (\ldots, f_{j-2}, \tilde{f}_{j-1}, f_j, f_{j+1}, f_{j+2}, f_{j+3}, \ldots) & \text{en otro caso.} \end{cases} \tag{3.12}$$

  y definimos

  $$\tilde{f} := \begin{cases} (\ldots, f_{j-3}, f_{j-2}, \tilde{f}_{j-1}, \tilde{f}_j, \tilde{f}_{j+1}, \tilde{f}_{j+2}, \tilde{f}_{j+3}, f_{j+4} \ldots) & \text{si } |\Delta^4_{j-2} \tilde{f}| \leq |\Delta^4_{j-1} \tilde{f}|, \\ (\ldots, f_{j-3}, \tilde{f}_{j-2}, \tilde{f}_{j-1}, \tilde{f}_j, \tilde{f}_{j+1}, \tilde{f}_{j+2}, f_{j+3}, f_{j+4} \ldots) & \text{en otro caso.} \end{cases} \tag{3.13}$$

  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 3.1. 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 3.2.

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 3.3. 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 3.4. A continuación se considerarán las diferencias divididas de tercer orden indicadas en la Figura 3.5 y por tanto se cambiará el valor f_{j-2} tal y como se ve en la Figura 3.6.

En el apartado siguiente vamos a explicar cómo llevar a cabo la reconstrucción en 2D, realizando un proceso basado en producto tensor. Otra posibilidad sería el realizar una reconstrucción con las mismas ideas usadas para definir la reconstrucción *PPH* directamente utilizando polinomios en dos variables sin recurrir al caso de una variable.

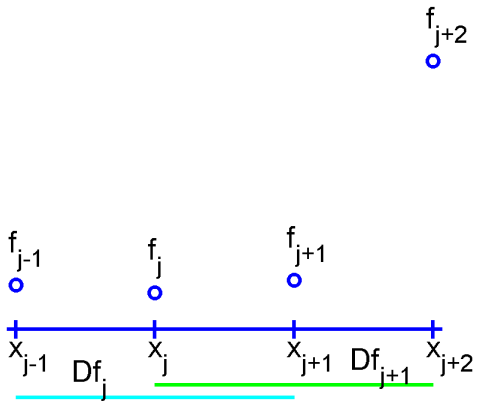

Figura 3.1: Primer paso de la reconstrucción *PPH* con 4 puntos

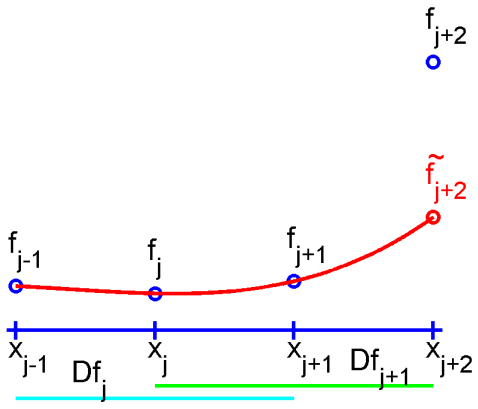

Figura 3.2: Construcción de la reconstrucción *PPH* con 4 puntos con los valores modificados

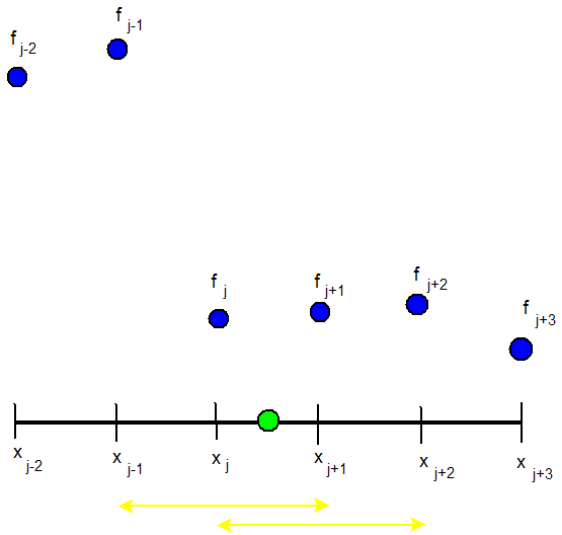

Figura 3.3: Primer paso de la reconstrucción *PPH* con 6 puntos

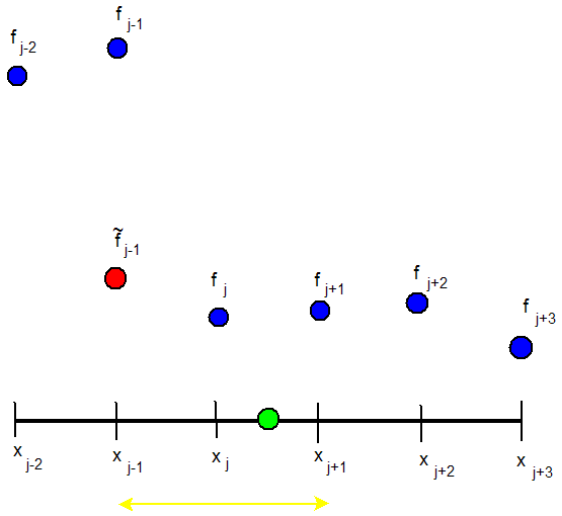

Figura 3.4: Modificación del primer valor en la construcción de la reconstrucción *PPH* con 6 puntos

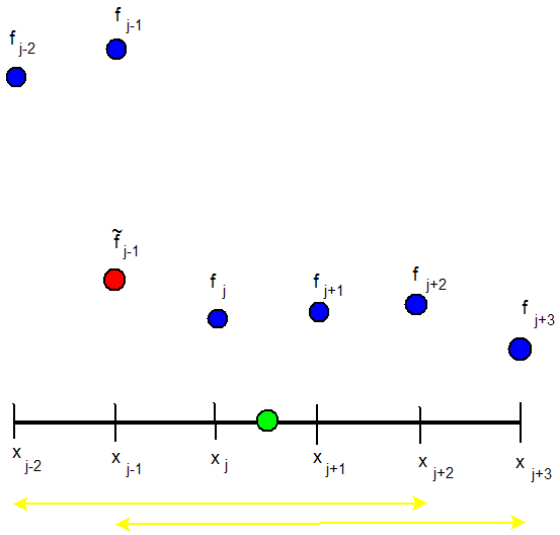

Figura 3.5: Segundo paso de la reconstrucción *PPH* con 6 puntos

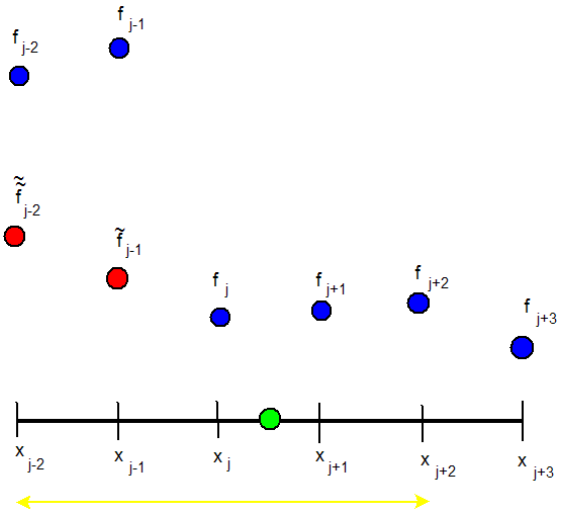

Figura 3.6: Modificación del segundo valor en la construcción de la reconstrucción *PPH* con 6 puntos

3.2. Construcción de las reconstrucciones en 2D

En la primera imagen (ver figura 3.7) está representado el mallado de puntos donde se dispone de datos originales, los cuales podrían corresponder a los valores de altitud de una cierta región montañosa o a la profundidad de un entorno marino en cual es difícil llevar a cabo más mediciones. Se trata de obtener una mayor resolución de los datos interpolando los valores disponibles. Para realizar la reconstrucción via producto tensor, primero se realiza la reconstrucción por filas y una vez realizado este proceso se vuelve a realizar la reconstrucción, pero esta vez por columnas. Este proceso puede verse en las figuras 3.8 y 3.9, respectivamente.

Otra posibilidad para obtener una reconstrucción sería trabajar directamente en 2D, construyendo una expresión polinómica que no venga de hacer producto tensor. En este sentido hablaríamos de reconstrucciones no lineales no separables, las cuales presentan ventajas notables a la hora de tratar de evitar las discontinuidades presentes.

También cabe mencionar en este apartado que las reconstrucciones pueden utilizarse también para definir esquemas de subdivisión, que se emplean para hacer zoom de una manera más rápida y menos costosa computacionalmente. En este sentido puede consultarse el trabajo [17].

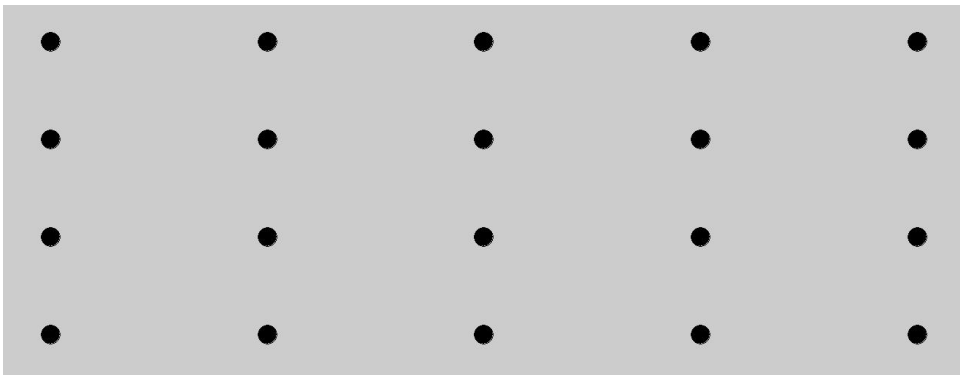

Figura 3.7: Datos originales

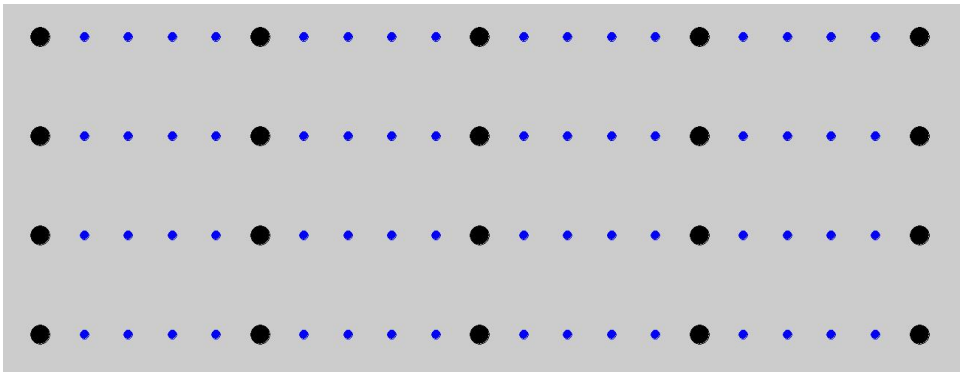

Figura 3.8: Primer paso de la reconstrucción en 2D. Interpolación por filas

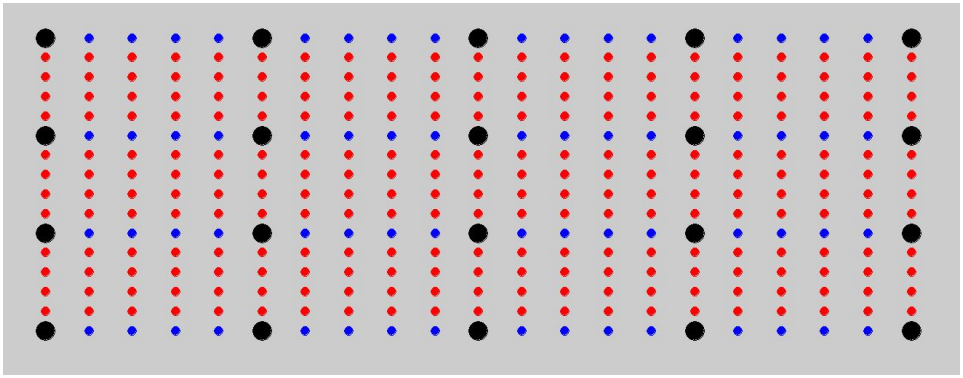

Figura 3.9: Segundo paso de la reconstrucción en 2D. Interpolación por columnas

3.3. Algunas aplicaciones

En este apartado vamos a comentar brevemente algunas aplicaciones de estos operadores de reconstrucción. Entre ellas podemos citar el zoom de datos, la compresión de imágenes digitales y la obtención de fórmulas de cuadratura para la integración numérica.

3.3.1. Zoom de datos

Los operadores de reconstrucción descritos pueden ser utilizados para llevar a cabo un zoom de unos datos discretos obtenidos mediante experimentación. Supongamos el caso de tener datos que provienen de mediciones de la altura de un área montañosa o de la profundidad de una fosa marina.

Dichos datos pueden ser costosos de obtener, y así para obtener una mejor resolución una posibilidad es realizar un zoom de los datos disponibles.

Este zoom se realiza construyendo una reconstrucción como las descritas en este proyecto y posteriormente evaluándola en un número mayor de puntos, usualmente en los puntos centrales de los intervalos considerados. Este proceso puede repetirse de manera secuencial y constituye un zoom de datos, que matemáticamente también es llamado esquema de subdivisión. El esquema de subdivisión no es más pues que la aplicación recursiva del operador de predicción que tal como se definió en el capítulo 2 coincide con la aplicación de la reconstrucción seguida de la discretización en el nivel superior de resolución.

También se puede realizar mediante este proceso un zoom de imágenes digitales, si bien para realizar este proceso es conveniente centrarse en una discretización de los datos por medias en celda. Es decir, es mejor en este caso interpretar los valores de intensidad lumínica en cada pixel como una media de una cierta función integrable en la celda representada por el pixel.

3.3.2. Compresión de imágenes digitales

En este apartado nos debemos centrar también en el entorno de medias en celda. Consideremos el mallado anidado dado en $\mathbb{R}$ por:

$$X^k = \{x_j^k\}_{j\in\mathbb{Z}}, \quad x_j^k = jh_k, \quad h_k = 2^{-k},$$

y el operador de discretización por medias en celda dado por:

$$\mathcal{D}_k : L^1(\mathbb{R}) \to V^k, \qquad f_j^k = (\mathcal{D}_k f)_j = \frac{1}{h_k}\int_{x_{j-1}^k}^{x_j^k} f(x)dx, \quad j \in \mathbb{Z}, \quad (3.14)$$

donde $L^1(\mathbb{R})$ representa el espacio de las funciones absolutamente integrables en $\mathbb{R}$ y V^k es el espacio de secuencias en el nivel de resolución k.

De la propiedad aditiva de la integral obtenemos la expresión del operador de decimación que conecta los diferentes niveles de resolución

$$f_j^{k-1} = (\mathbf{D}_k^{k-1} f^k)_j = \frac{1}{h_{k-1}}\int_{x_{j-1}^{k-1}}^{x_j^{k-1}} f(x)dx = \frac{1}{2h_k}\int_{x_{2j-2}^k}^{x_{2j}^k} f(x)dx = \frac{1}{2}(f_{2j-1}^k + f_{2j}^k).$$

Un operador de reconstrucción consistente con el operador de discretización definido puede construirse usando técnicas que involucran los valores puntuales de la función primitiva de la función f de la que se obtuvieron las medias en celda. Para más detalles puede consultarse [2],[9],[25].

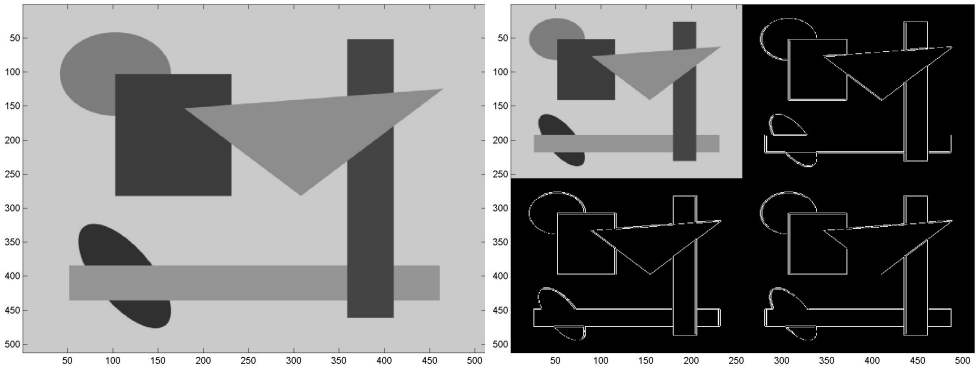

Figura 3.10: Izquierda: Imagen geométrica original. Derecha: Versión de multirresolución

Para realizar la compresión de imágenes se lleva a cabo una transformación de los datos utilizando la multirresolución de Harten de tal manera que la nueva representación de los datos es más susceptible a procesos de compresión ya que muchos de sus elementos son cero o cercanos a cero. El ahorro en memoria viene de considerar como cero muchos de estos elementos para que se puedan guardar con menor gasto. En la parte izquierda de la figura 3.10 vemos la representación original de una imagen geométrica, y en la parte derecha su versión de multirresolución una vez truncada. Vemos que una gran cantidad de coeficiente obtenidos en la representación de la parte derecha son cero y por tanto aparecen representados de color negro. La representación de multirresolución queda dividida en cuatro partes. La primera corresponde a un nivel inferior de resolución y aparece en la esquina superior izquierda. Las otras tres representan detalles y serán más pequeños cuanto mejor sea la reconstrucción. Esto puede verse también en la figura 3.11 donde se han comparado las versiones de multirresolución obtenidas por las reconstrucciones de *Lagrange* y *PPH* una vez adaptadas al entorno de medias en celda. Notar que en este caso se han realizado dos niveles de multirresolución.

Al descomprimir la imagen, es decir, aplicarle a la versión de multirresolución el algoritmo inverso, se obtiene una aproximación de la imagen original. En la figura 3.12 vemos las aproximaciones obtenidas por *Lagrange* y por *PPH* para una misma tasa de compresión. Es notable que la definición de los ejes para el método no lineal *PPH* es significativamente mejor que para *Lagrange*. Para ver más detalles puede consultarse [25]. Semejantes resultados se obtienen en la aplicación de estos operadores de reconstrucción al zoom de imágenes digitales.

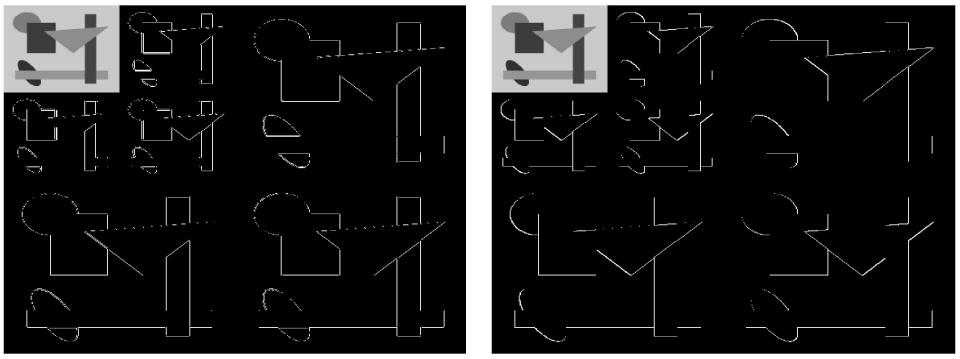

Figura 3.11: Izquierda: Versión de multirresolución del método de *Lagrange*. Derecha: Versión de multirresolución del método *PPH*. 90 % de compresión

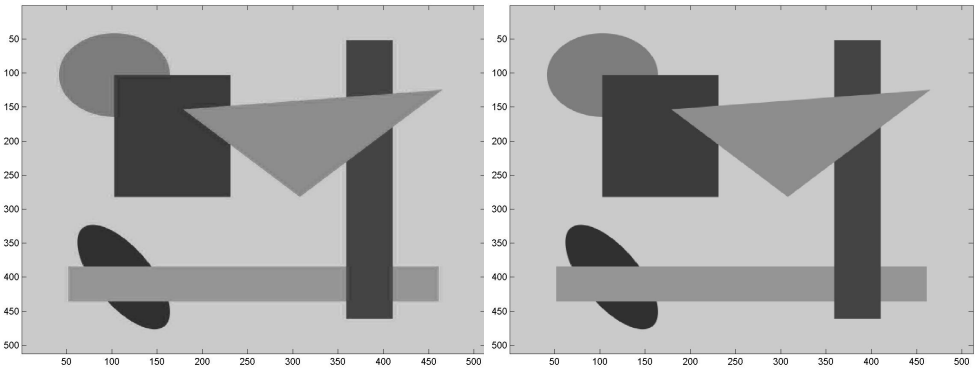

Figura 3.12: Izquierda: Reconstrucción obtenida por el esquema de multirresolución de *Lagrange*, Derecha: Reconstrucción obtenida por el esquema de multirresolución *PPH*. 90 % de compresión

3.3.3. Integración numérica

Estas reconstrucciones no lineales pueden emplearse también para el cálculo de integrales de manera numérica, y generar fórmulas de cuadratura para evitar discontinuidades. Tales aplicaciones se usan en campos como la aproximación de soluciones de ecuaciones diferenciales hiperbólicas. Vamos a tratar de aproximar la integral de una función evitando la discontinuidad que presenta en uno de los extremos. Como ejemplo calcularemos la integral de la función seno en el intervalo $[0,1]$ definida como $f_1(x) = \sin(2\pi x)$ pero haciendo uso de los datos que vienen de la función

$$f_2(x) = \begin{cases} \sin(2\pi x), & 0 \leq x \leq 0{,}8, \\ \sin(2\pi x) + 2, & 0{,}8 < x \leq 1. \end{cases} \tag{3.15}$$

Si aproximamos la integral por la integral de un polinomio $p_3(x)$ de grado 3 que pasa por los puntos $(x_0, f_2(x_0)) = (0, f_2(0))$, $(x_1, f_2(x_1)) = (\frac{1}{3}, f_2(\frac{1}{3}))$, $(x_2, f_2(x_2)) = (\frac{2}{3}, f_2(\frac{2}{3}))$, $(x_3, f_2(x_3)) = (1, f_2(1))$ obtenemos la siguiente regla

$$\int_0^1 f_2(x)dx \approx \int_0^1 p_3(x)dx \tag{3.16}$$

$$\int_0^1 p_3(x)dx = \frac{3h}{8}(f_2(x_0) + 3f_2(x_1) + 3f_2(x_2) + f_2(x_3)), \tag{3.17}$$

donde $h = b - a = 1 - 0 = 1$. Sustituyendo valores obtenemos la aproximación,

$$\int_0^1 f_2(x)dx \approx \frac{3}{8}(\sin(0) + 3\sin(\frac{2\pi}{3}) + 3\sin(\frac{4\pi}{3}) + \sin(2\pi) + 2) = 0{,}25. \tag{3.18}$$

Esta aproximación no es buena ni para la función $f_2(x)$ ya que la integral exacta vale 0,4 como puede comprobarse fácilmente, ni para aproximar la integral de la función $f_1(x)$ ya que su integral exacta vale 0.

Una cosa que podemos hacer para aproximar la integral de $f_1(x)$ teniendo como datos los provenientes de $f_2(x)$ es hacer un cambio de los datos para evitar la discontinuidad del último valor de la derecha. Así modificamos el valor de $f_2(x_3)$ obteniendo un $\tilde{f}_2(x_3)$ de la siguiente forma

$$\tilde{f}_2(x_3) = f_2(x_2) + f_2(x_1) - f_2(x_0) + 2H(\Delta_1^2 f, \Delta_2^2 f), \tag{3.19}$$

donde $H(x,y)$ representa la media armónica de dos valores

$$H(x,y) = \begin{cases} \frac{2xy}{x+y}, & xy > 0, \\ 0, & \text{en otro caso}, \end{cases} \tag{3.20}$$

y $\Delta_1^2 f$, $\Delta_2^2 f$ son las diferencias divididas de segundo orden calculadas como

$$\begin{aligned} \Delta_1^2 f &= f_2(x_0) - 2f_2(x_1) + f_2(x_2), \\ \Delta_2^2 f &= f_2(x_1) - 2f_2(x_2) + f_2(x_3). \end{aligned}$$

Realizando estos cálculos para nuestro ejemplo tenemos

$$\begin{aligned} \Delta_1^2 f &= -2{,}598076, \\ \Delta_2^2 f &= 4{,}598076, \\ \tilde{f}_2(x_3) &= \sin(\frac{4\pi}{3}) + \sin(\frac{2\pi}{3}) = 2{,}220446 \cdot 10^{-16}. \end{aligned}$$

Y así

$$\begin{aligned} \int_0^1 f_2(x)dx &\approx \int_0^1 p_3(x)dx = \frac{3h}{8}(f_2(x_0) + 3f_2(x_1) + 3f_2(x_2) + \tilde{f}_2(x_3)) \\ &= 1{,}110223 \cdot 10^{-16}. \end{aligned}$$

Capítulo 4

Experimentos numéricos

En esta sección vamos a usar los programas implementados en Matlab para realizar varios experimentos numéricos tanto en 1D como 2D. Para efectuar dichos experimentos se usarán funciones suaves y funciones con discontinuidades.

4.1. Experimentos en 1D

Este apartado esta dividido en *Función suave* y *Función con discontinuidad*. Los experimentos se realizarán usando tanto reconstrucción de *Lagrange* como *PPH*.

4.1.1. Función suave

Ahora nos centraremos en la reconstrucción de una función suave mediante los métodos de *Lagrange* y *PPH*. Para realizar dichos experimentos, hemos seleccionado la siguiente función,

$$f_1(x) = \sin 2\pi x. \tag{4.1}$$

En la figura 4.1 podemos ver representada dicha función en el intervalo [0,1].

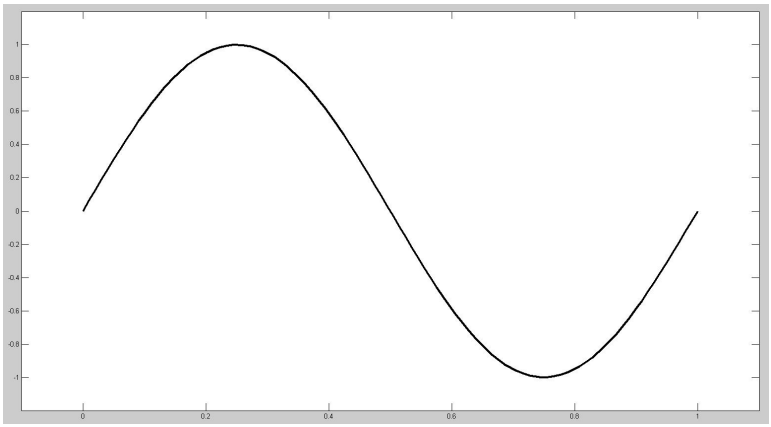

Figura 4.1: Representación de la función suave $f_1(x)$ usada para los experimentos numéricos en 1D

Comenzaremos usando la reconstrucción de *Lagrange*. El orden de la reconstrucción y el espaciado entre nodos de reconstrucción (ENR) tomarán distintos valores, mientras que el espaciado entre nodos de evaluación (ENE) siempre tendrá un valor constante para todas las reconstrucciones y éste será de 0.001.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma*** ∞
Lagrange	4	0.1	$1{,}6826 \cdot 10^{-3}$	$6{,}9540 \cdot 10^{-5}$	$3{,}5345 \cdot 10^{-3}$
		0.05	$1{,}0210 \cdot 10^{-4}$	$4{,}0293 \cdot 10^{-6}$	$2{,}2364 \cdot 10^{-4}$
		0.025	$6{,}1725 \cdot 10^{-6}$	$2{,}4059 \cdot 10^{-7}$	$1{,}4171 \cdot 10^{-5}$
	6	0.1	$1{,}2870 \cdot 10^{-4}$	$6{,}2848 \cdot 10^{-6}$	$2{,}8245 \cdot 10^{-4}$
		0.05	$2{,}1571 \cdot 10^{-6}$	$9{,}0096 \cdot 10^{-8}$	$4{,}5655 \cdot 10^{-6}$
		0.025	$3{,}2560 \cdot 10^{-8}$	$1{,}2924 \cdot 10^{-9}$	$7{,}2709 \cdot 10^{-8}$

Tabla 4.1: Errores para la reconstrucción de *Lagrange* en 1D con la función $f_1(x)$. Espacio entre nodos de Evaluación: 0.001

En la tabla 4.1 podemos ver los resultados numéricos usando la reconstrucción de *Lagrange*. Y en la figura 4.2, podemos ver la función original (en negro) y la reconstrucción usando *Lagrange* (rojo discontinuo), para orden 4, y un espaciado entre nodos de reconstrucción de 0.05 y entre nodos de e-

Método	**Orden**	**ENR**	***Orden numérico***		
			Norma L1	***Norma L2***	***Norma ∞***
Lagrange	4	0.1	4,0426	4,1092	3,9822
		0.05	4,0480	4,0658	3,9801
		0.025	4,0281	4,0333	3,9944
	6	0.1	5,8988	6,1242	5,9510
		0.05	6,0498	6,1233	5,9725
		0.025	6,0481	6,0699	5,9925

Tabla 4.2: Orden numérico para la reconstrucción de *Lagrange* en 1D con la función $f_1(x)$. Espacio entre nodos de Evaluación: 0.001

valuación de 0.001. Los resultados se ajustan a la teoría, y la reconstrucción alcanza orden 4 como puede verse en la Tabla 4.2. La reconstrucción por tanto se pega bien a la función original que proporcionó los datos iniciales.

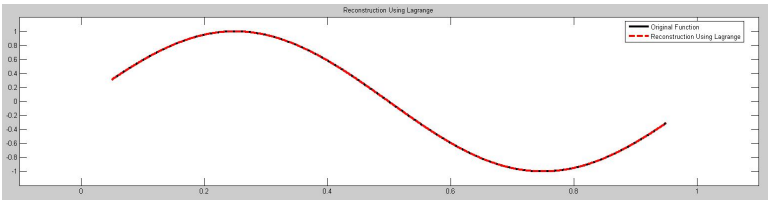

Figura 4.2: Función $f_1(x)$ (negro) y reconstrucción por el método de *Lagrange* (rojo discontinuo)

A continuación vamos a realizar el mismo proceso pero esta vez para la reconstrucción *PPH*. El orden de la reconstrucción, el espaciado entre nodos de evaluación y reconstrucción, serán los mismos que en el experimento anterior.

Como antes, la tabla 4.3 muestra los resultados obtenidos y en la figura 4.3 esta representada la función (negro) y la reconstrucción con *PPH* (verde discontinuo). Como podemos observar, ambas reconstrucciones se ajustan muy bien a la función original. También los ordenes numéricos son los esperados, como se muestra en la tabla 4.4.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma ∞***
PPH	4	0.1	$2{,}5998 \cdot 10^{-3}$	$1{,}0458 \cdot 10^{-4}$	$4{,}9380 \cdot 10^{-3}$
		0.05	$2{,}2368 \cdot 10^{-4}$	$9{,}3658 \cdot 10^{-6}$	$6{,}4531 \cdot 10^{-4}$
		0.025	$6{,}1870 \cdot 10^{-6}$	$2{,}4069 \cdot 10^{-7}$	$1{,}4174 \cdot 10^{-5}$
	6	0.1	$1{,}9484 \cdot 10^{-4}$	$1{,}0060 \cdot 10^{-4}$	$5{,}1311 \cdot 10^{-4}$
		0.05	$1{,}0268 \cdot 10^{-5}$	$7{,}6928 \cdot 10^{-7}$	$8{,}4129 \cdot 10^{-5}$
		0.025	$3{,}2560 \cdot 10^{-8}$	$1{,}2924 \cdot 10^{-9}$	$7{,}2709 \cdot 10^{-8}$

Tabla 4.3: Errores para la reconstrucción *PPH* en 1D con la función $f_1(x)$. Espacio entre nodos de Evaluación: 0.001

Método	**Orden**	**ENR**	***Orden numérico***		
			Norma L1	***Norma L2***	***Norma ∞***
PPH	4	0.1	3,5388	3,4811	2,9358
		0.05	5,1761	5,2821	5,5086
		0.025	4,0311	4,0339	3,9947
	6	0.1	4,24597	3,70903	2,6086
		0.05	8,30095	9,2173	10,1762
		0.025	6,0480	6,0699	5,99253

Tabla 4.4: Orden numérico para la reconstrucción *PPH* en 1D con la función $f_1(x)$. Espacio entre nodos de Evaluación: 0.001

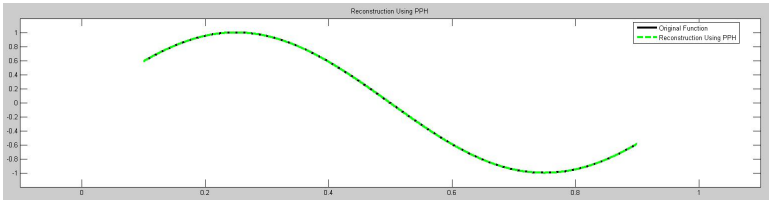

Figura 4.3: Función $f_1(x)$ (negro) y reconstrucción con *PPH* (verde discontinuo)

4.1.2. Función suave 2

En este apartado realizaremos los mismos experimentos que en la sección anterior cambiando la función $f_1(x)$ por la siguiente función,

$$f_2(x) = \sin(\pi x + \pi). \tag{4.2}$$

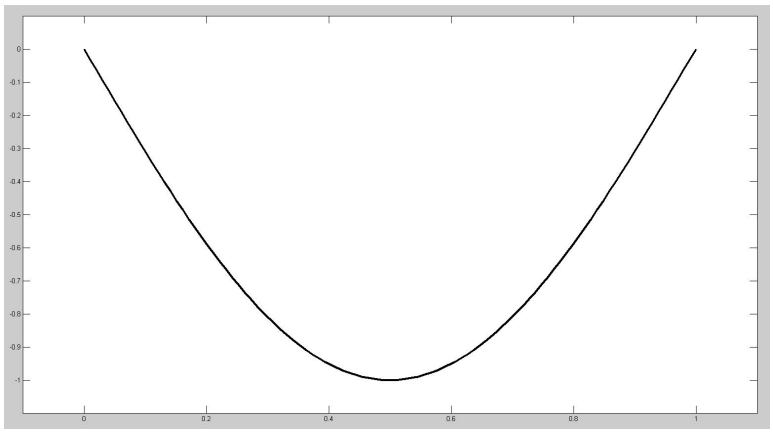

Figura 4.4: Representación de la función suave $f_2(x)$ usada para los experimentos numéricos en 1D

En la figura 4.4 vemos representada la función que vamos a usar en el intervalo [0,1].

Empezaremos usando la reconstrucción de *Lagrange*. El orden de la reconstrucción y el espaciado entre nodos de reconstrucción (ENR) tomarán distintos valores, mientras que el espaciado entre nodos de evaluación (ENE) tendrá un valor constante para todas las reconstrucciones y éste será de 0.001.

En la tabla 4.5 vemos los resultados numéricos usando la reconstrucción de *Lagrange*. Y en la figura 4.5, vemos la función original (en negro) y la reconstrucción usando *Lagrange* (rojo discontinuo), para orden 4, y un espaciado entre nodos de reconstrucción de 0.05 y entre nodos de evaluación de 0.001. Dichos resultados se ajustan a la teoría, y la reconstrucción alcanza orden 4 como puede verse en la Tabla 4.6. La reconstrucción por tanto se ajusta a la función original que proporcionó los datos iniciales.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma ∞***
Lagrange	4	0.1	$1{,}1200 \cdot 10^{-4}$	$4{,}5211 \cdot 10^{-6}$	$2{,}2364 \cdot 10^{-4}$
		0.05	$6{,}4817 \cdot 10^{-6}$	$2{,}5386 \cdot 10^{-7}$	$1{,}4195 \cdot 10^{-5}$
		0.025	$3{,}8728 \cdot 10^{-7}$	$1{,}5064 \cdot 10^{-8}$	$8{,}8908 \cdot 10^{-7}$
	6	0.1	$2{,}5721 \cdot 10^{-6}$	$1{,}1713 \cdot 10^{-7}$	$4{,}5655 \cdot 10^{-6}$
		0.05	$3{,}5744 \cdot 10^{-8}$	$1{,}4493 \cdot 10^{-9}$	$7{,}2840 \cdot 10^{-8}$
		0.025	$5{,}1619 \cdot 10^{-10}$	$2{,}0298 \cdot 10^{-11}$	$1{,}1419 \cdot 10^{-9}$

Tabla 4.5: Errores para la reconstrucción de *Lagrange* en 1D con la función $f_2(x)$. Espacio entre nodos de Evaluación: 0.001

Método	**Orden**	**ENR**	***Orden numérico***		
			Norma L1	***Norma L2***	***Norma ∞***
Lagrange	4	0.1	4,1109	4,1545	3,9776
		0.05	4,0649	4,0749	3,9969
		0.025	4,0323	4,0353	3,9986
	6	0.1	6,1691	6,3366	5,9698
		0.05	6,1136	6,1578	5,9951
		0.025	6,0637	6,0759	5,9981

Tabla 4.6: Orden numérico para la reconstrucción de *Lagrange* en 1D con la función $f_2(x)$. Espacio entre nodos de Evaluación: 0.001

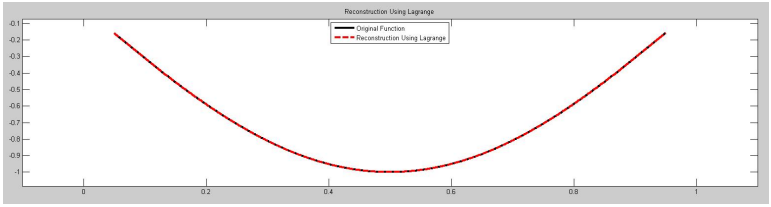

Figura 4.5: Función $f_2(x)$ (negro) y reconstrucción por el método de *Lagrange* (rojo discontinuo)

Seguidamente vamos a repetir el proceso pero esta vez para la reconstrucción *PPH*. El orden de la reconstrucción, el espacio entre nodos de evaluación y reconstrucción, serán los mismos que en el experimento anterior.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma*** ∞
PPH	4	0.1	$1{,}1902 \cdot 10^{-4}$	$4{,}7380 \cdot 10^{-6}$	$2{,}2429 \cdot 10^{-4}$
		0.05	$6{,}6182 \cdot 10^{-6}$	$2{,}5734 \cdot 10^{-7}$	$1{,}4198 \cdot 10^{-5}$
		0.025	$3{,}8958 \cdot 10^{-7}$	$1{,}5117 \cdot 10^{-8}$	$8{,}8910 \cdot 10^{-7}$
	6	0.1	$2{,}5643 \cdot 10^{-6}$	$1{,}1682 \cdot 10^{-7}$	$4{,}5642 \cdot 10^{-6}$
		0.05	$3{,}5731 \cdot 10^{-8}$	$1{,}4489 \cdot 10^{-9}$	$7{,}2840 \cdot 10^{-8}$
		0.025	$5{,}1618 \cdot 10^{-10}$	$2{,}0298 \cdot 10^{-11}$	$1{,}1419 \cdot 10^{-9}$

Tabla 4.7: Errores para la reconstrucción *PPH* en 1D con la función $f_2(x)$. Espacio entre nodos de Evaluación: 0.001

Método	**Orden**	**ENR**	***Orden numérico***		
			Norma L1	***Norma L2***	***Norma*** ∞
PPH	4	0.1	4,1687	4,2025	3,9816
		0.05	4,0864	4,0894	3,9972
		0.025	4,0386	4,0391	3,9986
	6	0.1	6,1652	6,3332	5,9694
		0.05	6,1131	6,1574	5,9951
		0.025	6,0636	6,0758	5,9981

Tabla 4.8: Orden numérico para la reconstrucción *PPH* en 1D con la función $f_2(x)$. Espacio entre nodos de Evaluación: 0.001

Como antes, la tabla 4.7 muestra los resultados obtenidos y la figura 4.6 representa la función (negro) y la reconstrucción con *PPH* (verde discontinuo). Como podemos observar, ambas reconstrucciones se ajustan muy bien a la función original. También los ordenes numéricos son los esperados, como se muestra en la tabla 4.8.

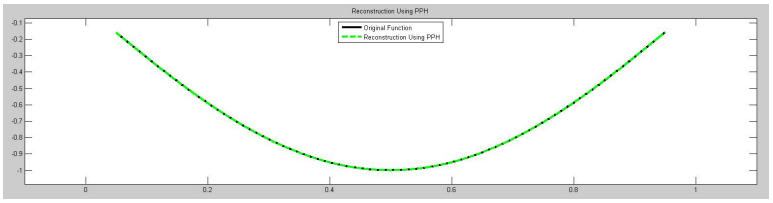

Figura 4.6: Función $f_1(x)$ (negro) y reconstrucción con PPH (verde discontinuo)

4.1.3. Función con discontinuidad

En este apartado realizaremos los experimentos con una función que presenta dos discontinuidades. Su fórmula es la siguiente,

$$f_3(x) = \begin{cases} \sin(2\pi x), & x \leq 0{,}2, \\ -50 + \sin(2\pi x)), & 0{,}2 < x \leq 0{,}6, \\ -50 + \sin(1{,}2\pi) - 10\sin(2\pi(x - 0{,}6)), & 0{,}6 < x \leq 1. \end{cases} \tag{4.3}$$

Podemos ver su representación en la figura 4.7.

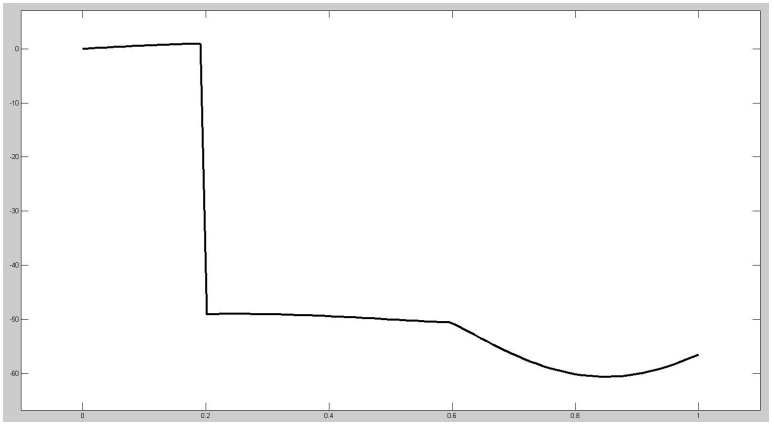

Figura 4.7: Representación de la función discontinua $f_3(x)$ usada para los experimentos numéricos en 1D

Como hicimos anteriormente para funciones continuas, vamos a realizar una tabla donde se muestran los errores numéricos de ambas reconstrucciones.
Empezaremos por la reconstrucción de *Lagrange*. Realizaremos el experimento con varios ordenes y espaciados entre nodos de reconstrucción. El espacio entre nodos de evaluación quedará fijado en 0.001.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma ∞***
Lagrange	4	0.1	3,6722	$3{,}6305 \cdot 10^{-1}$	$4{,}9580 \cdot 10^{1}$
		0.05	1,6048	$2{,}2654 \cdot 10^{-1}$	$4{,}9156 \cdot 10^{1}$
		0.025	$7{,}4361 \cdot 10^{-1}$	$1{,}4951 \cdot 10^{-1}$	$4{,}8294 \cdot 10^{1}$
	6	0.1	4,7619	$4{,}8513 \cdot 10^{-1}$	$4{,}9605 \cdot 10^{1}$
		0.05	1,9538	$2{,}5702 \cdot 10^{-1}$	$4{,}9204 \cdot 10^{1}$
		0.025	$8{,}4964 \cdot 10^{-1}$	$1{,}5920 \cdot 10^{-1}$	$4{,}8384 \cdot 10^{1}$

Tabla 4.9: Errores de la reconstrucción de *Lagrange* en 1D con la función $f_3(x)$. Espacio entre nodos de Evaluación: 0.001

Método	**Orden**	**ENR**	***Orden numérico***		
			Norma L1	***Norma L2***	***Norma ∞***
Lagrange	4	0.1	1,1947	$6{,}8036 \cdot 10^{-1}$	$1{,}2388 \cdot 10^{-2}$
		0.05	1,1096	$5{,}9953 \cdot 10^{-1}$	$2{,}5534 \cdot 10^{-2}$
		0.025	1,0893	$5{,}8041 \cdot 10^{-1}$	$5{,}4153 \cdot 10^{-2}$
	6	0.1	1,2820	$9{,}1647 \cdot 10^{-1}$	$1{,}1707 \cdot 10^{-2}$
		0.05	1,2010	$6{,}9101 \cdot 10^{-1}$	$2{,}4242 \cdot 10^{-2}$
		0.025	1,1283	$6{,}2167 \cdot 10^{-1}$	$5{,}1878 \cdot 10^{-2}$

Tabla 4.10: Orden numérico para la reconstrucción de *Lagrange* en 1D con la función $f_3(x)$. Espacio entre nodos de Evaluación: 0.001

En la tabla 4.9 podemos ver el error numérico cometido por la reconstrucción. Y en la figura 4.8 podemos ver la función original (negro) con la reconstrucción de *Lagrange* (rojo discontinuo) y se observa claramente el efecto de Gibbs producido en la reconstrucción alrededor de la discontinuidad de salto. La discontinuidad en 0,6 no afecta tanto a la reconstrucción ya que es una esquina, es decir, una discontinuidad de salto de la derivada y no de la

función. Por tanto es una discontinuidad más suave. Después veremos que estas oscilaciones debidas al fenómeno de Gibbs se atenúan con el método *PPH*. También es reseñable que en la figura aparecen 3 intervalos afectados ya que hemos utilizado orden 4 de reconstrucción. Si utilizamos orden 6 entonces el número de intervalos afectados también crece y se convierte en 5, como puede verse en la figura 4.9.
Notar que ahora el orden numérico de aproximación se pierde debido a la discontinuidad. Y esto es debido a que sólo tiene sentido hablar de orden de aproximación en zonas suaves de la función (ver las tablas 4.10 y 4.12).

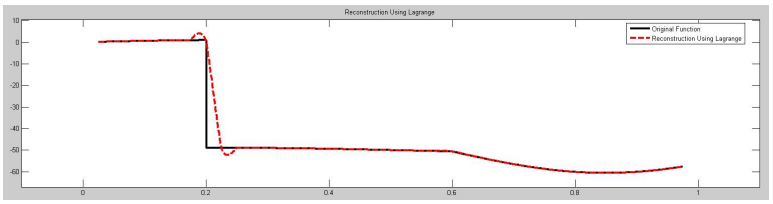

Figura 4.8: Función $f_3(x)$ (negro) y reconstrucción por el método de *Lagrange* (rojo discontinuo)

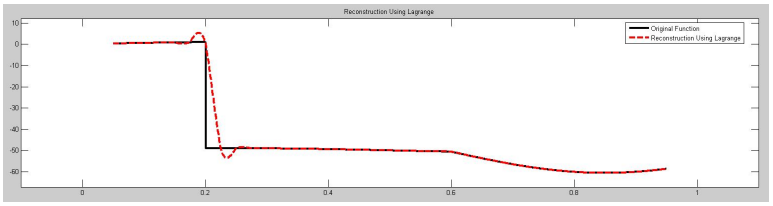

Figura 4.9: Función $f_3(x)$ (negro) y reconstrucción usando orden 6 por el método de *Lagrange* (rojo discontinuo)

Ahora vamos a realizar el mismo proceso para la reconstrucción *PPH*, con los mismos valores que para la reconstrucción de *Lagrange*.

En la tabla 4.9 vemos el error numérico cometido en la reconstrucción. Y en la figura 4.10 comparamos la función original (negro) con la reconstrucción por *PPH* (verde discontinuo) y vemos como el efecto de Gibbs se ha suavizado. Los valores de la reconstrucción son los siguientes, orden 4, espacio entre nodos de reconstrucción y evaluación 0.025 y 0.001 respectivamente.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma ∞***
PPH	4	0.1	2,8529	$3{,}3696 \cdot 10^{-1}$	$4{,}9416 \cdot 10^{1}$
		0.05	1,2462	$2{,}1007 \cdot 10^{-1}$	$4{,}8833 \cdot 10^{1}$
		0.025	$5{,}7645 \cdot 10^{-1}$	$1{,}3832 \cdot 10^{-1}$	$4{,}7667 \cdot 10^{1}$
	6	0.1	4,1935	$4{,}8319 \cdot 10^{-1}$	$4{,}9585 \cdot 10^{1}$
		0.05	1,5426	$2{,}5453 \cdot 10^{-1}$	$4{,}9168 \cdot 10^{1}$
		0.025	$6{,}7100 \cdot 10^{-1}$	$1{,}5763 \cdot 10^{-1}$	$4{,}8319 \cdot 10^{1}$

Tabla 4.11: Errores de la reconstrucción *PPH* en 1D con la función $f_3(x)$. Espacio entre nodos de Evaluación: 0.001

Método	**Orden**	**ENR**	***Orden numérico***		
			Norma L1	***Norma L2***	***Norma ∞***
PPH	4	0.1	1,1949	$6{,}8169 \cdot 10^{-1}$	$1{,}7117 \cdot 10^{-2}$
		0.05	1,1122	$6{,}0285 \cdot 10^{-1}$	$3{,}4866 \cdot 10^{-2}$
		0.025	1,1012	$5{,}8741 \cdot 10^{-1}$	$7{,}2286 \cdot 10^{-2}$
	6	0.1	1,4427	$9{,}2475 \cdot 10^{-1}$	$1{,}2201 \cdot 10^{-2}$
		0.05	1,2010	$6{,}9128 \cdot 10^{-1}$	$2{,}5131 \cdot 10^{-2}$
		0.025	1,1370	$6{,}2247 \cdot 10^{-1}$	$5{,}3211 \cdot 10^{-2}$

Tabla 4.12: Orden numérico para la reconstrucción *PPH* en 1D con la función $f_3(x)$. Espacio entre nodos de Evaluación: 0.001

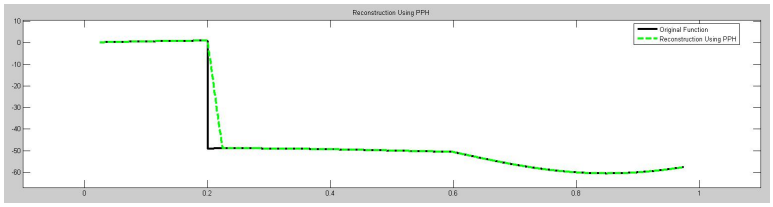

Figura 4.10: Función $f_3(x)$ (negro) y reconstrucción por el método *PPH* (verde discontinuo)

4.2. Experimentos en 2D

Este apartado también esta dividido en *Función suave* y *Función con discontinuidad*. Los experimentos como anteriormente se realizarán usando las reconstrucciones de *Lagrange* y *PPH*.

4.2.1. Función suave

Al igual que en 1D, vamos a comenzar los experimentos con una función sin discontinuidades, para ello hemos seleccionado la siguiente función,

$$g_1(x_1, x_2) = 3\cos\frac{\frac{(10(x_1-\frac{1}{2}))^2+(10(x_2-\frac{1}{2}))^2}{3}}{3+(10(x_1-\frac{1}{2}))^2+(10(x_2-\frac{1}{2}))^2} \tag{4.4}$$

Se puede ver su representación en la figura 4.11.

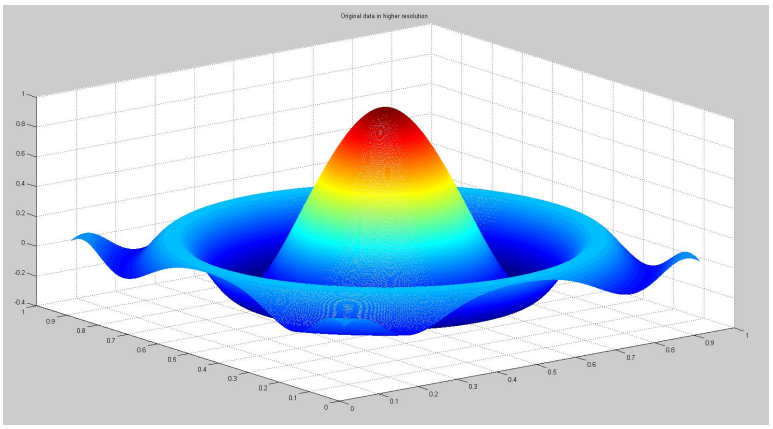

Figura 4.11: Representación de la función suave $g_1(x_1, x_2)$ usada para los experimentos numéricos en 2D

Comenzaremos obteniendo los errores que se obtienen con la reconstrucción de *Lagrange*. El orden (número de puntos usados para interpolar de manera local) y el espacio entre nodos de reconstrucción (ENR) para x e

y, irán variando dependiendo del experimento. Sin embargo, el espacio entre nodos de evaluación (ENE) para x e y permanecerá con valor constante 0.001.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma ∞***
Lagrange	4	0.1	$2{,}0571 \cdot 10^{-2}$	$2{,}6152 \cdot 10^{-2}$	$7{,}5368 \cdot 10^{-2}$
		0.05	$2{,}5891 \cdot 10^{-3}$	$3{,}5592 \cdot 10^{-3}$	$1{,}0750 \cdot 10^{-2}$
		0.025	$1{,}9439 \cdot 10^{-4}$	$2{,}6682 \cdot 10^{-4}$	$9{,}5676 \cdot 10^{-4}$
	6	0.1	$9{,}6493 \cdot 10^{-3}$	$1{,}3274 \cdot 10^{-2}$	$4{,}2633 \cdot 10^{-2}$
		0.05	$5{,}5074 \cdot 10^{-4}$	$7{,}8778 \cdot 10^{-4}$	$3{,}1860 \cdot 10^{-3}$
		0.025	$1{,}6534 \cdot 10^{-5}$	$2{,}5105 \cdot 10^{-5}$	$1{,}1176 \cdot 10^{-4}$

Tabla 4.13: Errores de la reconstrucción de *Lagrange* en 2D con la función $g_1(x_1, x_2)$. Espacio entre nodos de Evaluación en x e y: 0.001

Método	**Orden**	**ENR**	***Orden numérico***		
			Norma L1	***Norma L2***	***Norma ∞***
Lagrange	4	0.1	3,3899	3,3101	2,9442
		0.05	3,8592	3,8486	3,8059
		0.025	3,9698	3,9633	3,8951
	6	0.1	5,0439	5,0948	4,9380
		0.05	5,7913	5,7719	5,7409
		0.025	5,9445	5,9402	5,8463

Tabla 4.14: Orden numérico para la reconstrucción de *Lagrange* en 2D con la función $g_1(x_1, x_2)$. Espacio entre nodos de Evaluación en x e y: 0.001

Los resultados de dicha reconstrucción los podemos ver en la tabla 4.13. Y en la figura 4.12, está representada la reconstrucción con un orden de 4, espacio entre nodos de reconstrucción en x e y de 0.05 y un espacio entre nodos de evaluación en x e y de 0.001.

A continuación calcularemos las tablas de error para la reconstrucción de *PPH*. Los valores de orden, espaciado entre nodos de reconstrucción y evaluación serán equivalentes a los anteriores. Y los resultados se pueden ver en la tabla 4.15.

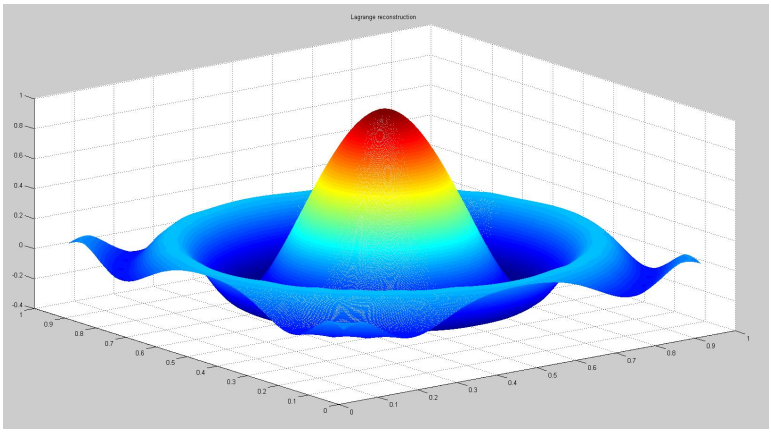

Figura 4.12: Reconstrucción de la función suave $g_1(x_1, x_2)$ por *Lagrange* en 2D

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma*** ∞
PPH	4	0.1	$3{,}2533 \cdot 10^{-2}$	$3{,}9271 \cdot 10^{-2}$	$1{,}0441 \cdot 10^{-1}$
		0.05	$5{,}3372 \cdot 10^{-3}$	$6{,}7574 \cdot 10^{-3}$	$2{,}1192 \cdot 10^{-2}$
		0.025	$4{,}4501 \cdot 10^{-4}$	$6{,}6901 \cdot 10^{-4}$	$3{,}3868 \cdot 10^{-3}$
	6	0.1	$1{,}5155 \cdot 10^{-2}$	$2{,}1128 \cdot 10^{-2}$	$6{,}5353 \cdot 10^{-2}$
		0.05	$2{,}1778 \cdot 10^{-3}$	$3{,}3049 \cdot 10^{-3}$	$1{,}5465 \cdot 10^{-2}$
		0.025	$9{,}9323 \cdot 10^{-5}$	$2{,}5586 \cdot 10^{-4}$	$2{,}5734 \cdot 10^{-3}$

Tabla 4.15: Errores de la reconstrucción *PPH* en 2D con la función $g_1(x_1, x_2)$. Espacio entre nodos de Evaluación en x e y: 0.001

En la figura 4.13 podemos ver la reconstrucción usando *PPH*. Al trabajar con una función suave, vemos como ambas reconstrucciones se aproximan mucho al original.

Los ordenes numéricos de aproximación pueden verse en la tablas 4.14 y 4.16 para las reconstrucciones de *Lagrange* y *PPH* respectivamente. Como puede verse se ajustan a lo esperado teóricamente.

Método	**Orden**	**ENR**	***Orden numérico***		
			Norma L1	***Norma L2***	***Norma ∞***
PPH	4	0.1	2,8616	2,7783	2,6348
		0.05	3,9867	3,7491	2,9604
		0.025	4,8852	4,9524	4,8714
	6	0.1	3,9239	3,7954	2,8346
		0.05	7,1317	6,6978	5,1649
		0.025	6,3899	6,9879	9,1420

Tabla 4.16: Orden numérico para la reconstrucción *PPH* en 2D con la función $g_1(x_1, x_2)$. Espacio entre nodos de Evaluación en x e y: 0.001

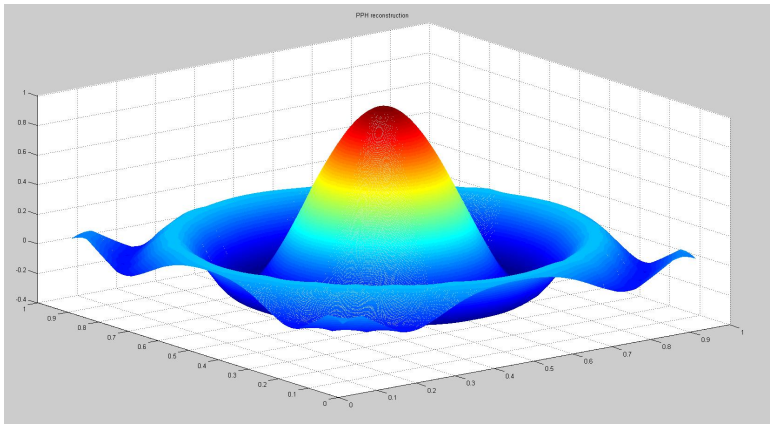

Figura 4.13: Reconstrucción de la función suave $g_1(x_1, x_2)$ por *PPH* en 2D

4.2.2. Función con discontinuidad

Finalizaremos el apartado de reconstrucciones en 2D realizando los experimentos con una función con discontinuidad cuya fórmula es la siguiente,

$$g_2(x_1, x_2) = \begin{cases} (x_1 - 0{,}5)^2 + (x_2 - 0{,}5)^2, & (x_1, x_2) \in [0,1]\backslash D, \\ \sin(x_1^2 + x_2), & (x_1, x_2) \in D, \end{cases} \tag{4.5}$$

donde D viene dado por

$$D = \{(x_1, x_2) \in \mathbb{R}^2 : (x_1 + \frac{1}{2})^2 + (x_2 + \frac{1}{2})^2 \leq \frac{1}{16}\}.$$

Podemos ver su representación gráfica en la figura 4.14.

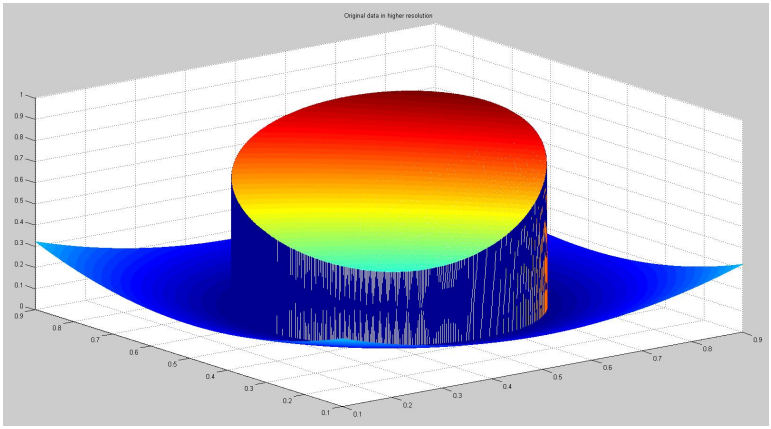

Figura 4.14: Representación de la función discontinua $g_2(x_1, x_2)$ usada para los experimentos numéricos en 2D

La tabla 4.17 muestra los resultados para la reconstrucción por *Lagrange*. Comentar que estos cálculos, al igual que en anteriores apartados están realizados con diferentes ordenes y espaciados entre nodos de reconstrucción y un espacio entre nodos de evaluación constante para toda la tabla con un valor de 0.001. Además podemos ver la gráfica resultante en la figura 4.15 al utilizar localmente 4 puntos en cada eje para construir la reconstrucción.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma ∞***
Lagrange	4	0.1	$5{,}2354 \cdot 10^{-2}$	$1{,}0046 \cdot 10^{-1}$	$6{,}0345 \cdot 10^{-1}$
		0.05	$2{,}7089 \cdot 10^{-2}$	$9{,}1468 \cdot 10^{-2}$	$8{,}3726 \cdot 10^{-1}$
		0.025	$1{,}1443 \cdot 10^{-2}$	$5{,}6925 \cdot 10^{-2}$	$8{,}2740 \cdot 10^{-1}$
	6	0.1	$8{,}6733 \cdot 10^{-2}$	$1{,}3302 \cdot 10^{-1}$	$6{,}0901 \cdot 10^{-1}$
		0.05	$3{,}7323 \cdot 10^{-2}$	$1{,}0441 \cdot 10^{-1}$	$8{,}4002 \cdot 10^{-1}$
		0.025	$1{,}3900 \cdot 10^{-2}$	$6{,}0924 \cdot 10^{-2}$	$8{,}3269 \cdot 10^{-1}$

Tabla 4.17: Errores de la reconstrucción de *Lagrange* en 2D con la función $g_2(x_1, x_2)$. Espacio entre nodos de Evaluación en x e y: 0.001

Se puede observar que la reconstrucción de *Lagrange* produce ahora oscilaciones alrededor de la discontinuidad y un suavizado inexistente. Por

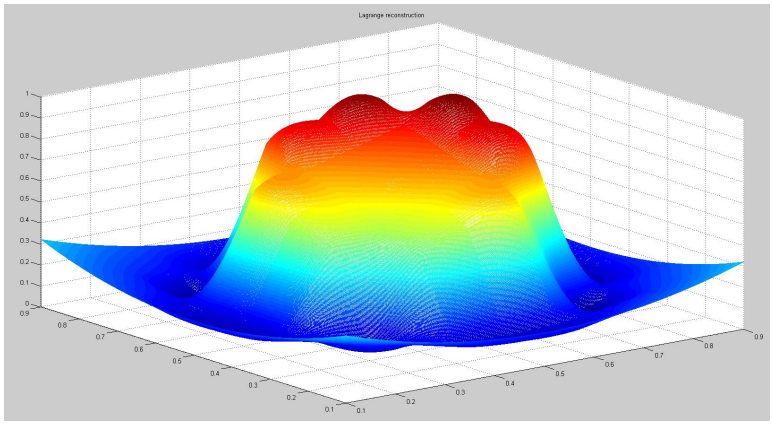

Figura 4.15: Reconstrucción de la función discontinua $g_2(x_1, x_2)$ por *Lagrange* en 2D

dicha razón los resultados numéricos ofrecidos en la Tabla 4.17 no son buenos. El orden numérico en este caso no será 4, ya que trabajamos con funciones discontinuas y como ya dijimos en el caso de 1D, no tiene mucho sentido hablar de orden con funciones discontinuas.

Ahora pasamos a analizar los resultados dados por la reconstrucción *PPH*. Esta reconstrucción ha sido realizada con un orden, espacio entre nodos de reconstrucción y evaluación igual a la anterior de *Lagrange*. Los resultados numéricos podemos verlos en la tabla 4.18. Y su representación gráfica en la figura 4.16.

Método	**Orden**	**ENR**	***Error***		
			Norma L1	***Norma L2***	***Norma ∞***
PPH	4	0.1	$4{,}4864 \cdot 10^{-2}$	$1{,}0435 \cdot 10^{-1}$	$6{,}3871 \cdot 10^{-1}$
		0.05	$2{,}2665 \cdot 10^{-2}$	$9{,}1239 \cdot 10^{-2}$	$8{,}3476 \cdot 10^{-1}$
		0.025	$9{,}6204 \cdot 10^{-3}$	$5{,}7102 \cdot 10^{-2}$	$8{,}1568 \cdot 10^{-1}$
	6	0.1	$7{,}6003 \cdot 10^{-2}$	$1{,}3274 \cdot 10^{-1}$	$5{,}9952 \cdot 10^{-1}$
		0.05	$2{,}8504 \cdot 10^{-2}$	$1{,}0147 \cdot 10^{-1}$	$8{,}3214 \cdot 10^{-1}$
		0.025	$1{,}0541 \cdot 10^{-2}$	$5{,}9386 \cdot 10^{-2}$	$8{,}1810 \cdot 10^{-1}$

Tabla 4.18: Errores de la reconstrucción *PPH* en 2D con la función $g_2(x_1, x_2)$. Espacio entre nodos de Evaluación en x e y: 0.001

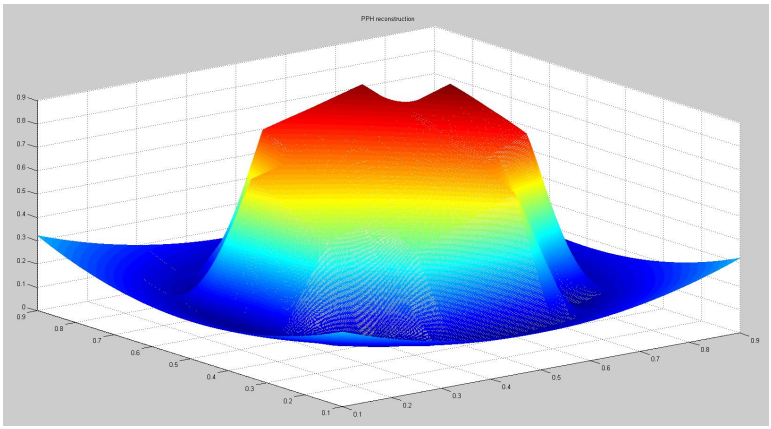

Figura 4.16: Reconstrucción de la función discontinua $g_2(x_1, x_2)$ por *PPH* en 2D

Como se puede observar comparando ambas reconstrucciones, la que obtenemos por *Lagrange* presenta el efecto de Gibbs en los extremos de las discontinuidades, mientras que la efectuada por *PPH* se ajusta mejor a la función original en las discontinuidades. Para este experimento la frontera debido a la discontinuidad parece no definirse demasiado bien tampoco para el método de reconstrucción *PPH*. Esto es debido a que el número de datos inicial es insuficiente para definirla correctamente. No hay más que ver los datos originales, que tienen poca resolución, en la figura 4.17.
En la figura 4.18 representamos la reconstrucción obtenida ahora con cuatro veces más nodos de reconstrucción por eje. Puede verse que esta vez la reconstrucción define mucho mejor la discontinuidad.

Por tanto, podemos concluir diciendo que si la función no presenta discontinuidades ambos métodos de reconstrucción son válidos, pero si la función original presenta una discontinuidad es mejor el ajuste ofrecido por la reconstrucción *PPH*.

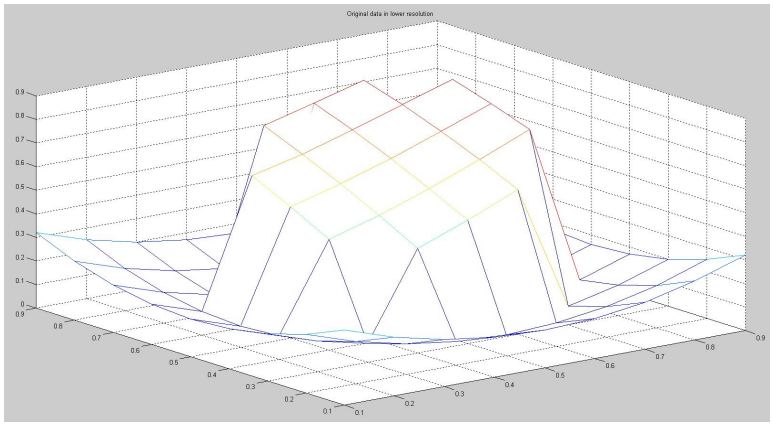

Figura 4.17: Representación de la función discontinua $g_2(x_1, x_2)$ usada para los experimentos numéricos en 2D en baja resolución

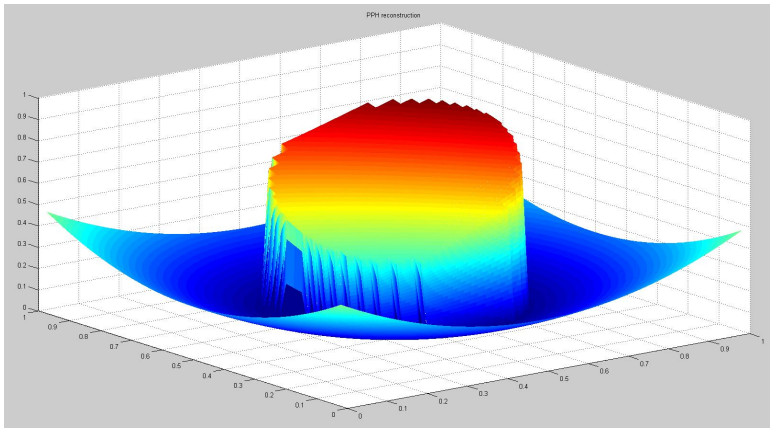

Figura 4.18: Reconstrucción de la función discontinua $g_2(x_1, x_2)$ por *PPH* en 2D con cuatro veces más nodos de reconstrucción por eje

4.3. Experimentos con datos en 2D: Zoom.

En esta sección vamos realizar un zoom de datos los cuales podrían representar una elevación del terreno o la profundidad de una región determinada. Dichos datos pueden verse en la figura 4.19.

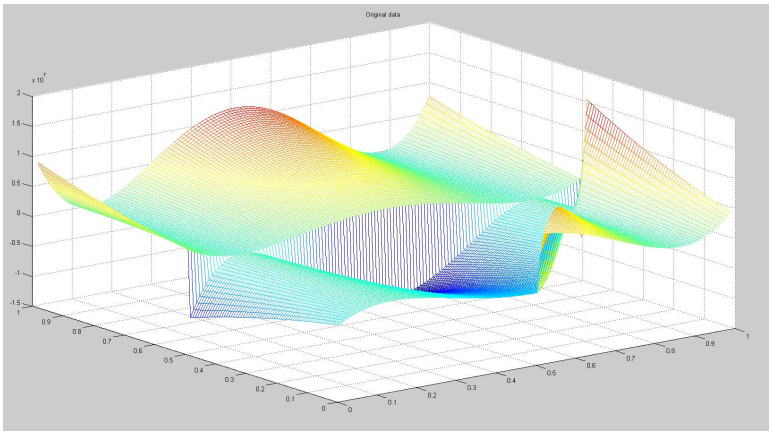

Figura 4.19: Datos originales en 2D seleccionados para los experimentos

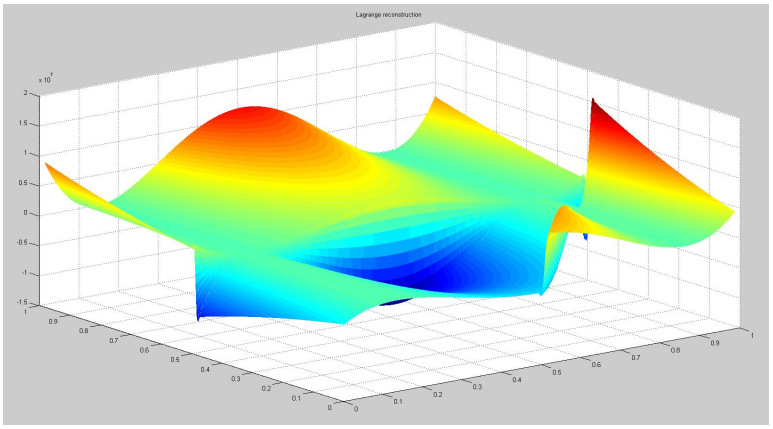

Figura 4.20: Reconstrucción por el método de *Lagrange* de los datos seleccionados

En la figura 4.20 podemos ver el resultado que nos proporciona la reconstrucción de *Lagrange*. Después se ha rotado el resultado para mostrar

un mejor ángulo de la discontinuidad, y como trabaja dicha reconstrucción cuando se enfrenta a estas singularidades. Dicha rotación puede observarse en la figura 4.21. Y en la figura 4.22 se muestra un zoom donde podemos observar claramente el efecto de Gibbs que presentan las reconstrucciones realizadas por el método de *Lagrange*.

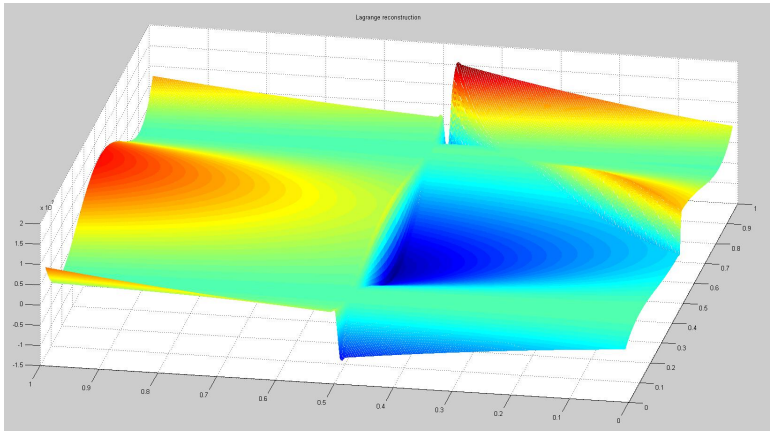

Figura 4.21: Reconstrucción por el método de *Lagrange* de los datos seleccionados, girada para mostrar el detalle de la discontinuidad

En las siguientes figuras se ha realizado el mismo proceso pero esta vez para la reconstrucción *PPH*. En la figura 4.23 podemos ver la reconstrucción. Y en las figuras 4.24 y 4.25 vemos la reconstrucción girada y un zoom de la discontinuidad, respectivamente. Podemos apreciar como la reconstrucción ha mejorado y se ha reducido el efecto de Gibbs que si se daba en la reconstrucción realizada por *Lagrange*. Es interesante comparar dichos resultados con los datos originales. A estos efectos, en la figura 4.26 vemos un zoom de la misma zona correspondiente a la discontinuidad. Como puede observarse la reconstrucción *PPH* conserva mejor el perfil original.

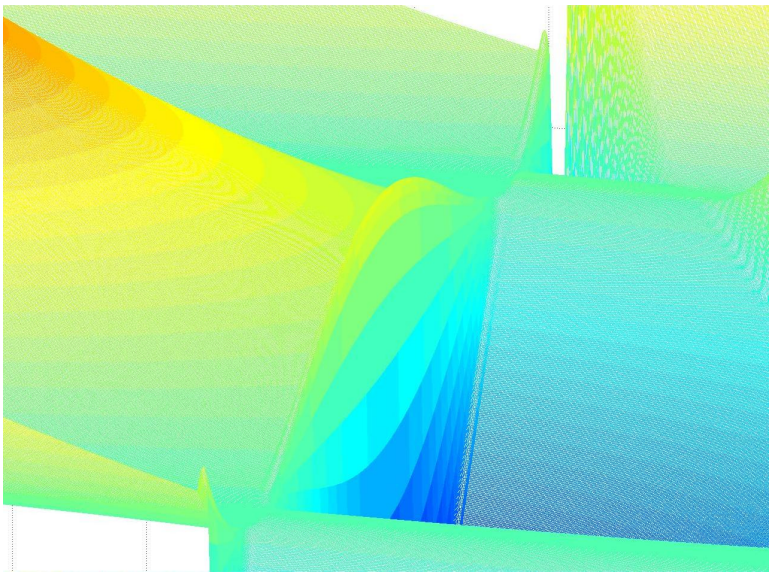

Figura 4.22: Zoom de la figura 4.21. Reconstrucción de *Lagrange*

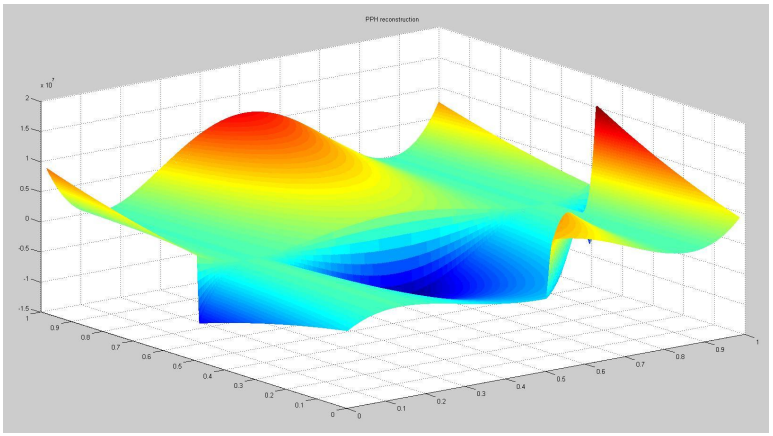

Figura 4.23: Reconstrucción por el método de *PPH* de los datos seleccionados

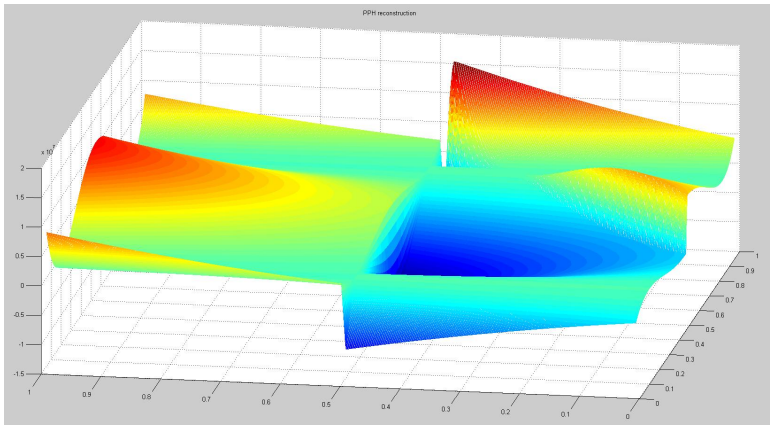

Figura 4.24: Reconstrucción por el método de *PPH* de los datos seleccionados, girada para mostrar el detalle de la discontinuidad

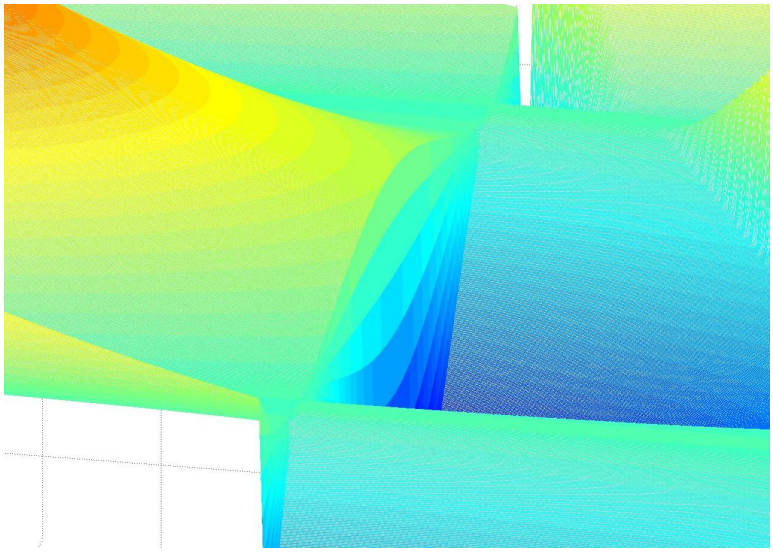

Figura 4.25: Zoom de la figura 4.24. Reconstrucción *PPH*

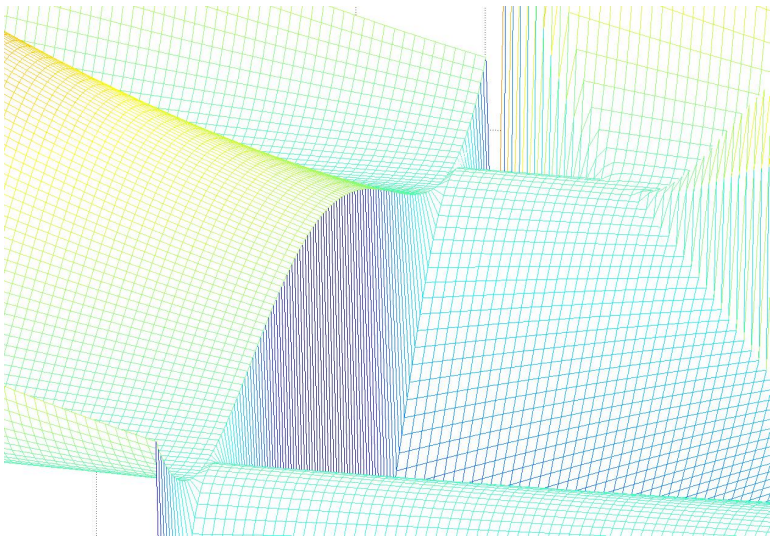

Figura 4.26: Zoom de la región de la discontinuidad de los datos originales en 2D.

Capítulo 5

Interfaz gráfica de usuario

Para poder probar los algoritmos numéricos implementados de manera más rápida y fiable se han desarrollado una serie de interfaces gráficas en el entorno de desarrollo GUIDE *(Graphical User Interface Development Environment)* que nos proporciona Matlab. Además, estas interfaces han sido testeadas ante posibles fallos que pueda cometer el usuario al introducir los distintos valores. También son fácilmente ampliables ante la eventualidad de añadir nuevas funciones y métodos de reconstrucción.

5.1. Ejecución de la interfaz gráfica

Para ejecutar la interfaz gráfica principal, debemos introducir en la linea de comandos de Matlab

$$>> \textit{Reconstrucciones}$$

Antes de ejecutar dicha instrucción debemos asegurarnos de que nos encontramos en el directorio que contiene la GUI, en caso contrario tendremos que cambiar nuestro directorio de trabajo por aquel que la contenga.
Al ejecutar este comando, nos aparecerá la ventana que se muestra en la Figura 5.1.

Figura 5.1: Interfaz Gráfica De Usuario

5.2. Documentación de la interfaz gráfica

Esta primera interfaz dispone de dos botones que nos enlazan con otras interfaces, uno para trabajar con reconstrucciones en aritmética normal y otro para trabajar con aritmética variable. Cada uno de ellos nos mostrará una nueva interfaz compuesta a su vez por otros tantos botones, que detallaremos a continuación. Ambas interfaces son análogas, y por tanto, solamente será necesaria la descripción de una de ellas.

Esta nueva interfaz se encuentra dividida en dos menús, un primer menú para reconstrucciones en una dimensión y otro para reconstrucciones en dos dimensiones. A su vez ambos menús cuentan con una serie de opciones, que permiten trabajar con funciones o con datos, y seleccionar el método de reconstrucción deseado. Cada uno de estas opciones abre una interfaz gráfica diferente. Para no extendernos en exceso en este capítulo pasaremos a describir la funcionalidad de dos de ellas. Hemos escogido las que nos proporcionan una reconstrucción de funciones en una dimensión y en dos dimensiones respectivamente, ambas con aritmética normal.

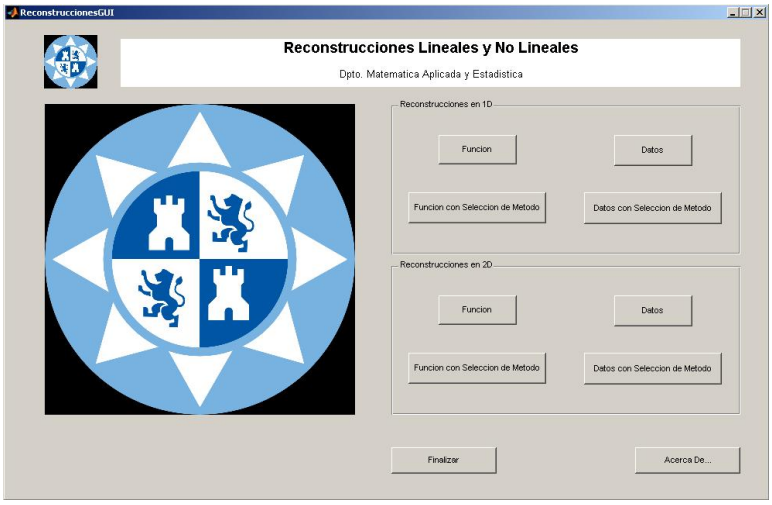

Figura 5.2: Interfaz selección del tipo de reconstrucción

5.3. La interfaz gráfica en 1D

Empezaremos describiendo la interfaz que nos proporciona una reconstrucción de funciones en una dimensión.

5.3.1. El menú *Selección*

Comenzaremos con el menú *Selección*, en el cual escogemos una función que posteriormente reconstruiremos, además, también daremos valor a sus discontinuidades (en el caso de que las tuviera). Este menú está dividido a su vez en dos submenús. El primero de ellos es el submenú *Función*, y el segundo, el submenú *Discontinuidad*. Ambos serán descritos con detalle a continuación. Además este menú también cuenta con una ventana capaz de representar la función seleccionada anteriormente.

Los submenús *Función* y *Discontinuidad*

A continuación detallaremos todas las funciones de los submenús contenidos en el menú *Selección*.

El submenú *Función* está compuesto por un menú desplegable, y dos botones, uno para la edición y otro para la carga de funciones. Comenzare-

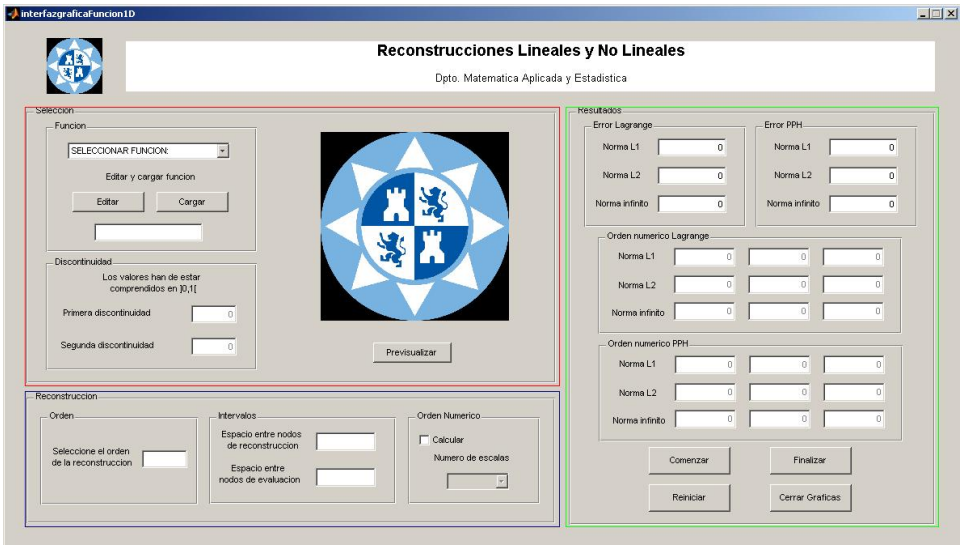

Figura 5.3: GUI para la generación de reconstrucciones en 1D.

mos con la descripción del menú desplegable, el cual mostrará las funciones que podemos usar, y bastará con seleccionar una para trabajar con ella.

El botón *Editar* nos abre el documento *fun_type.m*, el cual representa una función vacía, que debemos modificar en las líneas señaladas para crear nuestra nueva función.

Y por último, se encuentra el botón *Cargar*, que abrirá la ventana que muestra la Figura 5.4, donde seleccionaremos la función creada previamente en Matlab.

Ahora pasaremos a describir la funcionalidad del submenú *Discontinuidad*. La tarea de éste es que el usuario dé valor a la/s discontinuidad/es de la función seleccionada (en el caso de que las tuviera). Cabe resaltar que, dependiendo de la función seleccionada anteriormente, estarán activos los distintos cuadros de texto, como podemos ver en la Figura 5.5, esto también es aplicable si la función ha sido cargada manualmente mediante el botón *Cargar*.

Para terminar con este menú, comentar que, una vez seleccionada la función y dado valor a sus discontinuidades (en el caso de que las tuviera),

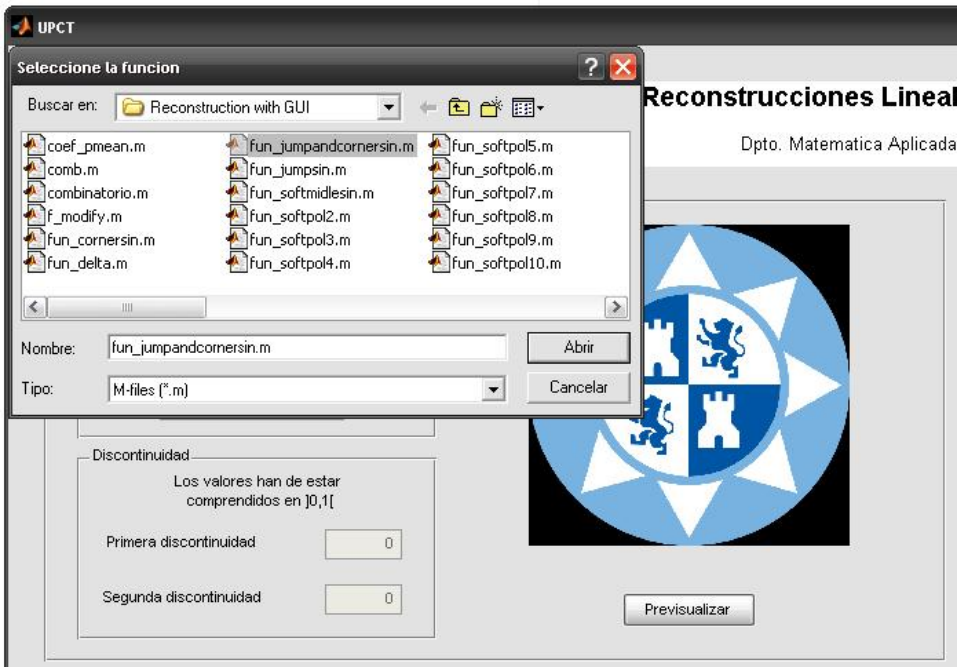

Figura 5.4: Ventana de selección de funciones

podemos obtener su representación pulsando sobre el botón *Previsualizar*, como se muestra en Figura 5.6.

5.3.2. El menú *Reconstrucción*

El siguiente es el menú *Reconstrucción*. Este menú al igual que el anterior también está dividido en varios submenús. En este menú seleccionaremos el orden de la reconstrucción, el espaciado entre los nodos que definen los diferentes intervalos y escogeremos el número de escalas para calcular el valor del orden numérico de la reconstrucción.

El submenú *Orden*

Este submenú es el primero con el que nos encontramos dentro del menú *Reconstrucción*, y en él indicaremos el orden de nuestra reconstrucción. Está compuesto por un cuadro de texto en el que se debe introducir un

Figura 5.5: Cuadros de texto activos para introducir el valor de las discontinuidades

valor par y mayor que cero, de lo contrario la Interfaz Gráfica mostrará un error, como el de la Figura 5.7.

El submenú *Intervalos*

El segundo es el submenú *Intervalos*. Este submenú está compuesto por dos cuadros de texto. En el primero introducimos el valor del espaciado entre nodos de reconstrucción. Notar que el intervalo donde consideramos las funciones es $[0, 1]$, y por tanto el espaciado requerido debe ser estrictamente menor que 1. Cuanto menor sea este valor, mayor el número de nodos de reconstrucción y por tanto mejor nuestra reconstrucción por trozos.
En el segundo, introduciremos el valor del espaciado entre nodos de evaluación, que nos indica el número de puntos donde evaluamos la reconstrucción entre cada dos nodos de reconstrucción, por lo que este valor deberá ser menor que el anterior.

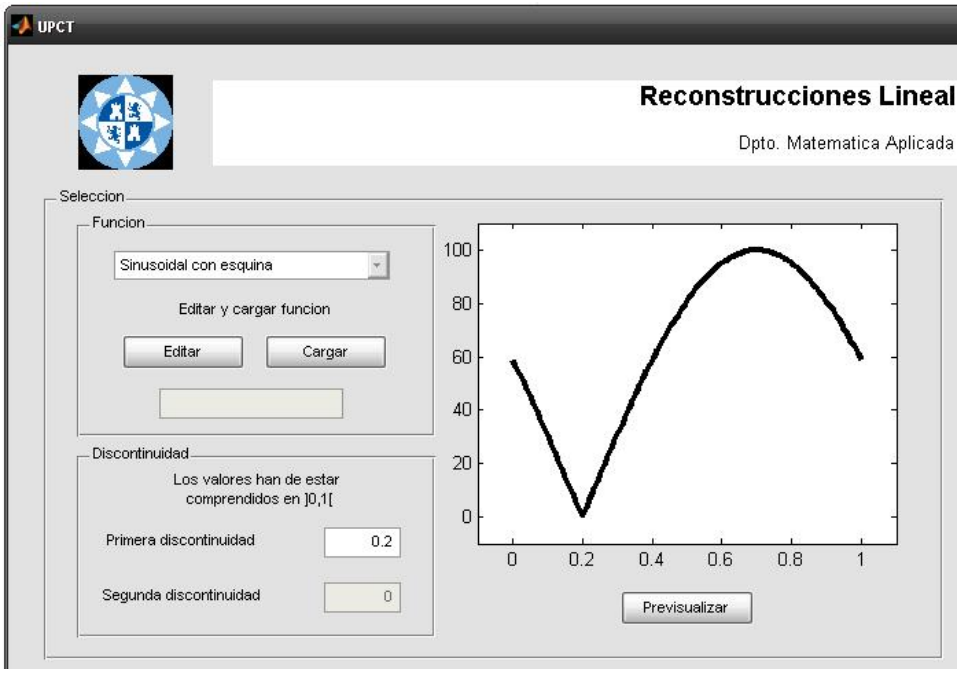

Figura 5.6: Previsualización de la función

El submenú *Orden Numérico*

Este es el último de los submenús y está compuesto por un checkbox y un menú desplegable. Con el checkbox habilitaremos/deshabilitaremos el menu desplegable, con el cual escogemos el número de escalas que se desean calcular. Al escoger la escala se habilitarán/deshabilitarán en el menú *Resultados* varios cuadros de texto (por defecto desactivados) en donde se mostrarán los resultados. Con esto hemos finalizamos la descripción del menú *Reconstrucción*.

5.3.3. El menú *Resultados*

Finalizaremos este apartado describiendo el menú *Resultados*. Este menú está compuesto por cuatro submenús, pero a diferencia de los anteriores, en este menú no introduciremos valores, solamente visualizaremos los resultados obtenidos. Ahora describiremos más detalladamente cada uno de los submenús.

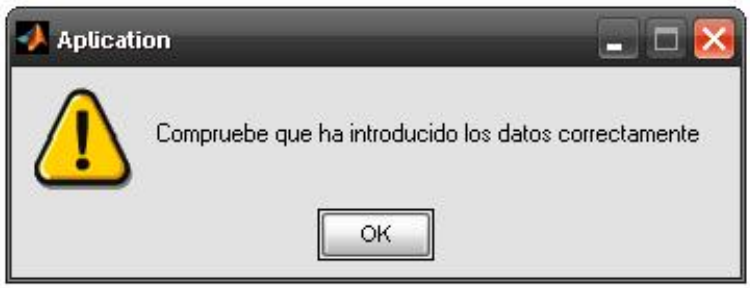

Figura 5.7: Ventana de error

Los submenús *Error Lagrange* y *Error PPH*

El primero de ellos es el submenú *Error Lagrange*, que como su nombre indica calcula el error cometido por la aproximación de *Lagrange* y nos muestra el resultado para las distintas normas.

El segundo es el submenú *Error PPH*, que al igual que el anterior, calcula el error cometido, pero en este caso por la aproximación *PPH* y muestra los resultados.

Los submenús *Orden Numérico Lagrange* y *Orden Numérico PPH*

Los dos siguientes submenús están compuestos por tres columnas de cuadros de texto que están inicialmente desactivados. Estos cuadros de texto representan las tres escalas que, como máximo, se pueden calcular en esta interfaz, y serán activados/desactivados dependiendo del orden que hayamos seleccionado en el submenú *Orden Numérico* del menú *Reconstrucción*.

Para finalizar con la descripción, solamente nos falta comentar los botones que se encuentran en la parte inferior derecha de la interfaz. Empezaremos por el botón *Comenzar*, sobre el que pulsaremos para comenzar la reconstrucción una vez que hayamos introducido todos los valores que se nos piden. Cuando finalice, además de los resultados en sus correspondientes cuadros de texto, también se nos mostrará una ventana como la de la Figura 5.8. Esta ventana nos muestra en su primera figura la función seleccionada y su reconstrucción por el algoritmo de *Lagrange* y en la segunda figura la función seleccionada y su reconstrucción por el algoritmo de *PPH*. El botón *Finalizar* cerrará la interfaz en la que nos encontramos trabajando. El botón

Reiniciar se encarga de poner los valores iniciales en todos los cuadros de texto de la interfaz, y por último el botón *Cerrar Gráficas* se encarga de cerrar las ventanas con las gráficas resultantes.

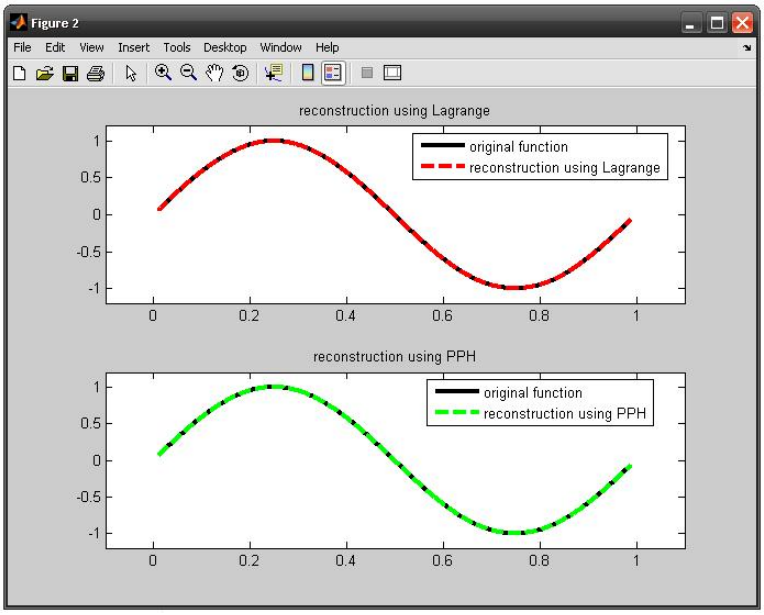

Figura 5.8: Ventana con la función reconstruida por *Lagrange* y *PPH*

5.4. La interfaz gráfica en 2D

Ahora pasamos a describir el funcionamiento de la segunda interfaz, la que nos proporciona reconstrucciones de funciones en 2D. Al ser esta interfaz muy similar a la descrita en el apartado anterior, solamente nos centraremos en las diferencias que presenta ésta última frente a la ya descrita.

5.4.1. El menú *Selección*

Este primer menú se divide en los siguientes tres submenús, *Función Principal*, *Discontinuidad* y *Segunda Función*. Tan sólo el primero de ellos está activo al iniciar la interfaz. Ahora pasaremos a detallar cada uno de estos submenús.

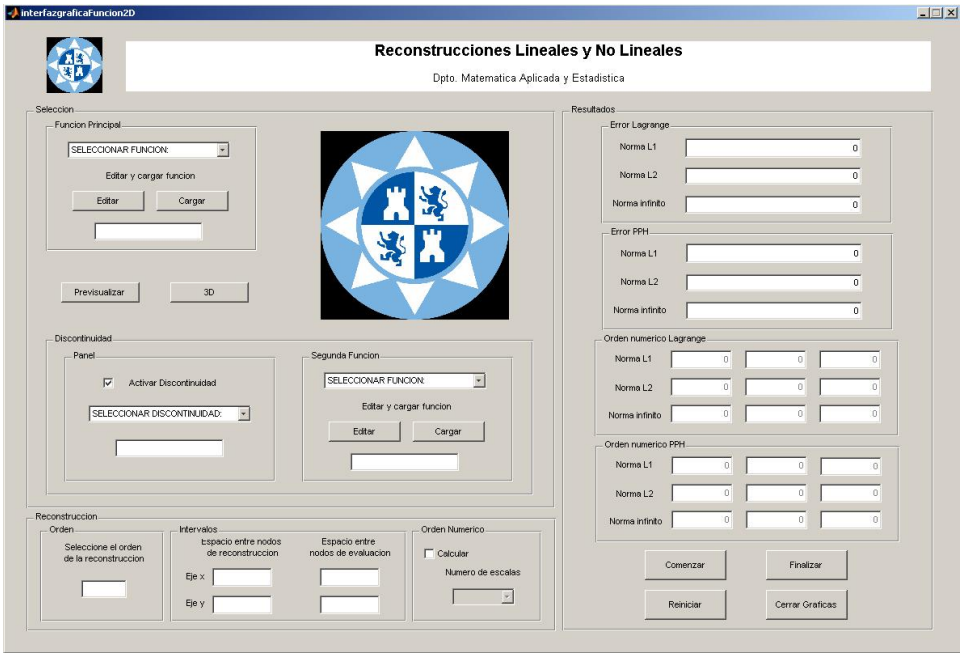

Figura 5.9: GUI para reconstrucciones en 2D

Los submenús *Función Principal*, *Discontinuidad* y *Segunda Función*

El primer submenú que vamos a analizar es el submenú *Función Principal*. Al igual que en la interfaz de 1D, está compuesto por un menú desplegable para seleccionar la función que posteriormente usaremos en la reconstrucción y dos botones para la edición y la carga de funciones.
El siguiente es el submenú *Discontinuidad*, compuesto por un checkbox que activa y desactiva los menús desplegables y los cuadros de texto de los submenús *Discontinuidad* y *Segunda Función*. Como hemos dicho antes este submenú también está compuesto por un menú desplegable que dispone de unas funciones predeterminadas para la selección de la discontinuidad y un cuadro de texto donde introducir manualmente la discontinuidad que deseemos.
Por último, decir del submenú *Segunda Función*, que es semejante al submenú *Función Principal*, por lo que no será necesaria una descripción de él.

5.4.2. El menú *Reconstrucción*

Este menú está compuesto por otros tres submenús, que son exactamente iguales a los de la interfaz en 1D, con el añadido, de seleccionar el espaciado entre nodos de evaluación y de reconstrucción tanto para el eje x como para el eje y.

5.4.3. El menú *Resultados*

Por ultimo se encuentra el menú *Resultados*, que nos dará la misma información que en su versión para 1D.

Finalizaremos este capítulo comentando que el uso de los botones de la parte inferior derecha de la interfaz es análogo al de la interfaz anteriormente analizada.

Capítulo 6

Códigos en Matlab

En esta sección vamos a ver los códigos Matlab que se programaron y que usamos para realizar los experimentos numéricos de la sección anterior.

6.1. Códigos con *aritmética normal*

6.1.1. Códigos en 1D

Vamos a empezar esta sección incluyendo los códigos que se usan en las reconstrucción con funciones.

```
function [xa, ap1, ap2, error_lag, error_pph, r_lag, r_pph] = ...
    function_reconstruction (n, inc1, inc2, fhandle, varargin)

% This function build an approximation to a function f defined
%     in the program using piecewise reconstructions
% [xa, ap1, ap2, error_lag, error_pph, r_lag, r_pph] = ...
%     function_reconstruction (n, inc1, inc2, fhandle, dis, sc)
% Input variables:
```

```
% n reconstruction order
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fhandle considered function to be approximated
% dis is used to indicate the location of the discontinuities
%      in the function
% sc optional parameter which indicates the number of scales to be
%      computed to obtain the numerical order of the reconstructions
% Output variables:
% xa points where the function f is approximated
% ap1 approximation at the points xa using Lagrange reconstruction
% ap2 approximation at the points xa using PPH reconstruction
% error_lag error in lagrange approximation in l1, l2 and infinite norm
% error_lag error in pph approximation in l1, l2 and infinite norm

% It close the previous figures
figure(2);
close

% This separates the different variables that contains varargin
x = 0 : inc1 : 1;
nx = length(x);
dis = [];
sc = 0;
param = nx;
if nargin == 5
    if varargin{1} < 1
        dis = varargin{1};
        param = [nx, dis];
    else
        param = nx;
        sc = varargin{1};
    end
elseif nargin == 6
    dis = varargin{1};
    param = [nx, dis];
    sc = varargin{2};
end

% Load function values
```

```
[nod, f] = fhandle (param);

% Abscisas where we approximate
x1 = x(1) : inc2 : x(nx);
xa = x1(find(x1 >= x(n / 2) & x1 <= x(nx - n / 2 + 1)));

% Loop to build the reconstruction piecewise
% It calculates the first element to the reconstruction
aux = xa(find(xa >= x(n / 2) & xa < x(n / 2 + 1)));
auxf = lagrange(f(1 : n), x(1 : n), aux);
ap1 = auxf;
auxf = pphpower(f(1 : n), x(1 : n), aux);
ap2 = auxf;
clear auxf;
clear aux;
% It calculates the others elements to the reconstruction
for i = n / 2 + 1 : nx - n / 2 - 1
    aux = xa(find(xa >= x(i) & xa < x(i + 1)));
    auxf = lagrange(f(i - n / 2 + 1 : i + n / 2), ...
        x(i - n / 2 + 1 : i + n / 2), aux);
    ap1 = [ap1, auxf];
    auxf = pphpower(f(i - n / 2 + 1 : i + n / 2), ...
        x(i - n / 2 + 1 : i + n / 2), aux);
    ap2 = [ap2, auxf];
    clear auxf;
    clear aux;
end
% It calculates the last element of the reconstruction
aux = xa(find(xa >= x(nx - n / 2) & xa <= x(nx - n / 2 + 1)));
auxf = lagrange(f(nx - n + 1 : nx), x(nx - n + 1 : nx), aux);
ap1 = [ap1, auxf];
auxf = pphpower(f(nx - n + 1 : nx), x(nx - n + 1 : nx), aux);
ap2 = [ap2, auxf];
clear auxf;
clear aux;

% Measurement of the approximation error
nx1=length(x1); param(1)=nx1;
[nod,f1] = fhandle (param);
nxa = length(xa);
```

```
f1 = f1((nx1-nxa)/2+1:nx1-(nx1-nxa)/2);

% It calculate the error in Lagrange approximation
nap1=length(ap1);
e1_lag = norm(ap1 - f1, 1)/nap1;
e2_lag = norm(ap1 - f1, 2)/nap1;
einf_lag = norm(ap1 - f1, 'inf');
error_lag = [e1_lag, e2_lag, einf_lag];

% It calculate the error in PPH approximation
nap2=length(ap2);
e1_pph = norm(ap2 - f1, 1)/nap2;
e2_pph = norm(ap2 - f1, 2)/nap2;
einf_pph = norm(ap2 - f1, 'inf');
error_pph = [e1_pph, e2_pph, einf_pph];

% We determine the approximation order of the reconstructions
if sc > 0
    if isempty(dis)
        [r_lag, r_pph] = order (sc, n, inc1, inc2, fhandle);
    else
        [r_lag, r_pph] = order (sc, n, inc1, inc2, fhandle, dis);
    end
end

% Plot of the approximation and the original function
figure(2)
hold on
% It computes the axis for the plot
    axm = min([ap1, ap2, f1]);
    axM = max([ap1, ap2, f1]);
    dif_axis = 0.1 * (axM - axm);
    % It draw the Lagrange reconstruction
    subplot(2, 1, 1), plot(xa, f1, '-k', 'LineWidth', 3), ...
        title('Reconstruction Using Lagrange');
    hold on
    subplot(2, 1, 1), plot(xa, ap1, '–r', 'LineWidth', 3);
    subplot(2, 1, 1), legend('Original Function', ...
        'Reconstruction Using Lagrange', 0);
    axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);
```

```
    %It draw the PPH reconstruction
    subplot(2, 1, 2), plot(xa, f1, '-k', 'LineWidth', 3), ...
        title('Reconstruction Using PPH');
    hold on
    subplot(2, 1, 2), plot(xa, ap2, '–g', 'LineWidth',3);
    subplot(2, 1, 2), legend('Original Function', ...
        'Reconstruction Using PPH', 0);
    axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);



function y = lagrange(f, nod, x)

% This function computes the lagrange reconstruction in one interval
% y = lagrange(f, nod, x)
% Input variables:
% f vector with the function values at the nodes
% nod vector with the nodes
% x set of abcisas to build the approximation
n = length(f);
m = length(x);
b = zeros(n, m);
for i=1:n
    b(i,1:m)=ones(1,m);
    for j=1:n
        if(j ~= i)
            b(i,1:m) = b(i,1:m).*(x-nod(j)*ones(1, m))/(nod(i)-nod(j));
        end
    end
end
y=zeros(1, length(x));
for i=1:n
    y(1:m)=y(1:m)+b(i,1:m)*f(i);
end

return
```

```
function [pwey, f1] = pphpower(f, nod, x)

% This function computes the pphpower reconstruction in one interval
% [pwey, f1] = pphpower(f, nod, x);
% Input variables:
% f vector with the function values at the nodes
% nod vector with the nodes
% x vector with the abscisas where we want to compute the reconstruction
% Output variables:
% pwey values of the reconstruction in the abscisas x
% f1 modified function values

inc = nod(2) - nod(1);
f1 = f_modify(f, inc);
pwey = lagrange(f1, nod, x);

return


function fm = f_modify(f, inc)

% This function computes the modified values for the pphpower
%      reconstruction from the initial values of the function f
% fm = f_modify(f);
% Input variables:
% f vector with the discrete values of a function f(x)
% inc step between partition nodes
% Output variables:
% fm vector with the discrete modified values

n = length(f);
% Loop for the recursive modifications
    for i = 1 : n / 2 - 1
        % Divided difference to the left
        vl = f(n / 2 - i : n / 2 + i);
        dl = diff(vl, 2 * i);
        % Divided difference to the right
        vr = f(n / 2 - i + 1 : n / 2 + i + 1);
        dr = diff(vr, 2 * i);
        coef = comb (2 * i);
```

```
        coef = [coef, fliplr(coef)];
        pmean = pwe(n - 2 * i, dl , dr, inc);
        if abs(dl) <= abs(dr)
            f(n / 2 + i + 1) = - f(n / 2 - i) + ...
                sum(coef .* f(n / 2 - i + 1 : n / 2 + i)) + 2 * pmean;
        else
            f(n / 2 - i) = - f(n / 2 + i + 1) + ...
                sum(coef .* f(n / 2 - i + 1 : n / 2 + i)) + 2 * pmean;
        end
    end
    fm = f;
    return


function c = comb(n)

% This is an auxiliary function which gives the coefficients of the
%      expansion of the modified values in the pphpower reconstruction
% c = comb(n);
% Input variables:
% n length of the vector of coefficients; it must be even
% Output variables:
% c vector with the coefficients of the modified expression

for i = 1 : n / 2
    c(i) = (-1)^(i+1)*(combinatorio(n, i)-combinatorio(n,i-1));
end


function c = combinatorio(n, m)

% This function computes the combinatorial number of n over m
% c = combinatorio(n, m)

if (n == m | m == 0)
    c = 1;
else
    c = combinatorio(n - 1, m) + combinatorio(n - 1, m - 1);
end
```

```
function med = pwe (p, x, y, inc)

% This function computes the p-mean of two quantities
% med = pwe (p, x, y, inc);
% Input variables:
% p order of the p-mean
% x, y numbers to compute the p-mean
% inc step between partition nodes
% Output variables:
% med value of the p-mean

% We create the cases to evaluate them with a switch
    if (x * y > 0)
        seleccion = 1;
    elseif x ~= 0 & y ~= 0
        seleccion = 2;
    else
        seleccion = 3;
    end
     % We evaluate the diferent cases
    switch seleccion
         % X and Y are mayor that zero
    case 1
         % When one of the variables is major that inc
        if abs(x) > inc | abs(y) > inc
            r = x - y;
            s = x + y;
            med = s / 2 * (1 - abs(r / s) ^ p);
         % When both varibles are minor
        else
            x = x + 1;
            y = y + 1;
            r = x - y;
            s = x + y;
            med = s / 2 * (1 - abs(r / s) ^ p);
            med = med - 1;
        end
     % X and Y different from zero
```

```
case 2
    % When one of the variables is major that inc
    if abs(x) > inc | abs(y) > inc
        [var1, ind] = min (abs([x, y]));
        % X is the minimum
        if ind == 1
            sg = sign(y);
            x1 = x + 2 * sg * var1;
            y1 = y + 2 * sg * var1;
        % Y is the minimum
        else
            sg = sign(x);
            x1 = x + 2 * sg * var1;
            y1 = y + 2 * sg * var1;
        end
        r = x1 - y1;
        s = x1 + y1;
        med = s / 2 * (1 - abs(r / s) ^ p);
        med = med - 2 * sg * var1;
    % When both variables are minor
    else
        [var1,ind] = min(abs([x, y]));
        % X is the minimum
        if ind == 1
            sg = sign(y);
            x1 = x + 2 * sg * var1 + sg * 1;
            y1 = y + 2 * sg * var1 + sg * 1;
        % Y is the minimum
        else
            sg = sign(x);
            x1 = x + 2 * sg * var1 + sg * 1;
            y1 = y + 2 * sg * var1 + sg * 1;
        end
        r = x1 - y1;
        s = x1 + y1;
        med = s / 2 * (1 - abs(r / s) ^ p);
        med = med - 2 * sg * var1 - sg * 1);
    end
% Other cases
case 3
```

```
        if x == 0 & y == 0
            med = 0;
        elseif x == 0
            sg = sign(y);
            x1 = sg * 1;
            y1 = y + sg * 1;
            r = x1 - y1;
            s = x1 + y1;
            med = s / 2 * (1 - abs(r / s) ^ p);
            med = med - sg * 1;
        else
            sg = sign(x);
            x1 = x + sg * 1;
            y1 = sg * 1;
            r = x1 - y1;
            s = x1 + y1;
            med = s / 2 * (1 - abs(r / s) ^ p);
            med = med - sg * 1;
        end
end



function [r_lag, r_pph] = order (sc, n, inc1, inc2, fhandle, dis)

% This function computes numerically the approximation order of the
% piecewise Lagrange and PPH reconstructions
% [r_lag, r_pph]=order(sc, n, inc1, inc2, fhandle, dis);
% Input variables:
% sc times that we subdivide the step size inc1
% n reconstruction order
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition. Be careful to take
%      it much more smaller indent than inc1
% fhandle considered function to be approximated
% dis is used to indicate the location of the discontinuities in the
%      function
% Output variables:
% r_lag numerical approximation orders for the piecewise Lagrange
%      reconstruction in l1,l2 and infinite norms
```

```
% r_pph numerical approximation orders for the piecewise PPH
%      reconstruction in l1,l2 and infinite norms

if nargin==6

    % The first case is done before the loop
    [xa, error_lag1, error_pph1] = reconstructionsimple ...
        (n, inc1, inc2, fhandle, dis);

    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1=inc1/2;

        [xa, error_lag, error_pph] = reconstructionsimple ...
            (n, inc1, inc2, fhandle, dis);

        r1_lag(i) = log2 (error_lag1(1)/error_lag(1));
        error_lag1(1) = error_lag(1);
        r2_lag(i) = log2 (error_lag1(2)/error_lag(2));
        error_lag1(2) = error_lag(2);
        r3_lag(i) = log2 (error_lag1(3)/error_lag(3));
        error_lag1(3) = error_lag(3);

        r1_pph(i) = log2 (error_pph1(1)/error_pph(1));
        error_pph1(1) = error_pph(1);
        r2_pph(i) = log2 (error_pph1(2)/error_pph(2));
        error_pph1(2) = error_pph(2);
        r3_pph(i) = log2 (error_pph1(3)/error_pph(3));
        error_pph1(3) = error_pph(3);

    end

else

    % The first case is done before the loop
    [xa, error_lag1, error_pph1] = reconstructionsimple ...
        (n, inc1, inc2, fhandle);
```

```
    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1=inc1/2;

        [xa, error_lag, error_pph] = reconstructionsimple ...
            (n, inc1, inc2, fhandle, dis);

        r1_lag(i) = log2 (error_lag1(1)/error_lag(1));
        error_lag1(1) = error_lag(1);
        r2_lag(i) = log2 (error_lag1(2)/error_lag(2));
        error_lag1(2) = error_lag(2);
        r3_lag(i) = log2 (error_lag1(3)/error_lag(3));
        error_lag1(3) = error_lag(3);

        r1_pph(i) = log2 (error_pph1(1)/error_pph(1));
        error_pph1(1) = error_pph(1);
        r2_pph(i) = log2 (error_pph1(2)/error_pph(2));
        error_pph1(2) = error_pph(2);
        r3_pph(i) = log2 (error_pph1(3)/error_pph(3));
        error_pph1(3) = error_pph(3);

    end
else

r_lag = [r1_lag; r2_lag; r3_lag];
r_pph = [r1_pph; r2_pph; r3_pph];


function [xa, error_lag, error_pph] = reconstructionsimple ...
    (n, inc1, inc2, fhandle, dis)

% This function build an approximation to a function f defined in the
%      program using piecewise reconstructions
% [xa, error_lag, error_pph] = reconstructionsimple ...
%      (n, inc1, inc2, fhandle, dis)
% Input data:
% n reconstruction order
% inc1 step between partition nodes
```

```
% inc2 step to subdivide each interval of the partition
% fhandle considered function to be approximated
% dis is used to indicate the location of the discontinuities in the function
% Output data:
% xa points where the function f is approximated

x = 0:inc1:1;
nx = length(x);

% Load function values
if nargin == 5
    param = [nx,dis];
else
    param = nx;
end
[nod,f] = fhandle (param);

% Abscisas where we approximate
x1 = x(1):inc2:x(nx);
xa = x1(find(x1>=x(n/2) & x1<=x(nx-n/2+1)));

% Loop to build the reconstruction piecewise
% It calculates the first element to the reconstruction
aux = xa(find(xa>=x(n/2) & xa<x(n/2+1)));
auxf = lagrange(f(1:n),x(1:n),aux);
ap1 = auxf;
auxf = pphpower(f(1:n),x(1:n),aux);
ap2 = auxf;
clear auxf; clear aux;

for i=n/2+1:nx-n/2-1
    aux = xa(find(xa>=x(i) & xa<x(i+1)));
    auxf = lagrange(f(i-n/2+1:i+n/2),x(i-n/2+1:i+n/2),aux);
    ap1 = [ap1,auxf];
    auxf = pphpower(f(i-n/2+1:i+n/2),x(i-n/2+1:i+n/2),aux);
    ap2 = [ap2,auxf];
    clear auxf; clear aux;
end

% It calculates the last element to the reconstruction
```

```
aux = xa(find(xa>=x(nx-n/2) & xa<=x(nx-n/2+1)));
auxf = lagrange(f(nx-n+1:nx),x(nx-n+1:nx),aux);
ap1 = [ap1,auxf];
auxf = pphpower(f(nx-n+1:nx),x(nx-n+1:nx),aux);
ap2 = [ap2,auxf];
clear clear auxf; clear aux;

% Measurement of the approximation error
nx1=length(x1); param(1)=nx1;
[nod,f1] = fhandle (param);
nxa = length(xa);
f1 = f1((nx1-nxa)/2+1:nx1-(nx1-nxa)/2);
% It calculate the error in Lagrange aproximation
nap1=length(ap1);
e1_lag = norm(ap1-f1,1)/nap1;
e2_lag = norm(ap1-f1,2)/nap1;
einf_lag = norm(ap1-f1,'inf');
error_lag = [e1_lag, e2_lag, einf_lag];
% It calculate the error in PPH aproximation
nap2=length(ap2);
e1_pph=norm(ap2-f1,1)/nap2;
e2_pph=norm(ap2-f1,2)/nap2;
einf_pph=norm(ap2-f1,'inf');
error_pph = [e1_pph, e2_pph, einf_pph];



function [xa, ap, error, r_recons] = function_select_reconstruction...
    (n, inc1, inc2, fhandle1, fhandle2, varargin)

% This function build an approximation to a function f defined
%     in the program using piecewise reconstructions
% [xa, ap, error, r_recons] = function_select_reconstruction ...
%     (n, inc1, inc2, fhandle1, fhandle2, varargin)
% Input variables:
% n reconstruction order
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fhandle1 considered function to be approximated
% dis is used to indicate the location of the discontinuities
```

```
%      in the function
% sc optional parameter which indicates the number of scales to be
%      computed to obtain the numerical order of the reconstructions
% Output variables:
% xa points where the function f is approximated
% ap approximation at the points xa using reconstruction
% error contains the error in the approximation in l1, l2 and infinite norm

% It close the previous figures
figure(2);
close

% This separates the different variables that contains varargin
x = 0 : inc1 : 1;
nx = length(x);
dis = [];
sc = 0;
param = nx;
if nargin == 6
    if varargin{1} < 1
        dis = varargin{1};
        param = [nx, dis];
    else
        param = nx;
        sc = varargin{1};
    end
elseif nargin == 7
    dis = varargin{1};
    param = [nx, dis];
    sc = varargin{2};
end

% Load function values
[nod, f] = fhandle1 (param);

% Abscisas where we approximate
x1 = x(1) : inc2 : x(nx);
xa = x1(find(x1 >= x(n / 2) & x1 <= x(nx - n / 2 + 1)));

% It calls to the method that calculates the reconstruction
```

```
ap = fhandle2 (n, inc1, f, nx, xa);

% Measurement of the approximation error
nx1 = length(x1);
param(1) = nx1;
[nod, f1] = fhandle1 (param);
nxa = length(xa);
f1 = f1((nx1-nxa)/2+1:nx1-(nx1-nxa)/2);

% It calculate the error in selected approximation
nap = length(ap);
e1 = norm(ap - f1, 1)/nap;
e2 = norm(ap - f1, 2)/nap;
einf = norm(ap - f1, 'inf');
error = [e1, e2, einf];

% We determine the approximation order of the reconstructions
if sc > 0
    if isempty(dis)
        [r_recons] = order (sc, n, inc1, inc2, fhandle1);
    else
        [r_recons] = order (sc, n, inc1, inc2, fhandle1, dis);
    end
end

% Plot of the approximation and the original function
figure(2)
hold on

% It computes the axis for the plot
axm = min([ap, f1]);
axM = max([ap, f1]);
dif_axis = 0.1 * (axM - axm);

% It draw the reconstruction
plot(xa, f1, '-k', 'LineWidth', 3), ...
    title('Reconstruction Using Selected Method');
hold on
plot(xa, ap, '–r', 'LineWidth', 3);
legend('Original Function', ...
```

```
    'Reconstruction Using Selected Method', 0);
axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);



function [reconstruction] = recons_1d_lagrange (n, x, f, xa)

% Loop to build the reconstruction piecewise
nx=length(x);

% It calculates the first elements of the reconstruction
aux = xa(find(xa >= x(n / 2) & xa < x(n / 2 + 1)));
auxf = lagrange(f(1 : n), x(1 : n), aux);
reconstruction = auxf;
clear auxf;
clear aux;

% It calculates the others elements of the reconstruction
for i = n / 2 + 1 : nx - n / 2 - 1
    aux = xa(find(xa >= x(i) & xa < x(i + 1)));
    auxf = lagrange(f(i - n / 2 + 1 : i + n / 2), ...
        x(i - n / 2 + 1 : i + n / 2), aux);
    reconstruction = [reconstruction, auxf];
    clear auxf;
    clear aux;
end

% It calculates the last element of the reconstruction
aux = xa(find(xa >= x(nx - n / 2) & xa <= x(nx - n / 2 + 1)));
auxf = lagrange(f(nx - n + 1 : nx), x(nx - n + 1 : nx), aux);
reconstruction = [reconstruction, auxf];
clear auxf;
clear aux;



function [reconstruction] = recons_1d_pph (n, x, f, xa)

% Loop to build the reconstruction piecewise
nx=length(x);
```

```
% It calculates the first elements of the reconstruction
aux = xa(find(xa >= x(n / 2) & xa < x(n / 2 + 1)));
auxf = pphpower(f(1 : n), x(1 : n), aux);
reconstruction = auxf;
clear auxf;
clear aux;

% It calculates the others elements of the reconstruction
for i = n / 2 + 1 : nx - n / 2 - 1
    aux = xa(find(xa >= x(i) & xa < x(i + 1)));
    auxf = pphpower(f(i - n / 2 + 1 : i + n / 2), ...
        x(i - n / 2 + 1 : i + n / 2), aux);
    reconstruction = [reconstruction, auxf];
    clear auxf;
    clear aux;
end

% It calculates the last element of the reconstruction
aux = xa(find(xa >= x(nx - n / 2) & xa <= x(nx - n / 2 + 1)));
auxf = pphpower(f(nx - n + 1 : nx), x(nx - n + 1 : nx), aux);
reconstruction = [reconstruction, auxf];
clear auxf;
clear aux;
```

Para las reconstrucciones con datos se usaran los siguientes códigos,

```
function [xa,a1_lag, a1_pph] = data_reconstruction(n, incx, data1d)

% This function builds an approximation of a 1D function from
%      discrete data using Lagrange and PPH reconstructions
% [xa, a1_lag, a1_pph] = data_reconstruction(n, incx, data1d)
% Input variables:
% n reconstruction order
% incx step between partition nodes in the x-axes
% data1d name of the file containing the discrete data used to reconstruct
%      the function
% Output variables:
% xa grid points where the reconstructions have been evaluated
```

```
% a1_lag approximation using Lagrange reconstruction
% a1_pph approximation using PPH reconstruction

load(data1d);

nx=length(a);
x=linspace(0,1,nx);

% Abscisas where we approximate
x1 = x(1) : incx : x(nx);
nx1=length(x1);

xa = x1(find(x1 >= x(n / 2) & x1 <= x(nx - n / 2 + 1)));
nxa=length(xa);

a1_lag=zeros(1,nxa);
a1_pph=zeros(1,nxa);

% We interpolate the data
[a1_lag(1:nxa)]=recons_1d_lagrange(n,x(1:nx),a(1:nx),xa(:)');
[a1_pph(1:nxa)]=recons_1d_pph(n,x(1:nx),a(1:nx),xa(:)');

% Plot of the reconstruction
figure(2)
hold on

% It computes the axis for the plot
axm = min([a1_lag, a1_pph, a]);
axM = max([a1_lag, a1_pph, a]);
dif_axis = 0.1 * (axM - axm);

% It draw the Lagrange reconstruction
subplot(2, 1, 1), plot(x,a , '-k', 'LineWidth', 3), ...
    title('Reconstruction Using Lagrange');
hold on
subplot(2, 1, 1), plot(xa, a1_lag, '–r', 'LineWidth', 3);
subplot(2, 1, 1), legend('Original Data', ...
    'Reconstruction Using Lagrange','Location','Best', 0);
axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);
```

```
%It draw the PPH reconstruccion
subplot(2, 1, 2), plot(x, a, '-k', 'LineWidth', 3), ...
    title('Reconstruction Using PPH');
hold on
subplot(2, 1, 2), plot(xa, a1_pph, '–g', 'LineWidth',3);
subplot(2, 1, 2), legend('Original Data', ...
    'Reconstruction Using PPH','Location','Best', 0);
axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);



function [xa, a1] = data_select_reconstruction(n, incx, data1d, fhandle)

% This function builds an approximation of a 1D function from
%      discrete data selecting reconstruction
% [xa, a1] = data_select_reconstruction(n, incx, data1d, fhandle)
% Input variables:
% n reconstruction order
% incx step between partition nodes in the x-axes
% data1d name of the file containing the discrete data used to
%      reconstruct the function
% fhandle name of the file containing the reconstruction type
% Output variables:
% xa grid points where the reconstructions have been evaluated
% a1 approximation using the selected method for the reconstruction

load(data1d);

nx=length(a);
x=linspace(0,1,nx);

% Abscisas where we approximate
x1 = x(1) : incx : x(nx);
nx1=length(x1);

xa = x1(find(x1 >= x(n / 2) & x1 <= x(nx - n / 2 + 1)));
nxa=length(xa);

a1=zeros(1,nxa);
```

```
% We interpolate the data
[a1(1:nxa)]=fhandle(n,x(1:nx),a(1:nx),xa(:)');

% Plot of the reconstruction
figure(2)
hold on

% It computes the axis for the plot
axm = min([a1, a]);
axM = max([a1, a]);
dif_axis = 0.1 * (axM - axm);

% It draw the reconstruction
plot(x,a , '-k', 'LineWidth', 3), ...
    title('Reconstruction Selected Method');
hold on
plot(xa, a1, '–r', 'LineWidth', 3);
legend('Original Data', ...
    'Reconstruction Using Selected Method','Location','Best', 0);
axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);
```

6.1.2. Códigos en 2D

Además de los códigos ya vistos en 1D también se se verán los siguientes,

```
function [Xa, Ya, mapa1_lag, mapa1_pph, error_lag, error_pph,r_lag, r_pph]...
    = reconstruction2D_funcion (n, inc1, inc2, fun1, sc, varargin)

% This functions build an approximation to a function f in 2D
%     defined in the program using piecewise reconstructions
% [Xa, Ya, mapa1_lag, mapa1_pph, error_lag, error_pph,r_lag, r_pph]...
%     = reconstruction2D_funcion (n, inc1, inc2, fun1, sc, varargin)
% Input variables:
```

```
% n reconstruction order
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fun1 considered function in a string to be approximated to one side
%      of the discontinuity
% sc=[nº,[xy]] optional parameter which indicates the number of scales
%      to be computed to obtain the numerical order of the reconstructions
%      and the selected direction x or y
% fun2 considered function in a string to be approximated to the other
%      side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%      in the function
% Output variables:
% [Xa,Ya] grid points where the reconstructions have been evaluated
% mapa1_lag approximation using Lagrange reconstruction
% mapa1_pph approximation using PPH reconstruction
% error_lag error in Lagrange approximation in l1, l2 and infinite norm
% error_pph error in PPH approximation in l1, l2 and infinite norm

% This separates the different variables contained in varargin
inc1_x=inc1(1); inc1_y=inc1(2);
x =0:inc1_x:1; nx=length(x);
y=0:inc1_y:1; ny=length(y);
intervalo=[0,1;0,1];
puntos=[nx,ny];

if nargin==5
    param={fun1,[],[],intervalo,puntos};
elseif nargin==7
    fun2=varargin{1};
    dis=varargin{2};
    param={fun1,fun2,dis,intervalo,puntos};
else
    display('Error');
    return;
end

% Load function values
[X,Y,f] = generar_funcion2D(param{1}, param{2}, param{3},...
    param{4}, param{5});
```

```
% Increment between evaluation nodes
inc2_x=inc2(1); inc2_y=inc2(2);

% Abscisas where we approximate
x1 = x(1) : inc2_x : x(nx); nx1=length(x1);
y1 = y(1) : inc2_y : y(ny); ny1=length(y1);

% We generate the real function values at a higher resolution
puntos1=[nx1,ny1];
if nargin==5
    param={fun1,[],[],intervalo,puntos1};
elseif nargin==7
    param={fun1,fun2,dis,intervalo,puntos1};
end

[X1,Y1,f1] =generar_funcion2D(param{1},param{2},param{3},...
    param{4}, param{5});

% We crop the borders as much in the xy-domain as in the
%     function values
indice_xa=find(x1 >= x(n / 2) & x1 <= x(nx - n / 2 + 1));
xa = x1(indice_xa);
indice_ya=find(y1 >= y(n / 2) & y1 <= y(ny - n / 2 + 1));
ya = y1(indice_ya);
nxa=length(xa); nya=length(ya);
[Xa,Ya]=meshgrid(ya,xa);
fa=f1(indice_xa,indice_ya);

% First, we interpolate the data along all rows
for i=1:nx
    [mapa1_lag(i,1:nya)]= recons_1d_lagrange(n,y(1:ny),f(i,1:ny),ya(:)');
    [mapa1_pph(i,1:nya)]=recons_1d_pph(n,y(1:ny),f(i,1:ny),ya(:)');
end

% Secondly, we interpolate the data along all columns
for j=1:nya
    [mapa1_lag(1:nxa,j)]= recons_1d_lagrange(n, x(1:nx),...
        mapa1_lag(1:nx,j)',xa(:)')';
    [mapa1_pph(1:nxa,j)]=recons_1d_pph(n, x(1:nx),...
```

```
        mapa1_pph(1:nx,j)',xa(:)')';
end

% We compute the approximation errors
% In the Lagrange approximation
error1_lag=sum(sum(abs(fa-mapa1_lag)))/nxa/nya;
error2_lag=sqrt(sum(sum((fa-mapa1_lag).^2))/nxa/nya);
errorinf_lag=max(max(abs(fa-mapa1_lag)));
error_lag=[error1_lag,error2_lag,errorinf_lag];

% In the PPH approximation
error1_pph=sum(sum(abs(fa-mapa1_pph)))/nxa/nya;
error2_pph=sqrt(sum(sum((fa-mapa1_pph).^2))/nxa/nya);
errorinf_pph=max(max(abs(fa-mapa1_pph)));
error_pph=[error1_pph,error2_pph,errorinf_pph];

% We determine the approximation order of the reconstructions
orden = sc;
if orden > 0
    if nargin == 5
        [r_lag, r_pph] = order2...
            (sc, n, inc1, fun1, error_lag, error_pph, xa, ya, fa);
    else
        [r_lag, r_pph] = order2...
            (sc, n, inc1, fun1, error_lag, error_pph, xa, ya, fa, fun2, dis);
    end
else
    r_lag=0;
    r_pph=0;
end

% Plot of the reconstruction
figure(1);
mesh(X(n/2:nx-n/2+1,n/2:ny-n/2+1),Y(n/2:nx-n/2+1,n/2:ny-n/2+1),...
    f(n/2:nx-n/2+1,n/2:ny-n/2+1));
title('Original data in lower resolution');
figure(2);
mesh(Xa,Ya,fa);
title('Original data in higher resolution');
figure(3);
```

```
mesh(Xa,Ya,mapa1_lag)
title('Lagrange reconstruction');
figure(4);
mesh(Xa,Ya,mapa1_pph)
title('PPH reconstruction');



function[X, Y, g] = generar_funcion2D (f1, f2, dis, intervalo, puntos)

% This program evaluates a 2D function can have a discontinuity in
%      the curve given rectangle discretized with as many points
%      as specified
% [X, Y, g] = generar_funcion2D (f1, f2, dis, intervalo, puntos)
% Input variables:
% f1 string with the expression of the function with variables
%      x1, x2, ...
% f2 string with the expression of the function to evaluate the other side of

%      the discontinuity (<0) if equipped with variable x1, x2, ...
% dis discontinuity curve
% intervalo rectangle where the function is defined
% puntos vector indicating the number of points that each variable
%      is discretized

syms x1 x2;

% We defined the mesh which to evaluate the function
x=linspace(intervalo(1,1),intervalo(1,2),puntos(1));
y=linspace(intervalo(2,1),intervalo(2,2),puntos(2));
[X,Y]=meshgrid(y,x);

% We evaluate the function
if isempty(dis)
    g=double(subs(f1,{x1,x2},{X,Y}));
else
    g=(0<=double(subs(dis,{x1,x2},{X,Y}))).*double(subs(f1,{x1,x2},{X,Y}))...
        +(0>double(subs(dis,{x1,x2},{X,Y}))).*(double(subs(f2,{x1,x2},{X,Y})));
end
```

```
function [r_lag,r_pph] = order2...
    (sc, n, inc1, fun1, error_lag1, error_pph1, xa, ya, fa, varargin)

% This function computes numerically the approximation order of the
%     piecewise Lagrange and PPH reconstructions in 2D
% [r_lag, r_pph] = order...
%     (sc, n, inc1, fun1, error_lag1, error_pph1, xa, ya, fa, varargin);
% Input variables:
% sc times that we subdivide the step size inc1
% n reconstruction order
% inc1 step between partition nodes
% fun1 considered function to be approximated
% error_lag1 error in Lagrange approximation in l1, l2 and infinite norm
% error_pph1 error in PPH approximation in l1, l2 and infinite norm
% xa, ya, fa information of the function at a high resolution
% fun2 considered function in a string to be approximated to the other
%     side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%     in the function
% Output variables:
% r_lag numerical approximation orders for the piecewise Lagrange
%     reconstruction in l1,l2 and infinite norms
% r_pph numerical approximation orders for the piecewise PPH
%     reconstruction in l1,l2 and infinite norms

% Smooth function
if nargin == 9

    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1(1)=inc1(1)/2;
        inc1(2)=inc1(2)/2;

        [error_lag, error_pph] = reconstruction2Dsimple...
            (n, inc1, fun1, xa, ya, fa);
```

```
        r1_lag(i) = log(error_lag1(1)/error_lag(1))/log(2);
        error_lag1(1) = error_lag(1);
        r2_lag(i) = log(error_lag1(2)/error_lag(2))/log(2);
        error_lag1(2) = error_lag(2);
        r3_lag(i) = log(error_lag1(3)/error_lag(3))/log(2);
        error_lag1(3) = error_lag(3);

        r1_pph(i) = log(error_pph1(1)/error_pph(1))/log(2);
        error_pph1(1) = error_pph(1);
        r2_pph(i) = log(error_pph1(2)/error_pph(2))/log(2);
        error_pph1(2) = error_pph(2);
        r3_pph(i) = log(error_pph1(3)/error_pph(3))/log(2);
        error_pph1(3) = error_pph(3);

    end

% Function with one discontinuity
else

    fun2 = varargin{1};
    dis = varargin{2};

    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1(1)=inc1(1)/2;
        inc1(2)=inc1(2)/2;


        [error_lag, error_pph] = reconstruction2Dsimple...
            (n, inc1, fun1, xa, ya, fa, fun2, dis);

        r1_lag(i) = log(error_lag1(1)/error_lag(1))/log(2);
        error_lag1(1) = error_lag(1);
        r2_lag(i) = log(error_lag1(2)/error_lag(2))/log(2);
        error_lag1(2) = error_lag(2);
        r3_lag(i) = log(error_lag1(3)/error_lag(3))/log(2);
        error_lag1(3) = error_lag(3);
```

```
        r1_pph(i) = log(error_pph1(1)/error_pph(1))/log(2);
        error_pph1(1) = error_pph(1);
        r2_pph(i) = log(error_pph1(2)/error_pph(2))/log(2);
        error_pph1(2) = error_pph(2);
        r3_pph(i) = log(error_pph1(3)/error_pph(3))/log(2);
        error_pph1(3) = error_pph(3);

    end

end

r_lag = [r1_lag; r2_lag; r3_lag];
r_pph = [r1_pph; r2_pph; r3_pph];



function [error_lag, error_pph] = reconstruction2Dsimple...
    (n, inc1, fun1, xa, ya, fa, varargin)

% This function build an approximation to a function f in
%     2D defined in the program using piecewise reconstructions
% [error_lag, error_pph] = reconstruction2Dsimple...
%     (n, inc1, fun1, xa, ya, fa, varargin)
% Input variables:
% n reconstruction order
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fun1 considered function in a string to be approximated to one
%     side of the discontinuity
% xa, ya, fa information of the function at a high resolution
% fun2 considered function in a string to be approximated to the
%     other side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%     in the function
% Output variables:
% error_lag error in Lagrange approximation in l1, l2 and infinite norm
% error_pph error in PPH approximation in l1, l2 and infinite norm

% This separates the different variables contained in varargin
inc1_x=inc1(1);
```

```
inc1_y=inc1(2);
x =0:inc1_x:1; nx=length(x);
y=0:inc1_y:1; ny=length(y);

if nargin==6
    param={fun1,[],[],x,y};
elseif nargin==8
    fun2=varargin{1};
    dis=varargin{2};
    param={fun1,fun2,dis,x,y};
else
    display('Error');
    return;
end

% Load function values
[X,Y,f] =generar_funcion2D_version1(param{1}, param{2}, param{3},...
    param{4}, param{5});
nxa=length(xa); nya=length(ya);
mapa1_lag=zeros(nxa,nya); mapa1_pph=zeros(nxa,nya);

% First, we interpolate the data along all rows
for i=1:nx
    [mapa1_lag(i,1:nya)]=recons_1d_lagrange_simple(n,y(1:ny),f(i,1:ny),ya(:)');
    [mapa1_pph(i,1:nya)]=recons_1d_pph_simple(n,y(1:ny),f(i,1:ny),ya(:)');
end

% Secondly, we interpolate the data along all columns
for j=1:nya
    [mapa1_lag(1:nxa,j)]=recons_1d_lagrange_simple(n, x(1:nx),mapa1_lag(1:nx,j)',xa(:)')';
    [mapa1_pph(1:nxa,j)]=recons_1d_pph_simple(n, x(1:nx),mapa1_pph(1:nx,j)',xa(:)')';
end

% We compute the approximation errors
% In the Lagrange approximation
error1_lag=sum(sum(abs(fa-mapa1_lag)))/nxa/nya;
error2_lag=sqrt(sum(sum((fa-mapa1_lag).^2))/nxa/nya);
errorinf_lag=max(max(abs(fa-mapa1_lag)));
error_lag=[error1_lag,error2_lag,errorinf_lag];
```

```
% In the PPH approximation
error1_pph=sum(sum(abs(fa-mapa1_pph)))/nxa/nya;
error2_pph=sqrt(sum(sum((fa-mapa1_pph).^2))/nxa/nya);
errorinf_pph=max(max(abs(fa-mapa1_pph)));
error_pph=[error1_pph,error2_pph,errorinf_pph];
```

Si deseamos seleccionar el método de reconstrucción, los códigos serán los siguientes,

```
function [Xa, Ya, mapa1_recons, error_recons, r_recons] =...
    reconstruction2D_funcion_seleccionar (n, inc1, inc2,...
    fun1, fhandle, sc, varargin)

% This functions build an approximation to a function f in 2D
%     defined in the program using piecewise reconstructions
% [Xa, Ya, mapa1_lag, mapa1_pph, error_lag, error_pph] =...
%     reconstruction2D_funcion_seleccionar (n, inc1, inc2,...
%     fun1, fhandle, sc, varargin)
% Input variables:
% n reconstruction order
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fun1 considered function in a string to be approximated to one side
%     of the discontinuity
% fhandle method of reconstruction selected
% sc=[nº,[xy]] optional parameter which indicates the number of scales
%     to be computed to obtain the numerical order of the reconstructions
%     and the selected direction x or y
% fun2 considered function in a string to be approximated to the other
%     side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%     in the function
% Output variables:
% [Xa,Ya] grid points where the reconstructions have been evaluated
% mapa1_recons approximation using selected reconstruction
% error_recons error in reconstruction approximation in l1, l2 and infinite
norm

% This separates the different variables contained in varargin
```

```
inc1_x=inc1(1); inc1_y=inc1(2);
x =0:inc1_x:1; nx=length(x);
y=0:inc1_y:1; ny=length(y);
intervalo=[0,1;0,1];
puntos=[nx,ny];

if nargin==6
    param={fun1,[],[],intervalo,puntos};
elseif nargin==8
    fun2=varargin{1};
    dis=varargin{2};
    param={fun1,fun2,dis,intervalo,puntos};
else
    display('Error');
    return;
end

% Load function values
[X,Y,f] = generar_funcion2D(param{1}, param{2}, param{3},...
    param{4}, param{5});

% Increment between evaluation nodes
inc2_x=inc2(1); inc2_y=inc2(2);

% Abscisas where we approximate
x1 = x(1) : inc2_x : x(nx); nx1=length(x1);
y1 = y(1) : inc2_y : y(ny); ny1=length(y1);

% We generate the real function values at a higher resolution
puntos1=[nx1,ny1];

if nargin==6
    param={fun1,[],[],intervalo,puntos1};
elseif nargin==8
    param={fun1,fun2,dis,intervalo,puntos1};
end

[X1,Y1,f1] =generar_funcion2D(param{1},param{2},param{3},...
    param{4}, param{5});
```

```
% We crop the borders as much in the xy-domain as in the
%     function values
indice_xa=find(x1 >= x(n / 2) & x1 <= x(nx - n / 2 + 1));
xa = x1(indice_xa);
indice_ya=find(y1 >= y(n / 2) & y1 <= y(ny - n / 2 + 1));
ya = y1(indice_ya);
nxa=length(xa); nya=length(ya);
[Xa,Ya]=meshgrid(ya,xa);
fa=f1(indice_xa,indice_ya);

% First, we interpolate the data along all rows
for i=1:nx
    [mapa1_recons(i,1:nya)]= fhandle(n,y(1:ny),f(i,1:ny),ya(:)');

% Secondly, we interpolate the data along all columns
for j=1:nya
    [mapa1_recons(1:nxa,j)]= fhandle(n, x(1:nx),mapa1_recons(1:nx,j)',xa(:)')';
end

% We compute the approximation errors
% In the selected approximation
error1_recons=sum(sum(abs(fa-mapa1_recons)))/nxa/nya;
error2_recons=sqrt(sum(sum((fa-mapa1_recons).^2))/nxa/nya);
errorinf_recons=max(max(abs(fa-mapa1_recons)));
error_recons=[error1_recons,error2_recons,errorinf_recons];

% We determine the approximation order of the reconstructions
orden = sc;
if orden > 0
    if nargin == 6
        [r_recons] = order2_select...
            (sc, n, inc1, fun1, fhandle, error_recons, xa, ya, fa);
    else
        [r_recons] = order2_select...
            (sc, n, inc1, fun1, fhandle, error_recons, xa, ya, fa, fun2, dis);
    end
else
    r_recons=0;
end
```

```
% Plot of the reconstruction
figure(1);
mesh(X(n/2:nx-n/2+1,n/2:ny-n/2+1),Y(n/2:nx-n/2+1,n/2:ny-n/2+1),...
    f(n/2:nx-n/2+1,n/2:ny-n/2+1));
title('Original data in lower resolution');
figure(2);
mesh(Xa,Ya,fa);
title('Original data in higher resolution');
figure(3);
mesh(Xa,Ya,mapa1_recons)
title('Reconstruction using selected method');
```

```
function [r_recons] = order2_select...
    (sc, n, inc1, fun1, fhandle, error_recons1, xa, ya, fa, varargin)

% This function computes numerically the approximation order of the
%     piecewise Lagrange and PPH reconstructions in 2D
% [r_lag, r_pph] = order...
%     (sc, n, inc1, fun1, error_lag1, error_pph1, xa, ya, fa, varargin);
% Input variables:
% sc times that we subdivide the step size inc1
% n reconstruction order
% inc1 step between partition nodes
% fun1 considered function to be approximated
% error_lag1 error in Lagrange approximation in l1, l2 and infinite norm
% error_pph1 error in PPH approximation in l1, l2 and infinite norm
% xa, ya, fa information of the function at a high resolution
% fun2 considered function in a string to be approximated to the other
%     side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%     in the function
% Output variables:
% r_lag numerical approximation orders for the piecewise Lagrange
%     reconstruction in l1,l2 and infinite norms
% r_pph numerical approximation orders for the piecewise PPH
%     reconstruction in l1,l2 and infinite norms
```

```
% Smooth function
if nargin == 9

    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1(1)=inc1(1)/2;
        inc1(2)=inc1(2)/2;

        [error_recons] = reconstruction2Dsimple_select...
            (n, inc1, fun1, fhandle, xa, ya, fa);

        r1_recons(i) = log(error_recons1(1)/error_recons(1))/log(2);
        error_recons1(1) = error_recons(1);
        r2_recons(i) = log(error_recons1(2)/error_recons(2))/log(2);
        error_recons1(2) = error_recons(2);
        r3_recons(i) = log(error_recons1(3)/error_recons(3))/log(2);
        error_recons1(3) = error_recons(3);

    end

% Function with one discontinuity
else

    fun2 = varargin{1};
    dis = varargin{2};

    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1(1)=inc1(1)/2;
        inc1(2)=inc1(2)/2;

        [error_recons] = reconstruction2Dsimple_select...
            (n, inc1, fun1, fhandle, xa, ya, fa, fun2, dis);

        r1_recons(i) = log(error_recons1(1)/error_recons(1))/log(2);
```

```
        error_recons1(1) = error_recons(1);
        r2_recons(i) = log(error_recons1(2)/error_recons(2))/log(2);
        error_recons1(2) = error_recons(2);
        r3_recons(i) = log(error_recons1(3)/error_recons(3))/log(2);
        error_recons1(3) = error_recons(3);

    end

end

r_recons = [r1_recons; r2_recons; r3_recons];
```

```
function [error_recons] =reconstruction2Dsimple_select...
    (n, inc1, fun1, fhandle, xa, ya, fa, varargin)

% This function build an approximation to a function f in 2D defined
%     in the program using piecewise reconstructions
% [error_recons] = reconstruction2Dsimple_select...
%     (n, inc1, fun1, xa, ya, fa, varargin)
% Input variables:
% n reconstruction order
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fun1 considered function in a string to be approximated to one
%     side of the discontinuity
% xa, ya, fa information of the function at a high resolution
% fun2 considered function in a string to be approximated to the other
%     side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%     in the function
% Output variables:
% error_recons error in the approximation in l1, l2 and infinite norm

% This separates the different variables contained in varargin
inc1_x=inc1(1);
inc1_y=inc1(2);
```

```
x =0:inc1_x:1; nx=length(x);
y =0:inc1_y:1; ny=length(y);

if nargin==7
    param={fun1,[],[],x,y};
elseif nargin==9
    fun2=varargin{1};
    dis=varargin{2};
    param={fun1,fun2,dis,x,y};
else
    display('Error');
    return;
end

% Load function values
[X,Y,f] =generar_funcion2D_version1(param{1}, param{2}, param{3},...
    param{4}, param{5});
nxa=length(xa); nya=length(ya);
mapa1_recons=zeros(nxa,nya);

% First, we interpolate the data along all rows
for i=1:nx
    [mapa1_recons(i,1:nya)]=fhandle(n,y(1:ny),f(i,1:ny),ya(:)');
end

% Secondly, we interpolate the data along all columns
for j=1:nya
    [mapa1_recons(1:nxa,j)]=fhandle(n, x(1:nx),mapa1_recons(1:nx,j)',xa(:)')';
end

% We compute the approximation errors
% In the selected approximation
error1_recons=sum(sum(abs(fa-mapa1_recons)))/nxa/nya;
error2_recons=sqrt(sum(sum((fa-mapa1_recons).^2))/nxa/nya);
errorinf_recons=max(max(abs(fa-mapa1_recons)));
error_recons=[error1_recons,error2_recons,errorinf_recons];
```

Y para las reconstrucciones con datos,

```
function [Xa,Ya,mapa1_lag, mapa1_pph] = reconstruction2D_datos...
    (n, inc, datos2d)

% This function builds an approximation of a 2D function from
%      discrete data using Lagrange and PPH reconstructions
% [Xa, Ya, mapa1_lag, mapa1_pph] = reconstruction2D_datos...
%      (n, inc, datos2d)
% Input variables:
% n reconstruction order
% inc step between partition nodes in the x-axes. The corresponding step
%      size for the y-axes is computed to make the grid uniform
% datos2D name of the file containing the discrete data used to
%      reconstruct the function
% Output variables:
% [Xa, Ya] grid points where the reconstructions have been evaluated
% mapa1_lag approximation using Lagrange reconstruction
% mapa1_pph approximation using PPH reconstruction

load (datos2d);

[nx,ny]=size(mapa);
x=linspace(0,1,nx);
y=linspace(0,1,ny);
[X,Y]=meshgrid(y,x);

incx = inc(1);
incy=inc(2);

% Abscisas where we approximate
x1 = x(1) : incx : x(nx);
y1 = y(1) : incy : y(ny);
[X1,Y1]=meshgrid(y1,x1);
[nx1,ny1]=size(X1);

xa = x1(find(x1 >= x(n / 2) & x1 <= x(nx - n / 2 + 1)));
ya = y1(find(y1 >= y(n / 2) & y1 <= y(ny - n / 2 + 1)));
nxa=length(xa); nya=length(ya);
[Xa,Ya]=meshgrid(ya,xa);

% First, we interpolate the data along all rows
```

```
for i=1:nx
    [mapa1_lag(i,1:nya)]=recons_1d_lagrange(n,y(1:ny),mapa(i,1:ny),ya(:)');
    [mapa1_pph(i,1:nya)]=recons_1d_pph(n,y(1:ny),mapa(i,1:ny),ya(:)');
end

% Secondly, we interpolate the data along all columns
for j=1:nya
    [mapa1_lag(1:nxa,j)]=recons_1d_lagrange(n, x(1:nx),mapa1_lag(1:nx,j)',xa(:)')';
    [mapa1_pph(1:nxa,j)]=recons_1d_pph(n, x(1:nx),mapa1_pph(1:nx,j)',xa(:)')';
end

% Plot of the reconstruction
figure(1);
mesh(X(n/2:nx-n/2+1,n/2:ny-n/2+1),Y(n/2:nx-n/2+1,n/2:ny-n/2+1),...
    mapa(n/2:nx-n/2+1,n/2:ny-n/2+1));
title('Original data');
figure(2);
mesh(Xa,Ya,mapa1_lag)
title('Lagrange reconstruction');
figure(3);
mesh(Xa,Ya,mapa1_pph)
title('PPH reconstruction');
```

6.2. Códigos con *aritmética variable*

En esta sección además de incluir los nuevos códigos implementados para trabajar con aritmética variable, también se añadirán los que han sufrido alguna variación significativa con respecto a su versión en *aritmética normal*.

6.2.1. Códigos en 1D

```
function [x1, ap1, ap2, error_lag, error_pph, r_lag, r_pph] = ...
    function_reconstruction (n, digitos, ...
    inc1, inc2, fhandle, varargin)

% This functions build an approximation to a function f defined
%     in the program using piecewise reconstructions
% [x1, ap1, ap2, error_lag, error_pph, r_lag, r_pph] = ...
%     function_reconstruction (n, digitos, ...
%     inc1, inc2, fhandle, varargin)
% Input variables:
% n reconstruction order
% digitos number of digits to perform the matlab operations
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fhandle considered function to be approximated
% dis is used to indicate the location of the discontinuities
%     in the function
% sc optional parameter which indicates the number of scales to be
%     computed to obtain the numerical order of the reconstructions
% Output variables:
% x1 points where the function f is approximated
% ap1 approximation at the points xa using Lagrange reconstruction
% ap2 approximation at the points xa using PPH reconstruction
% error_lag error in Lagrange approximation in l1, l2 and infinite norm
% error_pph error in PPH approximation in l1, l2 and infinite norm
% r_lag numerical approximation order for Lagrange in l1, l2 and
%     infinite norm
% r_pph numerical approximation order for PPH in l1, l2 and
%     infinite norm

% It closes the previous figures
figure(2);
close

% Definition of symbolic variables
digits(digitos);
syms x inc1_x inc2_x;
```

```
% This separates the different variables that contains varargin
inc1_x=vpa(inc1);
x =vpa(0):inc1_x:vpa(1); nx=length(x);
dis = [];
sc = 0;
param = [];
if nargin == 6
    if varargin{1} < 1
        dis = varargin{1};
        param = [param,dis];
    else
        sc = varargin{1};
    end
elseif nargin == 7
    dis = varargin{1};
    param = [param,dis];
    sc = varargin{2};
end

% Load function values
[f] = fhandle (x,param);

% Increment between evaluation nodes
inc2_x=vpa(inc2);

% Abscisas where we approximate
x1 = x(1) : inc2_x : x(nx); nx1=length(x1);

% Loop to build the reconstruction piecewise
% It calculates the first elements of the reconstruction
aux=x1(find(smaller(x1, x(n/2+1)*vpa(ones(size(x1))))));
auxf = lagrange(f(1 : n), x(1 : n), aux);
ap1 = auxf;
auxf = lagrange(f(1 : n), x(1 : n), aux);
ap2 = auxf;
clear auxf;
clear aux;

% It calculates the other elements of the reconstruction
```

```
for i=n/2+1:nx-n/2-1
    aux = x1(find(lequal(x(i)*vpa(ones(size(x1))),x1) & ...
        smaller(x1, x(i + 1)*vpa(ones(size(x1))))));
    auxf = lagrange(f(i-n/2+1:i+n/2),x(i-n/2+1:i+n/2), aux);
    ap1 = [ap1, auxf];
    auxf = pphpower(f(i-n/2+1:i+n/2),x(i-n/2+1:i+n/2), aux);
    ap2 = [ap2, auxf];
    clear auxf;
    clear aux;
end

% It calculates the last element of the reconstruction
aux = x1(find(lequal(x(nx-n/2)*vpa(ones(size(x1))),x1)));
auxf = lagrange(f(nx-n+1:nx),x(nx-n+1:nx),aux);
ap1 = [ap1, auxf];
auxf = lagrange(f(nx-n+1:nx),x(nx-n+1:nx),aux);
ap2 = [ap2, auxf];
clear auxf;
clear aux;

% Measurement of the approximation error
[f1] = fhandle (x1,param);
% It calculates the error in Lagrange approximation
e1_lag=sum(abs(f1-ap1))/nx1;
e2_lag=sqrt(sum((f1-ap1).^2)/nx1);
einf_lag=maxim(abs(f1-ap1));
error_lag=[e1_lag,e2_lag,einf_lag];
% It calculates the error in PPH approximation
e1_pph=sum(abs(f1-ap2))/nx1;
e2_pph=sqrt(sum((f1-ap2).^2)/nx1);
einf_pph=maxim(abs(f1-ap2));
error_pph=[e1_pph,e2_pph,einf_pph];

% We determine the approximation order of the reconstructions
if sc > 0
    if isempty(dis)
        [r_lag, r_pph] = order (sc, n, inc1, error_lag,error_pph, ...
            x1, f1, fhandle);
    else
        [r_lag, r_pph] = order (sc, n, inc1,error_lag,error_pph, ...
```

```
            x1, f1, fhandle, dis);
    end
end

% Plot of the approximation and the original function
figure(2)
hold on
% It computes the axis for the plot
axm = min([double(ap1), double(ap2), double(f1)]);
axM = max([double(ap1), double(ap2), double(f1)]);
dif_axis = 0.1 * (axM - axm);
% It draws the Lagrange reconstruction
subplot(2, 1, 1), plot(double(x1), double(f1), '-k', 'LineWidth', 3), ...
    title('Reconstruction Using Lagrange');
hold on
subplot(2, 1, 1), plot(double(x1), double(ap1), '–r', 'LineWidth', 3);
subplot(2, 1, 1), legend('Original Function', ...
    'Reconstruction Using Lagrange', 0);
axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);
%It draws the PPH reconstruction
subplot(2, 1, 2), plot(double(x1), double(f1), '-k', 'LineWidth', 3), ...
    title('Reconstruction Using PPH');
hold on
subplot(2, 1, 2), plot(double(x1), double(ap2), '–g', 'LineWidth',3);
subplot(2, 1, 2), legend('Original Function', ...
'Reconstruction Using PPH', 0);
    axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);



function [C] = smaller(A,B)

% This function checks if each element in matrix A are lower
%      than the corresponding element in matrix B. The
%      calculus are done in variable precision arithmetic.
%      The output is logical.
% [C] = smaller(A,B);
% Input variables:
% A,B matrices to check their elements
% Output variables:
```

```
% C binary image containing 0 or 1

[n,m]=size(A);
[n1,m1]=size(B);

if n~=n1 | m~=m1
    h= errordlg('The matrices A and B must be the same size.',...
        'Dimensions mismatch');
else

    for i=1:n
        for j=1:m
            f=char(A(i,j)-B(i,j));
            if f(1)=='-' & f(2) ='0'
                C(i,j)=logical(1);
            else
                C(i,j)=logical(0);
            end
        end
    end
end



function y = lagrange(f, nod, x)

% This function computes the lagrange reconstruction in one interval
% y = lagrange(f, nod, x)
% Input variables:
% f vector with the function values at the nodes
% nod vector with the nodes
% x set of abcisas to build the approximation
n = length(f);
m = length(x);
b = vpa(zeros(n, m));
for i=1:n
    b(i,1:m)=vpa(ones(1,m));
    for j=1:n
        if(j ~= i)
            b(i,1:m) = b(i,1:m).*(x-nod(j)*vpa(ones(1,m)))/(nod(i)-nod(j));
```

```
        end
    end
end
y = vpa(zeros(1, m));
for i=1:n
    y(1:m)=y(1:m)+b(i,1:m)*f(i);
end

return



function [C] = lequal(A,B)

% This function checks if each element in matrix A are lower or
%      equal than the corresponding element in matrix B. The
%      calculus are done in variable precision arithmetic.
%      The output is logical.
% [C] = lequal(A,B);
% Input variables:
% A,B matrices to check their elements
% Output variables:
% C binary image containing 0 or 1

[n,m]=size(A);
[n1,m1]=size(B);

if n~=n1 | m~=m1
    h= errordlg('The matrices A and B must be the same size.',...
        'Dimensions mismatch');
else

    for i=1:n
        for j=1:m
            f=char(A(i,j)-B(i,j));
            if f(1)=='-' | f(1)=='0'
                C(i,j)=logical(1);
            else
                C(i,j)=logical(0);
            end
```

```
        end
    end
end



function fm = f_modify(f, inc)

% This function computes the modified values for the pphpower
%      reconstruction from the initial values of the function f
% fm = f_modify(f, inc);
% Input variables:
% f vector with the discrete values of a function f(x)
% inc step between partition nodes
% Output variables:
% fm vector with the discrete modified values

n = length(f);
% Loop for the recursive modifications
for i = 1 : n / 2 - 1
    % Divided difference to the left
    vl = f(n / 2 - i : n / 2 + i);
    dl = diff_vp(vl, 2 * i);
    % Divided difference to the right
    vr = f(n / 2 - i + 1 : n / 2 + i + 1);
    dr = diff_vp(vr, 2 * i);
    coef = vpa(comb (2 * i));
    coef = [coef, fliplr(coef)];
    pmean = pwe(n - 2 * i, dl , dr, inc);
    if lequal(abs(dl),abs(dr))
        f(n / 2 + i + 1) = - f(n / 2 - i) +...
            sum(coef .* f(n / 2 - i + 1 : n / 2 + i)) + vpa(2) * pmean;
    else
        f(n / 2 - i) = - f(n / 2 + i + 1) +...
            sum(coef .* f(n / 2 - i + 1 : n / 2 + i)) + vpa(2) * pmean;
    end
end
    fm = f;
return
```

```
function [d] = diff_vp(v,n)

% This function generates the finite differences of order n of a given vector
% [d] = diff_vp(v,n);
% Input variables:
% v vector from which we want to compute its differences
% n order of the finite differences
% Output variables:
% d vector which contains the finite differences

d=v;
m=length(d);

for j=1:n
    for i=1:m-j
        d(i)=v(i+1)-v(i);
    end
    v(1:m-j)=d(1:m-j);
end

d=d(1:m-n);



function med = pwe (p, x, y, inc)

% This function computes the p-mean of two quantities in
%      variable precision
% med = pwe (p, x, y, inc);
% Input variables:
% p order of the p-mean
% x, y numbers to compute the p-mean
% inc step between partition nodes
% Output variables:
% med value of the p-mean

% We create the cases to evaluate them with a switch
if smaller(0,x* y)
```

```
    seleccion = 1;
elseif x ~= vpa(0) & y ~= vpa(0)
    seleccion = 2;
else
    seleccion = 3;
end
% We evaluate the diferent cases
switch seleccion
    % X and Y are mayor that zero
    case 1
        % When one of the variables is major that inc
        if smaller(inc,abs(x)) | smaller(inc,abs(y))
            r = x - y;
            s = x + y;
            med = s / vpa(2) * (vpa(1) - abs(r / s) ^ p);
        % When both varibles are minor
        else
            x = x + vpa(1);
            y = y + vpa(1);
            r = x - y;
            s = x + y;
            med = s / vpa(2) * (vpa(1) - abs(r / s) ^ p);
            med = med - vpa(1);
        end
    % X and Y different from zero
    case 2
        % When one of the variables is major that inc
        if smaller(inc,abs(x)) | smaller(inc,abs(y))
            [var1, ind] = minim (abs([x, y]));
            % X is the minimum
            if ind == 1
                sg = vpa( sign(double(y)));
                x1 = x + vpa(2) * sg * var1;
                y1 = y + vpa(2) * sg * var1;
            % Y is the minimum
            else
                sg = vpa(sign(double(x)));
                x1 = x + vpa(2) * sg * var1;
                y1 = y + vpa(2) * sg * var1;
            end
```

```
        r = x1 - y1;
        s = x1 + y1;
        med = s / vpa(2) * (vpa(1) - abs(r / s) ^ p);
        med = med - vpa(2) * sg * var1;
    % When both variables are minor
    else
        [var1,ind] = minim(abs([x, y]));
        % X is the minimum
        if ind == 1
            sg = vpa(sign(double(y)));
            x1 = x + vpa(2) * sg * var1 + sg * vpa(1);
            y1 = y + vpa(2) * sg * var1 + sg * vpa(1);
        % Y is the minimum
        else
            sg = vpa(sign(double(x)));
            x1 = x + vpa(2) * sg * var1 + sg * vpa(1);
            y1 = y + vpa(2) * sg * var1 + sg * vpa(1);
        end
        r = x1 - y1;
        s = x1 + y1;
        med = s / vpa(2) * (vpa(1) - abs(r./ s).^ p);
        med = med - vpa(2) * sg * var1 - sg * vpa(1);;
    end
% Other cases
case 3
    if x == vpa(0) & y == vpa(0)
        med = vpa(0);
    elseif x == vpa(0)
        sg = vpa(sign(double(y)));
        x1 = sg * vpa(1);
        y1 = y + sg * vpa(1);
        r = x1 - y1;
        s = x1 + y1;
        med = s / vpa(2) * (vpa(1) - abs(r./s).^ p);
        med = med - sg * vpa(1);
    else
        sg = vpa(sign(double(x)));
        x1 = x + sg * vpa(1);
        y1 = sg * vpa(1);
        r = x1 - y1;
```

```
            s = x1 + y1;
            med = s / vpa(2) * (vpa(1) - abs(r./s).^ p);
            med = med - sg * vpa(1);
        end
end



function [mini,index] = minim(A)

% Function to compute the minimum value of the elements in a
%      given vector or matrix, allowing the use of variable
%      precision with D digits
% (digits(D))
% [mini,index] = minim(A)
% Input Variables:
% A vector for which we want to compute the minimum value
% Output Variables:
% mini minimum value
% index first index at which the minimum is attained

% In case A is a matrix, the minimum is calculated for each row
[n,m]=size(A);

% Case of a vector
if n==1
    mini=A(1);
    index=1;

    for i=2:m
        f=char(A(i)-mini);
        if f(1)=='-' & f(2)~='0'
            mini=A(i);
            index=i;
        end
    end
else
    for j=1:n
        mini(j)=A(j,1);
        index(j)=1;
```

```
        for i=2:m
            f=char(A(j,i)-mini(j));
            if f(1)=='-' & f(2)~='0'
                mini(j)=A(j,i);
                index(j)=i;
            end
        end
    end
end
```

```
function [r_lag, r_pph] = order (sc, n, inc1, error_lag1, error_pph1, ...
    x1,f1, fhandle, varargin)

% This function computes numerically the approximation order of the
%     piecewise Lagrange and PPH reconstructions
% [r_lag, r_pph] = order(sc, n, inc1, error_lag1, error_pph1, ...
%     x1, f1, fhandle, varargin);
% Input variables:
% sc times that we subdivide the step size inc1
% n reconstruction order
% inc1 step between partition nodes
% error_lag1 error in Lagrange approximation in l1, l2 and infinite norm
% error_pph1 error in PPH approximation in l1, l2 and infinite norm
% x1, f1 information of the function at a high resolution
% fhandle considered function to be approximated
% dis is used to indicate the location of the discontinuities in the
%     function
% Output variables:
% r_lag numerical approximation orders for the piecewise Lagrange
%     reconstruction in l1,l2 and infinite norms
% r_pph numerical approximation orders for the piecewise PPH
%     reconstruction in l1,l2 and infinite norms

% Non smooth function
if nargin==9
    dis = varargin{1};
```

```
    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1=inc1/2;

        [error_lag, error_pph] = reconstructionsimple ...
            (n, inc1, x1, f1, fhandle, dis);

        r1_lag(i) = log(error_lag1(1)/error_lag(1))/log(2);
        error_lag1(1) = error_lag(1);
        r2_lag(i) = log(error_lag1(2)/error_lag(2))/log(2);
        error_lag1(2) = error_lag(2);
        r3_lag(i) = log(error_lag1(3)/error_lag(3))/log(2);
        error_lag1(3) = error_lag(3);

        r1_pph(i) = log(error_pph1(1)/error_pph(1))/log(2);
        error_pph1(1) = error_pph(1);
        r2_pph(i) = log(error_pph1(2)/error_pph(2))/log(2);
        error_pph1(2) = error_pph(2);
        r3_pph(i) = log(error_pph1(3)/error_pph(3))/log(2);
        error_pph1(3) = error_pph(3);

    end

else

    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1=inc1/2;

        [error_lag, error_pph] = reconstructionsimple ...
            (n, inc1, x1, f1, fhandle);

        r1_lag(i) = log(error_lag1(1)/error_lag(1))/log(2);
        error_lag1(1) = error_lag(1);
        r2_lag(i) = log(error_lag1(2)/error_lag(2))/log(2);
        error_lag1(2) = error_lag(2);
```

```
        r3_lag(i) = log(error_lag1(3)/error_lag(3))/log(2);
        error_lag1(3) = error_lag(3);

        r1_pph(i) = log(error_pph1(1)/error_pph(1))/log(2);
        error_pph1(1) = error_pph(1);
        r2_pph(i) = log(error_pph1(2)/error_pph(2))/log(2);
        error_pph1(2) = error_pph(2);
        r3_pph(i) = log(error_pph1(3)/error_pph(3))/log(2);
        error_pph1(3) = error_pph(3);

    end

end

r_lag = [r1_lag; r2_lag; r3_lag];
r_pph = [r1_pph; r2_pph; r3_pph];



function [maxi,index] = maxim(A)

% Function to compute the maximum value of the elements in a
%      given vector or matrix, allowing the use of variable
%      precision with D digits
% (digits(D))
% [maxi,index] = maxim(A)
% Input Variables:
% A vector for which we want to compute the maximum value
% Output Variables:
% maxi maximum value
% index first index at which the maximum is attained

% In case A is a matrix, the maximum is calculated for each row
[n,m]=size(A);

% Case of a vector
if n==1
    maxi=A(1);
    index=1;
```

```
    for i=2:m
        f=char(maxi-A(i));
        if f(1)=='-' & f(2)~='0'
            maxi=A(i);
            index=i;
        end
    end
else
    for j=1:n
        maxi(j)=A(j,1);
        index(j)=1;

        for i=2:m
            f=char(maxi(j)-A(j,i));
            if f(1)=='-' & f(2)~='0'
                maxi(j)=A(j,i);
                index(j)=i;
            end
        end
    end
end
```

```
function [error_lag, error_pph] = reconstructionsimple ...
    (n, inc1, x1, f1, fhandle, varargin)

% This function build an approximation to a function f defined in the
%      program using piecewise lagrange and reconstructions
% [error_lag, error_pph] = reconstructionsimple ...
%      (n, inc1, x1, f1, fhandle, varargin)
% Input data:
% n reconstruction order
% inc1 step between partition nodes
% x1, f1 information of the function at a high resolution
% fhandle considered function to be approximated
% dis is used to indicate the location of the discontinuities in the function
% Output data:
% error_lag error in Lagrange approximation in l1, l2 and infinite norm
% error_pph error in PPH approximation in l1, l2 and infinite norm
```

```
syms x inc1_x;

inc1_x=vpa(inc1);
x =vpa(0):inc1_x:vpa(1); nx=length(x);

% Load function values
param=[];
if nargin == 6
    dis=varargin{1};
    param = [param,dis];
end
[f] = fhandle(x,param);

% Loop to build the reconstruction piecewise
% It calculates the first elements of the reconstruction
aux=x1(find(smaller(x1, x(n/2+1)*vpa(ones(size(x1))))));
auxf = lagrange(f(1 : n), x(1 : n), aux);
ap1 = auxf;
auxf = lagrange(f(1 : n), x(1 : n), aux);
ap2 = auxf;
clear auxf;
clear aux;

% It calculates the others elements of the reconstruction
for i=n/2+1:nx-n/2-1
    aux = x1(find(lequal(x(i)*vpa(ones(size(x1))),x1) & ...
        smaller(x1, x(i + 1)*vpa(ones(size(x1))))));
    auxf = lagrange(f(i-n/2+1:i+n/2),x(i-n/2+1:i+n/2), aux);
    ap1 = [ap1, auxf];
    auxf = pphpower(f(i-n/2+1:i+n/2),x(i-n/2+1:i+n/2), aux);
    ap2 = [ap2, auxf];
    clear auxf;
    clear aux;
end

% It calculates the last element of the reconstruction
aux = x1(find(lequal(x(nx-n/2)*vpa(ones(size(x1))),x1)));
auxf = lagrange(f(nx-n+1:nx),x(nx-n+1:nx),aux);
ap1 = [ap1, auxf];
```

```
auxf = lagrange(f(nx-n+1:nx),x(nx-n+1:nx),aux);
ap2 = [ap2, auxf];
clear auxf;
clear aux;

% Measurement of the approximation error
nx1=length(x1);

% In the Lagrange approximation
error1_lag=sum(abs(f1-ap1))/nx1;
error2_lag=sqrt(sum((f1-ap1).^2)/nx1);
errorinf_lag=maxim(abs(f1-ap1));
error_lag=[error1_lag,error2_lag,errorinf_lag];

% In the PPH approximation
error1_pph=sum(abs(f1-ap2))/nx1;
error2_pph=sqrt(sum((f1-ap2).^2)/nx1);
errorinf_pph=maxim(abs(f1-ap2));
error_pph=[error1_pph,error2_pph,errorinf_pph];
```

Y para datos usaremos el siguiente código,

```
function [x1, a1_lag, a1_pph] = data_reconstruction ...
    (n, digitos, inc, data1d)

% This functions builds an approximation of a 1D function from
%      discrete data using Lagrange and PPH reconstructions
% [x1, a1_lag, a1_pph] = data_reconstruction ...
%      (n, digitos,inc, data1d)
% Input variables:
% n reconstruction order
% digitos number of digits to be used in the computations
% inc step between partition nodes in the x-axes
% data1d name of the file containing the discrete data used to reconstruct
%      the function
% Output variables:
% x1 grid points where the reconstructions have been evaluated
% a1_lag approximation using Lagrange reconstruction
% a1_pph approximation using PPH reconstruction
```

```
digits(digitos);
syms x incx;

incx=vpa(inc);
load(data1d);
a=vpa(a);

nx=length(a);
x=linspace(vpa(0),vpa(1),nx);

% Abscisas where we approximate
x1 = x(1) : incx : x(nx);
nx1=length(x1);

a1_lag=vpa(zeros(1,nx1));
a1_pph=vpa(zeros(1,nx1));

% We interpolate the data
[a1_lag(1:nx1)]=reconstruction1D_lagrange(n,x(1:nx),a(1:nx),nx,x1(1:nx1));
[a1_pph(1:nx1)]=reconstruction1D_pph(n,x(1:nx),a(1:nx),nx,x1(1:nx1));

% Plot of the reconstruction
figure(2)
hold on

% It computes the axis for the plot
axm = min([double(a1_lag), double(a1_pph), double(a)]);
axM = max([double(a1_lag), double(a1_pph), double(a)]);
dif_axis = 0.1 * (axM - axm);

% It draw the Lagrange reconstruction
subplot(2, 1, 1), plot(double(x),double(a) , '-k', 'LineWidth', 3), ...
    title('Reconstruction Using Lagrange');
hold on
subplot(2, 1, 1), plot(double(x1), double(a1_lag), '–r', 'LineWidth', 3);
subplot(2, 1, 1), legend('Original Data', ...
    'Reconstruction Using Lagrange','Location','Best', 0);
axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);
%It draw the PPH reconstruccion
```

```
subplot(2, 1, 2), plot(double(x), double(a), '-k', 'LineWidth', 3), ...
    title('Reconstruction Using PPH');
hold on
subplot(2, 1, 2), plot(double(x1), double(a1_pph), '–g', 'LineWidth',3);
subplot(2, 1, 2), legend('Original Data', ...
    'Reconstruction Using PPH','Location','Best', 0);
axis([-0.1 1.1 axm - dif_axis axM + dif_axis]);
```

6.2.2. Códigos en 2D

Además de los códigos ya vistos en 1D también se se verán los siguientes,

```
function [Xa, Ya, mapa1_lag, mapa1_pph, error_lag, error_pph,...
    r_lag, r_pph] = reconstruction2D_funcion (n, digitos, inc1,...
    inc2, fun1, sc, varargin)

% This function build an approximation to a function f in 2D defined
%     in the program using piecewise reconstructions
% [Xa, Ya, mapa1_lag, mapa1_pph, error_lag, error_pph, ...
%     r_lag, r_pph] = reconstruction2D_funcion (n, digitos, inc1,...
%     inc2, fun1, sc, varargin)
% Input variables:
% n reconstruction order
% digitos number of decimal precision
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fun1 considered function in a string to be approximated to one side
%     of the discontinuity
% sc optional parameter which indicates the number of scales to be
%     computed to obtain the numerical order of the reconstructions
% fun2 considered function in a string to be approximated to the other
%     side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%     in the function
```

```
% Output variables:
% [Xa, Ya] grid points where the reconstructions have been evaluated
% mapa1_lag approximation using Lagrange reconstruction
% mapa1_pph approximation using PPH reconstruction
% error_lag error in Lagrange approximation in l1, l2 and infinite norm
% error_pph error in PPH approximation in l1, l2 and infinite norm

% Definition of symbolic variables
digits(digitos);
syms x y inc1_x inc1_y inc2_x inc2_y;

% This separates the different variables contained in varargin
inc1_x=vpa(inc1(1)); inc1_y=vpa(inc1(2));
x =vpa(0):inc1_x:vpa(1); nx=length(x);
y=vpa(0):inc1_y:vpa(1); ny=length(y);

if nargin==6
    param={fun1,[],[],x,y};
elseif nargin==8
    fun2=varargin{1};
    dis=varargin{2};
    param={fun1,fun2,dis,x,y};
else
    display('Error');
    return;
end

% Load function values
[X, Y, f] = generar_funcion2D(param{1}, param{2}, param{3},...
    param{4}, param{5});

% Increment between evaluation nodes
inc2_x=vpa(inc2(1)); inc2_y=vpa(inc2(2));

% Abscisas where we approximate
x1 = x(1) : inc2_x : x(nx); nx1=length(x1);
y1 = y(1) : inc2_y : y(ny); ny1=length(y1);

% We generate the real function values at a higher resolution
if nargin==6
```

```
    param={fun1,[],[],x1,y1};
elseif nargin==8
    param={fun1,fun2,dis,x1,y1};
end

[X1,Y1,f1] =generar_funcion2D(param{1},param{2},param{3},...
    param{4}, param{5});

% We crop the borders as much in the xy-domain as in the function values
indice_xa = find(lequal(x(n/2)*vpa(ones(size(x1))),x1) & lequal...
    (x1, x(nx - n / 2 + 1)*vpa(ones(size(x1)))));
xa = x1(indice_xa);
indice_ya = find(lequal(y(n/2)*vpa(ones(size(y1))),y1) & lequal...
    (y1, y(ny - n / 2 + 1)*vpa(ones(size(y1)))));
ya = y1(indice_ya);
nxa=length(xa); nya=length(ya);
[Xa,Ya]=meshgrid(double(ya),double(xa));
fa=f1(indice_xa,indice_ya);

% First, we interpolate the data along all rows
mapa1_lag=vpa(zeros(nxa,nya));
mapa1_pph=vpa(zeros(nxa,nya));

for i=1:nx
    [mapa1_lag(i,1:nya)]=recons_1d_lagrange(n,y(1:ny),f(i,1:ny),ya(:)');
    [mapa1_pph(i,1:nya)]=recons_1d_pph(n,y(1:ny),f(i,1:ny),ya(:)');
end

% Secondly, we interpolate the data along all columns
for j=1:nya
    [mapa1_lag(1:nxa,j)]=recons_1d_lagrange(n, x(1:nx),mapa1_lag...
        (1:nx,j)',xa(:)')';
    [mapa1_pph(1:nxa,j)]=recons_1d_pph(n, x(1:nx),mapa1_pph...
        (1:nx,j)',xa(:)')';
end

% We compute the approximation errors
% In the Lagrange approximation
error1_lag=sum(sum(abs(fa-mapa1_lag)))/nxa/nya;
error2_lag=sqrt(sum(sum((fa-mapa1_lag).^2))/nxa/nya);
```

```
errorinf_lag=maxim(maxim(abs(fa-mapa1_lag)));
error_lag=[error1_lag,error2_lag,errorinf_lag]

% In the PPH approximation
error1_pph=sum(sum(abs(fa-mapa1_pph)))/nxa/nya;
error2_pph=sqrt(sum(sum((fa-mapa1_pph).^2))/nxa/nya);
errorinf_pph=maxim(maxim(abs(fa-mapa1_pph)));
error_pph=[error1_pph,error2_pph,errorinf_pph]

% We determine the approximation order of the reconstructions
orden = sc;
if orden > 0
    if nargin == 5
        [r_lag, r_pph] = order2 (sc, n, inc1, fun1, error_lag,...
            error_pph, xa, ya, fa)
    else
        [r_lag, r_pph] = order2 (sc, n, inc1, fun1, error_lag,...
            error_pph, xa, ya, fa, fun2, dis)
    end
else
    r_lag=0;
    r_pph=0;
end

% Plot of the reconstruction
figure(1);
mesh(double(X(n/2:nx-n/2+1,n/2:ny-n/2+1)),double(Y(n/2:nx-n/2+1,...
    n/2:ny-n/2+1)),double(f(n/2:nx-n/2+1,n/2:ny-n/2+1)));
title('Original data in lower resolution');
figure(2);
mesh(double(Xa),double(Ya),double(fa))
title('Original data in higher resolution');
figure(3);
mesh(double(Xa),double(Ya),double(mapa1_lag))
title('Lagrange reconstruction');
figure(4);
mesh(double(Xa),double(Ya),double(mapa1_pph))
title('PPH reconstruction');
```

```
function [X, Y, g] = generar_funcion2D(f1, f2, dis, x, y)

% This program evaluates a 2D function can have a discontinuity in
%      the curve given rectangle discretized with as many points
%      as specified
% [X, Y, g] = generar_funcion2D (f1, f2, dis, x, y)
% Input variables:
% f1 string with the expression of the function with variables
%      x1, x2, ...
% f2 string with the expression of the function to evaluate the other
%      side of the discontinuity (<0) if equipped with variable x1, x2, ...
% dis discontinuity curve
% x, y vectors that contain the respective axes spatial discretizations
% Output variables:
% X,Y 2D grid where the function is evaluated
% g evaluation of the function at the grid points

syms x1 x2;

[X,Y]=meshgrid_vp(y,x);
[n,m]=size(X);
% We evaluate the function
if isempty(dis)
    g=vpa(subs(f1,{x1,x2},{X,Y}));
else
    g=lequal(vpa(zeros(n,m)),vpa(subs(dis,{x1,x2},{X,Y}))).*...
        vpa(subs(f1,{x1,x2},{X,Y}))+smaller(vpa(subs(dis,{x1,x2},...
        {X,Y})),vpa(zeros(n,m))).*(vpa(subs(f2,{x1,x2},{X,Y})));
end



function [X,Y] = meshgrid_vp(y,x)

% This function generates a 2D grid from the vectors y and x, which
%      give the partition nodes for each axes. All the
%      calculus are done in variable precision
% [X,Y] = meshgrid_vp(y,x)
% Input variables:
```

```
% y, x partition nodes for each axes
% Output variables:
% X, Y coordinates of the mesh

n=length(x);
m=length(y);

syms X Y;

X=vpa(zeros(m,n)); Y=vpa(zeros(m,n));

for i=1:m
    X(i,1:n)=x(1:n);
end

for i=1:n
    Y(1:m,i)=y(1:m)';
end



function [r_lag,r_pph] = order2 (sc, n, inc1, fun1, error_lag1, ...
    error_pph1, xa, ya, fa, varargin)

% This function computes numerically the approximation order
%     of the piecewise Lagrange and PPH reconstructions in 2D
% [r_lag,r_pph] = order2 (sc, n, inc1, fun1, error_lag1, ...
%     error_pph1, xa, ya, fa, varargin)
% Input variables:
% sc times that we subdivide the step size inc1
% n reconstruction order
% inc1 step between partition nodes
% fun1 considered function to be approximated
% error_lag1 error in Lagrange approximation in l1, l2 and infinite norm
% error_pph1 error in PPH approximation in l1, l2 and infinite norm
% xa, ya, fa information of the function at a high resolution
% fun2 considered function in a string to be approximated to the other
%     side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%     in the function
```

```
% Output variables:
% r_lag numerical approximation orders for the piecewise Lagrange
%       reconstruction in l1,l2 and infinite norms
% r_pph numerical approximation orders for the piecewise PPH
%       reconstruction in l1,l2 and infinite norms

% Smooth function
if nargin == 9

    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1(1)=inc1(1)/2;
        inc1(2)=inc1(2)/2;

        [error_lag, error_pph] = reconstruction2Dsimple...
            (n, inc1, fun1,xa, ya, fa);

        r1_lag(i) = log(error_lag1(1)/error_lag(1))/log(2);
        error_lag1(1) = error_lag(1);
        r2_lag(i) = log(error_lag1(2)/error_lag(2))/log(2);
        error_lag1(2) = error_lag(2);
        r3_lag(i) = log(error_lag1(3)/error_lag(3))/log(2);
        error_lag1(3) = error_lag(3);

        r1_pph(i) = log(error_pph1(1)/error_pph(1))/log(2);
        error_pph1(1) = error_pph(1);
        r2_pph(i) = log(error_pph1(2)/error_pph(2))/log(2);
        error_pph1(2) = error_pph(2);
        r3_pph(i) = log(error_pph1(3)/error_pph(3))/log(2);
        error_pph1(3) = error_pph(3);

    end

% Function with one discontinuity
else

    fun2 = varargin{1};
    dis = varargin{2};
```

```
    % Loop for the different discretization steps
    for i = 1:sc

        % Step size update
        inc1(1)=inc1(1)/2;
        inc1(2)=inc1(2)/2;

        [error_lag, error_pph] = reconstruction2Dsimple...
            (n, inc1, fun1, xa, ya, fa, fun2, dis);

        r1_lag(i) = log(error_lag1(1)/error_lag(1))/log(2);
        error_lag1(1) = error_lag(1);
        r2_lag(i) = log(error_lag1(2)/error_lag(2))/log(2);
        error_lag1(2) = error_lag(2);
        r3_lag(i) = log(error_lag1(3)/error_lag(3))/log(2);
        error_lag1(3) = error_lag(3);

        r1_pph(i) = log(error_pph1(1)/error_pph(1))/log(2);
        error_pph1(1) = error_pph(1);
        r2_pph(i) = log(error_pph1(2)/error_pph(2))/log(2);
        error_pph1(2) = error_pph(2);
        r3_pph(i) = log(error_pph1(3)/error_pph(3))/log(2);
        error_pph1(3) = error_pph(3);

    end

end

r_lag = [r1_lag; r2_lag; r3_lag];
r_pph = [r1_pph; r2_pph; r3_pph];



function [error_lag, error_pph] = ...
    reconstruction2Dsimple(n, inc1, fun1, xa, ya, fa, varargin)

% This function build an approximation to a function f in 2D defined
%      in the program using piecewise reconstructions
% [error_lag, error_pph] = ...
```

```
%      reconstruction2Dsimple(n, inc1, fun1, xa, ya, fa, varargin)
% Input variables:
% n reconstruction order
% inc1 step between partition nodes
% inc2 step to subdivide each interval of the partition
% fun1 considered function in a string to be approximated to one side
%      of the discontinuity
% xa, ya, fa information of the function at a high resolution
% fun2 considered function in a string to be approximated to the other
%      side of the discontinuity
% dis is used to indicate the location of the discontinuity curve
%      in the function
% Output variables:
% error_lag error in Lagrange approximation in l1, l2 and infinite norm
% error_pph error in PPH approximation in l1, l2 and infinite norm

% Definition of symbolic variables
digits(60);
syms x y inc1_x inc1_y inc2_x inc2_y;

% This separates the different variables contained in varargin
inc1_x=vpa(inc1(1));
inc1_y=vpa(inc1(2));
x =vpa(0):inc1_x:vpa(1); nx=length(x);
y=vpa(0):inc1_y:vpa(1); ny=length(y);

if nargin==6
    param={fun1,[],[],x,y};
elseif nargin==8
    fun2=varargin{1};
    dis=varargin{2};
    param={fun1,fun2,dis,x,y};
else
    display('Error');
    return;
end

% Load function values
[X,Y,f] = generar_funcion2D(param{1}, param{2}, param{3},...
    param{4}, param{5});
```

```
nxa=length(xa); nya=length(ya);
mapa1_lag=vpa(zeros(nxa,nya)); mapa1_pph=vpa(zeros(nxa,nya));

% First, we interpolate the data along all rows
for i=1:nx
    [mapa1_lag(i,1:nya)]=recons_1d_lagrange_simple...
        (n,y(1:ny),f(i,1:ny),ya(:)');
    [mapa1_pph(i,1:nya)]=recons_1d_pph_simple...
        (n,y(1:ny),f(i,1:ny),ya(:)');
end

% Secondly, we interpolate the data along all columns
for j=1:nya
    [mapa1_lag(1:nxa,j)]=recons_1d_lagrange_simple(n, x(1:nx),...
        mapa1_lag(1:nx,j)',xa(:)')';
    [mapa1_pph(1:nxa,j)]=recons_1d_pph_simple(n, x(1:nx),...
        mapa1_pph(1:nx,j)',xa(:)')';
end

% We compute the approximation errors
% In the Lagrange approximation
error1_lag=sum(sum(abs(fa-mapa1_lag)))/nxa/nya;
error2_lag=sqrt(sum(sum((fa-mapa1_lag).^2))/nxa/nya);
errorinf_lag=maxim(maxim(abs(fa-mapa1_lag)));
error_lag=[error1_lag,error2_lag,errorinf_lag]
% % In the PPH approximation
error1_pph=sum(sum(abs(fa-mapa1_pph)))/nxa/nya;
error2_pph=sqrt(sum(sum((fa-mapa1_pph).^2))/nxa/nya);
errorinf_pph=maxim(maxim(abs(fa-mapa1_pph)));
error_pph=[error1_pph,error2_pph,errorinf_pph]



function c = different_vp(x,y)

% This function determines if the values of the variable precision
%      matrices x and y are equal coordinate to coordinate
% c = different_vp(x,y);
% Input variables:
```

```
% x, y variable arithmetic values, vectors or matrices
% Output variables:
% c matrix of the same size of x and y indicating if x and y are
%       different. It is a logical variable

[nx,mx]=size(x);
[ny,my]=size(y);

if nx~=ny | mx~=my
    errordlg...
        ('Dimensions of x and y must be equal for the function different_vp',...
        'Error in different_vp','Modal');
    return
else
    for i=1:nx
        for j=1:mx
            d=x(i,j)-y(i,j);
            f=char(d);
            if f(1)~='0' & (f(1)~='-' | f(2)~='0')
                c(i,j)=logical(1);
            else
                c(i,j)=logical(0);
            end
        end
    end
end



function c = equal_vp(x,y)

% This function determines if the values of the variable
%       precision matrices x and y are equal coordinate to coordinate
% c = equal_vp(x,y);
% Input variables:
% x, y variable arithmetic values, vectors or matrices
% Output variables:
% c matrix of the same size of x and y indicating if x and y are
%       equal. It is a logical variable
```

```
[nx,mx]=size(x);
[ny,my]=size(y);

if nx~=ny | mx~=my
    errordlg...
        ('Dimensions of x and y must be equal for the function equal_vp',...
        'Error in equal_vp','Modal');
    return
        else

    for i=1:nx
        for j=1:mx
            d=x(i,j)-y(i,j);
            f=char(d);
            if f(1)=='0' | (f(1)=='-' & f(2)=='0')
                c(i,j)=logical(1);
            else
                c(i,j)=logical(0);
            end
        end
    end
end
```

Para datos,

```
function [Xa, Ya, mapa1_lag, mapa1_pph] = reconstruction2D_datos ...
    (n, digitos, inc,datos2d)

% This function builds an approximation of a 2D function from
%     discrete data using Lagrange and PPH reconstructions
% [Xa, Ya, mapa1_lag, mapa1_pph] = reconstruction2D_datos ...
%     (n, digitos, inc, datos2d)
% Input variables:
% n reconstruction order
% digitos number of precision digits
% inc step between partition nodes in the x-axes. The corresponding step
%     size for the y-axes is computed to make the grid uniform
% datos2D name of the file containing the discrete data used
%     to reconstruct the function
```

```
% Output variables:
% [Xa, Ya] grid points where the reconstructions have been evaluated
% mapa1_lag approximation using Lagrange reconstruction
% mapa1_pph approximation using PPH reconstruction

% Definition of symbolic variables
digits(digitos);
syms x y incx incy;

% Reads the data
load (datos2d);
[nx,ny]=size(mapa);
mapa=vpa(mapa);

x=linspace(vpa(0),vpa(1),nx); y=linspace(vpa(0),vpa(1),ny);
[X,Y]=meshgrid(double(y),double(x));

incx = vpa(inc(1));
incy=vpa(inc(2));

% Abscisas where we approximate
x1 = x(1) : incx : x(nx);
y1 = y(1) : incy : y(ny);
[X1,Y1]=meshgrid(double(y1),double(x1));
[nx1,ny1]=size(X1);

indice_xa = find(lequal(x(n/2)*vpa(ones(size(x1))),x1) & lequal...
    (x1,x(nx-n/2+1)*vpa(ones(size(x1)))));
xa = x1(indice_xa);
indice_ya = find(lequal(y(n/2)*vpa(ones(size(y1))),y1) & lequal...
    (y1,y(ny-n/2+1)*vpa(ones(size(y1)))));
ya = y1(indice_ya);
nxa=length(xa); nya=length(ya);
[Xa,Ya]=meshgrid(double(ya),double(xa));

% First, we interpolate the data along all rows
for i=1:nx
    [mapa1_lag(i,1:nya)]=recons_1d_lagrange(n,y(1:ny),mapa(i,1:ny),ya(:)');
    [mapa1_pph(i,1:nya)]=recons_1d_pph(n,y(1:ny),mapa(i,1:ny),ya(:)');
end
```

```
% Secondly, we interpolate the data along all columns
for j=1:nya
    [mapa1_lag(1:nxa,j)]=recons_1d_lagrange(n, x(1:nx),mapa1_lag...
        (1:nx,j)',xa(:)')';
    [mapa1_pph(1:nxa,j)]=recons_1d_pph(n, x(1:nx),mapa1_pph...
        (1:nx,j)',xa(:)')';
end

% Plot of the reconstruction
figure(1);
mesh(X(n/2:nx-n/2+1,n/2:ny-n/2+1),Y(n/2:nx-n/2+1,n/2:ny-n/2+1),...
    double(mapa(n/2:nx-n/2+1,n/2:ny-n/2+1)));
title('Original data');
figure(2);
mesh(Xa,Ya,double(mapa1_lag))
title('Lagrange reconstruction');
figure(3);
mesh(Xa,Ya,double(mapa1_pph))
title('PPH reconstruction');
```

Capítulo 7

Conclusiones

En el presente proyecto fin de carrera se han analizado dos técnicas interpolatorias con el objetivo de aplicarlas a la reconstrucción de señales.

Uno de los objetivos de este proyecto era la programación de los distintos algoritmos en Matlab. Además de realizar dichos algoritmos también hemos incluido la opción de poder trabajar con *aritmética variable*, esto es, la posibilidad de usar tantas cifras significativas como deseemos (cabe tener en cuenta el gasto computacional requerido).

También se han diseñado una serie de *Interfaces Gráficas de Usuario*, con las que se consigue una mayor simplicidad a la hora de trabajar con los algoritmos implementados. Cabe destacar que dichas interfaces se pueden dividir en varios grupos. En primer lugar podemos seleccionar bien trabajar con *aritmética normal*, bien trabajar con *aritmética variable*. La segunda división consiste en separar los casos en los que se trabaja con funciones de los que se trabaja con los datos directamente.

Una de las conclusiones a la que hemos llegado es que cuando se trabaja con funciones con alguna discontinuidad la reconstrucción de *Lagrange* produce oscilaciones alrededor de la discontinuidad y un suavizado inexistente. Mientras que la reconstrucción que nos proporciona *PPH* se ajusta mejor a la función original en las discontinuidades, tanto para funciones en 1D como para 2D. En el caso de funciones suaves sin discontinuidades ambos métodos funcionan de una manera análoga.

Entre las posibles aplicaciones hemos visto el zoom de datos geológicos, la compresión de imágenes digitales y la construcción de fórmulas de integración numérica adaptadas a la presencia de discontinuidades.

Bibliografía

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

[2] Amat S., Aràndiga F., Cohen A., Donat R., García G. and von Oehsen M., (2001). Data compression with ENO schemes: A case study. *Applied and Computational Harmonic Analysis*, 11, 273-288.

[3] Amat S., Busquier S. and Trillo J.C. (2006), Nonlinear Harten's Multiresolution on the Quincunx Pyramid, *J. of Comp. and App. Math.*, 189, 555-567.

[4] Amat S., Cherif H. and Trillo J.C. (2005), Denoising using Linear and Nonlinear Multiresolutions, *Engineering Computations*, 22(7), 877-891.

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

[6] Amat S., Donat R., Liandrat J. and Trillo J.C. (2002), Analysis of a New Nonlinear Subdivision Scheme. Applications in Image Processing. *Foundations of Computational Mathematics*, 6(2), 193-225.

[7] Amat S., Donat R. and Trillo J.C. (2009), *On specific stability bounds for linear multiresolution schemes based on piecewise polynomial Lagrange interpolation* , Journal of Math. Anal. and Appl. , doi:10.1016/j.jmaa.2009.04.038.

[8] Amat S., Ruiz J. and Trillo J.C., Fast multiresolution algorithms and their related variational problems for image denoising. Submitted.

[9] Aràndiga F. and Donat R., (2000). Nonlinear Multi-scale Decomposition: The Approach of A.Harten, Numerical Algorithms, 23, pp. 175-216.

[10] Aràndiga F., Donat R. and Harten A., (1999) Multiresolution Based on Weighted Averages of the Hat Function I: Linear Reconstruction Operators, *SIAM J. Numer. Anal.*, 36, 160-203.

[11] Chambolle A., Devore R.A., Lee N. and Lucier B.J., (1998). Nonlinear wavelet image processing: variational problem, compression and noise removal through wavelet shrinkage, *IEEE Trans. Image. Processing.*, 7, 319-335.

[12] Cohen A., Daubechies I. anf Feauveau J.C., (1992). Biorthogonal bases of compactly supported wavelets, *Comm. in Pure and Appl. Math.*, 45, 485-560.

[13] Dahmen W. and Micchelli C.A., (1997). Biorthogonal wavelet expansions. *Constr. Approx.*, 13(3), 293-328.

[14] Delauries G. and Dubuc S., (1989). Symmetric Iterative Interpolation Scheme, *Constr. Approx.* 5, 49-68.

[15] Donoho D., (1994). Interpolating wavelet transforms. Prepint Standford University.

[16] Fatemi E., Jerome J. and Osher S.,(1991). Solution of the hydrodynamic device model using high order non-oscillatory shock capturing algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10, 232-244.

[17] Gálvez M., (2009). Compresión y Zoom de Datos Geológicos utilizando Algoritmos de Multirresolución. *PFC UPCT*.

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

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

[20] Harten A., Osher S.J., Engquist B. and Chakravarthy S.R., (1987). Some results on uniformly high-order accurate essentially non-oscillatory schemes, *Appl. Numer. Math.*, 2, 347-377.

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

[22] Lin E-B. and Ling Y., (2003). Image compression and denoising via nonseparable wavelet approximation, Journal of Computational and Applied Mathematics, to appear.

[23] Liu X. D., Osher S. and Chan T., (1994). Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115, 200-212.

[24] Rabbani M. and Jones P.W., (1991). Digital Image Compression Techniques. Tutorial Text, *Society of Photo-Optical Instrumentation Engineers (SPIE)*, TT07.

[25] Ruiz J., (2010). Image processing using nonlinear multiresolution schemes. *Tesis Doctoral UPCT*.

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

[27] Shu C.W., (1990). Numerical experiments on the accuracy of ENO and modified ENO schemes. *Journal of Scientific Computing*, 5, 127-149.