UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

RECONSTRUCCIÓN BASADA EN INTERPOLACIÓN DE HERMITE APLICADA A FUNCIONES DISCONTINUAS

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: **Luis Torrente Cantó**
Director: Juan Carlos Trillo Moya
Codirector: Juan Ruiz Álvarez

Cartagena, 4 de diciembre de 2018

RECONSTRUCCIÓN BASADA EN INTERPOLACIÓN DE HERMITE APLICADA A FUNCIONES DISCONTINUAS.

Luis Torrente Cantó

Diciembre de 2018

A toda mi familia, en especial a mis padres, por su paciencia, colaboración y apoyo durante todos estos años,

A mis directores Juan Carlos y Juan, por su ayuda y predisposición durante todo este tiempo.

A todos vosotros,

¡Mil Gracias!

Autor	Luis Torrente Cantó
E-mail del Autor	Luisto85@hotmail.com
Directores	Juan Carlos Trillo Moya Juan Ruiz Álvarez
E-mail del Director	jc.trillo@upct.es juan.ruiza@uah.es
Título del TFG	Reconstrucción basada en interpolación de Hermite aplicada a funciones discontinuas.
Descriptores	Reconstrucciones, Multirresolución
Resumen	El proyecto consiste en el estudio de los operadores de reconstrucción de Hermite para funciones suaves salvo en un conjunto finito de discontinuidades en la primera derivada. También se lleva a cabo la implementación de los algoritmos propuestos en programas e interfaces gráficas en Matlab.
Titulación	Grado en Ingeniería en Tecnologías Industriales
Departamento	Matemática Aplicada y Estadística
Fecha de Presentación	4 de diciembre de 2018

Índice de tablas

Índice de figuras

Índice general

Capítulo 1

Introducción.

El trabajo presentado en esta memoria está dedicado al estudio del operador de reconstrucción basado en el método segmentario de Hermite y adaptado a funciones suaves salvo en un conjunto finito de discontinuidades de esquina, nos centramos en la programación en Matlab de los algoritmos necesarios para la realización del estudio, y en la investigación de las posibles aplicaciones de dichos operadores y en la programación en Matlab de los algoritmos.

Un problema al que nos enfrentamos en este proyecto consiste en la reconstrucción de una función suave definida a trozos, definidos entre discontinuidades en la primera derivada (esquinas). Para abordar este problema, se ha utilizado un algoritmo capaz de detectar este tipo discontinuidades, [4], [6], [9] y otro para obtener el valor de la función y sus derivadas cerca de una discontinuidad [7] para poder realizar Hermite segmentario entre las discontinuidades en la primera derivada.

Uno de los objetivos de este estudio consiste en comparar la reconstrucción de Hermite con detección previa de esquinas y de Hermite segmentario al operar con una función que contiene discontinuidades en la primera derivada.
Mediante experimentos numéricos realizamos una comparación tanto gráfica como numérica entre ambos métodos. Para los ejemplos test propuestos se obtienen los errores cometidos numéricamente en la reconstrucción al aproximar la función original. Se realiza también un estudio del orden de aproximación numérico para ambos métodos. Para finalizar desarrollamos un caso práctico que consiste en el modelado de una pieza industrial.

En el capítulo 2 se expone con cierto detalle la explicación teórica de

los métodos de interpolación considerados y en particular la reconstrucción de Hermite para funciones discontinuas. En el capítulo 3 se detallan todos los códigos programados en Matlab que han sido desarrollados durante la realización de este proyecto. También cabe comentar que en el capítulo 4 se elabora un tutorial para el manejo de la Interfaz Gráfica, así como un ejemplo acerca del uso de la misma. En el capítulo 5 se llevan a cabo experimentos numéricos. Y por último, en el capítulo 6 se exponen las conclusiones obtenidas en este proyecto.

Capítulo 2

Explicación teórica

2.1. Los polinomios de Taylor

En esta sección consideramos el problema de encontrar un polinomio de grado específico que esté "cerca"de una función dada, alrededor de un punto dado. Un polinomio P coincidirá con una función f en un punto x_0 precisamente cuando $P(x_0) = f(x_0)$. El polinomio tendrá también la misma "dirección"que la función en f en $(x_0, f(x_0))$ si $P'(x_0) = f'(x_0)$. De manera similar, el polinomio de enésimo grado que mejor aproxima a la función f cerca de x_0 tendrá tantas derivadas en x_0 como sea posible que coincidan con las de f.
Ésta es precisamente la condición que satisface el polinomio de Taylor de enésimo grado para la función en x_0, es decir:

$$P(x_0) = f(x_0), P'(x_0) = f'(x_0), ..., P^{(n)}(x_0) = f^n(x_0).$$

La expresión resultante para el polinomio de grando n que satisface dichas condiciones viene dada por:

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x - x_0)^2}{2!} + \cdots + f^{(n)}(x_0)\frac{(x - x_0)^n}{n!},$$

El cual tiene un residuo o término de error,

$$R_n(x) = f(x) - P_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1},$$

para algún número $\xi(x)$ entre x y x_0.

Ejemplo de Taylor:

Calcular el polinomio de Taylor de tercer grado alrededor de $x_0 = 0$ para $f(x) = (1+x)^{1/2}$. Usar el polinomio para aproximar $\sqrt{1{,}1}$.

$$\begin{array}{ll}
f(x) = (1+x)^{1/2}, & f(0) = 1, \\
f'(x) = \frac{1}{2}(1+x)^{-1/2}, & f'(0) = \frac{1}{2}, \\
f''(x) = -\frac{1}{4}(1+x)^{-3/2}, & f''(0) = -\frac{1}{4}, \\
f'''(x) = \frac{3}{8}(1+x)^{-5/2}, & f'''(0) = \frac{3}{8}, \\
f^{(iv)}(x) = -\frac{15}{16}(1+x)^{-7/2}, & f^{(iv)}(\xi) = -\frac{15}{16}(1+\xi)^{-7/2},
\end{array}$$

donde ξ está entre 0 y x. Del teorema de Taylor,

$$\begin{aligned}
P_3(x) &= f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 \\
&= 1 + \frac{1}{2}x - \frac{1}{4}\cdot\frac{1}{2!}x^2 + \frac{3}{8}\cdot\frac{1}{3!}x^3 = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3,
\end{aligned}$$

es el polinomio de taylor de tercer grado pedido.

Usamos el polinomio anterior para aproximar $\sqrt{1{,}1}$ y la fórmula de error para encontrar una cota para el error cometido.

$$\sqrt{1{,}1} = f(0{,}1) \approx P_3(0{,}1) = 1 + \frac{1}{2}(0{,}1) - \frac{1}{8}(0{,}1)^2 + \frac{1}{16}(0{,}1)^3 = 1{,}0488125.$$

El error está dado por $R_3(0{,}1)$, y una cota se deriva como sigue:

$$| R_3(0,1) | = \frac{| -\frac{15}{16}(1+\xi)^{-7/2} |}{4!}(0,1)^4 \leq \frac{15}{16 \cdot 24}(0,1)^4 \underset{\xi \in (0,0,1)}{max} (1+\xi)^{-7/2} \leq 3{,}91 \times 10^{-6}.$$

2.1.1. Código en Matlab de los polinomios de Taylor

```
function [C,Ye]=polytaylor(x0,Y,a,b,varargin)

% Construcción del polinomio interpolador de Taylor de una función f
% cerca de un punto x0, que pertenece al intervalo [a b].
%
%
% [C,Ye]=polytaylor(x0,f,n,varargin);
%
%
% Variables de entrada:
%
% x0 es el punto cerca del cual queremos aproximar la función.
% Y es el vector que contiene los valores de las derivadas sucesivas de la
% función en el punto x0.
% a extremo izquierdo del intervalo de aproximación.
% b extremo derecho del intervalo de aproximación.
% xe es el punto en el que evaluamos el polinomio.
%
%
% Variables de salida:
%
% C es el poliniomio interpolador de la función.
% Ye es el valor de polinomio en el punto xe.
%
%
% Ejemplo:
%
% f=cos(x).
% Sabiendo que f(0.5)=0.8776; f'(0.5)=-0.4794; f"(0.5)=-0.8776.
% y f"(0.5)=0.4794.
% [C,Ye]=polytaylor(0.5,[0.8776 -0.4794 -0.8776 0.4794],0,2,1.5)


if nargin==6
    xe=varargin{1};
else
    xe=[ ];
end
n=length(Y);
```

```
fac=1;
syms x
C=0;
for k=1:n
    fac=fac*k;
    t=((x-x0)^(k))/fac;
    C=C+Y(k)*t;
end
if xe =0
    Ye=subs(C,xe);
else
    Ye='';
end
C=char(C);

% Dibujo del polinomio de Taylor y de los puntos iniciales.

% Polinomio.

dx=(b-a)/10;
A=a-dx;
B=b+dx;
h1=ezplot(C,[A,B]);

% Puntos de resolución.

if xe =isempty(xe)
    h3=plot(xe,Ye,'g*','MarKerSize',5);
    legend([h1,h3],'Polinomio de Taylor','Evaluación en las Abscisas');
else
    legend([h1],'Polinomio de Taylor');
end
```

2.2. Interpolación de Lagrange: forma de Newton

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

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

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

Resulta que el polinomio interpolador usando diferencias divididas toma la forma

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

donde las diferencias divididas se calculan según la regla:

$$\begin{aligned} f[x_0] &= f(x_0), \\ f[x_0, x_1] &= \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}, \\ &\vdots \\ f[x_0, x_1, ..., x_n] &= \frac{f[x_1, x_2, ..., x_n] - f[x_0, x_1, ..., x_{n-1}]}{x_n - x_0}. \end{aligned}$$

Y la fórmula del error como

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

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

Como ejemplo vamos a construir el polinomio de tercer grado que pasa por los puntos $(-2,4)$,$(-1,1)$,$(2,4)$ y $(3,9)$. La expresión de dicho polinomio será:

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

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

Los tres primeros sumandos darán el mismo polinomio de Lagrange que interpola los tres primeros puntos, y añadiendo el cuarto sumando dará el polinomio interpolador de Lagrange que interpola los 4 puntos, tal y como nos asegura la propiedad de permanencia.

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

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
-2	4	$\frac{1-4}{-1-(-2)} = -3$	$\frac{1-(-3)}{2-(-2)} = 1$	$\frac{1-1}{3-(-2)} = 0$
-1	1	$\frac{4-1}{2-(-1)} = 1$	$\frac{5-1}{3-(-1)} = 1$	
2	4	$\frac{9-4}{3-2} = 5$		
3	9			

Tabla 2.1: Tabla de diferencias divididas para el ejemplo de interpolación de Newton

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

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

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

$$= 4 - 3(x+2) + 1(x+2)(x+1) + 0(x+2)(x+1)(x-2) = x^2.$$

Fórmula de error de la interpolación de Newton:

$$|f(x) - p_3(x)| = |\frac{f^{IV}(\theta)}{4!}(x+2)(x+1)(x-2)(x-3)|,$$

donde $\theta \in (min\{x,-2,-1,2,3\}, max\{x,-2,-1,2,3\}) \Rightarrow (min\{x,-2\}, max\{x,3\})$, siendo x el punto donde se calcula el polinomio interpolador. Por ejemplo, si quiero saber el error que da el polinomio de interpolación para $x = 1$,

$$|f(1) - p_3(1)| \leq |\frac{f^{IV}(\theta)}{4!}||1+2||1+1||1-2||1-3| = |\frac{f^{IV}(\theta)}{4!}|12 =$$

$$= |\frac{f^{IV}(\theta)}{2}|,$$

con

$$\theta \in (min\{1,-2\}, max\{1,3\}) = (-2,3),$$

$$f(1) \approx p_3(1).$$

2.2.1. Código en Matlab de interpolación de Lagrange

```
function y=lagrange(f,nod,x)

% Este programa calcula la interpolacion de lagrange con n puntos.
%
%
% y=lagrange(f,nod,x);
%
%
% f valores de la funcion en los nodos.
% nod nodos para interpolar.
% x vector de abcisas donde evaluar el polinomio interpolador.

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
```

2.2.2. Código en Matlab de interpolación de Lagrange en la forma de Newton

```
function [C,N,Ye]=Newton(X,Y,varargin)

% Construcción del polinomio interpolador de Newton que pasa por los n+1
% puntos (xk,yk) para k=0,1,...,n.
%
%
% [C,N,Ye]=Newton(X,Y,Xe);
%
%
% Variables de entrada:
%
% X es un vector que contiene la lista de las abscisas.
% Y es un vector que contiene la lista de las ordenadas.
% Xe vector de abscisas donde se quiere evaluar el polinomio.
%
%
% Variables de salida:
%
% C es el vector que contiene los coeficientes del polinomio interpolador
% de Newton.
% N es la matriz que contiene los coeficientes de los polinomios
% base para construir el polinomio de Newton.
% Ye evaluación del polinomio de Newton en las abscisas Xe.
%
%
% Ejemplo:
%
%[C,N,Ye]=Newton([0 1 2 3 4],[1 0.5403023 -0.4161468 -0.98999925 -0.6536436])


if nargin==3
    Xe=varargin{1};
else
    Xe='';
    Ye='';
end
```

```
w=length(X);
N=zeros(w,w);

% Cálculo de las diferencias divididas.

D=diferencias_divididas(X,Y);


% Formación de los polinomios coeficientes de Newton.

for k=1:w

    V=1;
    for j=1:k-1
        V=conv(V,poly(X(j)));
    end

    N(k,w-length(V)+1:w)=V;

end

% Cálculo de los coeficientes del polinomio interpolador de Lagrange.

C=D*N;


% Evaluación en Xe.

if  isempty(Xe)
    Ye=polyval(C,Xe);
end


syms x
for i=0:w-1
    b(i+1)=x^(w-i-1);
end
Cx=dot(C,b);


% Dibujo del polinomio de Newton y de los puntos iniciales.
```

```
% Polinomio.

dx=(X(w)-X(1))/10;
A=X(1)-dx;
B=X(w)+dx;
h1=ezplot(Cx,[A,B]);

% Puntos iniciales.

hold on
h2=plot(X,Y,'ro','MarKerSize',10);

if  isempty(Xe)
    h3=plot(Xe,Ye,'g*','MarKerSize',5);
    legend([h1,h2,h3],'Polinomio de Newton','Nodos de interpolación','Evaluación
en las Abscisas');
else
    legend([h1,h2],'Polinomio de Newton','Nodos de interpolación');
end
```

2.2.3. Código en Matlab de cálculo de diferencias divididas

```
function [fila1,tabla]=diferencias_divididas(X,Y)

% Este programa calcula las diferencias divididas.
%
%
% [fila1,tabla]=diferencias_divididas(X,Y);
%
%
% Variables de entrada:
%
% X son las abscisas.
% Y son las ordenadas.
%
%
% Variables de salida:
%
% fila1 contiene los coeficientes para formar el polinomio interpolador
% de Newton.
% tabla contiene el cuadro completo de las diferencias divididas.
%
%
% Ejemplo:
%
% Construir la tabla de diferencias divididas con los datos.
% x0=1; x1=2;
% f(x0)=1, f(x1)=-1;
% [fila1,tabla]=diferencias_divididas([1,2],[1,-1])


% Número de nodos.

n=length(X);


% Inicializamos la tabla a cero.

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

```
% Completamos las primeras dos columnas.

tabla(:,1)=X';
tabla(:,2)=Y';

% Completamos el resto de la tabla.

for i=3:n+1  % Columna.
    for j=1:n-i+2  % Fila.


        tabla(j,i)=(tabla(j+1,i-1)-tabla(j,i-1))/(tabla(j+i-2)-tabla(j,1));


    end
end

fila1=tabla(1,2:end);
```

2.3. Interpolación de Hermite

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

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

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

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

Si contamos el número de condiciones que imponemos en cada punto,

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

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

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

Diferencias divididas generalizadas:

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

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

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

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

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

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

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

El error en la interpolación de Hermite viene dado por:

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

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

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

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

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

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

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

Error de Hermite:

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

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

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

Ejemplo Hermite 2:

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

$$\begin{aligned} &p(-1) = -11, \qquad p(2) = 4, \\ &p'(-1) = 14, \qquad p'(2) = 5. \end{aligned}$$

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

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

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
$x_0 = -1$	-11	$f[x_0, x_0] = f'(-1) = 14$	$f[x_0, x_0, x_1] = -3$	$f[x_0, x_0, x_1, x_1] = 1$
$x_0 = -1$	-11	$f[x_0, x_1] = 5$	$f[x_0, x_1, x_1] = 0$	
$x_1 = 2$	4	$f[x_1, x_1] = f'(2) = 5$		
$x_1 = 2$	4			

Tabla 2.2: Tabla de diferencias divididas generalizadas para el Ejemplo 2 de interpolación de Hermite.

Tabla de diferencias generalizadas:

Donde:

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

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

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

Entonces el polinomio queda:

$$\begin{aligned} p_3(x) &= f(x_0) + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 \\ &+ f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1) \\ &= -11 + 14(x + 1) - 3(x + 1)^2 + (x + 1)^2(x - 2) \\ &= x^3 - 3x^2 + 5x - 2, \end{aligned}$$

y la fórmula del error de Hermite:

$$f(x) - p_3(x) = \frac{f^4(\theta)}{4!}(x - x_0)^2(x - x_1)^2,$$

$$f(x) - p_3(x) = \frac{f^4(\theta)}{4!}(x + 1)^2(x - 2)^2,$$

$$\theta \in (min\{x, -1\}, max\{x, 2\}).$$

2.3.1. Código en Matlab de interpolación de Hermite

```
function [C,Ye]=Hermite(Xnodos,Ycondiciones,Xe)

% Esta función construye el polinomio interpolador de Hermite y lo
% evalua en ciertas abscisas Xe.
%
%
% [C,Ye]=Hermite(Xnodos,Ycondiciones,Xe);
%
%
% Variables de entrada:
%
% Xnodos son las abscisas donde se imponen las condiciones.
% Ycondiciones es una celda que contiene tantos vectores como nodos.
% Cada vector contiene los datos [f(xi),f'(xi),f"(xi)/2!,
% ...,f^(ni)(xi)/(ni)!].
% Xe valores de las abscisas donde se quiere evaluar el polinomio.
%
%
% Variables de salida:
%
% C coeficientes del polinomio de Hermite.
% Ye valores del polinomio en las abscisas Xe.
%
%
% Ejemplo:
%
% Polinomio que satisface las condiciones.
% x0=1; x1=2;
% p(x0)=1, p'(x0)=2 .
% p(x1)=-1, p'(x1)=3, p"(x1)=4.
% y su evaluación en x=1.5.
% [C,Ye]=Hermite([1,2],{[1,2],[-1,3,2]},1.5)


% Tabla de diferencias divididas generalizadas.

[D,tabla]=diferencias_divididas_generalizadas(Xnodos,Ycondiciones);

n=length(Xnodos);
```

```
Nodos=tabla(:,1);
w=length(Nodos);
N=zeros(w,w);

% Formación de los polinomios coeficientes de Hermite.

for k=1:w

    V=1;
    for j=1:k-1
        V=conv(V,poly(Nodos(j)));
    end

    N(k,w-length(V)+1:w)=V;

end


% Cálculo de los coeficientes del polinomio interpolador de Hermite.

C=D(2:end)*N;


% Evaluación en Xe.

if  isempty(Xe)
    Ye=polyval(C,Xe);
else
    Ye=[];
end
```

2.3.2. Código en Matlab de cálculo de diferencias divididas generalizadas

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

% Este programa calcula las diferencias divididas generalizadas.
%
%
% [fila1,tabla,nc]=diferencias_divididas_generalizadas(nodos,condiciones);
%
%
% Variables de entrada:
%
% Nodos son las abscisas donde se imponen las condiciones.
% Condiciones es una celda que contiene tantos vectores como nodos.
% Cada vector contiene los datos [f(xi),f'(xi),f"(xi)/2!,
% ...,f^(ni)(xi)/(ni)!].
%
%
% Variables de salida:
%
% Fila1 contiene los coeficientes para formar el polinomio interpolador
% de Hermite.
% Tabla contiene el cuadro completo de las diferencias divididas
% nc número de condiciones que hay en cada nodo.
%
%
% Ejemplo:
%
% Construir la tabla de diferencias divididas generalizadas con los datos:
% x0=1; x1=2.
% f(x0)=1, f'(x0)=2.
% f(x1)=-1, f'(x1)=3, f"(x1)=4.
% [fila1,tabla,nc]=diferencias_divididas_generalizadas([1,2],{[1,2],[-1,3,2]})


% Número de nodos.

n=length(nodos);

% Número de condiciones en cada nodo.
```

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


% Inicializamos la tabla a cero.

tabla=zeros(nc_total,nc_total+1);


% Completamos las primeras dos columnas.

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

% Completamos el resto de la tabla.

for i=3:nc_total+1  % Columna.
    for j=1:nc_total-i+2  % Fila.

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

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

        end

    end
end
```

fila1=tabla(1,:);

2.4. Hermite para funciones discontinuas

2.4.1. Detección de las discontinuidades

Cuando se trabaja con datos discontinuos, dependiendo de las discretización que se esté utilizando puede ser posible detectar la posición de las discontinuidades. Si la función es suave por partes con una discontinuidad de salto aislada en la función y la reconstrucción se hace utilizando muestras de la función $f_i = f(x_i)$, entonces no hay esperanza de detectar la posición de la discontinuidad dentro del intervalo donde está contenido, ya que se ha perdido la posición exacta en el proceso de muestreo. Con el fin de detectar discontinuidades de salto, el muestreo de valor puntual debe ser reemplazado por un promedio local, por medias en celda (cell-averages) [4]. Cuando la discontinuidad no es de salto sino de esquina, es decir, una discontinuidad en la primera derivada, entonces la discretización por valores puntuales es adecuada. Y es en este caso en el que no vamos a centrar.

ENO-Subcell resolution [9] y [4] usa una estrategia para detectar discontinuidades en la primera derivada dentro de intervalos (asumiendo que las discontinuidades están lo suficientemente alejadas unas de otras). Intenta mejorar las aproximaciones que no usan información de ambos lados de la discontinuidad, como la técnica ENO, que cerca de una esquina utiliza sólo información de un lado. Debido a este hecho, la técnica de resolución subcelda es una buena mejora a priori de los algoritmos clásicos de multiresolución a costa de ser más caro desde el punto de vista computacional.

La técnica de resolución subcelda fue introducida por Harten en [9] con el objetivo de capturar los choques de flujo con tanta exactitud como fuera posible en el contexto de las leyes de conservación. Se suponía que era una mejora de los esquemas ENO y, de hecho, lo es en muchos casos. Sin embargo, presenta un inconveniente importante, el paso final de esta técnica consiste en una extrapolación, la cual se sabe que es un proceso inestable. En [5], los autores aplican la técnica de resolución subcelda con el fin de mejorar los resultados obtenidos por la interpolación no lineal ENO cuando se trabaja con imágenes en escala de grises. También presentan una comparación entre los esquemas de resolución ENO-subcell y otros métodos para la interpolación de imágenes en el contexto de las medias en celda. En [3], los autores comparan los esquemas de resolución PPH-subcell con los esquemas de resolución ENO-subcell y otros métodos. Nuestro objetivo es combinar una técnica de detección de discontinuidad que supera los problemas de la resolución subcelda con el métido de Hermite segmentario, con el fin de elevar el orden de la

aproximación cerca de las discontinuidades. Más específicamente haremos uso de la estrategia analizada en [6]. Aunque el método es válido para mallados no uniformemente espaciados, pasamos a explicar el caso relativo a mallados uniformes, ya que la adaptación es inmediata y las expresiones toman una forma más simplificada. Siendo h el espaciado de la celda, el algoritmo hace uso de las diferencias finitas (numerador de las diferencias divididas) de segundo orden,

$$\triangle_h^2 f(x) = f(x) - 2f(x+h) + f(x+2h),$$

como indicadores de la presencia de celdas sospechosas de contener una discontinuidad. El mecanismo de detección define un conjunto de intervalos que están etiquetados como B, que son candidatos potenciales de contener singularidades. En [6] los autores proponen las siguientes reglas,

1. Si

$$\mid \triangle_h^2 f((k-1)h) \mid > \mid \triangle_h^2 f((k-1 \pm n)h) \mid, \quad n = 1, \cdots, m,$$

Entonces los intervalos $I_{k-1} = [x_{k-1}, x_k]$ y $I_k = [x_k, x_{k+1}]$ son etiquetados como B. Este caso considera también la situación donde la discontinuidad cae en un punto del mallado. En principio con esta única condición no es posible saber entre que intervalos está contenida la singularidad, por lo que ambos son etiquetados como sospechosos.

2. Si

$$\mid \triangle_h^2 f(kh) \mid > \mid \triangle_h^2 f((k+n)h) \mid, \quad n = 1, \cdots, m-1,$$

y,

$$\mid \triangle_h^2 f(k-1) \mid > \mid \triangle_h^2 f((k-1-n)h) \mid, \quad n = 1, \cdots, m-1,$$

entonces el intervalo es etiquetado como B. En este caso, las dos diferencias divididas más grandes incluyen el intervalo I_k, que es un candidato para contener la discontinuidad.

Todos los demás intervalos están etiquetados como G y se supone que no contienen una singularidad. El algoritmo está diseñado de tal manera que para un h lo suficientemente pequeño, el intervalo que contiene la singularidad ha sido etiquetado como B, mientras que todos los intervalos etiquetados como G están en regiones suaves de f. Es posible que un intervalo I_k esté etiquetado como B en una región suave. Para detectar la posición de la discontinuidad, elegimos una aproximación simple: construimos polinomios de Lagrange de orden m a la izquierda y a la derecha del intervalo sospechoso, etiquetado como B o BB. Entonces, definimos una función G como la diferencia entre los dos polinomios, y se puede demostrar que que hay una única raíz de esta función dentro del intervalo sospechoso. Ahora, la posición de la potencial discontinuidad puede ser obtenida fácilmente encontrando las raíces de un polinomio para tamaños de h lo suficientemente pequeños. Usando el algoritmo de la bisección o el algoritmo de Newton con una condición inicial obtenida por el método de la bisección, se puede obtener fácilmente la posición de la discontinuidad hasta la precisión requerida. Si estamos trabajando en el entorno de discretización por valores puntuales seremos capaces de dectectar las esquinas con el algoritmo indicad. En el entorno de discretización por medias en celda se pueden detectar y localizar discontinuidades de salto utilizando la función primitiva. Otra cuestión es que la técnica mostrada anteriormente está diseñada específicamente para aproximación e interpolación, aplicaciones en las que los datos se distribuyen generalmente a lo largo de mallados no necesariamente uniformemente. Trabajando con Hermite es deseable que la detección de discontinuidades y obtención de su posición se haga usando sitios arbitrariamente distribuidos. El mecanismo de detección propuesto en [6] y explicado antes puede ser directamente adaptado a mallados no uniformes usando diferencias divididas en lugar de diferencias finitas.

2.4.1.1. Código en Matlab de detección de los posibles candidatos a contener una discontinuidad

```
function candidatos=BB(d2,N)

% Este programa marca la posición de los posibles candidatos a contener
% una discontinuidad.
%
%
% candidatos=BB(d2,N);
%
%
% Variables de entrada:
%
% d2 vector que contiene las diferencias segundas de los datos.
% N indica el orden N+1 de aproximación requerido.
%
%
% Variables de salida:
%
% Candidatos vector que contiene en cada posición:
% 0 si ese intervalo no es candidato.
% 1 si ese intervalo contiene una
% discontinuidad justo en el nodo inicial.
% 2 si ese intervalor contiene una
% discontinuidad intermedia.



n=length(d2);
candidatos=zeros(size(d2));
peso=1.5;


for j=N+2:n-N-1

    M=max(abs([d2(j-N-1:j-1),d2(j+1:j+N+1)]));

    if abs(d2(j))>peso*M
        candidatos(j)=1;
```

```
    end

    Mi=max(abs(d2(j-N-1:j-1)));

    Md=max(abs(d2(j+2:j+N+1)));

    if abs(d2(j+1))>peso*Md & abs(d2(j))>peso*Mi
        candidatos(j)=2;
    end

end
```

2.4.1.2. Código en Matlab de la función que aproxima la raíz de la función diferencia de los dos polinomios de Lagrange construidos a partir de los datos

```
function [raiz]=Mu(nodosI,yI,nodosD,yD)

% Esta función aproxima la raíz de la función diferencia de los
% dos polinomios de Lagrange construidos a partir de los datos
% (nodosI,yI) y (nodosD,yD) respectivamente.
%
%
% [raiz]=Mu(nodosI,yI,nodosD,yD);
%
%
% Variables de entrada:
%
% nodosI valores de las abscisas para construir el polinomio de Lagrange
% de orden N de la izquierda.
% yI valores de la función en los nodosI.
% nodosD valores de las abscisas para construir el polinomio de Lagrange
% de orden N de la derecha.
% yD valores de la función en los nodosD.
%
%
% Variables de salida:
%
% raiz aproximación obtenida de la raíz por el método de Bisección.



tol=10^(-15);
tolf=10^(-20);


%Condiciones de convergencia.
%1.f(a)*f(b)<0.

n=length(nodosI);


a=nodosI(n);
```

```
b=nodosD(1);

pa=yI(n);
qb=yD(1);




fa=pa-lagrange(yD,nodosD,a);
fb=lagrange(yI,nodosI,b)-qb;




if abs(fa)<tolf
    raiz=a;
    return;
end

if abs(fb)<tolf
    raiz=b;
    return;
end

if fa*fb>0
    error ('La raiz no se encuentra en el intervalo(a,b)')
end



%Comenzamos las iteraciones.

n=ceil(log(abs(b-a)/tol)/log(2.0));
err=(b-a)/(2^(n));


for i=1:n
```

```
    c=0.5*(a+b);
    fc=lagrange(yI,nodosI,c)-lagrange(yD,nodosD,c);


    if abs(fc)<tolf
        raiz=c;
        return
    else
        if fb*fc<0
            a=c;
            fa=fc;
        else
            b=c;
            fb=fc;
        end
    end

end

raiz=(a+b)/2;
```

2.4.2. Obtencion del valor de f(x) y sus derivadas en las discontinuidades

En esta sección, veremos un método para obtener el valor de la función y sus derivadas por la izquierda y por la derecha de una discontinuidad. Estos valores serán necesarios para imponer las condiciones de contorno que dependerán del tipo de Hermite escogido.

2.4.2.1. Obtención del valor de f(x) y sus derivadas en discontinuidades usando valores puntuales

Una vez conocida la posición de la discontinuidad, usando la técnica explicada anteriormente (sección 2.4.1), podemos intentar obtener información sobre la función y sus derivadas por la izquierda y por la derecha de la discontinuidad que hemos detectado. Asumiremos que estamos trabajando con discontinuidades de esquina. Como hemos mencionado antes y como analizaremos después, las discontinuidades de salto pueden ser detectadas y posicionadas si las muestras son obtenidas mediante las medias en celdas de f(x).

Por ejemplo, si trabajamos con Hermite con dos puntos y condiciones en la función en la función y en su primera derivada en cada uno de los puntos, se puede obtener el valor de la función y sus derivadas por la izquierda y por la derecha en la localización x^* de la discontinuidad. Esto asegurará con precisión $O(h^4)$ igual a la obtenida por este método de Hermite particular. Supongamos que la discontinuidad está colocada en x^*, a una distancia α de x_j en el intervalo $[x_j, x_{j+1}]$, consideramos los valores $\{f^+_{j-3}, f^+_{j-2}, f^+_{j-1}, f^+_j, f^-_{j+1}, f^-_{j+2}, f^-_{j+2}, f^-_{j+3}, f^-_{j+4}\}$ colocadas en las posiciones $\{x_{j-3}, x_{j-2}, x_{j-1}, x_j, x_{j+1}, x_{j+2}, x_{j+2}, x_{j+3}, x_{j+4}\}$. Entonces, podemos obtener los valores de las derivadas de f desde ambos lados de la discontinuidad sólo usando Taylor alrededor de x^* y estableciendo los siguientes sistemas de ecuaciones,

$$
\begin{aligned}
f_j^+ &= f^+(x^*) - f_x^+(x^*)\alpha + \frac{1}{2}f_{xx}^+(x^*)\alpha^2 - \frac{1}{3!}f_{xxx}^+(x^*)\alpha^3 \\
f_{j-1}^+ &= f^+(x^*) - f_x^+(x^*)(h+\alpha) + \frac{1}{2}f_{xx}^+(x^*)(h+\alpha^2) - \frac{1}{3!}f_{xxx}^+(x^*)(h+\alpha)^3 \\
f_{j-2}^+ &= f^+(x^*) - f_x^+(x^*)(2h+\alpha) + \frac{1}{2}f_{xx}^+(x^*)(2h+\alpha^2) - \frac{1}{3!}f_{xxx}^+(x^*)2(h+\alpha)^3 \\
f_{j-3}^+ &= f^+(x^*) - f_x^+(x^*)(3h+\alpha) + \frac{1}{2}f_{xx}^+(x^*)(3h+\alpha^2) - \frac{1}{3!}f_{xxx}^+(x^*)(3h+\alpha)^3, \qquad (1)
\end{aligned}
$$

y

$$
\begin{aligned}
f_{j+1}^- &= f^-(x^*) - f_x^-(x^*)(h-\alpha) + \frac{1}{2}f_{xx}^-(x^*)(h-\alpha^2) - \frac{1}{3!}f_{xxx}^-(x^*)(h-\alpha)^3 \\
f_{j+2}^- &= f^-(x^*) - f_x^-(x^*)(2h-\alpha) + \frac{1}{2}f_{xx}^-(x^*)(2h-\alpha^2) - \frac{1}{3!}f_{xxx}^-(x^*)(2h-\alpha)^3 \\
f_{j+3}^- &= f^-(x^*) - f_x^-(x^*)(3h-\alpha) + \frac{1}{2}f_{xx}^-(x^*)(3h-\alpha^2) - \frac{1}{3!}f_{xxx}^-(x^*)(3h-\alpha)^3 \\
f_{j+4}^- &= f^-(x^*) - f_x^-(x^*)(4h-\alpha) + \frac{1}{2}f_{xx}^-(x^*)(4h-\alpha^2) - \frac{1}{3!}f_{xxx}^-(x^*)(4h-\alpha)^3, \qquad (2)
\end{aligned}
$$

con $\alpha = x^* - x_j$. Por simplicidad hemos supuesto un mallado uniforme. Invirtiendo los dos sistemas (1) y (2), donde $f^+(x^*), f_x^+(x^*), f_{xx}^+(x^*), f_{xxx}^+(x^*)$ y $f^-(x^*), f_x^-(x^*), f_{xx}^-(x^*), f_{xxx}^-(x^*)$ son desconocidos, los valores de la función y sus derivadas pueden ser fácilmente obtenidos.
Es interesante parar aquí y pensar sobre la precisión obtenida cuando aproximamos los valores en la discontinuidad en la forma explicada. En este caso vamos a trabajar con un mallado uniforme h. Los resultados para un mallado no uniforme son muy similares, y pueden obtenerse fácilmente a partir de las expresiones ofrecidas en este trabajo. Podemos expresar los sistemas (1) y (2) en forma matricial con el correspondiente error de truncamiento local,

$$
\begin{pmatrix} f_j^+ \\ f_{j-1}^+ \\ f_{j-2}^+ \\ f_{j-3}^+ \end{pmatrix} = \begin{pmatrix} 1 & \alpha & \frac{1}{2}\alpha^2 & -\frac{1}{3!}\alpha^3 \\ 1 & (h+\alpha) & \frac{1}{2}(h+\alpha)^2 & -\frac{1}{3!}(h+\alpha)^3 \\ 1 & (h+\alpha) & \frac{1}{2}(2h+\alpha)^2 & -\frac{1}{3!}(2h+\alpha)^3 \\ 1 & (h+\alpha) & \frac{1}{2}(3h+\alpha)^2 & -\frac{1}{3!}(3h+\alpha)^3 \end{pmatrix} \begin{pmatrix} f^+(x^*) \\ f_x^+(x^*) \\ f_{xx}^+(x^*) \\ f_{xxx}^+(x^*) \end{pmatrix} + \begin{pmatrix} O(h^4) \\ O(h^4) \\ O(h^4) \\ O(h^4) \end{pmatrix}. \qquad (3)
$$

Obteniendo ahora la inversa del sistema matricial:

$$A = \begin{pmatrix} \frac{6h^3+11h^2\alpha+6h\alpha^2+\alpha^3}{6h^3} & -\frac{\alpha(6h^2+5h\alpha+\alpha^2)}{2h^2} & -\frac{\alpha(3h^2+4h\alpha+\alpha^2)}{2h^3} & -\frac{\alpha(2h^2+3h\alpha+\alpha^2)}{6h^3} \\ \frac{11h^2+12h\alpha+3\alpha^2}{6h^3} & -\frac{6h^2+10h\alpha+3\alpha^2}{2h^3} & \frac{3h^2+8h\alpha+3\alpha^2}{2h^3} & -\frac{2h^2+6h\alpha+3\alpha^2}{6h^3} \\ \frac{2h+\alpha}{h^3} & -\frac{5h+3\alpha}{h^3} & \frac{4h+3\alpha}{h^3} & -\frac{h+\alpha}{h^3} \\ h^{-3} & -3h^{-3} & 3h^{-3} & -h^{-3} \end{pmatrix}. \quad (4)$$

Como $\alpha = O(h)$, a partir de la expresión de A en (3), es fácil obtener la precisión alcanzada,

$$\begin{pmatrix} f^+(x^*) \\ f_x^+(x^*) \\ f_{xx}^+(x^*) \\ f_{xxx}^+(x^*) \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} f_j^+ \\ f_{j-1}^+ \\ f_{j-2}^+ \\ f_{j-3}^+ \end{pmatrix} + \begin{pmatrix} O(h^4) \\ O(h^4) \\ O(h^4) \\ O(h^4) \end{pmatrix}. \quad (5)$$

Haciendo esta multiplicación matricial, directamente llegamos a la expresión para la función y sus derivadas en la discontinuidad $x = x^*$. Para el sistema de (2), se sigue el mismo procedimiento,

$$\begin{pmatrix} f_{j+1}^- \\ f_{j+2}^- \\ f_{j+3}^- \\ f_{j+4}^- \end{pmatrix} = \begin{pmatrix} 1 & h-\alpha & \frac{(h-\alpha)^2}{2} & \frac{(h-\alpha)^3}{6} \\ 1 & 2h-\alpha & \frac{(2h-\alpha)^2}{2} & \frac{(2h-\alpha)^3}{6} \\ 1 & 3h-\alpha & \frac{(3h-\alpha)^2}{2} & \frac{(3h-\alpha)^3}{6} \\ 1 & 4h-\alpha & \frac{(4h-\alpha)^2}{2} & \frac{(4h-\alpha)^3}{6} \end{pmatrix} \begin{pmatrix} f^-(x^*) \\ f_x^-(x^*) \\ f_{xx}^-(x^*) \\ f_{xxx}^-(x^*) \end{pmatrix} + \begin{pmatrix} O(h^4) \\ O(h^4) \\ O(h^4) \\ O(h^4) \end{pmatrix}. \quad (6)$$

La inversa de la matriz del sistema (6) solo depende de α y h.

$$B = \begin{pmatrix} -\frac{-24h^3+26h^2\alpha-9h\alpha^2+\alpha^3}{6h^3} & -\frac{(-h+\alpha)(12h^2-7h\alpha+\alpha^2)}{2h^3} & -\frac{(-h+\alpha)(8h^2+-6h\alpha+\alpha^2)}{2h^3} & \frac{(-h+\alpha)(6h^2-5h\alpha+\alpha^2)}{6h^3} \\ -\frac{26h^2-18h\alpha+3\alpha^2}{6h^3} & \frac{19h^2-16h\alpha+3\alpha^2}{2h^3} & -\frac{14h^2-14h\alpha+3\alpha^2}{2h^3} & \frac{11h^2-12h\alpha+3\alpha^2}{6h^3} \\ -\frac{-3h+\alpha}{h^3} & \frac{-8h+3\alpha}{h^3} & -\frac{7h+3\alpha}{h^3} & \frac{-2h+\alpha}{h^3} \\ -h^{-3} & 3h^{-3} & -3h^{-3} & h^{-3} \end{pmatrix}. \quad (7)$$

A continuación, como $\alpha = O(h)$, observando la expresión de B en (7), es fácil ver que,

$$\begin{pmatrix} f^-(x^*) \\ f_x^-(x^*) \\ f_{xx}^-(x^*) \\ f_{xxx}^-(x^*) \end{pmatrix} = \begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{pmatrix} \begin{pmatrix} f_{j+1}^- \\ f_{j+2}^- \\ f_{j+3}^- \\ f_{j+4}^- \end{pmatrix} + \begin{pmatrix} O(h^4) \\ O(h^4) \\ O(h^4) \\ O(h^4) \end{pmatrix}. \quad (8)$$

Así, tenemos la exactitud de la aproximación de f y sus derivadas en $x = x^*$ a ambos lados de la discontinuidad.

2.4.2.2. Código en Matlab de la función que obtiene las derivadas por la izquierda o por la derecha en la aproximación

```
function df=get_derivatives(x,fx,d)

% Esta función obtiene las derivadas por la izquierda o por la derecha en la
% aproximación calculada a la discontinuidad.
%
%
% df=get_derivatives(x,fx,d);
%
%
% Variables de entrada:
%
% x abscisas donde se plantean los desarrollos de Taylor.
% [x_j,x_{j-1}, ...,x_{j-N-1}] si es por el lado izquierdo.
% [x_{j+1},x_{j+2},...,x_{j+N}] si es por el lado derecho.
% fx valores conocidos de la función en las abscisas x.
% d aproximación calculada a la discontinuidad.
%
%
% Variables de salida:
%
% df aproximación de orden N a las derivadas por un lado en la
% discontinuidad.



% Orden de aproximación.

N=length(x);

% x-x*.

c=x-d;

% Matriz de Vandermonde.

A=fliplr(vander(c));

for i=2:N
```

```
    A(:,i)=A(:,i)/factorial(i-1);
end

% Vector de términos independientes.

b=fx';


% Resolvemos el sistema lineal.

df=A\b;
```

2.4.3. Hermite a trozos entre discontinuidades

El algoritmo que se va a exponer en la sección 2.4.3.1 va a consistir en tres pasos:

1. Detección de las discontinuidades, explicado anteriormente en la sección 2.4.1. Si no hay discontinuidades realiza Hermite segmentario en todo el intervalo.

2. Cálculo de los datos necesarios para las condiciones de contorno en las discontinuidades calculadas.
   Según el Hermite escogido necesitaremos el valor de la función y sus derivadas sucesivas hasta cierto orden en el nuevo nodo añadido, es decir, en la posición de la discontinuidad encontrada. Para ello utilizamos el método explicado en la sección 2.4.2.

3. Cómputo de la reconstrucción entre cada dos discontinuidades.
   Una vez calculadas las posiciones de las discontinuidades presentes pasamos a hacer la reconstrucción. Supongamos que se han encontrado s discontinuidades en las posiciones $d_1, d_2, ..., d_s$ con $a < d_1 < d_2 < .... < d_s < b$.
   La reconstrucción se lleva a cabo en tres pasos:

   a) Se utiliza el algoritmo de Hermite segmentario en el intervalo $[a, d_1]$.

   b) Para cada $i = 1, ..., s-1$ se utiliza el algoritmo de Hermite segmentario en el intervalo $[d_i, d_{i+1}]$.

   c) Se utiliza el algoritmo de Hermite segmentario en el intervalo $[d_s, b]$.

2.4.3.1. Código en Matlab de la función que calcula la reconstrucción de Hermite a trozos de una función discontinua dada

```
function er=Hermite_D(fcell,Dcell,n,mev,n_condi,l,r)


% Esta función calcula la reconstrucción de Hermite a trozos de una función
% discontinua dada, partiendo justo en las discontinuidades para no
% cruzarlas.
%
%
% er=Hermite_D(fcell,Dcell,n,mev,n_condi,l,r);
%
%
% Variables de entrada:
%
% fcell variable de celda conteniendo las expresiones de las funciones
% entre discontinuidades.
% Dcell variable de celda conteniendo las posiciones de las
% discontinuidades. Dcell empieza por el extremo inicial a y
% acaba en el extremo final b.
% n número de puntos en que discretizamos el intervalo [a,b].
% mev número de puntos del mallado fino (m >> n) donde discretizamos
% el intervalo [a,b].
% n_condi vector que indica el número de derivadas en cada punto del
% conjunto de puntos usado para interpolar localmente.
% l número de puntos a la izquierda de x_j incluyendo x_j.
% r número de puntos a la derecha de x_{j+1} incluyendo x_{j+1}.
%
%
% Variables de salida:
%
% er error cometido entre la reconstrucción construida y la función
% original cuando evaluamos ambas en un mallado fino.
%
%
% Ejemplo 1:
% er=Hermite_D({10*sin(pi*x),-50*cos(pi*x)+10,20*sin(pi*x)-40,-40+
% 5*sin(pi*x)},{0,0.5,2,3,2*pi},150,1000,[1,1,1,1],2,2);
% Ejemplo 2:
```

```
% er=Hermite_D({-50*cos(pi*x)+10,20*sin(pi*x)-40},{0,2,2*pi},150,1000,
%,[1,1,1,1],2,2);

syms x

close all % Cierra todas las ventanas existentes.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construimos la función.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


m=length(Dcell); % Número de discontinuidades más los dos extremos a
y b.

a=Dcell{1}; % Extremo izquierdo.
b=Dcell{m}; % Extremo derecho.
h=(b-a)/n; % Espaciado entre nodos de reconstrucción.

xa=linspace(a,b,n); % Mallado usado para aproximar.

xev=linspace(a,b,mev); % Mallado usado para evaluar y medir errores de

% aproximación.


y=[]; % Inicializamos el vector con los valores numéricos de la función.
raiz=[];

dizq=Dcell{1};

% Número de condiciones. Coincide con el orden de aproximación local.


N=sum(n_condi)+length(n_condi);

% Derivada de mayor orden considerada.

md=max(n_condi)+1;

H=zeros(md,n); % Matriz que contiene en cada fila los datos de
```

```
% la función y sus derivadas.

fevE=zeros(1,mev); % Vector que contiene los datos de la función
% en los nodos de evaluación.

fevD=zeros(1,mev); % Vector que contiene los datos de la
% reconstrucción en los nodos de evaluación.



% Evaluación de la función en los nodos de evaluación.

n_nue=0;

for i=2:m

    dder=Dcell{i};
    ind_aux=find( (dizq <=xev) & (xev< dder));
    x_aux=xev(ind_aux);
    n_ant=n_nue;
    n_nue=n_nue+length(x_aux);
    faux=fcell{i-1};
    y_aux=double(subs(faux,{x},x_aux));
    fevE(n_ant+1:n_nue)=y_aux;
    dizq=dder;

end
yb=subs(faux,{x},b);
fevE(mev)=yb;


% Evaluación de la función en los nodos de reconstrucción.

dizq=Dcell{1};
n_nue=0;

for i=2:m

    dfaux{i-1}{1}=fcell{i-1};
    dder=Dcell{i};
    ind_aux=find( (dizq <=xa) & (xa< dder));
```

```
    x_aux=xa(ind_aux);
    n_ant=n_nue;
    n_nue=n_nue+length(x_aux);
    for j=1:md
        faux=dfaux{i-1}{j};
        y_aux=subs(faux,{x},x_aux);
        H(j,n_ant+1:n_nue)=y_aux;
        dfaux{i-1}{j+1}=diff(faux);
    end

    dizq=dder;

end


for j=1:md
    faux=dfaux{i-1}{j};
    yb=subs(faux,{x},b);
    H(j,n)=yb;
end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dibujamos la función y las derivadas.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


for j=1:md
    figure
    minH=min(H(j,:)); maxH=max(H(j,:));
    plot(xa,H(j,:),'-')
    axis([xa(1)-0.1*(xa(end)-xa(1)) xa(end)+0.1*(xa(end)-xa(1)) minH-0.1*
(maxH-minH) maxH+0.1*(maxH-minH)])
        str=['Función después de derivar',blanks(1), num2str(j-1),blanks(1),'veces
la función original'];
    title(str);
end

minH=min(H(1,:)); maxH=max(H(1,:));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Detectamos dónde están las discontinuidades.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y=H(1,:);

% Primero marcamos los intervalos sospechosos.

% calculamos las diferencias finitas de segundo orden.

d2=diff(y,2);
candidatos=BB(d2,N-1);

% Establecemos las condiciones del artículo de Rosa Donat.

i=0;
for j=1:length(d2)

    if candidatos(j)==1
        i=i+1;
        nodosI=xa(j-N+2:j+1);
        yI=y(j-N+2:j+1);
        nodosD=xa(j+2:j+N+1);
        yD=y(j+2:j+N+1);
        indice_raiz(i)=j+1;
        xa(indice_raiz(i)-1:indice_raiz(i)+1)
        [raiz(i)]=Mu(nodosI,yI,nodosD,yD);
        candidatos(j+1)=0;
    elseif candidatos(j)==2
        i=i+1;
        nodosI=xa(j-N+2:j+1);
        yI=y(j-N+2:j+1);
        nodosD=xa(j+2:j+N+1);
        yD=y(j+2:j+N+1);
        indice_raiz(i)=j+1;
        xa(indice_raiz(i)-1:indice_raiz(i)+1)
        [raiz(i)]=Mu(nodosI,yI,nodosD,yD);
        candidatos(j+1)=0;
```

```
    end

end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aproximación de la función por trozos entre discontinuidades si las hay.
%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if isempty(raiz)

    % Inicializamos el vector que contendrá los valores de la función
    % reconstruida para ir completándolos.

    fevD=[];

    % Número de raíces intermedias encontradas.

    nr=length(raiz);


    % Partimos nuestro intervalo en trozos en cada raíz.

    % Primer trozo: desde el inicio a la primera raíz.

    % Cálculo del valor de la función y las derivadas en la primera raíz por
    % la izquierda.

    dfizq=get_derivatives(xa(indice_raiz(1)-N+1:indice_raiz(1)),
H(:,indice_raiz(1)-N+1:indice_raiz(1)),raiz(1));
    dfizq=dfizq(1:md)';

    xa_aux=[xa(1:indice_raiz(1)),raiz(1)];
    H_aux=[H(:,1:indice_raiz(1)),dfizq];
    ind_xev_aux=find(xev<=raiz(i));
    xev_aux=xev(ind_xev_aux);
    [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);


    fevD=[fevD,fevD_aux];
```

```
    % Trozos centrales entre raíces.

    for i=1:nr-1

        % Cálculo del valor de la función y las derivadas en la raíz i por la
        % derecha y de la raíz i+1 por la izquierda.

        dfder=get_derivatives(xa(indice_raiz(i)+1:indice_raiz(i)+N)
,H(:,indice_raiz(i)+1:indice_raiz(i)+N),raiz(i));
        dfder=dfder(1:md)';

        dfizq=get_derivatives(xa(indice_raiz(i+1)-N+1:indice_raiz(i+1)),
H(:,indice_raiz(i+1)-N+1:indice_raiz(i+1)),raiz(i+1));
        dfizq=dfizq(1:md)';


        xa_aux=[raiz(i),xa(indice_raiz(i)+1:indice_raiz(i+1)),raiz(i+1)];
        H_aux=[dfder,H(:,indice_raiz(i)+1:indice_raiz(i+1)),dfizq];

        ind_xev_aux=find((xev>raiz(i)) & (xev <= raiz(i+1)));
        xev_aux=xev(ind_xev_aux);

        [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
        fevD=[fevD,fevD_aux];


    end

    % Último trozo: entre la última raíz y el extremo b.

    % Cálculo del valor de la función y las derivadas en la última raíz por la
    % derecha.

    dfder=get_derivatives(xa(indice_raiz(nr)+1:indice_raiz(nr)+N),
H(:,indice_raiz(nr)+1:indice_raiz(nr)+N),raiz(nr));
    dfder=dfder(1:md)';

    xa_aux=[raiz(nr),xa(indice_raiz(nr)+1:end)];
    H_aux=[dfder,H(:,indice_raiz(nr)+1:end)];
```

```
    ind_xev_aux=find(xev>raiz(nr));
    xev_aux=xev(ind_xev_aux);

    [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
    fevD=[fevD,fevD_aux];
    fevD=double(fevD);


else % Hace Hermite.

    [fevD]=Hermite_segmentario(xa,H,n_condi,l,r,xev);
    fevD=double(fevD);



end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dibujamos la función y la reconstrucción obtenida.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


figure
plot(xev,fevD,'r');
hold on
plot(xev,fevE,'b');
axis([xa(1)-0.1*(xa(end)-xa(1)) xa(end)+0.1*(xa(end)-xa(1)) minH-0.1*(maxH-
minH) maxH+0.1*(maxH-minH)]);
title('Reconstrucción por Hermite a trozos');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Imprimimos la norma del error.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % for i=1:length(raiz)
    % h
    % N
```

```
    % h^(N-2)
    % raiz(i)-h^(N-3)
    % ind_aux=find((xev>=raiz(i)-h^(N-1)) & (xev<=raiz(i)+h^(N-1)))
    % figure
    % plot(xev(ind_aux),fevE(ind_aux),'b');
    % hold on
    % plot(xev(ind_aux),fevD(ind_aux),'r');

end


er=max(abs(fevE(1:end)-fevD(1:end)));



end
```

2.4.3.2. Algoritmo de la función que calcula la reconstrucción de Hermite a trozos de una función discontinua para un mallado no uniforme

```
function er=Hermite_nu_D(fcell,Dcell,n,mev,n_condi,l,r)


% Esta función calcula la reconstrucción de Hermite a trozos de una función
% discontinua dada, partiendo justo en las discontinuidades para no
% cruzarlas.
%
%
% er=Hermite_nu_D(fcell,Dcell,n,mev,n_condi,l,r);
%
%
% Variables de entrada:
%
% fcell variable de celda conteniendo las expresiones de las funciones
% entre discontinuidades.
% Dcell variable de celda conteniendo las posiciones de las
% discontinuidades. Dcell empieza por el extremo inicial a y
% acaba en el extremo final b.
% n número de puntos en que discretizamos el intervalo [a,b].
% mev número de puntos del mallado fino (m >> n) donde discretizamos
% el intervalo [a,b].
% n_condi vector que indica el número de derivadas en cada punto del
% conjunto de puntos usado para interpolar localmente.
% l número de puntos a la izquierda de x_j incluyendo x_j.
% r número de puntos a la derecha de x_{j+1} incluyendo x_{j+1}.
%
%
% Ejemplo 1:
%
% er=Hermite_nu_D({10*sin(pi*x),-50*cos(pi*x)+10,20*sin(pi*x)-40,
% -40+5*sin(pi*x)},{0,0.5,2,3,2*pi},150,1000,[1,1,1,1],2,2);
%
%
% Ejemplo 2:
%
% er=Hermite_nu_D({-50*cos(pi*x)+10,20*sin(pi*x)-40},{0,2,2*pi}
% 150,1000, [1,1,1,1],2,2);
```

```
syms x

close all % Cierra todas las ventanas existentes.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construimos la función.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


m=length(Dcell); % Número de discontinuidades más los dos extremos a
y b.

a=Dcell{1}; % Extremo izquierdo.
b=Dcell{m}; % Extremo derecho.

xa=linspace(a,b,n); % Mallado usado para aproximar.
hxa=(b-a)/n;
rxa=0.1*hxa*rand(size(xa));
xa=xa+rxa;
xa(1)=a;
xa(n)=b;


h=diff(xa); % Vector de espaciados no uniformes.

xev=linspace(a,b,mev); % Mallado usado para evaluar y medir errores
% de aproximación.


y=[]; % Inicializamos el vector con los valores numéricos de la función.
raiz=[];

dizq=Dcell{1};

% Número de condiciones. Coincide con el orden de aproximación local.


N=sum(n_condi)+length(n_condi);

% Derivada de mayor orden considerada.
```

```
md=max(n_condi)+1;

H=zeros(md,n); % Matriz que contiene en cada fila los datos de
% la función y sus derivadas.

fevE=zeros(1,mev); % Vector que contiene los datos de la función en
% los nodos de evaluación.

fevD=zeros(1,mev); % Vector que contiene los datos de la reconstrucción
% en los nodos de evaluación.


% Evaluación de la función en los nodos de evaluación.

n_nue=0;

for i=2:m

    dder=Dcell{i};
    ind_aux=find( (dizq <=xev) & (xev< dder));
    x_aux=xev(ind_aux);
    n_ant=n_nue;
    n_nue=n_nue+length(x_aux);
    faux=fcell{i-1};
    y_aux=double(subs(faux,{x},x_aux));
    fevE(n_ant+1:n_nue)=y_aux;
    dizq=dder;

end
yb=subs(faux,{x},b);
fevE(mev)=yb;


% Evaluación de la función en los nodos de reconstrucción.


dizq=Dcell{1};
n_nue=0;

for i=2:m
```

```
    dfaux{i-1}{1}=fcell{i-1};
    dder=Dcell{i};
    ind_aux=find( (dizq <=xa) & (xa< dder));
    x_aux=xa(ind_aux);
    n_ant=n_nue;
    n_nue=n_nue+length(x_aux);
    for j=1:md
        faux=dfaux{i-1}{j};
        y_aux=subs(faux,{x},x_aux);
        H(j,n_ant+1:n_nue)=y_aux;
        dfaux{i-1}{j+1}=diff(faux);
    end

    dizq=dder;

end


for j=1:md
    faux=dfaux{i-1}{j};
    yb=subs(faux,{x},b);
    H(j,n)=yb;
end


% Dibujamos la función y las derivadas.



for j=1:md
    figure
    minH=min(H(j,:)); maxH=max(H(j,:));
    plot(xa,H(j,:),'-')
    axis([xa(1)-0.1*(xa(end)-xa(1)) xa(end)+0.1*(xa(end)-xa(1)) minH-0.1*
(maxH-minH) maxH+0.1*(maxH-minH)])
    str=['Función después de derivar',blanks(1), num2str(j-1),blanks(1),'veces
la función original'];
    title(str);
end
```

```
minH=min(H(1,:)); maxH=max(H(1,:));

% Detectamos dónde están las discontinuidades.

y=H(1,:);

% Primero marcamos los intervalos sospechosos.

% Calculamos las diferencias divididas de segundo orden.


for i=2:n-1
    d2(i-1)=1/(h(i-1)*(h(i-1)+h(i)))*y(i-1)-1/(h(i-1)*h(i))*y(i)+1/(h(i)*(h(i-
1)+h(i)))*y(i+1);
end
candidatos=BB(d2,N-1);


% Establecemos las condiciones del artículo de Rosa Donat.

i=0;
for j=1:length(d2)

    if candidatos(j)==1
        i=i+1;
        nodosI=xa(j-N+2:j+1);
        yI=y(j-N+2:j+1);
        nodosD=xa(j+2:j+N+1);
        yD=y(j+2:j+N+1);
        indice_raiz(i)=j+1;
        xa(indice_raiz(i)-1:indice_raiz(i)+1)
        [raiz(i)]=Mu(nodosI,yI,nodosD,yD);
        candidatos(j+1)=0;
    elseif candidatos(j)==2
        i=i+1;
        nodosI=xa(j-N+2:j+1);
        yI=y(j-N+2:j+1);
        nodosD=xa(j+2:j+N+1);
        yD=y(j+2:j+N+1);
        indice_raiz(i)=j+1;
        xa(indice_raiz(i)-1:indice_raiz(i)+1)
```

```
        [raiz(i)]=Mu(nodosI,yI,nodosD,yD);
        candidatos(j+1)=0;
    end

end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aproximación de la función por trozos entre discontinuidades si las hay.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if isempty(raiz)

    % Inicializamos el vector que contendrá los valores de la función
    % reconstruida para ir completándolos.

    fevD=[];

    % Número de raíces intermedias encontradas.

    nr=length(raiz);


    % Partimos nuestro intervalo en trozos en cada raíz.

    % Primer trozo: desde el inicio a la primera raíz.

    % Cálculo del valor de la función y las derivadas en la primera raíz por
    % la izquierda.

    dfizq=get_derivatives(xa(indice_raiz(1)-N+1:indice_raiz(1)),
H(:,indice_raiz(1)-N+1:indice_raiz(1)),raiz(1));
    dfizq=dfizq(1:md)';

    xa_aux=[xa(1:indice_raiz(1)),raiz(1)];
    H_aux=[H(:,1:indice_raiz(1)),dfizq];
    ind_xev_aux=find(xev<=raiz(i));
    xev_aux=xev(ind_xev_aux);
    [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
```

```
    fevD=[fevD,fevD_aux];


    % Trozos centrales entre raíces.

    for i=1:nr-1

        % Cálculo del valor de la función y las derivadas en la raíz i por la
        % derecha y de la raíz i+1 por la izquierda.

        dfder=get_derivatives(xa(indice_raiz(i)+1:indice_raiz(i)+N),
H(:,indice_raiz(i)+1:indice_raiz(i)+N),raiz(i));
        dfder=dfder(1:md)';

        dfizq=get_derivatives(xa(indice_raiz(i+1)-N+1:indice_raiz(i+1)),
H(:,indice_raiz(i+1)-N+1:indice_raiz(i+1)),raiz(i+1));
        dfizq=dfizq(1:md)';


        xa_aux=[raiz(i),xa(indice_raiz(i)+1:indice_raiz(i+1)),raiz(i+1)];
        H_aux=[dfder,H(:,indice_raiz(i)+1:indice_raiz(i+1)),dfizq];

        ind_xev_aux=find((xev>raiz(i)) & (xev <= raiz(i+1)));
        xev_aux=xev(ind_xev_aux);

        [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
        fevD=[fevD,fevD_aux];


    end

    % Último trozo: entre la última raíz y el extremo b.

    % Cálculo del valor de la función y las derivadas en la última raíz por la
    % derecha.

    dfder=get_derivatives(xa(indice_raiz(nr)+1:indice_raiz(nr)+N),
H(:,indice_raiz(nr)+1:indice_raiz(nr)+N),raiz(nr));
    dfder=dfder(1:md)';
```

```
    xa_aux=[raiz(nr),xa(indice_raiz(nr)+1:end)];
    H_aux=[dfder,H(:,indice_raiz(nr)+1:end)];
    ind_xev_aux=find(xev>raiz(nr));
    xev_aux=xev(ind_xev_aux);

    [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
    fevD=[fevD,fevD_aux];



else % Hace Hermite.

    [fevD]=Hermite_segmentario(xa,H,n_condi,l,r,xev);



end


% Dibujamos la función y la reconstrucción obtenida.

figure
plot(xev,fevD,'r');
hold on
plot(xev,fevE,'b');
axis([xa(1)-0.1*(xa(end)-xa(1)) xa(end)+0.1*(xa(end)-xa(1)) minH-0.1*
(maxH-minH) maxH+0.1*(maxH-minH)]);
title('Reconstrucción por Hermite a trozos');


% Imprimimos la norma del error.


er=max(abs(fevE(1:end)-fevD(1:end)));



end
```

Capítulo 3

Algoritmos codificados en Matlab.

A continuación se presentan una generalización todos los códigos de los programas utilizados. Estos algoritmos han sido codificados en Matlab, y se han implementado de la manera más general posible.

3.1. Algoritmo de los polinomios de Taylor

```
function [C,Ye]=polytaylor(x0,Y,a,b,varargin)

% Construcción del polinomio interpolador de Taylor de una función f
% cerca de un punto x0, que pertenece al intervalo [a b].
%
%
% [C,Ye]=polytaylor(x0,f,n,varargin);
%
%
% Variables de entrada:
%
% x0 es el punto cerca del cual queremos aproximar la función.
% Y es el vector que contiene los valores de las derivadas sucesivas de la
% función en el punto x0.
% a extremo izquierdo del intervalo de aproximación.
% b extremo derecho del intervalo de aproximación.
% xe es el punto en el que evaluamos el polinomio.
%
%
```

```
% Variables de salida:
%
% C es el poliniomio interpolador de la función.
% Ye es el valor de polinomio en el punto xe.
%
%
% Ejemplo:
%
% f=cos(x).
% Sabiendo que f(0.5)=0.8776; f'(0.5)=-0.4794; f"(0.5)=-0.8776.
% y f"(0.5)=0.4794.
% [C,Ye]=polytaylor(0.5,[0.8776 -0.4794 -0.8776 0.4794],0,2,1.5)


if nargin==6
    xe=varargin{1};
else
    xe=[ ];
end
n=length(Y);
fac=1;
syms x
C=0;
for k=1:n
    fac=fac*k;
    t=((x-x0)^(k))/fac;
    C=C+Y(k)*t;
end
if xe =0
    Ye=subs(C,xe);
else
    Ye=";
end
C=char(C);

% Dibujo del polinomio de Taylor y de los puntos iniciales.

% Polinomio.

dx=(b-a)/10;
A=a-dx;
```

```
B=b+dx;
h1=ezplot(C,[A,B]);

% Puntos de resolución.

if xe =isempty(xe)
    h3=plot(xe,Ye,'g*','MarKerSize',5);
    legend([h1,h3],'Polinomio de Taylor','Evaluación en las Abscisas');
else
    legend([h1],'Polinomio de Taylor');
end
```

3.2. Algoritmo de interpolación de Lagrange

```
 function y=lagrange(f,nod,x)

% Este programa calcula la interpolacion de lagrange con n puntos.
%
%
% y=lagrange(f,nod,x);
%
%
% f valores de la funcion en los nodos.
% nod nodos para interpolar.
% x vector de abcisas donde evaluar el polinomio interpolador.

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
```

3.3. Algoritmo de interpolación de Lagrange: forma de Newton

```
function [C,N,Ye]=Newton(X,Y,varargin)

% Construcción del polinomio interpolador de Newton que pasa por los n+1
% puntos (xk,yk) para k=0,1,...,n.
%
%
% [C,N,Ye]=Newton(X,Y,Xe);
%
%
% Variables de entrada:
%
% X es un vector que contiene la lista de las abscisas.
% Y es un vector que contiene la lista de las ordenadas.
% Xe vector de abscisas donde se quiere evaluar el polinomio.
%
%
% Variables de salida:
%
% C es el vector que contiene los coeficientes del polinomio interpolador
% de Newton.
% N es la matriz que contiene los coeficientes de los polinomios
% base para construir el polinomio de Newton.
% Ye evaluación del polinomio de Newton en las abscisas Xe.
%
%
% Ejemplo:
%
%[C,N,Ye]=Newton([0 1 2 3 4],[1 0.5403023 -0.4161468 -0.98999925 -0.6536436])


if nargin==3
    Xe=varargin{1};
else
    Xe='';
    Ye='';
end
```

```
w=length(X);
N=zeros(w,w);

% Cálculo de las diferencias divididas.

D=diferencias_divididas(X,Y);


% Formación de los polinomios coeficientes de Newton.

for k=1:w

    V=1;
    for j=1:k-1
        V=conv(V,poly(X(j)));
    end

    N(k,w-length(V)+1:w)=V;

end

% Cálculo de los coeficientes del polinomio interpolador de Lagrange.

C=D*N;


% Evaluación en Xe.

if  isempty(Xe)
    Ye=polyval(C,Xe);
end


syms x
for i=0:w-1
    b(i+1)=x^(w-i-1);
end
Cx=dot(C,b);
```

```
% Dibujo del polinomio de Newton y de los puntos iniciales.

% Polinomio.

dx=(X(w)-X(1))/10;
A=X(1)-dx;
B=X(w)+dx;
h1=ezplot(Cx,[A,B]);

% Puntos iniciales.

hold on
h2=plot(X,Y,'ro','MarKerSize',10);

if  isempty(Xe)
    h3=plot(Xe,Ye,'g*','MarKerSize',5);
    legend([h1,h2,h3],'Polinomio de Newton','Nodos de interpolación','Evaluación
en las Abscisas');
else
    legend([h1,h2],'Polinomio de Newton','Nodos de interpolación');
end
```

3.4. Algoritmo de cálculo de diferencias divididas

```
function [fila1,tabla]=diferencias_divididas(X,Y)

% Este programa calcula las diferencias divididas.
%
%
% [fila1,tabla]=diferencias_divididas(X,Y);
%
%
% Variables de entrada:
%
% X son las abscisas.
% Y son las ordenadas.
%
%
% Variables de salida:
%
% fila1 contiene los coeficientes para formar el polinomio interpolador
% de Newton.
% tabla contiene el cuadro completo de las diferencias divididas.
%
%
% Ejemplo:
%
% Construir la tabla de diferencias divididas con los datos.
% x0=1; x1=2;
% f(x0)=1, f(x1)=-1;
% [fila1,tabla]=diferencias_divididas([1,2],[1,-1])


% Número de nodos.

n=length(X);


% Inicializamos la tabla a cero.

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

```
% Completamos las primeras dos columnas.

tabla(:,1)=X';
tabla(:,2)=Y';

% Completamos el resto de la tabla.

for i=3:n+1  % Columna.
    for j=1:n-i+2  % Fila.


        tabla(j,i)=(tabla(j+1,i-1)-tabla(j,i-1))/(tabla(j+i-2)-tabla(j,1));


    end
end

fila1=tabla(1,2:end);
```

3.5. Algoritmo de interpolación de Hermite

```
function [C,Ye]=Hermite(Xnodos,Ycondiciones,Xe)

% Esta función construye el polinomio interpolador de Hermite y lo
% evalua en ciertas abscisas Xe.
%
%
% [C,Ye]=Hermite(Xnodos,Ycondiciones,Xe);
%
%
% Variables de entrada:
%
% Xnodos son las abscisas donde se imponen las condiciones.
% Ycondiciones es una celda que contiene tantos vectores como nodos.
% Cada vector contiene los datos [f(xi),f'(xi),f"(xi)/2!,
% ...,f^(ni)(xi)/(ni)!].
% Xe valores de las abscisas donde se quiere evaluar el polinomio.
%
%
% Variables de salida:
%
% C coeficientes del polinomio de Hermite.
% Ye valores del polinomio en las abscisas Xe.
%
%
% Ejemplo:
%
% Polinomio que satisface las condiciones.
% x0=1; x1=2;
% p(x0)=1, p'(x0)=2 .
% p(x1)=-1, p'(x1)=3, p"(x1)=4.
% y su evaluación en x=1.5.
% [C,Ye]=Hermite([1,2],{[1,2],[-1,3,2]},1.5)


% Tabla de diferencias divididas generalizadas.

[D,tabla]=diferencias_divididas_generalizadas(Xnodos,Ycondiciones);
```

```
n=length(Xnodos);
Nodos=tabla(:,1);
w=length(Nodos);
N=zeros(w,w);

% Formación de los polinomios coeficientes de Hermite.

for k=1:w

    V=1;
    for j=1:k-1
        V=conv(V,poly(Nodos(j)));
    end

    N(k,w-length(V)+1:w)=V;

end


% Cálculo de los coeficientes del polinomio interpolador de Hermite.

C=D(2:end)*N;


% Evaluación en Xe.

if  isempty(Xe)
    Ye=polyval(C,Xe);
else
    Ye=[];
end
```

3.6. Algoritmo de cálculo de diferencias divididas generalizadas

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

% Este programa calcula las diferencias divididas generalizadas.
%
%
% [fila1,tabla,nc]=diferencias_divididas_generalizadas(nodos,condiciones);
%
%
% Variables de entrada:
%
% Nodos son las abscisas donde se imponen las condiciones.
% Condiciones es una celda que contiene tantos vectores como nodos.
% Cada vector contiene los datos [f(xi),f'(xi),f"(xi)/2!,
% ...,f^(ni)(xi)/(ni)!].
%
%
% Variables de salida:
%
% Fila1 contiene los coeficientes para formar el polinomio interpolador
% de Hermite.
% Tabla contiene el cuadro completo de las diferencias divididas
% nc número de condiciones que hay en cada nodo.
%
%
% Ejemplo:
%
% Construir la tabla de diferencias divididas generalizadas con los datos:
% x0=1; x1=2.
% f(x0)=1, f'(x0)=2.
% f(x1)=-1, f'(x1)=3, f"(x1)=4.
% [fila1,tabla,nc]=diferencias_divididas_generalizadas([1,2],{[1,2],[-1,3,2]})


% Número de nodos.

n=length(nodos);
```

```
% Número de condiciones en cada nodo.

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


% Inicializamos la tabla a cero.

tabla=zeros(nc_total,nc_total+1);


% Completamos las primeras dos columnas.

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

% Completamos el resto de la tabla.

for i=3:nc_total+1  % Columna.
    for j=1:nc_total-i+2  % Fila.

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

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

        end

    end
```

```
end

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

3.7. Algoritmo de detección de los posibles candidatos a contener una discontinuidad

```
function candidatos=BB(d2,N)

% Este programa marca la posición de los posibles candidatos a contener
% una discontinuidad.
%
%
% candidatos=BB(d2,N);
%
%
% Variables de entrada:
%
% d2 vector que contiene las diferencias segundas de los datos.
% N indica el orden N+1 de aproximación requerido.
%
%
% Variables de salida:
%
% Candidatos vector que contiene en cada posición:
% 0 si ese intervalo no es candidato.
% 1 si ese intervalo contiene una
% discontinuidad justo en el nodo inicial.
% 2 si ese intervalor contiene una
% discontinuidad intermedia.



n=length(d2);
candidatos=zeros(size(d2));
peso=1.5;


for j=N+2:n-N-1

    M=max(abs([d2(j-N-1:j-1),d2(j+1:j+N+1)]));
```

```
    if abs(d2(j))>peso*M
        candidatos(j)=1;
    end

    Mi=max(abs(d2(j-N-1:j-1)));

    Md=max(abs(d2(j+2:j+N+1)));

    if abs(d2(j+1))>peso*Md & abs(d2(j))>peso*Mi
        candidatos(j)=2;
    end

end
```

3.8. Algoritmo que aproxima la raíz de la función diferencia de los dos polinomios de Lagrange construidos a ambos lados de la discontinuidad

```
function [raiz]=Mu(nodosI,yI,nodosD,yD)

% Esta función aproxima la raíz de la función diferencia de los
% dos polinomios de Lagrange construidos a partir de los datos
% (nodosI,yI) y (nodosD,yD) respectivamente.
%
%
% [raiz]=Mu(nodosI,yI,nodosD,yD);
%
%
% Variables de entrada:
%
% nodosI valores de las abscisas para construir el polinomio de Lagrange
% de orden N de la izquierda.
% yI valores de la función en los nodosI.
% nodosD valores de las abscisas para construir el polinomio de Lagrange
% de orden N de la derecha.
% yD valores de la función en los nodosD.
%
%
% Variables de salida:
%
% raiz aproximación obtenida de la raíz por el método de Bisección.



tol=10^(-15);
tolf=10^(-20);


%Condiciones de convergencia.
%1.f(a)*f(b)<0.

n=length(nodosI);
```

```
a=nodosI(n);
b=nodosD(1);

pa=yI(n);
qb=yD(1);




fa=pa-lagrange(yD,nodosD,a);
fb=lagrange(yI,nodosI,b)-qb;




if abs(fa)<tolf
    raiz=a;
    return;
end

if abs(fb)<tolf
    raiz=b;
    return;
end

if fa*fb>0
    error ('La raiz no se encuentra en el intervalo(a,b)')
end



%Comenzamos las iteraciones.

n=ceil(log(abs(b-a)/tol)/log(2.0));
err=(b-a)/(2^(n));
```

```
for i=1:n

    c=0.5*(a+b);
    fc=lagrange(yI,nodosI,c)-lagrange(yD,nodosD,c);


    if abs(fc)<tolf
        raiz=c;
        return
    else
        if fb*fc<0
            a=c;
            fa=fc;
        else
            b=c;
            fb=fc;
        end
    end

end

raiz=(a+b)/2;
```

3.9. Algoritmo que obtiene las derivadas por la izquierda o por la derecha en la aproximación

```
function df=get_derivatives(x,fx,d)

% Esta función obtiene las derivadas por la izquierda o por la derecha en la
% aproximación calculada a la discontinuidad.
%
%
% df=get_derivatives(x,fx,d);
%
%
% Variables de entrada:
%
% x abscisas donde se plantean los desarrollos de Taylor.
% [x_j,x_{j-1}, ...,x_{j-N-1}] si es por el lado izquierdo.
% [x_{j+1},x_{j+2},...,x_{j+N}] si es por el lado derecho.
% fx valores conocidos de la función en las abscisas x.
% d aproximación calculada a la discontinuidad.
%
%
% Variables de salida:
%
% df aproximación de orden N a las derivadas por un lado en la
% discontinuidad.



% Orden de aproximación.

N=length(x);

% x-x*.

c=x-d;

% Matriz de Vandermonde.
```

```
A=fliplr(vander(c));

for i=2:N
    A(:,i)=A(:,i)/factorial(i-1);
end

% Vector de términos independientes.

b=fx';


% Resolvemos el sistema lineal.

df=A\b;
```

3.10. Algoritmo de Hermite segmentario

```
function [fevD]=Hermite_segmentario(xa,H,n_condi,l,r,xev)


% Esta función implementa el método de interpolación de Hermite
% segmentario.
%
%
% [fevD]=Hermite_segmentario(xa,H,n_condi,l,r,xev)
%
%
% Variables de entrada:
%
% xa abscisas x usadas para realizar la reconstrucción.
% H matriz que contiene en cada fila los valores de las derivadas sucesivas
% de una hipotética función en las abscisas xa.
% n_condi vector que indica el número de derivadas en cada punto del
% conjunto de puntos usado para interpolar localmente.
% l número de puntos a la izquierda de x_j incluyendo x_j.
% r número de puntos a la derecha de x_{j+1} incluyendo x_{j+1}.
% xev valores de las abscisas x donde calculamos la reconstrucción.
%
%
% Variables de salida:
%
% fevD evaluación de la reconstrucción en las abscisas xev.
%
%
% Ejemplo:
%
% syms x
% xa=0:1/100:1;
% xev=0:1/1000:1;
% f=3*sin(x+7);
% df=diff(f,x);
% H(1,:)=subs(f,{x},xa);
% H(2,:)=subs(df,{x},xa);
% [fevD]=Hermite_segmentario(xa,H,[1,1,1,1],2,2,xev);
```

```
% Número de nodos de interpolación.

n=length(xa);

% Número de puntos en cada stencil local.

nd=l+r;


% Realizamos la reconstrucción en los primeros l-1 intervalos.


% Realizamos el primer intervalo separadamente.

fevD=[];

if l>1

    i=1;
    n_condi_aux=[n_condi(l-i+1:nd),n_condi(1:l-i)];
    nodos_aux=xa(1:nd);
    H_aux=H(:,1:nd);

    for in=1:nd
        condiciones{in}=[];
        for j=1:n_condi_aux(in)
            condiciones{in}=[condiciones{in},H_aux(j,in)/factorial(j-1)];
        end
    end


    ind_xev_aux=find((xev>=nodos_aux(i)) & (xev <= nodos_aux(i+1)));
    xev_aux=xev(ind_xev_aux);
    [p_aux,fevD_aux]=Hermite(nodos_aux,condiciones,xev_aux);

    fevD=[fevD,fevD_aux];



    % Ahora desde el segundo intervalo al l-1.
```

```
for i=2:l-1
    n_condi_aux=[n_condi(l-i+1:nd),n_condi(1:l-i)];
    nodos_aux=xa(1:nd);
    H_aux=H(:,1:nd);

    for in=1:nd
        condiciones{in}=[];
        for j=1:n_condi_aux(in)
            condiciones{in}=[condiciones{in},H_aux(j,in)/factorial(j-1)];
        end
    end

    ind_xev_aux=find((xev>nodos_aux(i)) & (xev <= nodos_aux(i+1)));
    xev_aux=xev(ind_xev_aux);
    [p_aux,fevD_aux]=Hermite(nodos_aux,condiciones,xev_aux);

    fevD=[fevD,fevD_aux];




end



% Realizamos la reconstrucción en los intervalos desde el l hasta el
% n-r.

for i=l:n-r

    nodos_aux=xa(i-l+1:i+r);
    H_aux=H(:,i-l+1:i+r);

    for in=1:nd
        condiciones{in}=[];
        for j=1:n_condi(in)
            condiciones{in}=[condiciones{in},H_aux(j,in)/factorial(j-1)];
        end
    end
```

```
        ind_xev_aux=find((xev>nodos_aux(l)) & (xev <= nodos_aux(l+1)));
        xev_aux=xev(ind_xev_aux);
        [p_aux,fevD_aux]=Hermite(nodos_aux,condiciones,xev_aux);

        fevD=[fevD,fevD_aux];



    end


else

    % Realizamos la reconstrucción en los intervalos desde el l hasta el
    % n-r.

    i=l;
    nodos_aux=xa(i-l+1:i+r);
    H_aux=H(:,i-l+1:i+r);

    for in=1:nd
        condiciones{in}=[];
        for j=1:n_condi(in)
            condiciones{in}=[condiciones{in},H_aux(j,in)/factorial(j-1)];
        end
    end

    ind_xev_aux=find((xev>=nodos_aux(l)) & (xev <= nodos_aux(l+1)));
    xev_aux=xev(ind_xev_aux);
    [p_aux,fevD_aux]=Hermite(nodos_aux,condiciones,xev_aux);

    fevD=[fevD,fevD_aux];



    for i=l+1:n-r

        nodos_aux=xa(i-l+1:i+r);
        H_aux=H(:,i-l+1:i+r);

        for in=1:nd
```

```
            condiciones{in}=[];
            for j=1:n_condi(in)
                condiciones{in}=[condiciones{in},H_aux(j,in)/factorial(j-1)];
            end
        end

        ind_xev_aux=find((xev>nodos_aux(l)) & (xev <= nodos_aux(l+1)));
        xev_aux=xev(ind_xev_aux);
        [p_aux,fevD_aux]=Hermite(nodos_aux,condiciones,xev_aux);

        fevD=[fevD,fevD_aux];



    end



end



% Realizamos la reconstrucción en los últimos r-1 intervalos.


for i=1:r-1
    n_condi_aux=[n_condi(nd-i+1:nd),n_condi(1:nd-i)];
    nodos_aux=xa(n-nd+1:n);
    H_aux=H(:,n-nd+1:n);

    for in=1:nd
        condiciones{in}=[];
        for j=1:n_condi_aux(in)
            condiciones{in}=[condiciones{in},H_aux(j,in)/factorial(j-1)];
        end
    end

    ind_xev_aux=find((xev>nodos_aux(l+i)) & (xev <= nodos_aux(l+i+1)));
    xev_aux=xev(ind_xev_aux);
    [p_aux,fevD_aux]=Hermite(nodos_aux,condiciones,xev_aux);
```

```
    fevD=[fevD,fevD_aux];



end
```

3.11. Algoritmo que calcula la reconstrucción de Hermite a trozos de una función discontinua dada

```
    function er=Hermite_D(fcell,Dcell,n,mev,n_condi,l,r)


% Esta función calcula la reconstrucción de Hermite a trozos de una función
% discontinua dada, partiendo justo en las discontinuidades para no
% cruzarlas.
%
%
% er=Hermite_D(fcell,Dcell,n,mev,n_condi,l,r);
%
%
% Variables de entrada:
%
% fcell variable de celda conteniendo las expresiones de las funciones
% entre discontinuidades.
% Dcell variable de celda conteniendo las posiciones de las
% discontinuidades. Dcell empieza por el extremo inicial a y
% acaba en el extremo final b.
% n número de puntos en que discretizamos el intervalo [a,b].
% mev número de puntos del mallado fino (m >> n) donde discretizamos
```

```
% el intervalo [a,b].
% n_condi vector que indica el número de derivadas en cada punto del
% conjunto de puntos usado para interpolar localmente.
% l número de puntos a la izquierda de x_j incluyendo x_j.
% r número de puntos a la derecha de x_{j+1} incluyendo x_{j+1}.
%
%
% Variables de salida:
%
% er error cometido entre la reconstrucción construida y la función
% original cuando evaluamos ambas en un mallado fino.
%
%
% Ejemplo 1:
% er=Hermite_D({10*sin(pi*x),-50*cos(pi*x)+10,20*sin(pi*x)-40,-40+
% 5*sin(pi*x)},{0,0.5,2,3,2*pi},150,1000,[1,1,1,1],2,2);
% Ejemplo 2:
% er=Hermite_D({-50*cos(pi*x)+10,20*sin(pi*x)-40},{0,2,2*pi},150,1000,
%,[1,1,1,1],2,2);

syms x

close all % Cierra todas las ventanas existentes.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construimos la función.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


m=length(Dcell); % Número de discontinuidades más los dos extremos a
y b.

a=Dcell{1}; % Extremo izquierdo.
b=Dcell{m}; % Extremo derecho.
h=(b-a)/n; % Espaciado entre nodos de reconstrucción.

xa=linspace(a,b,n); % Mallado usado para aproximar.

xev=linspace(a,b,mev); % Mallado usado para evaluar y medir errores de

% aproximación.
```

```
y=[]; % Inicializamos el vector con los valores numéricos de la función.
raiz=[];

dizq=Dcell{1};

% Número de condiciones. Coincide con el orden de aproximación local.


N=sum(n_condi)+length(n_condi);

% Derivada de mayor orden considerada.

md=max(n_condi)+1;

H=zeros(md,n); % Matriz que contiene en cada fila los datos de
% la función y sus derivadas.

fevE=zeros(1,mev); % Vector que contiene los datos de la función
% en los nodos de evaluación.

fevD=zeros(1,mev); % Vector que contiene los datos de la
% reconstrucción en los nodos de evaluación.



% Evaluación de la función en los nodos de evaluación.

n_nue=0;

for i=2:m

    dder=Dcell{i};
    ind_aux=find( (dizq <=xev) & (xev< dder));
    x_aux=xev(ind_aux);
    n_ant=n_nue;
    n_nue=n_nue+length(x_aux);
    faux=fcell{i-1};
    y_aux=double(subs(faux,{x},x_aux));
    fevE(n_ant+1:n_nue)=y_aux;
```

```
    dizq=dder;

end
yb=subs(faux,{x},b);
fevE(mev)=yb;


% Evaluación de la función en los nodos de reconstrucción.

dizq=Dcell{1};
n_nue=0;

for i=2:m

    dfaux{i-1}{1}=fcell{i-1};
    dder=Dcell{i};
    ind_aux=find( (dizq <=xa) & (xa< dder));
    x_aux=xa(ind_aux);
    n_ant=n_nue;
    n_nue=n_nue+length(x_aux);
    for j=1:md
        faux=dfaux{i-1}{j};
        y_aux=subs(faux,{x},x_aux);
        H(j,n_ant+1:n_nue)=y_aux;
        dfaux{i-1}{j+1}=diff(faux);
    end

    dizq=dder;

end


for j=1:md
    faux=dfaux{i-1}{j};
    yb=subs(faux,{x},b);
    H(j,n)=yb;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dibujamos la función y las derivadas.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j=1:md
   figure
   minH=min(H(j,:)); maxH=max(H(j,:));
   plot(xa,H(j,:),'-')
   axis([xa(1)-0.1*(xa(end)-xa(1)) xa(end)+0.1*(xa(end)-xa(1)) minH-0.1*
(maxH-minH) maxH+0.1*(maxH-minH)])
        str=['Función después de derivar',blanks(1), num2str(j-1),blanks(1),'veces
la función original'];
   title(str);
end

minH=min(H(1,:)); maxH=max(H(1,:));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Detectamos dónde están las discontinuidades.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y=H(1,:);

% Primero marcamos los intervalos sospechosos.

% calculamos las diferencias finitas de segundo orden.

d2=diff(y,2);
candidatos=BB(d2,N-1);

% Establecemos las condiciones del artículo de Rosa Donat.

i=0;
for j=1:length(d2)

   if candidatos(j)==1
      i=i+1;
      nodosI=xa(j-N+2:j+1);
```

```
        yI=y(j-N+2:j+1);
        nodosD=xa(j+2:j+N+1);
        yD=y(j+2:j+N+1);
        indice_raiz(i)=j+1;
        xa(indice_raiz(i)-1:indice_raiz(i)+1)
        [raiz(i)]=Mu(nodosI,yI,nodosD,yD);
        candidatos(j+1)=0;
    elseif candidatos(j)==2
        i=i+1;
        nodosI=xa(j-N+2:j+1);
        yI=y(j-N+2:j+1);
        nodosD=xa(j+2:j+N+1);
        yD=y(j+2:j+N+1);
        indice_raiz(i)=j+1;
        xa(indice_raiz(i)-1:indice_raiz(i)+1)
        [raiz(i)]=Mu(nodosI,yI,nodosD,yD);
        candidatos(j+1)=0;
    end

end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aproximación de la función por trozos entre discontinuidades si las hay.
%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if  isempty(raiz)

    % Inicializamos el vector que contendrá los valores de la función
    % reconstruida para ir completándolos.

    fevD=[];

    % Número de raíces intermedias encontradas.

    nr=length(raiz);


    % Partimos nuestro intervalo en trozos en cada raíz.
```

```
    % Primer trozo: desde el inicio a la primera raíz.

    % Cálculo del valor de la función y las derivadas en la primera raíz por
    % la izquierda.

    dfizq=get_derivatives(xa(indice_raiz(1)-N+1:indice_raiz(1)),
H(:,indice_raiz(1)-N+1:indice_raiz(1)),raiz(1));
    dfizq=dfizq(1:md)';

    xa_aux=[xa(1:indice_raiz(1)),raiz(1)];
    H_aux=[H(:,1:indice_raiz(1)),dfizq];
    ind_xev_aux=find(xev<=raiz(i));
    xev_aux=xev(ind_xev_aux);
    [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);


    fevD=[fevD,fevD_aux];


    % Trozos centrales entre raíces.

    for i=1:nr-1

        % Cálculo del valor de la función y las derivadas en la raíz i por la
        % derecha y de la raíz i+1 por la izquierda.

        dfder=get_derivatives(xa(indice_raiz(i)+1:indice_raiz(i)+N)
,H(:,indice_raiz(i)+1:indice_raiz(i)+N),raiz(i));
        dfder=dfder(1:md)';

        dfizq=get_derivatives(xa(indice_raiz(i+1)-N+1:indice_raiz(i+1)),
H(:,indice_raiz(i+1)-N+1:indice_raiz(i+1)),raiz(i+1));
        dfizq=dfizq(1:md)';


        xa_aux=[raiz(i),xa(indice_raiz(i)+1:indice_raiz(i+1)),raiz(i+1)];
        H_aux=[dfder,H(:,indice_raiz(i)+1:indice_raiz(i+1)),dfizq];

        ind_xev_aux=find((xev>raiz(i)) & (xev <= raiz(i+1)));
        xev_aux=xev(ind_xev_aux);
```

```
        [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
        fevD=[fevD,fevD_aux];


    end

    % Último trozo: entre la última raíz y el extremo b.

    % Cálculo del valor de la función y las derivadas en la última raíz por la
    % derecha.

    dfder=get_derivatives(xa(indice_raiz(nr)+1:indice_raiz(nr)+N),
H(:,indice_raiz(nr)+1:indice_raiz(nr)+N),raiz(nr));
    dfder=dfder(1:md)';

    xa_aux=[raiz(nr),xa(indice_raiz(nr)+1:end)];
    H_aux=[dfder,H(:,indice_raiz(nr)+1:end)];
    ind_xev_aux=find(xev>raiz(nr));
    xev_aux=xev(ind_xev_aux);

    [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
    fevD=[fevD,fevD_aux];
    fevD=double(fevD);


else % Hace Hermite.

    [fevD]=Hermite_segmentario(xa,H,n_condi,l,r,xev);
    fevD=double(fevD);



end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dibujamos la función y la reconstrucción obtenida.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
figure
plot(xev,fevD,'r');
hold on
plot(xev,fevE,'b');
axis([xa(1)-0.1*(xa(end)-xa(1)) xa(end)+0.1*(xa(end)-xa(1)) minH-0.1*(maxH-minH) maxH+0.1*(maxH-minH)]);
title('Reconstrucción por Hermite a trozos');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Imprimimos la norma del error.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


   % for i=1:length(raiz)
   % h
   % N
   % h^(N-2)
   % raiz(i)-h^(N-3)
   % ind_aux=find((xev>=raiz(i)-h^(N-1)) & (xev<=raiz(i)+h^(N-1)))
   % figure
   % plot(xev(ind_aux),fevE(ind_aux),'b');
   % hold on
   % plot(xev(ind_aux),fevD(ind_aux),'r');

end


er=max(abs(fevE(1:end)-fevD(1:end)));



end
```

3.12. Algoritmo que calcula la reconstrucción de Hermite a trozos de una función discontinua dada en un mallado no uniforme

```
function er=Hermite_nu_D(fcell,Dcell,n,mev,n_condi,l,r)


% Esta función calcula la reconstrucción de Hermite a trozos de una función
% discontinua dada, partiendo justo en las discontinuidades para no
% cruzarlas.
%
%
% er=Hermite_nu_D(fcell,Dcell,n,mev,n_condi,l,r);
%
%
% Variables de entrada:
%
% fcell variable de celda conteniendo las expresiones de las funciones
% entre discontinuidades.
% Dcell variable de celda conteniendo las posiciones de las
% discontinuidades. Dcell empieza por el extremo inicial a y
% acaba en el extremo final b.
% n número de puntos en que discretizamos el intervalo [a,b].
% mev número de puntos del mallado fino (m >> n) donde discretizamos
% el intervalo [a,b].
% n_condi vector que indica el número de derivadas en cada punto del
% conjunto de puntos usado para interpolar localmente.
% l número de puntos a la izquierda de x_j incluyendo x_j.
% r número de puntos a la derecha de x_{j+1} incluyendo x_{j+1}.
%
%
% Ejemplo 1:
%
% er=Hermite_nu_D({10*sin(pi*x),-50*cos(pi*x)+10,20*sin(pi*x)-40,
% -40+5*sin(pi*x)},{0,0.5,2,3,2*pi},150,1000,[1,1,1,1],2,2);
%
%
% Ejemplo 2:
```

```
%
% er=Hermite_nu_D({-50*cos(pi*x)+10,20*sin(pi*x)-40},{0,2,2*pi}
% 150,1000, [1,1,1,1],2,2);

syms x

close all % Cierra todas las ventanas existentes.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construimos la función.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


m=length(Dcell); % Número de discontinuidades más los dos extremos a
y b.

a=Dcell{1}; % Extremo izquierdo.
b=Dcell{m}; % Extremo derecho.

xa=linspace(a,b,n); % Mallado usado para aproximar.
hxa=(b-a)/n;
rxa=0.1*hxa*rand(size(xa));
xa=xa+rxa;
xa(1)=a;
xa(n)=b;


h=diff(xa); % Vector de espaciados no uniformes.

xev=linspace(a,b,mev); % Mallado usado para evaluar y medir errores
% de aproximación.


y=[]; % Inicializamos el vector con los valores numéricos de la función.
raiz=[];

dizq=Dcell{1};

% Número de condiciones. Coincide con el orden de aproximación local.
```

```
N=sum(n_condi)+length(n_condi);

% Derivada de mayor orden considerada.

md=max(n_condi)+1;

H=zeros(md,n); % Matriz que contiene en cada fila los datos de
% la función y sus derivadas.

fevE=zeros(1,mev); % Vector que contiene los datos de la función en
% los nodos de evaluación.

fevD=zeros(1,mev); % Vector que contiene los datos de la reconstrucción
% en los nodos de evaluación.


% Evaluación de la función en los nodos de evaluación.

n_nue=0;

for i=2:m

    dder=Dcell{i};
    ind_aux=find( (dizq <=xev) & (xev< dder));
    x_aux=xev(ind_aux);
    n_ant=n_nue;
    n_nue=n_nue+length(x_aux);
    faux=fcell{i-1};
    y_aux=double(subs(faux,{x},x_aux));
    fevE(n_ant+1:n_nue)=y_aux;
    dizq=dder;

end
yb=subs(faux,{x},b);
fevE(mev)=yb;


% Evaluación de la función en los nodos de reconstrucción.


dizq=Dcell{1};
```

```
n_nue=0;

for i=2:m

    dfaux{i-1}{1}=fcell{i-1};
    dder=Dcell{i};
    ind_aux=find( (dizq <=xa) & (xa< dder));
    x_aux=xa(ind_aux);
    n_ant=n_nue;
    n_nue=n_nue+length(x_aux);
    for j=1:md
        faux=dfaux{i-1}{j};
        y_aux=subs(faux,{x},x_aux);
        H(j,n_ant+1:n_nue)=y_aux;
        dfaux{i-1}{j+1}=diff(faux);
    end

    dizq=dder;

end


for j=1:md
    faux=dfaux{i-1}{j};
    yb=subs(faux,{x},b);
    H(j,n)=yb;
end


% Dibujamos la función y las derivadas.



for j=1:md
    figure
    minH=min(H(j,:)); maxH=max(H(j,:));
    plot(xa,H(j,:),'-')
    axis([xa(1)-0.1*(xa(end)-xa(1)) xa(end)+0.1*(xa(end)-xa(1)) minH-0.1*
(maxH-minH) maxH+0.1*(maxH-minH)])
    str=['Función después de derivar',blanks(1), num2str(j-1),blanks(1),'veces
la función original'];
```

```
    title(str);
end

minH=min(H(1,:)); maxH=max(H(1,:));

% Detectamos dónde están las discontinuidades.

y=H(1,:);

% Primero marcamos los intervalos sospechosos.

% Calculamos las diferencias divididas de segundo orden.


for i=2:n-1
    d2(i-1)=1/(h(i-1)*(h(i-1)+h(i)))*y(i-1)-1/(h(i-1)*h(i))*y(i)+1/(h(i)*(h(i-
1)+h(i)))*y(i+1);
end
candidatos=BB(d2,N-1);


% Establecemos las condiciones del artículo de Rosa Donat.

i=0;
for j=1:length(d2)

    if candidatos(j)==1
        i=i+1;
        nodosI=xa(j-N+2:j+1);
        yI=y(j-N+2:j+1);
        nodosD=xa(j+2:j+N+1);
        yD=y(j+2:j+N+1);
        indice_raiz(i)=j+1;
        xa(indice_raiz(i)-1:indice_raiz(i)+1)
        [raiz(i)]=Mu(nodosI,yI,nodosD,yD);
        candidatos(j+1)=0;
    elseif candidatos(j)==2
        i=i+1;
        nodosI=xa(j-N+2:j+1);
        yI=y(j-N+2:j+1);
        nodosD=xa(j+2:j+N+1);
```

```
        yD=y(j+2:j+N+1);
        indice_raiz(i)=j+1;
        xa(indice_raiz(i)-1:indice_raiz(i)+1)
        [raiz(i)]=Mu(nodosI,yI,nodosD,yD);
        candidatos(j+1)=0;
    end

end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aproximación de la función por trozos entre discontinuidades si las hay.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if isempty(raiz)

    % Inicializamos el vector que contendrá los valores de la función
    % reconstruida para ir completándolos.

    fevD=[];

    % Número de raíces intermedias encontradas.

    nr=length(raiz);


    % Partimos nuestro intervalo en trozos en cada raíz.

    % Primer trozo: desde el inicio a la primera raíz.

    % Cálculo del valor de la función y las derivadas en la primera raíz por
    % la izquierda.

    dfizq=get_derivatives(xa(indice_raiz(1)-N+1:indice_raiz(1)),
H(:,indice_raiz(1)-N+1:indice_raiz(1)),raiz(1));
    dfizq=dfizq(1:md)';

    xa_aux=[xa(1:indice_raiz(1)),raiz(1)];
    H_aux=[H(:,1:indice_raiz(1)),dfizq];
```

```
    ind_xev_aux=find(xev<=raiz(i));
    xev_aux=xev(ind_xev_aux);
    [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);


    fevD=[fevD,fevD_aux];


    % Trozos centrales entre raíces.

    for i=1:nr-1

        % Cálculo del valor de la función y las derivadas en la raíz i por la
        % derecha y de la raíz i+1 por la izquierda.

        dfder=get_derivatives(xa(indice_raiz(i)+1:indice_raiz(i)+N),
H(:,indice_raiz(i)+1:indice_raiz(i)+N),raiz(i));
        dfder=dfder(1:md)';

        dfizq=get_derivatives(xa(indice_raiz(i+1)-N+1:indice_raiz(i+1)),
H(:,indice_raiz(i+1)-N+1:indice_raiz(i+1)),raiz(i+1));
        dfizq=dfizq(1:md)';


        xa_aux=[raiz(i),xa(indice_raiz(i)+1:indice_raiz(i+1)),raiz(i+1)];
        H_aux=[dfder,H(:,indice_raiz(i)+1:indice_raiz(i+1)),dfizq];

        ind_xev_aux=find((xev>raiz(i)) & (xev <= raiz(i+1)));
        xev_aux=xev(ind_xev_aux);

        [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
        fevD=[fevD,fevD_aux];


    end

    % Último trozo: entre la última raíz y el extremo b.

    % Cálculo del valor de la función y las derivadas en la última raíz por la
    % derecha.
```

```
    dfder=get_derivatives(xa(indice_raiz(nr)+1:indice_raiz(nr)+N),
H(:,indice_raiz(nr)+1:indice_raiz(nr)+N),raiz(nr));
    dfder=dfder(1:md)';

    xa_aux=[raiz(nr),xa(indice_raiz(nr)+1:end)];
    H_aux=[dfder,H(:,indice_raiz(nr)+1:end)];
    ind_xev_aux=find(xev>raiz(nr));
    xev_aux=xev(ind_xev_aux);

    [fevD_aux]=Hermite_segmentario(xa_aux,H_aux,n_condi,l,r,xev_aux);
    fevD=[fevD,fevD_aux];



else % Hace Hermite.

    [fevD]=Hermite_segmentario(xa,H,n_condi,l,r,xev);



end


% Dibujamos la función y la reconstrucción obtenida.

figure
plot(xev,fevD,'r');
hold on
plot(xev,fevE,'b');
axis([xa(1)-0.1*(xa(end)-xa(1)) xa(end)+0.1*(xa(end)-xa(1)) minH-0.1*
(maxH-minH) maxH+0.1*(maxH-minH)]);
title('Reconstrucción por Hermite a trozos');


% Imprimimos la norma del error.


er=max(abs(fevE(1:end)-fevD(1:end)));
```

end

Capítulo 4

Interfaz Gráfica.

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

4.1. Ejecución de la Interfaz Gráfica

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

"./TFG/Interfaz Gráfica"

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

```
>> inicio_Interfaz_Hermite
```

Aparece la pantalla de “Inicio” describiendo información general del proyecto (Figura 4.1).

Figura 4.1: Inicio interfaz.

Otra manera de ejecutar la pantalla de inicio (Figura 4.2) es introducir el siguiente comando y pulsar “run”:

```
>> guide inicio_Interfaz_Hermite
```

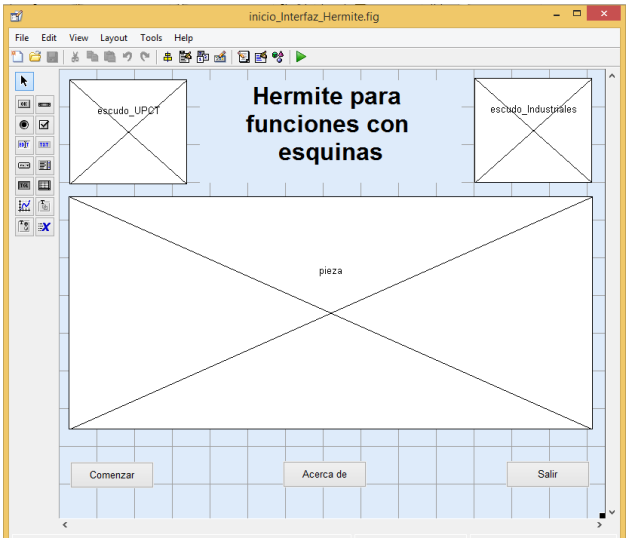

Figura 4.2: Guide inicio interfaz.

Para ejecutar la interfaz principal del programa, se puede ejecutar desde la pantalla de inicio de la interfaz pulsando el botón “Comenzar” o directamente escribiendo en la linea de comandos de Matlab el siguiente comando:

```
>> Interfaz_Hermite
```

Esta orden nos lleva directamente a la interfaz principal descrita en el siguiente apartado (Figura 4.3).

4.2. Descripción de la interfaz gráfica principal

A continuación vamos a describir la interfaz dedicada al método de Hermite objeto de estudio en este Trabajo Fin de Grado (Figura 4.3). La hemos dividido en dos partes: Menú ayuda y cuadro central.

Figura 4.3: Interfaz gráfica principal.

4.2.1. Menú de ayuda

La interfaz principal de usuario cuenta con un menú en la parte superior izquierda (Figura 4.4) que consiste en una opción que puede desplegarse para ver el contenido de la ayuda.

Figura 4.4: Menú de ayuda.

Vamos a describir dicho menú y las diferentes opciones.

- **Ayuda**
  Este menú contiene un tutorial del manejo de la interfaz gráfica e información general acerca del proyecto (Figura 4.5).

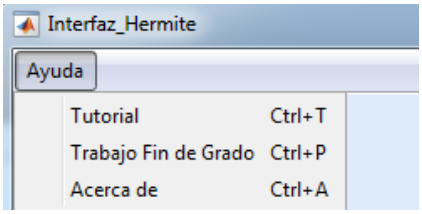

Figura 4.5: Submenús de ayuda.

  - **Tutorial**
    Seleccionando esta opción se abrirá un tutorial en formato pdf que contiene indicaciones sobre el manejo de la interfaz gráfica.
  - **Trabajo Fin de Grado**
    Seleccionando esta opción se podrá ver el TFG en formato pdf.
  - **Acerca de**
    Seleccionando esta opción recibiremos información acerca de la realización, autor y directores del proyecto.

4.2.2. Cuadro Central

Por "Cuadro Central" entendemos el resto de campos que forman la Interfaz Gráfica (Figura 4.3). Debemos configurar cada campo según los requerimientos. Iremos explicando campo a campo el significado de cada uno de ellos.

- **Algoritmo**
  Debemos empezar escogiendo el tipo de algoritmo con el que queramos trabajar, Hermite con detección previa de esquinas o Hermite. Esta opción permite comparar los resultados entre los dos algoritmos. También debemos introducir en el lado derecho el tipo de Hermite, es decir, si localmente estamos reconstruyendo en el intervalo $[x_j, x_{j+1}]$, debemos de indicar el número de puntos a la izquierda de x_j (incluyendo x_j), el número de puntos a la derecha de x_{j+1} (incluyendo x_{j+1}) y el vector de condiciones, que especifica cuantas derivadas se consideran en cada punto además del valor de la función. (Figura 4.6).

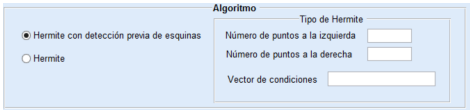

Figura 4.6: Elección de algoritmo.

- **Datos de entrada**
  En primer lugar debemos escoger el tipo de datos de entrada, tenemos dos opciones, datos que vienen de una función o datos que vienen un experimento (Figura 4.7).

  En la primera opción, "Datos que vienen de una fución", tenemos las entradas "Función", "Mallado" y "Abscisas de Evaluación" que bien pueden ser creadas por nosotros o cargadas.
  En el botón "crear" se abre el archivo crear_función.mat que sirve de plantilla e introducimos los datos tal y como viene indicado en los comentarios de ayuda que aparecen.
  En el botón "cargar" se abrirá una carpeta con archivos que hayamos guardado con anterioridad en un archivo de tipo .mat.

  Para la opción "Datos que vienen de un experimento" se procederá de manera similar al anterior, salvo que en este caso en vez de trabajar con la expresión de una función se trabajará con datos experimentales de inicio.

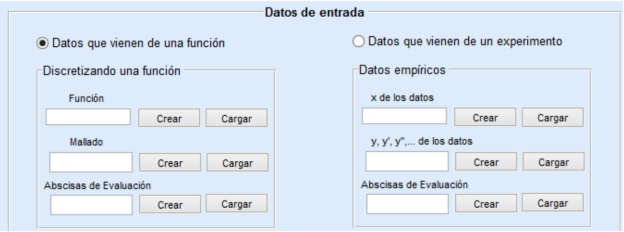

Figura 4.7: Datos de entrada.

- **Condiciones de contorno**
  En este caso, los marcadores siempre estarán activados, es una parte informativa (Figura 4.8). Se indica que en ambas fronteras se ha optado por tomar hacia el interior del intervalo los datos que dejan de estar disponibles.

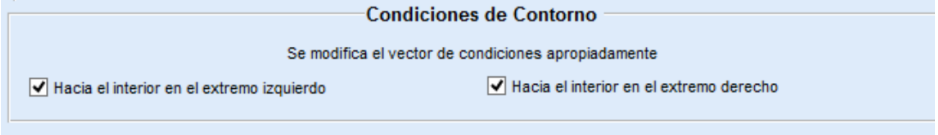

Figura 4.8: Condiciones de contorno.

- **Botones de acción**
  En esta parte de la interfaz gráfica principal tenemos una serie de pulsadores (Figura 4.9):

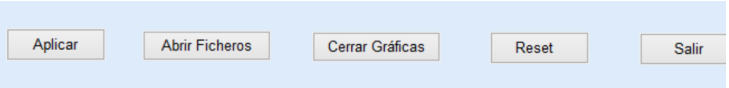

Figura 4.9: Botones de acción.

"Aplicar", pulsando este botón se recogerán todos los datos introducidos en la interfaz, y se llamará al algoritmo seleccionado para que se ejecute y devuelva las salidas previstas: pantallas emergentes con gráficas, errores, e informaciones del algoritmo, así como ficheros de datos contenidos en la carpeta llamada FicherosTxT.
"Abrir Ficheros", abre los archivos que contienen informaciones de salida del algoritmo seleccionado. Dichos archivos se encuentran en la carpeta FicherosTxT. Entre ellos encontramos polinomios_Hermite y

resul_valores_interpolados.
"Cerrar Gráficas", se cerrarán todas las gráficas que ha creado el programa en el cálculo realizado.
"Reset", se borrarán todos los valores que se han introducido en la interfaz y la dejará como al inicio para realizar nuevos cálculos.
"Salir", preguntará si deseamos salir del programa (Figura 4.10).

Figura 4.10: Salir.

4.3. Ejemplo de uso de la Interfaz Gráfica

A continuación se explica un ejemplo práctico del uso de la interfaz. Vamos completando los campos secuencialmente como se ha descrito en el apartado anterior. De esta manera, una de las posibles configuraciones podría ser la que se contempla en la Figura 4.11.

Figura 4.11: Ejemplo de configuración de la interfaz.

Una vez rellenados todos los campos, le damos al botón APLICAR y obtenemos las siguientes pantallas: el tiempo que tarda la CPU en realizar los cálculos (Figura 4.12), la localización de las esquinas (Figura 4.13), (Figura 4.14), (Figura 4.15), la función después de derivar 0 veces la función original (Figura 4.16), la función después de derivar 1 vez la función original (Figura 4.17), la reconstrucción de Hermite a trozos (Figura 4.18) y el error en norma infinito (Figura 4.19).

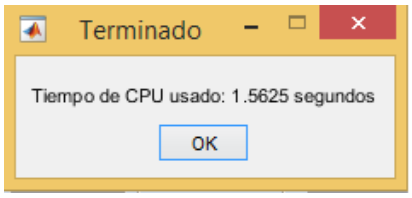

Figura 4.12: Tiempo de CPU.

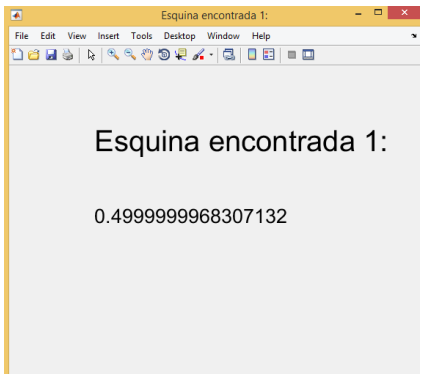

Figura 4.13: Esquina encontrada 1.

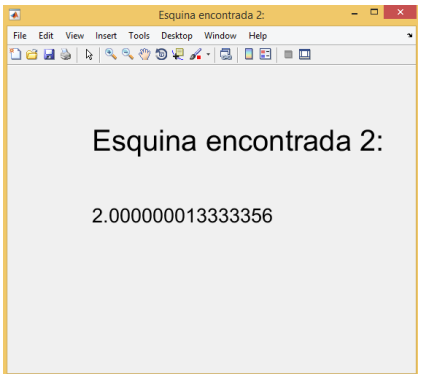

Figura 4.14: Esquina encontrada 2.

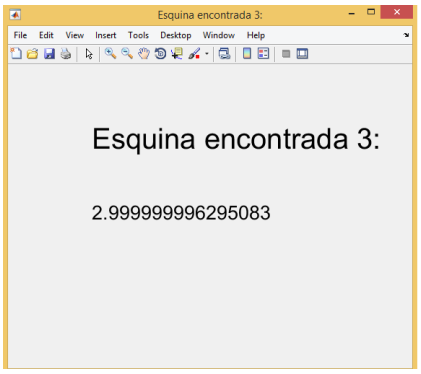

Figura 4.15: Esquina encontrada 3.

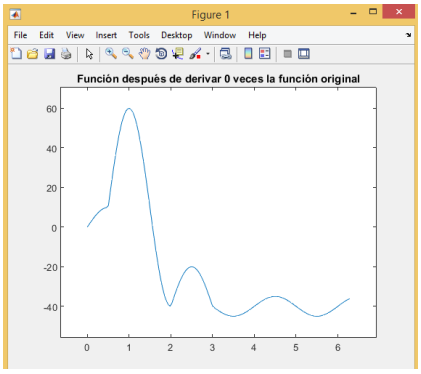

Figura 4.16: Función después de derivar 0 veces la función original.

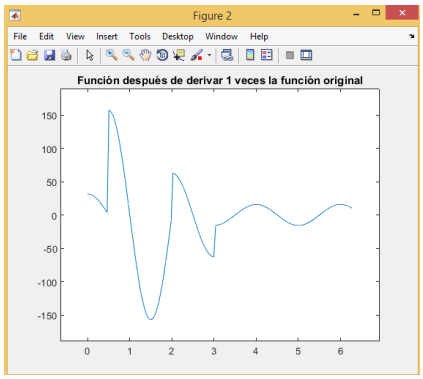

Figura 4.17: Función después de derivar 1 vez la función original.

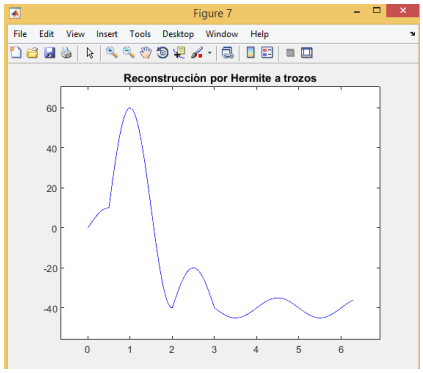

Figura 4.18: Reconstrucción por Hermite a trozos.

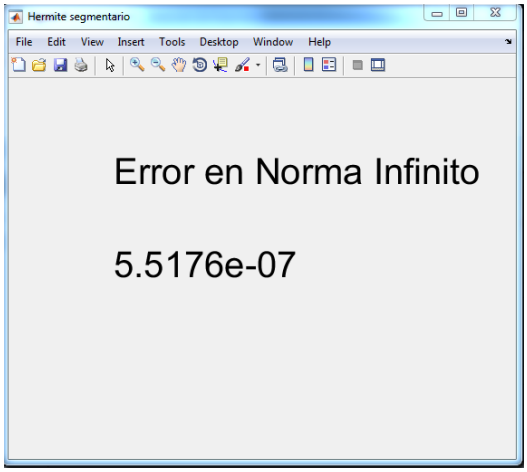

Figura 4.19: Error en norma infinito.

Por último, presionando el botón ABRIR FICHEROS, obtenemos dos ficheros. El primer fichero se ha llamado polinomios_Hermite_trozo_n (Figura 4.20), y contiene por filas los coeficientes de cada polinomio de Hermite para cada subintervalo entre dos raíces. Los coeficientes se han ordenado de mayor a menor grado. El segundo fichero que se obtiene es resul_valores_interpolados_trozo_n (Figura 4.21).En él se pueden observar dos columnas, la primera columna es la x de evaluación dentro del intervalo entre dos raíces, y la segunda columna contiene el valor de la reconstrucción en la x de evaluación.

```
126.16882348   -400.06473449   546.82802833   -377.56598930   112.24494257   -42.63970368   37.56761659   -0.37968475
-5.86755850   -0.03382898   25.50560522   -0.00023780   -51.67712069   -0.00000008   31.41592654   0.00000000
-5.66006938   -0.15822283   25.53682124   -0.00447897   -51.67678460   -0.00001557   31.41592692   -0.00000000
-5.35338916   -0.43161788   25.64017843   -0.02595014   -51.67413874   -0.00020890   31.41593467   -0.00000014
-4.95289174   -0.90619637   25.87976350   -0.09273861   -51.66303679   -0.00130916   31.41599486   -0.00000154
-4.46559688   -1.62695438   26.33491696   -0.25180483   -51.62981289   -0.00545634   31.41628130   -0.00000998
-3.90004230   -2.62998600   27.09529546   -0.57118898   -51.54953828   -0.01752930   31.41728728   -0.00004581
-3.26614175   -3.94098432   28.25504070   -1.14004316   -51.38245413   -0.04691662   31.42015308   -0.00016534
-2.57500127   -5.57405139   29.90631335   -2.06625761   -51.07120880   -0.10957650   31.42715061   -0.00049974
-1.83873707   -7.53077425   32.13237686   -3.47153038   -50.53956549   -0.23011196   31.44231478   -0.00131637
126.16882348   -400.06473449   546.82802833   -377.56598930   112.24494257   -42.63970368   37.56761659   -0.37968475
```

Figura 4.20: Ejemplo de interpolación de Hermite segmentaria entre el extremo izquierdo del intervalo y la primera raíz encontrada.

En este ejemplo de la (Figura 4.20) tenemos 11 subintervalos. Cada uno da lugar a una fila del fichero, donde aparecen los coeficientes del polinomio de grado 8 construido. Dichos coeficientes se corresponden con la ordenación de los monomios de mayor a menor grado.

```
150   0.468331 9.950550
151   0.471475 9.959873
152   0.474618 9.968224
153   0.477761 9.975604
154   0.480904 9.982010
155   0.484047 9.987444
156   0.487190 9.991904
157   0.490334 9.995389
158   0.493477 9.997900
159   0.496620 9.999436
160   0.499763 9.999997
```

Figura 4.21: Ejemplo de salida de resultados a un fichero, indicando el valor de la reconstrucción en las x de evaluación.

Los datos (Figura 4.21) corresponden al valor del último polinomio de Hermite antes de cambiar de trozo en el ejemplo analizado.

Capítulo 5

Experimentos Numéricos.

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

5.1. Ejemplos numéricos de test con funciones .

5.1.1. Experimento 1.

En el capítulo anterior hemos visto el funcionamiento de la interfaz gráfica desarrollada. Vamos a ver ahora varios ejemplos de su utilización. Queremos comparar el método de Hermite con y sin detección de esquinas, comparando los errores obtenidos que se producen a la hora de llevar a cabo la reconstrucción. Además se hará un estudio del orden de aproximación numérica.

En primer lugar, vamos a analizar las diferentes pantallas que nos proporciona el programa cuando introducimos los parámetros vector de condiciones, que indica el número de derivadas en cada del punto del conjunto de puntos usado para interpolar localmente, el número de puntos a la izquierda de x_j incluyendo x_j y el número de puntos a la derecha de x_{j+1} incluyendo x_{j+1}, tal como podemos ver en la Figura 5.1. Trabajamos con la siguiente función:

$$f(x) = \begin{cases} 10sin(\pi x) & 0 \leq x \leq 0.5, \\ -50cos(\pi x) + 10 & 0.5 \leq x \leq 2, \\ 20cos(\pi x) - 40 & 2 \leq x \leq 3, \\ -40 + 5sin(\pi x) & 3 \leq x \leq 2\pi, \end{cases} \tag{5.1}$$

y consideramos el vector de condiciones $[0, 0, 0, 0]$ y los parámetros $l = 2$ y

$r = 2$.

La función se ha discretizado en el intervalo $[0, 2\pi]$ inicialmente en un mallado no uniforme de 197 puntos. Dicho mallado no uniforme ha sido generado con una función que introduce una variación aleatoria de hasta el 10 por ciento en la posición de los nodos del mallado uniforme. Se utiliza un mallado uniforme de evaluación con $n = 10000$ puntos.

Figura 5.1: Datos interfaz gráfica. Experimento 1.

El primer trozo de la función $f_1(x) = 10sin(\pi x)$ se aplica en el intervalo $[0, 0.5]$, la segunda función $f_2(x) = -50cos(\pi x) + 10$ se aplica en el intervalo $[0.5, 2]$, la tercera función $f_3(x) = 20cos(\pi x) - 40$ se aplica en el intervalo $[2, 3]$ y la cuarta función $f_4(x) = -40 + 5sin(\pi x)$ en el intervalo $[3, 2\pi]$. Se trata de funciones de clase infinito. Sin embargo f es derivable sólo en cada trozo, formando esquinas en los puntos 0.5, 2 y 3.

Con estos datos se obtienen las gráficas de las Figuras 5.2 y 5.3, que corresponden a la función original y a la reconstrucción por Hermite a trozos. También se obtienen las Figuras 5.4, 5.5 y 5.6, que corresponden a la localización de las esquinas. Por último obtenemos el error en norma infinito en la Figura 5.7.

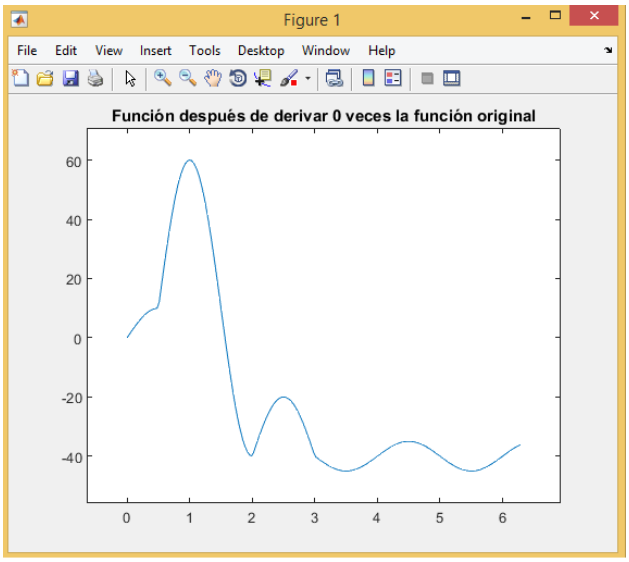

Figura 5.2: Función original. Experimento 1.

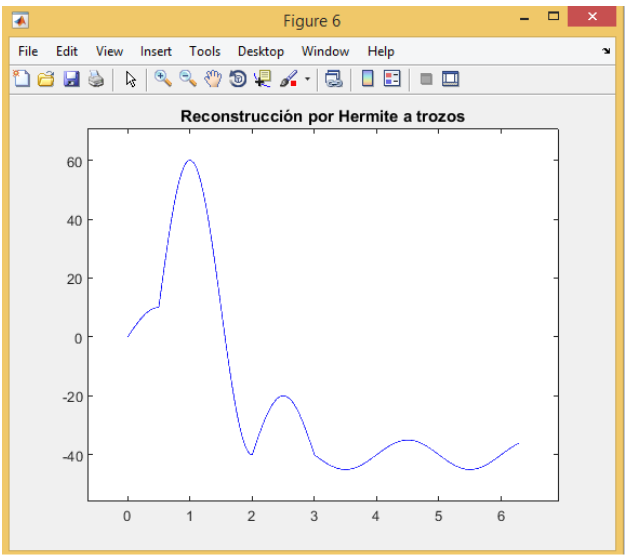

Figura 5.3: Recontrucción por Hermite a trozos. Experimento 1.

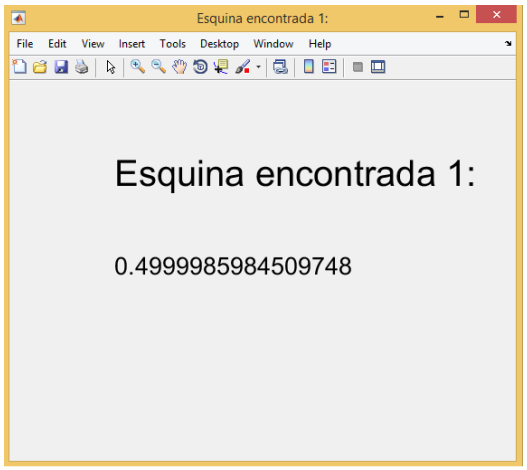

Figura 5.4: Localización de la esquina 1. Experimento 1.

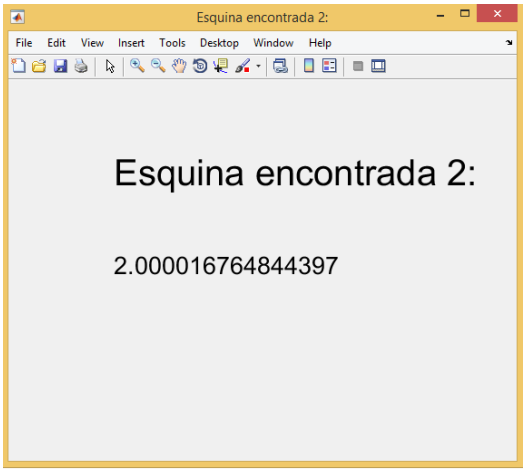

Figura 5.5: Localización de la esquina 2. Experimento 1.

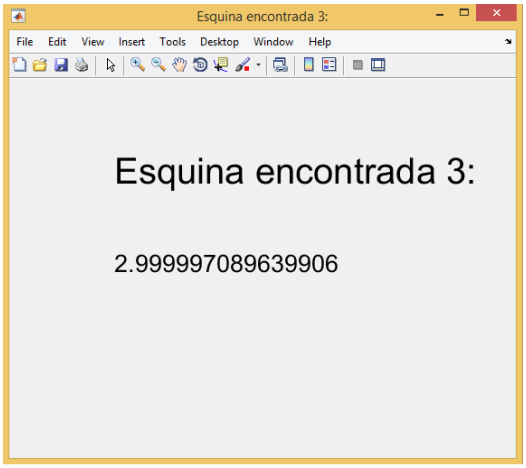

Figura 5.6: Localización de la esquina 3. Experimento 1.

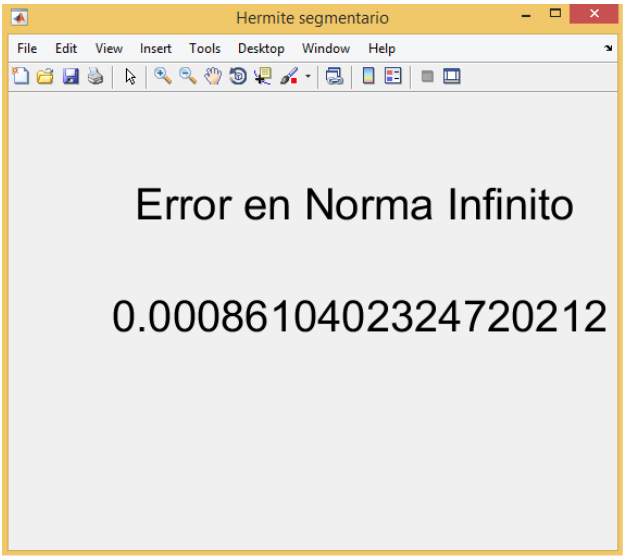

Figura 5.7: Error en norma infinito. Experimento 1.

Ahora vamos a hacer una comparación gráfica en la zona de la segunda esquina encontrada de la reconstrucción cuando usamos Hermite y Hermite con detección previa de esquinas.
Para funciones suaves (sin discontinuidades) en ambos métodos la reconstrucción mejora cuando aumentamos el grado de los polinomios. Por otro lado tampoco se han usado polinomios de grado muy alto porque producen muchas oscilaciones.
Vamos a ejemplificar los efectos producidos al aplicar Hermite para funciones con discontinuidades de esquina. Para ello usamos de nuevo la función $f(x)$ definida en 5.1.
Para los vectores de condiciones $[0, 0]$, que corresponde a una interpolación lineal, $[0, 1, 1]$ y $[1, 1, 1, 1]$, vamos a observar en las Figuras 5.8, 5.9, 5.10, 5.11, 5.12 y 5.13 que cuando aumentamos el grado de los polinomios, en el caso de Hermite segmentario, el error de la reconstrucción en las discontinuidades es más alto aunque el error de forma global disminuya. Sin embargo en el caso de Hermite con detección previa de esquinas el error de la reconstrucción es menor cuando se aumenta el grado de los polinomios. Si bien ya hemos dicho que no es conveniente considerar polinomios de grado muy alto.

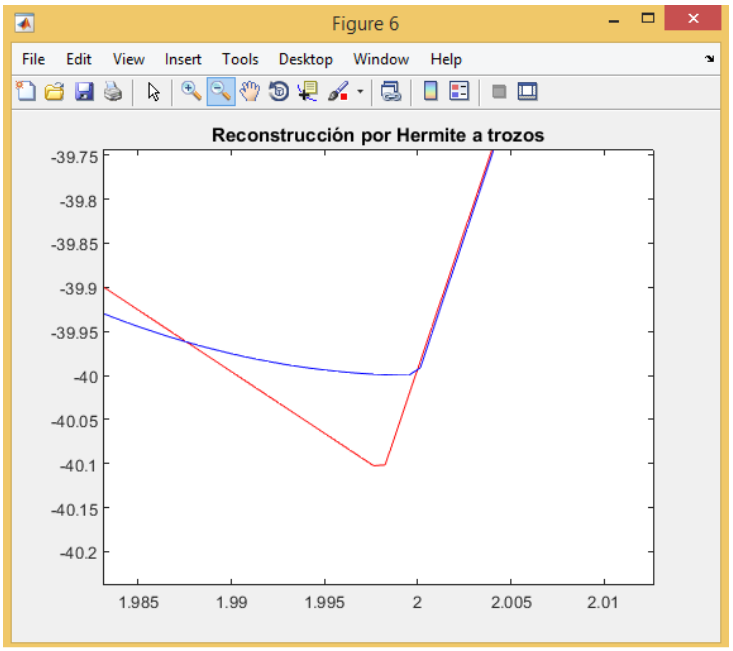

Figura 5.8: Reconstrucción de la segunda esquina encontrada para el vector de condiciones $[0, 0]$. Hermite con detección previa de esquinas.

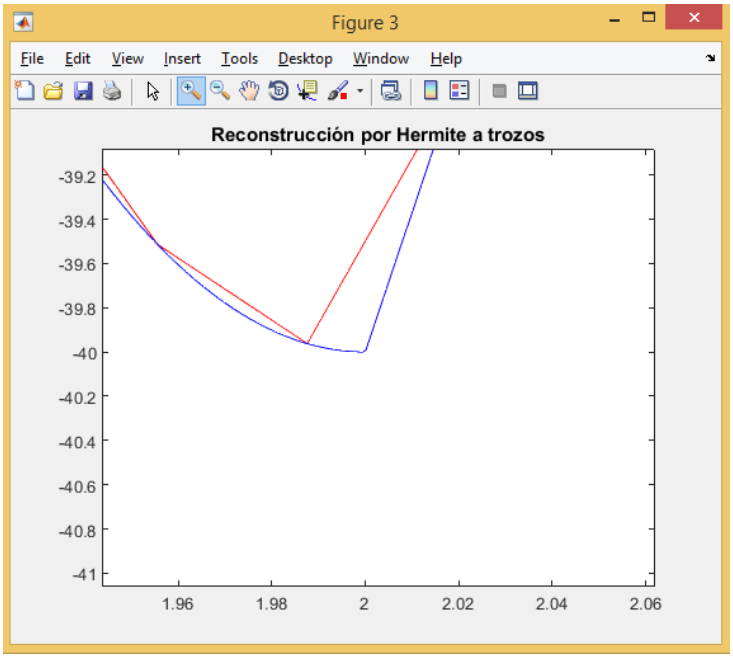

Figura 5.9: Reconstrucción de la segunda esquina encontrada para el vector de condiciones $[0, 0]$. Hermite segmentario.

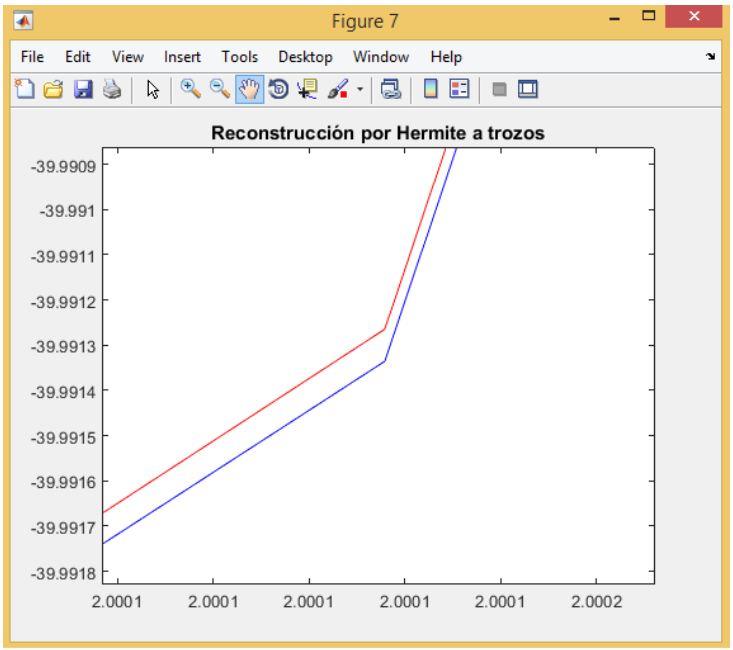

Figura 5.10: Reconstrucción de la segunda esquina encontrada para el vector de condiciones $[0, 1, 1]$. Hermite con detección previa de esquinas.

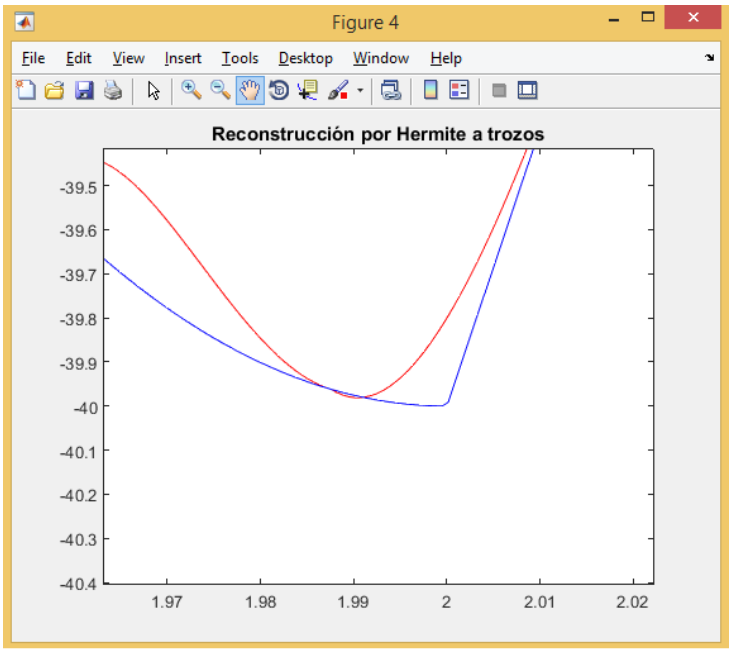

Figura 5.11: Reconstrucción de la segunda esquina encontrada para el vector de condiciones $[0, 1, 1]$. Hermite segmentario.

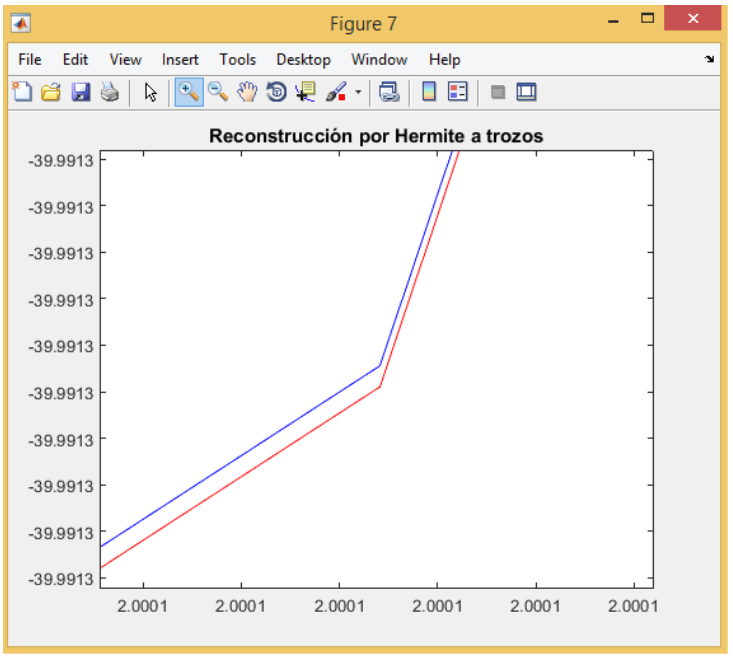

Figura 5.12: Reconstrucción de la segunda esquina encontrada para el vector de condiciones $[1, 1, 1, 1]$. Hermite con detección previa de esquinas.

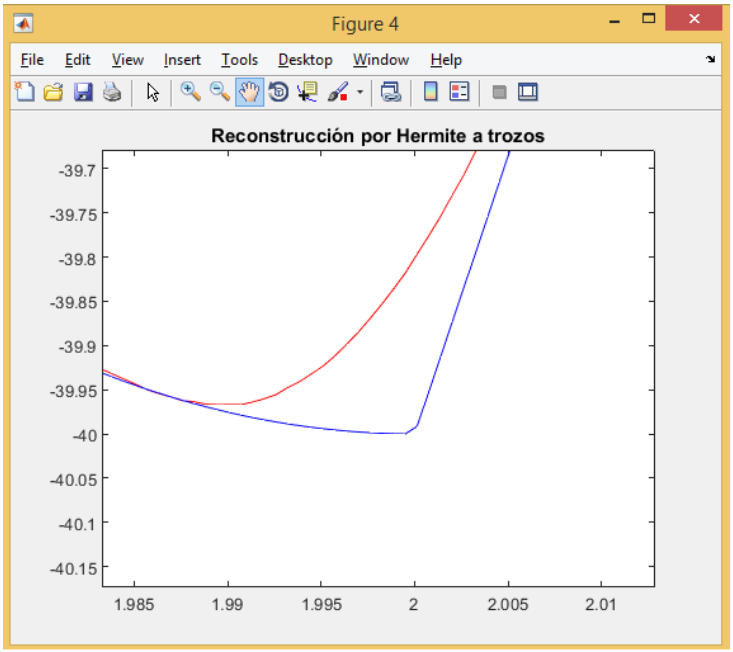

Figura 5.13: Reconstrucción de la segunda esquina encontrada para el vector de condiciones $[1, 1, 1, 1]$. Hermite segmentario.

Por último vamos a realizar un estudio del orden numérico p de aproximación numérico:

$$E_h \approx Ch^p,$$

$$E_{\frac{h}{2}} \approx C(\frac{h}{2})^p,$$

$$\frac{E_h}{E_{\frac{h}{2}}} \approx 2^p \Rightarrow p = log_2(\frac{E_h}{E_{\frac{h}{2}}}).$$

Ya que Hermite segmentario con funciones suaves es de orden h^N, donde N es el número de condiciones y la detección de esquinas propuesta también consigue orden h^N, el método combinado que realiza Hermite segmentario entre discontinuidades, debe ser también de orden h^N. Este hecho vamos a comprobarlo númericamente en algunos casos prácticos.
Consideramos como primer test sobre el orden numérico el caso del vector de condiciones $[0, 0, 0, 0]$, que debe dar $p = N = 4$. En la tabla 1.1 vemos los resultado obtenidos. Para generar la tabla se ha refinado el mallado inicial, y para cada nuevo mallado se ha ejecutado el algoritmo recién explicado para calcular el orden numérico.

puntos en el mallado	Error	Orden($p = log_2(E_h/E_{h/2})$)
197	$8{,}61 \cdot 10^{-4}$	
393	$1{,}77 \cdot 10^{-4}$	2,28226
785	$5{,}245 \cdot 10^{-6}$	5,0766
1569	$5{,}168 \cdot 10^{-8}$	6,6651
3137	$3{,}257 \cdot 10^{-9}$	3,9884

Tabla 5.1: Tabla de orden de aproximación para el método de Hermite con detección previa de esquinas obtenida para un mallado no uniforme inicial y los cuatro mallados derivados que se obtienen por subdivisiones sucesivas binarias. Parámetros $l = 2$, $r = 2$ y vector de condiciones $[0, 0, 0, 0]$.

En la Tabla 1.2 aparece el mismo tipo de estudio sobre el orden de aproximación, pero ahora para el método de Hermite sin detección previa de

esquinas. Vemos que en este caso el orden numérico p no converge a 4. Esto es debido a que la función para la que realizamos el test presenta discontinuidad en la primera derivada. Si la función fuera suave sí convergería a 4.

puntos en el mallado	Error	Orden($p = log_2(E_h/E_{h/2})$)
197	0,3476	
393	0,12410	1,48587
785	0,08893	0,48083
1569	0,03037	1,55003
3137	0,01127	1,42986

Tabla 5.2: Tabla de orden de aproximación para el método de Hermite sin detección previa de esquinas obtenida para un mallado no uniforme inicial y los cuatro mallados derivados que se obtienen por subdivisiones sucesivas binarias. Parámetros $l = 2$, $r = 2$ y vector de condiciones $[0, 0, 0, 0]$.

Con la función del ejemplo 2 haremos otro estudio numérico del orden de aproximación utilizando Hermite con un vector de condiciones distinto.

5.1.2. Experimento 2.

Trabajamos en este ejemplo con la función:

$$f(x) = \begin{cases} -50cos(\pi x) + 10 & 0 \leq x \leq 2, \\ 20sin(\pi x) - 40 & 2 \leq x \leq 2\pi, \end{cases} \tag{5.2}$$

y consideramos el vector de condiciones $[1, 1]$ y los parámetros $l = 1$ y $r = 1$.

Esta función ha sido discretizada en el intervalo $[0, 2\pi]$ inicialmente en un mallado no uniforme de 393 puntos. Se utiliza un mallado uniforme de evaluación con $n = 10000$ puntos.

Con estos datos de entrada se obtienen las Figuras 5.14, 5.15 y 5.16, que corresponden a la función original, a la función después derivar 1 vez la función original y a la reconstrucción por Hermite a trozos respectivamente. También se obtiene la Figura 5.17, que corresponde a la localización de la esquina. Por último obtenemos el error en norma infinito, Figura 5.18.

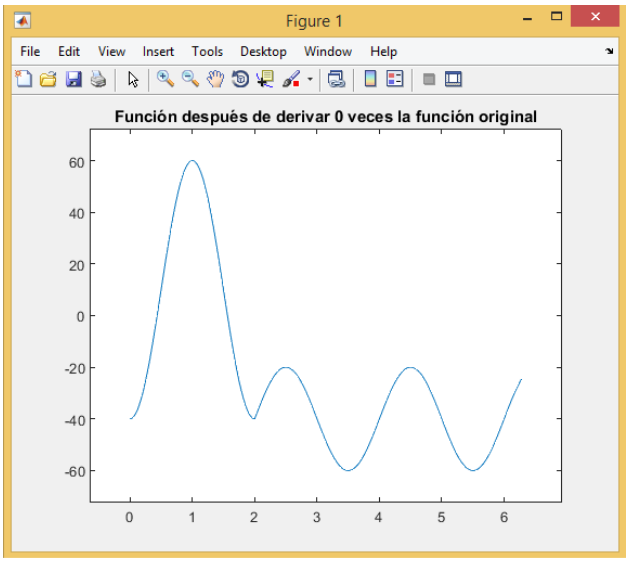

Figura 5.14: Función original. Experimento 2.

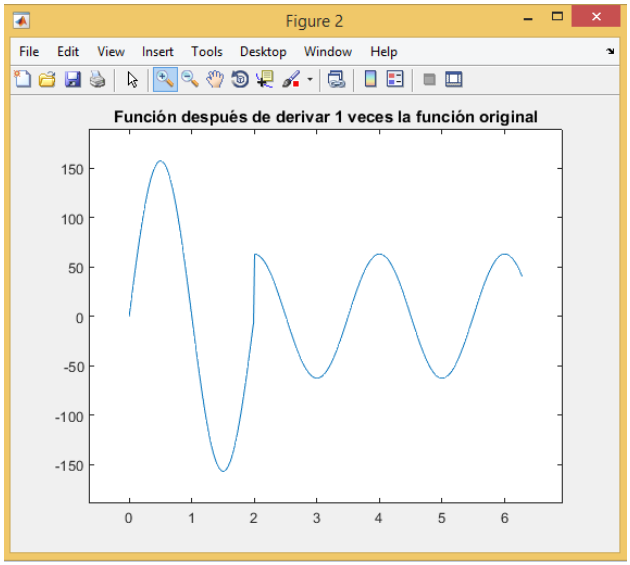

Figura 5.15: Función después derivar 1 vez la función original. Experimento 2.

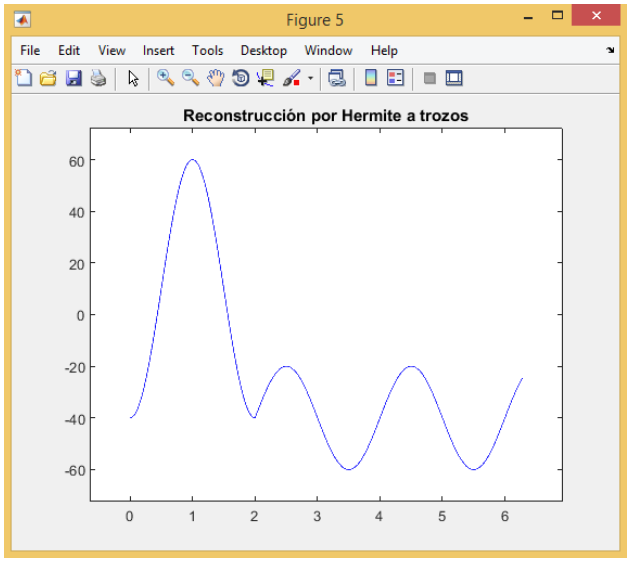

Figura 5.16: Recontrucción por Hermite a trozos. Experimento 2.

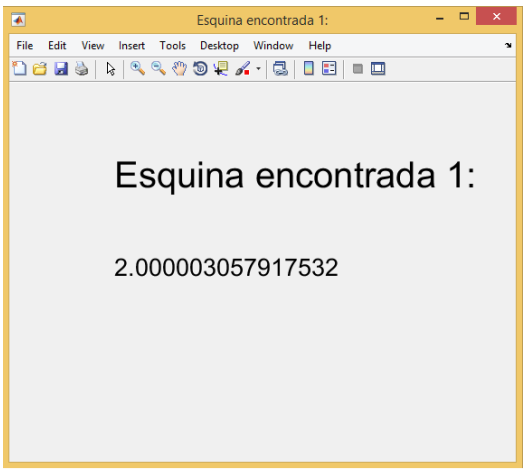

Figura 5.17: Localización de la esquina. Experimento 2.

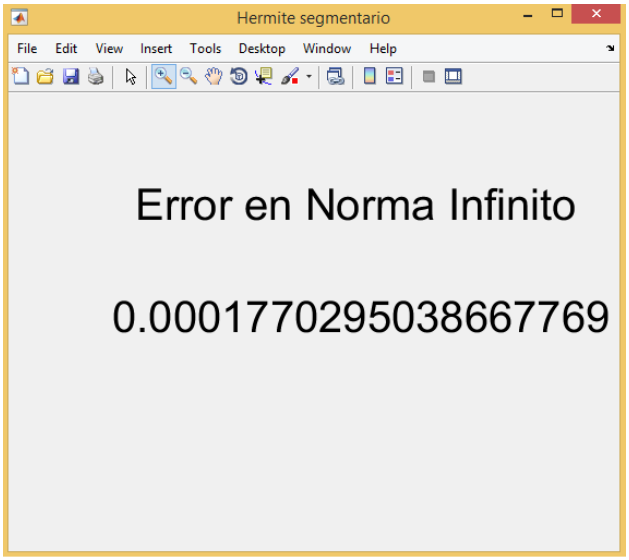

Figura 5.18: Error en norma infinito. Experimento 2.

Comparación gráfica alrededor de la esquina encontrada para el caso de Hermite con detección previa de esquinas (Figura 5.19) y Hermite segmentario (Figura 5.20) del ejemplo anterior.

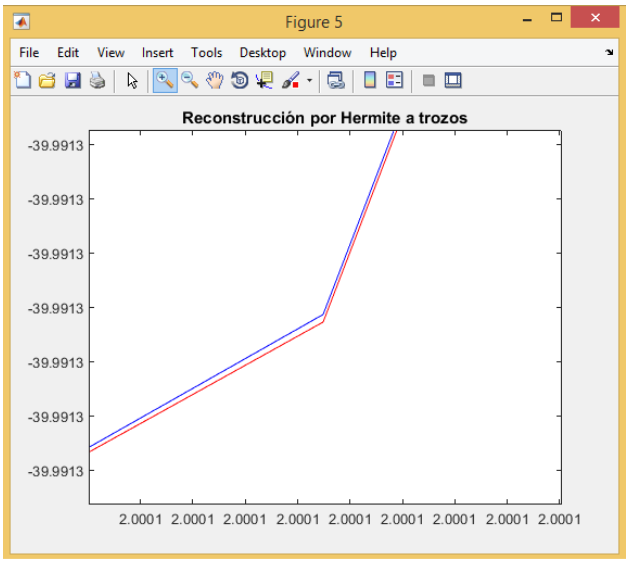

Figura 5.19: Reconstrucción de la esquina encontrada para el vector de condiciones $[1, 1]$. Hermite con detección previa de esquinas. Experimento 2.

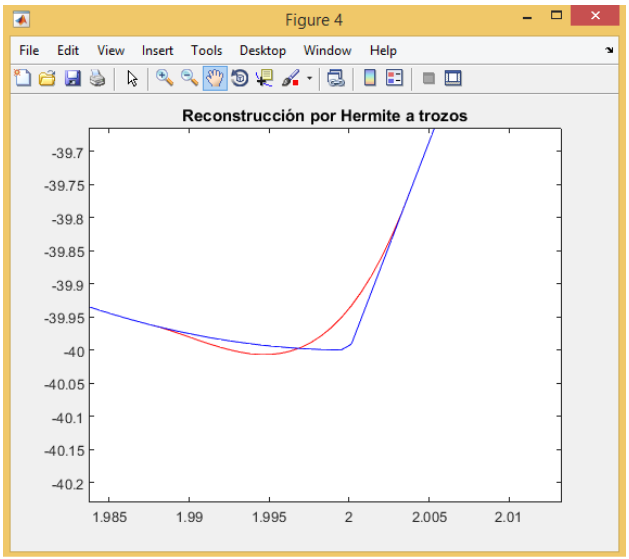

Figura 5.20: Reconstrucción de la esquina encontrada para el vector de condiciones $[1, 1]$. Hermite segmentario. Experimento 2.

A continuación vamos a realizar un estudio del orden de aproximación numérico p para este segundo experimento.

Vamos a comprobar que para el vector de condiciones $[1, 1]$ el orden numérico debe dar $p = N = 4$. En la tabla 1.3 vamos resultados obtenidos. Para generar la tabla se ha refinado el mallado inicial, y para cada nuevo mallado se ha ejecutado el algoritmo para calcular el orden numérico.

puntos en el mallado	Error	Orden($p = log_2(E_h/E_{h/2})$)
393	$1{,}7703 \cdot 10^{-4}$	
785	$5{,}21435 \cdot 10^{-6}$	5,08536
1569	$9{,}11699 \cdot 10^{-9}$	9,15971
3137	$2{,}03798 \cdot 10^{-10}$	5,48334
6273	$1{,}28039 \cdot 10^{-11}$	3,99249

Tabla 5.3: Tabla de orden de aproximación para el método de Hermite con detección previa de esquinas obtenida para un mallado no uniforme inicial y los cuatro mallados derivados que se obtienen por subdivisiones sucesivas binarias. Parámetros $l = 1$, $r = 1$ y vector de condiciones $[1, 1]$.

En la tabla 1.4 ponemos los resultados para Hermite sin detección de esquinas donde vamos a poder ver que no converge a $p = 4$.

puntos en el mallado	Error	Orden($p = log_2(E_h/E_{h/2})$)
393	0,05808	
785	0,05732	0,01894
1569	0,02034	1,49479
3137	0,00359	2,50029
6273	0,00382	−0,08859

Tabla 5.4: Tabla de orden de aproximación para el método de Hermite sin detección previa de esquinas obtenida para un mallado no uniforme inicial y los cuatro mallados derivados que se obtienen por subdivisiones sucesivas binarias. Parámetros $l = 1$, $r = 1$ y vector de condiciones $[1, 1]$.

5.2. Caso práctico

En esta sección vamos a ver el modelado de un deflector (Figura 5.21). El deflector es la parte final de un tubo de chimenea o caldera, y al estar siempre en exposición al aire libre, debemos poner especial cuidado en esta pieza en concreto. Tienen diferentes funciones para cada tipo de necesidad: el deflector antilluvia, el giratorio o los más completos: el deflector antirrevoco, el deflector aspirador o remates de chimeneas. El ejemplo viene de la toma de datos en una estación de medición.
Nosotros como no tenemos acceso a ninguna estación de medición hemos optado por simular los datos de la siguiente función.

$$f(x) = \begin{cases} -sqrt(-x) & -1 \leq x \leq 0, \\ -sqrt(x) & 0 \leq x \leq 1. \end{cases} \tag{5.3}$$

Figura 5.21: Deflector de una chimenea.

La función se ha discretizado en el intervalo $[-1, 1]$ con 250 valores, es decir, para el caso práctico expuesto partimos de 250 mediciones disponibles.

Se utiliza un mallado de evaluación uniforme con $n = 10000$ puntos para tener una reconstrucción precisa.

Se puede ver como queda la interfaz con los parámetros de entrada anteriores en la Figura 5.22, donde se han completado los ficheros datosx.mat (Figura 5.23) , datosy.mat (Figura 5.24) y evaluación.mat (Figura 5.25) con los datos requeridos. Se va a utilizar el método de Hermite con detección previa de esquinas.

Figura 5.22: Interfaz gráfica con los datos experimentales.

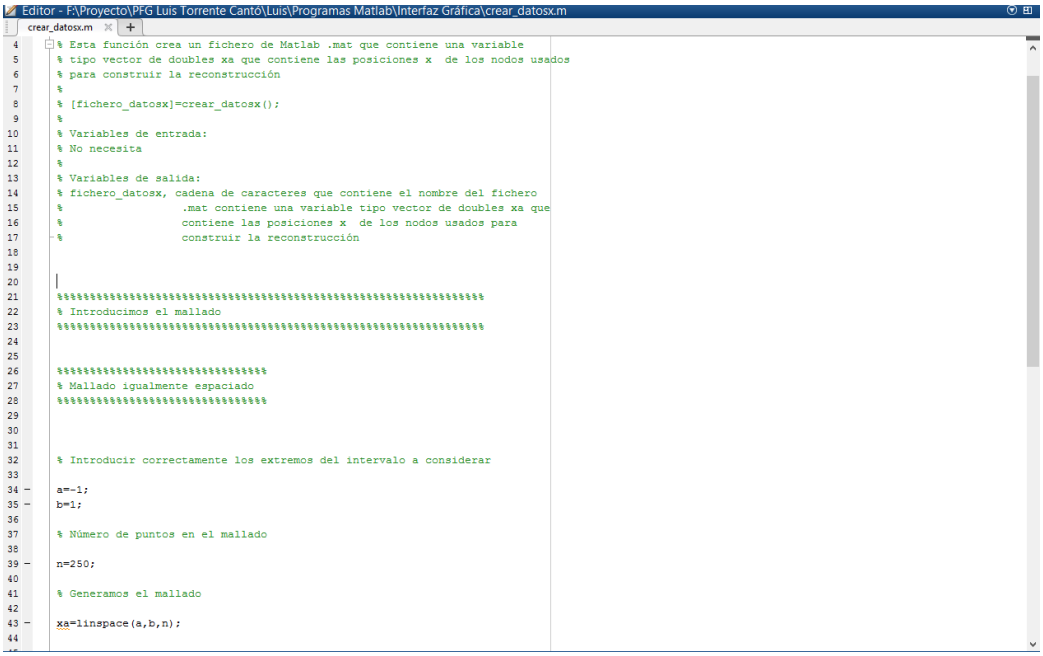

Figura 5.23: Fichero datosx.mat.

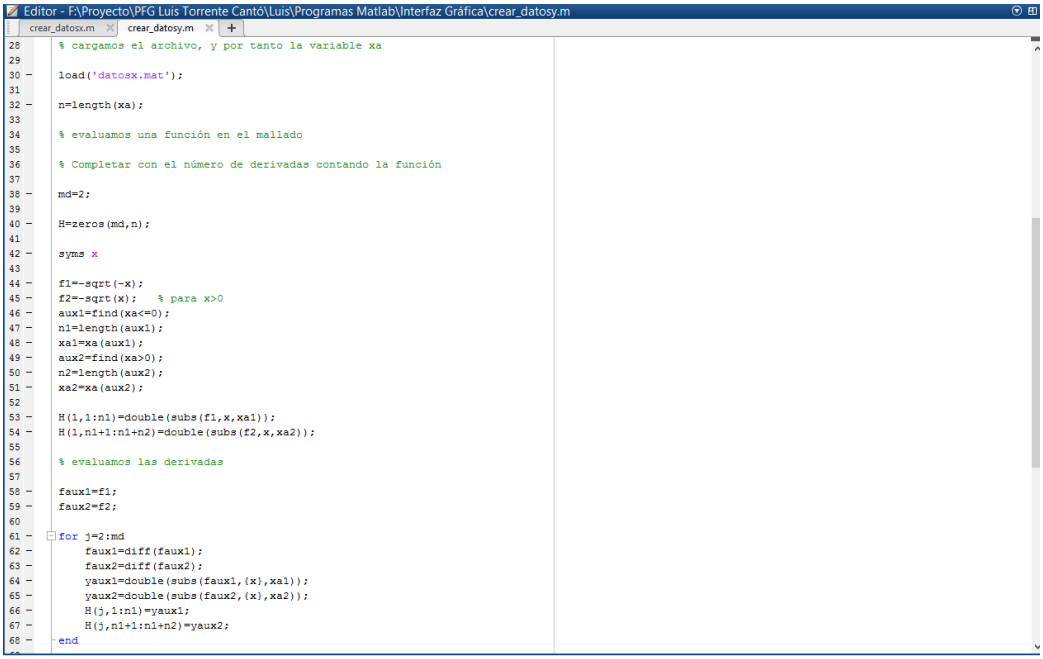

Figura 5.24: Fichero datosy.mat.

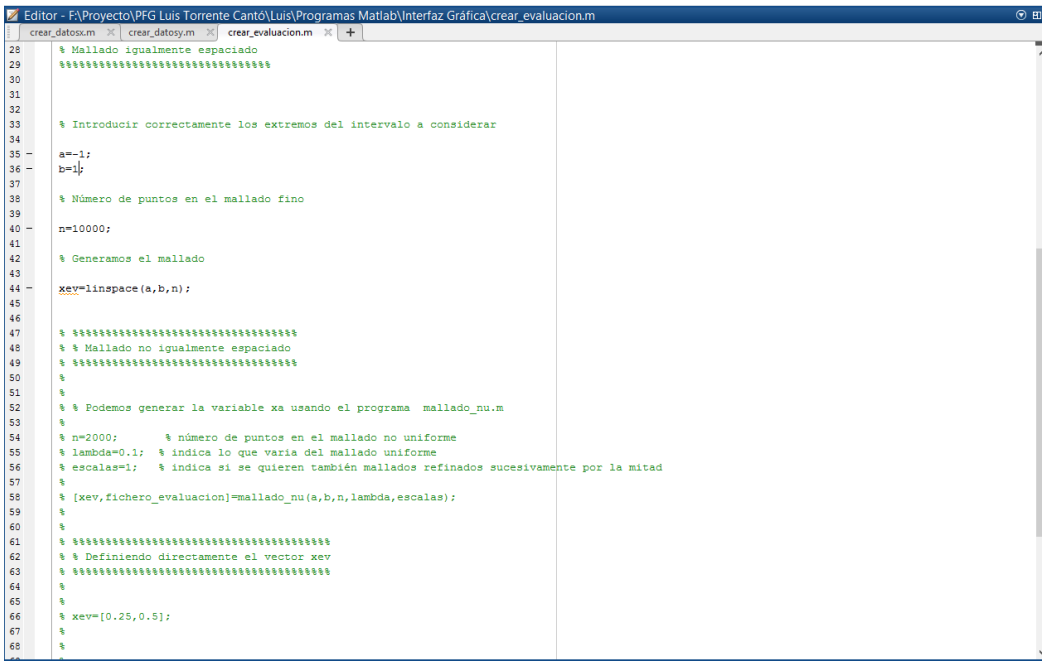

Figura 5.25: Fichero evaluación.mat.

A partir de los datos discretizados calculamos la reconstrucción (Figura 5.27) que podemos comparar con el perfil del deflector (Figura 5.26). Además en la Figura 5.28 aparece la localización de la esquina en la reconstrucción.

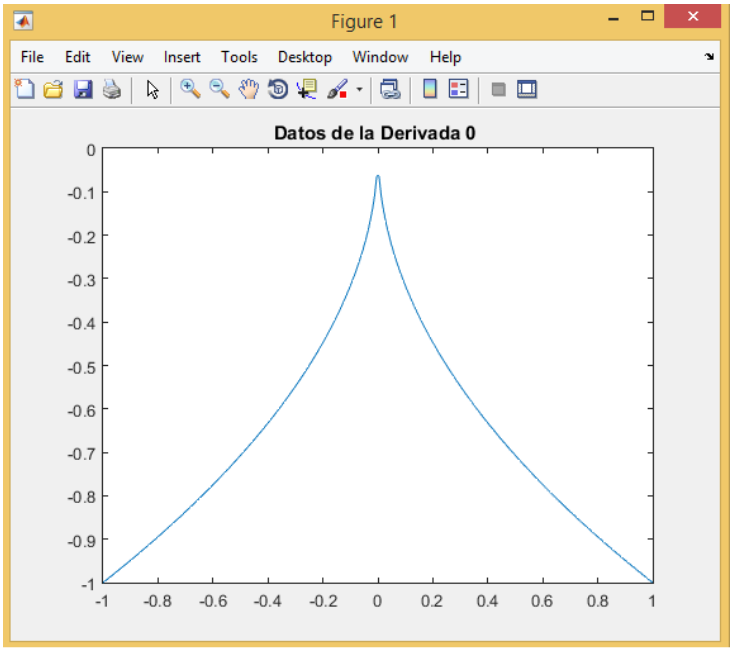

Figura 5.26: Datos iniciales.

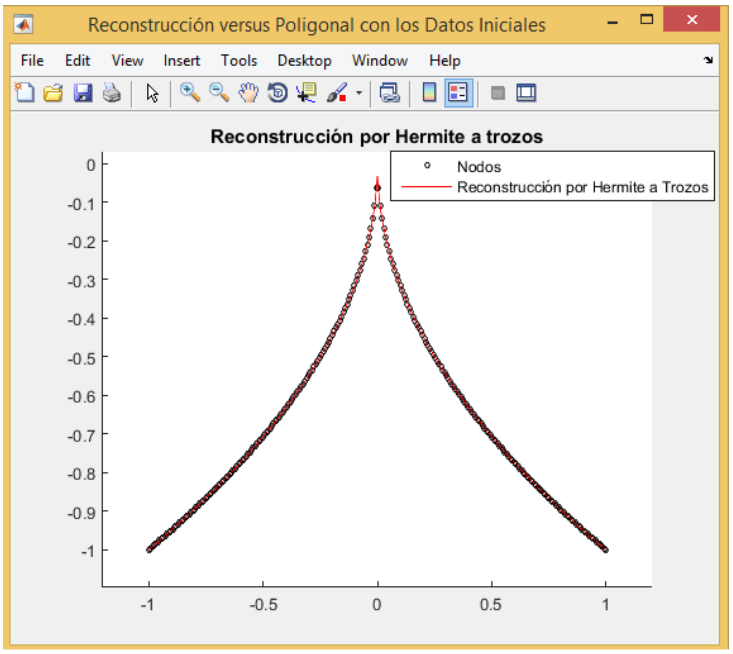

Figura 5.27: Reconstrucción vs datos iniciales.

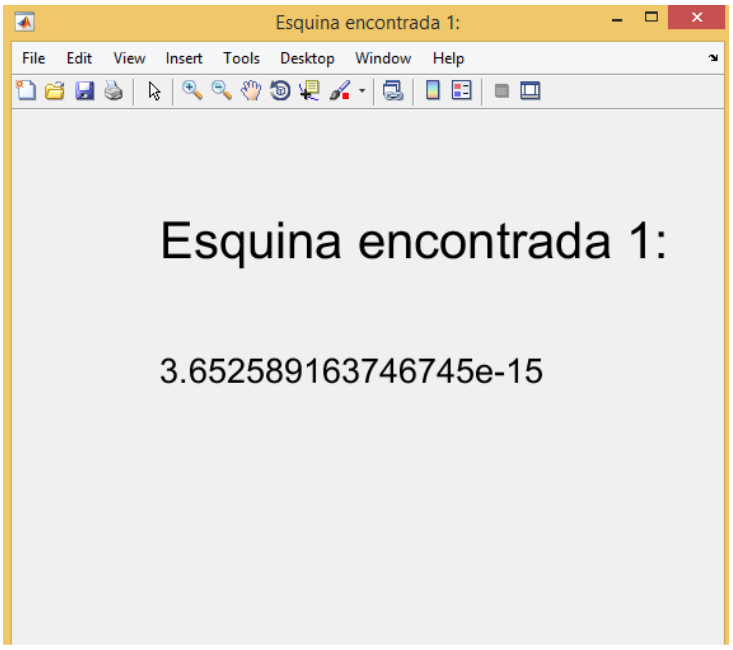

Figura 5.28: Esquina encontrada.

Capítulo 6

Conclusiones.

En este proyecto hemos realizado un estudio teórico práctico del algoritmo de Hermite con detección previa de esquinas, siendo uno de nuestros objetivos finales el aplicarlo al modelado de piezas industriales.

Una de las metas principales de este proyecto era la programación de los distintos algoritmos en Matlab. Además de implementar dichos códigos también se ha diseñado una Interfaz Gráfica de Usuario, con la que se consigue una mayor simplicidad a la hora de trabajar con los algoritmos estudiados. Para analizar la conveniencia de la utilización de los métodos presentados que incluyen una detección previa de esquinas, hemos realizado una comparación con el algoritmo de Hermite segmentario.

Hemos aplicado los algoritmos estudiados a algunos ejemplos numéricos.

Una de las conclusiones a la que hemos llegado es que a nivel práctico podemos ratificar los resultados teóricos, ya que hemos observado que cuando se trabja con funciones con alguna discontinuidad en la primera derivada la reconstrucción de Hermite segmentaria es menos precisa alrededor de la discontinuidad que cuando usamos Hermite con detección previa de esquinas. En el caso de funciones suaves (sin discontinuidades) ambos métodos funcionan de forma análoga.

Entre las posibles aplicaciones hemos visto el diseño de una pieza industrial simulando los datos de una estación de medición.

Bibliografía

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

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

[3] S. Amat, J. Liandrat, J Ruiz, J. Trillo, *On a compact non-extrapoling scheme for adaptive image interpolation.*, J. Franklin Inst., **349**(5), 1637-1647, (2012).

[4] F. Aràndiga and R. Donat, *Nonlinear multiscale descompositions: the approach of A. Harten, Numerical Algorithms*, **23**(2-3), 175-176,(2000).

[5] F. Aràndiga, R. Donat, P. Mulet, *Adaptive interpolation of images, Signal Process*, **83** (2), 459-464, (2003).doi:10.1016/S0165-1684(02)00445-0. URL http:// dx.doi.org/10.1016/S0165-1684(02)00445-0

[6] F.Aràndiga, A. Cohen and R. Donat, N. Dyn, *Interpolation and approximation of piecewise smooth functions. SIAM J. Numer. Anal*, **43**(1), 41-57, (2005).

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

[8] R. Burden, D. Faires, *Polinomios de Taylor,Numerical Analysis*, PWS,Boston , E.E.U.A, 94-99, (1985).

[9] A. Harten, *Eno schemes with subcell resolution, Journal of Computational Physics*, **83**(1), 148-184, (1989).