Capítulo V

Programación de Splines Cúbicos Suavizantes en Matlab

Para la programación de la función Spline, se ha elegido MATLAB, o también el denominado *''Laboratorio de matrices''* (abreviatura de *MATrix LABoratory*), que es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M).

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI), y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

Para la resolución de los splines cúbicos suavizantes, se ha creado una función capaz de resolver el algoritmo que se planteó y resolvió en el capítulo anterior, se ha tomado como punto de partida el caso de los spline cúbicos interpolantes desarrollado en un proyecto anterior por Irene Gallego Valdellós, (véase bibliografía). Para resolver dicho algoritmo en esta función tendremos que proporcionar una serie de *inputs*, y ésta a su vez nos devolverá una serie de *outputs,* la solución de nuestro problema.

Para el programa creado, ‘’Splines Cúbicos Suavizantes’’, tendremos que proporcionar las coordenadas, tanto de abscisas como de ordenadas, de los puntos donde se va a realizar la aproximación. En caso de que queramos evaluar la función spline en unos puntos determinados, habrá que facilitarle las abscisas de dichos puntos al programa.

Pero como no siempre utilizaremos esta evaluación, lo vamos a incluir como una opción. Por tanto como dato de entrada introduciremos $\mathrm{opx} = \text{'s'/'n'}$ para indicar si se calculan o no; y el retículo x, que serán puntos donde queremos evaluar la función polinómica a trozos.

Además de los nodos o puntos donde queremos realizar la aproximación y sus respectivos valores de la función, recordamos que estos datos no eran suficientes para poder llegar a la solución, se necesitaban dos ecuaciones adicionales, las cuales se obtenían de las condiciones de contorno en los extremos.
Para una aproximación mediante splines suavizantes, habrá que proporcionar al programa datos sobre las condiciones en los extremos izquierdo y derecho, CI y CD respectivamente. Por lo tanto proporcionaremos al programa los valores de la primera derivada, en caso de condiciones de contorno de primer tipo, de la segunda derivada para segundo tipo, y en caso de ser periódicas no necesitaremos dar datos adicionales.

Por último, habrá que proporcionar otro input, que será un vector con los coeficientes de peso o de suavizado, también conocido como ‘’weigth parameter’’, ρ. Como ya sabemos para valores pequeños de ρ, la función pasará por los puntos o cerca de éstos. Para valores altos de ρ, la función no pasará necesariamente por los nodos, como ya se demostró en el capítulo anterior.

Este valor del parámetro de suavizado puede ser dado por decisión propia del usuario, el cual podrá probar diferentes valores y decidir cual se ajusta mejor a sus necesidades, o incluso podrá elegirlo en función del grado de error o ruido que contengan sus mediciones de los nodos.

Como datos de salida u *outputs* se obtienen, aparte de los valores aproximados si es el caso, los polinomios de tercer grado que definirán el spline.

Asimismo se podrán visualizar las gráficas de la función spline, así como las funciones de la primera y segunda derivada, las cuales tienen un alto interés para verificar la pendiente o giro de la curva y las propiedades de *suavidad* del spline, pudiendo comprobar visualmente la uniformidad de la curvatura.

5.1 Scripts empleados.

En esta sección realizaremos una descripción de los scripts de Matlab que hemos utilizado para definir las matrices de coeficientes necesarias para resolver el sistema de ecuaciones. En este apartado incluiremos una descripción de los ficheros de datos que hemos utilizado. El programa con el que se ha realizado la resolución de splines cúbicos suavizantes, consiste en una serie de scripts de Matlab, junto con algunos archivos de datos en formato .txt, donde quedará reflejada la solución del sistema.

Para poder ejecutar dichos scripts es necesario que se encuentren todos en el directorio de trabajo de Matlab donde estamos trabajando, de no ser así el programa no funcionaría, es muy importante tener esto en cuenta.

En este apartado presentaremos los scripts que hemos desarrollado:

condicion_derivadasA.m

En este script se define la matriz de coeficientes A que acompaña a las η en el sistema de ecuaciones $\tilde{A}\eta = 6\ddot{H}z + F$, el cual se obtiene de la condición de continuidad de la primera derivada de la función spline cúbico suavizante.

```
function A=condicion_derivadas_A(h,CI,CD)

% Esta matriz serán los coeficientes que acompañan a los eta para
% condiciones en la frontera CI=1 y CD=1 es decir de segundo primer
% tipo en ambos extremos
% A es el término que acompaña a los eta tal que
% Ãeta=6Hz+F
% Este sistema de ecuaciones viene de la condición de continuidad de
% la
% primera derivada de la función Spline.
```

```
% Variables de entrada:
% h=vector h de distancias entre las abscisas
n=length(h);

% La primera ecuación
if CI==1
    A(1,1)=h(1)/3; A(1,2)=h(1)/6;
elseif CI==2
    A(1,1)=1;
elseif CI==3
    A(1,1)=1; A(1,n+1)=-1;
end

% Definimos el bucle que calculará los términos intermedios
for i=2:n
    A(i,i)=2*(h(i-1)+h(i));
    A(i,i+1)=h(i);
    A(i,i-1)=h(i-1);
end

% La última ecuación
if CD==1
    A(n+1,n)=-h(n)/6; A(n+1,n+1)=-h(n)/3;
elseif CD==2
    A(n+1,n+1)=1;
elseif CD==3
   A(n+1,1)=2*h(1); A(n+1,2)=h(1); A(n+1,n)=h(n); A(n+1,n+1)=2*h(n);
end
```

condicion_derivadasH.m

En este script se define la matriz de coeficientes H que acompaña a las z en el sistema de ecuaciones $\tilde{A}\eta = 6\ddot{H}z + F$, el cual se obtiene de la condición de continuidad de la primera derivada de la función spline cúbico suavizante.

```
function H=condicion_derivadas_H(h,CI,CD)

% Esta matriz serán los coeficientes que acompañan a los z para
% condiciones en la frontera CI=1 y CD=1 es decir de primer tipo en
ambos
% extremos
% H es el término que acompaña a los Z tal que
% Ãeta=6Hz+F
% Este sistema de ecuaciones viene de la condición de continuidad de
% la
% primera derivada de la función Spline.
% Variables de entrada
% h=vector h de distancias entre las abscisas

n=length(h);
H=zeros(n+1,n+1);
```

```
% La primera ecuación
if CI==1
    H(1,1)=-1/(6*h(1)); H(1,2)=1/(6*h(1));
elseif CI==2
    H(1,1)=0; H(1,2)=0;
elseif CI==4
    H(1,1)=1;

end

% Definimos el bucle que calculará los términos intermedios
for i=2:n
    H(i,i)=-(1/h(i-1)+1/h(i));
    H(i,i+1)=1/h(i);
    H(i,i-1)=1/h(i-1);
end

% La dos última ecuación
if CD==1
    H(n+1,n)=-1/(6*h(n)); H(n+1,n+1)=1/(6*h(n));
elseif CD==2
    H(n+1,n)=0; H(n+1,n+1)=0;
elseif CD==3
   H(n,n+1)=0; H(n,1)=1/h(n);
   H(n+1,1)=-1/h(1)-1/h(n); H(n+1,2)=1/h(1); H(n+1,n)=1/h(n);
H(n+1,n+1)=0;
elseif CD==4
   H(n+1,n+1)=1;
end
```

relacion_EtaLambda_U.m

En este script se define la matriz de coeficientes U que acompaña a las η en la ecuación $U\,\eta = V\lambda$. Esta ecuación se obtiene de la expresión desarrollada del funcional al diferenciar respecto de las η.

```
function U=relacion_EtaLambda_U(h)

% Matriz coeficientes obtenida de la expresión del funcional,
(dJ(s))/dEta)
% U*Eta=V*lambda
% U es la matriz que acompaña a lambda

% Variables de entrada:
% h=vector h de distancias entre las abscisas

% n es la dimensión del vector h
n=length(h);
```

```
% Definimos los términos invariantes
U(1,1)=2/3*h(1);
U(1,2)=h(1)/3;
U(n+1,n)=h(n)/3;
U(n+1,n+1)=2/3*h(n);

% Definimos el bucle que calculará el resto de términos
for i=2:n
    U(i,i)=2/3*(h(i-1)+h(i));
    U(i,i+1)=h(i)/3;
    U(i,i-1)=h(i-1)/3;
end
```

relacion_EtaLambda_V.m

En este script se define la matriz de coeficientes V que acompaña a los λ en la ecuación $U\,\eta = V\lambda$. Esta ecuación se obtiene de la minimización del funcional al diferenciar respecto de las η.

```
function V=relacion_EtaLambda_V(h,CI,CD)

% Matriz coeficientes obtenida de la expresión del funcional,
(dJ(s))/dEta)
% U*Eta=V*lambda
% V es la matriz q acompaña a lambda

% Variables de entrada:
% h=vector h de distancias entre las abscisas

% n es la dimensión del vector h
n=length(h);

% Las 2 primeras ecuaciones
if CI==1
    V(1,1)=h(1)/3; V(1,2)=h(1)/6;
    V(2,1)=h(1)/6; V(2,2)=(h(1)+h(2))/3; V(3,2)=h(2)/6;
elseif CI==2
    V(1,1)=-1; V(1,2)=h(1)/6;
    V(2,1)=0; V(2,2)=(h(1)+h(2))/3; V(3,2)=h(2)/6;

elseif CI==3
    V(1,1)=-1; V(1,2)=h(1)/6; V(1,n+1)=h(1)/3;
    V(2,1)=0; V(2,2)=1/3*(h(1)+h(2)); V(2,3)=h(2)/6; V(2,n+1)=h(1)/6;
elseif CI==4
    V(1,1)=0; V(1,2)=h(1)/6; V(1,n+1)=h(1)/3;
    V(2,1)=0; V(2,2)=1/3*(h(1)+h(2)); V(2,3)=h(2)/6; V(2,n+1)=h(1)/6;
end
```

```
% Definimos el bucle que calculará los términos intermedios
for i=3:n-1
    V(i,i)=(h(i-1)+h(i))/3;
    V(i-1,i)=h(i-1)/6;
    V(i+1,i)=h(i)/6;
end

% Las dos últimas ecuaciones
if CD==1
    V(n+1,n)=h(n)/6; V(n+1,n+1)=-h(n)/3;
    V(n,n-1)=h(n-1)/6; V(n,n)=(h(n-1)+h(n))/3; V(n,n+1)=-h(n)/6;
elseif CD==2
     V(n+1,n)=h(n)/6; V(n+1,n+1)=-1;
    V(n,n-1)=h(n-1)/6; V(n,n)=(h(n-1)+h(n))/3; V(n,n+1)=0;
elseif CD==3
    V(n+1,1)=1; V(n+1,n)=h(n)/6; V(n+1,n+1)=h(n)/3;
    V(n,n-1)=h(n-1)/6; V(n,n)=1/3*(h(n-1)+h(n)); V(n,n+1)=h(n)/6;
elseif CD==4
    V(n+1,1)=0; V(n+1,n)=h(n)/6; V(n+1,n+1)=0;
    V(n,n-1)=h(n-1)/6; V(n,n)=2/3*(h(n-1)+h(n)); V(n,n+1)=h(n)/6;

end
```

relacion_EtaLambda_J.m

En este script se define la matriz de coeficientes J, la cual se encuentra en la ecuación obtenida en el capítulo anterior $Z = Y + \frac{1}{2}\rho \, \mathrm{J} \, \lambda$.

Esta ecuación se obtiene de la minimización del funcional al diferenciar respecto de las z.

```
function J=relacion_ZetaLambda_J(h,CI,CD)

% Variables de entrada
% h=vector espaciado nodos
% De la minimización del funcional  y  (dJ(s))/dz
% se obtiene la relación
% Z=Y+1/2*rho*lambda*J
% En este programa definimos la matriz J
% n es la dimensión del vector h

n=length(h);
J=zeros(n+1,n+1);

% La primera ecuación
if CI==1
    J(1,1)=1/h(1); J(1,2)=-1/h(1);
    J(2,1)=-1/h(1); J(2,2)=1/h(1)+1/h(2); J(2,3)=-1/h(2);
elseif CI==2
    J(1,1)=0; J(1,2)=-1/h(1);
    J(2,1)=0; J(2,2)=1/h(1)+1/h(2); J(2,3)=-1/h(2);
elseif CI==3
    J(1,1)=0; J(1,2)=-1/2*1/h(1); J(1,n)=-1/2*1/h(n);
J(1,n+1)=1/2*1/h(1)+1/2*1/h(n);
    J(2,1)=0; J(2,2)=1/h(1)+1/h(2); J(2,3)=-1/h(2); J(2,n+1)=-1/h(1);
```

```
elseif CI==4
    J(1,1)=-1; J(1,2)=-1/h(1);
    J(2,1)=0; J(2,2)=1/h(1)+1/h(2); J(2,3)=-1/h(2);
end

% Definimos el bucle que calculará calculará los términos intermedios
for i=3:n-1
    J(i,i)=1/h(i-1)+1/h(i);
    J(i-1,i)=-1/h(i-1);
    J(i+1,i)=-1/h(i);
end

% La última ecuación
if CD==1
    J(n+1,n)=-1/h(n); J(n+1,n+1)=-1/h(n);
    J(n,n-1)=-1/h(n-1); J(n,n)=1/h(n-1)+1/h(n); J(n,n+1)=1/h(n);
elseif CD==2
    J(n+1,n)=-1/h(n); J(n+1,n+1)=0;
    J(n,n-1)=-1/h(n-1); J(n,n)=1/h(n-1)+1/h(n); J(n,n+1)=0;
elseif CD==3
     J(n,n-1)=-1/h(n-1); J(n,n)=1/h(n-1)+1/h(n); J(n,n+1)=-1/h(n);
     J(n+1,2)=-1/2*1/h(1); J(n+1,n)=-1/2*1/h(n);
J(n+1,n+1)=1/2*1/h(n)+1/2*1/h(1);
elseif CD==4
     J(n,n-1)=-1/h(n-1); J(n,n)=1/h(n-1)+1/h(n);
     J(n+1,n)=1/h(n); J(n+1,n+1)=-1;

end
```

5.2 Ficheros de resultados.

Estos ficheros serán los resultados obtenidos de la aproximación mediante splines cúbicos suavizantes, los ficheros que el usuario podrá visualizar son:

polinomios_splines.txt

En este archivo se incluye la fecha, día, mes y año, y la hora en la que se ejecutó el archivo. También se encuentra de manera simbólica el polinomio de tercer grado que define el spline en cada segmento.

resul_valores_aproximados.txt

En este archivo de nuevo se muestran además de la fecha y hora en que se ejecutó el programa, los resultados de los valores aproximados.

5.3 Fases del programa.

El programa Splines Cúbicos Suavizantes ha sido dividido en varias etapas perfectamente diferenciadas.

Primeramente es definida la función, en ella se indica el nombre del programa, las variables de entrada y las variables de salida. A continuación el usuario encontrará una serie de ayudas a las que podrá acceder mediante el comando ''help'' seguido del nombre del programa, en este caso ''splines_suavizantes''.

En una segunda etapa, son definidos una serie de errores que el usuario con pocos conocimientos sobre aproximación mediante splines puede encontrar muy útiles. Los errores con los que el usuario podrá encontrarse serán:

-En caso de que el usuario no introduzca al menos tres puntos o nodos de aproximación, no se podrá construir la función spline, por lo tanto el usuario será avisado mediante un mensaje de error.

-Si el usuario introdujera como condiciones en la frontera en alguno de los dos extremos condiciones de tercer tipo, implícitamente estará diciendo que en el extremo opuesto también serán de tercer tipo. En caso de que no sea así, el programa no funcionará y mostrará un mensaje de error.

En una tercera fase se definirán las matrices involucradas en los sistemas de ecuaciones lineales a resolver, las cuales como ya se ha explicado anteriormente dependen de las condiciones de contorno seleccionadas. Y se resuelven los sistemas obteniendo los coeficientes necesarios para la definición del spline.

En la cuarta fase del programa, se evaluarán los splines en los diferentes segmentos de acuerdo al polinomio obtenido en el capítulo anterior,

$$S(x) = S_i(x) = z_i(1-t) + z_{i+1}t - \frac{h_i^2}{6}t(1-t)[(2-t)\eta_i + (1+t)\eta_{i+1}].$$

Para concluir, se construirán de una manera simbólica los polinomios a partir de los coeficientes ya obtenidos. El programa dibujará mediante la función plot las diferentes gráficas de la función spline cúbico suavizante, así como su primera y segunda derivada. Además estos resultados, como ya se ha comentado, serán grabados en un fichero en formato .txt.

5.4 Código Matlab para la resolución de Splines Cúbicos Suavizantes.

```
function
Splines_Suavizantes(t,y,CI,S1a,S2a,CD,S1b,S2b,opx,x,opg1,opg2,opg3,opg
4,tol_rho)

% Esta función busca hallar las ecuaciones que definen los diferentes
% tramos que componen el spline
% suavizante y dibujar las gráficas de la curva, su primera y segunda
% derivada, así como los valores obtenidos al interpolar y su gráfica.
% Creando ficheros que recojan la información.
%
Splines_Suavizantes(t,y,CI,S1a,S2a,CD,S1b,S2b,opx,x,opg1,opg2,opg3,opg
4,tol_rho)
%
% Variables de entrada:
% t:abscisas de la tabla de valores de los nodos
% y:ordenadas de la tabla de valores de los nodos
% CI:condición de contorno en el extremo izquierdo
% CD:condición de contorno en  el extremo derecho

% Posibles condiciones de contorno:
% 1- De primer tipo: valores de la primera derivada
% 2- De segundo tipo: valores de la segunda derivada
% 3- De tercer tipo: periódicas de periodo T=b-a
%    S'(a)=S'(b); S''(a)=S''(b). Si CI=3->CD=3
% 4- De cuarto tipo: Tres veces diferenciable en los puntos t2 y/o tm-
1
%
% S1a:valor de la primera derivada en el extremo izquierdo
% S2a:valor de la segunda derivada en el extremo izquierdo
% S1b:valor de la primera derivada en el extremo derecho
% S2b:valor de la segunda derivada en el extremo derecho
%
% opx:parámetro que nos indica si debemos calcular o no los valores
interpolados
% 's':se calculan; 'n': no se calculan
% x:retículo de puntos donde queremos evaluar la función polinómica a
trozos
```

```
% opg1:parámetro que nos indica se debemos dibujar o no la gráfica de
los valoes interpolados
% opg2:parámetro que nos indica se debemos dibujar o no la gráfica de
los splines
% opg3:parámetro que nos indica se debemos dibujar o no la gráfica de
las primeras derivadas
% opg4:parámetro que nos indica se debemos dibujar o no la gráfica de
las segundas derivadas
% si opg1=1 entonces dibujará la gráfica
% tol_rho vector de tolerancias para los pesos del funcional, será el
que determine el
% grado de suavidad de la curva, es decir, que la curva realiza menos
oscilaciones
%
% EJEMPLO:
% t=[1,3,4,6]
% y=[2,-1,3,1]
% CI=2 ; CD=2
% S1a=0; S1b=0 (no se van a utilizar, da igual el dato)
% S2a=2; S2b=2
% opx='s'
% x=[1:0.2:6]
%
% Splines_Suavizantes([1,3,4,6],[2,-
1,3,1],2,0,0,2,0,0,'s',[1:0.2:6],1,1,1,1,[0.001,0.001,0.001,0.001]);

% Número de puntos en la tabla, longitud del vector nodos
n=length(t);
pe=length(tol_rho);
% Comprobamos que las entradas son correctas.
% El número mínimo de nodos para poder realizar un Spline es 3
if n<3
    uiwait(msgbox('El número de puntos de control debe ser mayor que
2', 'Mensaje de error',...
        'error','modal'));
    return;
% Si no se especifican las condiciones en la frontera, se avisará el
error
elseif CI==0 | CD==0
    uiwait(msgbox('Falta especificar alguna de las condiciones de
frontera', 'Mensaje de error',...
        'error','modal'));
    return;


% Si se marca condiciones periódicas o 3, va intrínseco que se tienen
% que dar en
% los tres exremos
elseif CI==3 & CD~=3
    uiwait(msgbox('Si CI=3 entonces CD debe ser también 3', 'Mensaje
de error',...
        'error','modal'));
    return;
elseif CD==3 & CI~=3
    uiwait(msgbox('Si CD=3 entonces CI debe ser también 3', 'Mensaje
de error',...
        'error','modal'));
    return;
```

```
elseif CD==3 & y(1)~=y(n)
    uiwait(msgbox('Para que la función sea periódica debe valer lo
mismo en los extremos', 'Mensaje de error',...
        'error','modal'));
    return;
% Con 3 nodos la condición CI=4 ya implica CD=4 y nos falta una
condición
elseif n==3 & CI==4 & CD==4
    uiwait(msgbox('Con 3 nodos la condición CI=4 ya implica CD=4 y nos
falta una condición', 'Mensaje de error',...
        'error','modal'));
    return;
elseif pe<3
     uiwait(msgbox('Se necesitan especificar al menos 3 coeficientes
de peso', 'Mensaje de error',...
        'error','modal'));
end


% Definimos el vector h de distancias entre las abscisas
h=diff(t,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%  Selección de las matrices  %%%%%%%%%%
%%%%%%  coeficiente en función     %%%%%%%%%%
%%%%%%  de las condiciones en la frontera %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Elegimos las matrices coeficiente en función de las condiciones de
contorno dadas
% Las matrices varían en función de las condiciones de la frontera
dato
% Elegiremos las matrices mediante un condicional

A=condicion_derivadas_A(h,CI,CD);
H=condicion_derivadas_H(h,CI,CD);
V=relacion_EtaLambda_V(h,CI,CD);
U=relacion_EtaLambda_U(h);
J=relacion_ZetaLambda_J(h,CI,CD);
F=relacion_EtaZeta_F(h,CI,CD,S1a,S1b,S2a,S2b);


% Definimos la matriz Auxiliar K1

K1_aux=6*J*V^(-1)*U*A^(-1)*H;
%rho=min(tol_rho,1./sum(abs(K1_aux)))
K1=diag(tol_rho)*K1_aux;

% Definimos la matriz Auxiliar K2
K2=diag(tol_rho)*J*inv(V)*U*inv(A)*F;

% Resolución del sistema lineal Cz=B
C=eye(n)-1/2*K1;
B=y+K2;
z=C\B;
```

```
% Cálculo de los valores de la segunda derivada en los nodos eta's

eta=A^(-1)*(6*H*z+F);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Evaluamos los Splines en x                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if opx=='s'

    % Encuentra el trozo polinómico a evaluar según cada entrada de x
    if x(1)==t(1)
        Si(1)=z(1);
        minim=Si; maxim=Si;
    else
        ind=find((x(1)>t)==0);
        ind=ind(1)-1;
        ind=ind(1);
        % Aplica la fórmula para calcular S_ind(x)
        tn=(x(1)-t(ind))/h(ind);
        Si(1)=z(ind)*(1-tn)+z(ind+1)*tn-h(ind)^2/6*tn*(1-tn)*((2-
tn)*eta(ind)+(1+tn)*eta(ind+1));
        minim=Si; maxim=Si;
    end

    for i=2:length(x)
        ind=find((x(i)>t)==0);
        ind=ind(1)-1;
        ind=ind(1);
        % Aplica la fórmula para calcular S_ind(x)
        tn=(x(i)-t(ind))/h(ind);
        Si(i)=z(ind)*(1-tn)+z(ind+1)*tn-h(ind)^2/6*tn*(1-tn)*((2-
tn)*eta(ind)+(1+tn)*eta(ind+1));
        if Si(i)<minim
            minim=Si(i);
        elseif Si(i)>maxim
            maxim=Si(i);
        end
    end

    if opg1==1
        figure(1);
        %Dibuja los puntos de control en la gráfica
        b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
        hold on;
        %Dibuja los valores interpolados en la gráfica
        b2=plot(x,Si,'ro','MarkerSize',3);
        hold on;
        axis([t(1)-0.1*(t(2)-t(1)) t(length(t))+...
            0.1*(t(2)-t(1)) min(min(y),minim)-...
            0.1*(max(y)-min(y)) max(max(y),maxim)+0.1*(max(y)-
min(y))]);
        axis equal;
        legend([b1,b2],'Puntos de Control','Valores Interpolados');
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construimos los polinomios de la interpolación %
%                 Segmentaria                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

syms r;

%%%%% Establecemos una variable simbólica%%%%%


for i=1:n-1
    S(i)=z(i)*(1-(r-t(i))/h(i))+z(i+1)*(r-t(i))/h(i)-...
        h(i)^2/6*(r-t(i))/h(i)*(1-(r-t(i))/h(i))*...
        ((2-(r-t(i))/h(i))*eta(i)+(1+(r-t(i))/h(i))*eta(i+1));
end


if opg2==1
    %Dibuja los puntos de control en la gráfica
    figure(2);
    b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
    hold on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Dibuja cada uno de los polinomios%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    Ms=0;
    ms=0;
    for i=1:n-1
        b3=ezplot(S(i),[t(i),t(i+1)]);
        auxX=t(i):(t(i+1)-t(i))/10:t(i+1);
        auxY=subs(S(i),auxX); MauxY=max(auxY); mauxY=min(auxY);
        if MauxY>Ms
            Ms=MauxY;
        end
        if mauxY<ms
            ms=mauxY;
        end
    end
    axis([t(1)-0.1*(t(2)-t(1)) t(length(t))+...
        0.1*(t(2)-t(1))  ms-0.1*(Ms-ms)  Ms+0.1*(Ms-ms)]);
    axis equal;
    legend([b1,b3],'Puntos de Control','Splines Cúbicos');
    title('Ajuste por Splines Cúbicos Suavizantes');




    % dibuja los polinomio splines también en la interfaz gráfica
    axes(handles.grafica);


    b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
    hold on;
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Dibuja cada uno de los polinomios%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    for i=1:n-1
        b3=ezplot(S(i),[t(i),t(i+1)]);
    end
    axis([t(1)-0.1*(t(2)-t(1)) t(length(t))+...
        0.1*(t(2)-t(1))  ms-0.1*(Ms-ms)  Ms+0.1*(Ms-ms)]);
    axis equal;
    legend([b1,b3],'Puntos de Control','Splines Cúbicos');
    title('Ajuste por Splines Cúbicos Suavizantes');
    hold off

end


if opg3==1
    %Dibuja cada una de las primeras derivadas
    figure(3);
    hold on
    Ms=0;
    ms=0;

% Diferenciamos el polinomio construido, para obtener el polinomio de
la
% primera derivada.
    S1=diff(S,1);
    for i=1:n-1
        b4=ezplot(S1(i),[t(i),t(i+1)]);
        auxX=t(i):(t(i+1)-t(i))/10:t(i+1);
        auxY=subs(S1(i),auxX); MauxY=max(auxY); mauxY=min(auxY);
        if MauxY>Ms
            Ms=MauxY;
        end
        if mauxY<ms
            ms=mauxY;
        end
    end
    axis([t(1)-0.1*(t(2)-t(1)) t(length(t))+...
        0.1*(t(2)-t(1))  ms-0.1*(Ms-ms)  Ms+0.1*(Ms-ms)]);
    legend(b4,'Primera Derivada Splines Cúbicos');
    title('Primera Derivada de los Splines Cúbicos Suavizantes');
end


if opg4==1
    %Dibuja cada una de las segundas derivadas
    figure(4)
    hold on
```

```
% Diferenciamos el polinomio construido dos veces, para obtener el
polinomio de la
% segunda derivada.
    S2=diff(S,2);
    Msd=subs(S2(1),t(1));
    msd=subs(S2(1),t(1));
    for i=1:n-1
        b5=ezplot(S2(i),[t(i),t(i+1)]);
        aux=subs(S2(i),t(i+1));
        if aux>Msd
            Msd=aux;
        elseif aux<msd
            msd=aux;
        end
    end
    axis([t(1)-0.1*(t(2)-t(1)) t(length(t))+...
        0.1*(t(2)-t(1)) msd-0.1*(Msd-msd)  Msd+0.1*(Msd-msd)]);
    legend(b5,'Segunda Derivada Splines Cúbicos');
    title('Segunda Derivada de los Splines Cúbicos Suavizantes');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Salida de los polinomio a un fichero
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

if opx=='s'
    fid=fopen('resul_valores_aproximados.txt','w');

fprintf(fid,'**********************************************************
**********************\n');
    hora=clock;
    fprintf(fid,'Resultado ejecutado el dia %d-%d-%d a las %d : %d :
%d\n',...
        hora(3),hora(2),hora(1),hora(4),hora(5),fix(hora(6)));

fprintf(fid,'**********************************************************
**********************\n\n');
    for i=1:length(x)
        fprintf(fid,'%f %f \n',x(i),Si(i));
    end
    fclose(fid);
end

%  Se abre o crea un archivo y se escribe en él para los resultados de
los polinomios del Spline

fid=fopen('polinomios_splines.txt','w');
fprintf(fid,'**********************************************************
**********************\n');
hora=clock;
fprintf(fid,'Resultado ejecutado el dia %d-%d-%d a las %d : %d :
%d\n',...
    hora(3),hora(2),hora(1),hora(4),hora(5),fix(hora(6)));
fprintf(fid,'**********************************************************
**********************\n\n');
signos='----';
```

```
for i=1:length(t)-1
    coefi=sym2poly(expand(S(i)));
    for j=1:4
        if coefi(j)>=0
            signos(j)='+';
        end
    end
    fprintf(fid,' %c %f x^3 %c %f x^2 %c %f x %c %f  \n',...
        signos(1),abs(coefi(1)),signos(2),abs(coefi(2)),signos(3),...
        abs(coefi(3)),signos(4),abs(coefi(4)));
end
fprintf(fid,'\n');
fclose(fid);
```

5.5 Ejemplo de ejecución del código en Matlab.

Para ejecutar el código creado, introducimos en la ventana de comandos o ‘’command window’’ lo siguiente:

>>Splines_Suavizantes([1 3 4 6]',[2 -1 3 2]',1,2,0,1,2,0,'s',[1:0.2:6],1,1,1,1,[0.021, 0.021, 0.021, 0.021])

Esto quiere decir que vamos a realizar una aproximación mediante splines cúbicos suavizante en los puntos con valores de abcisas [1 3 4 6] y ordenadas [2 -1 3 2]. Las condiciones en la frontera izquierda serán de primer tipo, con un valor de la primera derivada igual a 2, y en la frontera derecha también serán de primer tipo con un valor de la primera derivada igual a 2. El intervalo donde se construirá el spline será [1,6]. Los siguientes cuatro valores indican que queremos que se nos muestren los gráficos de los puntos donde se realiza la aproximación, la función spline construida y la primera y segunda derivada de la función. Por último a los coeficientes de peso se les ha dado un valor de 0.021, permitiendo cierto grado de suavizado.

Los gráficos obtenidos fueron:

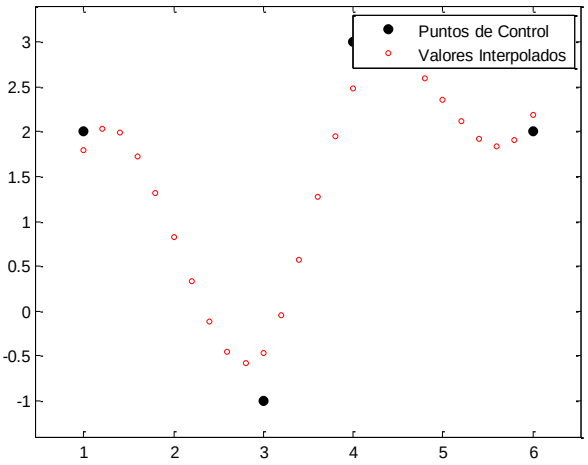

Figura 5.1: Puntos de control.

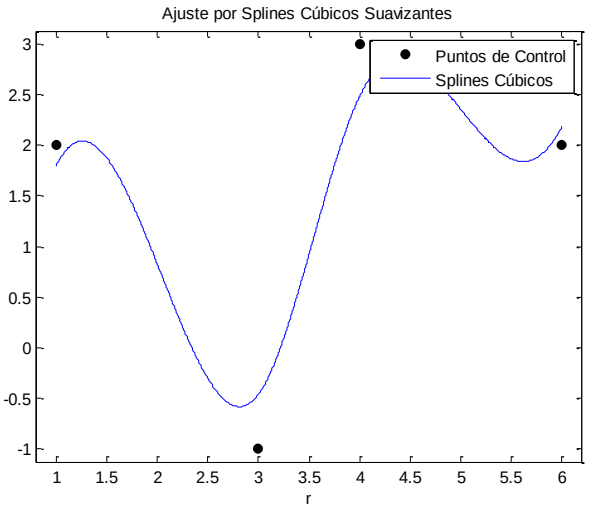

Figura 5.2: Función de ajuste mediante splines cúbicos suavizantes.

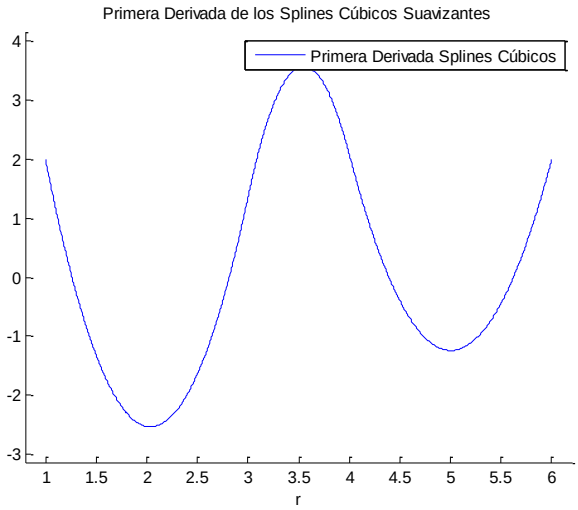

Figura 5.3: Primera derivada de la función spline.

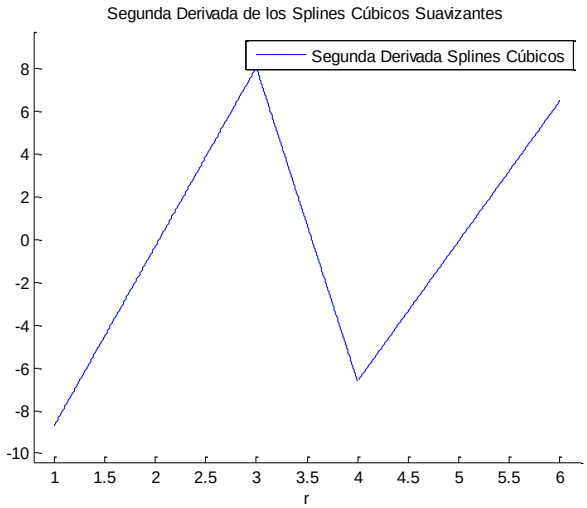

Figura 5.4: Segunda derivada de la función spline.

El resultado obtenido muestra una función spline en la cual efectivamente el valor de la primera derivada coincide con el valor de la primera derivada en los extremos introducido al ejecutar la función en matlab.

Los polinomios resultantes de la aproximación quedan reflejados en el archivo polinomios_splines.txt. Los polinomios resultantes son:

```
 + 1.401910 x^3 - 8.573740 x^2 + 14.941749 x - 5.976126=0
+ 2.450185 x^3 + 26.095119 x^2 + 89.064825 x + 98.030449=0
+ 1.095101 x^3 + 16.448314 x^2 + 81.108904 x + 128.867857=0
```