Universidad Politécnica de Cartagena

TRABAJO FIN DE GRADO

“Operadores de reconstrucción y de subdivisión para la generación de métodos numéricos. Aplicación al mundo Naval”.

ARQUITECTURA NAVAL E INGENIERIA EN SISTEMAS MARINOS

Autor: Jose Alfredo Costa Martínez
Director: Dr. Juan Carlos Trillo Moya

AGRADECIMIENTOS

Agradecer a mis padres PEPE y Mª ASCENSION y a mi hermana LAURA por todo su apoyo y ánimo para superar todos los obstáculos surgidos, por su confianza, su esfuerzo, por hacerme posible llegar hasta aquí.

Agradecer a mi pareja ARANTZATZU, por ser una persona tan especial, por su apoyo incondicional, por estar tantos años a mi lado y soportar la distancia.

Agradecer también a mi amigo JOSÉ GINÉS por su apoyo, sus ánimos y por hacerme una estancia lejos de mi casa llevadera y agradable, así como también a GUILLERMO por los momentos de estudios compartidos.

Agradecer a mi tutor JUAN CARLOS TRILLO MOYA, por brindare la oportunidad de realizar este trabajo y recibirme siempre sin problemas dispuesto a aclararme cualquier duda.

ÍNDICE

ÍNDICE FIGURAS

ÍNDICE TABLAS

1. INTRODUCCIÓN

El proyecto que aquí se inicia consiste en la realización y desarrollo de programas Matlab que nos permitan la representación de curvas y superficies, para con ello, poder dibujar y diseñar partes de un buque y otros sistemas de éste tales como el casco, palas de la hélice, etc.

Desde hace un tiempo, el ámbito del diseño y la construcción naval ha ido avanzando debido al desarrollo de la tecnología, modernizándose, complementando, e incluso sustituyendo los antiguos métodos de diseño, los cuales requerían de muchas horas de trabajo y eran bastante laboriosos de manejar para lograr curvas suaves sin puntos angulosos que describiesen superficies correctamente alisadas.

Cuando realizamos un plano de formas las curvas que se representan deben caracterizarse por ser curvas sin alteraciones, para conseguir que la carena del buque sea una superficie lo más lisa posible de forma que se reduzca la resistencia al avance del buque, lo que se traduce en una disminución en el consumo de combustible, así como en un aumento de la velocidad.

Estas curvas y superficies son de gran interés y tienen alta repercusión en el comportamiento hidrodinámico de un buque, por lo que nos centraremos en su estudio y su comprensión. Gracias a éstas podremos también elaborar planos y superficies que serán objeto de posteriores análisis estructurales, hidrostáticos, etc. También podremos utilizar estas curvas y superficies en las operaciones de reconstrucción facilitando así estos procesos y reduciendo los tiempos de estos procedimientos.

Los casos prácticos en los cuales aplicaremos este tipo de curvas y superficies serán los siguientes:

- Desarrollaremos programas Matlab para la representación de estas curvas y estas superficies.
- Reconstruiremos la sección de un buque en función de unos puntos de control dados.
- Construiremos la vista frontal de la pala de una hélice mediante Splines cúbicos interpolantes, así como mediante NURBS comparando los dos resultados obtenidos.
- Representaremos el recorrido a seguir por un contenedor para su descarga desde el buque al muelle mediante una grúa.
- Representaremos la pala de una hélice en 3D usando superficies NURBS.

2. CONCEPTOS NAVALES BÁSICOS [8, 11, 12, 14]

A continuación describiremos las principales dimensiones del buque, líneas de referencia y otras características, así como la representación gráfica de la hélice.

2.1 DIMENSIONES PRINCIPALES DEL BUQUE

ESLORA

Antes de definir la dimensión de la eslora tenemos que tener en cuenta el siguiente aspecto, y es que para una obra viva fija (zona del casco del buque que va inmersa en el agua) podemos tener encima diferentes y múltiples obras muertas, por lo que, la eslora no puede referirse a la obra muerta ya que no representaría ningún dato en concreto.

A continuación, describiremos las diferentes esloras que nos podemos encontrar:

Eslora total: longitud medida horizontalmente y paralela a la flotación de proyecto, entre las perpendiculares a la misma que pasan por los puntos más sobresalientes de proa y popa.

Eslora entre perpendiculares: longitud medida horizontalmente y paralela a la flotación entre perpendiculares de proa y popa.

- Perpendicular de popa (Ppp): perpendicular a la flotación de proyecto que pasa por el centro del eje del timón o mecha del timón.
- Perpendicular de proa (Ppr): perpendicular a la flotación de proyecto que pasa por la cara exterior o interior de la roda, según sea su estructura de acero fundido o armada respectivamente.

Eslora en la flotación: longitud medida horizontalmente y paralela a la flotación, entre las perpendiculares a la misma que pasan por las intersecciones del casco con la flotación en proa y en popa. La diferencia con respecto a la de perpendiculares esta únicamente en el punto de referencia en la popa.

Eslora de reglamento: puede ser cualquiera de las anteriores dependiendo del Reglamento o bien tiene una definición especial. Por ejemplo, el reglamento de Francobordo, indica que será la mayor longitud de entre, la eslora entre perpendiculares en la flotación al 85% del puntal mínimo de trazado o el 96% de la eslora total en esa misma flotación.

MANGA

Es la dimensión transversal del buque medida horizontalmente entre las perpendiculares a la flotación de proyecto. Tiene su valor mínimo en la proa y aumenta conforme nos acercamos a la Sección Maestra donde adquiere su máximo valor, volviendo a disminuir conforme nos acercamos a la popa.

La manga de un buque estará referida a la SECCIÓN o CUADERNA MAESTRA, que es la de mayor área transversal.

Manga máxima: longitud medida horizontalmente entre planos paralelos al plano de crujía, perpendiculares a la flotación de proyecto, y tangente a los costados del buque.

Manga de trazado o fuera de miembro: longitud medida horizontalmente entre las perpendiculares a la flotación de proyecto que pasan por las caras internas del forro del costado o cantos exteriores de las cuadernas.

Manga fuera forros: longitud medida horizontalmente entre las caras exteriores del forro del costado, es por tanto, la manga máxima.

Manga de flotación: manga de trazado en el plano de una flotación, dependiendo por tanto de la flotación que estemos considerando.

Manga de reglamento: es en general la de trazado, dependiendo no obstante del reglamento que estemos considerando.

La manga es la dimensión con mayor influencia sobre la magnitud del radio metacéntrico transversal y por tanto en la estabilidad inicial.

PUNTAL

Es la altura del buque medida en la sección transversal correspondiente a la Cuaderna Maestra.

Tenemos:

Puntal de trazado: altura del casco desde la cara alta de la quilla o canto bajo de la varenga (elemento estructural de fondo perpendicular a la quilla) en el centro, hasta el canto inferior de la cubierta al costado.

Puntal de sección: es el puntal de trazado de cada sección del buque.

Puntal de reglamento: generalmente es el puntal de trazado, en la cuaderna maestra o en la Sección Media y hasta la cubierta principal. Sin embargo en cada reglamento puede disponer de definición propia.

CALADO

Es la altura o profundidad de la carena, medida desde el punto más bajo del buque hasta los planos de las flotaciones, por lo que cada flotación tendrá su calado.

Podemos considerar los siguientes calados:

Calado de trazado: es la distancia vertical de trazado de la parte sumergida del casco del buque por debajo de la flotación de trazado o proyecto, medida en la sección media.

Calado en una flotación o calado real en esa flotación: es el calado medido desde la cara inferior o exterior de la quilla hasta el nivel de la flotación correspondiente.

El calado en proa se representa con Tpr y el calado a popa se representa con Tpp y hacen referencia a los calados reales del buque en las perpendiculares de proa Ppr y popa Ppp respectivamente. Éstos se trazan en las intersecciones de la línea de la flotación de proyecto con la roda y el codaste.

2.2 PLANOS Y LÍNEAS DE REFERENCIA

Se denomina casco del buque al conjunto estructuras del mismo formado por el forro exterior estanco y los refuerzos sobre los que se apoya. El casco de un buque se puede cortar según un conjunto de tres planos perpendiculares entre sí, paralelos a los planos de un triedro.

Estos planos son los siguientes:
-Plano de crujía: es el plano de simetría del barco en sentido longitudinal. La intersección de este plano con el casco se llama línea de crujía. Los planos paralelos al de crujía que cortan al casco del buque se llaman planos longitudinales, y a las líneas de corte de los mismos con el casco, se llaman longitudinales.

-Plano de flotación: es el plano perpendicular al de crujía que representa la superficie del agua sin oleaje. El plano de flotación de trazado, es el situado al calado de trazado o proyecto del buque. La intersección de este plano con el casco se llama línea de flotación de trazado. Las intersecciones de planos paralelos al de flotación con el casco se llaman líneas de agua. Se llama plano base al plano paralelo a la flotación de trazado que pasa por el canto superior de la quilla en la sección media. A la intersección del plano base con el de crujía se le denomina línea base.

-Plano transversal: es un plano perpendicular a los dos anteriores. Las intersecciones de planos transversales con el casco se denominan cuadernas de trazado o secciones. El conjunto de cuadernas que representan las formas de un buque se llama caja de cuadernas.

Con las líneas obtenidas de estos tres tipos de planos, es decir, las cuadernas, las líneas de agua y longitudinales, se forma el denominado plano de formas, el cual como su propio nombre indica representa las formas del casco del buque.

2.3 OTRAS CARACTERÍSTICAS DEL BUQUE

Las dimensiones definidas anteriormente nos pueden dar una idea errónea respecto a la forma del buque, para ello vamos a especificar otras características para definir prácticamente la "viga-casco":

Astilla muerta: altura en el costado del buque, que hay desde un plano paralelo a la flotación y que pasa por la quilla (plano base o línea base), y el plano tangente al fondo. La astilla muerta principal se mide en la Cuaderna Maestra

Asiento: es la inclinación que tiene la quilla respecto a la flotación, midiéndose por la diferencia de valores verticales.

Trimado: es la diferencia de calados entre la popa y la proa, que se le da al buque, para que navegue en las mejores condiciones.

Pantoque: es la superficie curva que une los costados y el fondo del buque.

Arrufo y brusca: las cubiertas del buque tienen formas específicas con doble curvatura denominándose arrufo a la que tiene en sentido longitudinal y brusca a la de sentido transversal.

El arrufo se mide en el costado del buque y es la distancia desde la cubierta hasta una línea de referencia trazada paralelamente a la flotación, por el punto más bajo de la cubierta, en el centro del buque.

La brusca se mide desde el punto medio del canto superior del bao (elemento de refuerzo de las cubiertas en sentido transversal), sobre una línea recta paralela a la flotación y que pasa por los extremos de dicho bao.

2.4 REPRESENTACIÓN GRÁFICA DE LA HÉLICE

La representación gráfica de las hélices se realiza dibujando una vista lateral y una desde la popa, incluyendo el núcleo. En el mismo se representa la forma de los distintos perfiles que constituyen la pala, así como el paso que tiene cada línea hélice en sus distintos radios, lo que se denomina ley de pasos.

A continuación, comentaremos algunas particularidades de cada una de las proyecciones de la hélice.

- Proyección lateral: consta de la proyección de la pala sobre un plano longitudinal, vista desde estribor, y de un corte ficticio dado a la pala por los puntos de en los que el espesor en cada radio fuera el máximo. Este corte se denomina ley de espesores. En esta proyección podemos medir el ángulo de lanzamiento.

- Proyección frontal: es una vista desde popa y en la que se representan la proyección transversal de la pala y del núcleo. También se pueden unir los puntos de máximo espesor en cada sección y que constituyen la llamada ley de espesores. La distancia entre la punta de la pala y la generatriz se llama divergencia. También se representa en esta vista el llamado contorno desarrollado de la pala que corresponde a la superficie que se obtendría si la cara de presión del helicoide estuviese hecha de una hoja fina y flexible pero inextensible y que pudiera estirarse sobre un papel.

- Perfiles expandidos: los perfiles obtenidos de la intersección de sucesivos cilindros con la pala se presentan expandidos, es decir, estirados sobre el plano en la siguiente vista. Cada perfil se dibuja sobre su radio correspondiente y la línea que une los extremos de las secciones se denomina contorno expandido de la pala.

3. POLINOMIOS DE BERSNTEIN [1, 15]

Básicamente los polinomios de Bernstein permiten aproximar una función continua f(x) definida en un intervalo [a, b], así como también ajustar curvas o superficies a un conjunto de datos. En este último caso entenderemos por ajuste a aquella función que se acerca a la función verdadera sin que reproduzca el conjunto de datos de forma exacta, es decir, la gráfica de la función que aproxima pasará cerca de los datos, pero no necesariamente sobre ellos.

El nombre de estos polinomios hace referencia al matemático Ucraniano Sergei Natanovich Bernstein. Son la base primordial de la demostración que Bernstein hizo del Teorema de aproximación de Weierstrass. El fondo de esta prueba es la construcción de una sucesión de polinomios que converjan de manera uniforme a una función continua. La demostración que hizo Bernstein apareció en 1912 y ha tenido un fuerte impacto en distintas áreas. Los polinomios de Bernstein están conectados por ejemplo con la teoría de probabilidad, problemas sobre momentos, con la teoría de sumas de series, etc. Así mismo los polinomios de Bernstein históricamente dieron lugar a las curvas de Bézier que estudiaremos en el apartado 4.

3.1 DEFINICIÓN

Un polinomio de Bernstein de grado n asociado con la función $f : [a, b] \rightarrow \mathbb{R}$ está definido por:

$$B_n(f; x) = \sum_{i=0}^{n} \binom{n}{i} \frac{(x-a)^i}{(b-a)^i} \frac{(b-x)^{n-i}}{(b-a)^{n-i}} f(x_i), \qquad (3.1)$$

donde

$$x_i = a + ih = a + \frac{i}{n}(b-a), \qquad i = 0,1,\dots,n.$$

En el caso especial con intervalo [0,1], la ecuación anterior se reduce a:

$$B_n(f; x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} f\left(\frac{i}{n}\right).$$

Los polinomios básicos de Bernstein se definen como:

$$B_{n,i}(x) = \binom{n}{i} x^i (1-x)^{n-i},$$

con

$$i = 0,1,\dots,n; \qquad n = 0,1,\dots$$

A continuación, se muestra la programación en Matlab que hemos realizado para construir los polinomios de Bernstein.

```
function [P]=Bernstein_polynomials(interval,n)

% This function computes the Bernstein polynomials Bni(x) in a given
% interval [a,b]. The different n+1 polynomials are plotted.
%
% [P]=Bernstein_polynomials(interval,n,dx)
% Input variables:
% interval contains the working interval [a,b]
% n degree of the desired Bernstein polynomials
%
% Output variables:
% P cell containing the expressions of the different Bernstein
polynomials
%   of degree n
%
% Example:
% [P]=Bernstein_polynomials([0,1],4)

% We declare the symbolic variable x

syms x

% We define the interval

a=interval(1);
b=interval(2);

% we calculate the values of the Bernstein polynomials at the
abscissae x

for i=0:n
    P{i+1}=sym2poly(expand(combinatorio(n,i)*(x-a)^(i)*(b-x)^(n-i)));
end


%%%%%%%%%%%%%%
% Plots
%%%%%%%%%%%%%%


mx=a; Mx=b;
my=-0.1; My=0;


for i=0:n

% we define a grid
    t=linspace(a,b,100);

% we evaluate the Bernstein polynomials at t
    pt=polyval(double(P{i+1}),t);

% we plot the Bernstein polynomials

    h(i+1)=plot(t,pt,'-','LineWidth',3,'Color',[i/n,i/n,0]);
    str{i+1}=['Polinomio de Bernstein',blanks(2),num2str(i)];
```

```
    hold on

% we adjust the limits of the plot

    my_new=min(pt);
    My_new=max(pt);


if my_new<my
        my=my_new;
end
if My_new>My
        My=My_new;
end

    dx=0.1*(Mx-mx);
    dy=0.1*(My-my);

    axis([mx-dx Mx+dx my-dy My+dy]);


    legend(h,str);
end

Bernstein=P{1};

for i=2:n+1
    Bernstein=[Bernstein; P{i}];

end
Bernstein
```

Ejemplos:

Ejemplo1

En este ejemplo representaremos los polinomios básicos de Bernstein de grado 4 en el intervalo de 0 a 1

>> [P]=Bernstein_polynomials([0,1],4)

Bernstein =

```
   1   -4    6   -4    1
  -4   12  -12    4    0
   6  -12    6    0    0
  -4    4    0    0    0
   1    0    0    0    0
```

El la figura 3.1 vemos la representación gráfica de los cinco polinomios básicos de Bernstein de grado cuatro cuyos coeficientes nos los ha proporcionado el programa Bernstein Polynomials.

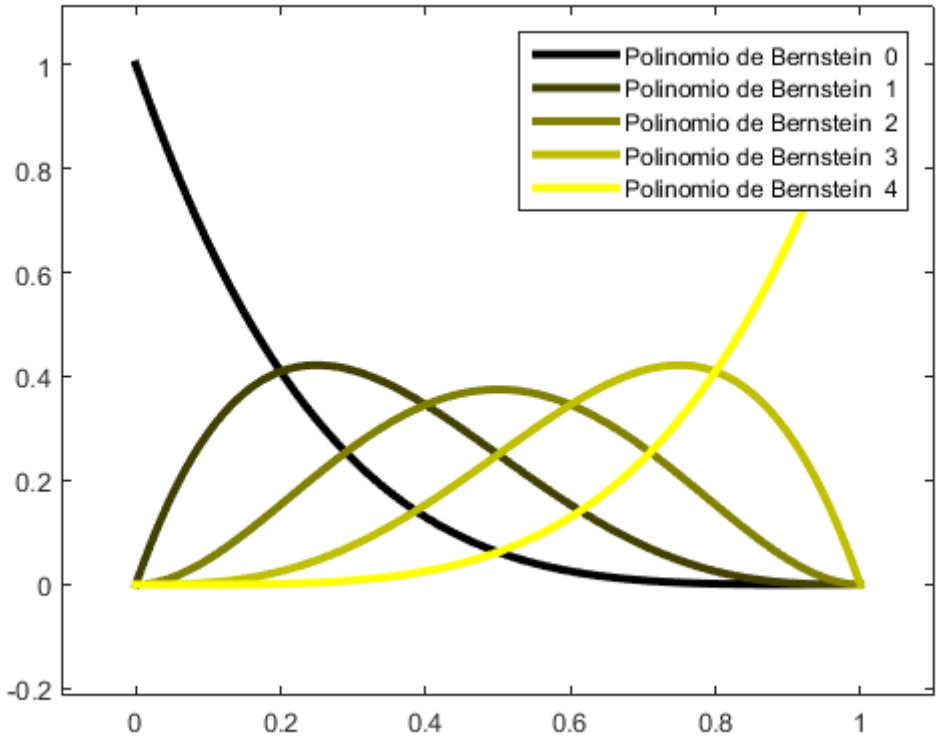


Fig3.1: Polinomios básicos de Bernstein de grado 4.

3.2 TEOREMA DE WEIERSTRASS

Podemos denotar f^* a una función que imita el comportamiento de otra función f, es decir, que f^* se encuentra cerca de f. El concepto de imitar necesita que fijemos lo que queremos decir cuando hablamos de proximidad.

El concepto geométrico de longitud de un vector tiene muchas aplicaciones naturales que están conectadas con espacios de funciones y con la teoría de aproximación. Al igual que medimos distancias en el espacio euclidiano $\mathbb{R}^n$ para saber la cercanía de dos elementos en el espacio vectorial, también podemos medir distancias en un espacio de funciones para así poder determinar la cercanía de una función f y su aproximación f^*.

Aquí consideramos un problema de aproximación concreto que planteó y resolvió Weierstrass el 1885. El teorema de Weierstrass asegura que podemos aproximar funciones continuas de manera uniforme a través de polinomios. Para entender esto fijaremos la notación. Llamaremos C $[a, b]$ al espacio de funciones continuas en el intervalo cerrado y acotado [a, b], es decir:

$$C\,[a,b] = \{f\colon [a,b] \to \mathbb{R}, \quad f\ continua\}.$$

En este espacio de funciones consideraremos la siguiente norma:

$$\|f\|_\infty = max_{a\le x\le b}|f(x)|.$$

Con esta definición la distancia entre dos funciones continuas $f(x)\ y\ g(x)$ se puede medir calculando

$$\|f-g\|_\infty = max_{a\le x\le b}|f(x)-g(x)|.$$

TEOREMA 2.1 Sea $f\colon [0,1] \to R$ continua. Para todo $\varepsilon > 0$ existe un polinomio P_ε tal que:

$$|P_\varepsilon(x) - f(x)| < \varepsilon, \ \ \forall\ x\,\epsilon\,[0,1].$$

La prueba de este teorema puede encontrarse en [3].

Según este el teorema 2.1 dado $\varepsilon > 0$ y dada una función continua f en $[0,1]$, existe un polinomio P_ε construido con los polinomios de Bernstein tal que:

$$\|f - P_\varepsilon\|_\infty < \varepsilon.$$

A continuación, se muestra la programación en Matlab que nos permite aproximar una función continua en un intervalo dado [a,b] mediante los polinomios de Bernstein de grados crecientes definidos en (3.1), tal y como indica la prueba del propio teorema de Weierstrass.

```
function
[xf,fxf,Bf,n,distance]=Weierstrass_Theorem(f,interval,nf,n,varargin)

% This function computes the uniform approximation to a given
continuous function in a given
% interval [a,b] making use of the Bernstein polynomials.
%
% [xf,fxf,Bf,n,distance]=Weierstrass_Theorem(f,interval,nf,n,varargin)
% Input variables:
% f continuous function to be approximated in the infinity norm
% interval contains the working interval [a,b]
% nf number of discretization points in the interval [a,b]
% n degree of the desired Bernstein polynomials used to compute the
%   approximation
% varargin contais the following optional input variables:
%    plot_aprox can take the values:
%                                  v plots the approximations for
%                                        degrees v(1) to v(length(v))
%                                        It must be a vector of
increasing
%                                        integer numbers between 0 and
n
%                                  'no'  does not make any plot
%    epsilon determines the allowed distance between the function and
the
%            approximation, and therefore it determines the value of n
```

```
%            It must always be strictly larger than 0
% Output variables:
% xf grid of the interval [a,b]
% fxf values of the function f at the grid xf
% Bf values of the succesive approximations of the function f at the
grid xf
% n larger value of the degree of the approximation
% distance infinity norm of the difference between the discretized
% function and the n-degree approximation
%
% Example1:
% [xf,fxf,Bf,n,distance]=Weierstrass_Theorem('sin(x)',[0,1],100,5,0:5)
% Example2:
%
[xf,fxf,Bf,n,distance]=Weierstrass_Theorem('sin(x)',[0,1],100,5,'no',0
.01)
% Example3:
%[xf,fxf,Bf,n,distance]=Weierstrass_Theorem('sin(x)',[0,1],100,5,0:5,0
.01)



% We declare the symbolic variable x

syms x

% We get the value of the optional input variables

n_aprox_plot=n;
if nargin==5
    n_aprox_plot=varargin{1};
elseif nargin==6
    n_aprox_plot=varargin{1};
    epsilon=varargin{2};
end

% We define the interval

a=interval(1);
b=interval(2);


% We define the grid for evaluation
xf=linspace(a,b,nf);

% We compute the values of f at xf
fxf=double(subs(f,'x',xf));

if ~exist('epsilon')


% If the option of plotting is active, it drawes the original function

if nargin==5 & ~ischar(n_aprox_plot)
        h(1)=plot(xf,fxf,'-b');
        ih=2;
        str{1}='Funcion original';
        hold on
```

```
end

% We define the approximation and inizialize it to zero
    Bf(1:n+1,1:nf)=zeros(n+1,nf);

% We start computing the approximations
for i=0:n

        xi=linspace(a,b,i+1);

% we calculate the succesive approximations for each abscissa xf
        Bf(i+1,1:nf)=double(subs(f,'x',a))*ones(1,nf)/(b-a)^i;
for j=1:i
            Bf(i+1,1:nf)=Bf(i+1,1:nf)+(combinatorio(i,j)*(xf-
a).^(j)).*(b-xf).^(i-j)*double(subs(f,'x',xi(j+1)))/(b-a)^i;
end

%Plot of the succesive approximations

if nargin==5 & ~ischar(n_aprox_plot) & ~isempty(find(n_aprox_plot==i))
            h(ih)=plot(xf,Bf(i+1,1:nf),'Color',[0,(i-
n_aprox_plot(1))/max(n_aprox_plot(end)-n_aprox_plot(1),1),0]);
            str{ih}=['Approximation for n=',blanks(2),num2str(i)];
            ih=ih+1;
end

end

% We insert a legend in the plot, and put the axis correctly
if nargin==5 & ~ischar(n_aprox_plot)
        my=min(min(Bf));
        My=max(max(Bf));
        dy=0.1*(My-my);
        axis([a-0.1*(b-a),b+0.1*(b-a),my-dy,My+dy]);
        legend(h,str)
end


   distance=norm(fxf-Bf(n+1,1:nf),'inf');

else

% If the option of plotting is active, it drawes the original function

if  ~ischar(n_aprox_plot)
            h(1)=plot(xf,fxf,'-b');
            ih=2;
            str{1}='Funcion original';
            hold on
            plot(xf,fxf-epsilon,'r-',xf,fxf+epsilon,'-r');
end

    distance=epsilon+1;

    i=0;

while distance>epsilon
```

```
% We start computing the approximation
        xi=linspace(a,b,i+1);

% we calculate the succesive approximations for each abscissa xf
        Bf(i+1,1:nf)=double(subs(f,'x',a))*(b-xf).^i/(b-a)^i;
for j=1:i
            Bf(i+1,1:nf)=Bf(i+1,1:nf)+(combinatorio(i,j)*(xf-
a).^(j)).*(b-xf).^(i-j)*double(subs(f,'x',xi(j+1)))/(b-a)^i;
end

        distance=norm(fxf-Bf(i+1,1:nf),'inf');
        i=i+1;

end

    n_input=n;
    n=i-1
    distance

%Plot of the last approximations

if ~ischar(n_aprox_plot)
        dn=max(n_aprox_plot(end)-n_aprox_plot(1),1);
        ndibuja=n-n_input;
        dibuja=n_aprox_plot+ndibuja*ones(size(n_aprox_plot));
        [aux,ind]=find(dibuja>0); dibuja=dibuja(ind);
for i=1:length(dibuja)
            h(ih)=plot(xf,Bf(dibuja(i)+1,1:nf),'Color',[0,(dibuja(i)-
dibuja(1))/dn,0]);
            str{ih}=['Approximation for
n=',blanks(2),num2str(dibuja(i))];
            ih=ih+1;
end
        my=min(min(Bf));
        My=max(max(Bf));
        dy=0.1*(My-my);
        axis([a-0.1*(b-a),b+0.1*(b-a),my-dy,My+dy]);
        legend(h,str)
end



end
```

Ejemplos

- Ejemplo 1

Este ejemplo se realizará con la función sin(x) en el intervalo [0,1] con 100 puntos de discretización en este intervalo. Se utilizarán polinomios de Bernstein de grado 5 para el cálculo de aproximación y se mostrarán las funciones desde grado 0 hasta grado 5 (si en vez de poner 0:5 ponemos [0,3] el programa calculará hasta la de grado 5 pero solo nos mostrará las funciones de grado 0 y 3)

>> [xf,fxf,Bf,n,distance]=Weierstrass_Theorem('sin(x)',[0,1],100,5,0:5)

xf =

Columns 1 through 13

0 0.0101 0.0202 0.0303 0.0404 0.0505 0.0606 0.0707 0.0808 0.0909 0.1010 0.1111 0.1212

Columns 14 through 26

0.1313 0.1414 0.1515 0.1616 0.1717 0.1818 0.1919 0.2020 0.2121 0.2222 0.2323 0.2424 0.2525

Columns 27 through 39

0.2626 0.2727 0.2828 0.2929 0.3030 0.3131 0.3232 0.3333 0.3434 0.3535 0.3636 0.3737 0.3838

Columns 40 through 52

0.3939 0.4040 0.4141 0.4242 0.4343 0.4444 0.4545 0.4646 0.4747 0.4848 0.4949 0.5051 0.5152

Columns 53 through 65

0.5253 0.5354 0.5455 0.5556 0.5657 0.5758 0.5859 0.5960 0.6061 0.6162 0.6263 0.6364 0.6465

Columns 66 through 78

0.6566 0.6667 0.6768 0.6869 0.6970 0.7071 0.7172 0.7273 0.7374 0.7475 0.7576 0.7677 0.7778

Columns 79 through 91

0.7879 0.7980 0.8081 0.8182 0.8283 0.8384 0.8485 0.8586 0.8687 0.8788 0.8889 0.8990 0.9091

Columns 92 through 100

0.9192 0.9293 0.9394 0.9495 0.9596 0.9697 0.9798 0.9899 1.0000

fxf =

Columns 1 through 13

0 0.0101 0.0202 0.0303 0.0404 0.0505 0.0606 0.0706 0.0807 0.0908 0.1008 0.1109 0.1209

Columns 14 through 26

0.1309 0.1409 0.1509 0.1609 0.1709 0.1808 0.1907 0.2006 0.2105 0.2204 0.2302 0.2401 0.2498

Columns 27 through 39

0.2596 0.2694 0.2791 0.2888 0.2984 0.3080 0.3176 0.3272 0.3367 0.3462 0.3557 0.3651 0.3745

Columns 40 through 52

0.3838 0.3931 0.4024 0.4116 0.4208 0.4300 0.4391 0.4481 0.4571 0.4661 0.4750 0.4839 0.4927

Columns 53 through 65

0.5014 0.5101 0.5188 0.5274 0.5360 0.5445 0.5529 0.5613 0.5696 0.5779 0.5861 0.5943 0.6024

Columns 66 through 78

0.6104 0.6184 0.6263 0.6341 0.6419 0.6496 0.6573 0.6648 0.6723 0.6798 0.6872 0.6945 0.7017

Columns 79 through 91

0.7089 0.7159 0.7230 0.7299 0.7368 0.7436 0.7503 0.7569 0.7635 0.7700 0.7764 0.7827 0.7889

Columns 92 through 100

0.7951 0.8012 0.8072 0.8131 0.8190 0.8247 0.8304 0.8360 0.8415

Bf =

Columns 1 through 13

0 0 0 0 0 0 0 0 0 0 0 0 0

0 0.0085 0.0170 0.0255 0.0340 0.0425 0.0510 0.0595 0.0680 0.0765 0.0850 0.0935 0.1020

0 0.0097 0.0193 0.0289 0.0385 0.0481 0.0577 0.0672 0.0767 0.0862 0.0957 0.1051 0.1145

0 0.0099 0.0198 0.0296 0.0395 0.0493 0.0591 0.0689 0.0786 0.0883 0.0980 0.1077 0.1173

0 0.0100 0.0200 0.0299 0.0398 0.0497 0.0596 0.0695 0.0793 0.0892 0.0990 0.1087 0.1185

0 0.0100 0.0200 0.0300 0.0400 0.0500 0.0599 0.0698 0.0797 0.0896 0.0995 0.1093 0.1191

Columns 14 through 26

0 0 0 0 0 0 0 0 0 0 0 0 0

0.1105 0.1190 0.1275 0.1360 0.1445 0.1530 0.1615 0.1700 0.1785 0.1870 0.1955 0.2040 0.2125

0.1239 0.1332 0.1426 0.1519 0.1612 0.1705 0.1797 0.1889 0.1981 0.2073 0.2164 0.2256 0.2346

0.1270 0.1366 0.1461 0.1557 0.1652 0.1747 0.1842 0.1936 0.2030 0.2124 0.2218 0.2312 0.2405

0.1282 0.1379 0.1476 0.1573 0.1669 0.1765 0.1861 0.1957 0.2052 0.2147 0.2242 0.2337 0.2431

0.1289 0.1387 0.1484 0.1582 0.1679 0.1775 0.1872 0.1968 0.2064 0.2160 0.2256 0.2351 0.2446

Columns 27 through 39

0 0 0 0 0 0 0 0 0 0 0 0 0

0.2210 0.2295 0.2380 0.2465 0.2550 0.2635 0.2720 0.2805 0.2890 0.2975 0.3060 0.3145 0.3230

0.2437 0.2528 0.2618 0.2708 0.2798 0.2887 0.2977 0.3066 0.3155 0.3243 0.3332 0.3420 0.3508

0.2498 0.2590 0.2683 0.2775 0.2866 0.2958 0.3049 0.3140 0.3231 0.3321 0.3411 0.3501 0.3590

0.2525 0.2619 0.2712 0.2805 0.2898 0.2991 0.3083 0.3175 0.3267 0.3358 0.3449 0.3540 0.3630

0.2541 0.2635 0.2729 0.2823 0.2917 0.3010 0.3103 0.3195 0.3288 0.3380 0.3471 0.3563 0.3654

Columns 40 through 52

0 0 0 0 0 0 0 0 0 0 0 0 0

0.3315 0.3400 0.3485 0.3570 0.3655 0.3740 0.3825 0.3910 0.3995 0.4080 0.4165 0.4250 0.4335

0.3595 0.3683 0.3770 0.3857 0.3943 0.4030 0.4116 0.4202 0.4288 0.4373 0.4458 0.4543 0.4628

0.3680 0.3768 0.3857 0.3945 0.4033 0.4121 0.4208 0.4295 0.4382 0.4469 0.4555 0.4641 0.4726

0.3720 0.3810 0.3900 0.3989 0.4078 0.4166 0.4254 0.4342 0.4429 0.4516 0.4603 0.4690 0.4776

0.3745 0.3835 0.3925 0.4015 0.4104 0.4193 0.4282 0.4370 0.4458 0.4545 0.4632 0.4719 0.4805

Columns 53 through 65

0 0 0 0 0 0 0 0 0 0 0 0 0

0.4420 0.4505 0.4590 0.4675 0.4760 0.4845 0.4930 0.5015 0.5100 0.5185 0.5270 0.5355 0.5440

0.4713 0.4797 0.4881 0.4965 0.5048 0.5132 0.5215 0.5297 0.5380 0.5462 0.5545 0.5626 0.5708

0.4811 0.4896 0.4981 0.5065 0.5149 0.5232 0.5315 0.5398 0.5481 0.5563 0.5645 0.5726 0.5807

0.4861 0.4946 0.5031 0.5116 0.5200 0.5284 0.5367 0.5450 0.5533 0.5615 0.5697 0.5778 0.5859

0.4891 0.4977 0.5062 0.5147 0.5231 0.5315 0.5399 0.5482 0.5564 0.5647 0.5729 0.5810 0.5891

Columns 66 through 78

0 0 0 0 0 0 0 0 0 0 0 0 0

0.5525 0.5610 0.5695 0.5780 0.5865 0.5950 0.6035 0.6120 0.6205 0.6290 0.6375 0.6460 0.6545

0.5789 0.5871 0.5952 0.6032 0.6113 0.6193 0.6273 0.6353 0.6432 0.6511 0.6590 0.6669 0.6748

0.5888 0.5969 0.6049 0.6129 0.6208 0.6287 0.6366 0.6444 0.6522 0.6599 0.6677 0.6754 0.6830

0.5940 0.6020 0.6100 0.6179 0.6258 0.6337 0.6415 0.6492 0.6569 0.6646 0.6723 0.6799 0.6874

0.5971 0.6051 0.6131 0.6210 0.6289 0.6367 0.6445 0.6522 0.6599 0.6675 0.6751 0.6826 0.6901

Columns 79 through 91

0 0 0 0 0 0 0 0 0 0 0 0 0

0.6630 0.6715 0.6800 0.6885 0.6970 0.7055 0.7140 0.7225 0.7310 0.7395 0.7480 0.7565 0.7650

0.6826 0.6904 0.6982 0.7059 0.7137 0.7214 0.7291 0.7367 0.7444 0.7520 0.7596 0.7671 0.7747

0.6906 0.6982 0.7057 0.7132 0.7207 0.7281 0.7355 0.7428 0.7501 0.7574 0.7646 0.7718 0.7790

0.6949 0.7024 0.7098 0.7171 0.7244 0.7317 0.7389 0.7461 0.7532 0.7603 0.7674 0.7743 0.7813

0.6976 0.7049 0.7123 0.7196 0.7268 0.7340 0.7411 0.7482 0.7552 0.7622 0.7691 0.7759 0.7827

Columns 92 through 100

0 0 0 0 0 0 0 0 0

0.7735 0.7820 0.7905 0.7990 0.8075 0.8160 0.8245 0.8330 0.8415

0.7822 0.7897 0.7972 0.8046 0.8120 0.8194 0.8268 0.8341 0.8415

0.7861 0.7931 0.8002 0.8072 0.8141 0.8210 0.8279 0.8347 0.8415

0.7882 0.7950 0.8018 0.8085 0.8152 0.8219 0.8285 0.8350 0.8415

0.7895 0.7962 0.8028 0.8094 0.8159 0.8224 0.8288 0.8352 0.8415

n =

5

distance =

0.0133

Finalmente, el programa nos devuelve tanto los valores de la función en el mallado utilizado como una representación gráfica de la función original y de las distintas funciones que ha ido aproximando dicha función original. Así mismo se calcula la distancia entre la función original y la aproximación mediante polinomios de Bernstein de grado 5. En la figura 3.2.1 se ve tanto la función original así como las distintas aproximaciones realizadas.

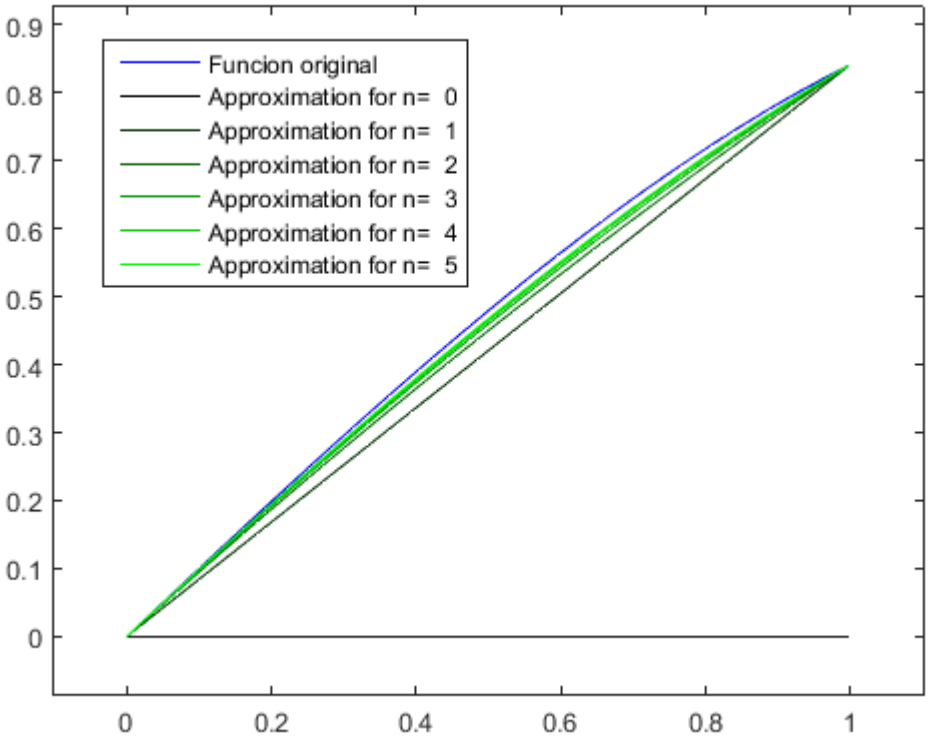


Fig 3.2.1: aproximación polinomios de grado 5.

Como podemos ver en la representación gráfica según crece el grado de la función más se van aproximando los polinomios de Bernstein a la función original.

-Ejemplo 2

En este caso consideramos el mismo escenario que en el caso anterior con la diferencia de que nosotros fijamos la distancia máxima que puede haber entre la función original y la que se aproxima. Esta distancia máxima permitida servirá de criterio de parada para el programa, que se detendrá cuando considere polinomios de Bernstein de grado n con n lo suficientemente grande para que la distancia de aproximación a la función original sea menor que dicha distancia fijada. Hemos establecido que dicha distancia sea 0.01.

```
>> [xf,fxf,Bf,n,distance]=Weierstrass_Theorem('sin(x)',[0,1],100,5,'no',0.01)

n =

     7


distance =

    0.0096


xf =

  Columns 1 through 6

         0    0.0101    0.0202    0.0303    0.0404    0.0505

  Columns 7 through 12

    0.0606    0.0707    0.0808    0.0909    0.1010    0.1111

  Columns 13 through 18

    0.1212    0.1313    0.1414    0.1515    0.1616    0.1717

  Columns 19 through 24

    0.1818    0.1919    0.2020    0.2121    0.2222    0.2323
```

Columns 25 through 30

0.2424 0.2525 0.2626 0.2727 0.2828 0.2929

Columns 31 through 36

0.3030 0.3131 0.3232 0.3333 0.3434 0.3535

Columns 37 through 42

0.3636 0.3737 0.3838 0.3939 0.4040 0.4141

Columns 43 through 48

0.4242 0.4343 0.4444 0.4545 0.4646 0.4747

Columns 49 through 54

0.4848 0.4949 0.5051 0.5152 0.5253 0.5354

Columns 55 through 60

0.5455 0.5556 0.5657 0.5758 0.5859 0.5960

Columns 61 through 66

0.6061 0.6162 0.6263 0.6364 0.6465 0.6566

Columns 67 through 72

0.6667 0.6768 0.6869 0.6970 0.7071 0.7172

Columns 73 through 78

0.7273 0.7374 0.7475 0.7576 0.7677 0.7778

Columns 79 through 84

0.7879 0.7980 0.8081 0.8182 0.8283 0.8384

Columns 85 through 90

0.8485 0.8586 0.8687 0.8788 0.8889 0.8990

Columns 91 through 96

0.9091 0.9192 0.9293 0.9394 0.9495 0.9596

Columns 97 through 100

0.9697 0.9798 0.9899 1.0000

fxf =

Columns 1 through 6

0 0.0101 0.0202 0.0303 0.0404 0.0505

Columns 7 through 12

0.0606 0.0706 0.0807 0.0908 0.1008 0.1109

Columns 13 through 18

0.1209 0.1309 0.1409 0.1509 0.1609 0.1709

Columns 19 through 24

0.1808 0.1907 0.2006 0.2105 0.2204 0.2302

Columns 25 through 30

0.2401 0.2498 0.2596 0.2694 0.2791 0.2888

Columns 31 through 36

0.2984 0.3080 0.3176 0.3272 0.3367 0.3462

Columns 37 through 42

0.3557 0.3651 0.3745 0.3838 0.3931 0.4024

Columns 43 through 48

0.4116 0.4208 0.4300 0.4391 0.4481 0.4571

Columns 49 through 54

0.4661 0.4750 0.4839 0.4927 0.5014 0.5101

Columns 55 through 60

0.5188 0.5274 0.5360 0.5445 0.5529 0.5613

Columns 61 through 66

0.5696 0.5779 0.5861 0.5943 0.6024 0.6104

Columns 67 through 72

0.6184 0.6263 0.6341 0.6419 0.6496 0.6573

Columns 73 through 78

0.6648 0.6723 0.6798 0.6872 0.6945 0.7017

Columns 79 through 84

0.7089 0.7159 0.7230 0.7299 0.7368 0.7436

Columns 85 through 90

0.7503 0.7569 0.7635 0.7700 0.7764 0.7827

Columns 91 through 96

0.7889 0.7951 0.8012 0.8072 0.8131 0.8190

Columns 97 through 100

0.8247 0.8304 0.8360 0.8415

Bf =

Columns 1 through 6

0	0	0	0	0	0
0	0.0085	0.0170	0.0255	0.0340	0.0425
0	0.0097	0.0193	0.0289	0.0385	0.0481
0	0.0099	0.0198	0.0296	0.0395	0.0493
0	0.0100	0.0200	0.0299	0.0398	0.0497
0	0.0100	0.0200	0.0300	0.0400	0.0500
0	0.0100	0.0201	0.0301	0.0401	0.0501
0	0.0101	0.0201	0.0301	0.0402	0.0502

Columns 7 through 12

0	0	0	0	0	0
0.0510	0.0595	0.0680	0.0765	0.0850	0.0935
0.0577	0.0672	0.0767	0.0862	0.0957	0.1051
0.0591	0.0689	0.0786	0.0883	0.0980	0.1077
0.0596	0.0695	0.0793	0.0892	0.0990	0.1087
0.0599	0.0698	0.0797	0.0896	0.0995	0.1093
0.0601	0.0700	0.0799	0.0899	0.0997	0.1096
0.0602	0.0701	0.0801	0.0900	0.0999	0.1098

Columns 13 through 18

0	0	0	0	0	0
0.1020	0.1105	0.1190	0.1275	0.1360	0.1445
0.1145	0.1239	0.1332	0.1426	0.1519	0.1612
0.1173	0.1270	0.1366	0.1461	0.1557	0.1652
0.1185	0.1282	0.1379	0.1476	0.1573	0.1669

0.1191	0.1289	0.1387	0.1484	0.1582	0.1679
0.1195	0.1293	0.1391	0.1489	0.1587	0.1684
0.1197	0.1296	0.1394	0.1493	0.1591	0.1688

Columns 19 through 24

0	0	0	0	0	0
0.1530	0.1615	0.1700	0.1785	0.1870	0.1955
0.1705	0.1797	0.1889	0.1981	0.2073	0.2164
0.1747	0.1842	0.1936	0.2030	0.2124	0.2218
0.1765	0.1861	0.1957	0.2052	0.2147	0.2242
0.1775	0.1872	0.1968	0.2064	0.2160	0.2256
0.1782	0.1879	0.1975	0.2072	0.2168	0.2264
0.1786	0.1883	0.1980	0.2077	0.2174	0.2270

Columns 25 through 30

0	0	0	0	0	0
0.2040	0.2125	0.2210	0.2295	0.2380	0.2465
0.2256	0.2346	0.2437	0.2528	0.2618	0.2708
0.2312	0.2405	0.2498	0.2590	0.2683	0.2775
0.2337	0.2431	0.2525	0.2619	0.2712	0.2805
0.2351	0.2446	0.2541	0.2635	0.2729	0.2823
0.2360	0.2455	0.2551	0.2645	0.2740	0.2834
0.2366	0.2462	0.2558	0.2653	0.2748	0.2842

Columns 31 through 36

0	0	0	0	0	0
0.2550	0.2635	0.2720	0.2805	0.2890	0.2975
0.2798	0.2887	0.2977	0.3066	0.3155	0.3243

0.2866	0.2958	0.3049	0.3140	0.3231	0.3321
0.2898	0.2991	0.3083	0.3175	0.3267	0.3358
0.2917	0.3010	0.3103	0.3195	0.3288	0.3380
0.2928	0.3022	0.3116	0.3209	0.3301	0.3394
0.2937	0.3031	0.3125	0.3218	0.3311	0.3404

Columns 37 through 42

0	0	0	0	0	0
0.3060	0.3145	0.3230	0.3315	0.3400	0.3485
0.3332	0.3420	0.3508	0.3595	0.3683	0.3770
0.3411	0.3501	0.3590	0.3680	0.3768	0.3857
0.3449	0.3540	0.3630	0.3720	0.3810	0.3900
0.3471	0.3563	0.3654	0.3745	0.3835	0.3925
0.3486	0.3578	0.3669	0.3760	0.3851	0.3942
0.3496	0.3589	0.3680	0.3772	0.3863	0.3954

Columns 43 through 48

0	0	0	0	0	0
0.3570	0.3655	0.3740	0.3825	0.3910	0.3995
0.3857	0.3943	0.4030	0.4116	0.4202	0.4288
0.3945	0.4033	0.4121	0.4208	0.4295	0.4382
0.3989	0.4078	0.4166	0.4254	0.4342	0.4429
0.4015	0.4104	0.4193	0.4282	0.4370	0.4458
0.4032	0.4121	0.4211	0.4300	0.4388	0.4477
0.4044	0.4134	0.4224	0.4313	0.4402	0.4490

Columns 49 through 54

0	0	0	0	0	0

0.4080	0.4165	0.4250	0.4335	0.4420	0.4505
0.4373	0.4458	0.4543	0.4628	0.4713	0.4797
0.4469	0.4555	0.4641	0.4726	0.4811	0.4896
0.4516	0.4603	0.4690	0.4776	0.4861	0.4946
0.4545	0.4632	0.4719	0.4805	0.4891	0.4977
0.4564	0.4652	0.4739	0.4825	0.4912	0.4997
0.4578	0.4666	0.4753	0.4840	0.4926	0.5012

Columns 55 through 60

0	0	0	0	0	0
0.4590	0.4675	0.4760	0.4845	0.4930	0.5015
0.4881	0.4965	0.5048	0.5132	0.5215	0.5297
0.4981	0.5065	0.5149	0.5232	0.5315	0.5398
0.5031	0.5116	0.5200	0.5284	0.5367	0.5450
0.5062	0.5147	0.5231	0.5315	0.5399	0.5482
0.5083	0.5168	0.5252	0.5336	0.5420	0.5503
0.5098	0.5183	0.5267	0.5351	0.5435	0.5519

Columns 61 through 66

0	0	0	0	0	0
0.5100	0.5185	0.5270	0.5355	0.5440	0.5525
0.5380	0.5462	0.5545	0.5626	0.5708	0.5789
0.5481	0.5563	0.5645	0.5726	0.5807	0.5888
0.5533	0.5615	0.5697	0.5778	0.5859	0.5940
0.5564	0.5647	0.5729	0.5810	0.5891	0.5971
0.5586	0.5668	0.5750	0.5831	0.5912	0.5993
0.5601	0.5684	0.5766	0.5847	0.5928	0.6008

Columns 67 through 72

0	0	0	0	0	0
0.5610	0.5695	0.5780	0.5865	0.5950	0.6035
0.5871	0.5952	0.6032	0.6113	0.6193	0.6273
0.5969	0.6049	0.6129	0.6208	0.6287	0.6366
0.6020	0.6100	0.6179	0.6258	0.6337	0.6415
0.6051	0.6131	0.6210	0.6289	0.6367	0.6445
0.6073	0.6152	0.6231	0.6310	0.6388	0.6465
0.6088	0.6168	0.6247	0.6325	0.6403	0.6480

Columns 73 through 78

0	0	0	0	0	0
0.6120	0.6205	0.6290	0.6375	0.6460	0.6545
0.6353	0.6432	0.6511	0.6590	0.6669	0.6748
0.6444	0.6522	0.6599	0.6677	0.6754	0.6830
0.6492	0.6569	0.6646	0.6723	0.6799	0.6874
0.6522	0.6599	0.6675	0.6751	0.6826	0.6901
0.6542	0.6619	0.6695	0.6770	0.6845	0.6920
0.6557	0.6633	0.6709	0.6784	0.6859	0.6933

Columns 79 through 84

0	0	0	0	0	0
0.6630	0.6715	0.6800	0.6885	0.6970	0.7055
0.6826	0.6904	0.6982	0.7059	0.7137	0.7214
0.6906	0.6982	0.7057	0.7132	0.7207	0.7281
0.6949	0.7024	0.7098	0.7171	0.7244	0.7317
0.6976	0.7049	0.7123	0.7196	0.7268	0.7340
0.6994	0.7067	0.7140	0.7212	0.7284	0.7355
0.7007	0.7080	0.7152	0.7224	0.7295	0.7366

Columns 85 through 90

0	0	0	0	0	0
0.7140	0.7225	0.7310	0.7395	0.7480	0.7565
0.7291	0.7367	0.7444	0.7520	0.7596	0.7671
0.7355	0.7428	0.7501	0.7574	0.7646	0.7718
0.7389	0.7461	0.7532	0.7603	0.7674	0.7743
0.7411	0.7482	0.7552	0.7622	0.7691	0.7759
0.7426	0.7496	0.7565	0.7634	0.7702	0.7770
0.7436	0.7506	0.7575	0.7643	0.7711	0.7778

Columns 91 through 96

0	0	0	0	0	0
0.7650	0.7735	0.7820	0.7905	0.7990	0.8075
0.7747	0.7822	0.7897	0.7972	0.8046	0.8120
0.7790	0.7861	0.7931	0.8002	0.8072	0.8141
0.7813	0.7882	0.7950	0.8018	0.8085	0.8152
0.7827	0.7895	0.7962	0.8028	0.8094	0.8159
0.7837	0.7904	0.7970	0.8035	0.8100	0.8164
0.7844	0.7910	0.7976	0.8040	0.8104	0.8168

Columns 97 through 100

0	0	0	0
0.8160	0.8245	0.8330	0.8415
0.8194	0.8268	0.8341	0.8415
0.8210	0.8279	0.8347	0.8415
0.8219	0.8285	0.8350	0.8415
0.8224	0.8288	0.8352	0.8415

0.8228 0.8291 0.8353 0.8415

0.8230 0.8292 0.8354 0.8415

n =

7

distance =

0.0096

Como se puede observar en los datos devueltos por el programa, aunque nosotros le hayamos dicho que el grado sea 5, para que la diferencia entre la función que aproxima y la función original sea $\leq$ 0.01 necesita calcular hasta la aproximación de grado siete, quedando una distancia de 0.0096. En este caso no tenemos una representación gráfica puesto que le hemos indicado que no queremos que nos dibuje usando la opción `no´ al llamar al programa.

-Ejemplo 3

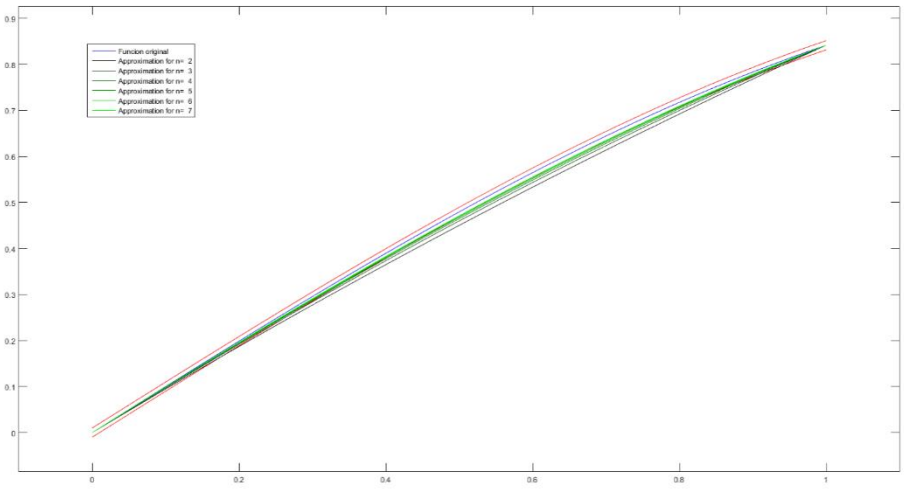


Fig 3.2.2: aproximación `polinomios grado 7, para satisfacer la distancia.

Igual que antes el programa nos dice que ha necesitado hacer funciones hasta grado siete para obtener la distancia que nosotros le hemos dicho, y puesto que nosotros le hemos dicho que nos represente de la [0:5] él lo que hace es que como ha necesitado 7 dibuja desde la 2 a la 7, es decir las 6 últimas.

4. CURVAS DE BÉZIER [2, 5, 16, 17, 18, 21]

4.1 INTRODUCCIÓN

Las curvas de Bézier surgen a partir de los polinomios de Bersntein. Éstos, tal y como se vio en el apartado 2, son una clase particular de polinomios en el campo de los números reales. Estos polinomios se utilizan en el análisis numérico para aproximar de manera uniforme una función continua en un intervalo [a, b]. Otra aplicación de los polinomios de Bernstein son las curvas de Bézier. Éstas se desarrollan entorno a los años 60 para el trazado de dibujos técnicos, así como para el diseño aeronáutico y automovilístico.

Paul Casteljau, ingeniero que trabajaba para Citroën desarrolló en 1959 un algoritmo que a día de hoy recibe su nombre, con el fin de generar mediante ordenador curvas sencillas e intuitivas de manipular que aproximen un conjunto dado de datos. En 1966 el ingeniero de Renault, Pierre Bézier, desarrollo basándose en los polinomios de Bernstein las curvas de Bézier. Tanto la teoría de Casteljau como la de Bézier eran equivalentes, pero reciben el nombre de Bézier porque él las publicó mientras que Casteljau mantuvo su trabajo como documentos internos. Aunque ambos algoritmos construyen las mismas curvas a nivel práctico el algoritmo de Casteljau es preferible. De hecho, el algoritmo de Casteljau se puede ver como un algoritmo de subdivisión, y se sabe que dichos algoritmos son computacionalmente rápidos.

Las curvas de Bézier son muy utilizadas en la realización de gráficos mediante ordenador para modelizar curvas de comportamiento ``suave´´, es decir, que no hayan puntos angulosos. Estas curvas se construyen utilizando un determinado número de *puntos de control.* Generalmente se utilizan tres o cuatro puntos de control, es decir, curvas de Bézier cuadráticas o curvas de Bézier cúbicas. La curva resultante une el primero y el último de estos puntos de control sin puntos angulosos, pero normalmente no pasa por los puntos intermedios ya que estos actúan como si tuviesen cierta capacidad de atracción sobre una cuerda que une los puntos extremos.

4.2 CURVAS DE BÉZIER CUADRÁTICAS

Los tres puntos de control que utilizamos para generar la curva serán P_0, P_1, P_2. Definimos Q_0 como un punto que recorre la recta que une P_0 y P_1, desplazándose por ésta según la ecuación $Q_0 = P_0 + t(P_1 - P_0)$, donde $0 \leq t \leq 1$. Podemos considerar t como un porcentaje del tramo recorrido, es decir, cuando $t = 0$ nos encontramos en $Q_0 = P_0$ y cuando $t = 1$ estamos en $Q_0 = P_1$. A la misma vez y de la misma forma un punto Q_1 recorre la recta que une P_1 y P_2 de acuerdo a la ecuación $Q_1 = P_1 + t(P_2 - P_1)$. Por último un punto C de la curva de Bézier está situado, para cada t, sobre el segmento Q_0 Q_1 siguiendo el mismo tipo de interpolación lineal $C = Q_0 + t(Q_1 - Q_0)$.

Por tanto la ecuación de la curva de Bézier viene dada por:

$$C(t) = Q_0 + t(Q_1 - Q_0),$$

donde Q_0 y Q_1 vienen dados como ya sabemos por:

$$Q_0 = P_0 + t(P_1 - P_0),$$

$$Q_1 = P_1 + t(P_2 - P_1).$$

Sustituimos en la ecuación y nos queda:

$$C(t) = t^2 P_2 + P_1 2t(1-t) + P_0(1-t)^2.$$

La Programación de las curvas de Bézier cuadráticas son un caso particular de la programación de las curvas de Bézier de grado n, y por tanto podemos usar el programa general que se da en la sección 2.3.

A continuación ofrecemos un ejemplo de construcción de curvas de Bézier cuadráticas:

Ejemplo matemático:

En este ejemplo construimos una curva de Bézier cuadrática (tres puntos de control), en la cual recorremos los segmentos que unen tanto los puntos P_0, P_1 así como P_1, P_2 a pasos de 0.05. También obtenemos el punto en la curva (punto verde) para t=0.5 a través del programa de Casteljau.

```
% Example
% [t,curve]=Bezier([1,2; 2,5; 0,7],0.05)
```

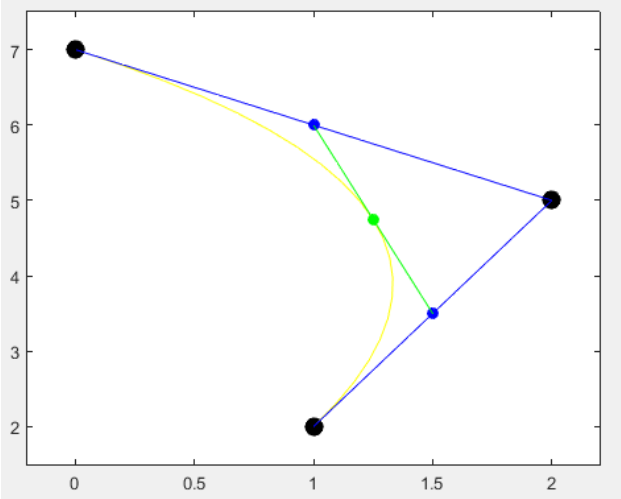


Fig. 4.2.1: Curva de Bézier cuadrática.

Mediante el programa de Casteljau.m, que se puede consultar en la sección 2.3, obtenemos el punto en la curva para t=0.5.

```
% Example
% [Pt,t,P]=Casteljau([1,2; 2,5; 0,7],0.5)
```

Finalmente tenemos los valores devueltos por el programa una vez ejecutado.

```
Pt =

   1.2500    4.7500


t =

   0.5000


P =

   [3x2 double]    [2x2 double]    [1x2 double]

>> P{1}

ans =

    1    2
    2    5
    0    7

>> P{2}

ans =

   1.5000    3.5000
   1.0000    6.0000
```

```
>> P{3}

ans =

   1.2500    4.750
```

4.3 CURVAS DE BÉZIER CÚBICAS

Los cuatro puntos de control que utilizamos para generar la curva serán P_0, P_1, P_2, P_3. Con P_0, P_1, P_2 haremos lo mismo que antes para obtener el punto R_0 y con P_1, P_2,P_3 hacemos lo propio para obtener R_1. Entre R_0 y R_1 hacemos la interpolación lineal correspondiente a t para obtener el punto B de la curva de Bézier producida por los puntos de control. Igual que antes obtenemos el polinomio de grado 3 que corresponde a esta curva con cuatro puntos de control:

$$B(t) = (1-t)R_0 + tR_1.$$

donde sustituimos R_0 y R_1 por:

$$R_0 = t^2P_2 + P_1 2t(1-t) + P_0(1-t)^2,$$

$$R_1 = t^2P_3 + P_2 2t(1-t) + P_1(1-t)^2,$$

y nos quedará:

$$B(t) = P_0(1-t)^3 + P_1 3t(1-t)^2 + P_2 3t^2(1-t) + P_3(1-t)^3.$$

La programación para las curvas de Bézier cúbicas la obtenemos también del caso general, solo tenemos que cambiar los datos iniciales introduciendo un punto de control más.

A continuación realizamos un ejemplo matemático de la curva de Bézier cúbica:

```
[t,curve]=Bezier([1,2; 2,5; 0,7; 0,4],0.05)

t =

  Columns 1 through 10

         0    0.0500    0.1000    0.1500    0.2000    0.2500    0.3000    0.3500    0.4000    0.4500

  Columns 11 through 20
```

0.5000 0.5500 0.6000 0.6500 0.7000 0.7500 0.8000 0.8500 0.9000 0.9500

Column 21

1.0000

curve =

1.0000 2.0000
1.1281 2.4420
1.2150 2.8660
1.2644 3.2690
1.2800 3.6480
1.2656 4.0000
1.2250 4.3220
1.1619 4.6110
1.0800 4.8640
0.9831 5.0780
0.8750 5.2500
0.7594 5.3770
0.6400 5.4560
0.5206 5.4840
0.4050 5.4580
0.2969 5.3750
0.2000 5.2320
0.1181 5.0260
0.0550 4.7540
0.0144 4.4130
0 4.0000

Ahora, igual que hemos hecho anteriormente con la curvas cuadráticas, utilizamos Calteljau.m para obtener el valor del punto en la curva para t=0.5. También podríamos obtenerlo mirando en los resultados anteriores, teniendo en cuenta que t=0.5 es el paso número 11 cuando utilizamos un espaciado dt=0.05.

```
[Pt,t,P]=Casteljau([1,2; 2,5; 0,7; 0,4],0.5)


Pt =


    0.8750    5.2500



t =


    0.5000



P =


    [4x2 double]    [3x2 double]    [2x2 double]    [1x2 double]
```

En la figura 4.3.1 vemos la construcción mediante el algoritmo de Casteljau de la curva de Bézier cúbica del ejemplo realizado, representada en amarillo.

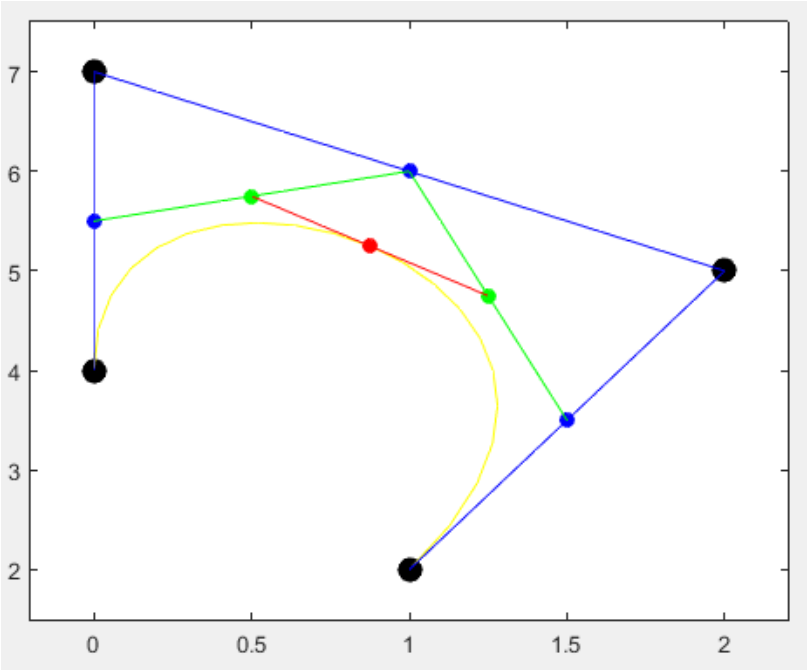


Fig. 4.3.1: Curva de Bézier cúbica.

4.4 CURVAS DE BÉZIER DE GRADO N

Generalizando el procedimiento seguido para curvas de Bézier cuadráticas y cúbicas, podemos obtener curvas de Bézier con grado mayor, aunque debido al esfuerzo computacional que requiere su cálculo no se suelen utilizar. Lo que se hace para facilitar el moldeo de perfiles complejos es utilizar varias curvas de Bézier de grado menor, con grupos de puntos de control próximos, de manera que encajen bien entre sí, igual que ocurre con la interpolación de Lagrange segmentaria.

En general las curvas de Bézier con n puntos de control (n=2,3,4,…) se expresan en la forma:

$$C(t) = \sum_{i=0}^{n} b_{i,n} P_i \, ,$$

$$b_{i,n} = \binom{n}{i} t^i (1-t)^{n-i} \, .$$

Los polinomios $b_{i,n}$ se conocen con el nombre de polinomios básicos de Bernstein de grado n y corresponden precisamente, a los coeficientes en el desarrollo del binomio de Newton de la expresión

$$1 = (t + (1-t))^n = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} = \sum_{i=0}^{n} b_{i,n} \, .$$

La programación en Matlab también es similar a la de las curvas de Bézier tanto cuadráticas como cúbicas, e igual que hemos dicho antes sólo tendremos que poner los puntos de control que nosotros queramos evaluar.

Programación en Matlab

```
function [t,curve]=Bezier(control_points,dt)

% This functions computes the Bezier curve of a given set of control
points
% [t,curve]=Bezier(control_points,dt);
% Input variables:
% control_points is a nx2 matrix which contains the (x,y) coordinates
of each
%                 control point by row
% dt is the step size in the variable t which parametrizes the curve
% Output variables:
% t discretization of the variable which parametrizes the curve
% curve values of the curve points for time t
%
% Example
% [t,curve]=Bezier([1,2; 2,5; 0,7],0.05)


% number of control points

n=size(control_points,1);



% we create the discretization in time

t=0:dt:1;
nt=length(t);

% we calculate the point at the Bezier Curve for a given time t

for j=1:nt

    curve(j,:)=[0,0];
for i=1:n
        b(i)=combinatorio(n-1,i-1)*t(j)^(i-1)*(1-t(j))^(n-i);
        curve(j,:)=curve(j,:)+b(i)*control_points(i,:);
end

end




%%%%%%%%%%%%%%
% Plots
%%%%%%%%%%%%%%
```

```
% list of colors

colors={'blue','green','red','cyan','magenta','yellow','black'};
colors2={'b','g','r','c','m','y','b'};

dt_plot=0.1;
t_plot=0:dt_plot:1;
nt_plot=length(t_plot);

for j=1:nt_plot


% we plot the control points

plot(control_points(:,1),control_points(:,2),'o','MarkerSize',10,'Mark
erEdgeColor','k','MarkerFaceColor','k')

hold on

% we adjust the limits of the plot

mx=min(control_points(:,1));
Mx=max(control_points(:,1));
my=min(control_points(:,2));
My=max(control_points(:,2));
dx=0.1*(Mx-mx);
dy=0.1*(My-my);

axis([mx-dx Mx+dx my-dy My+dy]);


% we plot the Bezier Curve

plot(curve(:,1),curve(:,2),'-','Color',colors{6})


% we plot the iterative process based on straight lines


P{1}=control_points;


for s=n-1:-1:1

        clear point_trectapoints

        points=P{n-s};

for c=1:s
            point_t(c,:)=(1-
t_plot(j))*points(c,:)+t_plot(j)*points(c+1,:);

for jaux=1:nt_plot
                recta(jaux,:)=(1-
t_plot(jaux))*points(c,:)+t_plot(jaux)*points(c+1,:);
end
```

```
        plot(recta(:,1),recta(:,2),'-','Color',colors{n-s});

end

        P{n-s+1}=point_t;

plot(point_t(:,1),point_t(:,2),'o','MarkerEdgeColor',colors2{n-
s},'MarkerFaceColor',colors2{n-s})

        pause(1);

end

      pause(2);

    hold off

end
```

Ejemplo matemático:

curva de Bézier con seis puntos de control

>> [t,curve]=Bezier([1,2; 2,5; 0,7; 0,4;0,2;-1,2],0.05)

t =

Columns 1 through 10

0 0.0500 0.1000 0.1500 0.2000 0.2500 0.3000 0.3500 0.4000 0.4500

Columns 11 through 20

0.5000 0.5500 0.6000 0.6500 0.7000 0.7500 0.8000 0.8500 0.9000 0.9500

Column 21

1.0000

curve =

1.0000 2.0000
1.1810 2.7203
1.2466 3.3649
1.2266 3.9142
1.1466 4.3552
1.0273 4.6807
0.8859 4.8885
0.7355 4.9815
0.5859 4.9664
0.4437 4.8535
0.3125 4.6563
0.1937 4.3904
0.0861 4.0736
-0.0132 3.7249
-0.1089 3.3640
-0.2070 3.0107
-0.3146 2.6848
-0.4393 2.4047
-0.5896 2.1877
-0.7737 2.0486
-1.0000 2.0000

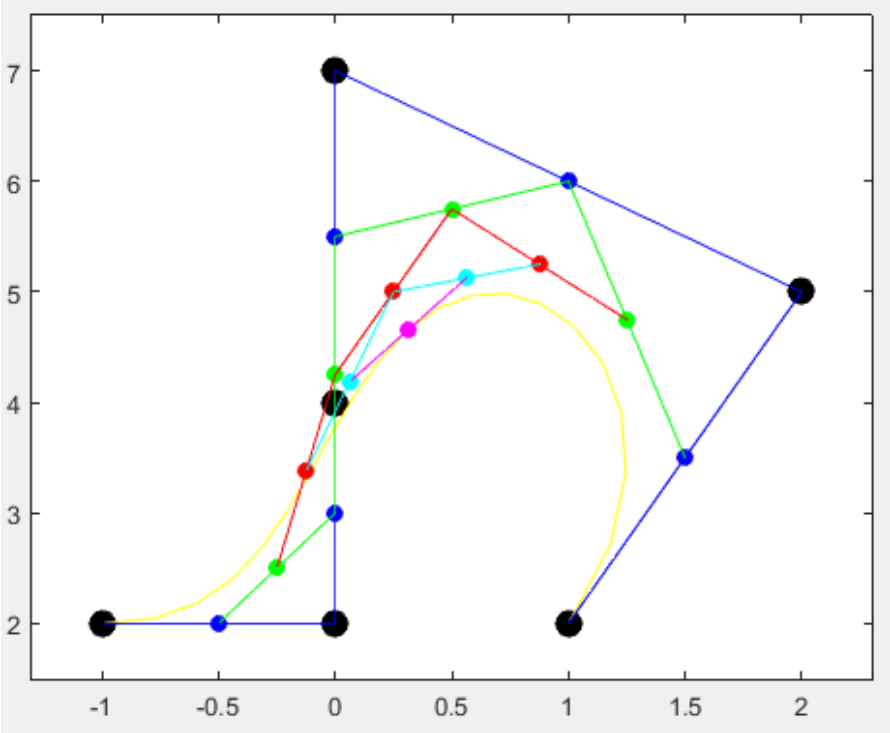


Fig. 4.4.1: Curva de Bézier de grado n=5.

El la figura 4.4.1 se ven los diferentes pasos para construir la curva de Bézier de grado n=5 representada en amarillo.

Encontramos de nuevo que la forma de calcular la curva de Bézier de grado n computacionalmente adecuada es mediante el algoritmo de subdivisión de Casteljau. Esta es una forma más rápida y eficiente de calcular la curva de Bézier.

4.5 ALGORITMO DE SUBDIVISIÓN DE CASTELJAU

Este algoritmo es un método recursivo para el cálculo de polinomios tanto en la forma de Bernstein como en las curvas de Bézier. Como hemos dicho anteriormente toma su nombre del ingeniero de Citroën Paul Casteljau.

La base de este algoritmo surge de requisitos gráficos en informática y se basa en que una restricción de una curva de Bézier sigue siendo una curva de Bézier. El algoritmo de Casteljau coincide de hecho con la forma en que se han presentado las curvas de Bézier cuadráticas y cúbicas, es decir, consiste en la iteración y cálculo de la posición de los respectivos puntos en cada segmento. Dado una curva de Bézier de grado n

$$B(t) = \sum_{i=0}^{n} P_i \; b_{i,n}(t),$$

donde $b_{i,n}(t)$ es un polinomio en la base de Bernstein, podemos calcular su valor en el punto t_0 con la siguiente relación de recurrencia.

Pseudocódigo del Algoritmo de Casteljau:

for i=0: n

$$P_i^0 = P_i$$

end

for j=1: n %pasos de recursividad

for i=0: n-j %puntos en cada paso

$$P_i^j = P_i^{(j-1)}(1 - t_0) + t_0 P_{i+1}^{(j-1)}$$

end

end

Programación Matlab:

```
function [Pt,t,P]=Casteljau(control_points,t)

% This functions computes the Casteljau algorithm to construct the
Bezier curve
% based on a given set of control points
% [Pt,t,P]=Casteljau(control_points,t);
% Input variables:
% control_points is a nx2 matrix which contains the (x,y) coordinates
of each
%                 control point by row
% t is the value of the parameter of the curve where we want to
evaluate
% Output variables:
% Pt is the point of the curve where we stay at value t
% t is the value of the parameter of the curve where we have evaluated
% P puntos usados en el algoritmo de recursión para calcular Pt
%
% Example
% [t,Pt,P]=Casteljau([1,2; 2,5; 0,7],0.5)


% number of control points

n=size(control_points,1);


% we start the Casteljau algorithm

P{1}=control_points;


for s=n-1:-1:1

    clear point_tpoints

    points=P{n-s};
```

```
for c=1:s
        point_t(c,:)=(1-t)*points(c,:)+t*points(c+1,:);
end


    P{n-s+1}=point_t;



end

Pt=P{n};
```

Ejemplo matemático. Resultados del programa:

>> [t,Pt,P]=Casteljau([1,2; 2,5; 0,7],0.5)

t =

1.2500 4.7500

Pt =

0.5000

P =

[3x2 double] [2x2 double] [1x2 double]

como podemos observar lo que obtenemos es el punto correspondiente a la curva en el punto medio de cada segmento.

4.5.1 BÉZIER CASTELJAU

Llamando de forma reiterada al algoritmo de Casteljau podemos obtener la Curva de Bézier con sus puntos correspondientes a cada paso dt.

Programación Matlab:

```
function [t,curve]=Bezier_Casteljau(control_points,dt)

% This functions computes the Bezier curve of a given set of control
points
% [t,curve]=Bezier_Casteljau(control_points,dt);
% Input variables:
% control_points is a nx2 matrix which contains the (x,y) coordinates
of each
%                 control point by row
% dt is the step size in the variable t which parametrizes the curve
% Output variables:
% t discretization of the variable which parametrizes the curve
% curve values of the curve points for time t
%
% Example
% [t,curve]=Bezier_Casteljau([1,2; 2,5; 3,7],0.05)


% number of control points

n=size(control_points,1);



% we create the discretization in time

t=0:dt:1;
nt=length(t);

% we calculate the point at the Bezier Curve for a given time t

for j=1:nt

    curve(j,:)=Casteljau(control_points,t(j));

end




%%%%%%%%%%%%%%
% Plots
%%%%%%%%%%%%%%


% we plot the control points

plot(control_points(:,1),control_points(:,2),'o','MarkerSize',10,'Mark
erEdgeColor','k','MarkerFaceColor','k')

hold on

% we adjust the limits of the plot
```

```
mx=min(control_points(:,1));
Mx=max(control_points(:,1));
my=min(control_points(:,2));
My=max(control_points(:,2));
dx=0.1*(Mx-mx);
dy=0.1*(My-my);

axis([mx-dx Mx+dx my-dy My+dy]);


% we plot the Bezier Curve

plot(curve(:,1),curve(:,2),'-','Color','blue');
```

Ejemplo matemático:

A continuación realizaremos el mismo ejemplo realizado en el apartado 2.1Curvas de Bézier cuadráticas, pero esta vez utilizando el algoritmo de Casteljau.

```
% Example
% [t,curve]=Bezier_Casteljau([1,2; 2,5; 3,7],0.05)
```

>> [t,curve]=Bezier_Casteljau([1,2; 2,5; 0,7],0.05)

t =

Columns 1 through 10

0 0.0500 0.1000 0.1500 0.2000 0.2500 0.3000 0.3500 0.4000 0.4500

Columns 11 through 20

0.5000 0.5500 0.6000 0.6500 0.7000 0.7500 0.8000 0.8500 0.9000 0.9500

Column 21

1.0000

curve =

1.0000	2.0000
1.0925	2.2975
1.1700	2.5900
1.2325	2.8775
1.2800	3.1600
1.3125	3.4375
1.3300	3.7100
1.3325	3.9775
1.3200	4.2400
1.2925	4.4975
1.2500	4.7500
1.1925	4.9975
1.1200	5.2400
1.0325	5.4775
0.9300	5.7100
0.8125	5.9375
0.6800	6.1600
0.5325	6.3775
0.3700	6.5900
0.1925	6.7975
0	7.0000

Como podemos observar en los datos devueltos por el programa, para cada paso tenemos el punto correspondiente a la curva de forma que si nos vamos al paso 11 que corresponde con el ejemplo anterior de t=0.5, tenemos que el punto es el mismo que antes P[1.25, 4.75].

En la figura 4.5.1.1 podemos observar la curva de Bézier obtenida para los tres puntos iniciales introducidos.

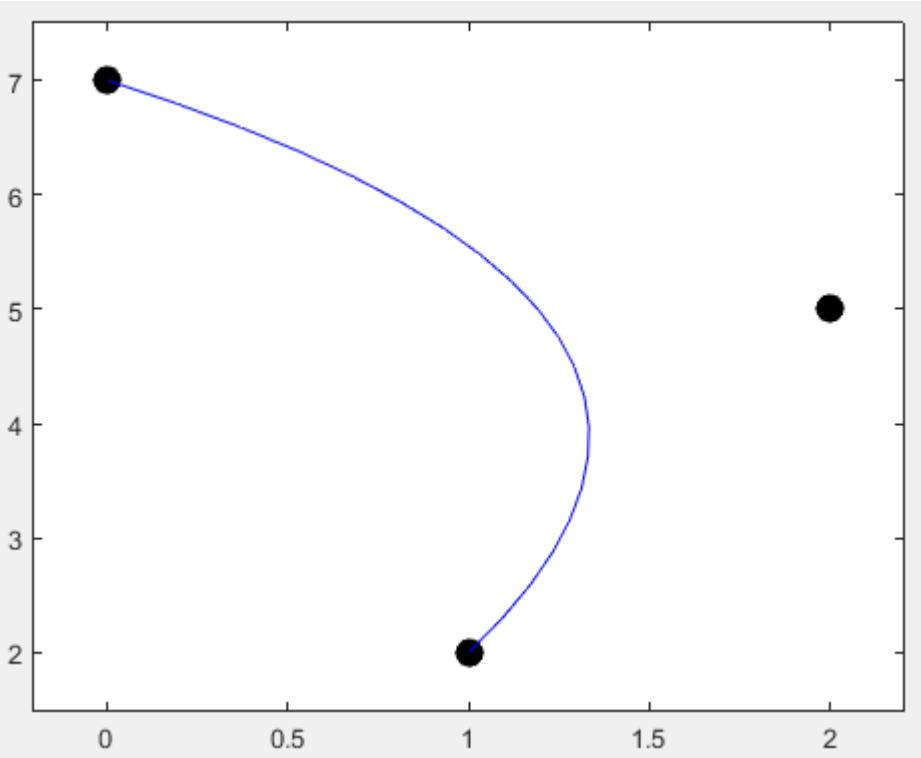


Fig. 4.5.1.1: Bézier_Casteljau

Recordemos que este algoritmo tiene menor coste computacional que el que utiliza los polinomios de Bernstein directamente.

4.6 CURVAS DE BÉZIER CON PESOS

En ocasiones también se tienen en cuenta curvas de Bézier para las que los puntos de control, P_0, P_1, P_2,…, P_n tienen pesos w_0, w_1, w_2…, w_n, lo que quiere decir que habrá que asignar poderes de atracción diferentes a cada punto de control y entonces la fórmula sería la siguiente:

$$C(t) = \frac{\sum_{i=0}^{n} b_{i,n} P_i w_i}{\sum_{i=0}^{n} b_{i,n} w_i}.$$

A estas curvas de Bézier con pesos también se les suele denominar curvas de Bézier racionales.

Programación Matlab:

```
function [t,curve]=Bezier_pesos(control_points,dt,w)

% This functions computes the Bezier curve of a given set of control
points
% taking into account the assigned weights to each point
% [t,curve]=Bezier_pesos(control_points,dt,w);
% Input variables:
```

```
% control_points is a nx2 matrix which contains the (x,y) coordinates
of each
%                  control point by row
% dt is the step size in the variable t which parametrizes the curve
% w is the vector with the weight values
% Output variables:
% t discretization of the variable which parametrizes the curve
% curve values of the curve points for time t
%
% Example
% [t,curve]=Bezier_pesos([1,2; 2,5; 3,7],0.05,[1,1,1])

% number of control points

n=size(control_points,1);



% we create the discretization in time

t=0:dt:1;
nt=length(t);

% we calculate the point at the Bezier curve for a given time t

for j=1:nt

    curve_nume(j,:)=[0,0];
    deno=0;
for i=1:n
        b(i)=combinatorio(n-1,i-1)*t(j)^(i-1)*(1-t(j))^(n-i);
        curve_nume(j,:)=curve_nume(j,:)+w(i)*b(i)*control_points(i,:);
        deno=deno+w(i)*b(i);
end

    curve(j,:)=curve_nume(j,:)/deno;

end




%%%%%%%%%%%%%%
% Plots
%%%%%%%%%%%%%%

% list of colors

colors={'blue','green','red','cyan','magenta','magenta','black'};
colors2={'b','g','r','c','m','y','b'};

dt_plot=0.1;
t_plot=0:dt_plot:1;
nt_plot=length(t_plot);

for j=1:nt_plot
```

```
% we plot the control points


plot(control_points(:,1),control_points(:,2),'o','MarkerSize',10,'Mark
erEdgeColor','k','MarkerFaceColor','k')

    hold on
for l=1:n
        nombre_punto=sprintf('P%d',l-1);

text(control_points(l,1)+0.15,control_points(l,2)+0.15,nombre_punto)

end

% we adjust the limits of the plot

    mx=min(control_points(:,1));
    Mx=max(control_points(:,1));
    my=min(control_points(:,2));
    My=max(control_points(:,2));
    dx=0.1*(Mx-mx);
    dy=0.1*(My-my);

    axis([mx-dx Mx+dx my-dy My+dy]);


% we plot the Bezier Curve

    plot(curve(:,1),curve(:,2),'-','Color',colors{6})


% we plot the iterative process based on straight lines


    P{1}=control_points;


for s=n-1:-1:1

        clear point_trectapoints

        points=P{n-s};

for c=1:s
            point_t(c,:)=(1-
t_plot(j))*points(c,:)+t_plot(j)*points(c+1,:);

for jaux=1:nt_plot
                recta(jaux,:)=(1-
t_plot(jaux))*points(c,:)+t_plot(jaux)*points(c+1,:);
end


            plot(recta(:,1),recta(:,2),'-','Color',colors{n-s});
```

```
end

        P{n-s+1}=point_t;

plot(point_t(:,1),point_t(:,2),'o','MarkerEdgeColor',colors2{n-
s},'MarkerFaceColor',colors2{n-s})

        pause(1);
end
    pause(2);
    hold off

end
```

Ejemplo matemático:

A continuación, realizaremos dos ejemplos en los cuales tendremos 3 puntos de control P_0, P_1, P_2 que serán los mismos, pero en el primer ejemplo la atracción de cada uno de los puntos sobre la curva será el mismo e igual a 1 y en el segundo ejemplo variaremos los pesos con la intención de observar cómo afecta esto a la curva. En el caso de los pesos igual a 1 la curva de Bézier con pesos coincide con la curva de Bézier usual.

- 1. Ptos[1,2; 2,5; 0,7] Pesos[1,1,1]

```
[t,curve]=Bezier_pesos([1,2; 2,5; 0,7],0.05,[1,1,1])

t =

 Columns 1 through 10

    0    0.0500    0.1000    0.1500    0.2000    0.2500    0.3000    0.3500    0.4000    0.4500

 Columns 11 through 20

  0.5000    0.5500    0.6000    0.6500    0.7000    0.7500    0.8000    0.8500    0.9000    0.9500
```

Column 21

1.0000

curve =

1.0000	2.0000
1.0925	2.2975
1.1700	2.5900
1.2325	2.8775
1.2800	3.1600
1.3125	3.4375
1.3300	3.7100
1.3325	3.9775
1.3200	4.2400
1.2925	4.4975
1.2500	4.7500
1.1925	4.9975
1.1200	5.2400
1.0325	5.4775
0.9300	5.7100
0.8125	5.9375
0.6800	6.1600
0.5325	6.3775
0.3700	6.5900
0.1925	6.7975
0	7.0000

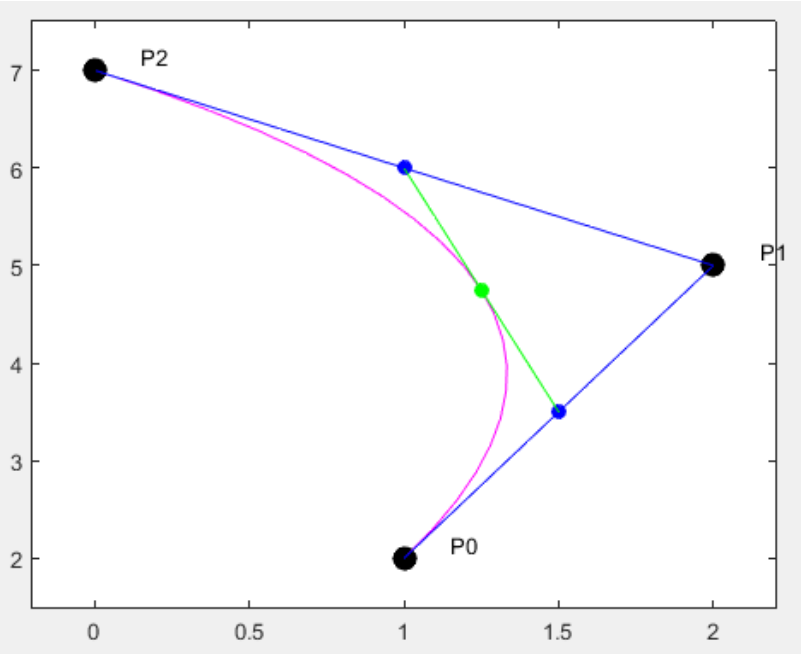


Fig. 4.6.1: Ejemplo 1

- 2. Ptos[1,2; 2,5; 0,7] Pesos[0.9,1.8,1.2]

[t,curve]=Bezier_pesos([1,2; 2,5; 0,7],0.05,[0.9,1.8,1.2])

t =

Columns 1 through 10

0 0.0500 0.1000 0.1500 0.2000 0.2500 0.3000 0.3500 0.4000 0.4500

Columns 11 through 20

0.5000 0.5500 0.6000 0.6500 0.7000 0.7500 0.8000 0.8500 0.9000 0.9500

Column 21

1.0000

curve =

1.0000	2.0000
1.1703	2.5354
1.2930	2.9690
1.3802	3.3307
1.4400	3.6400
1.4776	3.9104
1.4966	4.1517
1.4992	4.3710
1.4870	4.5739
1.4608	4.7648
1.4211	4.9474
1.3676	5.1248
1.3000	5.3000
1.2172	5.4757
1.1179	5.6547
1.0000	5.8400
0.8609	6.0348
0.6969	6.2429
0.5034	6.4690
0.2740	6.7188
0	7.0000

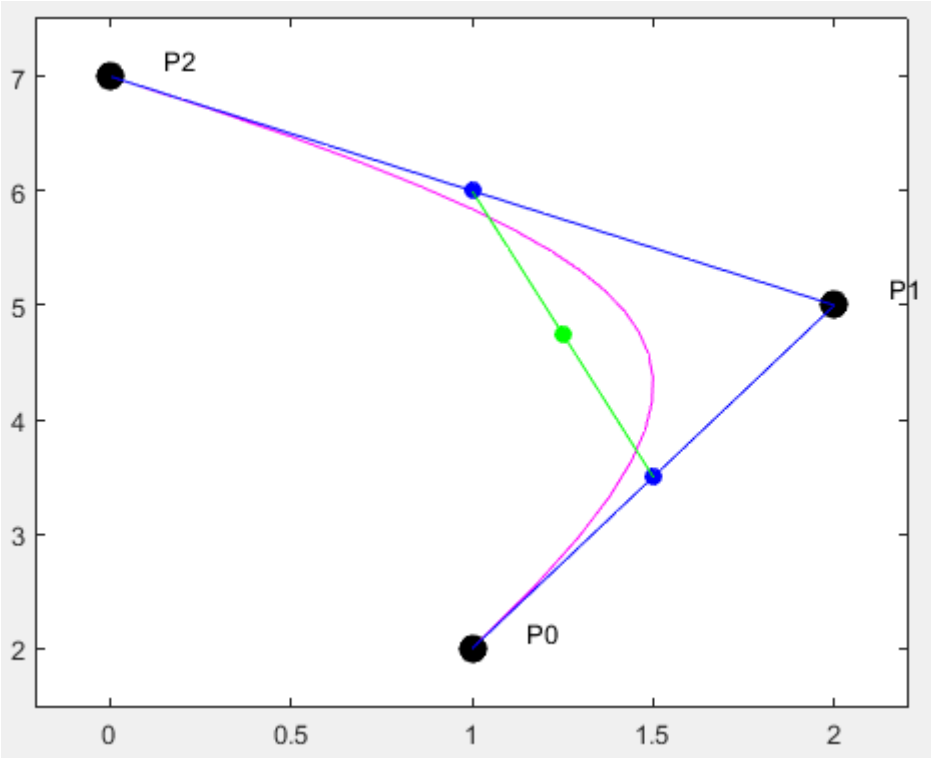


Fig. 4.6.2: Ejemplo 2

Si comparamos las dos imágenes figura 4.6.1 y figura 4.6.2 podemos ver cómo influye la atracción de cada punto de control sobre la curva según el peso asociado a dicho punto. En particular es evidente que la curva se acerca más al punto (2,5) cuando el peso asociado a este punto es mayor.

5. SPLINES [2, 3, 4, 9, 10, 19, 22]

5.1 INTRODUCCIÓN

El origen de este concepto viene del uso de una lámina de plástico delgada, a la cual se denomina curvígrafo, para el trazado de curvas suaves a través de un conjunto de puntos. Los splines más conocidos son ecuaciones cúbicas que modelan el comportamiento de las curvas a trazar, permitiéndonos unir de manera continua y suave una serie de puntos.

El método más sencillo de interpolación segmentaria es lineal, en el cual se unen los puntos a través de líneas rectas. El problema al utilizar este método es que en cada extremo de los intervalos no tenemos una unión continua de las derivadas. Este inconveniente lo podemos solucionar considerando polinomios de mayor grado, por ejemplo con grado 2 tendremos los splines cuadráticos que aseguran la suavidad de la primera derivada. Si también queremos lograr la continuidad de la segunda derivada entonces podemos considerar la interpolación cúbica segmentaria, que usa polinomios cúbicos diseñados de manera que se asegure la continuidad y suavidad de la función, la primera y la segunda derivada. La continuidad en la primera derivada significa que no hay puntos angulosos y la continuidad de la segunda derivada significa que no hay cambios bruscos de convexidad.

En este capítulo se definen splines de distintos órdenes utilizando polinomios de cualquier grado, aunque los más utilizados son los polinomios cúbicos.

5.2 DEFINICIÓN

Como ya hemos dicho, las funciones spline están definidas a trozos por varios polinomios, uniéndose entre sí mediante ciertas condiciones de continuidad. Con ellas obtenemos un mejor resultado a la hora de aproximar la forma de una función dada que si tomásemos polinomios de grado alto, ya que normalmente producen grandes oscilaciones.

5.3 DEFINICIÓN DE SPLINE DE GRADO N

Sea $a = x_1 < x_2 < \cdots < x_n$=b una partición del intervalo [a,b]. Se llama spline de grado $N \in \mathbb{N}$ con nodos en $x_1 < x_2 < \cdots < x_n$ a una función s(x) definida en [a,b] verificando

1. La restricción de s(x) a cada $[x_i,\ x_{i+1}]$, i=1,2,….,n-1 es un polinomio s_i(x) de grado menor o igual que N.
2. S(x) y sus derivadas sucesivas hasta el orden N-1 inclusive son continuas en [a,b].

5.4 DEFINICIÓN DE SPLINE LINEAL

Estos splines son funciones polinómicas de grado 1, es decir rectas de la forma f(x)=ax+b, cuya función es unir cada par de puntos con una recta.

Consideremos los n+1 puntos con abscisas y ordenadas:

$$x_1, x_2, \ldots\ldots, x_n,$$

$$y_1, y_2, \ldots\ldots, y_n.$$

Una función spline de grado 1 que interpole los datos consiste simplemente en unir los puntos con segmentos de recta.

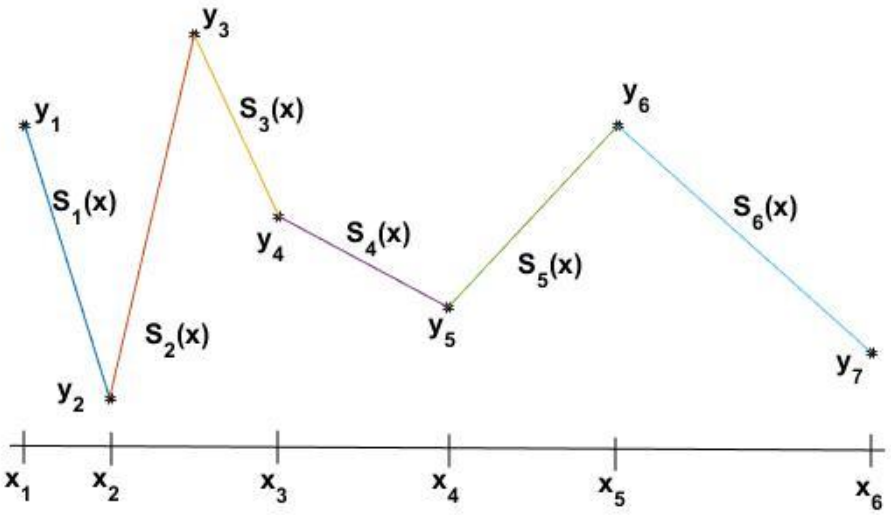


Fig 5.4.1: Gráfica genérica spline 1.

Como vemos en la figura 5.4.1 la función generada cumple con las condiciones de spline de grado 1. Por tanto para este caso tenemos:

$$S(x) = \begin{cases} s_1(x) = a_1x + b_1, & x \in (x_1, x_2)\ , \\ s_2(x) = a_2x + b_2, & x \in (x_2, x_3)\ , \\ \qquad\qquad \dots\dots & \\ s_{n-1}(x) = a_{n-1}x + b_{n-1}, & x \in (x_{n-1}, x_n)\ , \end{cases}$$

donde:

1. $s_j(x)$ *es un polinomio de grado menor o igual que* 1.
2. $S(x)$ *es una función continua.*
3. $s_j(x_j) = y_j, para\ j = 1,2, \dots, n.$

El spline de grado 1 también puede definirse como:

$$S(x) = \begin{cases} y_1 + f[x_1, x_2](x - x_1) & si\ \ x \in (x_1, x_2), \\ y_2 + f[x_2, x_3](x - x_2) & si\ \ x \in (x_2, x_3), \\ \qquad\qquad \dots\dots & \\ y_{n-1} + f[x_{n-1}, x_n](x - x_{n-1}) & si\ \ x \in (x_{n-1}, x_n), \end{cases}$$

donde $f[x_i, x_j]$ es la diferencia dividida de Newton.

A continuación, se muestra la programación en Matlab

```
function [Si]=Splines1_Interpolant(t,y,opx,x,opg1)



% This function builds the linear interpolant spline through the given
two
% dimensional data. Some plots of the spline functions, and
% the interpolated values are displayed. In a separate file the
coefficients of the piecewise polynomials
% of degree 1 are saved.
%
% [Si]=Splines1_Interpolant(t,y,opx,x,opg1)
%
% Input variables:
% t:abscissae of the nodes
% y:ordinates of the nodes
% opx: parameter that indicates if we need to compute or not
interpolated
% values
%     'y': interpolated values are computed ; 'n': they are not
computed
% x: net of points where we want to evaluate the spline
% opg1: parameter that indicates if we want to draw the plot with the
% interpolated values
%
% Output variables:
% Si vector that contains the interpolated values at x
%



% Number of nodes

m=length(t);

% We check that the entries are correct

if m<2
uiwait(msgbox('The number of control points must be larger than 1',
'Error message',...
'error','modal'));
return;
end



% We define the vector of distances between abcissae
h=diff(t,1);



% We compute the delta values

delta(1)=0;
for i=2:m
delta(i)=(y(i)-y(i-1))/h(i-1);
```

```
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluation of the spline at x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if opx=='y'

% It finds the polynomial piece to evaluate for each entrie of x
if x(1)==t(1)
minim=y(1); maxim=y(1);
Si(1)=y(1);
else
ind=find((x(1)>t)==0);
ind=ind(1)-1;
ind=ind(1);
% It applies the formule to compute S_ind(x)

Si(1)=y(ind)+delta(ind+1)*(x(1)-t(ind));
        minim=Si; maxim=Si;
end

for i=2:length(x)
ind=find((x(i)>t)==0);
ind=ind(1)-1;
ind=ind(1);
% It applies the formule to compute S_ind(x)

Si(i)=y(ind)+delta(ind+1)*(x(i)-t(ind));
if Si(i)<minim
            minim=Si(i);
elseif Si(i)>maxim
            maxim=Si(i);
end
end

if opg1==1
figure(1);
% It draws the control points in the plot
        b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
        hold on;
% It draws the interpolated values in the plot
        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))]);
legend([b1,b2],'Control points','Interpolated values');
        hold off
end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Polynomials S_i(x)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

syms r;



for i=1:m-1

S(i)=y(i)+delta(i+1)*(r-t(i));

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot of the spline
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% It draws the control points in the plot of the spline
figure;
b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
hold on;

% It draws each one of the polynomials
Ms=0;
ms=0;
for i=1:m-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(double([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([b1,b3],'Control points','Linear splines');
title('Approximation by interpolant linear splines');



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Output of the results to a file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% It opens or creates a new file to write the interpolated values

if opx=='y'
    fid=fopen('result1_interpolated_values.txt','w');
for i=1:length(x)
fprintf(fid,'%f %f \n',x(i),Si(i));
end
    fclose(fid);
end
```

```
%  It opens or creates a new file to write the piecewise polynomials
of the
%  spline
fid=fopen('spline1_polynomials.txt','w');
fprintf(fid,'**********************************************************
****************\n');
hour=clock;
fprintf(fid,'Results obtained at %d-%d-%d,  %d : %d : %d\n',...
    hour(3),hour(2),hour(1),hour(4),hour(5),fix(hour(6)));
fprintf(fid,'**********************************************************
****************\n\n');
signs='----';
for i=1:length(t)-1
    coefi=sym2poly(expand(S(i)));

if length(coefi)==2

for j=1:2
if coefi(j)>=0
                signs(j)='+';
end
end
fprintf(fid,' %c %f x %c %f  \n',...
            signs(1),abs(coefi(1)),signs(2),abs(coefi(2)));

elseif length(coefi)==1

if coefi(1)>=0
signs(1)='+';
end
fprintf(fid,' %c %f  \n',...
signs(1),abs(coefi(1)));

end

end
fprintf(fid,'\n');
fclose(fid);
```

Ejemplo:

```
>> t=[1,2,3,4,6,8,11];
 y=[2,-1,3,1,0,-1,-4];
 opx='y';
 x=[1:0.2:11];

 [Si]=Splines1_Interpolant(t,y,opx,x,1)
```

1.000000 2.000000

1.200000 1.400000

1.400000 0.800000

1.600000 0.200000

1.800000 -0.400000

2.000000 -1.000000

2.200000 -0.200000

2.400000 0.600000

2.600000 1.400000

2.800000 2.200000

3.000000 3.000000

3.200000 2.600000

3.400000 2.200000

3.600000 1.800000

3.800000 1.400000

4.000000 1.000000

4.200000 0.900000

4.400000 0.800000

4.600000 0.700000

4.800000 0.600000

5.000000 0.500000

5.200000 0.400000

5.400000 0.300000

5.600000 0.200000

5.800000 0.100000

6.000000 0.000000

6.200000 -0.100000

6.400000 -0.200000

6.600000 -0.300000

6.800000 -0.400000

7.000000 -0.500000

7.200000 -0.600000

7.400000 -0.700000

7.600000 -0.800000

7.800000 -0.900000

8.000000 -1.000000

8.200000 -1.200000

8.400000 -1.400000

8.600000 -1.600000

8.800000 -1.800000

9.000000 -2.000000

9.200000 -2.200000

9.400000 -2.400000

9.600000 -2.600000

9.800000 -2.800000

10.000000 -3.000000

10.200000 -3.200000

10.400000 -3.400000

10.600000 -3.600000

10.800000 -3.800000

11.000000 -4.000000

****

Results obtained at 16-1-2020, 13 : 18 : 35

****

- 3.000000 x + 5.000000

+ 4.000000 x + 9.000000

+ 2.000000 x + 9.000000

+ 0.500000 x + 3.000000

+ 0.500000 x + 3.000000

+ 1.000000 x + 7.000000

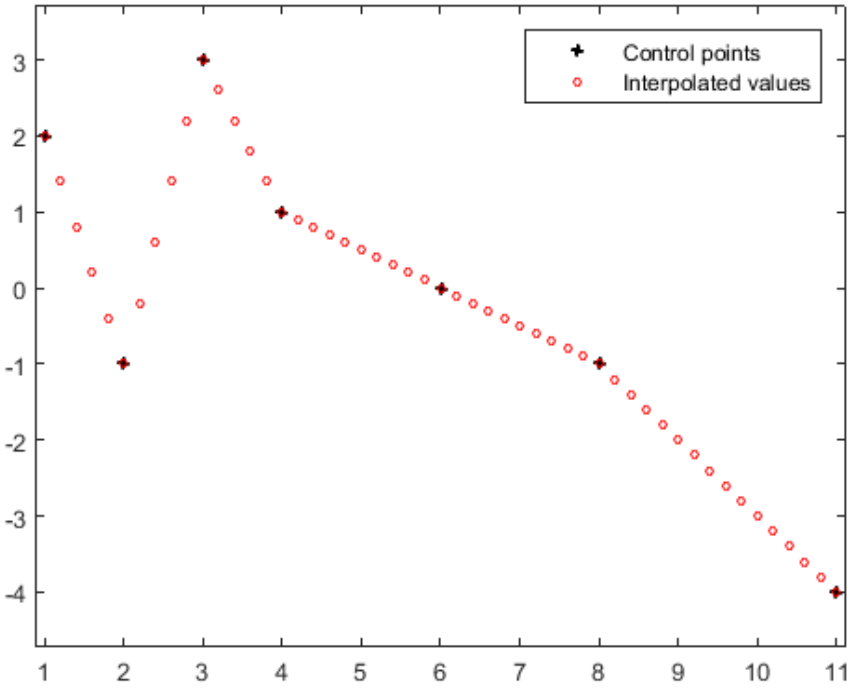


Fig 5.4.2: valores interpolados entre cada par de puntos.

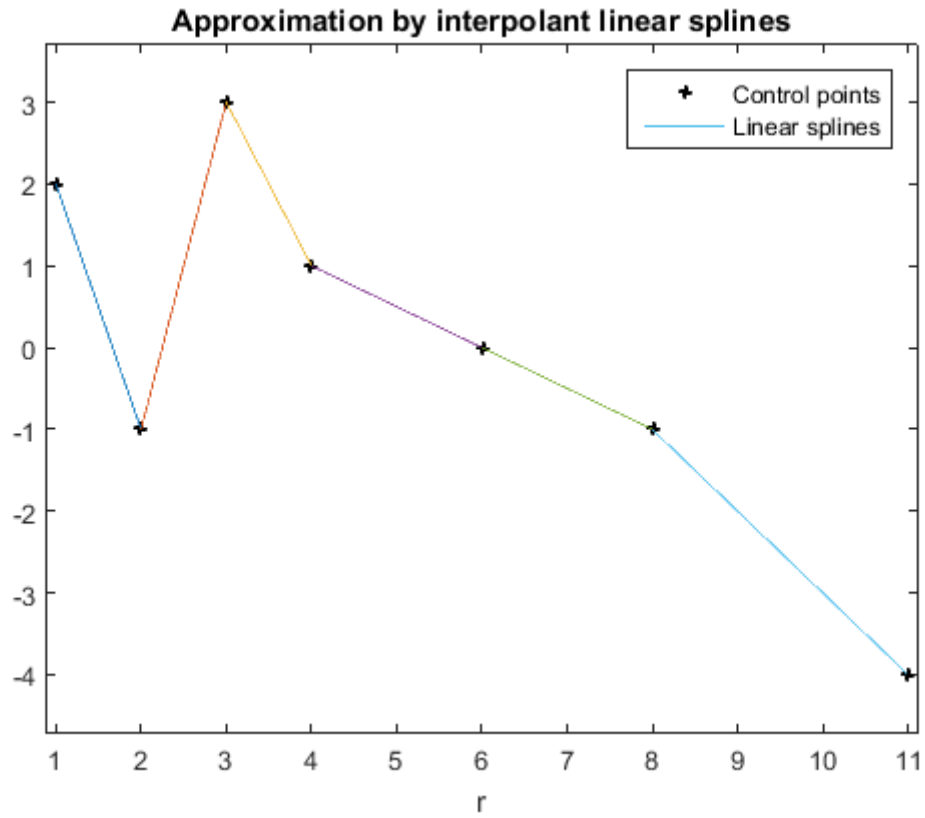


Fig 5.4.3: aproximación por splines lineales interpolantes.

5.5 DEFINICIÓN DE SPLINE CUADRÁTICO

Pasamos a definir una función spline con polinomios cuadráticos, es decir,

$$S(x) = s_i(x), \quad para \ x \in [x_i, x_{i+1}],$$

donde $s_i(x)$ es una función polinomial de grado dos en cada intervalo:

$s_i(x) = a_i(x - x_{i+1})(x - x_i) + b_i(x - x_i) + c_i.$

Ahora tendremos que hacer que el spline pase por los puntos de la tabla de datos, cumpliéndose que:

$S(x_1) = y_1\,, \quad S(x_2) = y_2, \ldots\ldots, \quad S(x_{n-1}) = y_{n-1}, \quad S(x_n) = y_n.$

Es decir,

$$s_i(x) \quad para \ x \in [x_i, x_{i+1}],$$

$$s_i(x_i) = y_i\,, \ s_i(x_{i+1}) = y_{i+1.}$$

De estas dos condiciones se deducen fácilmente los valores de c_i y b_i

$$s_i(x_i) = c_i = y_{i,}$$

$$s_i(x_{i+1}) = b_i h_i + c_i = y_{i+1} \ ==> b_i = \frac{y_{i+1} - y_i}{h_i} \qquad donde \ h_i = x_{i+1} - x_i.$$

Hacemos notar que hemos expresado el polinomio

$$s_i(x) = a_i(x - x_{i+1})(x - x_i) + b_i(x - x_i) + c_i,$$

en la base:

$$\{1, (x - x_i), (x - x_{i+1})(x - x_i)\}.$$

Como sabemos el valor de c_i y de b_i lo sustituimos en la expresión $s_i(x)$, y nos queda que:

$$s_i(x) = a_i(x - x_{i+1})(x - x_i) + \frac{y_{i+1} - y_i}{h_i}(x - x_i) + y_i.$$

En $[x_{i-1}, x_i]$

$$s_{i-1}(x) = a_{i-1}(x - x_i)(x - x_{i-1}) + \frac{y_i - y_{i-1}}{h_{i-1}}(x - x_{i-1}) + y_{i-1.}$$

Imponemos que:

$$s'_{i-1}(x_i) = s'_i(x_i).$$

Necesitamos por tanto realizar el cálculo de $s'_i(x)$ y $s'_{i-1}(x)$, que aparece a continuación:

$$s'_i(x) = a_i(x - x_{i+1}) + a_i(x - x_i) + \frac{y_{i+1} - y_i}{h_i},$$

$$s'_i(x_i) = -a_i h_i + \frac{y_{i+1} - y_i}{h_i}, \qquad (5.5.1)$$

$$s'_{i-1}(x) = a_{i-1}(x - x_{i-1}) + a_i(x - x_i) + \frac{y_i - y_{i-1}}{h_{i-1}},$$

$$s'_{i-1}(x_i) = a_{i-1}h_{i-1} + \frac{y_i - y_{i-1}}{h_{i-1}}. \qquad (5.5.2)$$

Igualando (5.5.1) y (5.5.2) llegamos a:

$$-a_i h_i + \frac{y_{i+1} - y_i}{h_i} = a_{i-1}h_{i-1} + \frac{y_i - y_{i-1}}{h_{i-1}}.$$

Definiendo:

$$\delta_{i+1} = \frac{y_{i+1} - y_i}{h_i},$$

$$\delta_i = \frac{y_i - y_{i-1}}{h_{i-1}},$$

la ecuación anterior se convierte en:

$$a_{i-1}h_{i-1} + a_i h_i = \delta_{i+1} - \delta_i \qquad para \quad i = 2, \dots \dots, n-1.$$

Tenemos por tanto el sistema de ecuaciones lineales siguiente:

$$\begin{cases} a_1h_1 + a_2h_2 = \delta_3 - \delta_2 \\ a_2h_2 + a_3h_3 = \delta_4 - \delta_3 \\ a_3h_3 + a_4h_4 = \delta_5 - \delta_4 \\ \quad . \\ a_{n-2}h_{n-2} + a_{n-1}h_{n-1} = \delta_n - \delta_{n-1} \end{cases} \qquad n-1 \; incógnitas\,, \;\; n-2 \; ecuaciones. \quad (5.5.3)$$

Si se supone la condición de frontera en la frontera izquierda:

$$S'(x_1) = y'_1 \qquad conocido,$$

operando llegamos a una nueva ecuación que completa el sistema:

$$s'_1(x_1)$$

$$s'_1(x) = a_1(x - x_2) + a_1(x - x_1) + \frac{y_2 - y_1}{h_1},$$

$$s'_1(x_1) = -a_1h_1 + \frac{y_2 - y_1}{h_1}.$$

Así, la primera ecuación del sistema queda:

$$-a_1 h_1 + \frac{y_2 - y_1}{h_1} = y_1', \tag{5.5.4}$$

donde y_1' es un valor dado conocido de la primera derivada de la función evaluada en x_1. Simplificando la ecuación (5.5.4) obtenemos:

$$a_1 h_1 = \delta_2 - y_1'. \tag{5.5.5}$$

Añadiendo la ecuación (5.5.5) al sistema (5.5.3) obtenemos el nuevo sistema de ecuaciones lineales compatible determinado:

$$\begin{pmatrix} h_1 & 0 & 0 & 0 \dots \dots \dots \dots & 0 & 0 \\ h_1 & h_2 & 0 & 0 \dots \dots \dots \dots \dots & 0 & 0 \\ 0 & h_2 & h_3 & 0 \dots \dots \dots \dots \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & h_{n-2} & h_{n-1} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \cdot \\ \cdot \\ \cdot \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} \delta_2 - y_1' \\ \delta_3 - \delta_2 \\ \cdot \\ \cdot \\ \cdot \\ \delta_n - \delta_{n-1} \end{pmatrix},$$

El sistema es fácilmente resoluble, y nos da los valores de los coeficientes a_i que quedaban por calcular:

$$\begin{cases} a_1 = \dfrac{\delta_2 - y_1'}{h_1}, \\ a_i = \dfrac{1}{h_i}(\delta_{i+1} - \delta_i - a_{i-1} h_{i-1}), \quad i = 2, \dots, n-1. \end{cases}$$

Si se supone la condición de frontera en la frontera derecha:

$$S_{n-1}'(x_n) = y_n' \qquad conocido,$$

operando llegamos a una nueva ecuación que completa el sistema:

$$s_{n-1}'(x_n)$$

$$s_{n-1}'(x) = a_{n-1}(x - x_{n-1}) + a_n(x - x_n) + \frac{y_n - y_{n-1}}{h_{n-1}},$$

$$s_{n-1}'(x_n) = a_{n-1} h_{n-1} + \frac{y_n - y_{n-1}}{h_{n-1}}.$$

Así, la última ecuación del sistema queda:

$$a_{n-1} h_{n-1} + \frac{y_n - y_{n-1}}{h_{n-1}} = y_n', \tag{5.5.6}$$

Donde y'_n es un valor dado conocido de la primera derivada de la función evaluada en x_n. Simplificando la ecuación (5.5.6) obtenemos:

$$a_{n-1}h_{n-1} = y'_n - \delta_n \tag{5.5.7}$$

Añadiendo la ecuación (5.5.7) al sistema (5.5.3) obtenemos el nuevo sistema de ecuaciones lineales compatible determinado:

$$\begin{pmatrix} h_1 & h_2 & 0 & \dots & \dots & \dots & 0 & 0 \\ 0 & h_2 & h_3 & \dots & \dots & \dots & 0 & 0 \\ & & & \vdots & & & & \\ 0 & 0 & 0 & \dots & \dots & \dots & h_{n-2} & h_{n-1} \\ 0 & 0 & 0 & \dots & \dots & \dots & 0 & h_{n-1} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ \vdots \\ h_1 \end{pmatrix} = \begin{pmatrix} \delta_3 - \delta_2 \\ \vdots \\ \vdots \\ \delta_n - \delta_{n-1} \\ y'_n - \delta_n \end{pmatrix},$$

El sistema es fácilmente resoluble, y nos da los valores de los coeficientes a_i que quedaban por calcular:

$$\begin{cases} a_{n-1} = \dfrac{y'_n - \delta_n}{h_1}, \\ a_i = \dfrac{1}{h_i}(\delta_{i+2} - \delta_{i+1} - a_{i+1}h_{i+1}), \quad i = n-2, \dots, 1. \end{cases}$$

A continuación, se muestra la programación en Matlab.

```
function [Si]=Splines2_Interpolant(t,y,CE,S1E,opx,x,opg1)
```

```
% This function builds the cuadratic interpolant spline through the
given two
% dimensional data. Some plots of the spline functions, its first
derivative and
% the interpolated values are displayed. In a separate file the
coefficients of the piecewise polynomials
% of degree 2 are saved.
%
% [Si]=Splines2_Interpolant(t,y,CE,S1E,opx,x,opg1)
%
% Input variables:
% t:abscissae of the nodes
% y:ordinates of the nodes
% CE: indicates if the boundary condition is taken at the left or at
the right extreme of the interval
%     CE='CL' boundary condition at the left extreme
%     CE='CR' boundary condition at the right extreme
% S1E: value of the first derivative at the specified extreme
% opx: parameter that indicates if we need to compute or not
interpolated
% values
%     'y': interpolated values are computed ; 'n': they are not
computed
% x: net of points where we want to evaluate the spline
```

```
% opg1: parameter that indicates if we want to draw the plot with the
% interpolated values
%
% Output variables:
% Si vector that contains the interpolated values at x
%
%
% Example:
% t=[1,2,3,4,6,8,11];
% y=[2,-1,3,1,0,-1,-4];
% CE='CR';
% S1E=0;
% opx='y';
% x=[1:0.2:11];
%
% [Si]=Splines2_Interpolant(t,y,CE,S1E,opx,x,1)




% Number of nodes
m=length(t);

% We check that the entries are correct

if m<3
uiwait(msgbox('The number of control points must be larger than 2',
'Error message',...
'error','modal'));
return;
end


% We define the vector of distances between abcissae
h=diff(t,1);

% Initialization
a=zeros(1,m-1);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Computation of the coefficients a_i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% We compute the delta values

delta(1)=0;
for i=2:m
delta(i)=(y(i)-y(i-1))/h(i-1);
end


% We compute the a_i depending on the boundary condition CE

if strcmp(CE,'CL')
```

```
a(1)=(delta(2)-S1E)/h(1);
for i=2:m-1
a(i)=1/h(i)*(delta(i+1)-delta(i)-a(i-1)*h(i-1));
end

elseif strcmp(CE,'CR')

a(m-1)=(S1E-delta(m))/h(m-1);
for i=m-2:-1:1
a(i)=1/h(i)*(delta(i+2)-delta(i+1)-a(i+1)*h(i+1));
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluation of the spline at x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if opx=='y'

% It finds the polynomial piece to evaluate for each entrie of x
if x(1)==t(1)
minim=y(1); maxim=y(1);
Si(1)=y(1);
else
ind=find((x(1)>t)==0);
ind=ind(1)-1;
ind=ind(1);
% It applies the formule to compute S_ind(x)

        Si(1)=a(ind)*(x(1)-t(ind+1))*(x(1)-t(ind))+delta(ind+1)*(x(1)-
t(ind))+y(ind);
        minim=Si; maxim=Si;
end

for i=2:length(x)
ind=find((x(i)>t)==0);
ind=ind(1)-1;
ind=ind(1);
% It applies the formule to compute S_ind(x)

        Si(i)=a(ind)*(x(i)-t(ind+1))*(x(i)-t(ind))+delta(ind+1)*(x(i)-
t(ind))+y(ind);
if Si(i)<minim
            minim=Si(i);
elseif Si(i)>maxim
            maxim=Si(i);
end
end

if opg1==1
figure(1);
% It draws the control points in the plot
        b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
        hold on;
% It draws the interpolated values in the plot
        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))]);
legend([b1,b2],'Control points','Interpolated values');
        hold off
end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Polynomials S_i(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

syms r;



for i=1:m-1

S(i)=a(i)*(r-t(i+1))*(r-t(i))+delta(i+1)*(r-t(i))+y(i);

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot of the spline
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% It draws the control points in the plot of the spline
figure;
b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
hold on;

% It draws each one of the polynomials
Ms=0;
ms=0;
for i=1:m-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(double([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([b1,b3],'Control points','Cuadratic splines');
title('Approximation by interpolant cuadratic splines');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Plot of the first derivative of the spline
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% It draws each one of the first derivatives
figure;
hold on
Ms=0;
ms=0;
S1=diff(S,1);
for i=1:m-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(double([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)]));
title('First derivative of the interpolant cuadratic splines');



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Output of the results to a file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% It opens or creates a new file to write the interpolated values

if opx=='y'
    fid=fopen('result2_interpolated_values.txt','w');
for i=1:length(x)
fprintf(fid,'%f %f \n',x(i),Si(i));
end
    fclose(fid);
end

%  It opens or creates a new file to write the piecewise polynomials
of the
%  spline
fid=fopen('spline2_polynomials.txt','w');
fprintf(fid,'*******************************************************
****************\n');
hour=clock;
fprintf(fid,'Results obtained at %d-%d-%d,  %d : %d : %d\n',...
    hour(3),hour(2),hour(1),hour(4),hour(5),fix(hour(6)));
fprintf(fid,'*******************************************************
****************\n\n');
signs='----';
for i=1:length(t)-1
    coefi=sym2poly(expand(S(i)));

if length(coefi)==3
```

```
for j=1:3
if coefi(j)>=0
                signs(j)='+';
end
end
fprintf(fid,' %c %f x^2 %c %f x %c %f  \n',...
           signs(1),abs(coefi(1)),signs(2),abs(coefi(2)),signs(3),...
abs(coefi(3)));

elseif length(coefi)==2

for j=1:2
if coefi(j)>=0
                signs(j)='+';
end
end
fprintf(fid,' %c %f x %c %f  \n',...
           signs(1),abs(coefi(1)),signs(2),abs(coefi(2)));

elseif length(coefi)==1

if coefi(1)>=0
signs(1)='+';
end
fprintf(fid,' %c %f  \n',...
signs(1),abs(coefi(1)));

end

end
fprintf(fid,'\n');
fclose(fid);
```

A continuación realizaremos dos ejemplos, uno dando como condición de frontera la derivada en el extremo derecho (ejemplo 1) y otro dando como condición de frontera la primera derivada en el extremo izquierdo (ejemplo 2):

-Ejemplo 1:

```
>> t=[1,2,3,4,6,8,11];

 y=[2,-1,3,1,0,-1,-4];

 CE='CR';

 S1E=0;

 opx='y';

 x=[1:0.2:11];


 [Si]=Splines2_Interpolant(t,y,CE,S1E,opx,x,1)

1.000000 2.000000
```

1.200000 -0.680000

1.400000 -2.320000

1.600000 -2.920000

1.800000 -2.480000

2.000000 -1.000000

2.200000 0.760000

2.400000 2.040000

2.600000 2.840000

2.800000 3.160000

3.000000 3.000000

3.200000 2.600000

3.400000 2.200000

3.600000 1.800000

3.800000 1.400000

4.000000 1.000000

4.200000 0.630000

4.400000 0.320000

4.600000 0.070000

4.800000 -0.120000

5.000000 -0.250000

5.200000 -0.320000

5.400000 -0.330000

5.600000 -0.280000

5.800000 -0.170000

6.000000 0.000000

6.200000 0.170000

6.400000 0.280000

6.600000 0.330000

6.800000 0.320000

7.000000 0.250000

7.200000 0.120000

7.400000 -0.070000

7.600000 -0.320000

7.800000 -0.630000

8.000000 -1.000000

8.200000 -1.386667

8.400000 -1.746667

8.600000 -2.080000

8.800000 -2.386667

9.000000 -2.666667

9.200000 -2.920000

9.400000 -3.146667

9.600000 -3.346667

9.800000 -3.520000

10.000000 -3.666667

10.200000 -3.786667

10.400000 -3.880000

10.600000 -3.946667

10.800000 -3.986667

11.000000 -4.000000

****

Results obtained at 16-1-2020, 10 : 16 : 44

****

+ 13.000000 x^2 - 42.000000 x + 31.000000

+ 6.000000 x^2 + 34.000000 x + 45.000000

+ 2.000000 x + 9.000000

+ 0.750000 x^2 + 8.000000 x + 21.000000

+ 0.750000 x^2 + 10.000000 x + 33.000000

+ 0.333333 x^2 + 7.333333 x + 36.333333

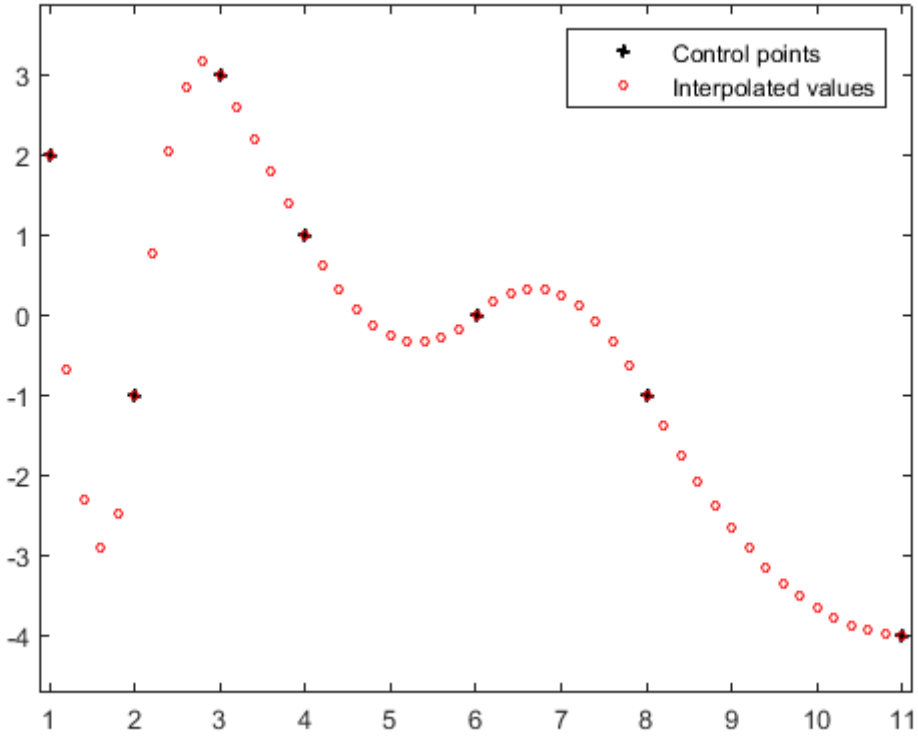


Fig 5.6.1: valores interpolados para las abscisas indicadas.

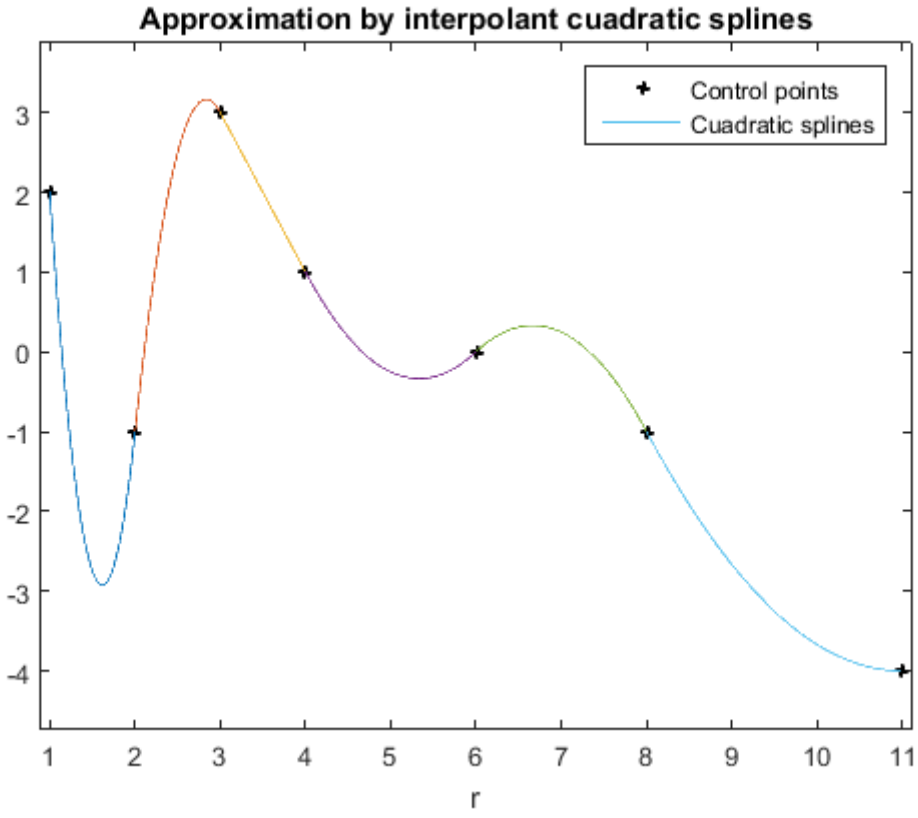


Fig 5.6.2: aproximación mediante splines cuadráticos interpolantes usando como condición de frontera la primera derivada en el extremo derecho.

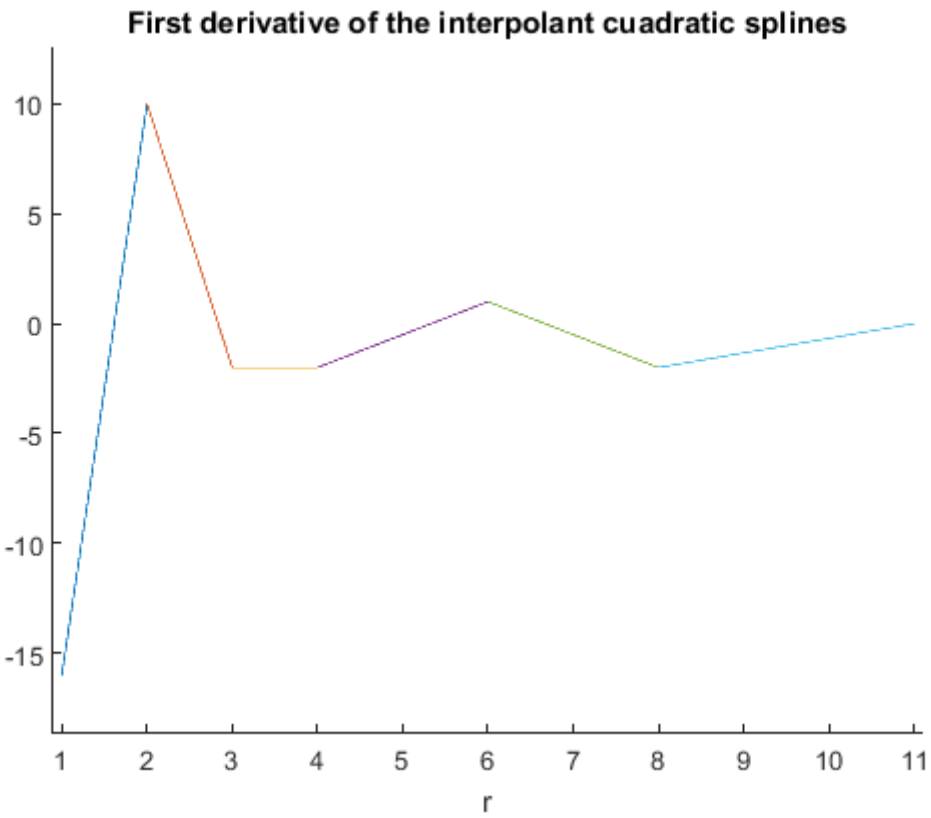


Fig 5.6.3: primera derivada del spline cuadrático interpolante.

Ejemplo 2:

```
>>  t=[1,2,3,4,6,8,11];
 y=[2,-1,3,1,0,-1,-4];
 CE='CL';
 S1E=0;
 opx='y';
 x=[1:0.2:11];

 [Si]=Splines2_Interpolant(t,y,CE,S1E,opx,x,1)

1.000000 2.000000

1.200000 1.880000

1.400000 1.520000
```

1.600000 0.920000

1.800000 0.080000

2.000000 -1.000000

2.200000 -1.800000

2.400000 -1.800000

2.600000 -1.000000

2.800000 0.600000

3.000000 3.000000

3.200000 5.160000

3.400000 6.040000

3.600000 5.640000

3.800000 3.960000

4.000000 1.000000

4.200000 -2.250000

4.400000 -4.800000

4.600000 -6.650000

4.800000 -7.800000

5.000000 -8.250000

5.200000 -8.000000

5.400000 -7.050000

5.600000 -5.400000

5.800000 -3.050000

6.000000 0.000000

6.200000 3.050000

6.400000 5.400000

6.600000 7.050000

6.800000 8.000000

7.000000 8.250000

7.200000 7.800000

7.400000 6.650000

7.600000 4.800000

7.800000 2.250000

8.000000 -1.000000

8.200000 -4.373333

8.400000 -7.293333

8.600000 -9.760000

8.800000 -11.773333

9.000000 -13.333333

9.200000 -14.440000

9.400000 -15.093333

9.600000 -15.293333

9.800000 -15.040000

10.000000 -14.333333

10.200000 -13.173333

10.400000 -11.560000

10.600000 -9.493333

10.800000 -6.973333

11.000000 -4.000000

****

Results obtained at 16-1-2020, 10 : 46 : 46

****

- 3.000000 x^2 + 6.000000 x - 1.000000

+ 10.000000 x^2 + 46.000000 x + 51.000000

+ 16.000000 x^2 + 110.000000 x + 183.000000

+ 8.750000 x^2 + 88.000000 x + 213.000000

+ 8.750000 x^2 + 122.000000 x + 417.000000

+ 5.666667 x^2 + 108.666667 x + 505.666667

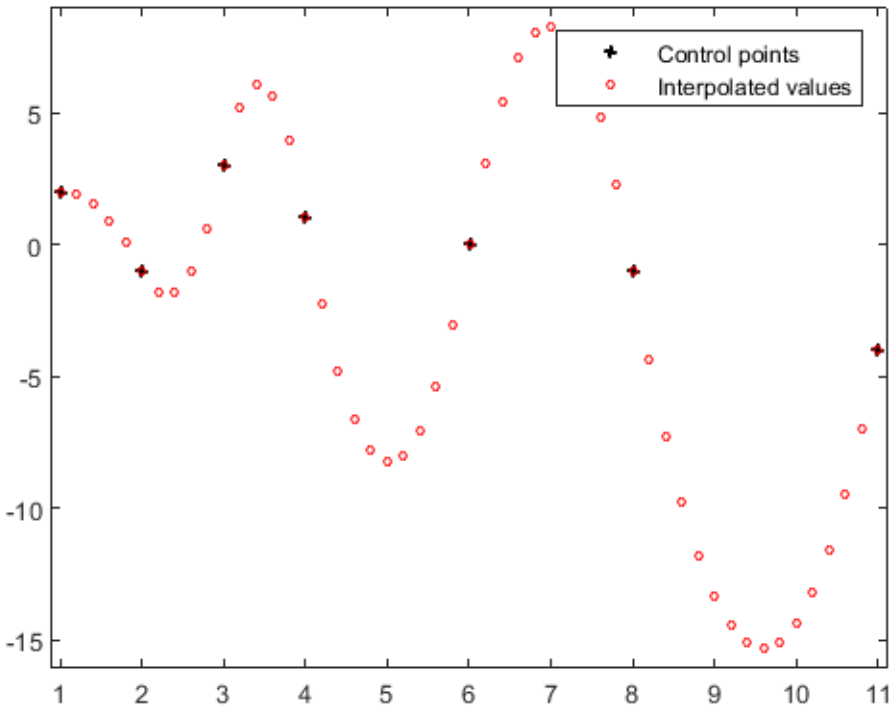


Fig 5.6.4: valores interpolados para las abscisas indicadas.

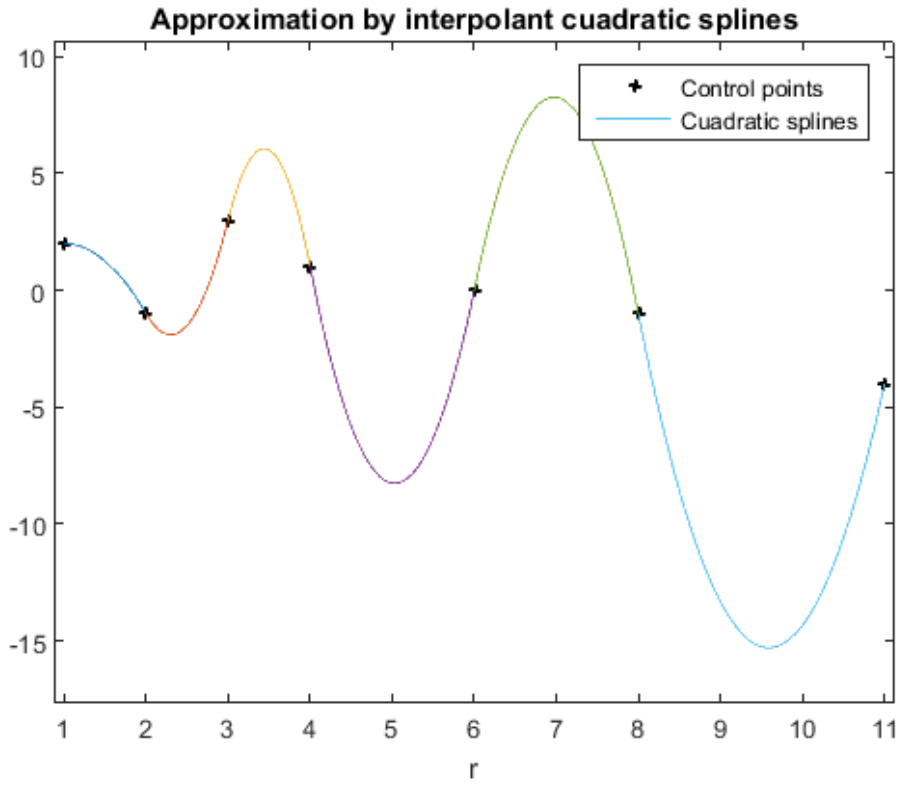


Fig 5.6.5: aproximación mediante splines cuadráticos interpolantes usando como condición de frontera la primera derivada en el extremo izquierdo.

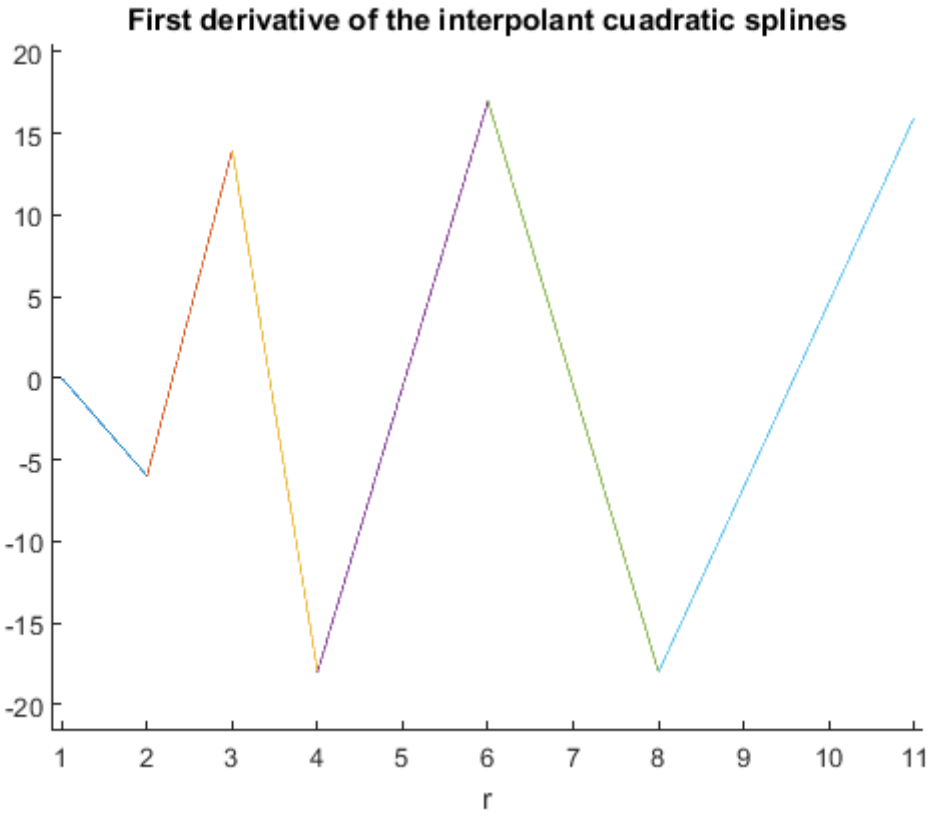


Fig 5.6.6: primera derivada del spline cuadrático interpolante.

5.6 DEFINICIÓN DE SPLINE CÚBICO

Se llama spline cúbico interpolante $S(x)$ asociado a un vector de nodos $a = x_1 < x_2 < \cdots < x_n$=b a una función que:

1) En cada uno de los intervalos $[x_i, x_{i+1}]$ $\forall\, i = 1, \dots, n-1$ es un polinomio de tercer grado,

$$S(x) = s_i(x) = a_0^{(i)} + a_1^{(i)}(x - x_i) + a_2^{(i)}(x - x_i)^2 + a_3^{(i)}(x - x_i)^3.$$

Como sabemos, el spline es un polinomio de tercer grado que se define a través de cuatro coeficientes durante todo el intervalo. Tenemos n-1 intervalos, por lo tanto necesitaremos hallar 4(n-1) números para poder definir el spline completamente

$$a_0^{(i)}, a_1^{(i)}, a_2^{(i)}, a_3^{(i)}, \quad \forall\, i = 1, \dots, n-1.$$

2) Es dos veces diferenciable con continuidad en el intervalo [a,b], perteneciendo así a la clase $C^2[a, b]$.

Con esta condición lo que tenemos es continuidad en la función $S(x)$ y sus derivadas primera y segunda $S'(x)$ y $S''(x)$ en todos los nodos internos del retículo. En el resto del intervalo es clara la continuidad debido a ser una función polinómica a trozos. Como el número de nodos internos es n-2, tenemos 3(n-2) condiciones más.

3) Satisface las condiciones

$$S(x_i) = f(x_i) = y_i \quad \forall\, i = 1, \dots, n.$$

Junto con las condiciones de continuidad, tenemos $3(\mathrm{n}-2)+\mathrm{n} = 4\mathrm{n}-6$ condiciones (ecuaciones).

Nos faltan dos ecuaciones, para tener las $4(n-1) = 4n-4$ condiciones necesarias que definen el spline. Éstas las formularemos como restricciones sobre los valores del spline y/o sus derivadas en los extremos del intervalo [a, b].

Normalmente para construir el spline cúbico se pueden utilizar cuatro tipos de condiciones:

1- Primer tipo: se dan los valores que ha de tener la derivada primera de $S(x)$ en los extremos del intervalo [a,b]:

$$S'(a) = f'(a)\,;\;\; S'(b) = f'(b).$$

2- Segundo tipo: se dan los valores que tiene que tener la derivada segunda de $S(x)$ en los extremos del intervalo [a,b]

$$S''(a) = f''(a)\,;\;\; S''(b) = f''(b).$$

3- Tercer tipo: condiciones periódicas:

$$S'(a) = S'(b);\;\; S''(a) = S''(b).$$

Es normal imponer esta condición cuando la función a interpolar es periódica de periodo $T = b - a$.

4- Cuarto tipo:

$$S'''(y, t_2 - 0) = S'''(y, t_2 + 0),$$

$$S'''(y, t_{n-1} - 0) = S'''(y, t_{n-1} + 0).$$

Por lo general la derivada tercera de $S(x)$ es discontinua en los puntos interiores del retículo. Gracias a este tipo de condiciones podemos tener un número de puntos de discontinuidad reducido. En este supuesto, el spline que obtenemos es tres veces diferenciable con continuidad en los puntos t_2 y t_{n-1} acercándose por la izquierda (-0) o por la derecha (+0).

Uno de los problemas más importantes en la interpolación es la elección de las condiciones de contorno, adquiriendo especial importancia a la hora de garantizar que el spline $S(x)$ tenga una precisión alta en las proximidades de los extremos del intervalo [a,b]. Estas condiciones tienen una influencia notable en cómo se comporta el spline cerca de los extremos, la cual se va debilitando conforme nos alejamos de ellos. Para elegir las condiciones de contorno, vamos a depender normalmente de si conocemos datos adicionales acerca del comportamiento del spline.

Si en los extremos del intervalo conocemos los valores de la derivada primera, es decir, la dirección de la tangente de la curva en los extremos, será lógico que utilicemos las condiciones de contorno de primer tipo. Sin embargo si lo que conocemos son los valores de la segunda derivada, lo lógico será utilizar las condiciones de contorno de segundo tipo.

Si podemos elegir entre condiciones de primer tipo o de segundo, tendrán preferencia las de primer tipo y si la función fuese periódica, deberemos elegir condiciones de contorno de tercer tipo.

En el supuesto de no tener información adicional del comportamiento de la función, podemos utilizar las condiciones naturales de contorno:

$$S''(a) = 0 \ ; \ S''(b) = 0.$$

Es posible que con estas condiciones la precisión en la aproximación disminuya notablemente en las proximidades de los extremos del intervalo [a, b].

También existe la opción de utilizar condiciones de contorno de primer o segundo tipo con valores aproximados, es decir, se tomarán sus aproximaciones en diferencias y no los valores exactos de sus derivadas.

Normalmente el uso del cuarto tipo de condiciones de contorno da buenas aproximaciones.

Si la tercera derivada $f'''(x)$ presenta discontinuidades en algún punto del intervalo, incluiremos estos puntos entre los nodos de interpolación con el objetivo de mejorar la interpolación.

Si la segunda derivada $f''(x)$ fuese discontinua, tendremos que tomar ciertas medidas para evitar oscilaciones del spline cerca de los puntos de discontinuidad. Normalmente, se eligen los nodos de interpolación de forma que los puntos con discontinuidad de la segunda derivada estén contenidos en un intervalo $[x_i, \ x_{i+1}]$ tal que: $h_i = \propto min\{h_{i-1}, h_{i+1}\}$ donde $\propto \ll 1. \propto$ se puede elegir empíricamente, pero normalmente es suficiente con tomar $\propto = 0.01$.

Si la discontinuidad se da en la primera derivada $f'(x)$, podemos utilizar distintas técnicas para superar las dificultades que surgen. Una de las más simples es partir el intervalo de aproximación en intervalos de continuidad de la derivada y en cada uno de ellos construir un spline **[3].**

5.6.1 CONSTRUCCIÓN DEL SPLINE CÚBICO INTERPOLANTE

Interpolaremos mediante un spline cúbico interpolante una función $f(x)$ sobre unos nodos $x_i, \quad \forall\, i = 1, \dots, n$, siendo $x_1 = a \;\; y \;\; x_n = \; b$. Denotamos por $y_i = f(x_i)$ los valores conocidos de la función en los nodos. Denominamos $h_i = x_{i+1} - x_i$ a los diferentes espaciados entre los nodos. Sea $s_i(x)$ la restricción de la función spline $S(x)$ al intervalo $[x_i, x_{i+1}], \;\; \forall \;\; i = 1, \dots, n-1$. Como la función $S(x)$ está definida en [a, b] como: $S(x) = \; s_i(x), \;\; si \; x \; \in \; [x_i, x_{i+1}]$, bastará con conocer cada uno de los trozos cúbicos $s_i(x)$.

Sabemos que $s_i(x)$ es un polinomio cúbico que satisface las condiciones:

$$s_i(x) = \; y_i \;\; \forall \;\; i = 1, \dots, n.$$

Y la condición que asegura que $s_i(x) \;\; \in \;\; C^2[a, b]$ $(clase\; C2)$, establece:

$$s_i'(x_i) = \; s_{i+1}'(x_i) \;\; \forall \;\; i = 1, \dots, n-1,$$

$$s_i''(x_i) = \; s_{i+1}''(x_i) \;\; \forall \;\; i = 1, \dots, n-1.$$

Al ser $s_i(x)$ un polinomio de grado tres, su primera derivada será un polinomio de grado dos y su segunda derivada será un polinomio de grado uno.

Si denotamos por $z_i \;\forall \;\; i = 1, \dots, n$ los valores que toma $s_i''(x_i)$ para cada valor de i, tendremos que para que las segundas derivadas de la función enganchen bien entre los distintos trozos de la función $S''(x)$, ésta tendrá que ser de la siguiente forma, tal y como se ve en la figura 5.6.1.1:

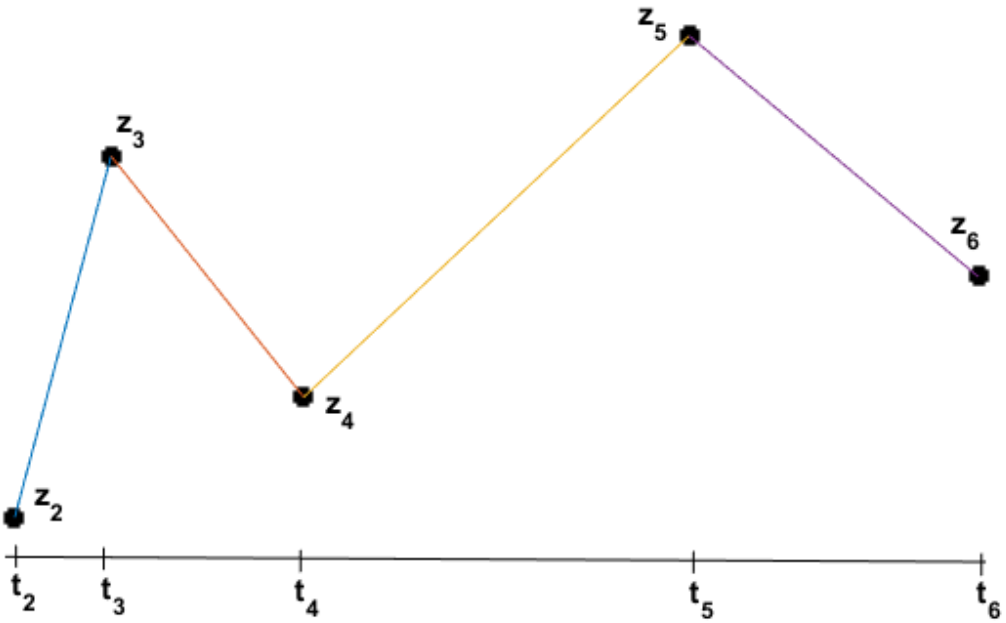


Fig 5.6.1.1: Gráfica general del spline cúbico interpolante.

Para que sea así, construiremos cada trozo lineal $s_i''(x)$ de manera que $s_i''(x_i) = z_i$ y $s_i''(x_{i+1}) = z_{i+1}$. Usando el método de interpolación de Lagrange tenemos que:

$$s_i''(x) = z_{i+1}\frac{x - x_i}{h_i} + z_i\frac{x_{i+1} - x}{h_i}.$$

Integrando dos veces esta función, obtenemos:

$$s_i(x) = \frac{z_{i+1}}{6h_i}(x - x_i)^3 + \frac{z_i}{6h_i}(x_{i+1} - x)^3 + C_i(x - x_i) + D_i(x_{i+1} - x).$$

De las condiciones $s_i(x_i) = y_i \ \forall\ i = 1, \dots, n$ sabemos que: $s_i(x_i) = y_i$, $s_i(x_{i+1}) = y_{i+1}$, y a partir de estas relaciones podemos deducir los valores de C_i y D_i, llegando a la siguiente expresión:

$$s_i(x) = \frac{z_{i+1}}{6h_i}(x - x_i)^3 + \frac{z_i}{6h_i}(x_{i+1} - x)^3 + \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right)(x - x_i) + \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right)(x_{i+1} - x).$$

Con esta expresión de $s_i(x)$ para cada intervalo $[x_i, x_{i+1}]$ garantizamos la continuidad de $S(x)$ y coincidirá con $f(x_i)$ en los puntos $x_i\ \forall\ i = 1, \dots, n$.

Nos falta satisfacer la continuidad de la primera derivada de $S(x)$en los puntos interiores:

$$s_i'(x_i) = s_{i-1}'(x_i)\ \forall\ i = 2, \dots, n-1.$$

Calculamos $s_i'(x_i)$ y $s_{i-1}'(x_i)$ a partir de:

$$s_i'(x) = \frac{z_{i+1}}{2h_i}(x - x_i)^2 - \frac{z_i}{2h_i}(x_{i+1} - x)^2 + \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right) - \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right),$$

$$s_{i-1}'(x) = \frac{z_i}{2h_i}(x - x_{i-1})^2 - \frac{z_{i-1}}{2h_{i-1}}(x_i - x)^2 + \left(\frac{y_i}{h_{i-1}} - \frac{z_i h_{i-1}}{6}\right) - \left(\frac{y_{i-1}}{h_{i-1}} - \frac{z_{i-1}h_{i-1}}{6}\right).$$

Imponiendo que $s_i'(x_i) = s_{i-1}'(x_i)$, simplificando la expresión y teniendo en cuenta que $i = 2, \dots, n-1$, llegamos al sistema de ecuaciones lineales:

$$h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}).$$

Si definimos:

$$b_i = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}) \quad \forall\, i = 2, \dots, n-1,$$

queda el siguiente sistema:

$$\begin{cases} h_1 z_1 + 2\,(h_1 + h_2) z_2 + h_2 z_3 = b_2 \\ h_2 z_2 + 2\,(h_2 + h_3) z_3 + h_3 z_4 = b_3 \\ \dots \\ h_{n-2} z_{n-2} + 2\,(h_{n-2} + h_{n-1}) z_{n-1} + h_{n-1} z_n = b_{n-1}. \end{cases}$$

Se trata de $n-2$ ecuaciones lineales con n incógnitas quedándonos por tanto dos variables libres. Para hallarlas tenemos que aplicar las condiciones de contorno.

Como se ha explicado anteriormente, escogeremos las condiciones de contorno en función de los datos adicionales que conozcamos sobre el comportamiento de la función $f(x)$. Ahora detallaremos la construcción de las dos ecuaciones que faltan para resolver el sistema, en función del tipo de condiciones de contorno que utilicemos.

1) Condiciones de primer tipo, es decir, conocemos los valores de la primera derivada de $S(x)$ en los extremos del intervalo [a, b]; obtendremos las ecuaciones de la siguiente manera:
   - Obtenemos la derivada del trozo de spline correspondiente al extremo.
   - Evaluamos esta ecuación en el punto extremo.
   - Lo igualamos al valor conocido de la primera derivada.

   - Para el extremo izquierdo a:

$$s_1'(x_1) = \frac{z_1}{2h_1}(x_2 - x_1)^2 + \frac{y_2}{h_1} - \frac{z_2 h_1}{6} - \frac{y_1}{h_1} + \frac{z_1 h_1}{6} = f'(a).$$

   Ordenando componentes:

$$-\frac{h_1}{3} z_1 - \frac{h_1}{6} z_2 = f'(a) - \frac{y_2 - y_1}{h_1}.$$

   - Para el extremo derecho b:

$$s_{n-1}'(x_n) = \frac{z_n}{2h_{n-1}}(x_n - x_{n-1})^2 + \frac{y_n}{h_{n-1}} - \frac{z_n h_{n-1}}{6} - \frac{y_{n-1}}{h_{n-1}} + \frac{z_{n-1} h_{n-1}}{6} = f'(b).$$

   Ordenando componentes:

$$-\frac{h_{n-1}}{3}z_n - \frac{h_{n-1}}{6}z_{n-1} = f'(b) - \frac{y_n - y_{n-1}}{h_{n-1}}.$$

2) Para condiciones de segundo tipo, es decir, damos los valores de la segunda derivada en los extremos del intervalo [a,b], es decir, se han de dar los valores z_1 y z_n:

$$f''(a) = z_1; \quad f''(b) = z_n.$$

3) En el supuesto de que la función sea periódica con periodo T= b-a, en particular, $f(a) = f(b)$ y por tanto $s_1(a) = s_{n-1}(b)$, con lo cual se usarán condiciones de tercer tipo llegando a las siguientes ecuaciones:

- $s_1'(x_1) = s_{n-1}'(n)$,

$$\frac{h_1}{3}z_1 + \frac{h_1}{6}z_2 + \frac{h_{n-1}}{6}z_{n-1} + \frac{h_{n-1}}{3}z_n = \frac{y_{n-1} - y_n}{h_{n-1}} + \frac{y_2 - y_1}{h_1}.$$

- $s_1''(x_1) = s_{n-1}''(x_n)$,

$$z_1 - z_n = 0.$$

4) Si lo que queremos es utilizar condiciones de cuarto tipo, que en los extremos del intervalo también tengan continuidad en la tercera derivada de $S(x)$, de manera que $s_1 = s_2$, $s_{n-2} = s_{n-1}$ sabiendo que satisface $s_i(x) \in C^2[a,b]$; lo que debemos hacer es igualar $s_{i-1}'''(x_i)$ a $s_i'''(x_i)$, $i = 2, n-1$. Los splines que se construyen de esta manera se suelen llamar splines no nodo.

Sabiendo que $s_i''(x)$ es una recta la cual pasa por los puntos (x_{i-1}, z_{i-1}) y (x_i, z_i), su pendiente y su derivada será un coeficiente. De este modo:

$$\frac{z_{i+1} - z_i}{x_i - x_{i-1}} = \frac{z_{i+1} - z_i}{h_i}.$$

- En el extremo izquierdo:

$$s_1'''(x_2) = s_2'''(x_2),$$

$$\frac{z_2 - z_1}{h_1} = \frac{z_3 - z_2}{h_2}.$$

Ordenando componentes:

$$-h_2 z_1 + (h_2 + h_1)z_2 - h_1 z_3 = 0.$$

- En el extremo derecho:

$$s'''_{n-1}(x_{n-1}) = s'''_{n-2}(x_{n-1}),$$

$$\frac{z_n - z_{n-1}}{h_{n-1}} = \frac{z_n - z_{n-2}}{h_{n-2}}.$$

Ordenando componentes:

$$-h_{n-1}z_{n-2} + (h_{n-1} + h_{n-2})z_{n-1} - h_{n-2}z_n = 0.$$

Las condiciones que podemos tener en cada extremo pueden ser diferentes, ya que no siempre conocemos los mismos datos en ambos extremos del intervalo [a,b]. Únicamente en las condiciones de tercer tipo que exigen que la función sea periódica de periodo T=b-a y en particular $f(a) = f(b)$ se impone el mismo tipo de condición en la frontera forzosamente.

Cabe recordar por último que las condiciones de primer tipo (primera derivada conocida) tienen preferencia sobre las de segundo tipo (segunda derivada conocida) y en el caso de que no se conozcan los valores de las derivadas, podemos aproximar. En el caso de no poder aproximar, se podría asumir que $z_1 = z_n = 0$, en contra de la precisión de aproximación en el supuesto de que la derivada segunda no fuese nula. A estos splines se les denomina splines naturales. Normalmente se obtienen buenos resultados si utilizamos condiciones de cuarto tipo cuando desconocemos las demás.

A continuación, se muestra la programación en Matlab

```
function
[Si]=Splines3_Interpolant(t,y,CL,S1a,S2a,CR,S1b,S2b,opx,x,opg1)



% This function builds the interpolant spline through the given two
% dimensional data. Some plots of the spline functions, its first and
second derivatives and
% the interpolated values are displayed. In a separate file the
coefficients of the piecewise polynomials
% of degree 3 are saved.
%
% [Si]=Splines3_Interpolant(t,y,CL,S1a,S2a,CR,S1b,S2b,opx,x,opg1)
%
% Input variables:
% t:abscissae of the nodes
% y:ordinates of the nodes
% CL: boundary condition at the left of the interval
% CR: boundary condition at the right of the interval
%
% Possible boundary conditions:
% 1- First type: values of the first derivative
% 2- Second type: values of the second derivative
% 3- Third type: periodic conditions with period T=b-a
%    S'(a)=S'(b); S''(a)=S''(b). If CL=3->CR=3
% 4- Fourth type: Three times differentiable at the points t2 and/or
tm-1
```

```
%
% S1a: value of the first derivative at the left extreme
% S2a: value of the second derivative at the left extreme
% S1b: value of the first derivative at the right extreme
% S2b: value of the second derivative at the right extreme
%
% opx: parameter that indicates if we need to compute or not
interpolated
% values
% 'y': interpolated values are computed ; 'n': they are not computed
% x: net of points where we want to evaluate the spline
% opg1: parameter that indicates if we want to draw the plot with the
% interpolated values
%
% Output variables:
% Si vector that contains the interpolated values at x
%
%
% Example:
% t=[1,2,3,4,6,8,11];
% y=[2,-1,3,1,0,-1,-4];
% CL=2 ; CR=2;
% S1a=0; S1b=0;
% S2a=0; S2b=0;
% opx='y';
% x=[1:0.2:11];
%
% [Si]=Splines3_Interpolant(t,y,CL,S1a,S2a,CR,S1b,S2b,opx,x,1)




% Number of nodes
m=length(t);

% We check that the entries are correct

if m<3
uiwait(msgbox('The number of control points must be larger than 2',
'Error message',...
'error','modal'));
return;
elseif CL==0 | CR==0
uiwait(msgbox('It lacks to specify at least one boundary condition',
'Error message',...
'error','modal'));
return;
elseif CL==3 & CR~=3
uiwait(msgbox('If CL=3 then CR must be also 3', 'Error message',...
'error','modal'));
return;
elseif CL==3 & CR~=3
uiwait(msgbox('If CR=3 then CL must be also 3', 'Error message',...
'error','modal'));
return;
elseif CR==3 & y(1)~=y(m)
uiwait(msgbox('For the function to be periodic it must take the same
value at the boundaries', 'Mensaje de error',...
'error','modal'));
```

```
return;
elseif m==3 & CL==4 & CR==4
uiwait(msgbox('With 3 nodes the condition CL=4 already implies CR=4
and we need one more condition', 'Mensaje de error',...
'error','modal'));
return;
end


% We define the vector of distances between abcissae
h=diff(t,1);

% Initialization
z=zeros(1,m);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Computation of the second derivatives
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% We define the coefficient matrix of the linear system
A=zeros(m,m);
% We define the right hand side of the system
B=zeros(m,1);



% We define the first equation of the system depending on CL

if CL==1
A(1,1)=-h(1)/3;
A(1,2)=-h(1)/6;
B(1)=S1a+(y(1)-y(2))/h(1);
elseif CL==2
B(1)=S2a;
A(1,1)=1;
elseif CL==3
% first equation
    A(1,1)=h(1)/3; A(1,2)=h(1)/6; A(1,m-1)=h(m-1)/6; A(1,m)=h(m-1)/3;
B(1)=(y(m-1)-y(m))/h(m-1)+(y(2)-y(1))/h(1);
% last equation
A(m,1)=1; A(m,m)=-1;
elseif CL==4
A(1,1)=-h(2); A(1,2)=h(1)+h(2); A(1,3)=-h(1);
end


% We define the m-2 intermidiate equations


for i=2:m-1
A(i,i-1)=h(i-1);
A(i,i)=2*(h(i-1)+h(i));
A(i,i+1)=h(i);
end

% Now we define the matrix B
```

```
for i=2:m-1
B(i)=6/h(i)*(y(i+1)-y(i))-6/h(i-1)*(y(i)-y(i-1));
end



% We define the last equation of the system depending on CD

if CR==1
A(m,m-1)=h(m-1)/6; A(m,m)=h(m-1)/3;
B(m)=S1b+(y(m-1)-y(m))/h(m-1);
elseif CR==2
    B(m)=S2b;
A(m,m)=1;
elseif CR==4
    A(m,m-2)=-h(m-1); A(m,m-1)=h(m-1)+h(m-2); A(m,m)=-h(m-2);
end


% Solution of the system

z(1:m)=A\B;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluation of the spline at x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if opx=='y'

% It finds the polynomial piece to evaluate for each entrie of x
if x(1)==t(1)
minim=y(1); maxim=y(1);
Si(1)=y(1);
else
ind=find((x(1)>t)==0);
ind=ind(1)-1;
ind=ind(1);
% It applies the formule to compute S_ind(x)

Si(1)=z(ind+1)/(6*h(ind))*(x(1)-t(ind))^3+...
            z(ind)/(6*h(ind))*(t(ind+1)-x(1))^3+(y(ind+1)/h(ind)-...
z(ind+1)*h(ind)/6)*(x(1)-t(ind))+(y(ind)/h(ind)-...
z(ind)*h(ind)/6)*(t(ind+1)-x(1));
        minim=Si; maxim=Si;
end

for i=2:length(x)
ind=find((x(i)>t)==0);
ind=ind(1)-1;
ind=ind(1);
% It applies the formule to compute S_ind(x)

Si(i)=z(ind+1)/(6*h(ind))*(x(i)-t(ind))^3+...
            z(ind)/(6*h(ind))*(t(ind+1)-x(i))^3+(y(ind+1)/h(ind)-...
z(ind+1)*h(ind)/6)*(x(i)-t(ind))+(y(ind)/h(ind)-...
z(ind)*h(ind)/6)*(t(ind+1)-x(i));
```

```
if Si(i)<minim
            minim=Si(i);
elseif Si(i)>maxim
            maxim=Si(i);
end
end

if opg1==1
figure(1);
% It draws the control points in the plot
        b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
        hold on;
% It draws the interpolated values in the plot
        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],'Control points','Interpolated values');
        hold off
end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Polynomials S_i(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

syms r;



for i=1:m-1
    S(i)=z(i+1)/(6*h(i))*(r-t(i))^3+z(i)/(6*h(i))*(t(i+1)-r)^3+...
        (y(i+1)/h(i)-z(i+1)*h(i)/6)*(r-t(i))+(y(i)/h(i)-...
z(i)*h(i)/6)*(t(i+1)-r);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot of the spline
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% It draws the control points in the plot of the spline
figure;
b1=plot(t,y,'ko','MarkerSize',2,'LineWidth',3);
hold on;

% It draws each one of the polynomials
Ms=0;
ms=0;
for i=1:m-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(double([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],'Control points','Cubic splines');
title('Approximation by interpolant cubic splines');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot of the first derivative of the spline
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% It draws each one of the first derivatives
figure;
hold on
Ms=0;
ms=0;
S1=diff(S,1);
for i=1:m-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(double([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)]));
title('First derivative of the interpolant cubic splines');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot of the second derivative of the spline
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% It draws each one of the second derivative
figure;
hold on
S2=diff(S,2);
Msd=subs(S2(1),t(1));
msd=subs(S2(1),t(1));
str=[];
for i=1:m-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(double([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)]));
title('Second derivative of the interpolant cubic splines');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Output of the results to a file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% It opens or creates a new file to write the interpolated values

if opx=='y'
    fid=fopen('result3_interpolated_values.txt','w');
for i=1:length(x)
fprintf(fid,'%f %f \n',x(i),Si(i));
end
    fclose(fid);
end

%  It opens or creates a new file to write the piecewise polynomials
of the
%  spline
fid=fopen('spline3_polynomials.txt','w');
fprintf(fid,'************************************************************
****************\n');
hour=clock;
fprintf(fid,'Results obtained at %d-%d-%d,  %d : %d : %d\n',...
    hour(3),hour(2),hour(1),hour(4),hour(5),fix(hour(6)));
fprintf(fid,'************************************************************
****************\n\n');
signs='----';
for i=1:length(t)-1
    coefi=sym2poly(expand(S(i)));

if length(coefi)==4

for j=1:4
if coefi(j)>=0
                signs(j)='+';
end
end
fprintf(fid,' %c %f x^3 %c %f x^2 %c %f x %c %f  \n',...
           signs(1),abs(coefi(1)),signs(2),abs(coefi(2)),signs(3),...
abs(coefi(3)),signs(4),abs(coefi(4)));

elseif length(coefi)==3

for j=1:3
if coefi(j)>=0
                signs(j)='+';
end
end
```

```
fprintf(fid,' %c %f x^2 %c %f x %c %f  \n',...
            signs(1),abs(coefi(1)),signs(2),abs(coefi(2)),signs(3),...
abs(coefi(3)));

elseif length(coefi)==2

for j=1:2
if coefi(j)>=0
                signs(j)='+';
end
end
fprintf(fid,' %c %f x %c %f  \n',...
            signs(1),abs(coefi(1)),signs(2),abs(coefi(2)));

elseif length(coefi)==1

if coefi(1)>=0
signs(1)='+';
end
fprintf(fid,' %c %f  \n',...
signs(1),abs(coefi(1)));

end



end
fprintf(fid,'\n');
fclose(fid);
```

Ejemplo:

```
% t=[1,2,3,4,6,8,11];
% y=[2,-1,3,1,0,-1,-4];
% CL=2 ; CR=2;
% S1a=0; S1b=0;
% S2a=0; S2b=0;
% opx='y';
% x=[1:0.2:11];
% [Si]=Splines3_Interpolant(t,y,CL,S1a,S2a,CR,S1b,S2b,opx,x,1)
```

1.000000 2.000000

1.200000 0.956097

1.400000 0.023170

1.600000 -0.687806

1.800000 -1.065854

2.000000 -1.000000

2.200000 -0.434243

2.400000 0.467515

2.600000 1.486394

2.800000 2.403515

3.000000 3.000000

3.200000 3.116873

3.400000 2.834771

3.600000 2.294232

3.800000 1.635795

4.000000 1.000000

4.200000 0.500580

4.400000 0.144043

4.600000 -0.089906

4.800000 -0.221563

5.000000 -0.271226

5.200000 -0.259191

5.400000 -0.205755

5.600000 -0.131213

5.800000 -0.055863

6.000000 0.000000

6.200000 0.020057

6.400000 0.003903

6.600000 -0.044887

6.800000 -0.122739

7.000000 -0.226078

7.200000 -0.351332

7.400000 -0.494925

7.600000 -0.653283

7.800000 -0.822833

8.000000 -1.000000

8.200000 -1.181761

8.400000 -1.367296

8.600000 -1.556334

8.800000 -1.748607

9.000000 -1.943845

9.200000 -2.141779

9.400000 -2.342138

9.600000 -2.544654

9.800000 -2.749057

10.000000 -2.955076

10.200000 -3.162444

10.400000 -3.370889

10.600000 -3.580144

10.800000 -3.789937

11.000000 -4.000000

****

Results obtained at 16-1-2020, 10 : 2 : 44

****

+ 2.311995 x^3 - 6.935984 x^2 + 1.623989 x + 5.000000

+ 4.559973 x^3 + 34.295822 x^2 + 80.839623 x + 59.975741

+ 2.927898 x^3 + 33.095013 x^2 + 121.332884 x + 142.196765

+ 0.422844 x^3 + 7.113881 x^2 + 39.502695 x + 72.250674

+ 0.074461 x^3 + 1.837601 x^2 + 14.206199 x + 35.167116

+ 0.005615 x^3 + 0.185310 x^2 + 0.987871 x + 0.081761

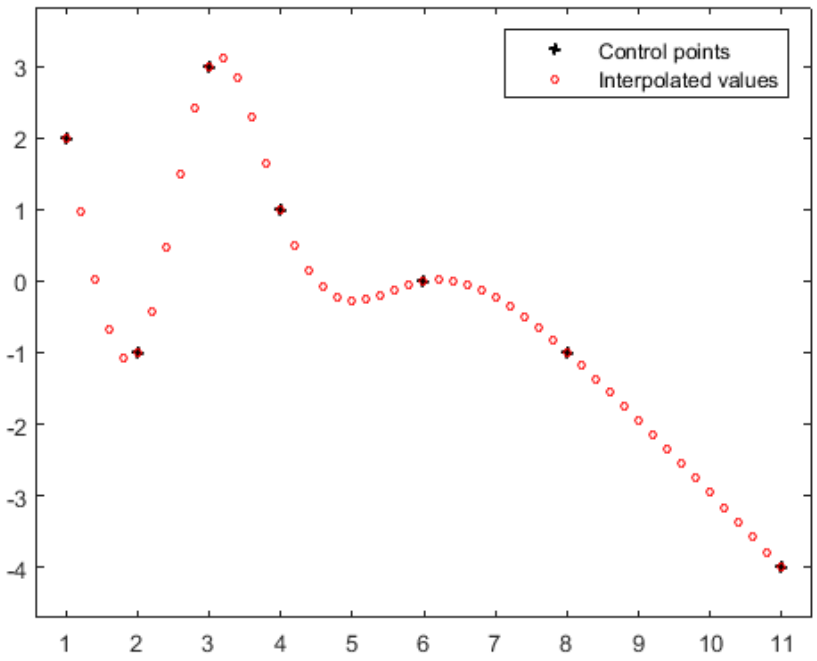


Fig 5.6.1.2: valores interpolados para las abscisas indicadas.

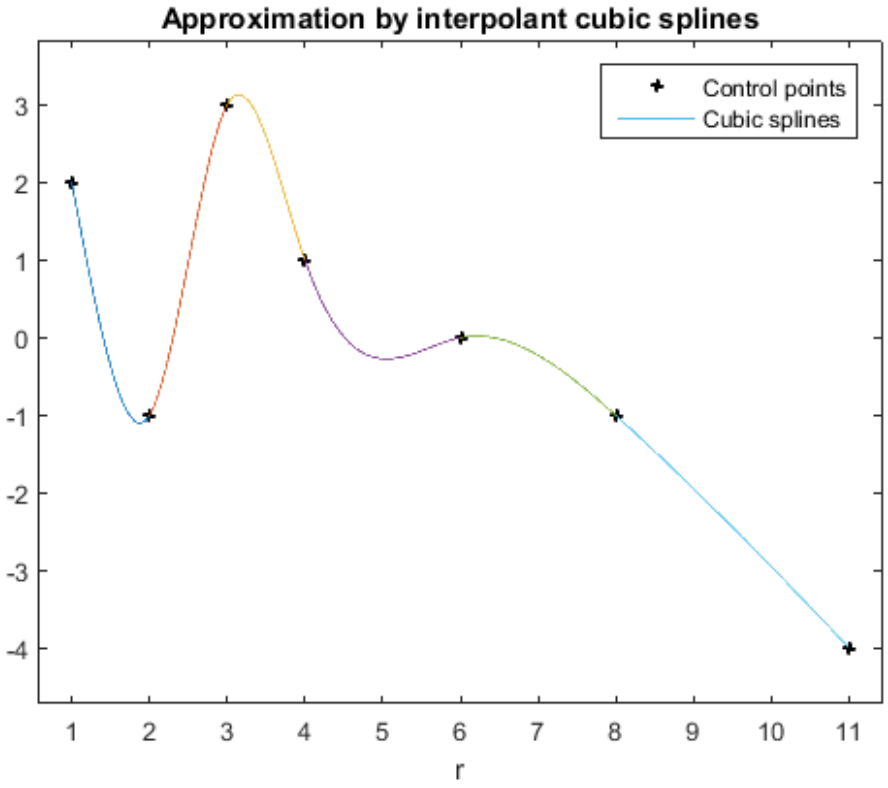


Fig 5.6.1.3: aproximación por splines cúbicos interpolantes.

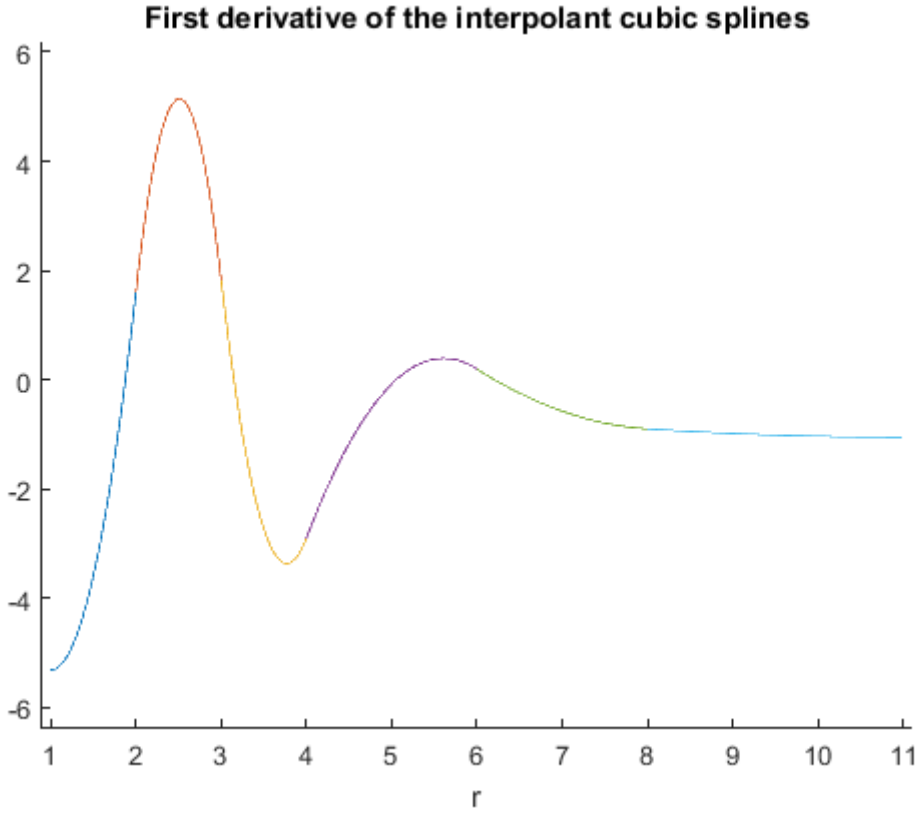


Fig 5.6.1.4: primera derivada del spline cúbico interpolante.

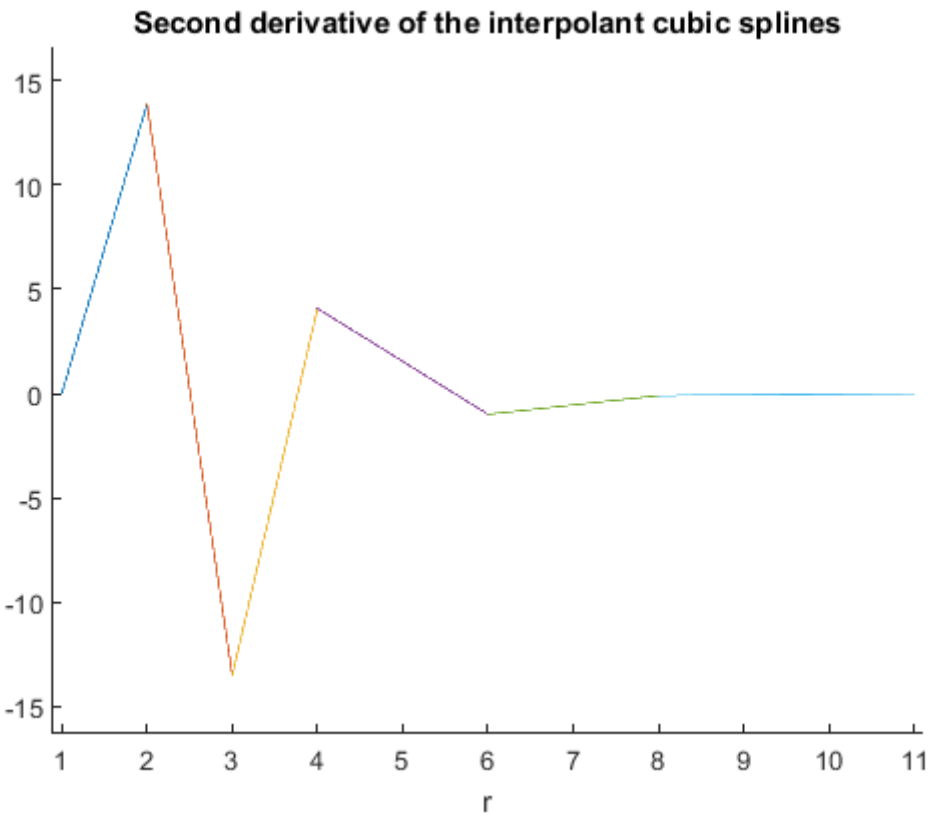


Fig 5.6.1.5: segunda derivada del spline cúbico interpolante.

6. B-SPLINES [2, 10, 13, 14, 17, 18, 19, 21]

6.1 INTRODUCCIÓN

Los B-splines se desarrollan para solucionar la falta de control local, la complejidad para tener continuidad y que la cantidad de puntos que se utilizan imponga el grado como en las curvas de Bézier. Se trata pues de una evolución de las curvas de Bézier.

Los B-splines también permiten tratar con datos discontinuos de manera fácil introduciendo nodos repetidos.

6.2 DEFINICIÓN

Vamos a describir la construcción de un B-spline de orden k con k un número natural. Para ello vamos a utilizar polinomios de grado k-1.

Tenemos un vector de nodos T= $[t_0, t_1, \ldots, t_m]$, el cual está formado por una lista de m+1 números reales no decrecientes, de forma que el mismo valor no aparezca más de k veces (k=orden del B-spline)

$$t_j \leq t_{j+1} \text{ para todo j=0,...,m-1.}$$

Definiremos las m+1-k funciones base B-spline $N_{i\,k}(t)$ de orden k (grado= k-1) como:

Cuando k=1

$$N_{i,1}(t) = \begin{cases} 1 & si \quad t_i \leq t \leq t_{i+1}, \\ 0 & si\ es\ de\ otro\ modo. \end{cases}$$

Cuando k >1

$$N_{ik}(t) = \frac{t-t_i}{t_{i+k-1}-t_i} N_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1}(t).$$

Estas funciones base cumplen las siguientes propiedades básicas:

1. Hay n+1 funciones base, con n+k=m, siendo n+1 el número de puntos de control.
2. $N_{ik}(t) > 0$, $\quad para \quad t_i < t < t_{i+k}.$
3. $N_{ik}(t) = 0$, $\quad para \quad t_0 \leq t \leq t_{i,} \quad t_{i+k} \leq t \leq t_{\_m}.$
4. $\sum_{i=0}^{n} N_{ik}(t) = 1$, $\quad t \in [t_k, t_{n+1}]$, $\quad propiedad\ de\ normalización$.

A continuación, se muestra la programación en Matlab.

```
function [Bt0]=Bsplines_basis(T,iT,k,t0,nt)

% This function computes the basis functions of B-splines
% [Bt0]=Bsplines_basis(T,iT,k,t0,nt);
% Input variables:
% T knot vector
% iT number of basis function in which we are interested
% k order of the basis function
% t0 point we want to evaluate
% nt number of points to discretize the interval T(1):T(m)
%
% Example:
% [Bt0]=Bsplines_basis([0,1,2,3,4,5],3,2,2,1000);


% Number of knots
m=length(T);

% iT must have enough data available at the extremes

ifiT<1 | iT>m-k
str=['Thenumber of theconsideredbasisfunctionmust be between 1 and m-
k'];
uiwait(msgbox(str, 'Error message','error','modal'))
    Bt0=[];
return;
end

% Discretization of the knot vector

t=linspace(T(1),T(m),nt);

% We plot the knot vector

figure(1);
plot(T,zeros(size(T)),'+k','MarkerSize',10);
axis([T(1)-0.1*(T(2)-T(1)) T(m)+0.1*(T(m)-T(m-1)) -0.1 1.5])
holdon

for i=1:nt
    [Bt(i)]=Bspline(T,iT,k,t(i));
end

plot(t,Bt,'r');


[Bt0]=Bspline(T,iT,k,t0);
```

Dentro de este programa se llama a este otro

```
function [Bt]=Bspline(T,iT,k,t)
```

```
% This function computes the basis functions of B-splines
% [Bt0]=Bsplines_basis(T,iT,k,t0);
% Input variables:
% T knot vector
% iT number of basis function in which we are interested
% k order of the basis function
% t0 point we want to evaluate
%
% Example:
% [Bt0]=Bspline([0,1,2,3,4,5],3,2,2);



% number of knots
m=length(T);

% iT must have enough data available at the extremes

if iT<1 | iT>m-k
    str=['The number of the considered basis function must be between
1 and m-k'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end



if k==1

    if iT==m-k

        if t>=T(iT) & t<=T(iT+1)
            Bt=1;
        else
            Bt=0;
        end
```

```
    else
        if t>=T(iT) & t<T(iT+1)
            Bt=1;
        else
            Bt=0;
        end
 end



else


 if abs(T(iT+k-1)-T(iT))<10^(-15)
     if abs(T(iT+k)-T(iT+1))<10^(-15)
         Bt=0;
     else
         Bt=(T(iT+k)-t)/(T(iT+k)-T(iT+1))*Bspline(T,iT+1,k-1,t);
     end
 else
     if abs(T(iT+k)-T(iT+1))<10^(-15)
         Bt=(t-T(iT))/(T(iT+k-1)-T(iT))*Bspline(T,iT,k-1,t);
     else
         Bt=(t-T(iT))/(T(iT+k-1)-T(iT))*Bspline(T,iT,k-1,t)+(T(iT+k)-
t)/(T(iT+k)-T(iT+1))*Bspline(T,iT+1,k-1,t);
     end
 end


end
```

Vamos a realizar un ejemplo donde se construyen las funciones base para un B-spline uniforme de grado 1 con vector de nodos [0,1,2,3,4,5]. A continuación se realizaran también los B-spline de grado dos y tres con el mismo vector de nodos:

-caso uniforme

-grado 1 (orden 2)

```
>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],3,2,2,1000);
```

```
>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],1,2,2,1000);
```

>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],2,2,2,1000);

>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],4,2,2,1000);

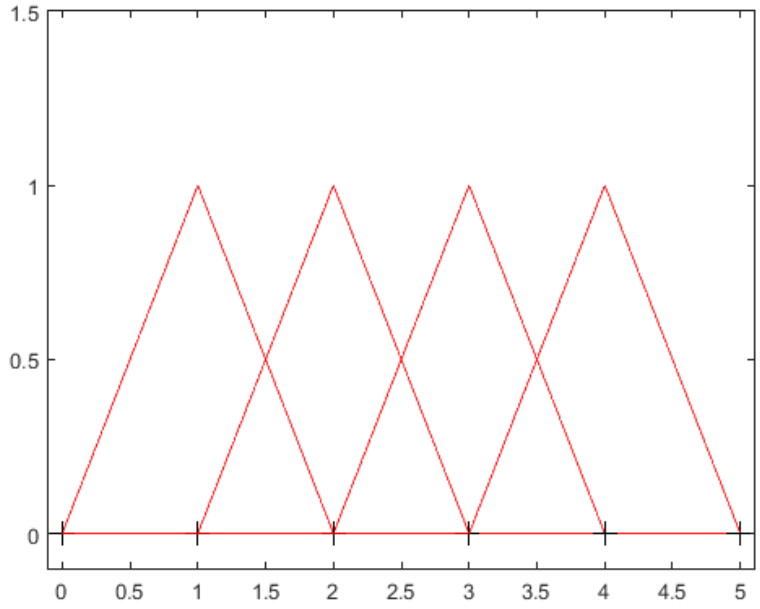


Fig 6.2.1: caso uniforme grado 1.

-grado 2 (orden 3)

>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],1,3,2,1000);

>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],2,3,2,1000);

>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],3,3,2,1000);

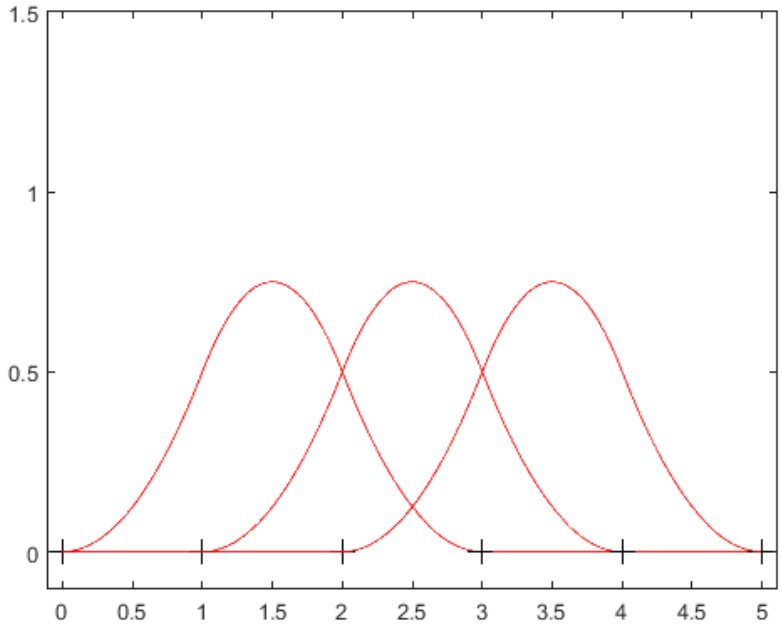


Fig 6.2.2: caso uniforme grado 2.

-grado 3 (orden 4)

>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],1,4,2,1000);

>> [Bt0]=Bsplines_basis([0,1,2,3,4,5],2,4,2,1000);

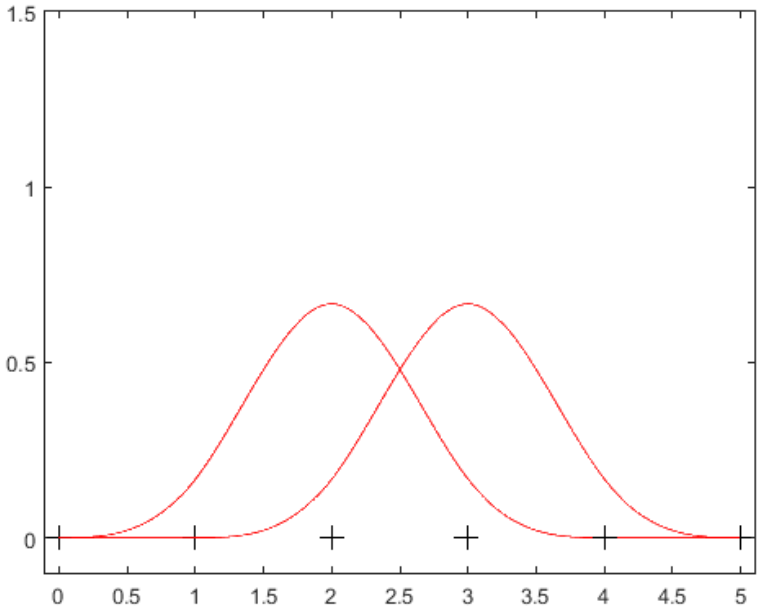


Fig 6.2.3: caso uniforme grado 3.

Los B-spline también pueden utilizar un vector de nodos no uniforme como los casos que se muestran a continuación:

-caso no-uniforme

-grado 1 (orden 2)

>> [Bt0]=Bsplines_basis([0,7,10,21,28,40],1,2,2,1000);

>> [Bt0]=Bsplines_basis([0,7,10,21,28,40],2,2,2,1000);

>> [Bt0]=Bsplines_basis([0,7,10,21,28,40],4,2,2,1000);

>> [Bt0]=Bsplines_basis([0,7,10,21,28,40],5,2,2,1000);

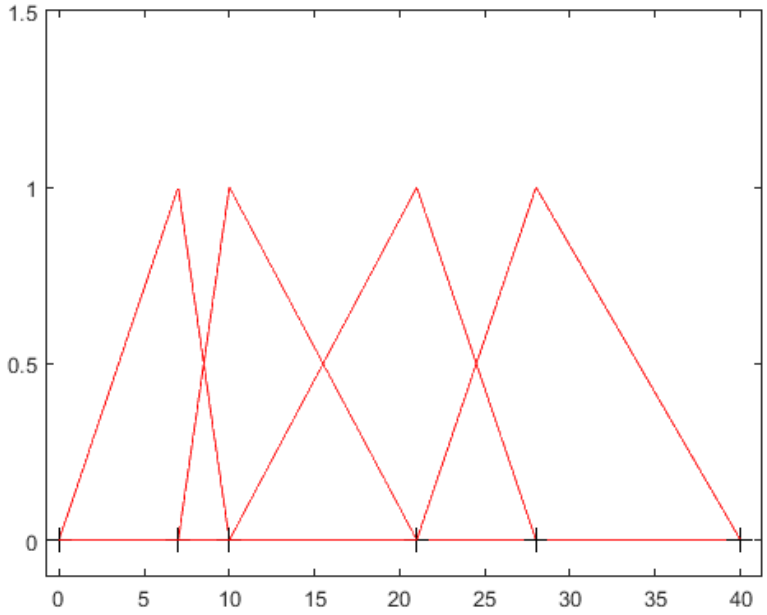


Fig 6.2.4: caso no-uniforme grado 1.

-grado 2 (orden 3)

>> [Bt0]=Bsplines_basis([0,7,10,21,28,40],1,3,2,1000);

>> [Bt0]=Bsplines_basis([0,7,10,21,28,40],2,3,2,1000);

>> [Bt0]=Bsplines_basis([0,7,10,21,28,40],3,3,2,1000);

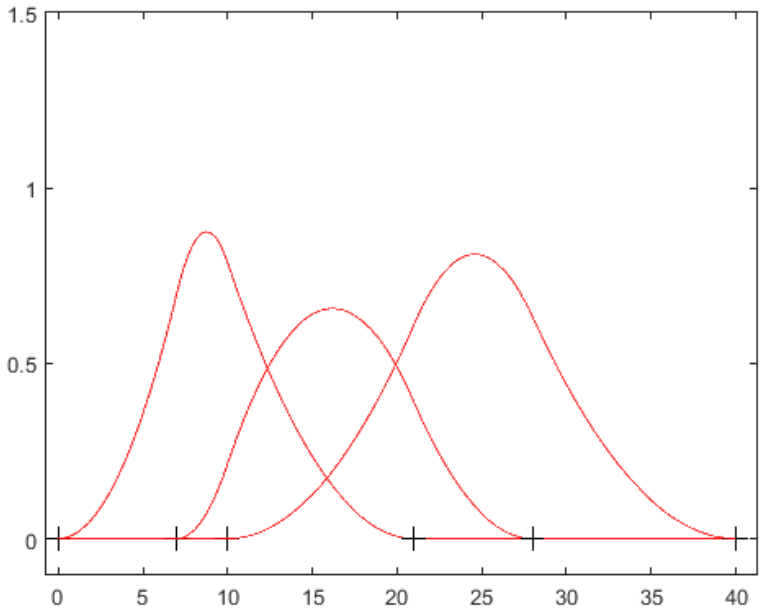


Fig 6.2.5: caso no-uniforme grado 2.

-grado 3 (orden 4)

```
>> [Bt0]=Bsplines_basis([0,7,10,21,24,40,46,55,59,90],1,4,2,1000);
>> [Bt0]=Bsplines_basis([0,7,10,21,24,40,46,55,59,90],2,4,2,1000);
>> [Bt0]=Bsplines_basis([0,7,10,21,24,40,46,55,59,90],3,4,2,1000);
>> [Bt0]=Bsplines_basis([0,7,10,21,24,40,46,55,59,90],4,4,2,1000);
>> [Bt0]=Bsplines_basis([0,7,10,21,24,40,46,55,59,90],5,4,2,1000);
>> [Bt0]=Bsplines_basis([0,7,10,21,24,40,46,55,59,90],6,4,2,1000);
```

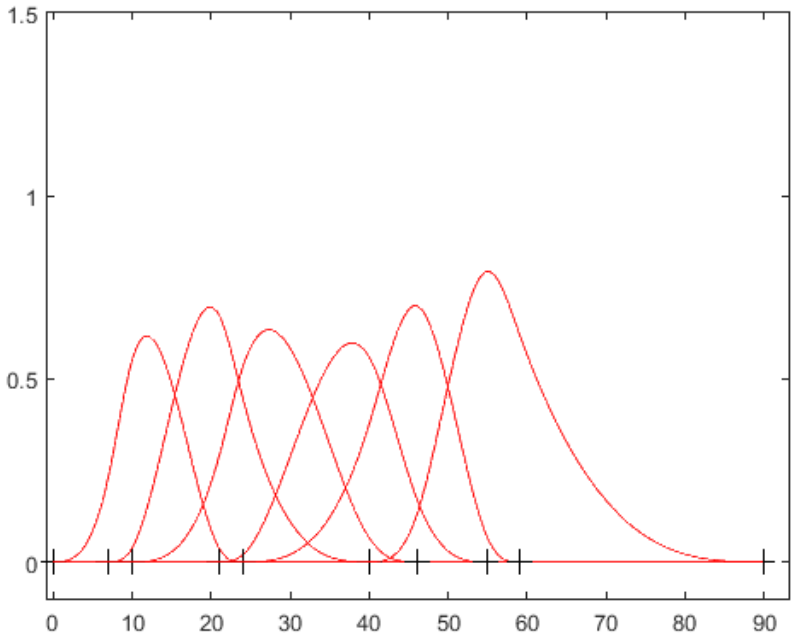


Fig 6.2.6: caso no-uniforme grado 3.

En el vector de nodos asociado a los B-spline se pueden repetir nodos con el objetivo de generar puntos donde la función presenta menor suavidad, así por ejemplo se pueden obtener curvas que reproducen picos.

- Efecto de la repetición de nodos

-grado 1 (orden 2)

```
>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],1,2,2,1000);
>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],1,2,2,1000);
>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],2,2,2,1000);
>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],3,2,2,1000);
>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],4,2,2,1000);
```

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],5,2,2,1000);

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],6,2,2,1000);

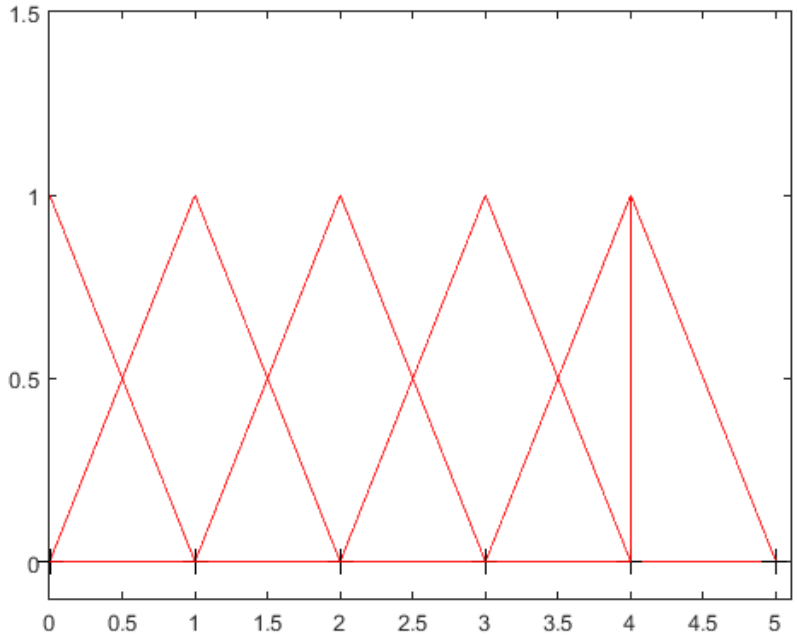


Fig 6.2.7: repetición de nodos grado 1.

Podemos ver que al repetir el nodo cero se nos genera la primera función base, que utiliza un soporte de tres puntos, utilizando un nodo repetido y esto provoca una discontinuidad de salto en el nodo cero. De igual forma al repetir el nodo cuatro también se produce una discontinuidad de salto n dicho nodo.

- grado 2 (orden 3), repitiendo dos veces un mismo nodo:

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],1,3,2,1000);

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],2,3,2,1000);

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],3,3,2,1000);

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],4,3,2,1000);

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],5,3,2,1000);

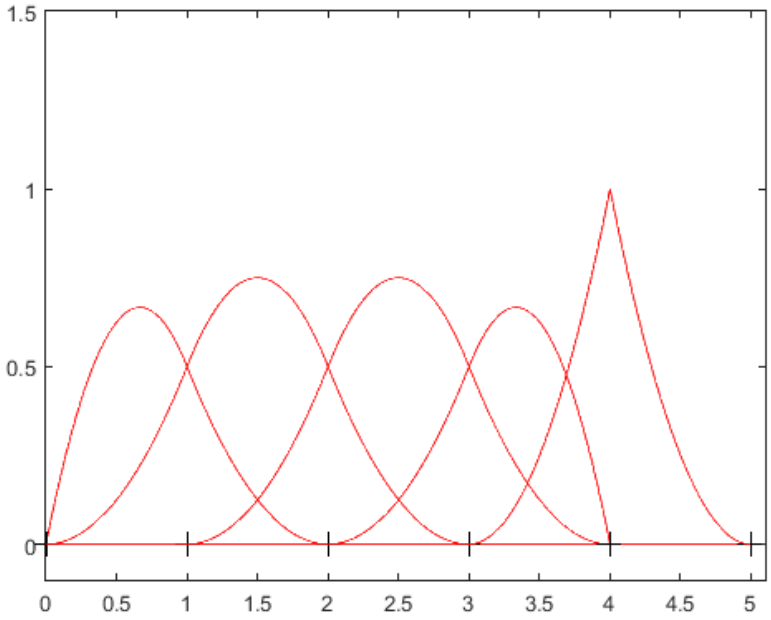


Fig 6.2.8: repetición de nodos grado 2, repitiendo dos veces un mismo nodo.

Podemos observar que al repetir el nodo 4 y obtener la quinta función base la función cambia totalmente de forma, dando origen a una discontinuidad en la primera derivada. Esto mismo también ocurre en la primera función base ya que se ha repetido el nodo 0.

- grado 2 (orden 3), repitiendo tres veces un mismo nodo:

>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],1,3,2,1000)

>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],2,3,2,1000)

>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],3,3,2,1000)

>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],4,3,2,1000)

>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],5,3,2,1000)

>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],6,3,2,1000)

>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],7,3,2,1000)

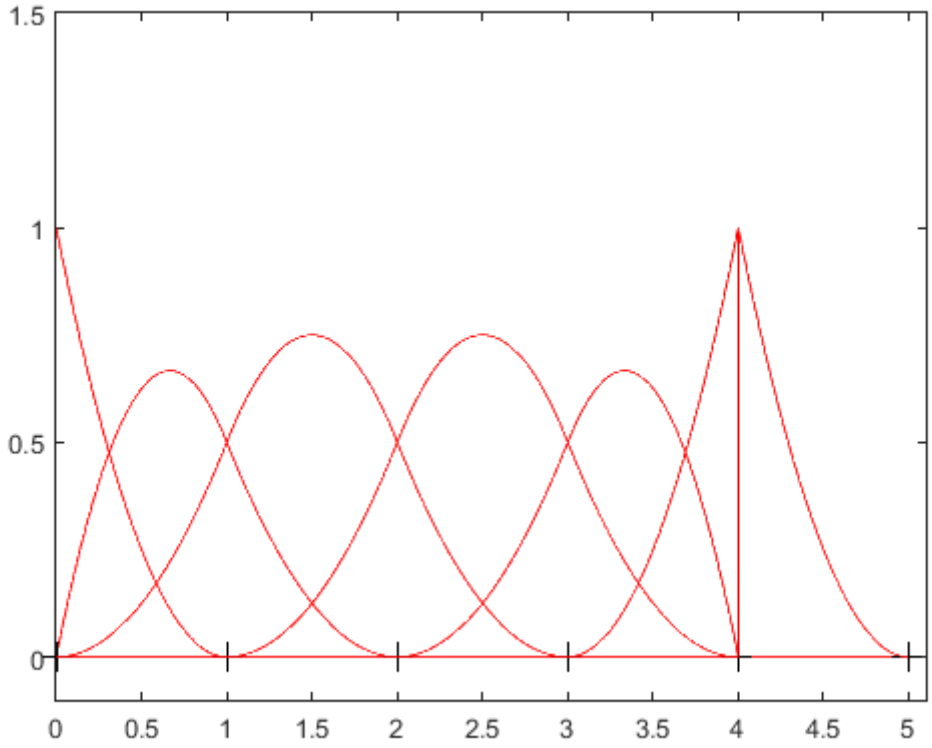


Fig 6.2.9: repetición de nodos grado 2, repitiendo tres veces un mismo nodo.

En este caso como se ha repetido tres veces el nodo 0 y el nodo 4, se han generado discontinuidades de salto en las funciones base correspondientes.

-grado 3 (orden 4), repitiendo dos veces un mismo nodo:

```
>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],1,4,2,1000);

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],2,4,2,1000);

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],3,4,2,1000);

>> [Bt0]=Bsplines_basis([0,0,1,2,3,4,4,5],4,4,2,1000);
```

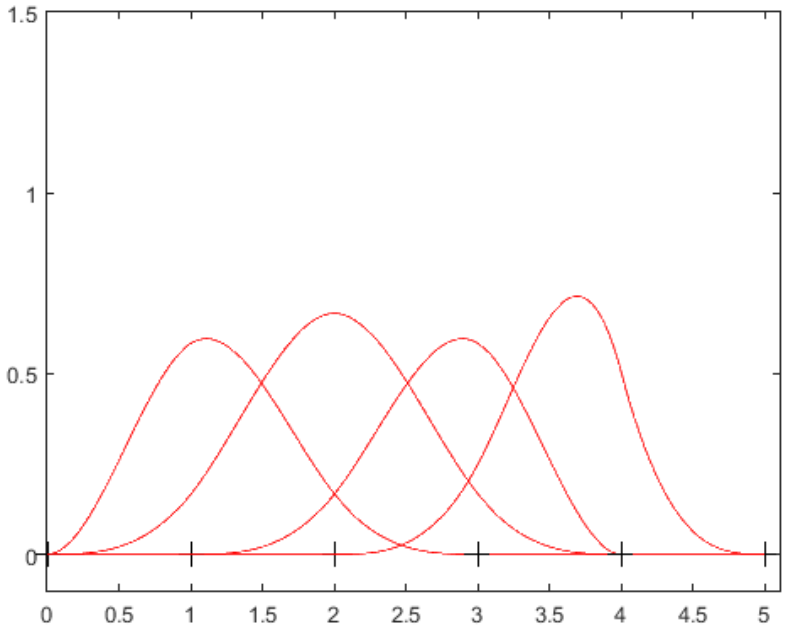


Fig 6.2.10: repetición de nodos grado 3, repitiendo dos veces un mismo nodo.

Podemos ver que con grado 3 sucede el mismo patrón que con los casos anteriores.

-grado 3 (orden 4), repitiendo tres veces un mismo nodo:

```
>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],1,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],1,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],2,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],3,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],4,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],5,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,1,2,3,4,4,4,5],6,4,2,1000)
```

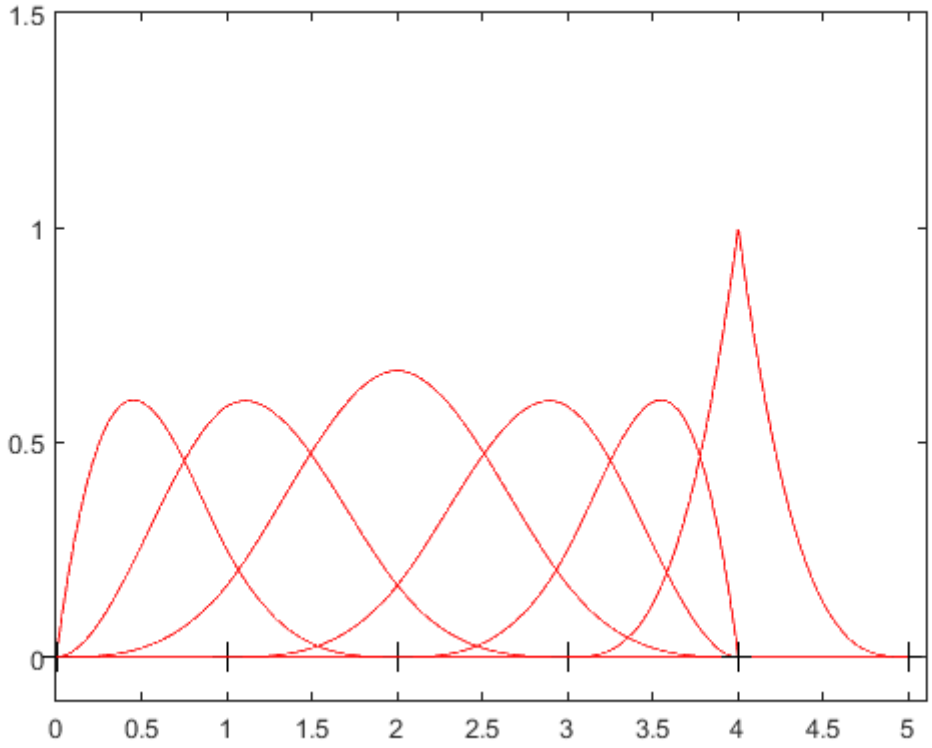


Fig 6.2.11: repetición de nodos grado 3, repitiendo tres veces un mismo nodo.

-grado 3 (orden 4), repitiendo cuatro veces un mismo nodo:

```
>> [Bt0]=Bsplines_basis([0,0,0,0,1,2,3,4,4,4,4,5],1,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,0,1,2,3,4,4,4,4,5],2,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,0,1,2,3,4,4,4,4,5],3,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,0,1,2,3,4,4,4,4,5],4,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,0,1,2,3,4,4,4,4,5],5,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,0,1,2,3,4,4,4,4,5],6,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,0,1,2,3,4,4,4,4,5],7,4,2,1000)
>> [Bt0]=Bsplines_basis([0,0,0,0,1,2,3,4,4,4,4,5],8,4,2,1000)
```

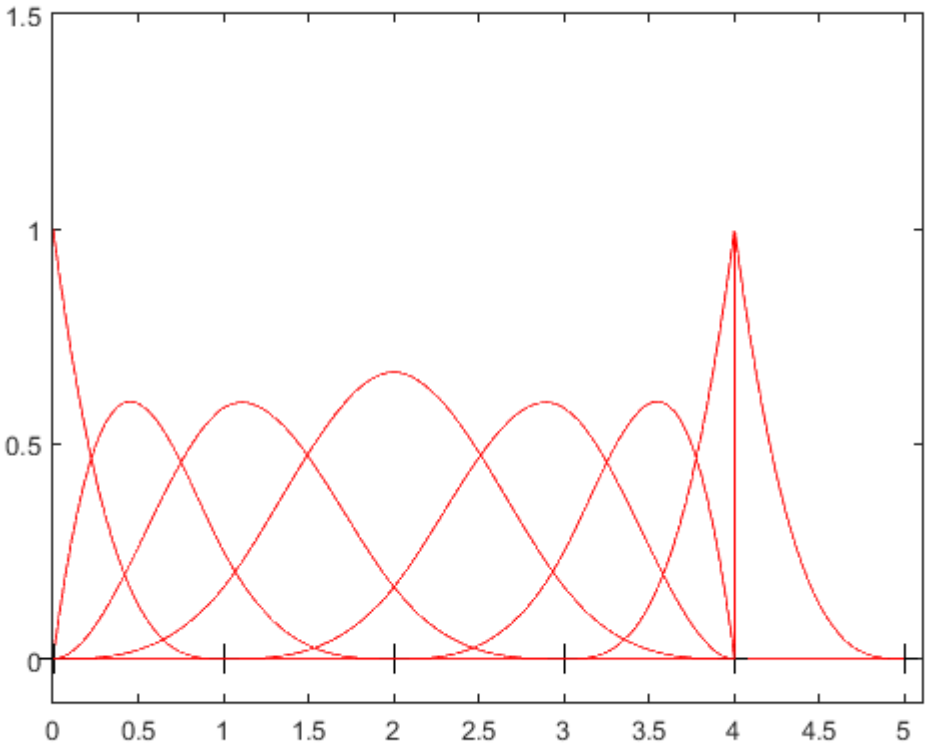


Fig 6.2.12: repetición de nodos grado 3, repitiendo cuatro veces un mismo nodo.

Tal y como podemos comprobar en las figuras anteriores, cuando utilizamos un B-spline de grado k-1 las funciones base presentan suavidad de clase C^{K-2}. Cada vez que repetimos un nodo la continuidad baja en un orden de magnitud, así por ejemplo cuando utilizamos un B-spline de grado 3 si no repetimos ningún nodo, tenemos que las funciones base son de clase C^2, es decir, continuas con 1ª y 2ª derivada continuas también. Si repetimos un nodo una vez, en ese nodo la función base será de clase C^1, si lo repetimos dos veces será de clase C^0, es decir, solo continua con un pico y si lo repetimos tres veces tendremos una discontinuidad de salto en ese nodo. Esto se puede comprobar también con los B-spline de grado 2, sólo que en este caso tanto el pico como la discontinuidad de salto se dan al repetir una y dos veces respectivamente un mismo nodo.

A continuación vamos a comprobar que se cumple la propiedad de normalización de las funciones base mediante un programa Matlab diseñado al efecto. Recordamos que la propiedad de normalización dice:

$$\sum_{i=0}^{n} N_{ik}(t) = 1 \ , \qquad t \in \left[t_{k,} t_{n+1}\right]$$

Donde N_{ik} son las n+1 funciones base, con n+k=m siendo k el grado del B-spline y m+1 el número de nodos totales.

-Propiedad de Normalización de las funciones Base

```
function [Nt0]=normalizing_property_Bsplines(T,k,t0,nt)

% This function proves computationally the normalizing property of B-
spline basis functions
% [Nt0]normalizing_property_Bsplines(T,k,t0,nt);
% Input variables:
% T knot vector
% It number of basis function in which we are interested
% k order of the basis function
% t0 point we want to evaluate
% nt number of points to discretize the interval T(1):T(m)
% Output variables:
% Nt0 sum of the Bspline basis functions at t0
%
% Example:
% [Nt0]=normalizing_property_Bsplines([0,1,2,3,4,5],3,2,1000);


% Number of knots
m=length(T);


% We plot each basis function for order k


Nt0=0;
foriT=1:m-k
    [Bt0]=Bsplines_basis(T,iT,k,t0,nt);
aux(iT)=Bt0;
    Ms=12-2*nnz(~(Bt0-aux(1:iT-1)));
plot(t0,Bt0,'co','MarkerSize',Ms,'MarkerFaceColor','Cyan','MarkerEdgeC
olor','Black');
    Nt0=Nt0+Bt0;
end

% We plot a parellel line to the x-axis with value 1

plot(T,ones(1,m),'g','LineWidth',3);

% We plot Nt0 at the abscisa t0

plot([t0,t0],[0,1],'r','LineWidth',1);
plot(t0,Nt0,'mo','MarkerSize',12,'MarkerFaceColor','Magenta','MarkerEd
geColor','Black');
text(t0,1.1,'Nt0=1');
```

Ejemplo:

>> [Nt0]=normalizing_property_Bsplines([0,1,2,3,4,5],3,2,1000);

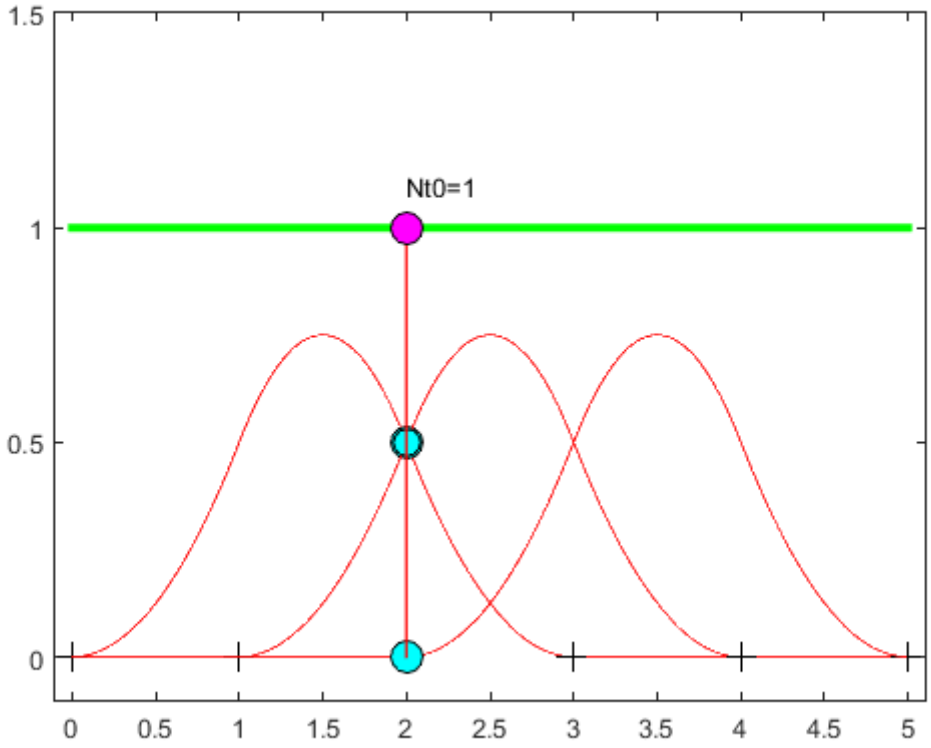


Fig 6.2.13: propiedad de normalización de la funciones base.

Como podemos observar en la gráfica se cumple la propiedad de normalización, es decir, las funciones base sumadas en cualquier t del rango de parámetros suman 1, en particular en el valor t=2 representado en la gráfica.

6.3 B-SPLINE CURVE

Dado un conjunto de n + 1 puntos de control (llamados puntos de Boor) P_i (i = 0, ..., n) y un vector de nodo T = [t_0, t_1, ..., t_{m-1}, t_m] se define un B-spline S (t) de orden k como:

$$S(t) = \sum_{i=0}^{n} P_i N_{ik}(t).$$

Dónde N_{ik} (t) describe la función base B-spline de grado k asociado con el vector de nodo T.

Se debe cumplir la siguiente regla entre el número de puntos de control y el número de nodos:

$$m + 1 = (n + 1) + k;$$

donde m+1 es el número de nodos (entradas del vector de nodos), n+1 es el número de puntos de control, y K es el orden de la curva. Simplificando:

$$m = n + k\ .$$

Los vectores de nodos se pueden clasificar como:

- Periódico / Uniforme

  La influencia de cada función básica está limitada a k intervalos y el rango de parámetros está compuesto por:

  $$T(k) \leq t \leq T(n+1).$$

  Que la distancia $t_i - t_{i-1} = C$ sea contante entre nodos determina el carácter uniforme del B-spline. En la frontera del rango de parámetros se añaden k-1 nodos tanto a la izquierda como a la derecha manteniendo el carácter uniforme.

  Ejemplo: n=3, k=3, m=6, T= [0 1 2 3 4 5 6].

- No periódico

  Podemos repetir los valores del vector nodo en los extremos, de manera que la curva interpola los puntos de control primero y último. Las condiciones de frontera se toman en este caso hacia el interior del intervalo.

  El rango de parámetros es el mismo que en el caso periódico:

  $$T(k) \leq t \leq T(n+1).$$

  -

  Ejemplo: n=3, k=3, m=6, T= [0 0 0 1 2 2 2].

- No uniformes (NURBS)

  No uniforme hace referencia a que la distancia entre nodos sucesivos no es constante. Este hecho permite que las funciones base sean más apuntadas o planas en diferentes zonas y así conseguir mejores resultados al definir perfiles. Los NURBS son un tipo de curvas B-splines no uniformes, de hecho se trata de curvas B-spline racionales basadas en vectores de nodos no uniformes (Non Uniform Rational B-splines, o también se les conoce como Nobody Understand Rational B-splines debido a que poca gente llega a comprender bien su naturaleza matemática y su uso correcto en la práctica.

  Ejemplo: T= [0 1 2 3 3 3 4], o T= [0 0 1 2 4 5 5].

En la representación de Bézier, es un caso simple de B-spline, cuando:

- Número de puntos de control= n+1= orden del B-spline.

- Se considera un vector de nodo no periódico dado por la mitad inicial de valores 0's, n+1 valores, y la otra mitad 1's.

A continuación se muestra la programación en Matlab para este apartado y los ejemplos realizados:

-Programa Matlab

```
function [Bt0]=Bspline_Curve(P,T,k,nt,type_Bs,varargin)


% This function computes the B-spline curve associated with the Knot
vector

% T and the control points P

% [Bt0]=Bspline_Curve(P,T,k,nt,type_Bs,varargin);

% Input variables:

% P control points

% T knot vector satifying

%       number of knots= number of control points + order of the
Bspline

% k order of the basis function

% nt number of points to discretize the interval T(1):T(m)

% type_Bs  'P' for Periodic, 'NP' for Non Periodic

% t0 (optional) point where we want to evaluate

% Output variables:

% Bt0 point in the curve for value of t=t0

%

% Example1:

%
[Bt0]=Bspline_Curve([1,2;2,4;3,2;4,1;5,3],[0,1,2,3,4,5,6,7],3,1000,'P'
,2.5);

%

% Example2:

%
[Bt0]=Bspline_Curve([1,2;2,4;3,2;4,1;5,3],[0,0,0,1,2,3,3,3],3,1000,'NP
',2.5);

%

% Example3:

%
[Bt0]=Bspline_Curve([1,2;2,5;3,7],[0,0.2,0.4,0.6,0.8,1],3,1000,'P',0.5
);

%

% Example4:

%
[Bt0]=Bspline_Curve([1,2;3,7;4,4;5,6;7,9;8,10;10,12],[0,0,0,0,0.25,0.5
,0.75,1,1,1,1],4,1000,'NP',0.4);
```

```
% We obtain the length of the knot vector and the number of control
points



m=length(T);

n=size(P,1);



% We check that the equation m=n+k is satisfied


if m~=n+k

    str=['The number of knots must be equal to the number of control
points plus the order k of the Bspline'];

    uiwait(msgbox(str, 'Error message','error','modal'))

    return;

end

% We see if there is any t0 to evaluate


if nargin==5


    % We initialize Bt0 as empty when there is no t0 where to evaluate

    Bt0=[];


elseif nargin==6

    Bt0=[];

    t0=varargin{1};


end




% We discretize the knot vector
```

```
t=linspace(T(k),T(n+1),nt);

switch type_Bs

    case 'P'

        aux=T(1:k); aux=diff(aux);
        aux1=T(n+1:m); aux1=diff(aux1);

        if ~all(aux)| ~all(aux1)
            str=['The vector of knots must be periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end


    case 'NP'

        aux=T(1:k); aux=aux-T(1);
        aux1=T(n+1:m); aux1=aux1-T(m);

        if any(aux)| any(aux1)
            str=['The vector of knots must be non-periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end


end


% We calculate the point in the curve for each value of t

for it=1:nt-1
```

```
    xt(it,:)=[0,0];

    % Expression for computing the Bspline curve

    for in=1:n
        xt(it,:)=xt(it,:)+P(in,:)*Bspline(T,in,k,t(it));
    end


end


% We plot in 2D the curve
    plot(xt(:,1),xt(:,2),'k-','MarkerSize',10);
    hold on


% We plot the control polygon


plot(P(1:in,1),P(1:in,2),'r-');
plot(P(1:in,1),P(1:in,2),'bo','MarkerFaceColor','b');

mx=min(P(1:in,1));
Mx=max(P(1:in,1));
my=min(P(1:in,2));
My=max(P(1:in,2));
axis([mx-0.1*(Mx-mx) Mx+0.1*(Mx-mx) my-0.1*(My-my) My+0.1*(My-my)]);


% We evaluate at t0 if it is the case

if nargin==6

    xt=[0,0];
```

```
    for in=1:n
        xt=xt+P(in,:)*Bspline(T,in,k,t0);
    end

    Bt0=[xt(1),xt(2)];

    % We plot in 2D the curve
    plot(xt(1),xt(2),'go','MarkerSize',10);

end
```

-Ejemplo 1

En este ejemplo realizaremos el caso de periódico:

>> [Bt0]=Bspline_Curve([1,2;2,4;3,2;4,1;5,3],[0,1,2,3,4,5,6,7],3,1000,'P',2.5);

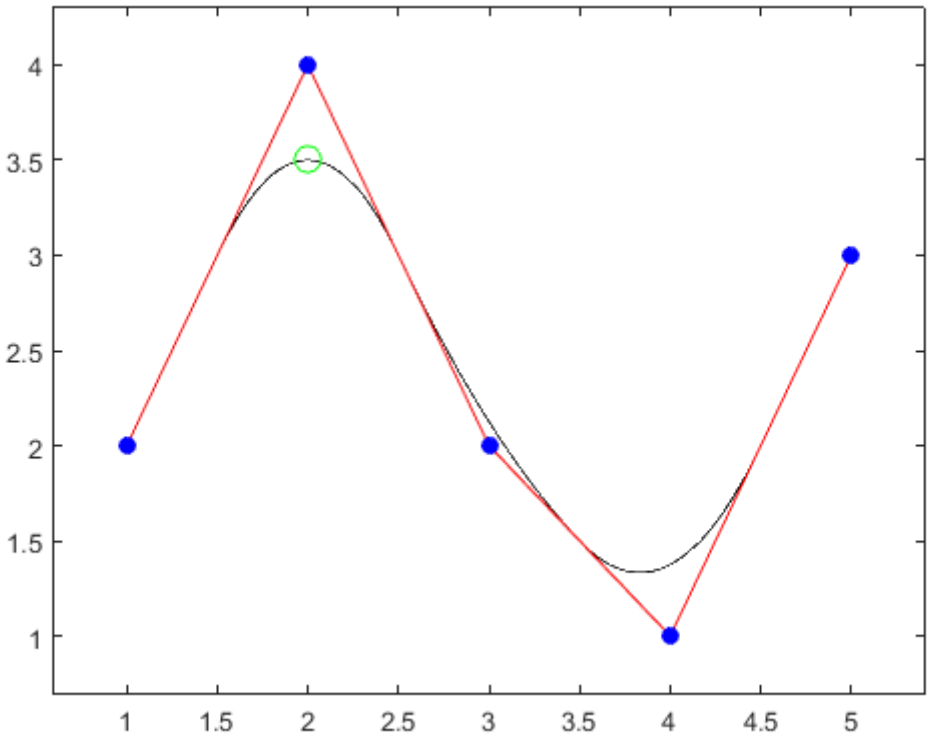


Fig. 6.3.1: ejemplo 1 Bspline_Curve caso periódico.

-Ejemplo 2

En este ejemplo se realizaremos el caso de no periódico:

>> [Bt0]=Bspline_Curve([1,2;2,4;3,2;4,1;5,3],[0,0,0,1,2,3,3,3],3,1000,'NP',2.5);

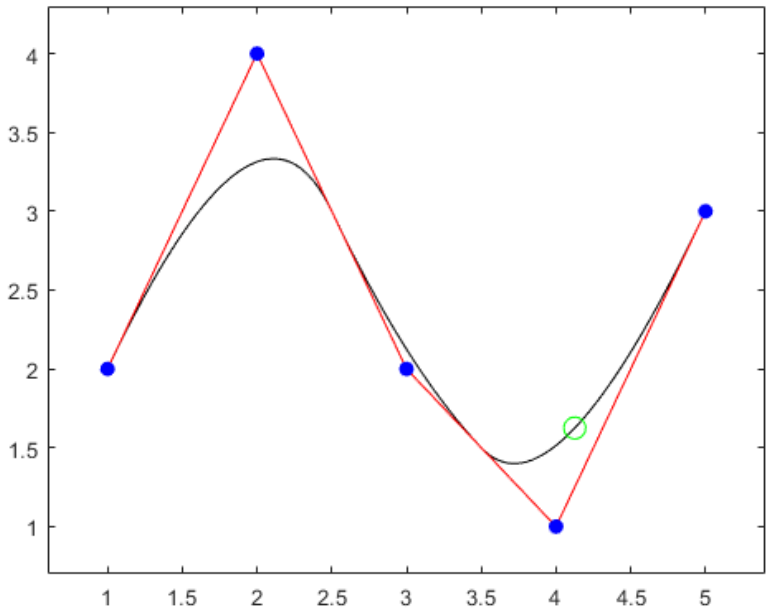


Fig. 6.3.2: ejemplo 2 Bspline_Curve caso no periódico.

- Ejemplo 3

>> [Bt0]=Bspline_Curve([1,2; 2,5;3,7],[0,0.2,0.4,0.6,0.8,1],3,1000,'P',0.5);

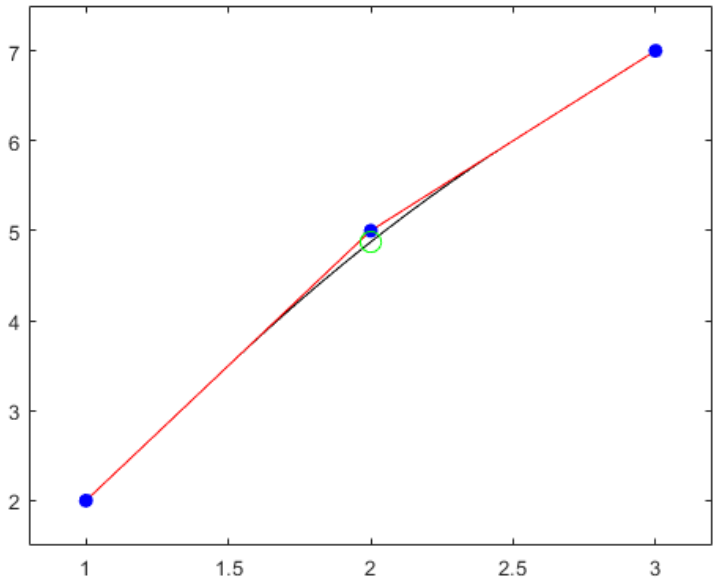


Fig. 6.3.3: ejemplo 3 Bspline_Curve caso periódico.

- Ejemplo 4

>>[Bt0]=Bspline_Curve([1,2;3,7;4,4;5,6;7,9;8,10;10,12],[0,0,0,0,0.25,0.5,0.75,1,1,1,1],4,1000,'NP',0.4);

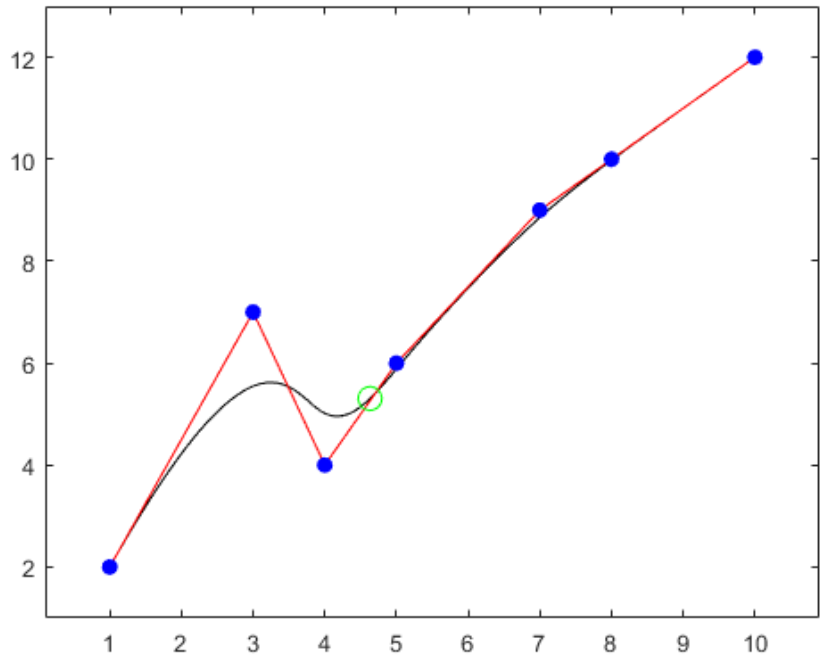


Fig. 6.3.4: ejemplo 4 Bspline_Curve caso no periódico.

```
function [Bt0]=Rational_Bspline_Curve3(P,w,T,k,nt,type_Bs,varargin)

% This function computes the Rational Bspline Curve with weights
%
% [Bt0]=Rational_Bspline_Curve3(P,w,T,k,nt,type_Bs,varargin);
%
% Input variables:
%
% P 2D cell array containing the three spatial coordinates of the
control points
% w weights vector associated with the control points
% T knot vector satifying
%       number of knots= number of control points + order of the
Bspline
% k order of the basis function
% nt number of points to discretize the interval T(1):T(m)
% type_Bs  'P' for Periodic, 'NP' for Non Periodic
% t0 (optional) point where we want to evaluate
%
% Output variables:
%
% Bt0 point in the curve for value of t=t0
%
% Example1:
```

```
% syms u
% C(u)=[cos(u),sin(u),u];
% nu=20;
% ud=linspace(0,2*pi,nu);
% P=cell(nu,1);
% for i=1:nu
%    P{i}=double(subs(C(u),u,ud(i)));
% end
% w=ones(1,nu); w(1)=100; w(nu)=100;
% k=4;
% T=1:nu+k;
% t0=10.5;
% nt=100;
% [Bt0]=Rational_Bspline_Curve3(P,w,T,k,nt,'P',t0);


% We obtain the length of the knot vector and the number of control
points


m=length(T);
n=size(P,1);
s=length(w);


% We check that the equation m=n+k is satisfied

if m~=n+k
    str=['The number of knots must be equal to the number of control
points plus the order k of the Bspline'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end

% We check that we have a weight for each point

if n~=s
    str=['The number of points must be equal to the lenght of weights
vector w'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end



% We see if there is any t0 to evaluate

if nargin==6

    % We initialize Bt0 as empty when there is no t0 where to evaluate
    Bt0=[];

elseif nargin==7

    t0=varargin{1};

end
```

```
% We discretize the knot vector

switch type_Bs

    case 'P'

        t=linspace(T(k),T(n+1),nt);

    case 'NP'

        t=linspace(T(1),T(m-k+1),nt);

end


% We calculate the point in the curve for each value of t

xt=cell(nt,1);


for it=1:nt

    xt{it}=[0,0,0];    % It is going to contain only the numerator of
the
                % rational B-spline till a final division

    % Initialization of the denominator of the Rational B-spline xt
for
    % each value of t

    d=0;

    % Expression for computing the Bspline curve

    for in=1:n
        xt{it}=xt{it}+w(in)*P{in}*Bspline(T,in,k,t(it));
        d=d+w(in)*Bspline(T,in,k,t(it));
    end

    % We perfom the mentioned division in Rational B-splines
    xt{it}=xt{it}/d;

    % We plot in 2D the curve
    Paux=xt{it};
    plot3(Paux(1),Paux(2),Paux(3),'ko-','MarkerSize',3);
    hold on

end


% We plot the control polygon

for i=1:n
      aux=P{i};
      Px(i)=aux(1);
```

```
        Py(i)=aux(2);
        Pz(i)=aux(3);
end

plot3(Px,Py,Pz,'r-');
plot3(Px,Py,Pz,'bo','MarkerFaceColor','b');

mx=min(Px);
Mx=max(Px);
my=min(Py);
My=max(Py);
mz=min(Pz);
Mz=max(Pz);
axis([mx-0.1*(Mx-mx) Mx+0.1*(Mx-mx) my-0.1*(My-my) My+0.1*(My-my) mz-
0.1*(Mz-mz) Mz+0.1*(Mz-mz) ]);


% We evaluate at t0 if it is the case

if nargin==7

    % Initialization of the denominator of the Rational B-spline xt
for
    % t=t0

    d=0;

    xt=[0,0,0];
    for in=1:n
        xt=xt+w(in)*P{in}*Bspline(T,in,k,t0);
        d=d+w(in)*Bspline(T,in,k,t0);
    end

    % We perfom the mentioned division in Rational B-splines
    xt=xt/d;
    Bt0=[xt(1),xt(2),xt(3)];

    % We plot in 2D the point
    plot3(xt(1),xt(2),xt(3),'go','MarkerSize',10);

end
```

6.4 ALGORITMO DE BOOR

Este algoritmo fue desarrollado por Boor en 1972.

Consideramos la combinación lineal

$$S(t) = \sum_i c_i^0 N_i^n(t).$$

De B-splines de grado n sobre una secuencia de nodos (t_i). Sin pérdida de generalidad podemos suponer que la secuencia de nodos y la sumatoria se extienden de $-\infty$ a ∞. Por la forma de los soportes locales de los N_i^n esta suma es siempre finita para cualquier t dado. Supongamos que $t \in [t_n, t_{n+1}]t$, entonces

$$S(t) = \sum_{i=0}^{n} c_i^0 N_i^n(t).$$

Utilizando repetidamente la recursión para B-splines y agrupando términos obtenemos:

$$S(t) = \sum_{i=1}^{n} c_i^1 N_i^{n-1}(t)$$

$$\vdots$$

$$= \sum_{i=n}^{n} c_i^n N_i^0(t) = c_n^n \,,$$

donde los c_i^r están dados por las combinaciones afines

$$c_i^r = (1-\alpha)c_{i-1}^{r-1} + \alpha\, c_i^{r-1}, \qquad \alpha = \alpha_i^{n-r} = \frac{t - t_i}{t_{i+n+1-r} - t_i}.$$

Note que $\alpha \in [0,1]$ pues $t \in [t_n, t_{n+1})$, y por tanto, las convinaciones afines son convexas.

Los puntos c_i^r los podemos ordenar en el siguiente esquema triangular, en el la regla de recursión es la anterior recursión afín.

$$c_0^0$$

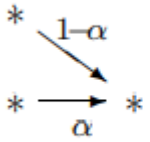
$$c_1^0 \quad c_1^1$$

$$c_2^0 \quad c_2^1 \quad c_2^2$$

$$\vdots \qquad\qquad \ddots$$

$$c_n^0 \quad c_n^1 \quad c_n^2 \cdots c_n^n$$

regla



(α depende de la posición)

Una consecuencia importante del algoritmo de Boor es que el spline $S(t)$ sobre un intervalo intermodal (entre dos nodos consecutivos) es una combinación convexa de n+1 coeficientes consecutivos c_i. Por tanto si los c_i son puntos en un espacio afín, tenemos que $S(t)$ también es un punto del espacio afín. Debido a esto, los c_i se llaman puntos de control de $S(t)$.

Es más, el spline está en la capsula convexa de los puntos de control, lo que implica

$$\sum_{i=0}^{n} 1\, N_i^n\,(t) = 1 \qquad para \qquad t \in [t_n, t_{n+1}),$$

los B-splines forman una partición de la unidad.

Tendremos que tener en cuenta que para $t \in \mathbb{R}$, el algoritmo de Boor aplicado a los puntos $c_0^0, \ldots, c_n^0$ calcula el polinomio $S_n(t)$, que coincide con $S(t)$ sobre el intervalo intermodal $[t_n, t_{n+1})$.

Este algoritmo generaliza el algoritmo de Casteljau de la misma manera que los B-splines generalizan las curvas de Bézier.

De hecho cualquier curva de Bézier se puede generar con el algoritmo de Cox de Boor usando un vector de nodos de doble longitud que la longitud del vector de puntos de control y con orden del B-spline igual al número de puntos de control, y por tanto de un grado de los polinomios B-spline una unidad inferior. Ese vector de nodos consiste de la mitad de valores cero y la mitad de valores 1.

Esto se comprueba mediante el siguiente ejemplo realizado con los programas desarrollados:

[Pt,t,P]=Casteljau([1,2;3,7;4,4;5,6;7,9;8,10;10,12],0.5)

Pt=[5.3438, 6.7344]

[Bu]=Boor_algorithm_Bsplines([1,2;3,7;4,4;5,6;7,9;8,10;10,12],7,[0,0,0,0,0,0,0,1,1,1,1,1,1,1],0.5)

Bu==[5.3438, 6.7344].

Como se puede observar hemos obtenido el mismo resultado por ambos métodos.

A continuación se muestra la programación en Matlab

```
function [Bu,P]=Boor_algorithm_Bsplines(C,p,U,u)

% This function evaluates in u the B-spline of degree p which makes
% use of the control points P and the knot vector U
%
% [Bu]=Algoritmo_Boor_Bsplines(C,p,U,u)
% Input variables:
% C vector of control points
% p order of the B-spline
% U knot vector of length m
% u point to evaluate the B-spline, U(p)<= u <= U(m+1-p)
% Output variables:
% Bu value of the B-spline at u
% P successive control points
%
```

```
% Example 1:
%
% Enunciate:
% Given the control points
% P1=[1,2], P2=[3,7], p3=[4,4], p4=[2,-1], p5=[1,2]
% and the knot vector
% U=[0,0,0,0.3,0.6,1,1,1]
% evaluate the B-spline of degree 2 in u=0.5
%
% Solution:
% [Bu]=Boor_algorithm_Bsplines([1,2;3,7;4,4;2,-
1;1,2],3,[0,0,0,0.3,0.6,1,1,1],0.5);
%
% Example 2:
%
[Bu]=Boor_algorithm_Bsplines([1,2;3,7;4,4],3,[0,0.2,0.4,0.6,0.8,1],0.5
);
%
% Example 3:
%
[Bu]=Boor_algorithm_Bsplines([1,2;3,7;4,4;5,6;7,9;8,10;10,12],4,[0,0,0
,0,0.25,0.5,0.75,1,1,1,1],0.4);


% We find the interval such that U(i)<=u<=U(i+1)


m=length(U);
n=length(C);
ind=find(U<=u);
k=max(ind);



% We define the initial points to iterate


P=zeros(p,n,2);
a=zeros(p-1,n);
P(1,1:k,:)=C(1:k,:);
display('Puntos iniciales usados')
```

```
P0=[P(1,:,1);P(1,:,2)]'



% We start the iteration
for r=1:p-1
    for i=k-p+r+1:k
        a(r,i)=(u-U(i))/(U(i+p-r)-U(i));  % weights
        P(r+1,i,:)=(1-a(r,i))*P(r,i-1,:)+a(r,i)*P(r,i,:);
    end
    display(['Pesos calculados en el paso',blanks(2),num2str(r)])
    Pesos=a(r,:)'
    display(['Puntos calculados en el paso',blanks(2),num2str(r)])
    Paso=[P(r+1,:,1);P(r+1,:,2)]'
    pause
end

% We print the aproximated point

display(['Punto aproximado para la abscisa x=',num2str(u)])
Bu=[P(p,k,1),P(p,k,2)]
```

6.5 RATIONAL B-SPLINE CURVE

Si a la definición dada en el apartado anterior nosotros le introducimos pesos w_{ij}, podemos obtener la rational B-spline curve mediante la siguiente expresión:

$$R(t) = \frac{\sum_{i=0}^{n} P_i w_i N_{ik}(t)}{\sum_{i=0}^{n} w_i N_{ik}(t)}$$

A continuación se muestra la programación en Matlab

```
function [Bt0]=Rational_Bspline_Curve(P,w,T,k,nt,type_Bs,varargin)

% This function computes the Rational Bspline Curve with weights
%
% [Bt0]=Rational_Bspline_Curve(P,w,T,k,nt,type_Bs,varargin);
%
% Input variables:
```

```
%
% P control points
% w weights vector associated with the control points
% T knot vector satifying
%       number of knots= number of control points + order of the
Bspline
% k order of the basis function
% nt number of points to discretize the interval T(1):T(m)
% type_Bs  'P' for Periodic, 'NP' for Non Periodic
% t0 (optional) point where we want to evaluate
%
% Output variables:
%
% Bt0 point in the curve for value of t=t0
%
% Example1:
%
[Bt0]=Rational_Bspline_Curve([1,2;2,4;3,2;4,1;5,3],[1,2,2,2,1],[0,1,2,
3,4,5,6,7],3,1000,'P',2.5);
%
% Example2:
%
[Bt0]=Rational_Bspline_Curve([1,2;2,4;3,2;4,1;5,3],[1,2,1,0.5,2],[0,0,
0,1,2,3,3,3],3,1000,'NP',2.5);
%
% Example3: one circle
% [Bt0]=Rational_Bspline_Curve([1+sqrt(0.1875),0.75;1,(0.75)/(1-
sqrt(0.1875));1-
sqrt(0.1875),0.75;0,0;1,0;2,0;1+sqrt(0.1875),0.75],[1,1/2,1,1/2,1,1/2,
1],[0,0,0,1/3,1/3,2/3,2/3,1,1,1],3,1000,'NP');


% We obtain the length of the knot vector and the number of control
points


m=length(T);
n=size(P,1);
s=length(w);


% We check that the equation m=n+k is satisfied

if m~=n+k
    str=['The number of knots must be equal to the number of control
points plus the order k of the Bspline'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end

% We check that we have a weight for each point

if n~=s
    str=['The number of points must be equal to the lenght of weights
vector w'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end
```

```
% We see if there is any t0 to evaluate

if nargin==6

    % We initialize Bt0 as empty when there is no t0 where to evaluate
    Bt0=[];

elseif nargin==7

    t0=varargin{1};

end



% We discretize the knot vector

t=linspace(T(k),T(n+1),nt);

switch type_Bs

    case 'P'

        aux=T(1:k-1); aux=diff(aux);
        aux1=T(n+2:m); aux1=diff(aux1);

        if ~all(aux)| ~all(aux1)
            str=['The vector of knots must be periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end


    case 'NP'

        aux=T(1:k-1); aux=aux-T(1);
        aux1=T(n+2:m); aux1=aux1-T(m);

        if any(aux)| any(aux1)
            str=['The vector of knots must be non-periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end


end



% We calculate the point in the curve for each value of t

for it=1:nt

    xt=[0,0];    % It is going to contain only the numerator of the
```

```
                % rational B-spline till a final division

    % Initialization of the denominator of the Rational B-spline xt
for
    % each value of t

    d=0;

    % Expression for computing the Bspline curve

    for in=1:n
        xt=xt+w(in)*P(in,:)*Bspline(T,in,k,t(it));
        d=d+w(in)*Bspline(T,in,k,t(it));
    end

    % We perfom the mentioned division in Rational B-splines
    xt=xt/d;

    % We plot in 2D the curve
    plot(xt(1),xt(2),'ko-','MarkerSize',3);
    hold on

end


% We plot the control polygon


plot(P(1:in,1),P(1:in,2),'r-');
plot(P(1:in,1),P(1:in,2),'bo','MarkerFaceColor','b');

mx=min(P(1:in,1));
Mx=max(P(1:in,1));
my=min(P(1:in,2));
My=max(P(1:in,2));
axis([mx-0.1*(Mx-mx) Mx+0.1*(Mx-mx) my-0.1*(My-my) My+0.1*(My-my)]);


% We evaluate at t0 if it is the case

if nargin==7

    % Initialization of the denominator of the Rational B-spline xt
for
    % t=t0

    d=0;

    xt=[0,0];
    for in=1:n
        xt=xt+w(in)*P(in,:)*Bspline(T,in,k,t0);
        d=d+w(in)*Bspline(T,in,k,t0);
    end

    % We perfom the mentioned division in Rational B-splines
    xt=xt/d;
    Bt0=[xt(1),xt(2)];
```

```
    % We plot in 2D the point
    plot(xt(1),xt(2),'go','MarkerSize',10);

end
```

Ejemplo 1

>>
[Bt0]=Rational_Bspline_Curve([1,2;2,4;3,2;4,1;5,3],[1,2,2,2,1],[0,1,2,3,4,5,6,7],3,1000,'P',2.5)

Bt0 = 2.0667 3.6000

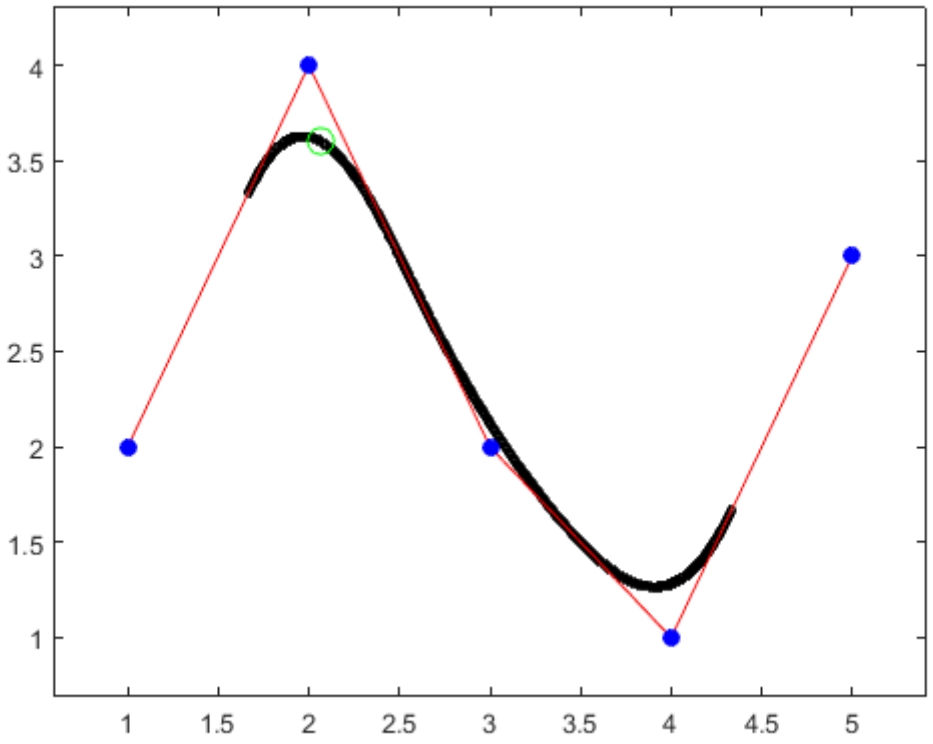


Fig 6.5.1: ejemplo 1 curva B-spline racional periódica.

Ejemplo 2

>>
[Bt0]=Rational_Bspline_Curve([1,2;2,4;3,2;4,1;5,3],[1,2,1,0.5,2],[0,0,0,1,2,3,3,3],3,1000,'NP',2.5)

Bt0 = 4.4000 2.2000

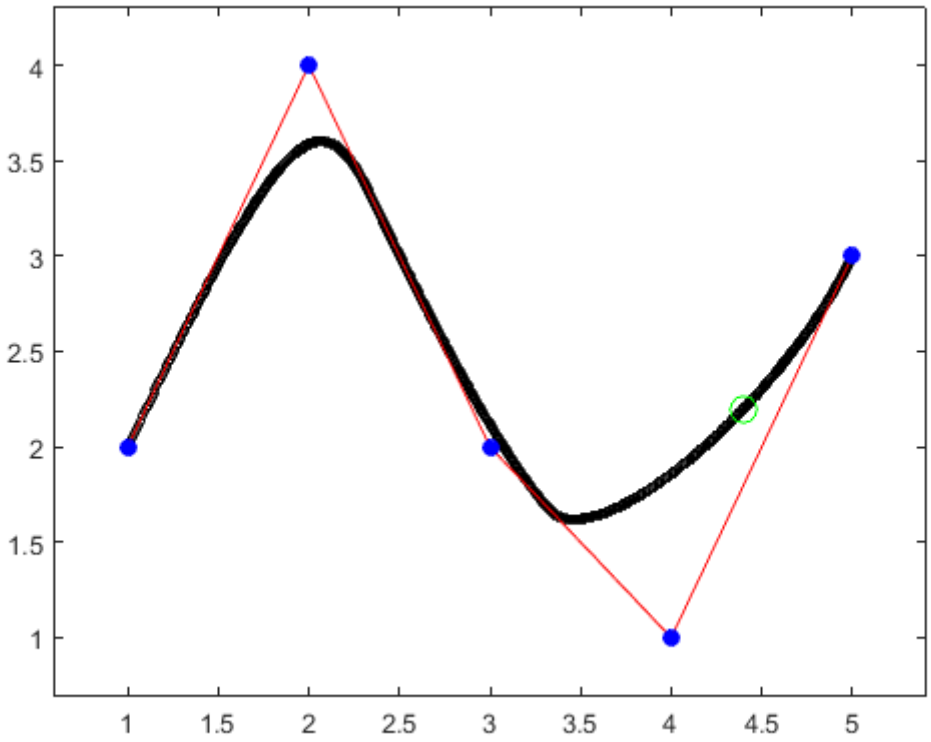


Fig 6.5.2: ejemplo 2 curva B-spline racional no periódica.

Ejemplo 3

>> [Bt0]=Rational_Bspline_Curve([1+sqrt(0.1875),0.75;1,(0.75)/(1-sqrt(0.1875));1-sqrt(0.1875),0.75;0,0;1,0;2,0;1+sqrt(0.1875),0.75],[1,1/2,1,1/2,1,1/2,1],[0,0,0,1/3,1/3,2/3,2/3,1,1,1],3,1000,'NP');

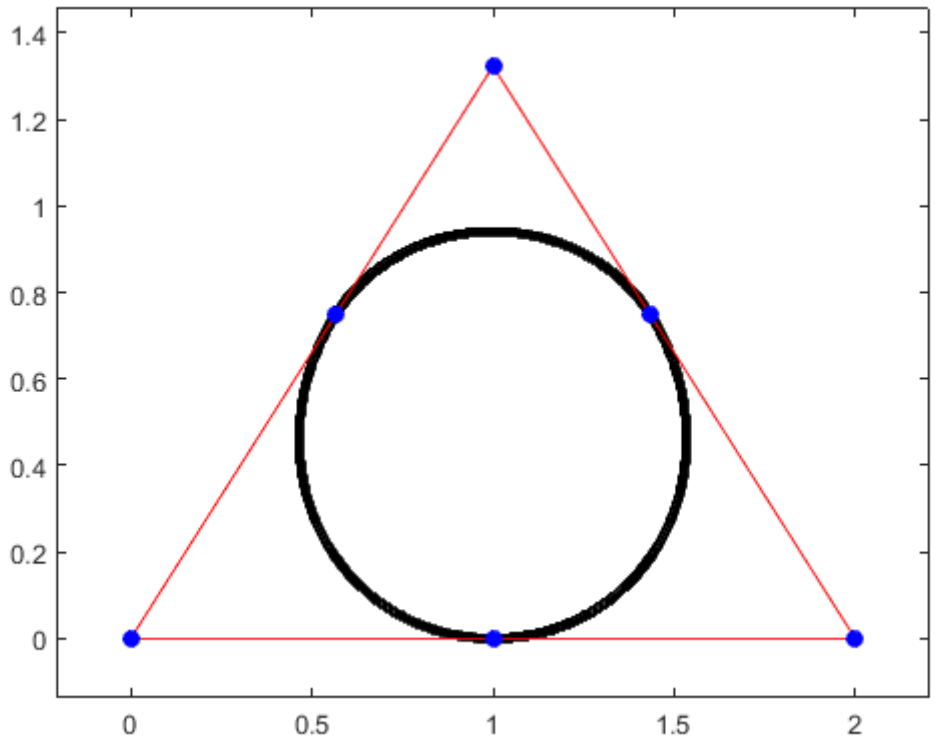


Fig 6.5.3: ejemplo 3 curva B-spline racional no periódica.

A continuación se muestra el mismo programa, pero para tres dimensiones

```
function [Bt0]=Rational_Bspline_Curve3(P,w,T,k,nt,type_Bs,varargin)

% This function computes the Rational Bspline Curve with weights
%
% [Bt0]=Rational_Bspline_Curve3(P,w,T,k,nt,type_Bs,varargin);
%
% Input variables:
%
% P 2D cell array containing the three spatial coordinates of the
control points
% w weights vector associated with the control points
% T knot vector satifying
%       number of knots= number of control points + order of the
Bspline
% k order of the basis function
% nt number of points to discretize the interval T(1):T(m)
% type_Bs  'P' for Periodic, 'NP' for Non Periodic
% t0 (optional) point where we want to evaluate
%
% Output variables:
%
% Bt0 point in the curve for value of t=t0
%
% Example1:
% syms u
% C(u)=[cos(u),sin(u),u];
% nu=20;
```

```
% ud=linspace(0,2*pi,nu);
% P=cell(nu,1);
% for i=1:nu
%     P{i}=double(subs(C(u),u,ud(i)));
% end
% w=ones(1,nu); w(1)=100; w(nu)=100;
% k=4;
% T=1:nu+k;
% t0=10.5;
% nt=100;
% [Bt0]=Rational_Bspline_Curve3(P,w,T,k,nt,'P',t0);


% We obtain the length of the knot vector and the number of control
points


m=length(T);
n=size(P,1);
s=length(w);


% We check that the equation m=n+k is satisfied

if m~=n+k
    str=['The number of knots must be equal to the number of control
points plus the order k of the Bspline'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end

% We check that we have a weight for each point

if n~=s
    str=['The number of points must be equal to the lenght of weights
vector w'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end



% We see if there is any t0 to evaluate

if nargin==6

    % We initialize Bt0 as empty when there is no t0 where to evaluate
    Bt0=[];

elseif nargin==7

    t0=varargin{1};

end
```

```
% We discretize the knot vector

t=linspace(T(k),T(n+1),nt);

switch type_Bs

    case 'P'

        aux=T(1:k-1); aux=diff(aux);
        aux1=T(n+2:m); aux1=diff(aux1);

        if ~all(aux)| ~all(aux1)
            str=['The vector of knots must be periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end


    case 'NP'

        aux=T(1:k-1); aux=aux-T(1);
        aux1=T(n+2:m); aux1=aux1-T(m);

        if any(aux)| any(aux1)
            str=['The vector of knots must be non-periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end


end

% We calculate the point in the curve for each value of t

xt=cell(nt,1);


for it=1:nt-1

    xt{it}=[0,0,0];   % It is going to contain only the numerator of
the
                % rational B-spline till a final division

    % Initialization of the denominator of the Rational B-spline xt
for
    % each value of t

    d=0;

    % Expression for computing the Bspline curve

    for in=1:n
        xt{it}=xt{it}+w(in)*P{in}*Bspline(T,in,k,t(it));
        d=d+w(in)*Bspline(T,in,k,t(it));
    end

    % We perfom the mentioned division in Rational B-splines
```

```
    xt{it}=xt{it}/d;

    % We plot in 2D the curve
    Paux=xt{it};
    plot3(Paux(1),Paux(2),Paux(3),'ko-','MarkerSize',3);
    hold on

end


% We plot the control polygon

for i=1:n
      aux=P{i};
      Px(i)=aux(1);
      Py(i)=aux(2);
      Pz(i)=aux(3);
end

plot3(Px,Py,Pz,'r-');
plot3(Px,Py,Pz,'bo','MarkerFaceColor','b');

mx=min(Px);
Mx=max(Px);
my=min(Py);
My=max(Py);
mz=min(Pz);
Mz=max(Pz);
axis([mx-0.1*(Mx-mx) Mx+0.1*(Mx-mx) my-0.1*(My-my) My+0.1*(My-my) mz-
0.1*(Mz-mz) Mz+0.1*(Mz-mz) ]);


% We evaluate at t0 if it is the case

if nargin==7

    % Initialization of the denominator of the Rational B-spline xt
for
    % t=t0

    d=0;

    xt=[0,0,0];
    for in=1:n
        xt=xt+w(in)*P{in}*Bspline(T,in,k,t0);
        d=d+w(in)*Bspline(T,in,k,t0);
    end

    % We perfom the mentioned division in Rational B-splines
    xt=xt/d;
    Bt0=[xt(1),xt(2),xt(3)];

    % We plot in 2D the point
    plot3(xt(1),xt(2),xt(3),'go','MarkerSize',10);

end
```

Ejemplo

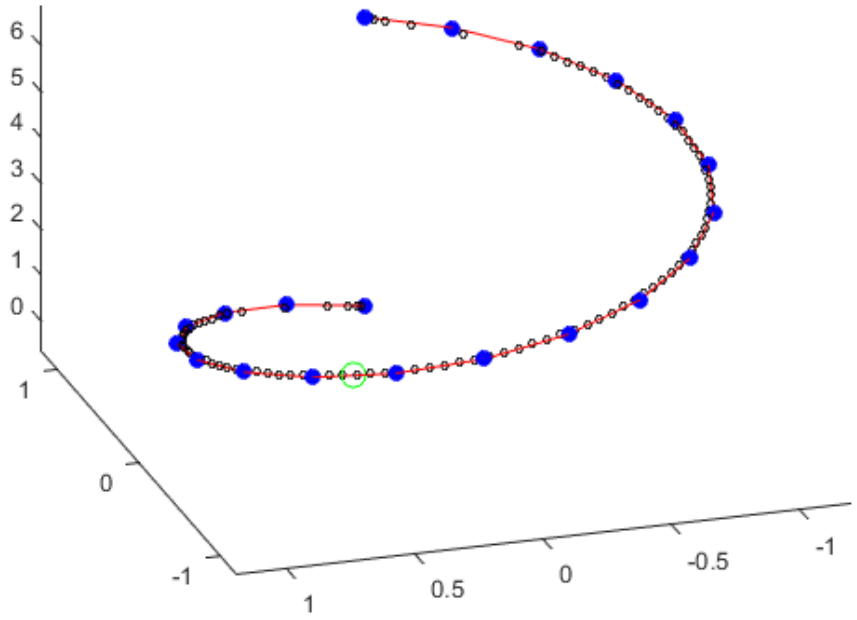


Fig 6.5.4: ejemplo curva B-spline racional en 2D.

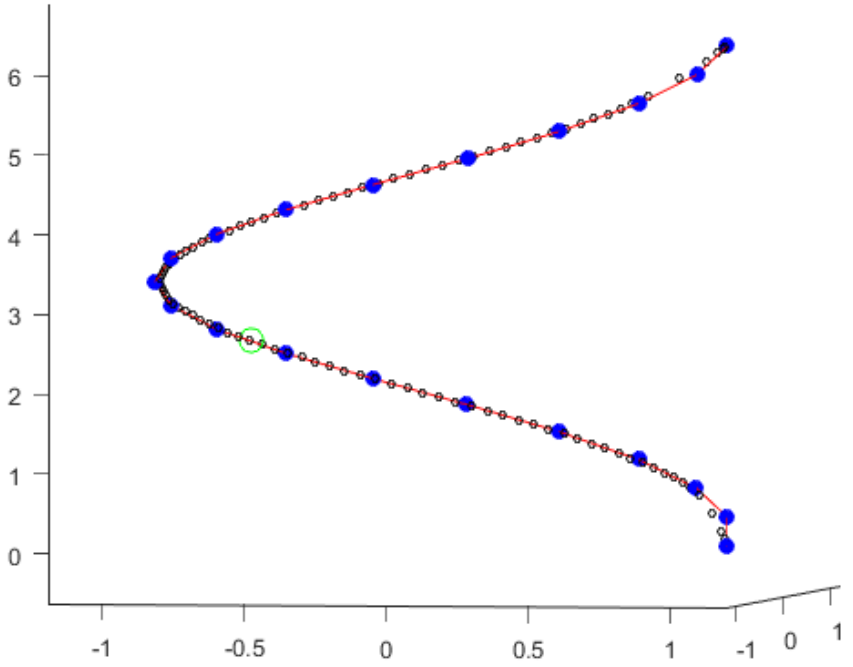


Fig 6.5.5: ejemplo curva B-spline racional en 2D.

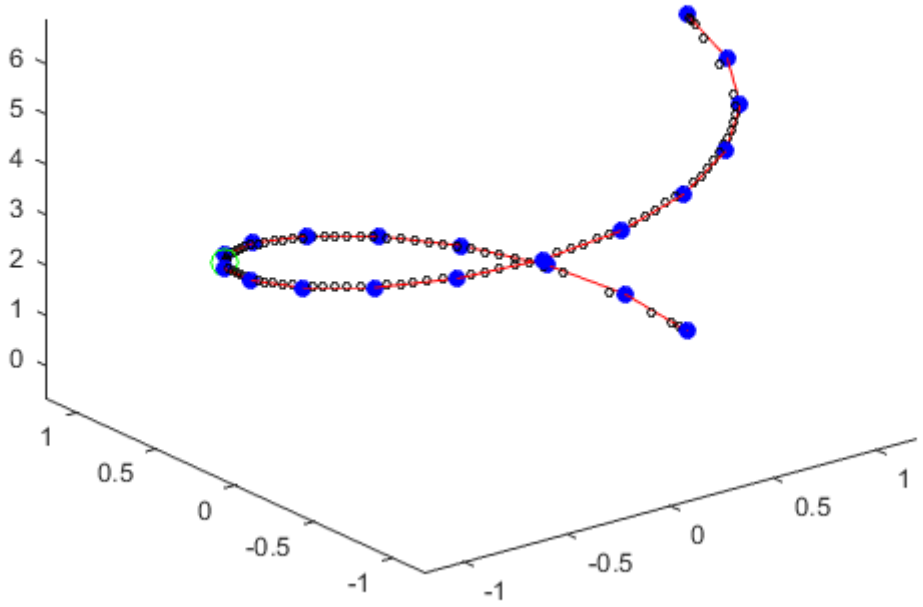


Fig 6.5.6: ejemplo curva B-spline racional en 2D.

7. NURBS [6, 10, 18, 19, 20, 23]

7.1 INTRODUCCIÓN

Cualquier objeto en tres dimensiones se compone de superficies y curvas. Las maneras más usuales que tenemos para representar una curva o superficie son de forma implícita (dos funciones dependiente de las variables de los ejes igualadas a cero en el caso de curvas o una función dependiente de las variables de los ejes igualada a cero en el caso de superficies) o de forma paramétrica (cada variable es función de un parámetro independiente en el caso de curvas o de dos parámetros independientes en el caso de superficies).

En dos dimensiones sólo podríamos hablar de curvas, en dicho caso una curva queda definida por una ecuación implícita o por su parametrización que expresa la posición en el plano sobre la curva para cada valor del parámetro t.

Un ejemplo de curva plana sería la circunferencia de centro el origen y radio 1, que queda representada por

$f\ (x,y) = x^2 + y^2 - 1 = 0$ de forma implícita.

$C(t) = [x(t), y(t)] \quad a \leq t \leq b \quad de$ forma paramétrica.

Si utilizamos la forma paramétrica para representar el primer cuadrante del circulo, esta representación no es única, sino que tiene varias formas:

1) $C(t) = \left[\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2}\right] \quad 0 \leq t \leq 1,$

2) $C(t) = [\cos t,\ \sin t\ ] \quad \sigma \leq t \leq \frac{\pi}{2},$

por ejemplo, ya que se puede parametrizar de muchas otras maneras.

Las NURBS son una clase de curvas y superficies paramétricas, las cuales se utilizan debido a que son procesadas con facilidad mediante ordenador, requieren de poca memoria y son estables ante un error en punto flotante y tienen facilidad para representar ciertas curvas y superficies.

Una curva NURBS sin nodos interiores, es una curva de Bézier racional. A demás las curvas NURBS contienen B-SPLINE no racionales y curvas de Bézier racionales y no racionales como casos especiales. De hecho NURBS significa Non Uniform Rational B-splines, es decir, que generaliza las curvas de Bézier permitiendo nodos interiores, a la vez que también generaliza los B-splines permitiendo asignar pesos de atracción a los puntos de control iniciales de la curva o superficie. Por tanto, el material contenido en este capítulo es necesariamente igual que el que podría encontrarse en el capítulo de B-splines y B-splines racionales.

Las NURBS están definidas por cuatro elementos fundamentales:

- Puntos de control: estos definen una aproximación a la curva. Podremos modificar la forma de la superficie moviendo en el espacio estos puntos. Cuando hablamos de una curva estos puntos forman un vector de puntos y el polígono que se forma al unir los puntos se llama polígono de control, es importante que el diseñador sepa que todos los puntos de control están dentro de este polígono, de manera que el primer y último punto son sus extremos. Que una curva este o no dentro de un plano se debe a que los

puntos de control de ésta estén o no en ese mismo plano, es decir, que la coordenada z de ese plano sea constante.

- Pesos: corresponden a la coordenada homogénea, en un espacio de cuatro dimensiones y existiendo únicamente en curvas y superficies racionales. Estos pesos van ligados a los puntos de control, de manera que si el peso de un punto es mayor que otro, este tendrá mayor atracción sobre la curva.

    - Si w= 1 la curva no se verá modificada con respecto a la curva no racional.
    - Si w>1 la atracción de ese punto respecto a la curva será mayor, produciéndose una aproximación de la curva a dicho punto.
    - Si w<1 se produciría un alejamiento de la curva con respecto al punto teniendo como limite la línea recta que une el punto de control anterior y siguiente.

  Nunca trabajaremos con un peso de valor negativo.

- Vector de nodos: parte la curva de manera que las funciones base, dependiendo el intervalo entre nodos en el que este, sean diferentes para obtener una mejor definición. En el caso de un B-spline clásico y como ya hemos visto antes tendrá sus nodos distribuidos de manera uniforme. Como las NURBS son un B-spline racional no uniforme, sus nodos no tienen esa distribución uniforme. Además como es racional tiene pesos y por tanto podemos actuar sobre ellos y lograr mayor o menor atracción de cada punto de control. Si hablamos de una curva sólo tendremos un vector de nodos en la dirección U y si estuviésemos hablando de una superficie habría un vector para cada dirección que serían U y V.

- Grado: la curva es representada por un polinomio de grado uno menos el orden de la curva, de manera que las curvas de segundo orden (representadas por polinomios lineales) se llaman curvas lineales, las curvas de tercer orden se llaman curvas cuadráticas y las curvas de cuarto orden se denominan curvas cúbicas, es decir, el grado nos indicará si la curva es cuadrática, cúbica, cuártica, etc.

Hemos de saber que al cambiar o modificar la posición de un punto de control, la superficie cambiará de manera local. Si eliminamos puntos de control se realizarán cambios en la superficie pudiendo tener una curva final distinta a la inicial. Igual sucede si insertamos nuevos puntos de control. En ambos casos la curva se verá afectada localmente.

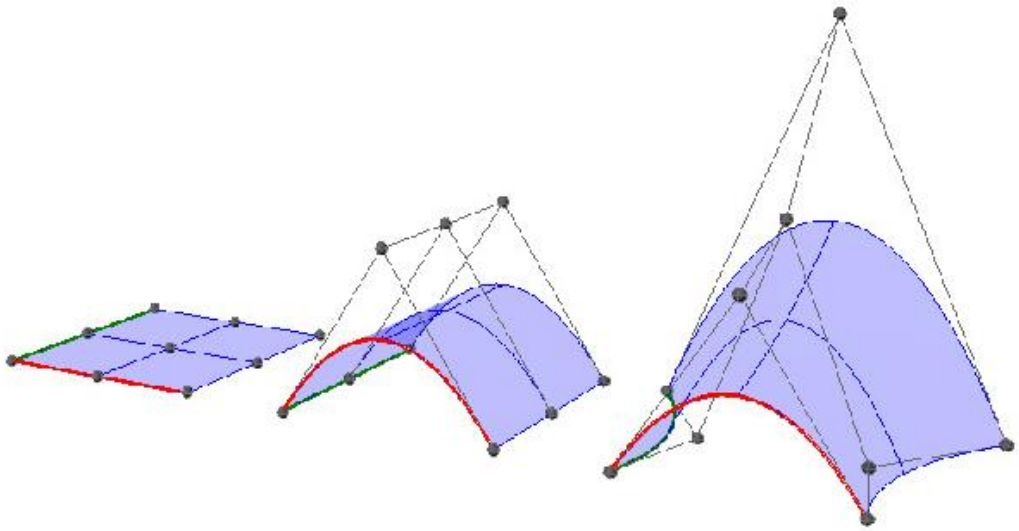


Fig. 7.1.1: variación de una misma superficie NURB.

Una de las causas por las que utilizar las curvas NURBS es su facilidad a la hora de controlar la suavidad, lo que nos permite obtener curvas sin cambios de dirección ni torceduras o si las hubiese tener un control exacto. En la figura 7.1.1 vemos la variación sufrida por una superficie NURB mediante la modificación de algunos puntos de control.

7.1 CURVAS NURBS Y SU DEFINICIÓN

Ya hemos visto cómo trabajan las funciones paramétricas, de manera que podemos emplearlas para definir las curvas NURBS. Teniendo una función a la que llamaremos Q y la cual evaluaremos en t para obtener una serie (x, y) con la que dibujar la curva.

7.1.1 PUNTOS DE CONTROL

Como ya hemos visto anteriormente la forma de la curva viene modelada por los puntos de control, los cuales se unen mediante líneas formando el denominado polígono de control. La curva cambia únicamente en el entorno cercano a un punto de control si éste se modifica. Esta característica nos permite realizar cambios en ciertos lugares sin cambiar la forma total de la curva, ya que cada punto de control solo tiene influencia sobre la zona de la curva más cercana a él, sin tener o tener muy poco protagonismo en las que están más alejadas. La curva viene expresada de la siguiente forma,

$$Q(t) = \sum_{i=0}^{n} B_i N_{i,k}(t).$$

7.1.2 FUNCIONES DE BASE

La función $N_{i,k}(t)$ tiene como valor un número real, dónde k determina la influencia que tiene el punto B_i sobre la curva en el tiempo t.

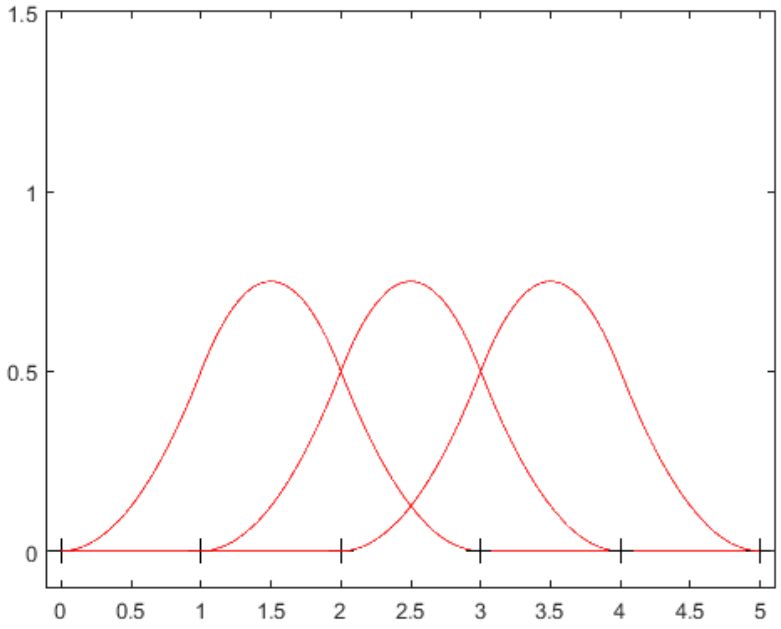


Fig. 7.1.2: funciones base.

En la figura 7.1.2 anterior tenemos un ejemplo de cómo podría ser una función base. Como podemos ver tiene su máximo efecto en un punto t concreto y su apuntamiento depende de la separación entre nodos. La función base influye de menor manera conforme uno se aleja de este punto.

Como ya hemos dicho anteriormente las curvas NURBS "Non Uniform Rational B-Spline" son una generalización de las curvas B-Spline, con la que también podemos representar cónicas mediante secciones de curvas racionales que se unen con ciertos grados de diferenciabilidad. Por tanto las siglas NURBS indican que la curva tiene funciones coordenadas racionales en términos de funciones base B-Spline definidas sobre un vector de nodos no uniforme.

La siguiente expresión muestra la definición de una curva NURBS de forma matemática. Dados los enteros positivos n, k, m, los puntos de control P_0, …, P_n, y los pesos w_0, …,w_n, una curva NURBS de grado k está definida por:

$$R(t) = \frac{\sum_{i=0}^{n} P_i \, w_i \, N_{i,k}\,(t)}{\sum_{i=0}^{n} w_i \, N_{i,k}\,(t)} \quad a \le t \le b.$$

donde cada $N_{i,k}\,(t)$ es una función base B-Spline sobre un vector de m+1 nodos no decrecientes y no uniformes.

$$U = \{a, \dots, a, u_{k+1}, \dots, u_{m-k-1}, b, \dots, b\},$$

donde el valor de a se repite k veces al inicio del vector y el valor de b también se repite k veces al final del vector si consideramos como en este caso condiciones de frontera no periódicas.

Si tomamos U={0,…,0,1,…,1} y n=p tenemos:

$$R(t) = \frac{\sum_{i=0}^{n} P_i \, w_i \, B_{i,k}}{\sum_{j=0}^{n} w_i \, B_{j,k}\,(t)} \qquad 0 \le t \le 1.$$

La cual es una expresión que define a la curva de Bézier racional.

Por otra parte si w_i=c con c constante para toda i, la propiedad de partición de la unidad de las funciones B-Spline implica que:

$$R(t) = \frac{c \sum_{i=0}^{n} P_i \, N_{i,p}\,(t)}{c \sum_{i=0}^{n} N_{i,p}\,(t)} = \sum_{i=0}^{n} N_{i,p}\,(t) P_i.$$

Esta ecuación coincide con la de la curva B-Spline.

8. BSPLINES EN DOS DIMENSIONES [2, 13]

Hoy en día las superficies de Bézier, y herramientas más sofisticadas como las superficies B-Spline y las superficies NURBS se utilizan como herramientas de diseño gráfico en programas como Adobe Photoshop, Adobe Illustrator, Inkscape, etc. O en programas de animación gráfica para controlar el movimiento en aplicaciones como Adobe Flash, Adobe After Effects, Adobe Shockwave, entre otros. También y esto es de nuestro particular interés se utilizan en programas de diseño naval como Rhinoceros y Maxsurf por citar dos.

A lo largo de este capítulo hablaremos y describiremos las superficies de Bézier, las superficies B-Spline y las superficies B-Spline racionales.

8.1 SUPERFICIES DE BÉZIER

A continuación veremos cómo aumentar la dimensión a partir de las curvas de Bézier definidas en el capítulo 4 para conseguir superficies de Bézier. Las técnicas que necesitamos para su construcción son las usuales para trabajar en varias dimensiones, y consisten fundamentalmente en trabajar en cada una de las dimensiones por separado.

Teniendo un conjunto finito de $(m+1)(n+1)$ puntos de $\mathbb{R}^3$ que nombraremos red de control $\{P_{i,j}\}_{(i,j)=0}^{(m,n)}$, definimos la superficie de Bézier χ: [0,1] x [0,1]→$\mathbb{R}^3$ como

$$\chi(\mathrm{u},\mathrm{v}) = \sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)\, B_j^n(v) P_{i,j} \ ,$$

$$(u,v) \in [0,1]x[0,1].$$

donde:

$B_i^m(u) = N_i^{m,m+1}(u)$ es el B-spline unidimensional de orden k=m+1 para la columna i, que en este caso coincide con la expresión de los polinomios de Bernstein.

$B_j^n(v) = N_j^{n,n+1}(v)$ es el B-spline unidimensional de orden k=n+1 para la fila j, que en este caso coincide con la expresión de los polinomios de Bernstein.

$P_{i,j}$ $forma\ una\ red\ bidireccional\ de\ puntos$.

8.2 SUPERFICIES B-SPLINE

A continuación seguiremos la misma técnica que se ha utilizado en las superficies de Bézier de atacar el problema dos dimensional vía la técnica del producto tensor que se basa en calcular los splines de una dimensión y multiplicarlos. Es decir, las técnicas que se usan para su construcción son las inmediatas para pasar a varias dimensiones. Tal y como se ha explicado en superficies de Bézier consisten en trabajar en cada dimensión separadamente. Vamos a explicarlas en el rectángulo [0,1] x [0,1], si bien las adaptaciones para trabajar en cualquier otro rectángulo general son obvias.

Teniendo un conjunto finito de $(m+1)(n+1)$ puntos de $\mathbb{R}^3$ que nombraremos red de control $\{P_{i,j}\}_{(i,j)=0}^{(m,n)}$, definimos la superficie B-spline χ: [0,1] x [0,1]→$\mathbb{R}^3$ como

$$\chi(\mathrm{u},\mathrm{v}) = \sum_{i=0}^{m}\sum_{j=0}^{n} N_i^{m,k}(u)\, N_j^{n,k}(v) P_{i,j} \ ,$$

$$(u,v) \in [0,1]x[0,1].$$

donde:

$N_i^{m,k}(u)$ es el B-spline unidimensional de orden k para la columna i.

$N_j^{n,k}(v)$ es el B-spline unidimensional de orden k para la fila j.

$P_{i,j}\ forma\ una\ red\ bidireccional\ de\ puntos.$

8.3 SUPERFICIES B-SPLINE RACIONALES

Estas superficies son una generalización de las superficies B-Spline. Una superficie B-Spline racional χ de orden k en la dirección u y orden l en la dirección v se define mediante la siguiente expresión:

$$\chi(\mathrm{u},\mathrm{v}) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} N_i^{m,k}(u)\, N_j^{n,l}(v) P_{i,j}\, w_{ij}}{\sum_{i=0}^{m}\sum_{j=0}^{n} N_i^{m,k}(u)\, N_j^{n,l}(v) w_{i,j}}$$

donde

$N_i^{m,k}(u)$ es el B-spline unidimensional de orden k para la columna i,

$N_j^{n,l}(v)$ es el B-spline unidimensional de orden l para la fila j.

$P_{i,j}\ forma\ una\ red\ bidireccional.$

$w_{ij}\ son\ los\ pesos, los\ cuales\ asumiremos\ que\ son\ mayores\ que\ cero\ para\ todo\ i,j.$

A continuación se muestra el programa elaborado en Matlab para el caso más general de las superficies B-splines racionales con vectores de nodos cualesquiera, es decir NURBS. El programa viene seguido de dos ejemplos:

```
function
[Pu0v0]=Rational_Bspline_Surface3(ControlPoints,W,U,k,V,l,nU,nV,typeU_
Bs,typeV_Bs,varargin)


% This function computes the Rational B-spline surface with weights
associated with the knot

% vectors U and V in each orthogonal direction and with the control
points
```

```
% P in a given bidirectional net
%
%
[Pu0v0]=Rational_Bspline_Surface3(ControlPoints,W,U,k,V,l,nU,nV,typeU_
Bs,typeV_Bs,varargin);
%
% Input variables:
% ControlPoints is a variable which gives the control points. They can
be
%              introduced in two different ways:
%
%              P rectangular matrix containing the value z of the
control points that belong to a given
%                bidirectional net
%
%              filename string containing the name of the Excel file
which in turn contains the rectangular array
%                with the control points. This information will be
converted in a 2D cell array using the progam
%                Excel2Surface
%
% W rectangular matrix containing the weights associated to each
control
%   point
% U knot vector in one direction satifying
%       number of knots in U= number of control points by row + order
k of the Bspline
% k order of the B-spline
% V knot vector in the second direction satifying
%       number of knots in V= number of control points by column +
order l of the Bspline
% l order of the B-spline
% nU number of points to discretize the interval U(1):U(p)
% nV number of points to discretize the interval V(1):V(q)
% typeU_Bs  'P' for Periodic, 'NP' for Non Periodic
% typeV_Bs  'P' for Periodic, 'NP' for Non Periodic
% [u0,v0] (optional) point inside the cartesian product [U(k),U(pp-
k+1)]x[V(1),V(q-1+1)] where we want to evaluate
%
```

```
% Output variables:
% Pu0v0 point in the surface for value of u=u0,v=v0
%
% Example1:
% syms x y
% n=15; m=10;
% xm=linspace(0,2*pi,m);
% ym=linspace(0,2*pi,n);
% [Xm,Ym]=meshgrid(xm,ym);
% Zm=double(subs('sin(x+y)',{x,y},{Xm,Ym}));
% P=cell(n,m);
% for i=1:n
%    for j=1:m
%       P{i,j}=[Xm(i,j),Ym(i,j),Zm(i,j)];
%    end
% end
% W=rand(size(P));
% k=4;
% U=1:n+k;U=U';
% l=4;
% V=1:m+l; V=V';
% nU=50;
% nV=50;
% typeU_Bs='P';
% typeV_Bs='P';
% u0v0=[5,5];
%
[Pu0v0]=Rational_Bspline_Surface3(P,W,U,k,V,l,nU,nV,typeU_Bs,typeV_Bs,
u0v0)
%
% Example2:
% syms x y
% n=15; m=10;
% xm=linspace(0,2*pi,m);
% ym=linspace(0,2*pi,n);
```

```
% [Xm,Ym]=meshgrid(xm,ym);
% Zm=double(subs('sin(x+y)',{x,y},{Xm,Ym}));
% P=cell(n,m);
% for i=1:n
%    for j=1:m
%       P{i,j}=[Xm(i,j),Ym(i,j),Zm(i,j)];
%    end
% end
% W=ones(size(P)); W(1,1:m)=100; W(n,1:m)=100; W(1:n,1)=100;
W(1:n,m)=100;
% k=4;
% U=1:n+k;U=U';
% l=4;
% V=1:m+l; V=V';
% nU=50;
% nV=50;
% typeU_Bs='P';
% typeV_Bs='P';
% u0v0=[5,5];
%
[Pu0v0]=Rational_Bspline_Surface3(P,W,U,k,V,l,nU,nV,typeU_Bs,typeV_Bs,
u0v0)
%
% Example3:
% clear all;
% ControlPoints='PALA HÉLICE_1.xlsx';
% n=11;
% m=42;
% W=ones(n,m); W(n,1:m)=1; W(1:n,1)=10000; W(1:n,m)=10000;
% k=4;
% U(1:k)=1; U(k+1:n)=2:n-k+1; U(n+1:n+k)=n-k+2; U=U';
% l=4;
% V(1:l)=1; V(l+1:m)=2:m-l+1; V(m+1:m+l)=m-l+2; V=V';
% nU=50;
% nV=50;
% typeU_Bs='NP';
```

```
% typeV_Bs='NP';
%
[Pu0v0]=Rational_Bspline_Surface3(ControlPoints,W,U,k,V,l,nU,nV,typeU_
Bs,typeV_Bs)
% axis equal
%
% Example4:
% clear all;
% ControlPoints='PALA HÉLICE_2.xlsx';
% n=13;
% m=42;
% W=ones(n,m); W(n,1:m)=1; W(1:n,1)=10000; W(1:n,m)=10000;
% k=4;
% U(1:k)=1; U(k+1:n)=2:n-k+1; U(n+1:n+k)=n-k+2; U=U';
% l=4;
% V(1:l)=1; V(l+1:m)=2:m-l+1; V(m+1:m+l)=m-l+2; V=V';
% nU=50;
% nV=50;
% typeU_Bs='NP';
% typeV_Bs='NP';
%
[Pu0v0]=Rational_Bspline_Surface3(ControlPoints,W,U,k,V,l,nU,nV,typeU_
Bs,typeV_Bs)
% axis equal




% We obtain the control points

if ischar(ControlPoints)

    P=Excel2Surface(ControlPoints);

else

    P=ControlPoints;
```

```
    clear ControlPoints;

end



% We obtain the size of the control points matrix


[n,m]=size(P);


% We get the number of knots in each direction

p=length(U);
q=length(V);


% We check that the equations p=m+k, q=n+l is satisfied

if p~=n+k
    str=['The number of knots in the x direction must be equal to the
number of control points in this direction plus the order k of the
Bspline'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
elseif q~=m+l
    str=['The number of knots in the y direction must be equal to the
number of control points in this direction plus the order l of the
Bspline'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end

% We see if there is any u0v0 to evaluate
```

```
if nargin==10

    % We initialize Bt0 as empty when there is no t0 where to evaluate
    Pu0v0=[];

elseif nargin==11

    u0v0=varargin{1};

end



% We discretize the knot vectors

 u=linspace(U(k),U(n+1),nU);
 v=linspace(V(l),V(m+1),nV);

switch typeU_Bs

    case 'P'

        aux=U(1:k); aux=diff(aux);
        aux1=U(n+1:p); aux1=diff(aux1);

        if ~all(aux)| ~all(aux1)
            str=['The vector U of knots must be periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end

    case 'NP'

        aux=U(1:k); aux=aux-U(1);
        aux1=U(n+1:p); aux1=aux1-U(p);
```

```
        if any(aux)| any(aux1)
            str=['The vector U of knots must be non-periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end


end


switch typeV_Bs

    case 'P'

        aux=V(1:l); aux=diff(aux);
        aux1=V(m+1:q); aux1=diff(aux1);

        if ~all(aux)| ~all(aux1)
            str=['The vector V of knots must be periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end

    case 'NP'

        aux=V(1:l); aux=aux-V(1);
        aux1=V(m+1:q); aux1=aux1-V(q);

        if any(aux)| any(aux1)
            str=['The vector V of knots must be non-periodic'];
            uiwait(msgbox(str, 'Error message','error','modal'))
            return;
        end
```

```
end



% We calculate the point in the surface for each value of u and v

Niuiv=cell(nU,nV);

for iu=1:nU-1
    for iv=1:nV-1

    Niuiv{iu,iv}=[0,0,0];
    Diuiv{iu,iv}=[0,0,0];

    % Expression for computing the Bspline surface

    for im=1:m
        for in=1:n

Niuiv{iu,iv}=Niuiv{iu,iv}+P{in,im}*W(in,im).*Bspline(U,in,k,u(iu))*Bsp
line(V,im,l,v(iv));

Diuiv{iu,iv}=Diuiv{iu,iv}+W(in,im)*Bspline(U,in,k,u(iu))*Bspline(V,im,
l,v(iv));

        end
    end

    Piuiv{iu,iv}=Niuiv{iu,iv}./Diuiv{iu,iv};

    end
end
```

```
for iu=1:nU-1
    for iv=1:nV-1
        aux=Piuiv{iu,iv};
        Piuiv_x(iu,iv)=aux(1);
        Piuiv_y(iu,iv)=aux(2);
        Piuiv_z(iu,iv)=aux(3);
    end
end


%plot3(Piuiv_x,Piuiv_y,Piuiv_z,'r-','LineWidth',3);
surf(Piuiv_x,Piuiv_y,Piuiv_z)
hold on

% We plot the control points

for im=1:m
    for in=1:n
        aux=P{in,im};
        Px(in,im)=aux(1);
        Py(in,im)=aux(2);
        Pz(in,im)=aux(3);
    end
end

plot3(Px,Py,Pz,'bo','MarkerFaceColor','b','MarkerSize',5);


% We evaluate at Pu0v0 if it is the case

if nargin==11

    Nu0v0=[0,0,0];
    Du0v0=[0,0,0];
```

```
    for im=1:m
        for in=1:n


Nu0v0=Nu0v0+P{in,im}*W(in,im)*Bspline(U,in,k,u0v0(1))*Bspline(V,im,l,u
0v0(2));

Du0v0=Du0v0+W(in,im).*Bspline(U,in,k,u0v0(1))*Bspline(V,im,l,u0v0(2));


        end
    end


    Pu0v0=Nu0v0./Du0v0;


    % We plot the point in the surface
    Pu0v0_x=Pu0v0(1); Pu0v0_y=Pu0v0(2); Pu0v0_z=Pu0v0(3);

plot3(Pu0v0_x,Pu0v0_y,Pu0v0_z,'go','MarkerSize',7,'MarkerFaceColor','g
');
else
    Pu0v0=[];
end
```

A continuación se muestran también los programas para pasar bidireccionalmente de una estructura de datos tipo celda la cual contiene los puntos de control, a un archivo Excel con el mallado rectangular que contiene la información de dichos puntos de control:

-Primer programa:

```
function Surface2Excel(P,filename,varargin)


% This function writes in an Excel file a 2D cell array containing the
% control points of a given surface in a rectangular net.
% The Excel file is organized as follows:
% One section in each row without empty rows
% For each section the points must be placed from the first to the
last and
% each point occupying three consecutive cells
%
% Surface2Excel(P,filename,varargin)
```

```
%
% Input variables:
% P  2D cell array containing the control points
% filename string with the name of the file where the data of the
control
%   control points is written
% varargin optional variable indicating the number of the sheet in
case
%          the excel file has several sheets
%
% Example 1:
% syms x y
% n=15; m=10;
% xm=linspace(0,2*pi,m);
% ym=linspace(0,2*pi,n);
% [Xm,Ym]=meshgrid(xm,ym);
% Zm=double(subs('sin(x+y)',{x,y},{Xm,Ym}));
% P=cell(n,m);
% for i=1:n
%    for j=1:m
%       P{i,j}=[Xm(i,j),Ym(i,j),Zm(i,j)];
%    end
% end
% filename='Example1.xls';
% Surface2Excel(P,filename)

% We get the size of the 2D cell array
[n,m]=size(P);

% We transform the data to a matrix array
A=zeros(n,3*m);

for i=1:n
    for j=1:m
        aux=P{i,j};
```

```
        j3=3*(j-1)+1;
        A(i,j3)=aux(1);
        A(i,j3+1)=aux(2);
        A(i,j3+2)=aux(3);
    end
end

% We write the matrix A to the Excel file
if nargin==3
    xlswrite(filename,A,sheet);
else
    xlswrite(filename,A);
end
```

-Segundo programa, que realiza la operación inversa.

```
function P=Excel2Surface(filename,varargin)

% This function transform a Excel file containing the rectangular
array
% of the control points into a 2D cell array.
% The control points must be organized in the Excel as follows:
% One section in each row without empty rows
% For each section the points must be placed from the first to the
last and
% each point occupying three consecutive cells
%
% P=Excel2Surface(filename,varargin)
%
% Input variables:
% filename is a string containing the name of the Excel file
% varargin optional variable indicating the number of the sheet in
case
%          the excel file has several sheets
%
```

```
% Output variables:
% P 2D cell array containing the control points
%
% Example:
% P=Excel2Surface(filename)

% We detect if the Excel file has several sheets

if nargin==2
    sheet=varargin{1};
    % We read the Excel file
    A=xlsread(filename,sheet);
else
    A=xlsread(filename);
end

% We determine the size of the matrix A
[na,ma]=size(A);

if mod(ma,3)~=0
    str=['The number of columns in the Excel file must be divisible by
3'];
    uiwait(msgbox(str, 'Error message','error','modal'))
    return;
end

% We define the size of the cell array containing the control points
n=na;
m=ma/3;

% We build the 2D cell array
P=cell(n,m);

for i=1:n
    for j=1:m
```

```
        ja=3*(j-1)+1;
        P{i,j}=A(i,ja:ja+2);
    end
end
```

Ejemplo 1

```
>> syms x y
n=15; m=10;
xm=linspace(0,2*pi,m);
ym=linspace(0,2*pi,n);
[Xm,Ym]=meshgrid(xm,ym);
Zm=double(subs('sin(x+y)',{x,y},{Xm,Ym}));
P=cell(n,m);
for i=1:n
  for j=1:m
    P{i,j}=[Xm(i,j),Ym(i,j),Zm(i,j)];
  end
end
W=rand(size(P));
k=4;
U=1:n+k;U=U';
l=4;
V=1:m+l; V=V';
nU=50;
nV=50;
typeU_Bs='P';
typeV_Bs='P';
u0v0=[5,5];
[Pu0v0]=Rational_Bspline_Surface3(P,W,U,k,V,l,nU,nV,typeU_Bs,typeV_Bs,u0v0);
```

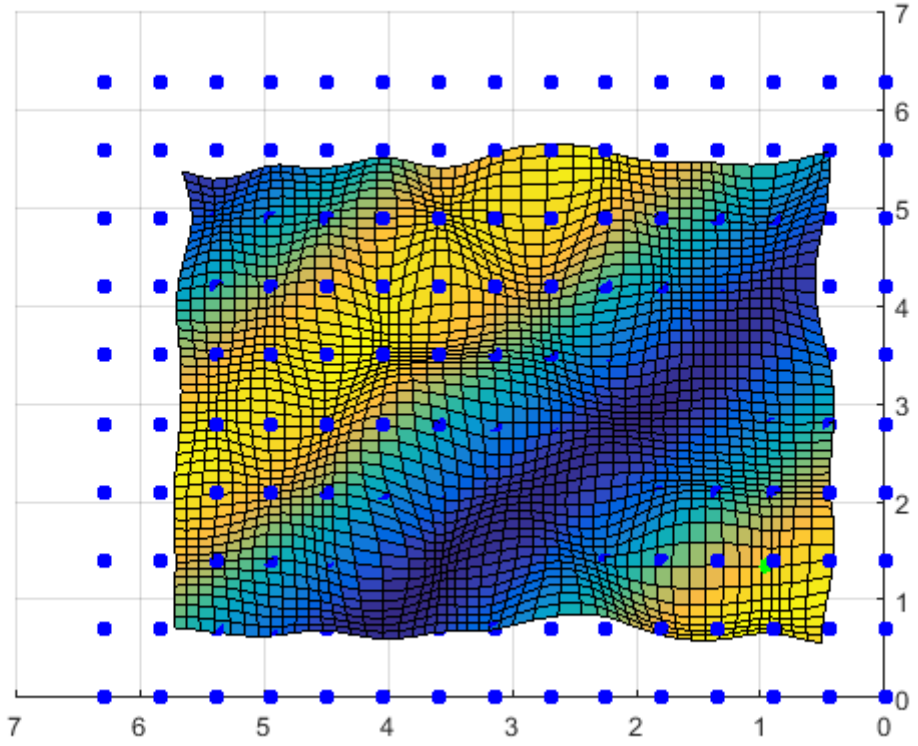


Fig 8.3.1: superficie B-spline racional

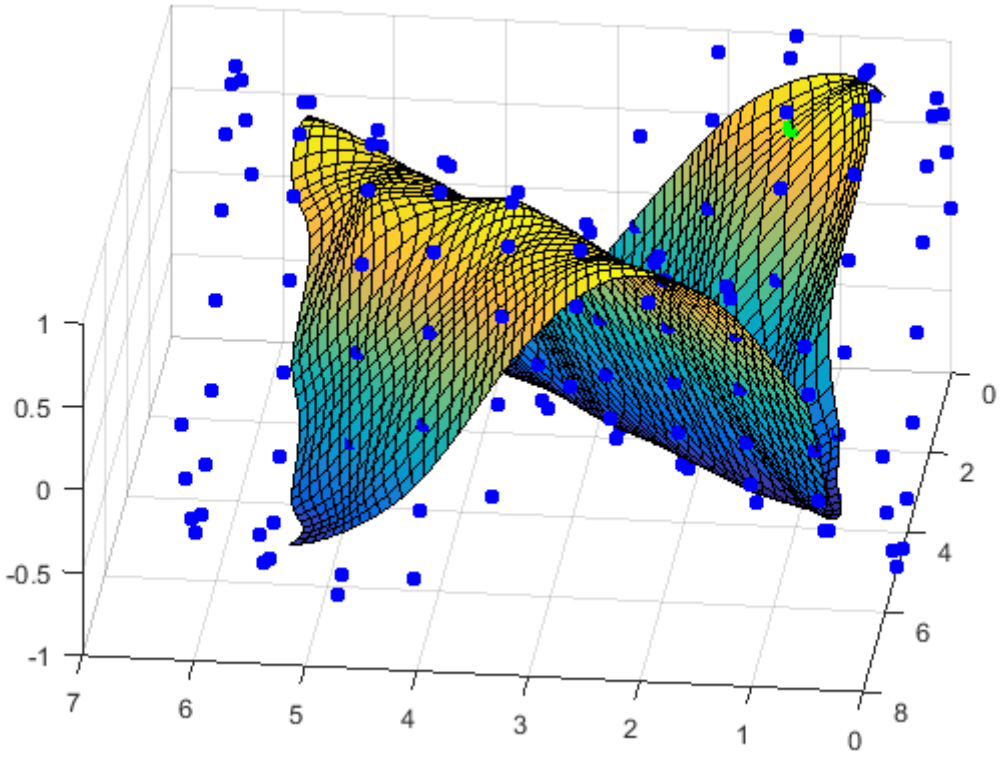


Fig 8.3.2: superficie B-spline racional

Ejemplo 2

```
>> syms x y
n=15; m=10;
xm=linspace(0,2*pi,m);
ym=linspace(0,2*pi,n);
[Xm,Ym]=meshgrid(xm,ym);
Zm=double(subs('sin(x+y)',{x,y},{Xm,Ym}));
P=cell(n,m);
for i=1:n
  for j=1:m
    P{i,j}=[Xm(i,j),Ym(i,j),Zm(i,j)];
  end
end
W=ones(size(P)); W(1,1:m)=100; W(n,1:m)=100; W(1:n,1)=100; W(1:n,m)=100;
k=4;
U=1:n+k;U=U';
l=4;
V=1:m+l; V=V';
nU=50;
nV=50;
typeU_Bs='P';
typeV_Bs='P';
u0v0=[5,5];
[Pu0v0]=Rational_Bspline_Surface3(P,W,U,k,V,l,nU,nV,typeU_Bs,typeV_Bs,u0v0)
```

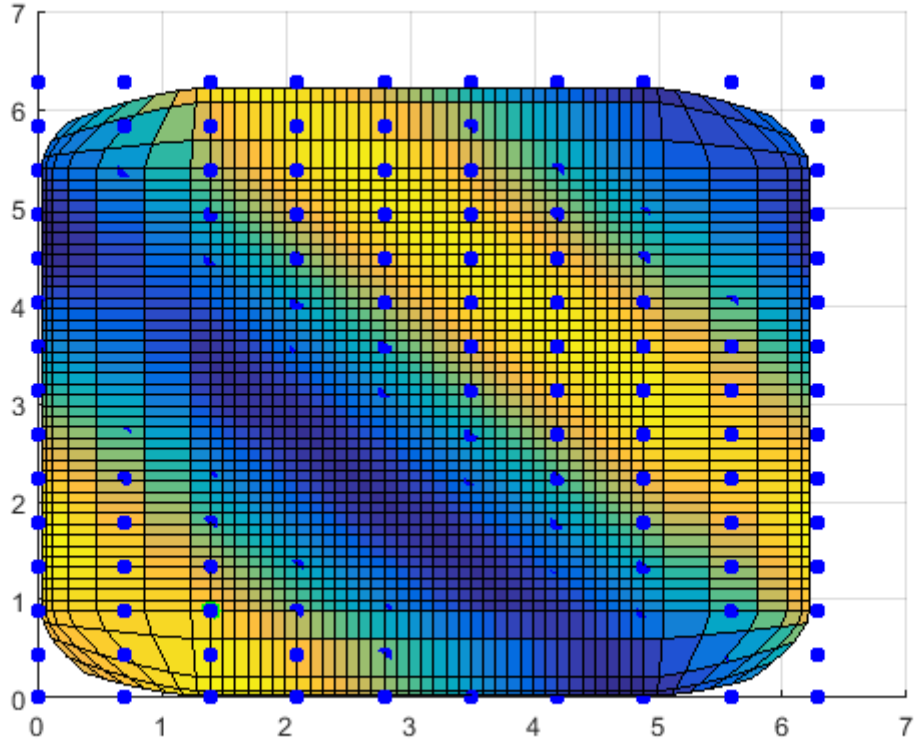


Fig 8.3.3: superficie B-spline racional

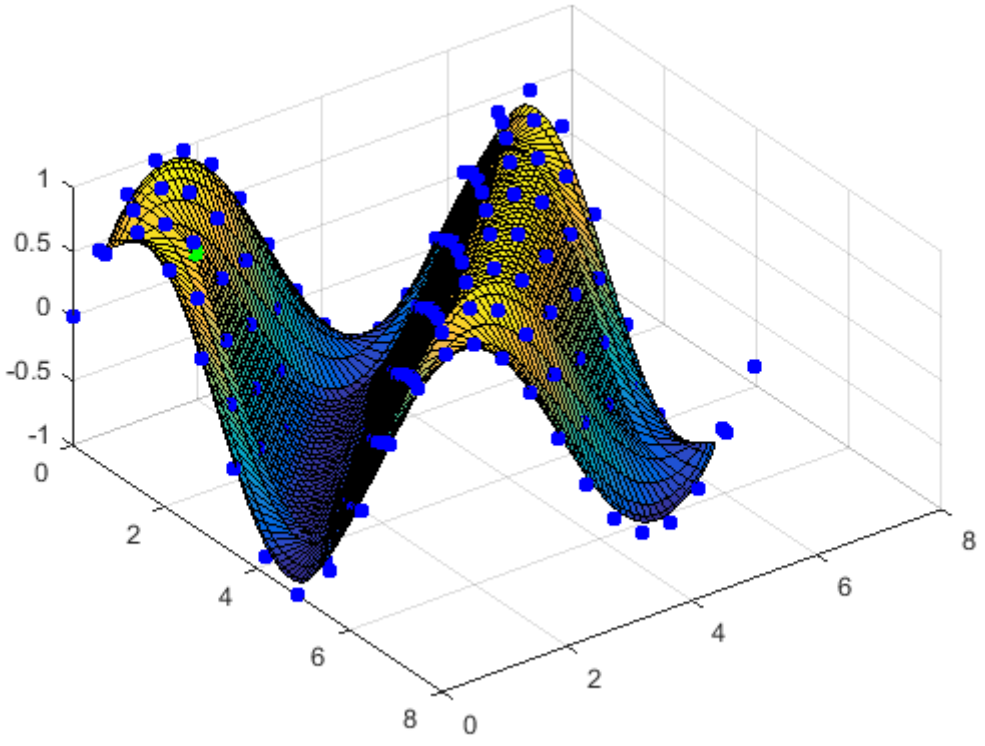


Fig 8.3.4: superficie B-spline racional

8.4 SUPERFICIERS NURBS

Como las superficies Nurbs están basadas en B-splines no uniformes racionales generalizan los dos casos anteriores 8.1 Y 8.2. En este apartado vamos a explicar la construcción para un rectángulo cualquiera $[a, b]x[c, d]$.

Teniendo un conjunto finito de $(m + 1)(n + 1)$ puntos de $\mathbb{R}^3$ que nombraremos red de control $\{P_{i,j}\}_{(i,j)=0}^{(m,n)}$, definimos la superficie NURBS χ: $[a, b]x[c, d]$→$\mathbb{R}^3$ como

$$\chi(\mathrm{u}, \mathrm{v}) = \sum_{i=0}^{m} \sum_{j=0}^{n} N_i^{m,k}(u)\, N_j^{n,k}(v) P_{i,j} \ ,$$

$$(u, v) \in [a, b]x[c, d]$$

donde

$N_i^{m,k}(u)$ es el B-spline no uniforme racional de orden k unidimensional para la columna con índice i, $N_j^{n,k}(v)$ es el B-spline no uniforme racional de orden k unidimensional para la fila j.

9. INTERFAZ GRÁFICA [7]

Hemos utilizado el entorno gráfico de Matlab (GUIDE) para la realización de una interfaz gráfica la cual posibilite a cualquier usuario la realización tanto de polinomios de Bernstein como la construcción de curvas y superficies. Se podrán utilizar métodos basados tanto en representaciones de Bézier como B-spline, de una manera fácil y rápida sin la obligación de poseer amplios conocimientos de Matlab.

9.1 EJECUCIÓN DE LA INTERFAZ

Una vez tenemos abierto el programa Matlab, en el directorio de trabajo tendremos que seleccionar la carpeta correspondiente a los programas de la interfaz gráfica.

Dicha interfaz se inicia escribiendo en la línea de comandos del Programa Matlab el comando que se muestra a continuación:

>> TFG_NURBS

Ejecutado el comando anterior nos aparecerá directamente la interfaz principal del programa para el usuario, figura 9.1.1.

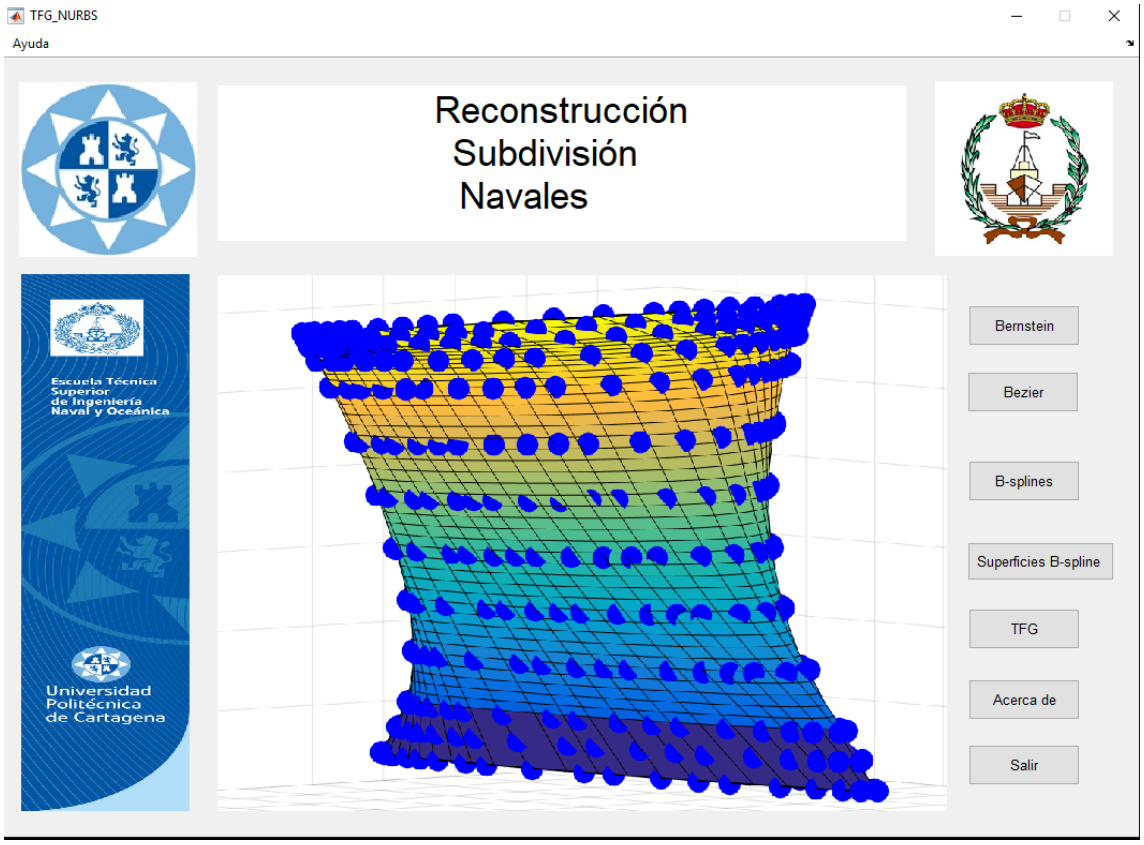


Fig 9.1.1: interfaz principal del programa de usuario.

9.2 DOCUMENTACIÓN DE LA INTERFAZ GRÁFICA

En este apartado vamos a describir nuestra interfaz principal de usuario, figura 9.1.1, la cual, como hemos podido observar se divide en siete pushbuttons y un menú de ayuda, los cuales pasaremos a explicar a continuación.

9.2.1 AYUDA

Tal y como podemos observar en la figura 9.2.1.1, este desplegable alberga un tutorial de manejo de la interfaz así como el trabajo de fin de grado por el cual se ha realizado.

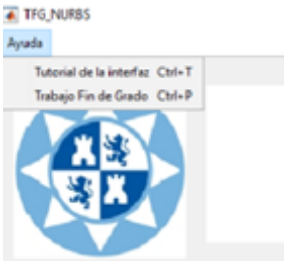


Fig 9.2.1.1: ayuda.

- **Tutorial de la interfaz**

  Si seleccionamos esta opción, nos aparecerá el tutorial con indicaciones sobre cómo utilizar la interfaz gráfica.

- **Trabajo Fin de Grado**

  Si seleccionamos esta otra opción, nos aparecerá el pdf con el Trabajo Fin de Grado.

En el resto del capítulo describimos con cierto detalle la manera de trabajar con cada una de las interfaces que se abren al presionar los diferentes pushbuttons que aparecen en esta pantalla de inicio. Empezaremos con la primera de las siete opciones decicada a los polinomios de Bernstein.

9.2.2 BERNSTEIN

Este pushbutton nos abre una nueva interfaz, la cual vemos en la siguiente figura:

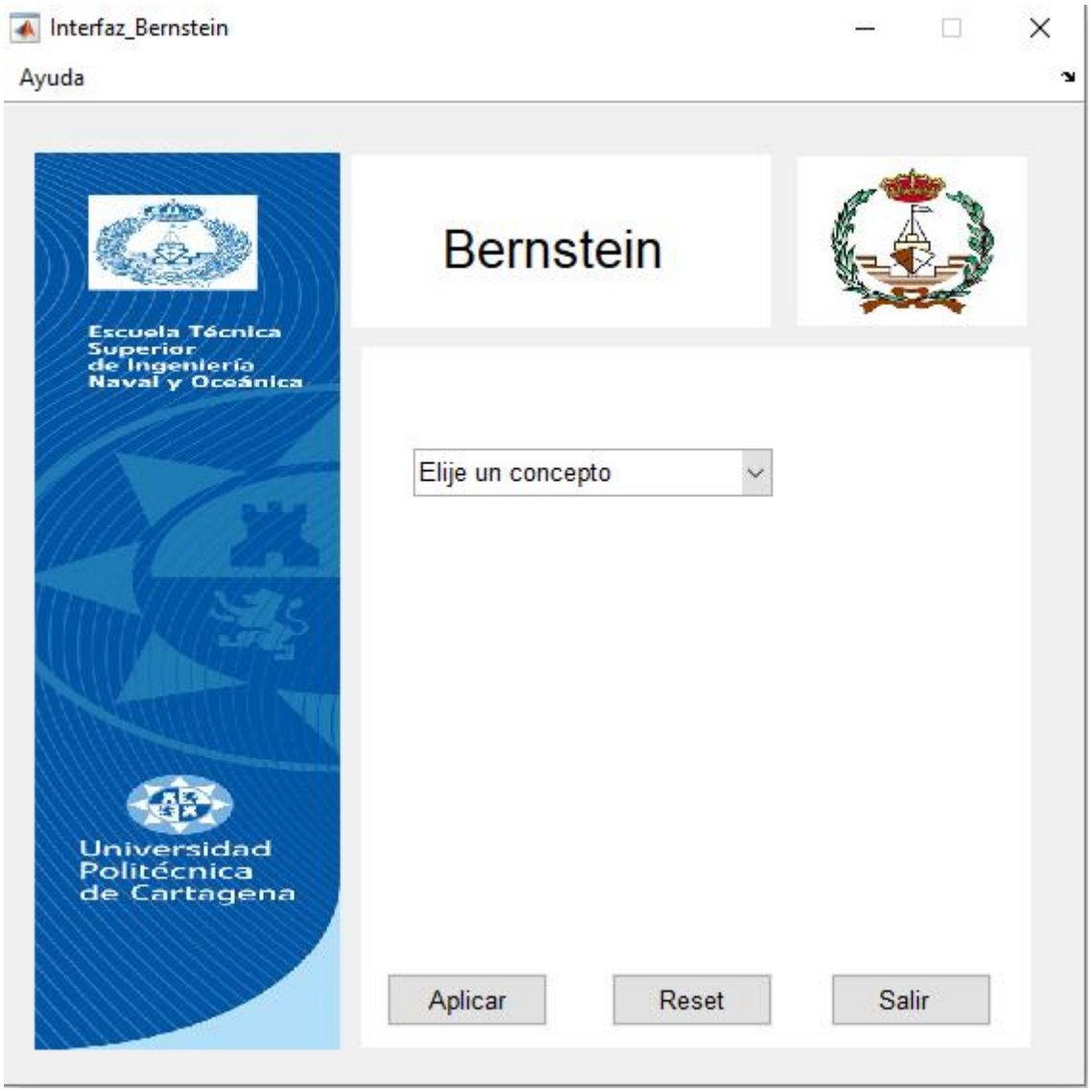


Fig 9.2.2.1: Interfaz_Bernstein.

A continuación se describen cada uno de los pushbuttons de dicha interfaz:

- **<u>Ayuda</u>**
  Tal y como podemos observar en la figura 9.2.2.2, este pushbutton alberga un tutorial de manejo de la interfaz así como información referente tanto a los Polinomios de Bernstein como al Teorema de Weierstrass:

Fig 9.2.2.2: Ayuda.

- **Tutorial de la interfaz**

  Si seleccionamos esta opción, nos aparecerá el tutorial con indicaciones sobre cómo utilizar la interfaz gráfica.

- **Polinomios de Bernstein**

  Haciendo click sobre esta opción, nos aparecerá la teoría referente a dichos polinomios.

- **Teorema de Weierstrass**

  Esta otra opción, nos proporciona el enunciado y la explicación del Teorema de aproximación de funciones de Weierstrass.

- **<u>Elije un concepto</u>**

  Haciendo click sobre este desplegable, tenemos la opción de elegir entre Polinomios de Bernstein o el Teorema de Weierstrass.

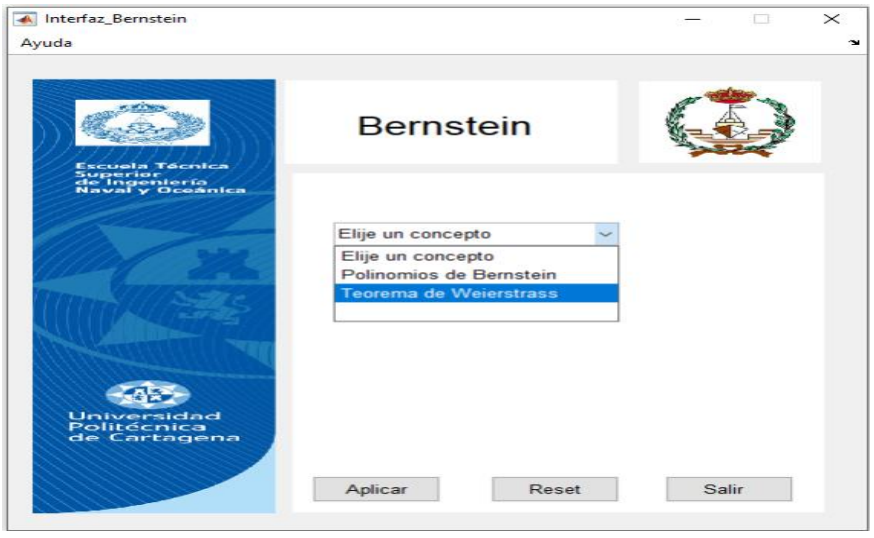


Fig 9.2.2.3: Elije un concepto.

A continuación explicamos con detalle cómo trabajar con cada uno de estos conceptos.

- **Polinomios de Bernstein**

  Seleccionando esta opción nos aparece lo siguiente:

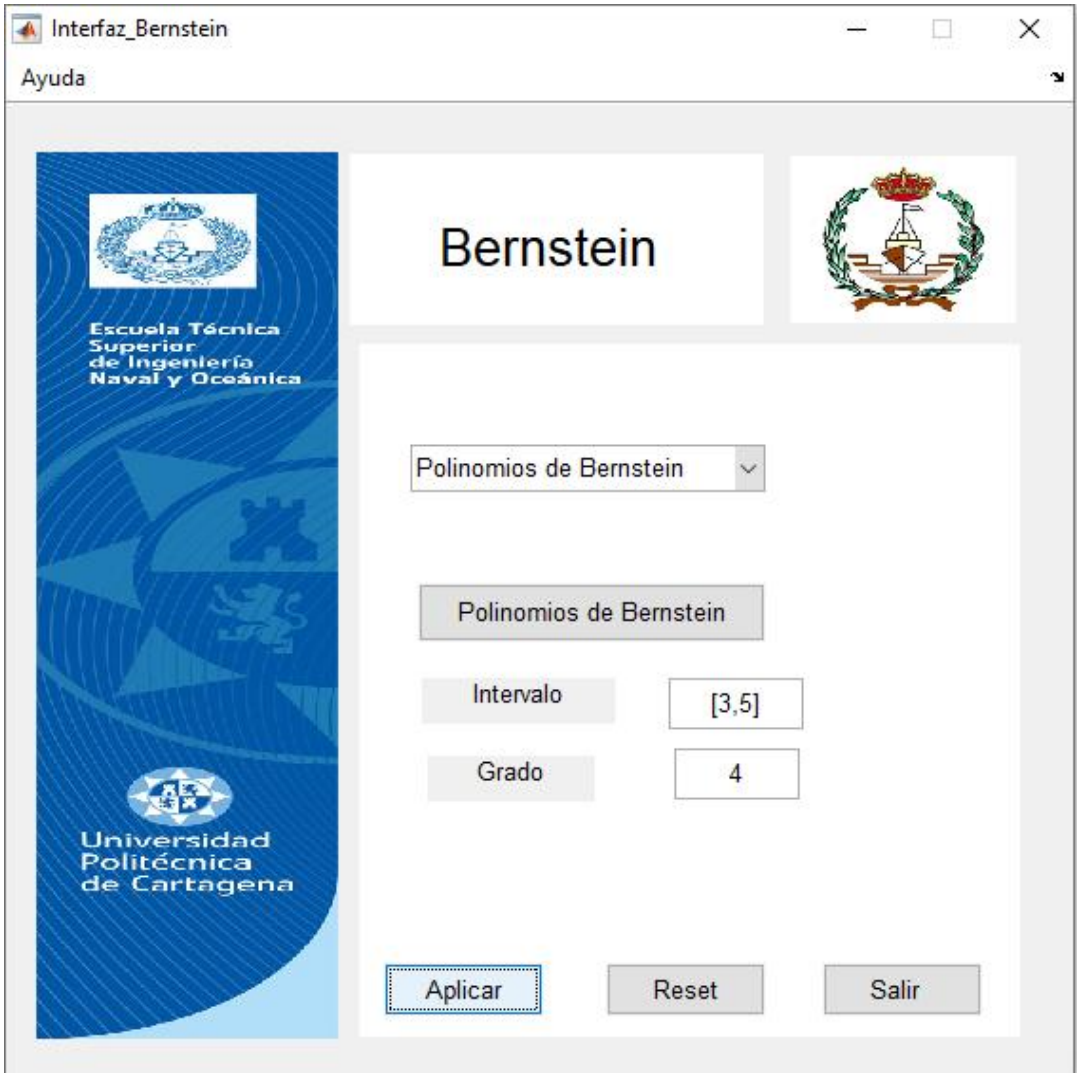


Fig 9.2.2.4: Polinomios de Bernstein.

Como podemos ver en la figura anterior, dentro de esta opción podemos seleccionar tanto el grado como el intervalo de definición del polinomio, y a continuación podemos:

- **Aplicar**
  Si le damos a Aplicar, se nos generarán los polinomios hasta el grado que hayamos dicho y en el intervalo que hayamos introducido.
- **Reset**
  Si pulsamos sobre reset, se nos pondrán de nuevo los datos por defecto.
- **Salir**

- **Teorema de Weierstrass**
  Al pulsar sobre la opción de teorema de Weierstrass nos aparece la siguiente interfaz con los datos que se ven por defecto:

Fig 9.2.2.5: Teorema de Weiestrass.

A continuación describiremos los distintos elementos de dicha interfaz:

- Función: aquí podemos seleccionar el tipo de función que queremos aproximar.
- Intervalo: aquí pondremos el intervalo en el que vamos a aproximar la función.

- nf de la discretización: en cuantos puntos vamos a discretizar el intervalo para dibujar tanto la función como la aproximación, y medir la distancia entre ambas.
- Grado n: es el grado de aproximación de la fórmula de Weiestrass.
- Distancia: aquí introduciremos la distancia máxima permitida en la aproximación.
- Gráficas: aquí introduciremos mediante un vector las gráficas que queremos que nos dibuje.

A continuación se mostraran dos ejemplos de uso de dicha interfaz, uno introduciendo una distancia máxima a cumplir y otro indicando simplemente el grado de la aproximación:

- Ejemplo 1: con distancia máxima

Fig 9.2.2.6: ejemplo 1, aplicación de la interfaz Teorema de Weiestrass con distancia.

Fig 9.2.2.7: ejemplo 1, aplicación de la interfaz Teorema de Weiestrass con distancia.

Como podemos ver una vez introducidos todos los datos y presionando el pushbutton Aplicar se nos muestra una venta que nos indica que el programa está calculando y que esperemos. Una vez terminados los cálculos el programa nos muestra las siguientes ventanas:

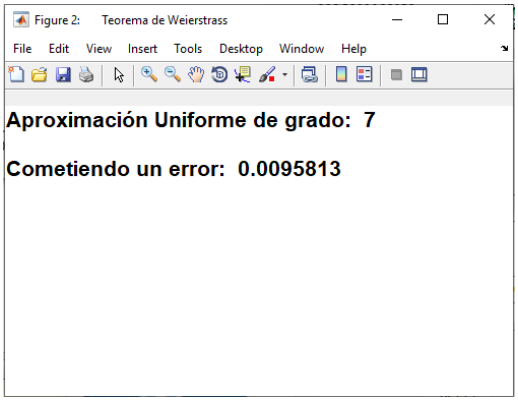


Fig 9.2.2.8: ejemplo 1, aplicación de la interfaz Teorema de Weiestrass con distancia.

Como podemos ver en la figura anterior, nos dice que a utilizado un polinomio de grado siete para realizar la aproximación y cumplir con la distancia máxima que nosotros le hemos indicado así como también nos dice a la distacia que se ha quedado respecto a la función original.

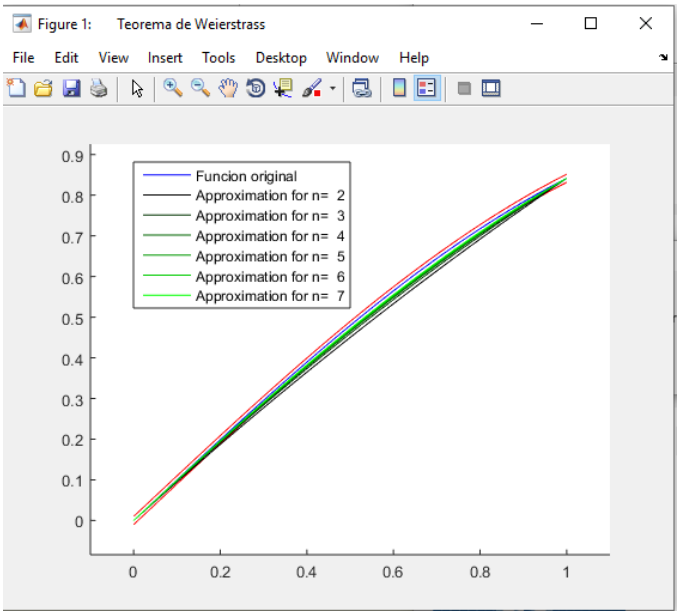


Fig 9.2.2.9: ejemplo 1, aplicación de la interfaz Teorema de Weiestrass con distancia.

Como podemos ver en la figura 9.2.2.8, nos dice que ha utilizado un polinomio de grado siete para realizar la aproximación y cumplir con la distancia máxima que nosotros le hemos indicado, así como también nos dice a la distancia que se ha quedado realmente respecto a la función original, que cumple la tolerancia permitida.

En la figura 9.2.2.9 tenemos la representación de la función original y de las distintas aproximaciones que se han realizado. Como podemos observar el programa nos representa la aproximación hasta grado 7 desde el grado 2, a pesar de que nosotros indicamos que el grado fuese 5 y nos representase desde grado 0 hasta el grado 5. Esto sucede para que se cumpla el límite de la distancia máxima que nosotros introdujimos y para ello se necesita una aproximación de grado mínimo n=7, y nos representa las últimas cinco aproximaciones. Se hace notar que esta es la forma de entender las entradas del vector de gráficas.

	A	B	C	D	E	F	G	H	I
1	0	0	0	0	0	0	0	0	0
2	0,001001	0,001001	0	0,00084231	0,00095969	0,00098246	0,00099051	0,00099426	0,0009963
3	0,002002	0,002002	0	0,00168463	0,00191915	0,0019647	0,00198084	0,00198836	0,00199247
4	0,003003	0,003003	0	0,00252694	0,00287837	0,00294672	0,00297099	0,00298231	0,0029885
5	0,004004	0,00400399	0	0,00336925	0,00383736	0,00392853	0,00396094	0,00397609	0,00398438
6	0,00500501	0,00500498	0	0,00421157	0,00479611	0,00491012	0,00495071	0,00496971	0,00498013
7	0,00600601	0,00600597	0	0,00505388	0,00575463	0,0058915	0,0059403	0,00596317	0,00597573
8	0,00700701	0,00700695	0	0,00589619	0,00671291	0,00687265	0,00692969	0,00695647	0,0069712
9	0,00800801	0,00800792	0	0,00673851	0,00767096	0,00785359	0,0079189	0,00794961	0,00796652
10	0,00900901	0,00900889	0	0,00758082	0,00862877	0,00883431	0,00890792	0,00894259	0,0089617
11	0,01001001	0,01000984	0	0,00842313	0,00958635	0,00981481	0,00989676	0,0099354	0,00995673
12	0,01101101	0,01101079	0	0,00926545	0,01054369	0,01079509	0,0108854	0,01092805	0,01095162
13	0,01201201	0,01201172	0	0,01010776	0,01150079	0,01177515	0,01187386	0,01192053	0,01194637
14	0,01301301	0,01301265	0	0,01095007	0,01245766	0,012755	0,01286213	0,01291285	0,01294098
15	0,01401401	0,01401356	0	0,01179239	0,0134143	0,01373462	0,01385021	0,01390501	0,01393543
16	0,01501502	0,01501445	0	0,0126347	0,0143707	0,01471403	0,01483809	0,014897	0,01492975
17	0,01601602	0,01601533	0	0,01347701	0,01532686	0,01569322	0,01582579	0,01588883	0,01592391

Fig 9.2.2.10: ejemplo 1, aplicación de la interfaz Teorema de Weiestrass con distancia.

En la anterior imagen, figura 9.2.2.10, se muestra el fichero Excel "Weiestrass Aproximation" que se genera en el directorio. En éste se copian los resultados de las aproximaciones que se han hecho. Las dos primeras columnas corresponden a las abscisas x y a los valores exactos de la función f respectivamente, y de la tercera en adelante serán las aproximaciones. Por otro lado el número de filas dependerá de los puntos en los que se haya discretizado el intervalo.

-Ejemplo 2: sin distancia máxima

Fig 9.2.2.11: ejemplo 2, aplicación de la interfaz Teorema de Weiestrass sin distancia.

Al igual que antes, una vez introducidos todos los datos y presionando el pushbutton Aplicar se nos muestra una venta que nos indica que el programa está calculando y que esperemos. Una vez terminados los cálculos el programa nos muestra las siguientes ventanas

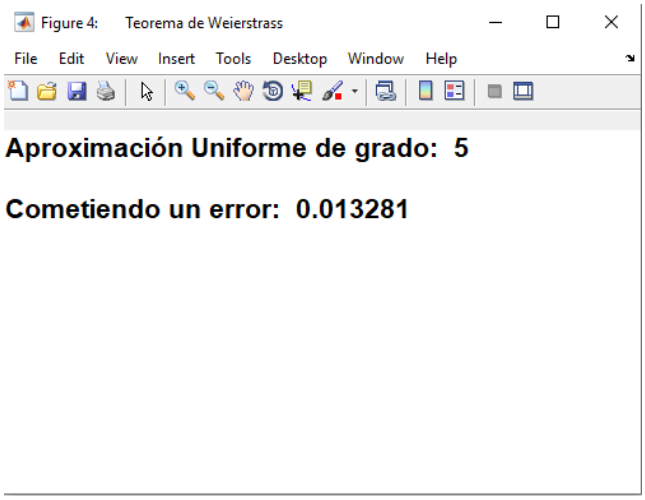


Fig 9.2.2.12: ejemplo 2, aplicación de la interfaz Teorema de Weiestrass sin distancia.

Como podemos ver en la figura anterior, nos dice que ha utilizado tal y como le hemos indicado un polinomio de grado cinco para realizar la aproximación, así como también nos indica la distancia de dicho polinomio respecto a la función original en norma infinito.

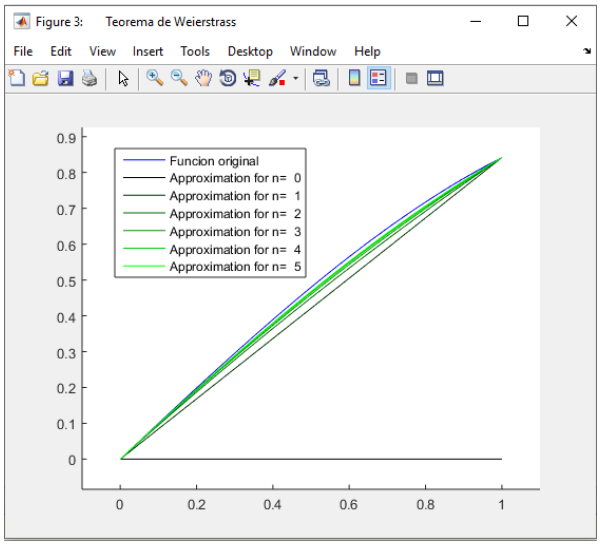


Fig 9.2.2.13: ejemplo 2, aplicación de la interfaz Teorema de Weiestrass sin distancia.

En l figura 9.2.2.13 tenemos la representación de la función original y de las distintas aproximaciones que se han realizado.

Por último al igual que antes se nos generará en el directorio el archivo Excel “Weiestrass Aproximation”. En este archivo se copian los resultados de las aproximaciones que se

han hecho. Las dos primeras columnas corresponden a las abscisas x y los valores exactos de la función f en dichas abscisas respectivamente. De la tercera columna en adelante serán las aproximaciones. Por otro lado el número de filas dependerá de los puntos en los que se ha discretizado el intervalo.

9.2.3 BÉZIER

Al presionar sobre el pushbutton Bézier se nos abrirá la siguiente interfaz:

Fig 9.2.3.1: interfaz Bézier.

A continuación describiremos los distintos elementos de esta interfaz figura 9.2.3.1:

- **Ayuda:** a pulsar aquí, podemos elegir entre:
  Tutorial de la interfaz gráfica: esta opción nos muestra un archivo pdf con un resumen de cómo usar la interfaz.
  Curvas de Bézier: esta otra opción nos muestra un archivo pdf con la teoría referente a Bézier.

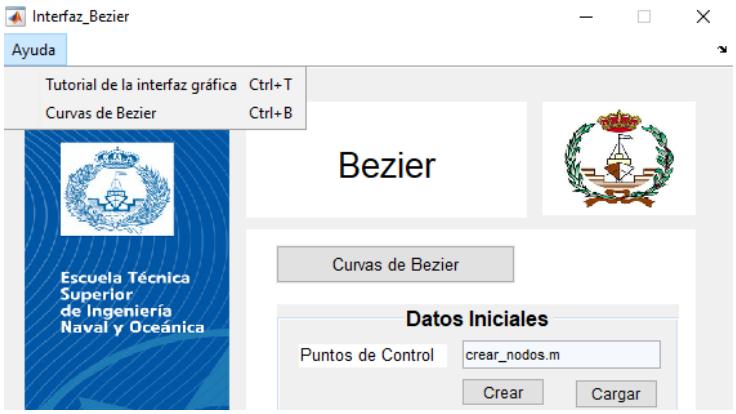


Fig 9.2.3.2: interfaz Bézier.

- **<u>Curvas de Bézier:</u>** nos muestra un archivo pdf con la teoría referente a Bézier.
- **<u>Datos iniciales:</u>**

  <u>Puntos de control:</u> aquí podemos tanto:

  Crear: creamos los nodos o puntos de control para nuestra curva en el siguiente programa que se nos abre. Es importante no confundir el uso del término nodos en Bézier con el vector de nodos que aparece en B-splines.

```
function control_points=crear_nodos()

% control_points is a nx2 matrix which contains the (x,y) coordinates of each
%                  control point by row
% Modify the following example with your own control points

control_points=[1,2; 2,5; 3,7];
```

Fig 9.2.3.3: programa crear_nodos.m interfaz de Bézier.

Cargar: nos abre una pantalla en la cual podemos seleccionar un archivo con unos puntos o nodos ya creados.

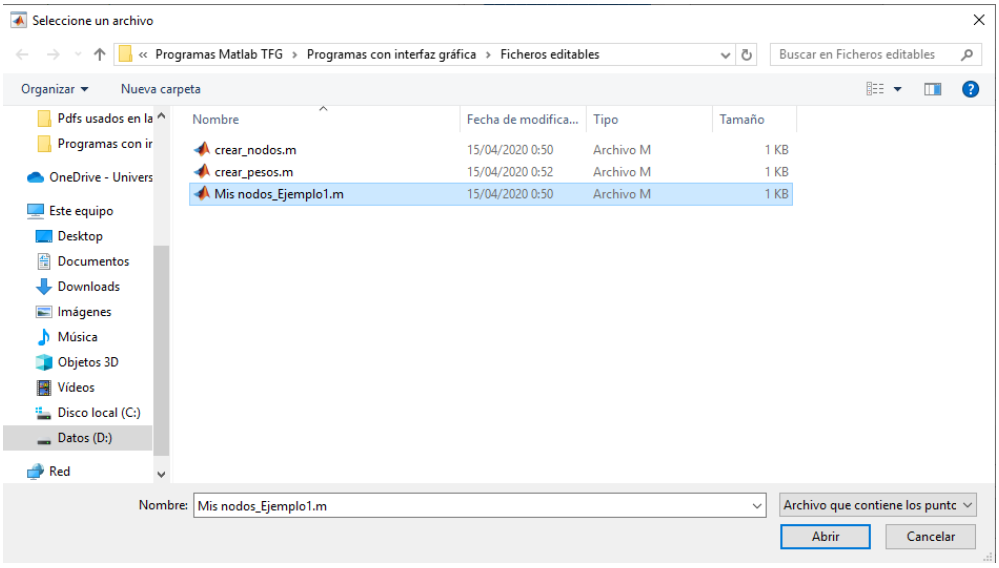


Fig 9.2.3.4: cargar nodos interfaz de Bézier.

- **Método:** aquí podemos elegir entre los siguientes métodos para la generación y representación de la curva:

  Construcción Curva de Bézier: nos construye la curva de Bézier paso a paso, según el paso de discretización indicado por el número de puntos que indiquemos.

  Paso de discretización: aquí indicaremos los valores del parámetro donde queremos que evalué la curva.

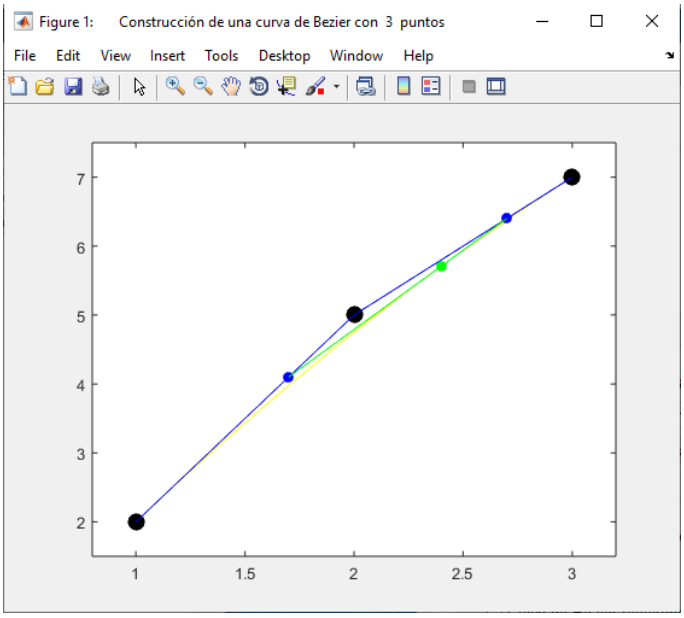


Fig 9.2.3.5: construcción curva de Bézier, interfaz de Bézier.

Curva de Bézier resultante: nos representa la curva de Bézier directamente en función del paso de discretización indicado.

Paso de discretización: aquí indicaremos los valores del parámetro donde queremos que evalué la curva.

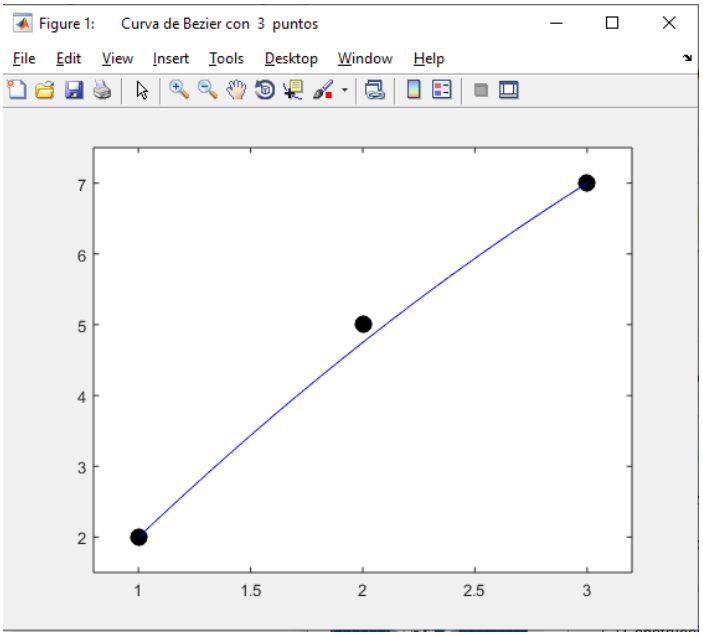


Fig 9.2.3.6: curva de Bézier, interfaz de Bézier.

Algoritmo de Casteljau: evalúa la curva de Bézier en el punto que nosotros indiquemos y nos da el algoritmo recursivo de evaluación rápida reduciendo el número de puntos en cada paso.

Punto donde evaluar: aquí indicamos el punto el que queremos evaluar.

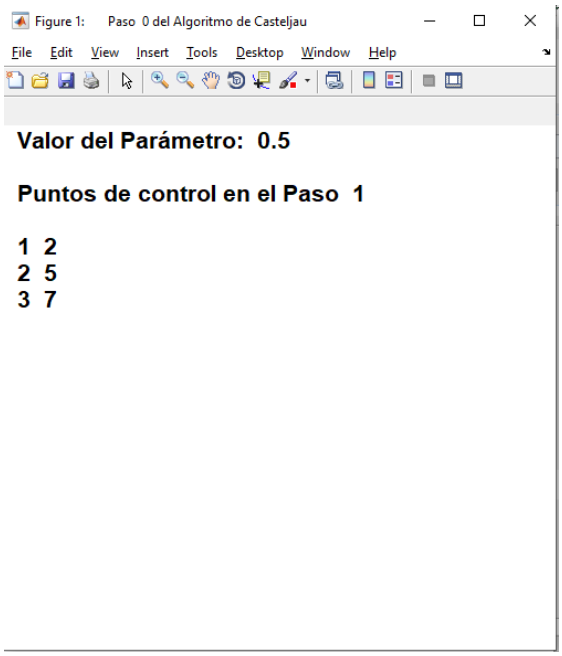


Fig 9.2.3.7: paso 0 algoritmo de Casteljau, interfaz de Bézier.

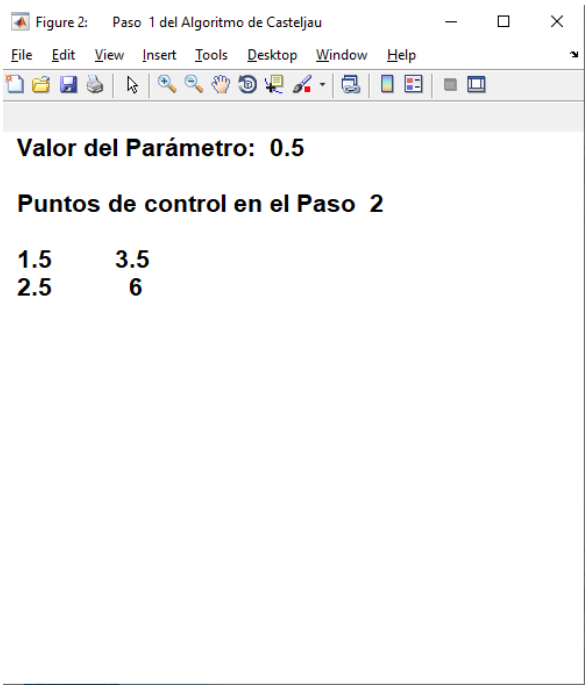


Fig 9.2.3.8: paso 1 algoritmo de Casteljau, interfaz de Bézier.

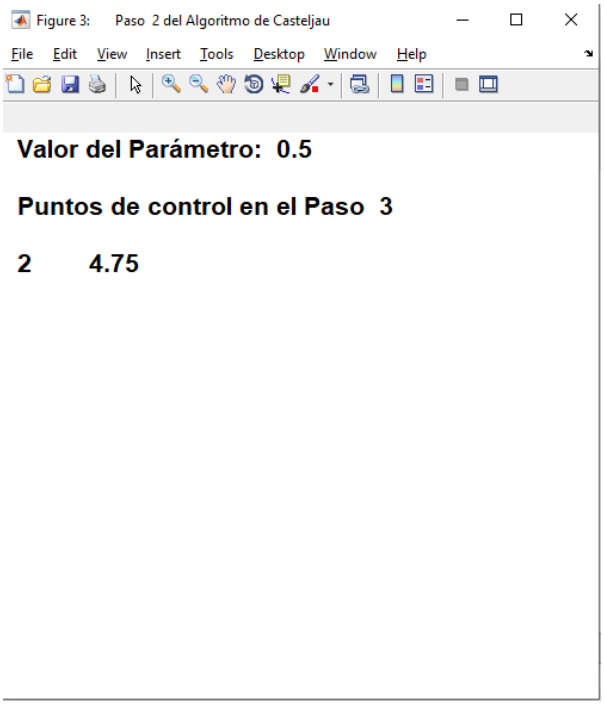


Fig 9.2.3.9: paso 2 algoritmo de Casteljau, interfaz de Bézier.

Construcción Curva de Bézier con Pesos: nos genera una curva de Bézier con los pesos que nosotros le hayamos indicado a cada punto.

Paso de discretización: aquí indicaremos la longitud de los subintervalos igualmente espaciados en que se divide el intervalo [0,1]. Éstos serán los valores que toma el parámetro donde queremos que evalué la curva.

Pesos: podemos tanto:

Crear: creamos los pesos asociados a cada punto de control.

```
function w=crear_pesos()

% w is the vector with the weight values
% Modify the following example with your own weights

w=[2,1,2];
```

Fig 9.2.3.10: programa crear_pesos.m interfaz de Bézier.

Cargar: nos abre una ventana en la cual podemos seleccionar un archivo ya creado con los pesos asociados a cada punto de control.

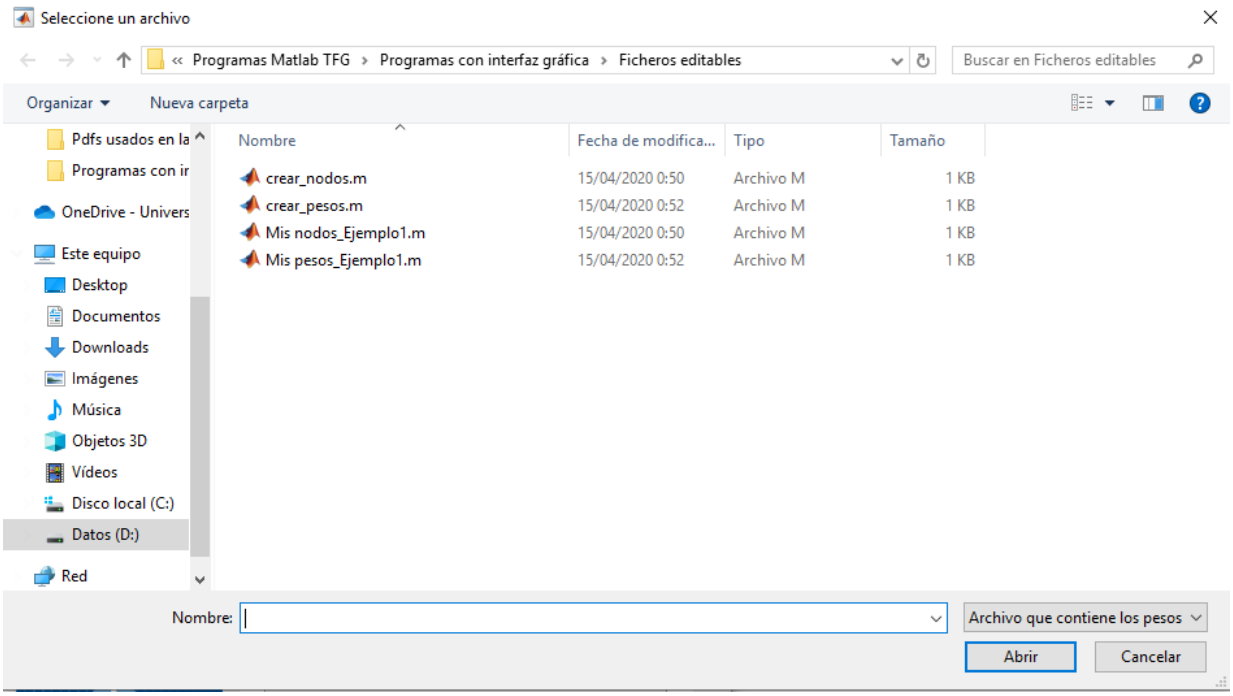


Fig 9.2.3.11: cargar pesos interfaz de Bézier.

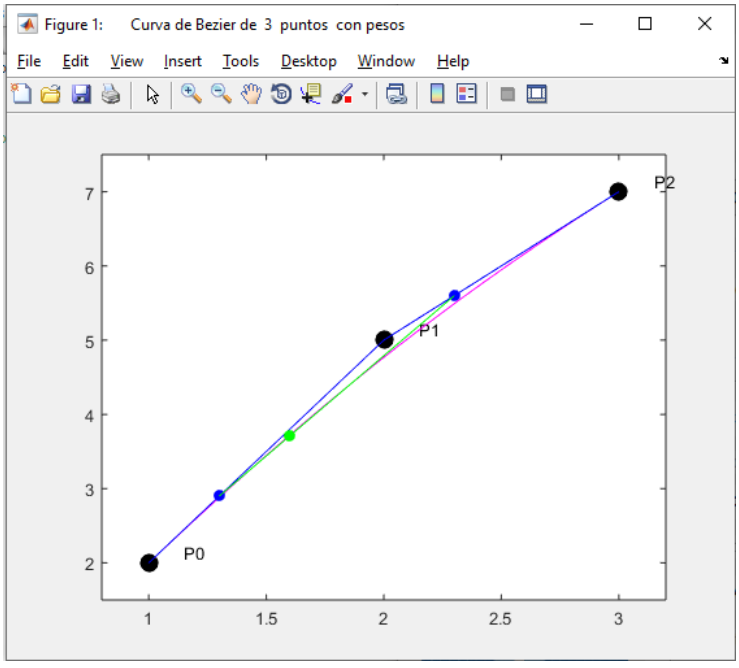


Fig 9.2.3.12: curva de Bézier con pesos, interfaz de Bézier.

- **Aplicar:** nos generará la curva según los parámetros y el método que nosotros hayamos indicado y en el caso del Algoritmo de Casteljau nos devolverá el algoritmo recursivo de evaluación rápida de la curva de Bézier a través de la disminución del número de puntos. También se nos creará un Excel con las coordenadas x e y de los puntos en cada paso de discretización seleccionado.

	A	B	C
1	Parámetro t	x puntos de control	y puntos de control
2	0	1	2
3	0,05	1,1	2,2975
4	0,1	1,2	2,59
5	0,15	1,3	2,8775
6	0,2	1,4	3,16
7	0,25	1,5	3,4375
8	0,3	1,6	3,71
9	0,35	1,7	3,9775
10	0,4	1,8	4,24
11	0,45	1,9	4,4975
12	0,5	2	4,75
13	0,55	2,1	4,9975
14	0,6	2,2	5,24
15	0,65	2,3	5,4775
16	0,7	2,4	5,71
17	0,75	2,5	5,9375
18	0,8	2,6	6,16
19	0,85	2,7	6,3775
20	0,9	2,8	6,59
21	0,95	2,9	6,7975
22	1	3	7

Fig 9.2.3.13: Excel creado, interfaz de Bézier.

- **Reset:** presionando este botón se nos vuelven a configurar los parámetros por defecto de la interfaz gráfica.
- **Salir:** cierra la interfaz.

9.2.4 B-SPLINES

Una vez presionado el pushbutton de B-splines sobre la interfaz principal, se nos abrirá una nueva interfaz gráfica:

Fig 9.2.4.1: interfaz B-spline.

A continuación describiremos los distintos elementos de la interfaz, figura 9.2.4.1:

- **Ayuda:** a pulsar aquí, podemos elegir entre:
  Tutorial de la interfaz gráfica: esta opción nos muestra un archivo pdf con un resumen de cómo usar la interfaz.
  B-spline: esta otra opción nos muestra un archivo pdf con la teoría referente a este capítulo.

Fig 9.2.4.2: interfaz B-spline.

- **<u>Elige un concepto:</u>** a pulsar aquí, podemos elegir entre:

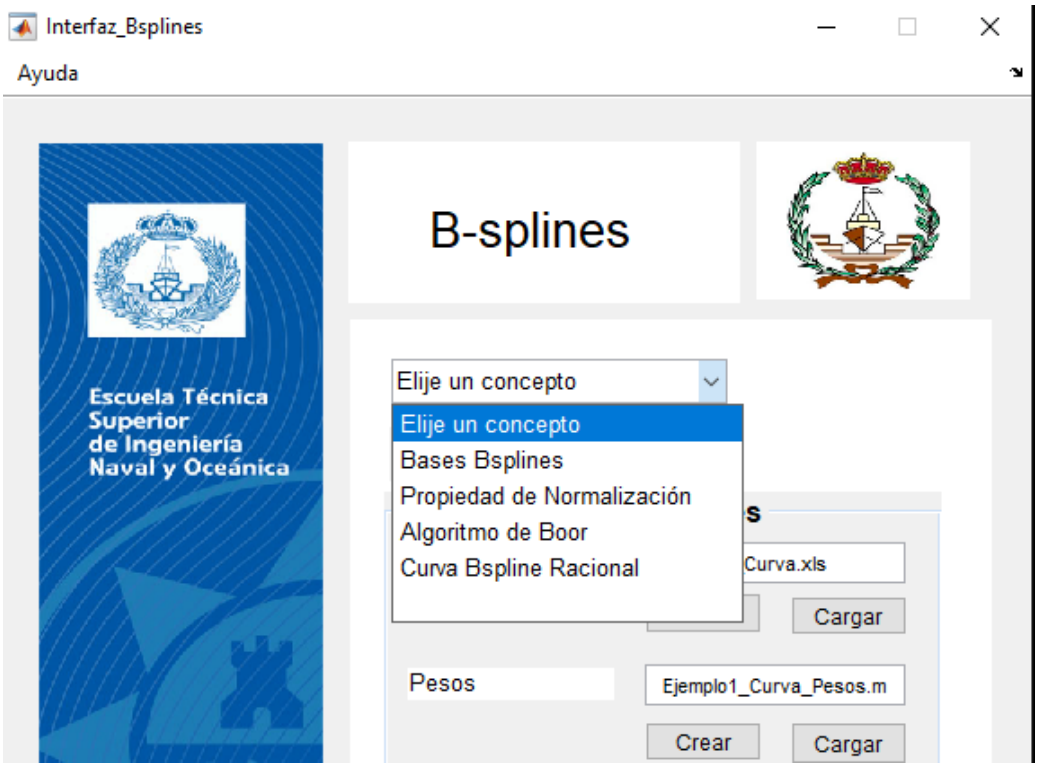


Fig 9.2.4.3: interfaz B-spline.

<u>Bases Bsplines:</u> esta opción nos permitirá representar las funciones base de un Bspline. Una vez seleccionada esta opción nos quedará la interfaz de la siguiente forma:

Fig 9.2.4.4: interfaz B-spline.

Como podemos apreciar en la figura 9.2.4.4, nos desaparecen en datos iniciales algunas opciones y nos aparecen otras, esto se debe a que lo que nos aparece son los datos necesarios para poder realizar este método.

A continuación comentaremos cada uno de ellos:

- Elementos base: aquí indicaremos en forma de vector los elementos de la base que queremos que nos dibuje. Hay m+1-k funciones base, donde m+1 es el número de entradas en el vector de nodos y k el orden del B-spline.

- Vector de nodos: esta opción nos permite tanto:
  -Crear: nos abre un programa Matlab en el que podremos crear nuestro vector de nodos.
  -Cargar: nos abre una ventana en la cual nosotros podremos seleccionar un archivo ya creado con un ejemplo de un vector de nodos.
- Orden B-spline: aquí introduciremos nuestro orden k del B-spline.
- Paso de discretización: aquí introduciremos el parámetro de discretización para la construcción de las funciones base.
- Tipo de Bspline: aquí indicaremos si nuestro vector de nodos es periódico o no periódico.
- Aplicar: pulsando sobre esta pushbutton se nos representarán las funciones base en función de los datos que nosotros le hayamos indicado.

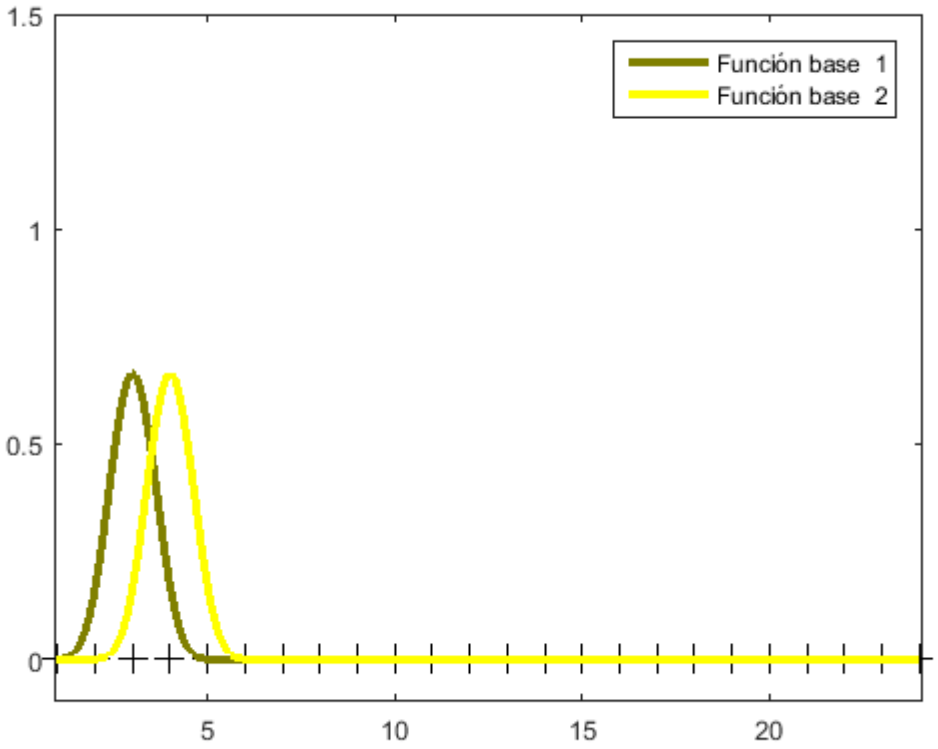


Fig 9.2.4.4: interfaz B-spline.

- Reset: nos dejara la interfaz con los valores por defecto que aparecen al seleccionar el método de Bases Bspline.
- Salir: se sale de la interfaz.

Propiedad de Normalización: esta opción nos permitirá comprobar la propiedad de normalización de las funciones base, la cual nos dice que las funciones base evaluadas en un punto suman 1.

Fig 9.2.4.5: interfaz B-spline.

Como podemos apreciar en la figura 9.2.4.5, nos vuelven a desaparecer en datos iniciales algunas opciones y nos aparecen otras, esto se debe también a que lo que nos aparece son los datos necesarios para poder realizar este método.

A continuación comentaremos cada uno de ellos:

- Vector de nodos: esta opción nos permite tanto:
  -Crear: nos abre un programa Matlab en el que podremos crear nuestro vector de nodos.

-Cargar: nos abre una ventana en la cual nosotros podremos seleccionar un archivo ya creado con un ejemplo de un vector de nodos.

- Orden B-spline: aquí introduciremos el orden k del B-spline.
- Paso de discretización: aquí introduciremos el parámetro de discretización para la construcción de las funciones base.
- Tipo de Bspline: aquí indicaremos si nuestro vector de nodos es periódico o no periódico.
- Evaluación en u: aquí indicaremos el punto en el que queremos evaluar y comprobar la propiedad de normalización de las funciones base. Este punto debe estar entre T(k) y T(m+1-k) donde T es el vector de nodos, k el orden del B-spline y m+1 representa el número de entradas del vector T.
- Aplicar: pulsando sobre este pushbutton se nos representarán las funciones base en función de los datos que nosotros le hayamos indicado y se evaluará la propiedad de normalización en el punto que nosotros hayamos introducido.

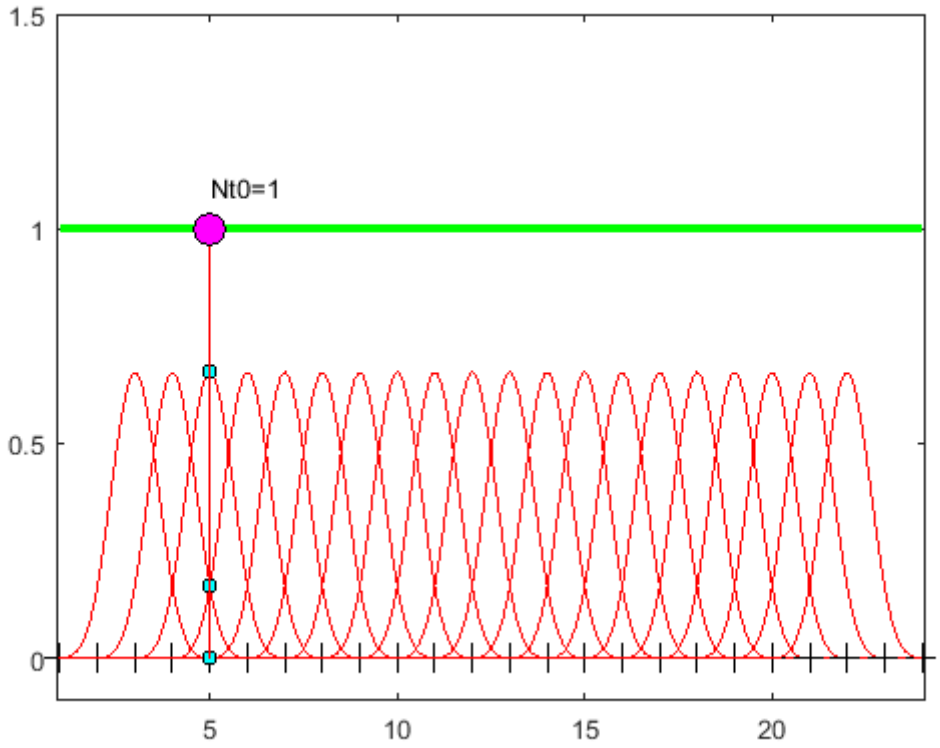


Fig 9.2.4.6: interfaz B-spline.

- Reset: nos dejará la interfaz con los valores por defecto que aparecen al seleccionar el método de Propiedad de Normalización.

- Salir: se sale de la interfaz.

Algoritmo de Boor: esta opción nos permitirá analizar cómo funciona el algoritmo recursivo de Boor para evaluar una curva B-spline en un valor del parámetro concreto.

Fig 9.2.4.7: interfaz B-spline.

Como podemos apreciar en la figura 9.2.4.7, nos desaparecen en datos iniciales algunas opciones y nos aparecen otras, esto se debe a que lo que nos aparece son los datos necesarios para poder realizar este método.

A continuación comentaremos cada uno de ellos:

- Puntos de control: aquí podemos tanto:
  -Crear: creamos los puntos de control para nuestra curva en un programa Matlab que se nos abre.

  -Cargar: nos abre una ventana en la cual nosotros podremos seleccionar un archivo ya creado con un ejemplo de un vector de puntos de control.

- Vector de nodos: esta opción nos permite tanto:
  -Crear: nos abre un programa Matlab en el que podremos crear nuestro vector de nodos.
  -Crear: nos abre una ventana en la cual nosotros podremos seleccionar un archivo ya creado con un ejemplo de un vector de nodos.

- Orden B-spline: aquí introduciremos el orden k del B-spline.

- Tipo de Bspline: aquí indicaremos si nuestro vector de nodos es periódico o no periódico.

- Evaluación en u: aquí indicaremos el punto en el que queremos evaluar la curva Bspline. Conviene recordar que el parámetro u debe pertenecer al rango de parámetros establecido según el vector de nodos y el orden del B-spline.

- Aplicar: pulsando sobre este pushbutton nos aparecerá unas ventanas con los puntos de control en cada paso del algoritmo hasta la obtención del punto de la curva, así como los pesos que se utilizan en cada paso para calcular el punto.

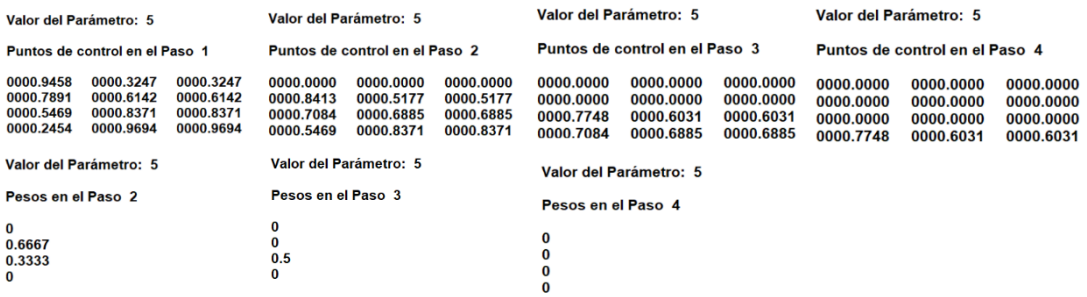


Fig 9.2.4.8: interfaz B-spline.

- Reset: nos dejará la interfaz con los valores por defecto que aparecen al seleccionar el método de Propiedad de Normalización.

- Salir: se sale de la interfaz.

Curva Bspline Racional: esta opción nos permitirá construir una curva B-spline racional. Al pulsar este pushbutton la interfaz se nos quedará de la siguiente forma:

Fig 9.2.4.9: interfaz B-spline.

Como podemos apreciar en la figura 9.2.4.9, debemos completar todos los campos que aparecen en la interfaz para poder realizar este método.

A continuación comentaremos cada uno de ellos:

- Puntos de control: aquí podemos tanto:

  -Crear: creamos los puntos de control para nuestra curva en un programa Matlab que se nos abre.

  -Cargar: nos abre una ventana en la cual nosotros podremos seleccionar un archivo ya creado con un ejemplo de un vector de puntos de control.

- Pesos: aquí podemos tanto:

  -Crear: creamos pesos asociados a cada uno de los puntos de control en un programa Matlab que se nos abre.

  -Cargar: nos abre una ventana en la cual nosotros podremos seleccionar un archivo ya creado con un ejemplo de un vector de pesos.

- Vector de nodos: esta opción nos permite tanto:
  -Crear: nos abre un programa Matlab en el que podremos crear nuestro vector de nodos.
  -Cargar: nos abre una ventana en la cual nosotros podremos seleccionar un archivo ya creado con un ejemplo de un vector de nodos.

- Orden B-spline: aquí introduciremos el orden k del B-spline.

- Tipo de Bspline: aquí indicaremos si nuestro vector de nodos es periódico o no periódico. Conviene aquí tener en cuenta que esta condición de frontera debe estar en concordancia con las primeras y últimas entradas del vector de nodos.

- Evaluación en u: aquí indicaremos el punto en el que queremos evaluar la curva Bspline. El parámetro u debe encontrarse dentro del rango de parámetros, el cual depende del número de entradas del vector de nodos y del orden del B-spline.

- Aplicar: pulsando sobre este pushbutton nos aparecerá la representación de la curva B-spline racional según los datos introducidos y el punto en que hemos querido evaluar, así como también se nos generará un archivo Excel que contendrá los puntos de la curva que ha generado.

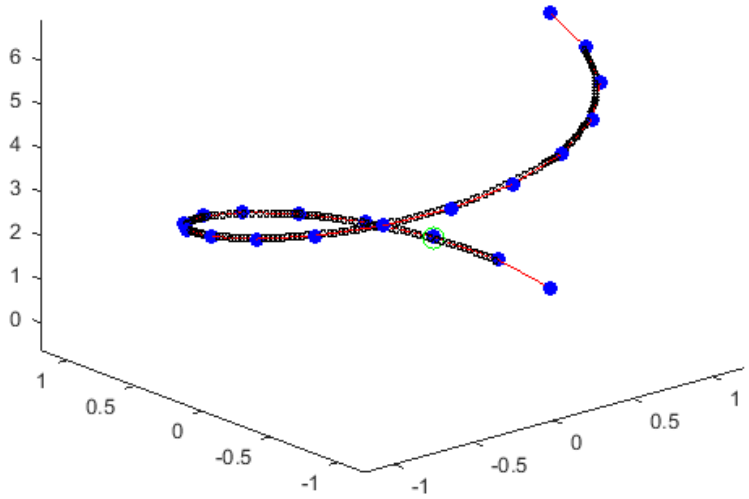
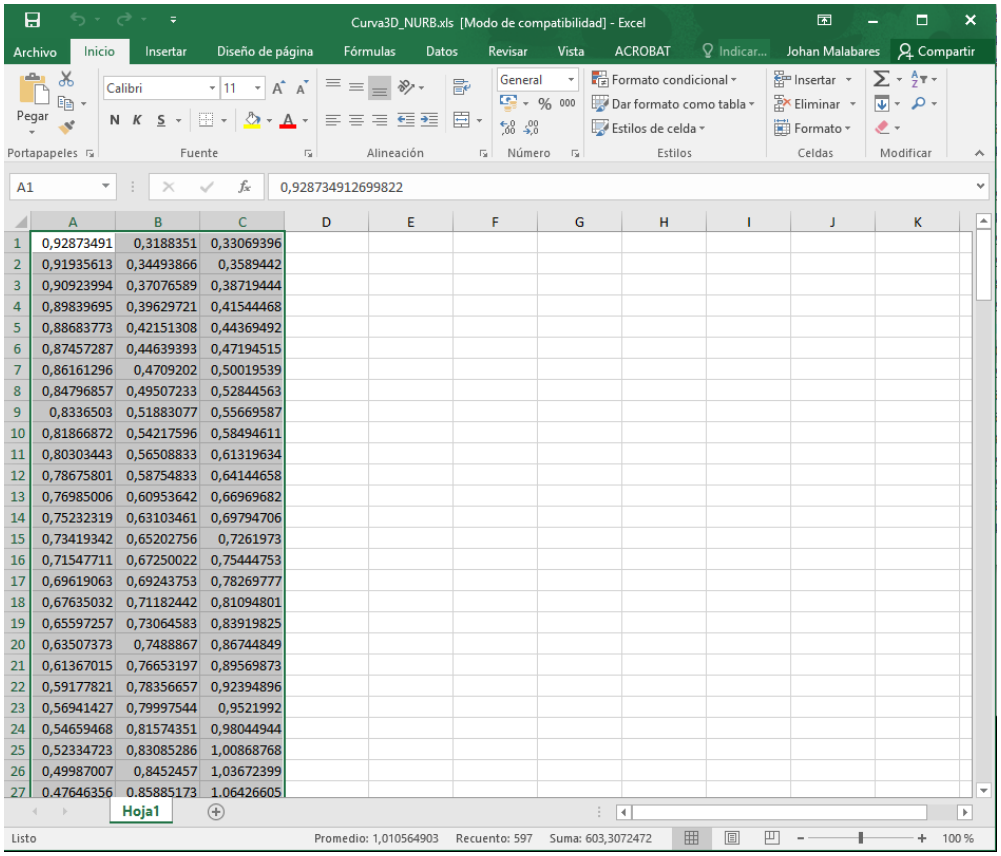


Fig 9.2.4.10: interfaz B-spline.



	A	B	C	D	E	F	G	H	I	J	K
1	0,92873491	0,3188351	0,33069396								
2	0,91935613	0,34493866	0,3589442								
3	0,90923994	0,37076589	0,38719444								
4	0,89839695	0,39629721	0,41544468								
5	0,88683773	0,42151308	0,44369492								
6	0,87457287	0,44639393	0,47194515								
7	0,86161296	0,4709202	0,50019539								
8	0,84796857	0,49507233	0,52844563								
9	0,8336503	0,51883077	0,55669587								
10	0,81866872	0,54217596	0,58494611								
11	0,80303443	0,56508833	0,61319634								
12	0,78675801	0,58754833	0,64144658								
13	0,76985006	0,60953642	0,66969682								
14	0,75232319	0,63103461	0,69794706								
15	0,73419342	0,65202756	0,7261973								
16	0,71547711	0,67250022	0,75444753								
17	0,69619063	0,69243753	0,78269777								
18	0,67635032	0,71182442	0,81094801								
19	0,65597257	0,73064583	0,83919825								
20	0,63507373	0,7488867	0,86744849								
21	0,61367015	0,76653197	0,89569873								
22	0,59177821	0,78356657	0,92394896								
23	0,56941427	0,79997544	0,9521992								
24	0,54659468	0,81574351	0,98044944								
25	0,52334723	0,83085286	1,00868768								
26	0,49987007	0,8452457	1,03672399								
27	0,47646356	0,85885173	1,06426605								

Fig 9.2.4.11: interfaz B-spline.

- Reset: nos dejará la interfaz con los valores por defecto que aparecen al seleccionar el método de Propiedad de Normalización.

- Salir: se sale de la interfaz.

9.2.5 SUPERFICIES B-SPLINES

Una vez presionado el pushbutton de Superficies B-spline sobre la interfaz principal, se nos abrirá una nueva interfaz gráfica:

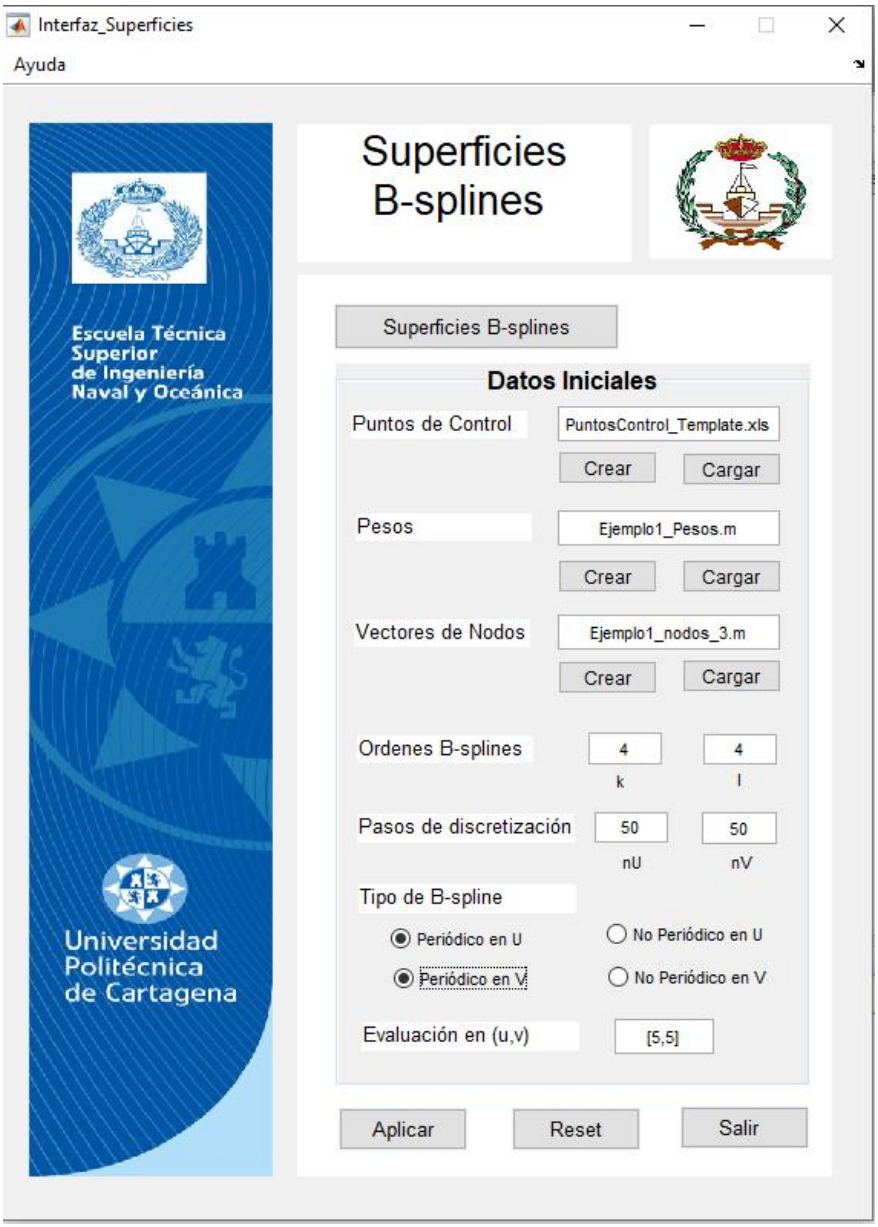


Fig 9.2.5.1: interfaz Superficies B-spline.

A continuación describiremos los distintos elementos de la interfaz, figura 9.2.5.1:

- **Ayuda:** al pulsar aquí, podemos elegir entre:
  Tutorial de la interfaz gráfica: esta opción nos muestra un archivo pdf con un resumen de cómo usar la interfaz.

Superficies B-spline: esta otra opción nos muestra un archivo pdf con la teoría referente a este capítulo.

- **Superficies B-spline:** nos muestra un archivo pdf con la teoría referente a las superficies B-spline.

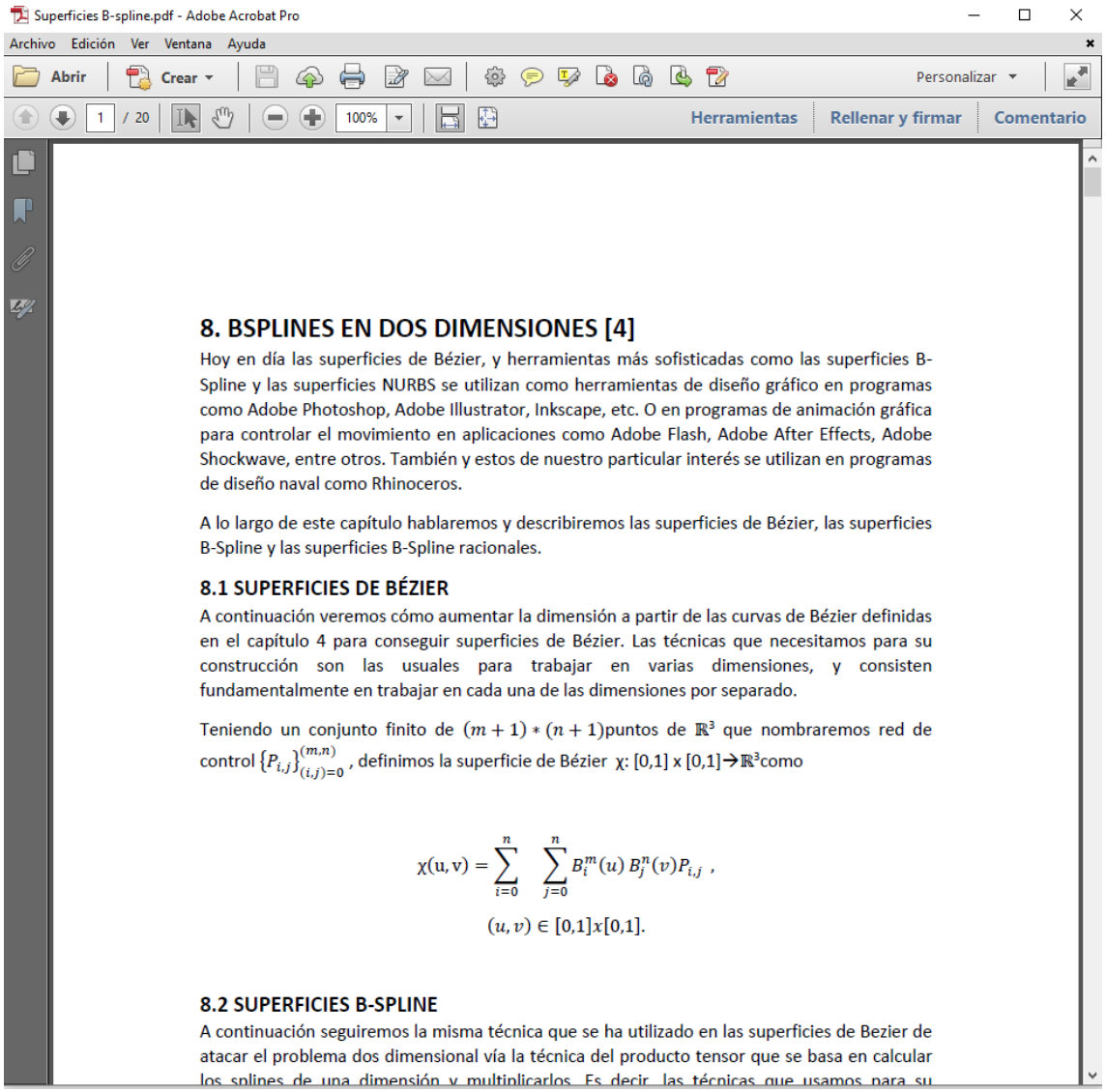
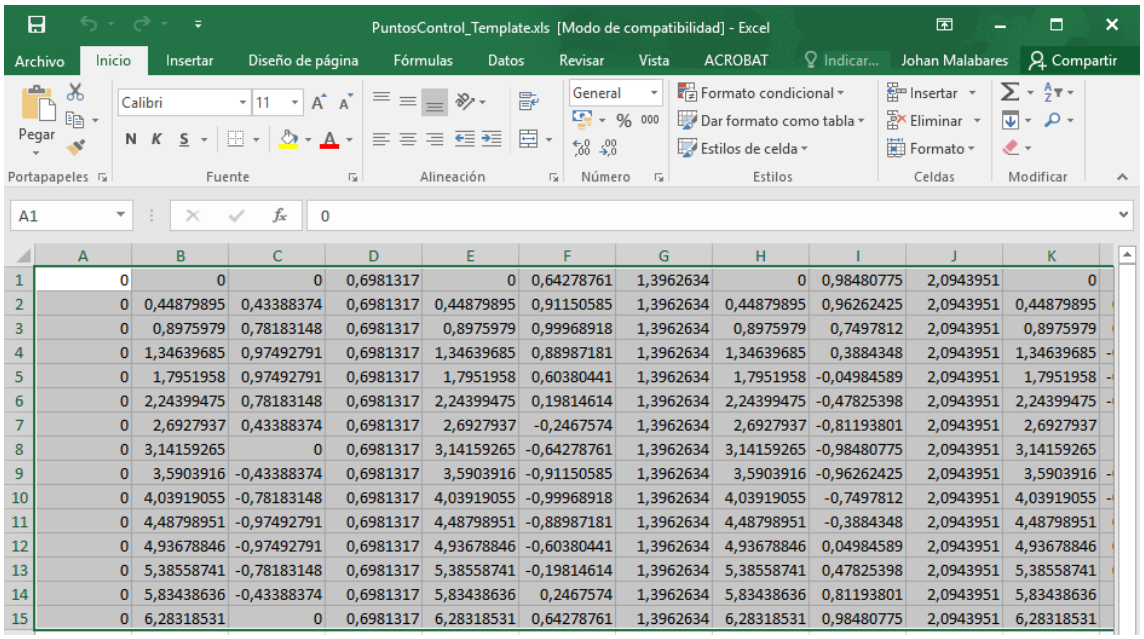


Fig 9.2.5.2: interfaz Superficies B-spline.

- **Datos iniciales:**

  Puntos de control: aquí podemos tanto:

  Crear: creamos los puntos de control para nuestra curva en el siguiente archivo Excel que se nos abre.



	A	B	C	D	E	F	G	H	I	J	K
1	0	0	0	0,6981317	0	0,64278761	1,3962634	0	0,98480775	2,0943951	0
2	0	0,44879895	0,43388374	0,6981317	0,44879895	0,91150585	1,3962634	0,44879895	0,96262425	2,0943951	0,44879895
3	0	0,8975979	0,78183148	0,6981317	0,8975979	0,99968918	1,3962634	0,8975979	0,7497812	2,0943951	0,8975979
4	0	1,34639685	0,97492791	0,6981317	1,34639685	0,88987181	1,3962634	1,34639685	0,3884348	2,0943951	1,34639685
5	0	1,7951958	0,97492791	0,6981317	1,7951958	0,60380441	1,3962634	1,7951958	-0,04984589	2,0943951	1,7951958
6	0	2,24399475	0,78183148	0,6981317	2,24399475	0,19814614	1,3962634	2,24399475	-0,47825398	2,0943951	2,24399475
7	0	2,6927937	0,43388374	0,6981317	2,6927937	-0,2467574	1,3962634	2,6927937	-0,81193801	2,0943951	2,6927937
8	0	3,14159265	0	0,6981317	3,14159265	-0,64278761	1,3962634	3,14159265	-0,98480775	2,0943951	3,14159265
9	0	3,5903916	-0,43388374	0,6981317	3,5903916	-0,91150585	1,3962634	3,5903916	-0,96262425	2,0943951	3,5903916
10	0	4,03919055	-0,78183148	0,6981317	4,03919055	-0,99968918	1,3962634	4,03919055	-0,7497812	2,0943951	4,03919055
11	0	4,48798951	-0,97492791	0,6981317	4,48798951	-0,88987181	1,3962634	4,48798951	-0,3884348	2,0943951	4,48798951
12	0	4,93678846	-0,97492791	0,6981317	4,93678846	-0,60380441	1,3962634	4,93678846	0,04984589	2,0943951	4,93678846
13	0	5,38558741	-0,78183148	0,6981317	5,38558741	-0,19814614	1,3962634	5,38558741	0,47825398	2,0943951	5,38558741
14	0	5,83438636	-0,43388374	0,6981317	5,83438636	0,2467574	1,3962634	5,83438636	0,81193801	2,0943951	5,83438636
15	0	6,28318531	0	0,6981317	6,28318531	0,64278761	1,3962634	6,28318531	0,98480775	2,0943951	6,28318531

Fig 9.2.5.3: interfaz Superficies B-spline

Cargar: nos abre una pantalla en la cual podemos seleccionar un archivo con unos puntos de control ya creados.

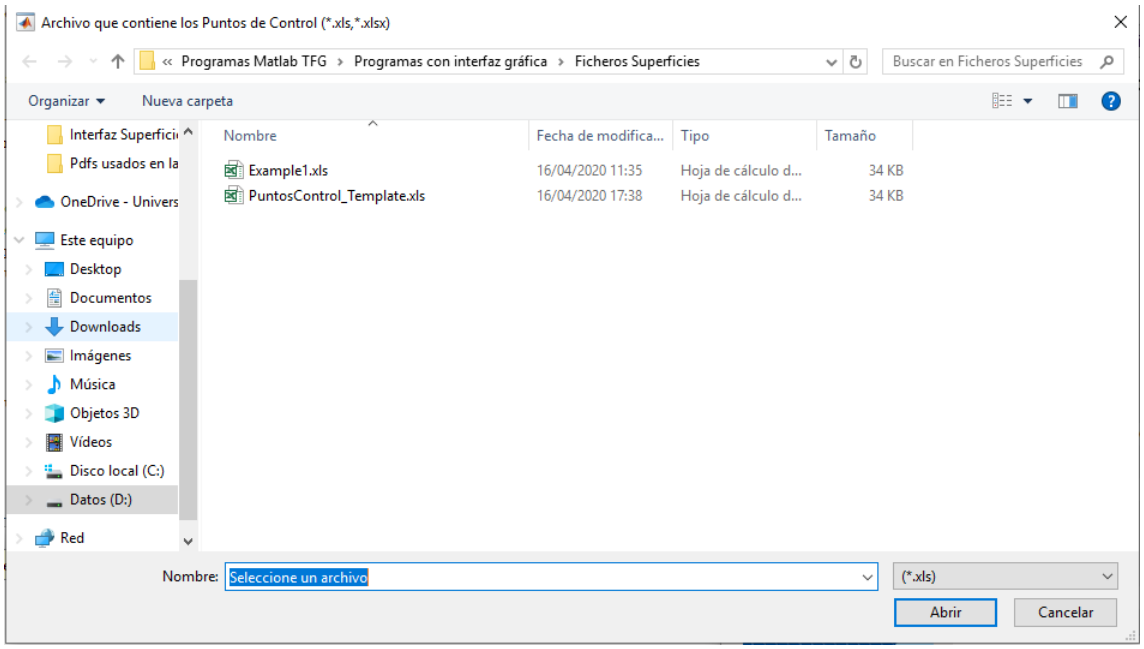


Fig 9.2.5.4: interfaz Superficies B-spline.

Pesos: podemos tanto:

Crear: creamos los pesos asociados a cada punto de control.

Se hace modificando el archivo de la figura 9.2.5.5.

```
function w=crear_pesos_3()

% Weights for surfaces in R^3
% w is the vector with the weight values
% Modify the following example with your own weights

n=15; m=10;
size_ControlPoints=[n,m];

W=ones(size_ControlPoints); W(1,1:m)=100; W(n,1:m)=100; W(1:n,1)=100; W(1:n,m)=100;
```

Fig 9.2.5.5: interfaz Superficies B-spline.

Cargar: nos abre una ventana en la cual podemos seleccionar un archivo ya creado con los pesos asociados a cada punto de control.

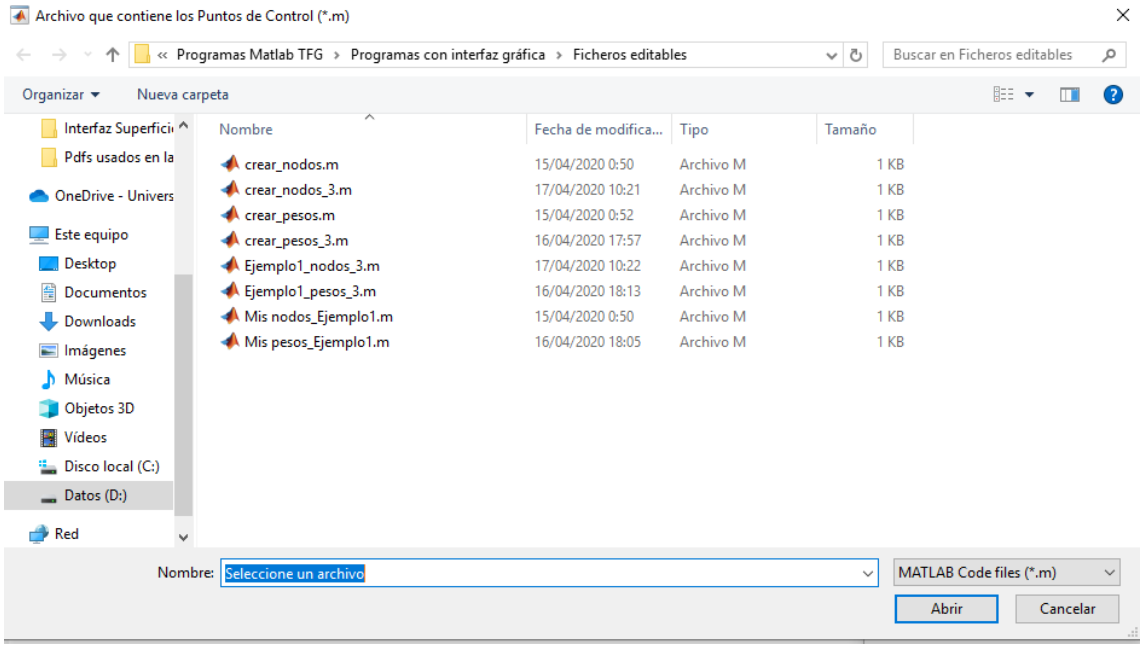


Fig 9.2.5.6: interfaz Superficies B-spline.

<u>Vector de nodos:</u> podemos tanto:

Crear: creamos el vector de nodos tanto en la dirección U como en la dirección V.

```
function [U,V]=crear_nodos_3()

% U is a nxl matrix which represents the knot-vector in one direction
% V is a nxl matrix which represents the knot-vector in the perpendicular direction
%    to U
% Modify the following example with your own knot vectors
% Notice that the length p of U must be equal to n+k where n means the
%        number of rows in the Control Points matrix and k the order of
%        the B-spline in that direction
% Notice that the length q of V must be equal to m+l where m means the
%        number of columns in the Control Points matrix and l the order of
%        the B-spline in that direction

p=19; q=14;

U=1:p;U=U';
V=1:q; V=V';
```

Fig 9.2.5.7: interfaz Superficies B-spline.

Cargar: nos abre una ventana en la cual podemos seleccionar un archivo con unos vectores de nodos ya creados.

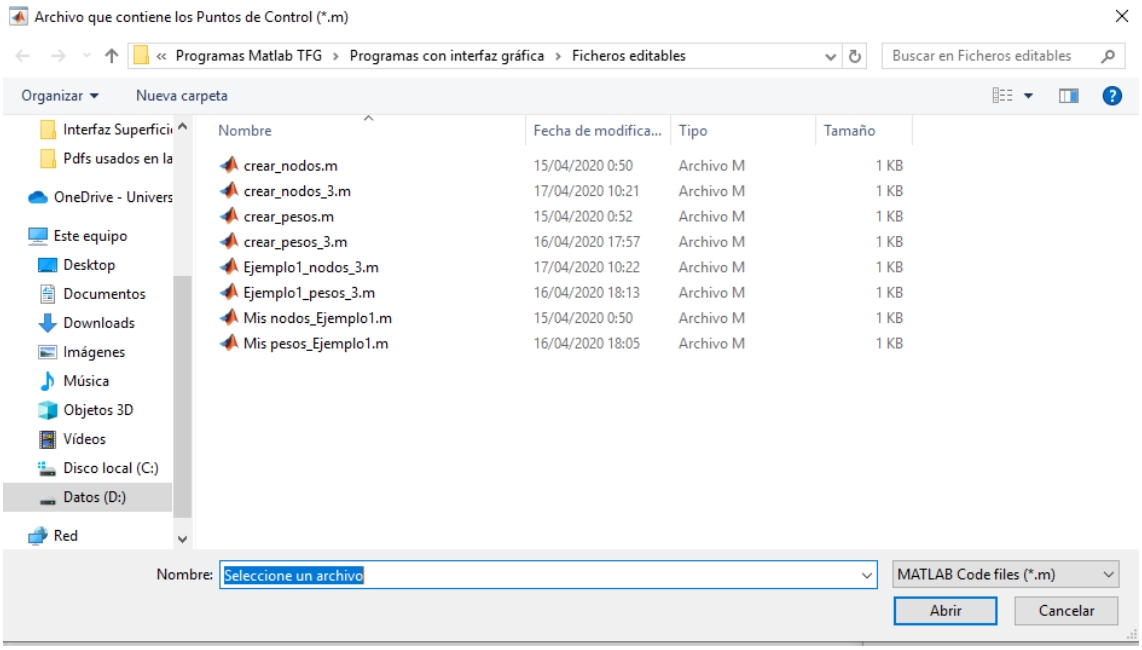


Fig 9.2.5.8: interfaz Superficies B-spline.

Ordenes B-spline: aquí introduciremos tanto el orden del B-spline en una dirección, k, como el orden del B-spline en la otra dirección, l.

Pasos de discretización: aquí introduciremos tanto el número de puntos para el cálculo del paso de discretización para el vector de nodos U, en nU, como también el número de puntos para el cálculo del paso de discretización para el vector de nodos V en nV.

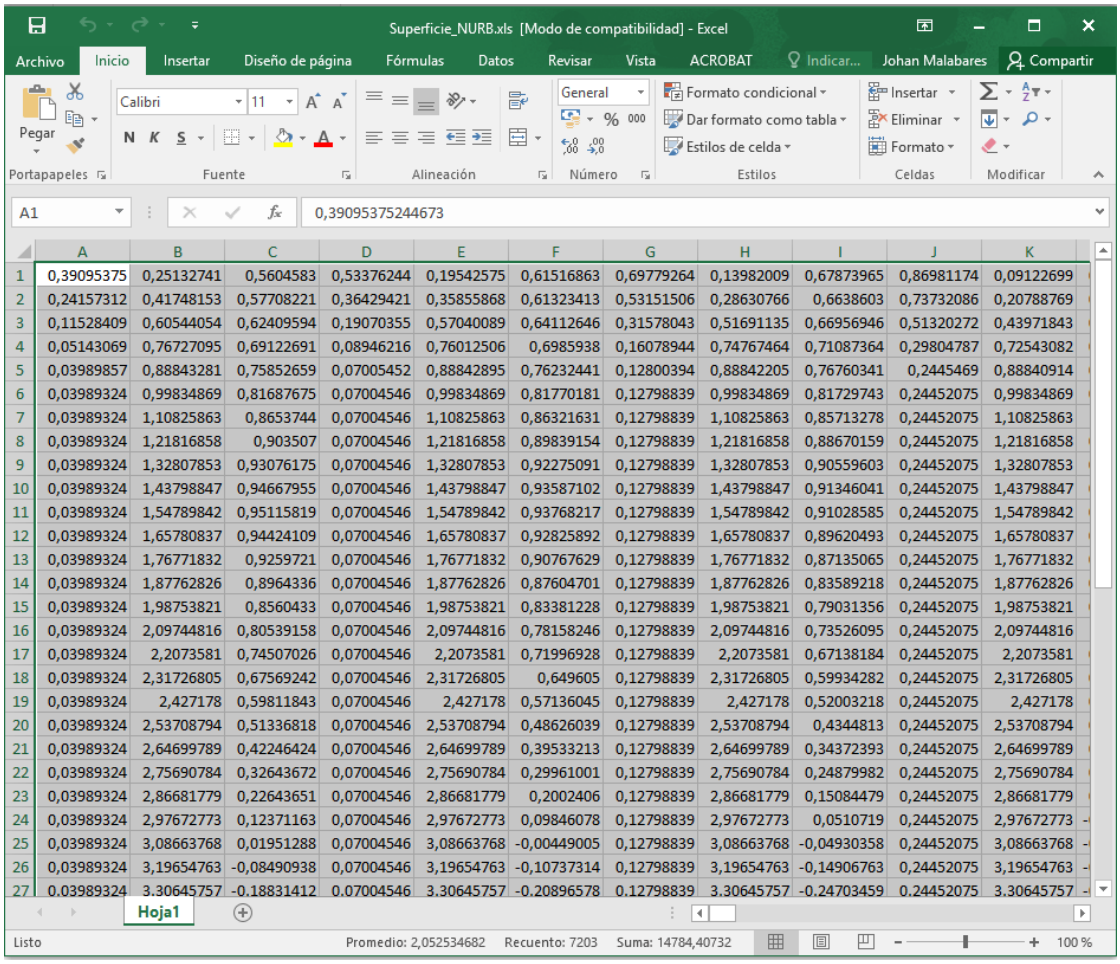
Tipo de B-spline: aquí indicaremos si nuestro vector de nodos es periódico o no periódico tanto en la dirección U como en la dirección V. Siempre tendremos una opción marcada para ambas direcciones. Estas condiciones implican que los

vectores de nodos que se han introducido concuerden en su principio y su final con el tipo de condiciones en la frontera que se imponen, es decir, si seleccionamos no periódico en una dirección lo que hacemos es poner condiciones de frontera hacia el interior y los primeros nodos tendrían que ser iguales en ese vector y si escogemos periódicos estos no pueden ser iguales, es decir, no repetiremos los nodos.

Evaluación en (u,v): aquí indicaremos el punto donde queremos evaluar en la superficie. Hay que tener en cuenta que el punto debe estar dentro del rango factible, es decir, tiene que estar entre los valores que están entre la posición k del vector de nodos y el valor que está en la posición m-k+1 del vector de nodos, que coincide con n+1 donde n es el número de puntos de control.

- **Aplicar:** al pulsar sobre este pushbutton nos aparece una ventana indicándonos que el programa se está ejecutando, y se nos genera la superficie según los parámetros que nosotros le hallamos indicado, así como también nos creará un archivo Excel con los puntos de la superficie en función de la discretización que hayamos indicado.

Fig 9.2.5.9: interfaz Superficies B-spline.



	A	B	C	D	E	F	G	H	I	J	K
1	0,39095375	0,25132741	0,5604583	0,53376244	0,19542575	0,61516863	0,69779264	0,13982009	0,67873965	0,86981174	0,09122699
2	0,24157312	0,41748153	0,57708221	0,36429421	0,35855868	0,61323413	0,53151506	0,28630766	0,6638603	0,73732086	0,20788769
3	0,11528409	0,60544054	0,62409594	0,19070355	0,57040089	0,64112646	0,31578043	0,51691135	0,66956946	0,51320272	0,43971843
4	0,05143069	0,76727095	0,69122691	0,08946216	0,76012506	0,6985938	0,16078944	0,74767464	0,71087364	0,29804787	0,72543082
5	0,03989857	0,88843281	0,75852659	0,07005452	0,88842895	0,76232441	0,12800394	0,88842205	0,76760341	0,2445469	0,88840914
6	0,03989324	0,99834869	0,81687675	0,07004546	0,99834869	0,81770181	0,12798839	0,99834869	0,81729743	0,24452075	0,99834869
7	0,03989324	1,10825863	0,8653744	0,07004546	1,10825863	0,86321631	0,12798839	1,10825863	0,85713278	0,24452075	1,10825863
8	0,03989324	1,21816858	0,903507	0,07004546	1,21816858	0,89839154	0,12798839	1,21816858	0,88670159	0,24452075	1,21816858
9	0,03989324	1,32807853	0,93076175	0,07004546	1,32807853	0,92275091	0,12798839	1,32807853	0,90559603	0,24452075	1,32807853
10	0,03989324	1,43798847	0,94667955	0,07004546	1,43798847	0,93587102	0,12798839	1,43798847	0,91346041	0,24452075	1,43798847
11	0,03989324	1,54789842	0,95115819	0,07004546	1,54789842	0,93768217	0,12798839	1,54789842	0,91028585	0,24452075	1,54789842
12	0,03989324	1,65780837	0,94424109	0,07004546	1,65780837	0,92825892	0,12798839	1,65780837	0,89620493	0,24452075	1,65780837
13	0,03989324	1,76771832	0,9259721	0,07004546	1,76771832	0,90767629	0,12798839	1,76771832	0,87135065	0,24452075	1,76771832
14	0,03989324	1,87762826	0,8964336	0,07004546	1,87762826	0,87604701	0,12798839	1,87762826	0,83589218	0,24452075	1,87762826
15	0,03989324	1,98753821	0,8560433	0,07004546	1,98753821	0,83381228	0,12798839	1,98753821	0,79031356	0,24452075	1,98753821
16	0,03989324	2,09744816	0,80539158	0,07004546	2,09744816	0,78158246	0,12798839	2,09744816	0,73526095	0,24452075	2,09744816
17	0,03989324	2,2073581	0,74507026	0,07004546	2,2073581	0,71996928	0,12798839	2,2073581	0,67138184	0,24452075	2,2073581
18	0,03989324	2,31726805	0,67569242	0,07004546	2,31726805	0,649605	0,12798839	2,31726805	0,59934282	0,24452075	2,31726805
19	0,03989324	2,427178	0,59811843	0,07004546	2,427178	0,57136045	0,12798839	2,427178	0,52003218	0,24452075	2,427178
20	0,03989324	2,53708794	0,51336818	0,07004546	2,53708794	0,48626039	0,12798839	2,53708794	0,4344813	0,24452075	2,53708794
21	0,03989324	2,64699789	0,42246424	0,07004546	2,64699789	0,39533213	0,12798839	2,64699789	0,34372393	0,24452075	2,64699789
22	0,03989324	2,75690784	0,32643672	0,07004546	2,75690784	0,29961001	0,12798839	2,75690784	0,24879982	0,24452075	2,75690784
23	0,03989324	2,86681779	0,22643651	0,07004546	2,86681779	0,2002406	0,12798839	2,86681779	0,15084479	0,24452075	2,86681779
24	0,03989324	2,97672773	0,12371163	0,07004546	2,97672773	0,09846078	0,12798839	2,97672773	0,0510719	0,24452075	2,97672773
25	0,03989324	3,08663768	0,01951288	0,07004546	3,08663768	-0,00449005	0,12798839	3,08663768	-0,04930358	0,24452075	3,08663768
26	0,03989324	3,19654763	-0,08490938	0,07004546	3,19654763	-0,10737314	0,12798839	3,19654763	-0,14906763	0,24452075	3,19654763
27	0,03989324	3,30645757	-0,18831412	0,07004546	3,30645757	-0,20896578	0,12798839	3,30645757	-0,24703459	0,24452075	3,30645757

Fig 9.2.5.10: interfaz Superficies B-spline.

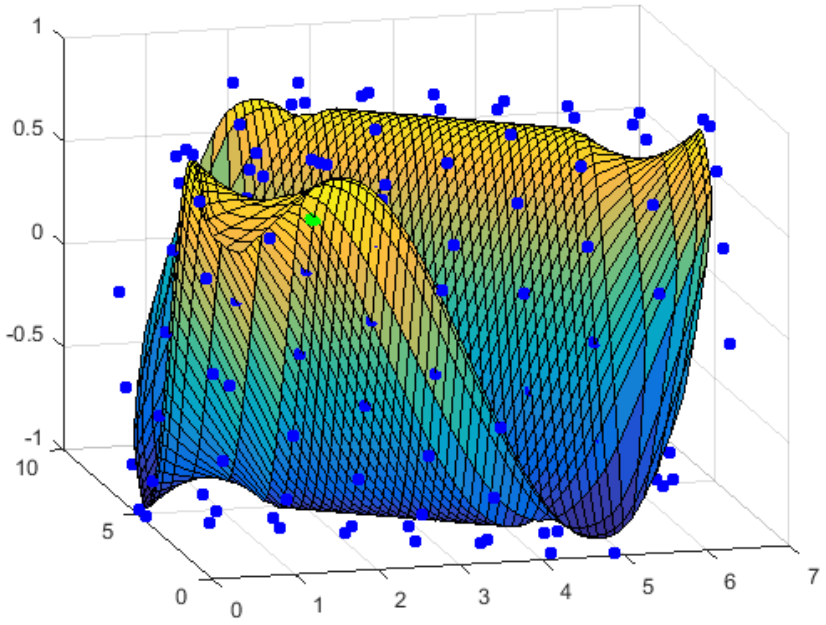


Fig 9.2.5.11: interfaz Superficies B-spline.

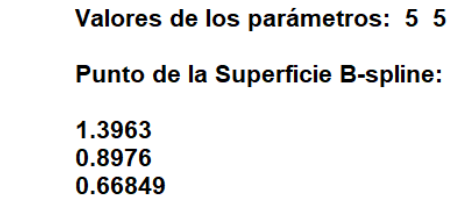


Fig 9.2.5.11: interfaz Superficies B-spline

- **Reset:** presionando este botón se nos vuelven a configurar los parámetros por defecto de la interfaz gráfica.
- **Salir:** cierra la interfaz.

9.2.6 TFG

Este pushbotton nos llevará directamente al trabajo de fin de grado

Fig 9.2.6.1: trabajo fin de grado.

9.2.7 ACERCA DE

Este nos abrirá una ventana en la cual el usuario podrá ver el siguiente resumen sobre la interfaz:

ACERCA DE

Esta interfaz gráfica permite al usuario poder obtener de una manera rápida y sencilla tanto curvas como superficies, las cuales se generan a partir de unos puntos de control que pueden ser introducidos por el usuario o pueden ser importados al programa mediante un archivo Excel externo.
También podremos modificar dichas curvas y superficies pudiendo así valorar y elegir la que mejor se ajuste a nuestras necesidades o simplemente para realizar un estudio o llevar a cabo la reconstrucción de alguna parte de un objeto o de alguna pieza. En este proyecto nos interesan las curvas y superficies del mundo naval. En particular las piezas de un buque, tipo casco, bulbo, palas de hélice, etcétera.

Dichas curvas y superficies podrán ser obtenidas y visualizadas utilizando algoritmos relacionados con Bernstein, Bézier, B-spline y Nurbs. Hemos incluido las herramientas en el orden cronológico en que se fueron desarrollando, apareciendo así primero los polinomios de Bernstein, después las curvas de Bezier, y posteriormente los B-splines. Dentro de B-splines se incluyen los B-splines racionales y los NURBS.

El motivo de crear esta interfaz ha sido facilitar el uso de los distintos programas Matlab creados en el Trabajo final de grado “Operadores de reconstrucción y de subdivisión para la generación de métodos numéricos. Aplicación al mundo Naval”.

Este trabajo ha sido elaborado y desarrollado por el alumno José Alfredo Costa Martínez bajo la dirección y supervisión del profesor Juan Carlos Trillo Moya en el departamento de Matemática Aplicada y Estadística de la ETSINO en la UPCT.

9.2.8 SALIR

Por último con este pushbutton se cerrará la interfaz gráfica.

10. CASOS PRÁCTICOS

En este capítulo llevaremos a la práctica algunos ejemplos mediante Splines cúbicos interpolantes y NURBS.

10.1 CASO PRÁCTICO 1

En este caso práctico trataremos el recorrido que debe hacer un contenedor desde que la grúa lo carga hasta su descarga en el muelle sobre el remolque de un camión y veremos cómo gracias a los Splines cúbicos interpolantes obtendremos una representación suave de dicho movimiento.

En la figura 10.1.1 se muestra la grúa que realizará la operación y el barco del que se descargará el contenedor.

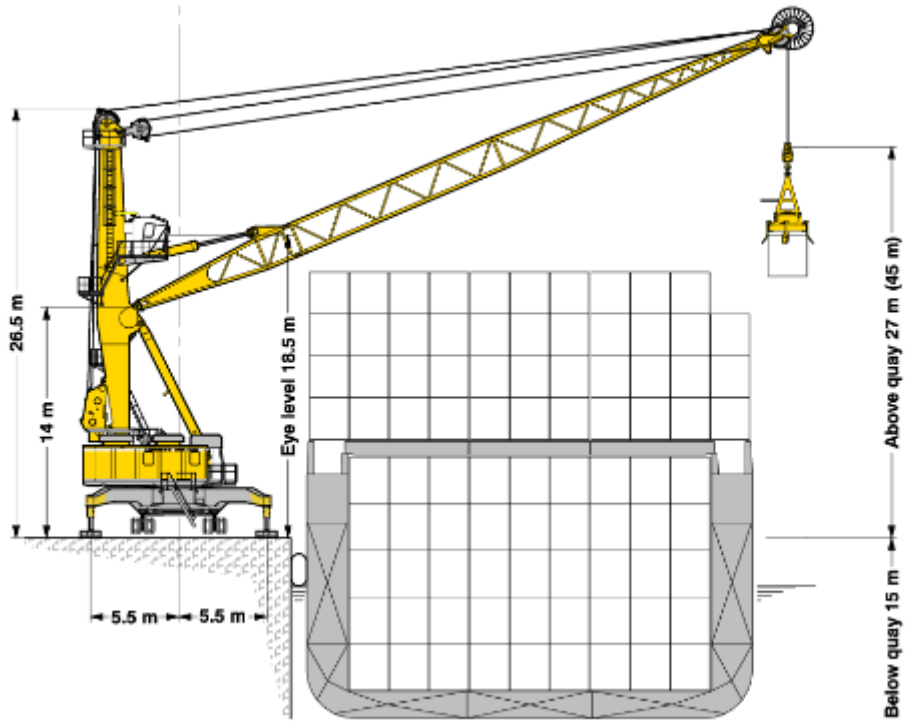


Fig. 10.1.1: representación de la descarga de un contenedor.

En la siguiente tabla se muestran las coordenadas del punto en el que se encuentra el contenedor, así como los puntos intermedios hasta su descarga sobre el camión situado en el muelle:

Puntos de paso	Coord. x (m)	Coord. y (m)	Coord. z (m)
1	38	0	12
2	38	0	27
3	26.87	26.87	27
4	-5.5	43.5	1.5

Tabla 10.1.1: coordenas por las que pasará el contenedor.

- punto 1: punto en el que se encuentra el contenedor en el buque.

- punto 2: punto hasta el cual se izará la carga una vez enganchado el contenedor.

-punto 3: punto intermedio de paso del contenedor.

-punto 4: punto de descarga del contenedor sobre el remolque del camión.

En la siguiente figura tenemos el ajuste de la variable x, donde los puntos en rojo serían los valores interpolados entre cada uno de los puntos.

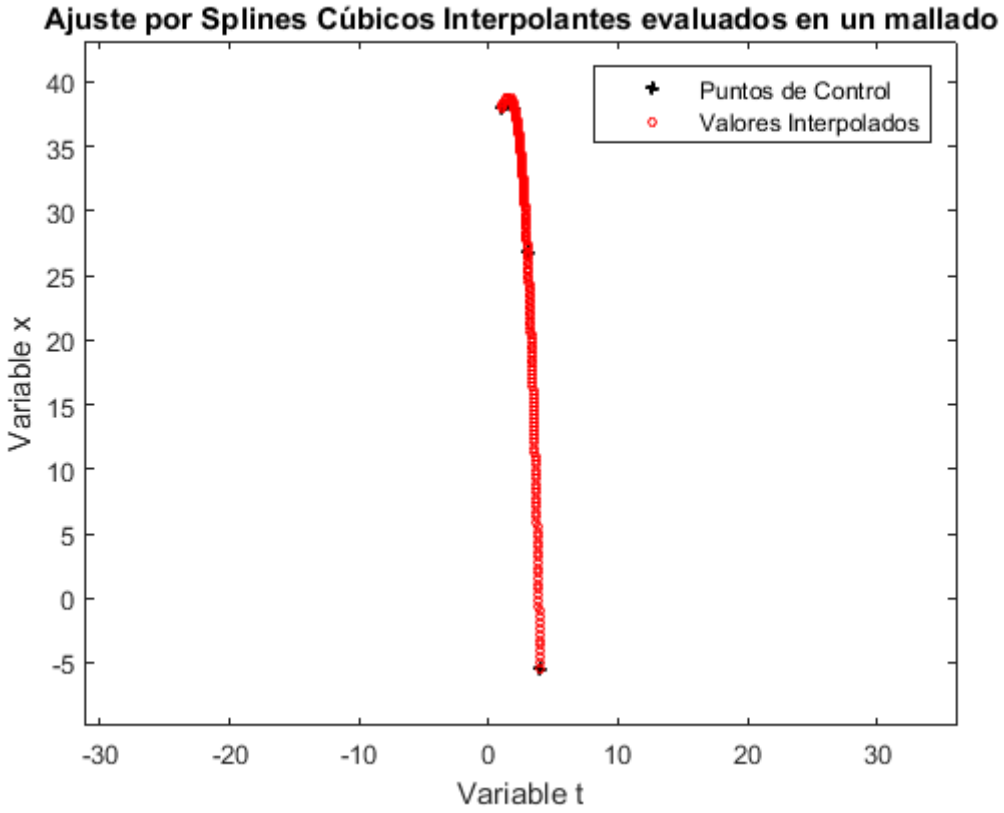


Fig. 10.1.2: ajuste para las coordenadas x.

En la figura 10.1.3 se muestra una segunda gráfica, donde se han dibujado los splines sin interpolar, es decir la expresión simbólica, en la cual vemos que hay tres trozos puesto que tenemos cuatro puntos.

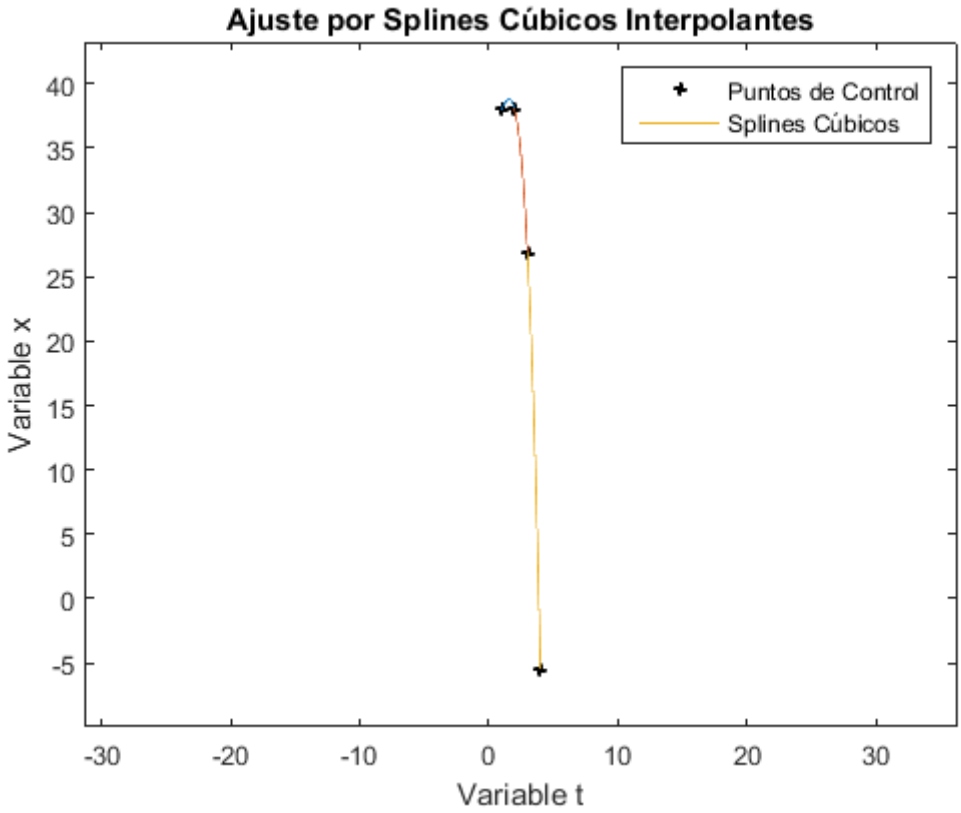


Fig. 10.1.3: ajuste por Splines Cúbicos Interpolantes para la variable x.

En la tercera gráfica, figura 10.1.4 tenemos la primera derivada de la función spline construída, en la que vemos que efectivamente la derivada de la función es suave.

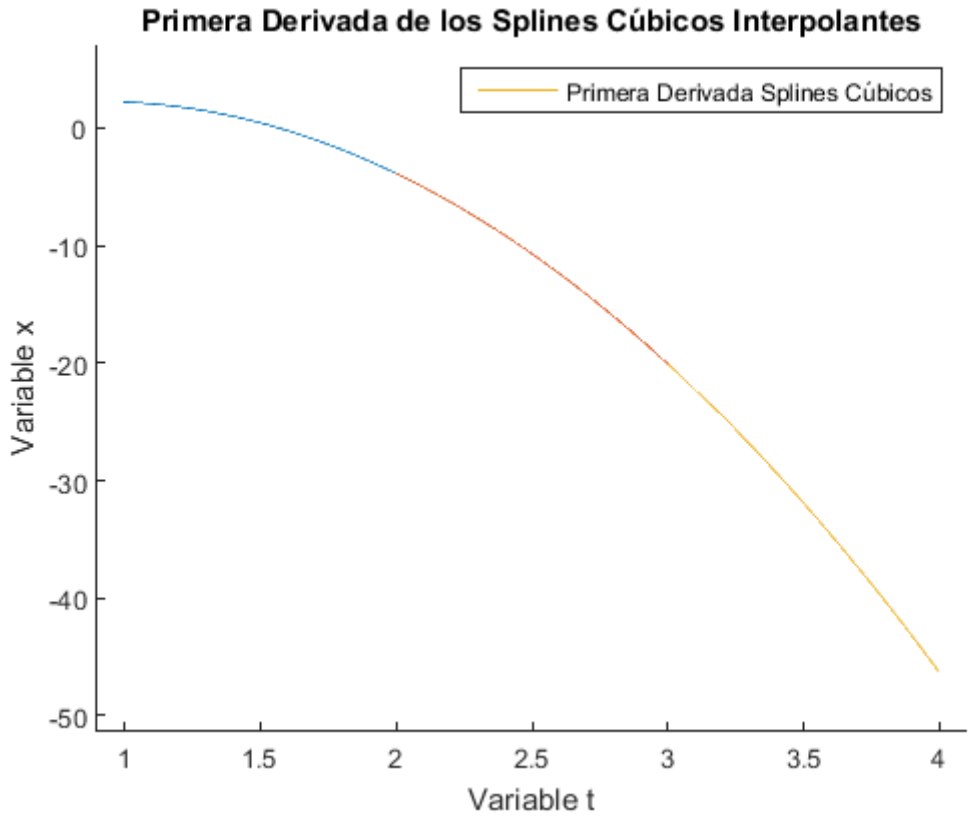


Fig. 10.1.4: representación de la primera derivada para la variable x.

En la última gráfica para la coordenada x se representa la segunda derivada de la función spline, que como podemos ver, todavía es continua. En la siguiente derivada, ya tendríamos una discontinuidad de salto.

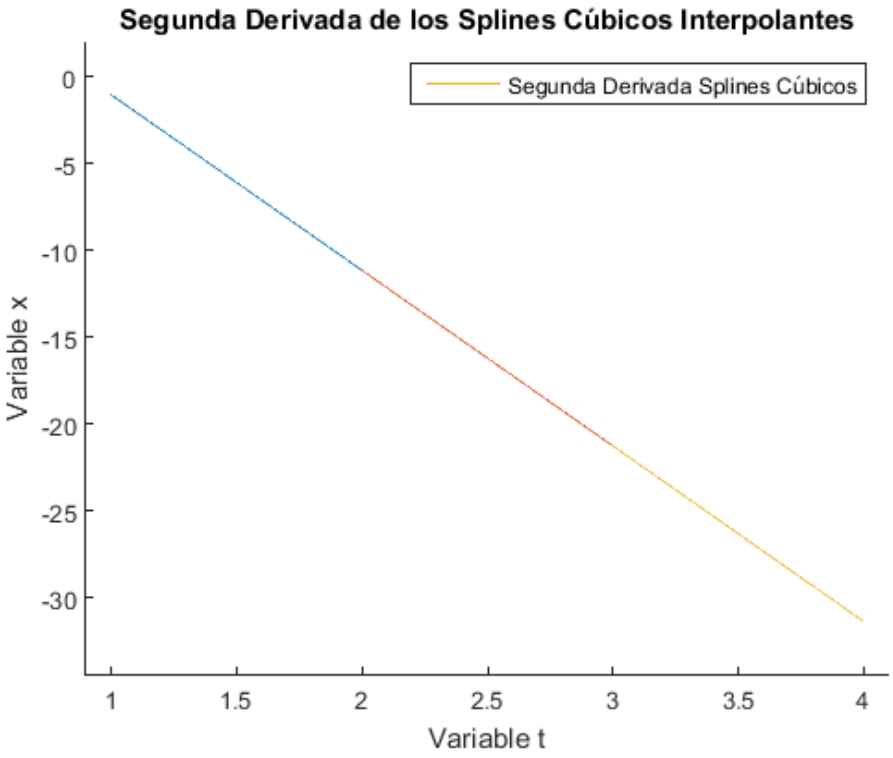


Fig. 10.1.5: representación de la segunda derivada para la variable x.

Seguidamente se muestran las gráficas obtenidas tanto para la variable y como para la variable z, las cuales tienen la misma explicación que la dada para la variable x:

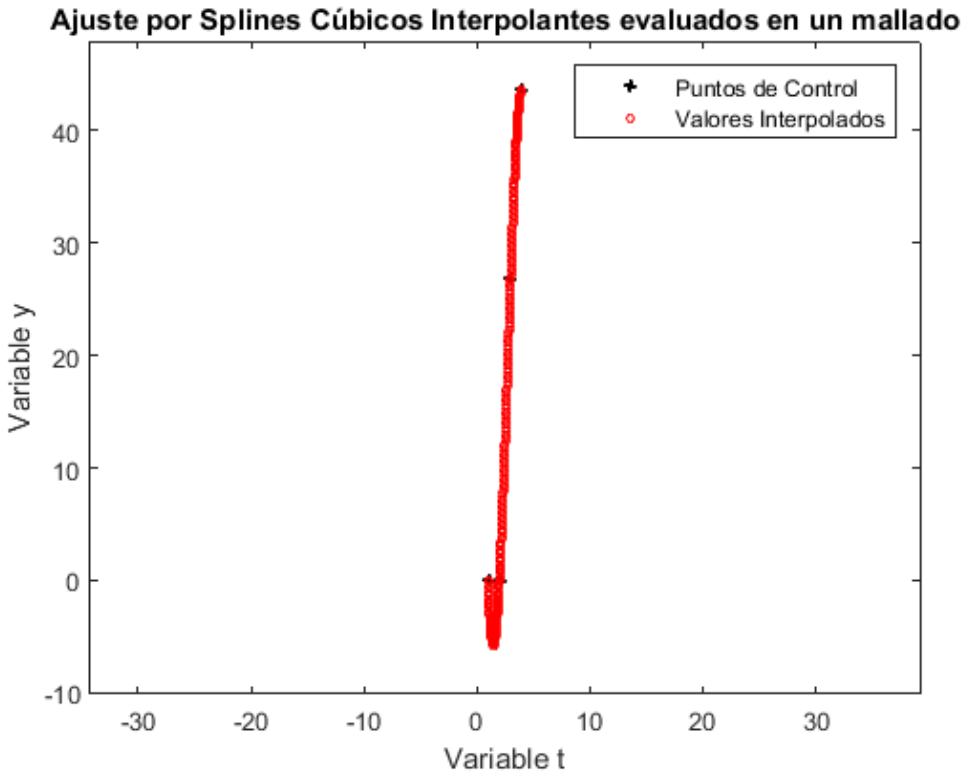


Fig. 10.1.6: ajuste para las coordenadas y.

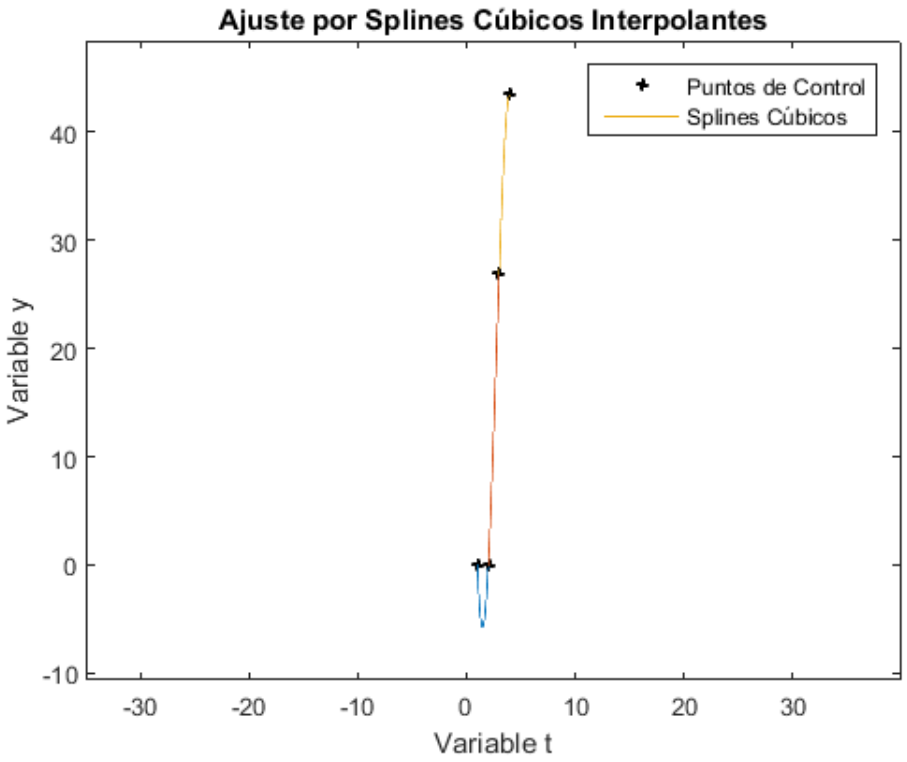


Fig. 10.1.7: ajuste por Splines Cúbicos Interpolantes para la variable y.

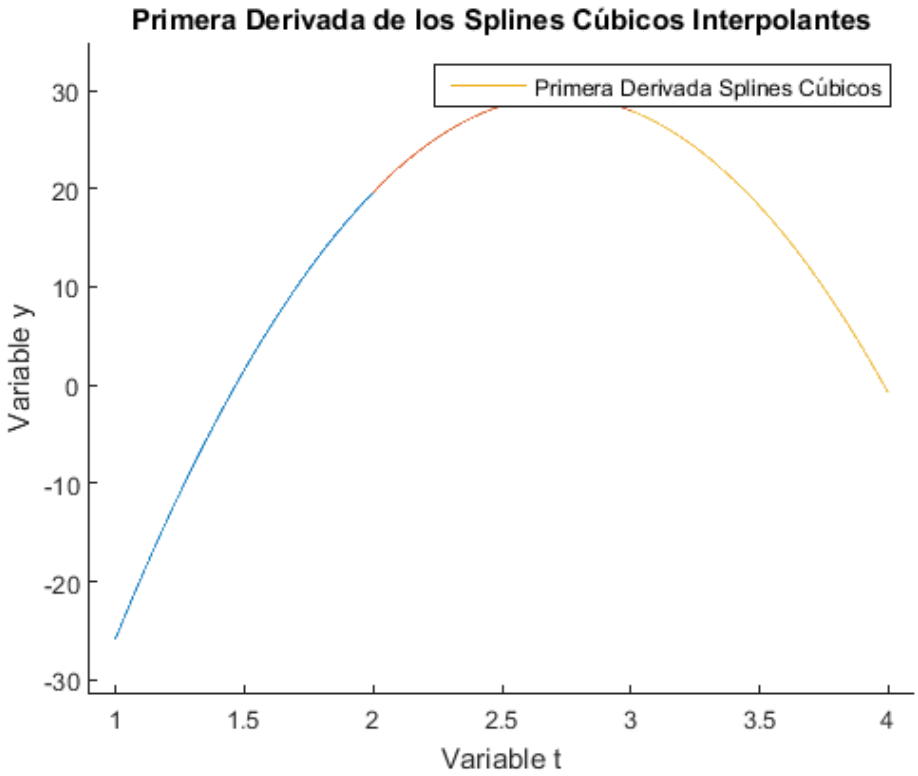


Fig. 10.1.8: representación de la primera derivada para la variable y.

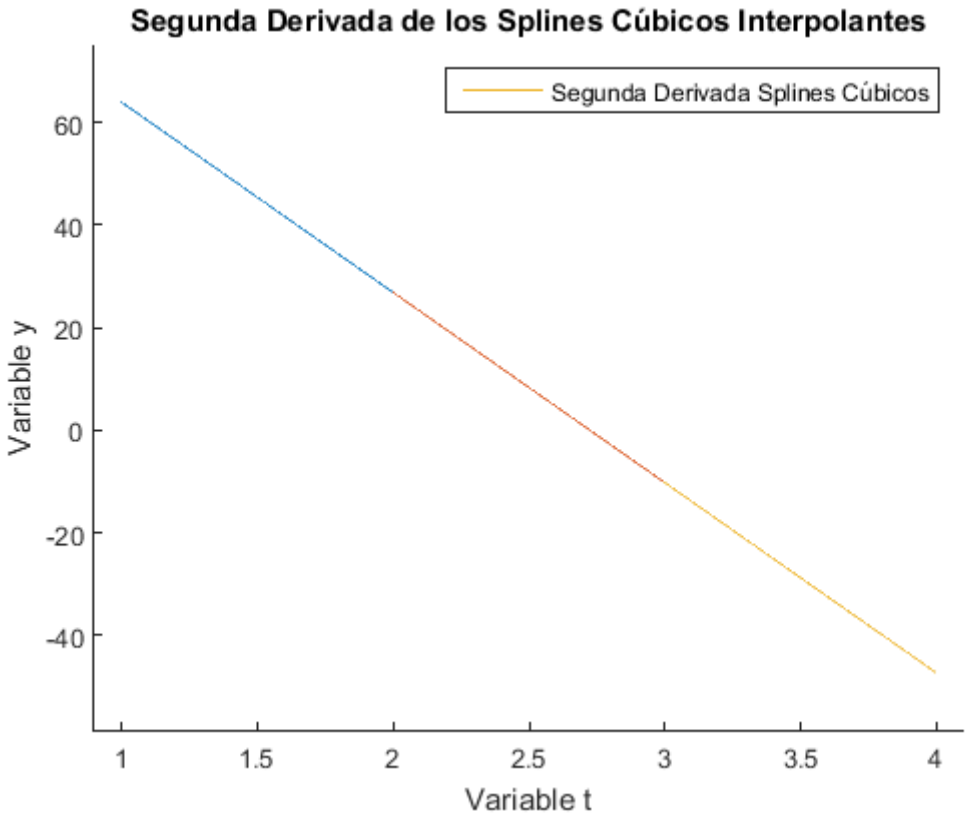


Fig. 10.1.9: representación de la segunda derivada para la variable y.

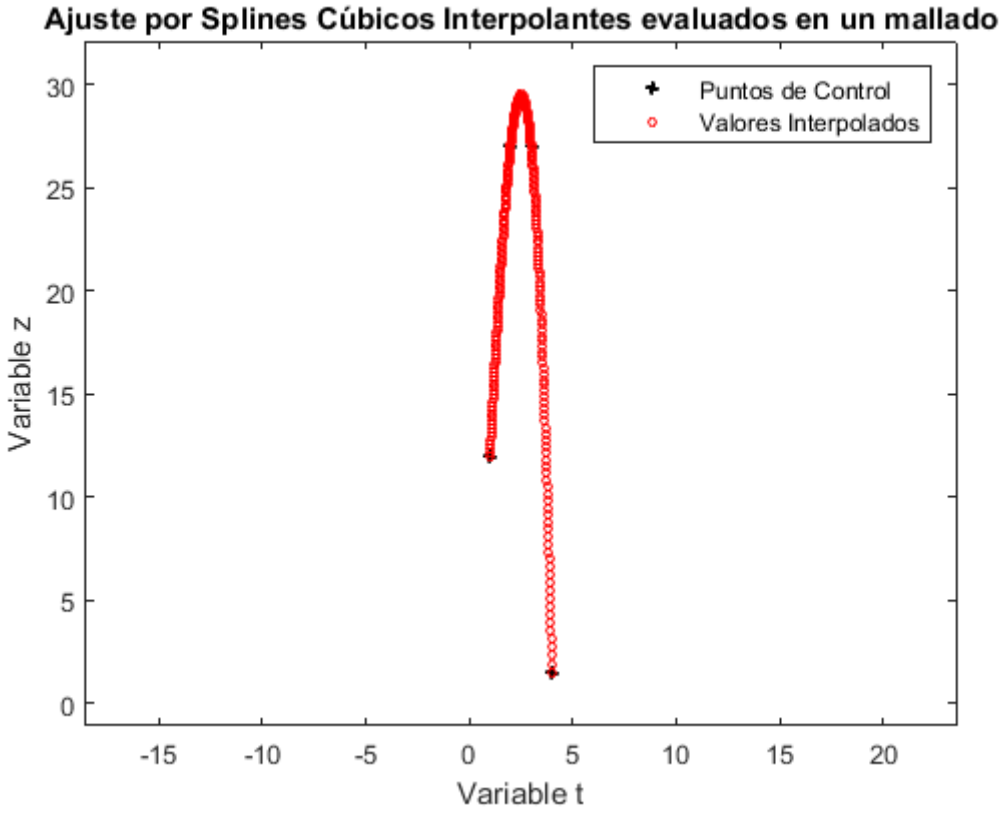


Fig. 10.1.10: ajuste para las coordenadas z.

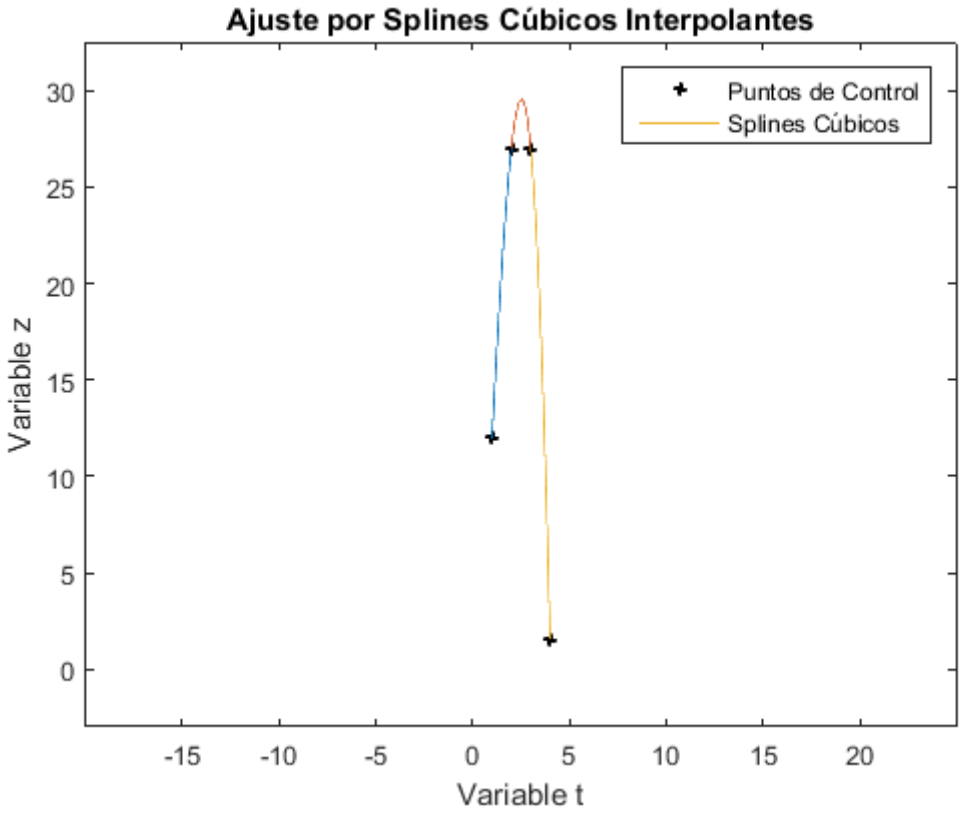


Fig. 10.1.11: ajuste por Splines Cúbicos Interpolantes para la variable z.

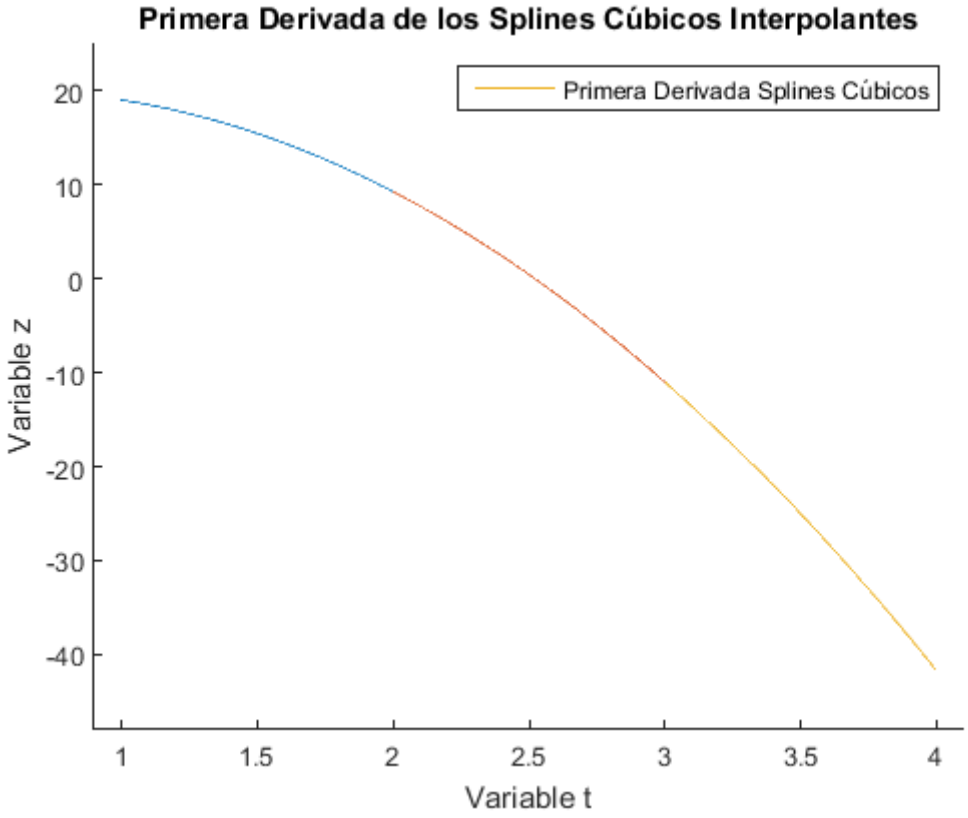


Fig. 10.1.12: representación de la primera derivada para la variable z.

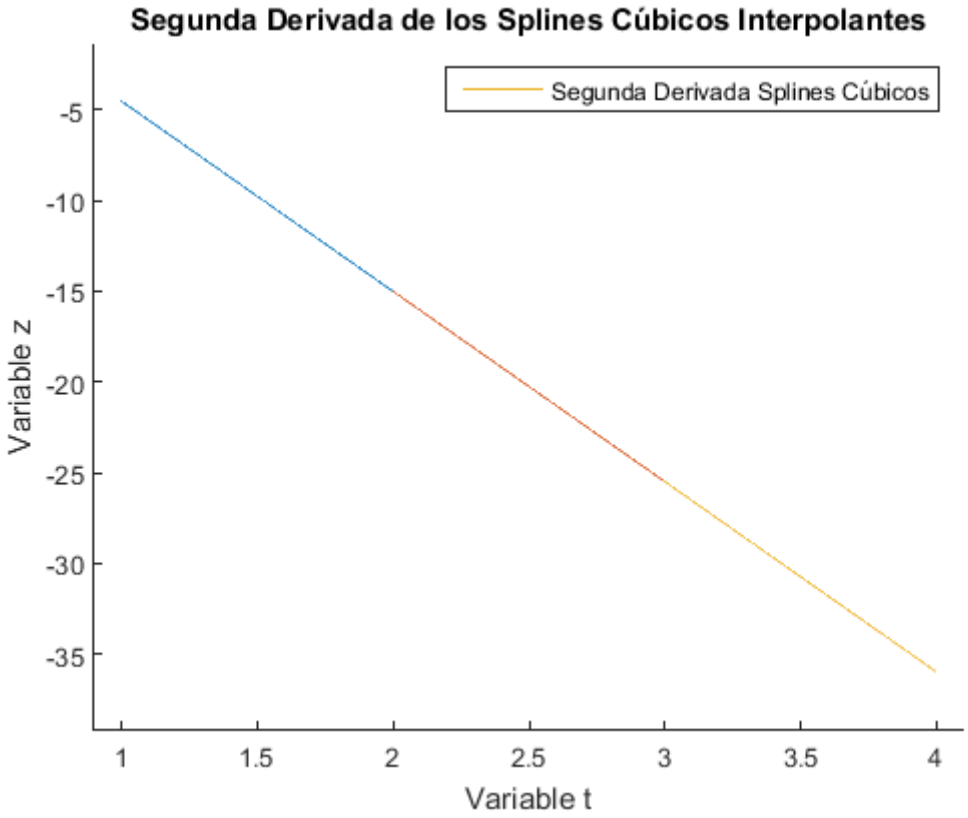


Fig. 10.1.13: representación de la segunda derivada para la variable z.

Finalmente, en la última gráfica que aparece en la figura 10.1.14 tenemos la curva final a seguir por el contenedor transportado por el brazo de grúa en el espacio. Esta curva es la resultante de representar juntas las coordenadas x, y, z.

En la figura 10.1.15 se muestra el recorrido desde otra perspectiva.

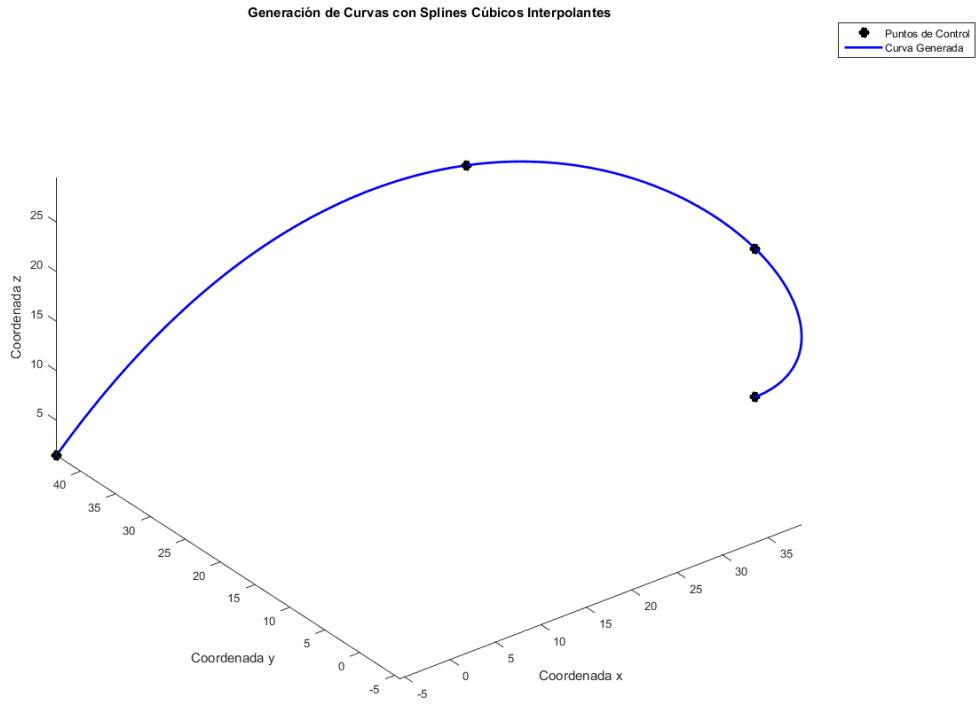


Fig. 10.1.14: curva final del movimiento del contenedor.

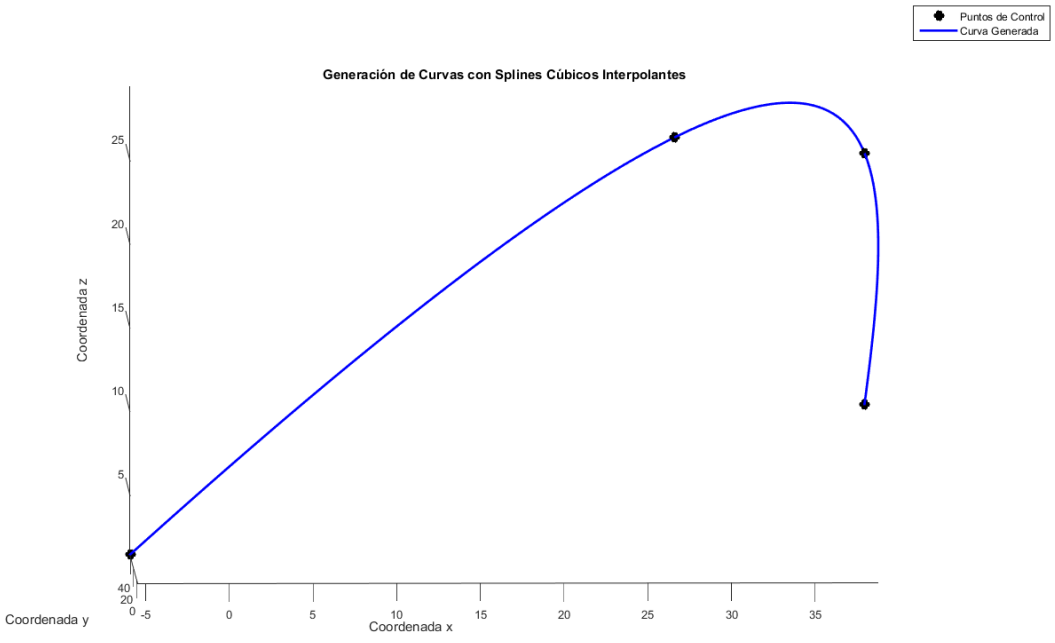


Fig. 10.1.15: curva final del movimiento del contenedor.

10.2 CASO PRÁCTICO 2

En este segundo caso práctico representaremos el contorno expandido de la pala de una hélice cuyas coordenadas se muestran en la siguiente tabla:

Sección	X	Y
0.2r	-171.78	210
0.3r	-205.52	315
0.4r	-235.82	420
0.5r	-261.18	525
0.6r	-284.42	630
0.7r	-300.87	735
0.8r	-312.86	840
0.9r	-320.22	945
0.95r	-261.05	997.5
1r	0	1050
0.95r	261.05	997.5
0.9r	320.22	945
0.8r	312.86	840
0.7r	300.87	735
0.6r	284.42	630
0.5r	267.52	525
0.4r	248.4	420
0.3r	229.9	315
0.2r	209.94	210

Tabla 10.2.1: coordenadas de la pala.

En la figura 10.2.1 tenemos el ajuste de la variable x, donde los puntos en rojo serían los valores interpolados entre cada uno de los puntos.

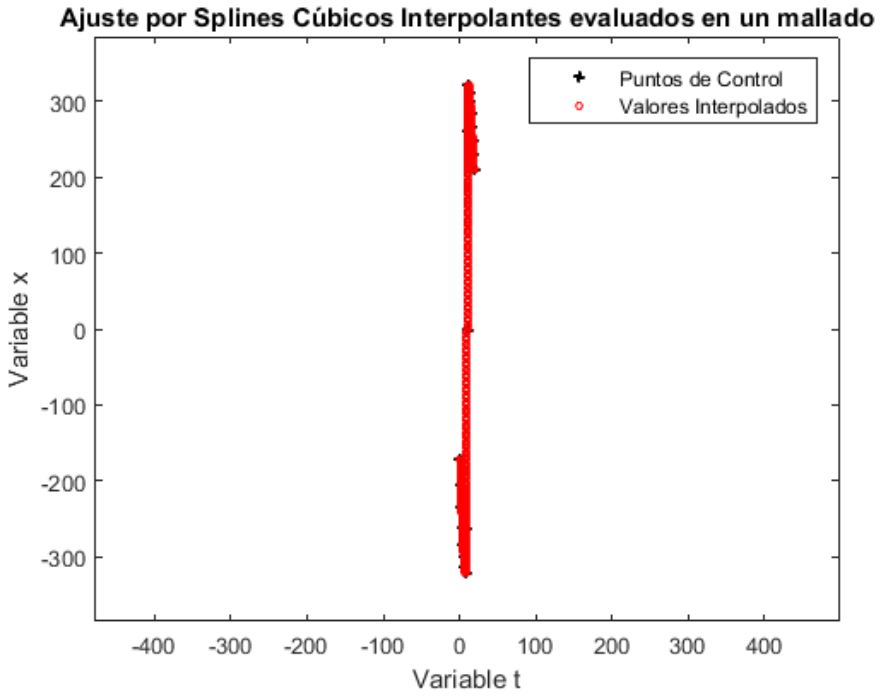


Fig. 10.2.1: ajuste para las coordenadas x.

En esta segunda gráfica, tenemos los splines sin interpolar, es decir la expresión simbólica, en la cual vemos que hay 18 trozos puesto que tenemos 19 puntos.

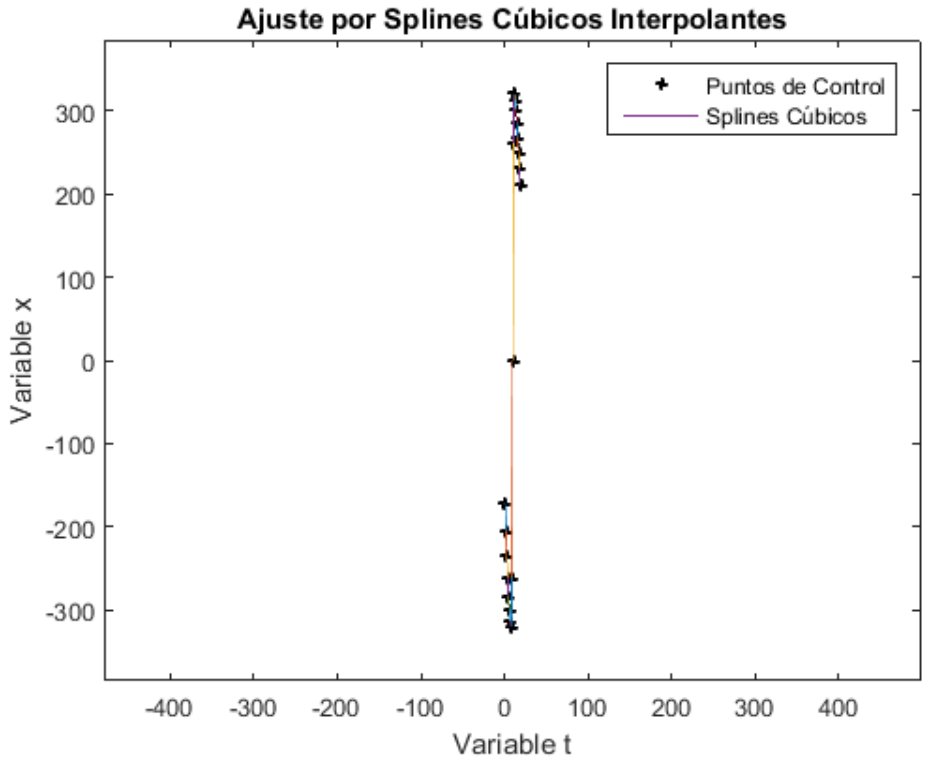


Fig. 10.2.2: ajuste por Splines Cúbicos Interpolantes para la variable x.

En esta tercera gráfica, figura 10.2.3, tenemos la primera derivada, donde podemos ver que efectivamente la función es continua.

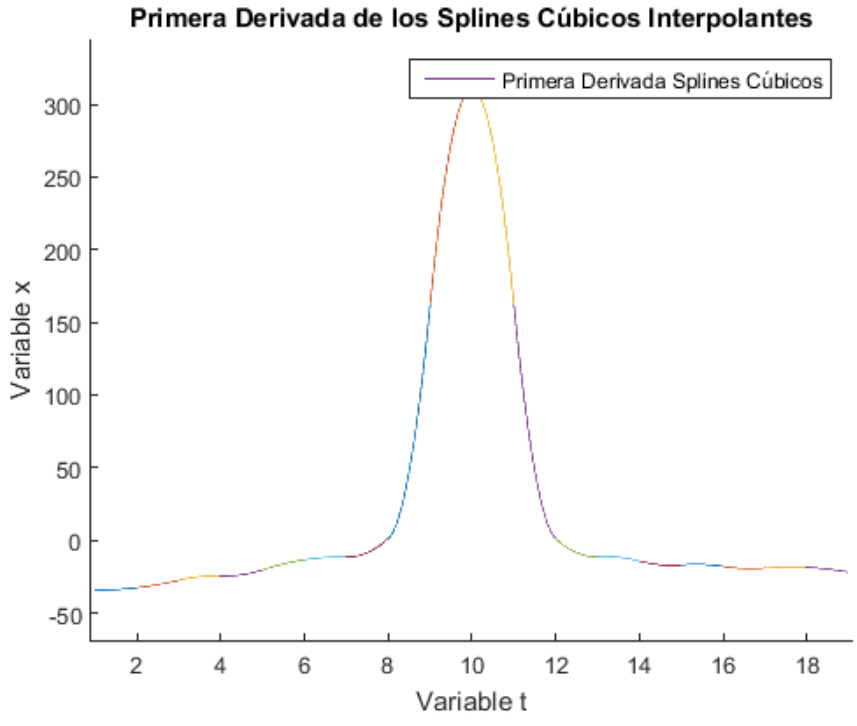


Fig. 10.2.3: representación de la primera derivada para la variable x.

En la última gráfica para la variable x se representa la segunda derivada, que como podemos ver, también es continua.

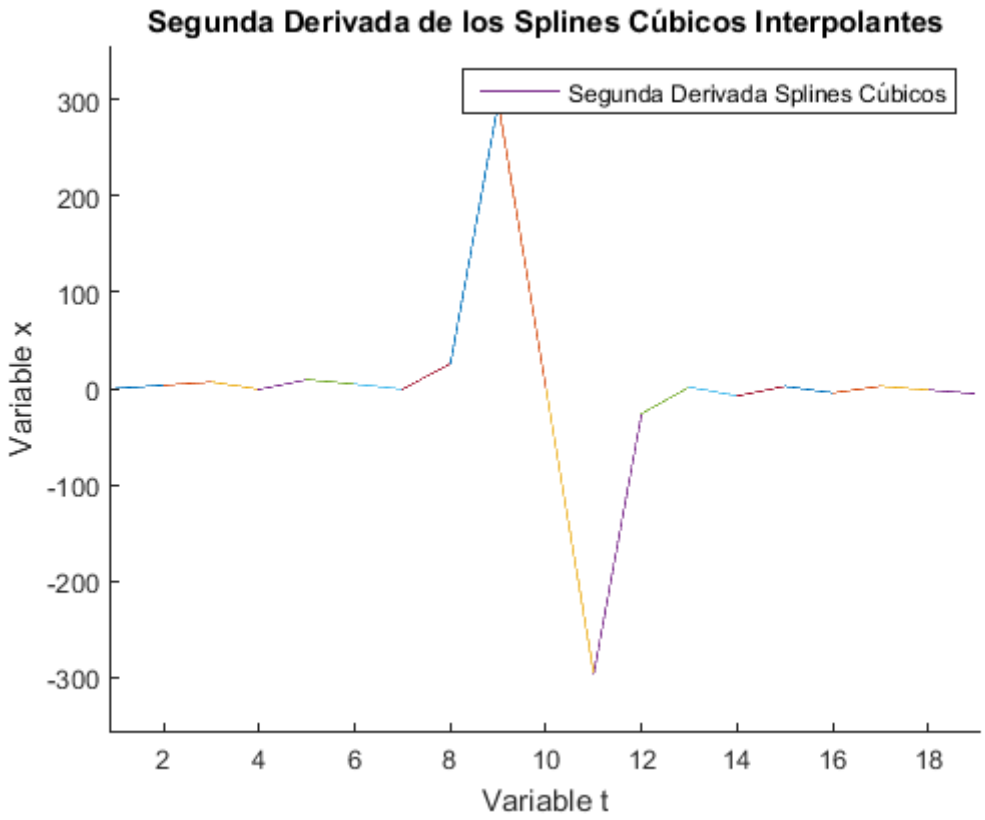


Fig. 10.2.4: representación de la segunda derivada para la variable x.

Pasamos ahora a mostrar los mismos gráficos para la coordena y.

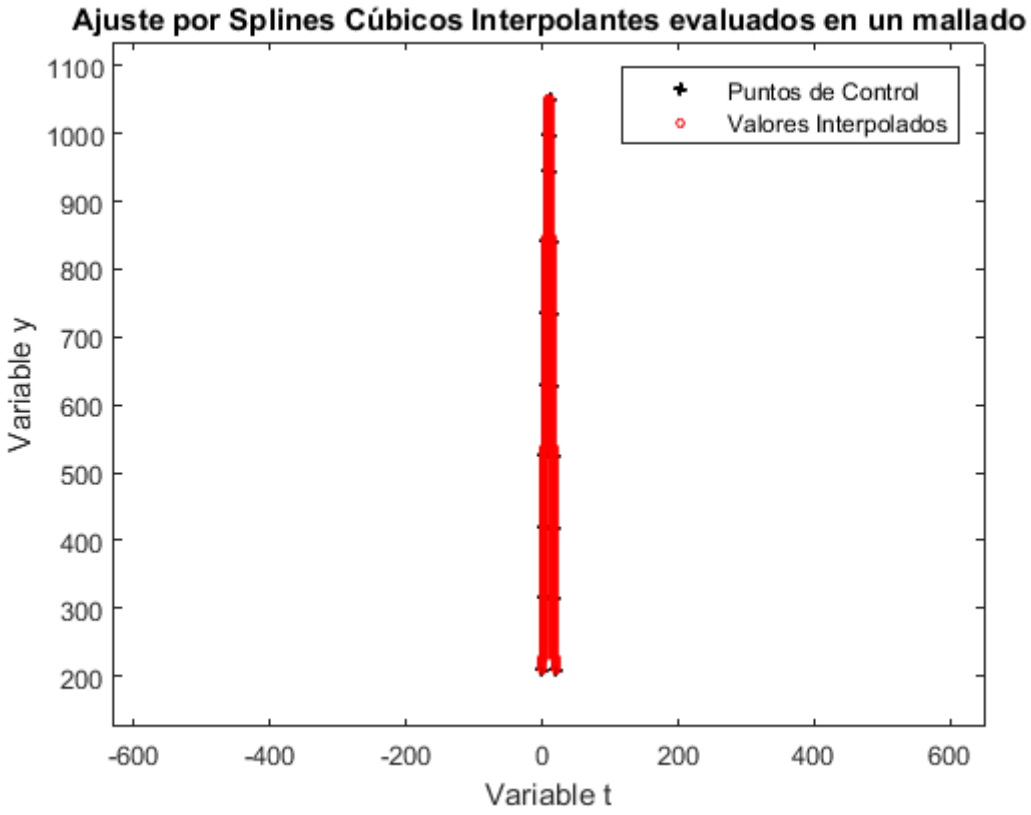


Fig. 10.2.5: ajuste para las coordenadas y.

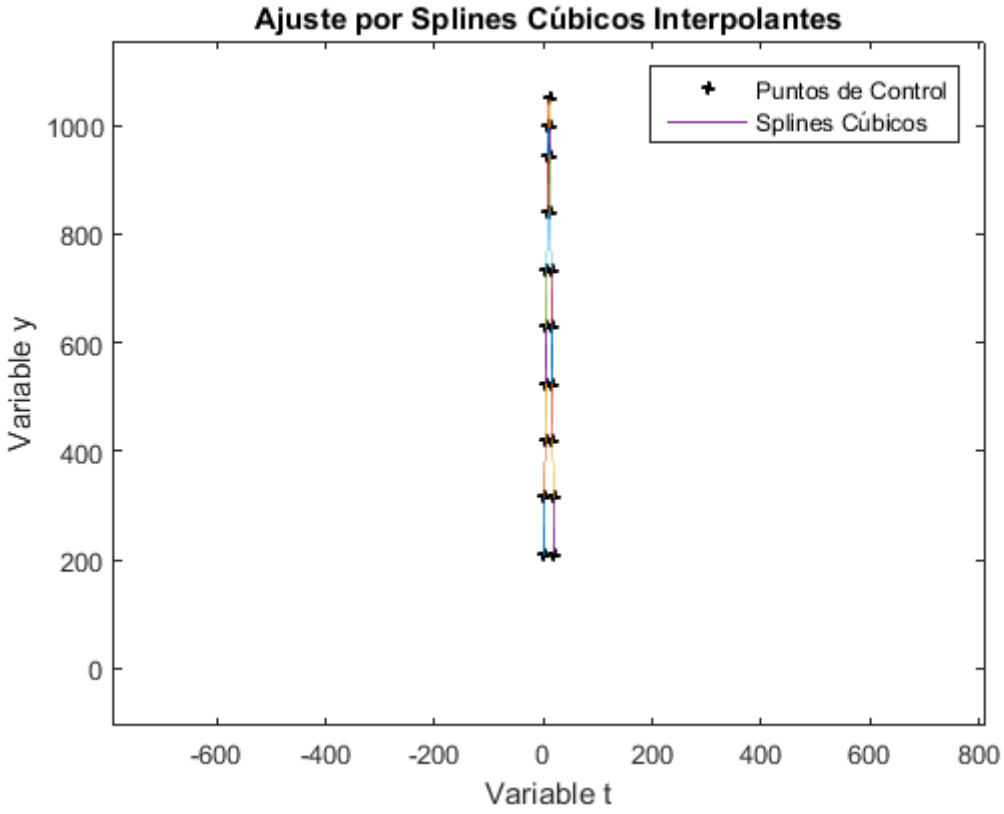


Fig. 10.2.6: ajuste por Splines Cúbicos Interpolantes para la variable y.

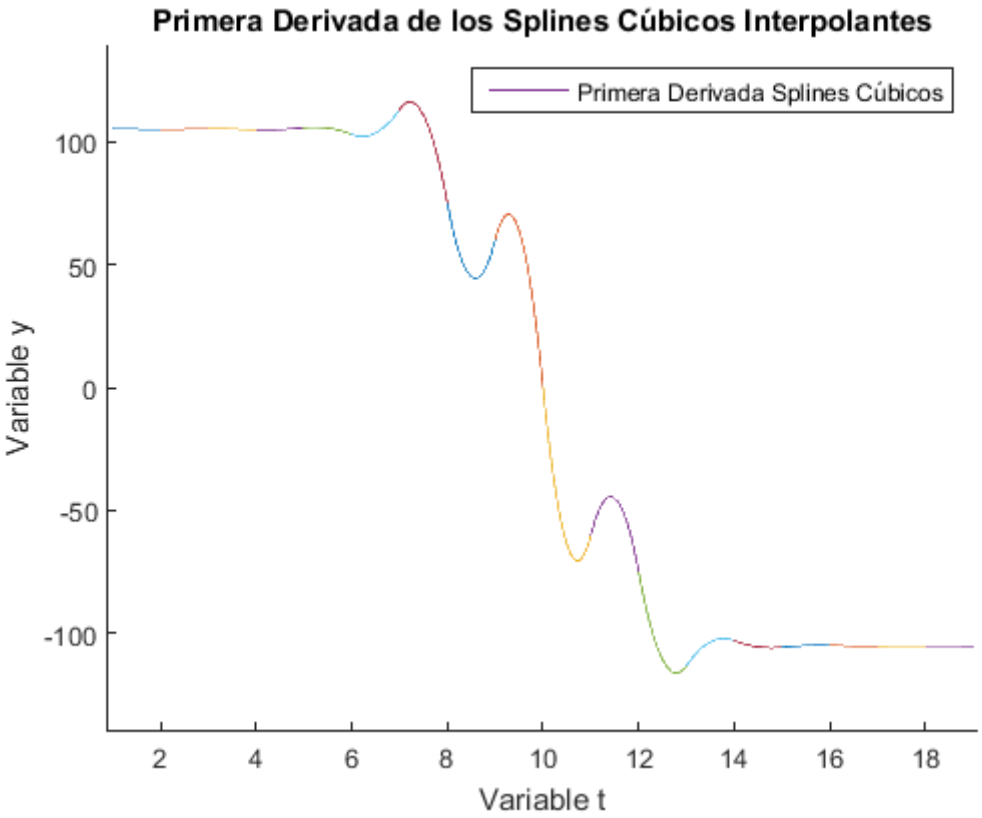


Fig. 10.2.7: representación de la primera derivada para la variable y.

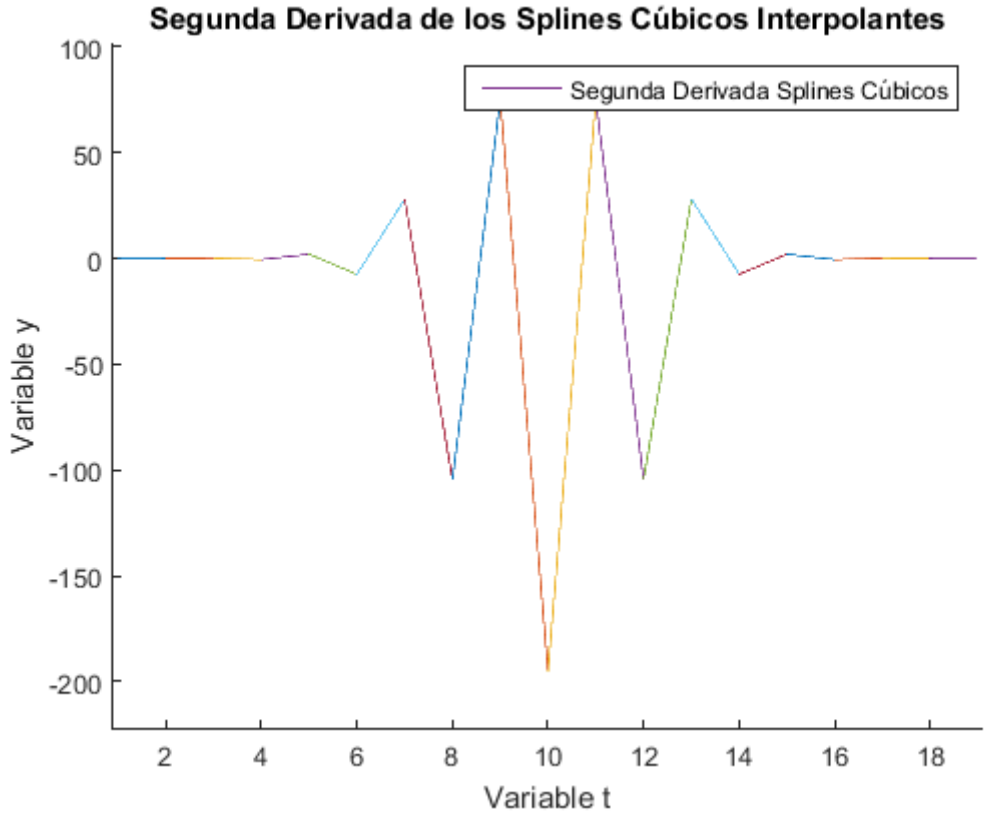


Fig. 10.2.8: representación de la segunda derivada para la variable y.

En este ejemplo práctico, como se tratra de la proyección frontal de una pala, trabajamos con una curva plana con lo cual no tenemos tercera coordenada. El resultado de unir los valores de cada una de las coordenadas obtenidas, nos arroja como resultado la aproximación discreta a la curva mostrada en la figura 10.2.9.

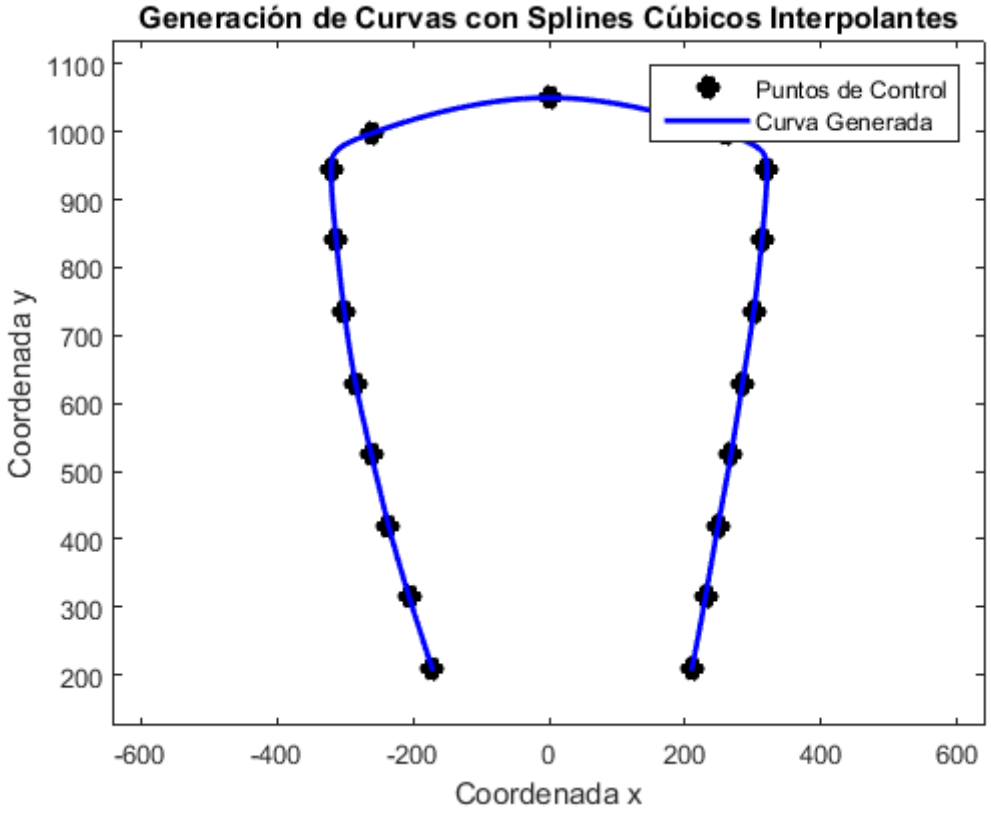


Fig. 10.2.9: representación de la curva generada mediante Splines Cúbicos Interpolantes.

10.3 CASO PRÁCTICO 3

En este tercer caso práctico vamos a realizar un estudio basándonos en la misma pala del ejemplo anterior. Utilizaremos los mismos datos de inicio, pero reconstruiremos la curva usando NURBS. Los resultados pueden verse en la figura 10.3.1. Las NURBS lo que hacen, es utilizar los puntos como puntos de control y generan una curva que se aproxima al contorno real de la pala en función del peso que le demos a cada punto, de esta forma cuanto mayor sea el peso asociado a cada punto más se aproximará la curva generada a dicho punto y al revés cuanto menor sea el peso. Además hay que definir apropiadamente el vector de nodos. En la figura 10.3.2 se ofrece una comparación gráfica de los resultados obtenidos por ambos métodos. En azul aparece la reconstrucción hecha mediante splines cúbicos interpolantes, y en trazo discontinúo con círculos negros la obtenida mediante NURBS.

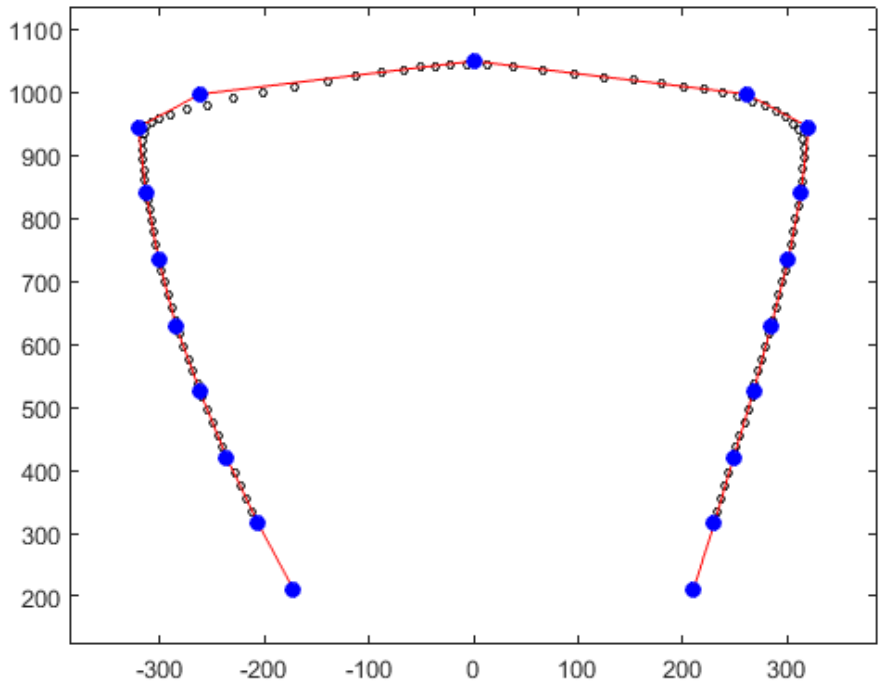


Fig. 10.3.3: representación de la pala mediante curvas NURBS.

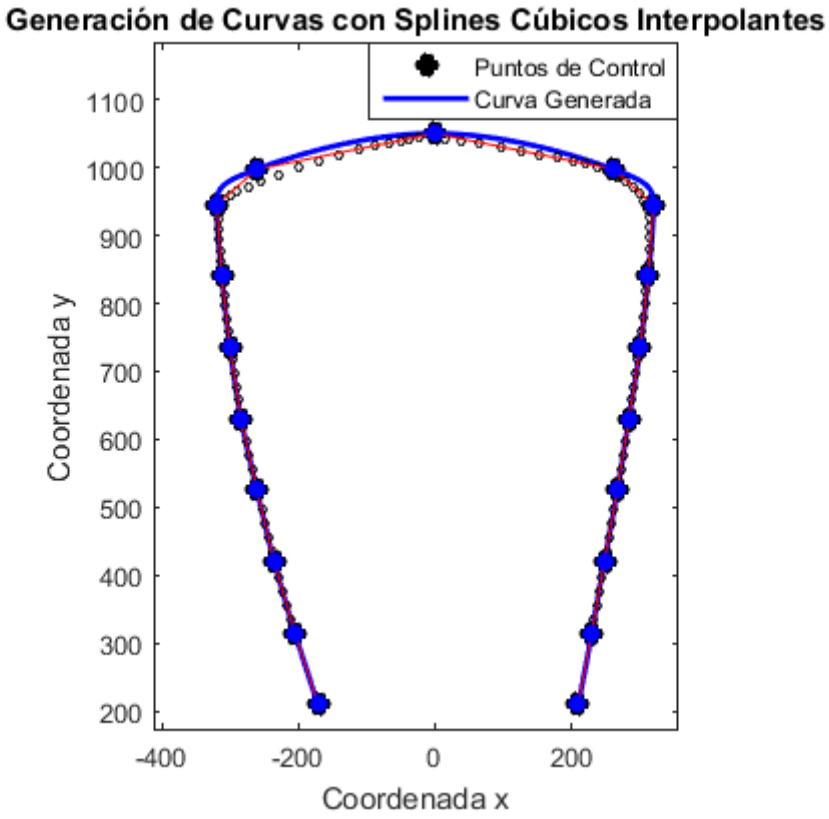


Fig. 10.3.2: comparación de la pala generada mediante Splines Cúbicos interpolante y NURBS.

10.4 CASO PRÁCTICO 4

Este caso práctico consistirá en dar forma a la cuaderna maestra de un buque, para lo que introduciremos los puntos de control que definen una aproximación a la curva que dará forma a nuestra sección. A continuación mediante NURBS, después de especificar adecuadamente las variables de entrada, se nos generará la forma de la cuaderna, la cual podremos modificar actuando sobre el peso que le corresponde a cada uno de los puntos de control.

En la figura 10.3.1 podemos observar la forma de nuestra cuaderna para el siguiente vector de puntos de control

P= [-8.1000, 7.0000; -8.1000, 6.0000; -8.1000, 5.0000; -8.1000, 4.0000; -8.0900, 3.0000;-8.0300, 2.0000; -7.8300, 1.0000; 0, 0; 0, 0; 7.8300, 1.0000;8.0300, 2.0000; 8.0900, 3.0000;8.1000, 4.0000; 8.1000, 5.0000; 8.1000, 6.0000;8.1000, 7.0000]

con los siguientes pesos ligados a cada punto

w= [50,1,1,1,1,2,2,2,2,2,2,1,1,1,1,50].

El vector de nodos se define con nodos no uniformes y periódico, de manera que se defina mejor el perfil.

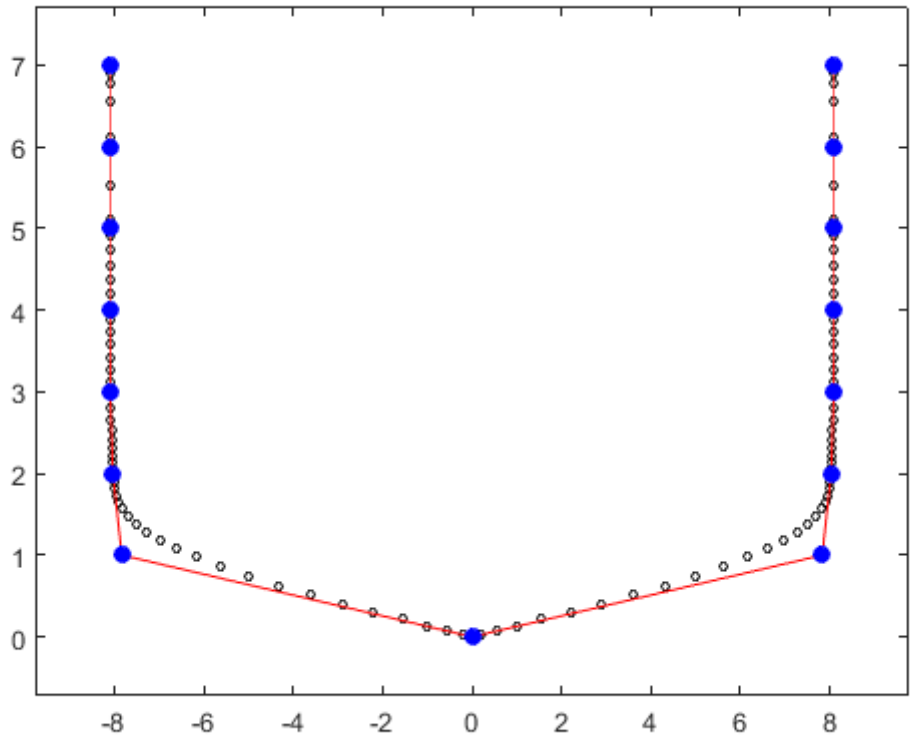


Fig. 10.3.1: cuaderna maestra 1.

La línea roja que se observa en la figura 10.3.1 sería el polígono de control que resulta de unir los puntos de control y la cuaderna resultante sería la nube de puntos negros, y como se puede observar tiene formas más suaves.

En la Fig 10.3.2 representaremos los mismos puntos de control, pero con una asignación de pesos distinta para cada uno de ellos. Se le ha asignado más poder de atracción a los puntos finales y centrales de la curva.

P= [-8.1000, 7.0000; -8.1000, 6.0000; -8.1000, 5.0000; -8.1000, 4.0000; -8.0900, 3.0000;-8.0300, 2.0000; -7.8300, 1.0000; 0, 0; 0, 0; 7.8300, 1.0000;8.0300, 2.0000; 8.0900, 3.0000;8.1000, 4.0000; 8.1000, 5.0000; 8.1000, 6.0000;8.1000, 7.0000]

W=[100,1,1,1,1,2,200,200,200,200,2,1,1,1,1,100]

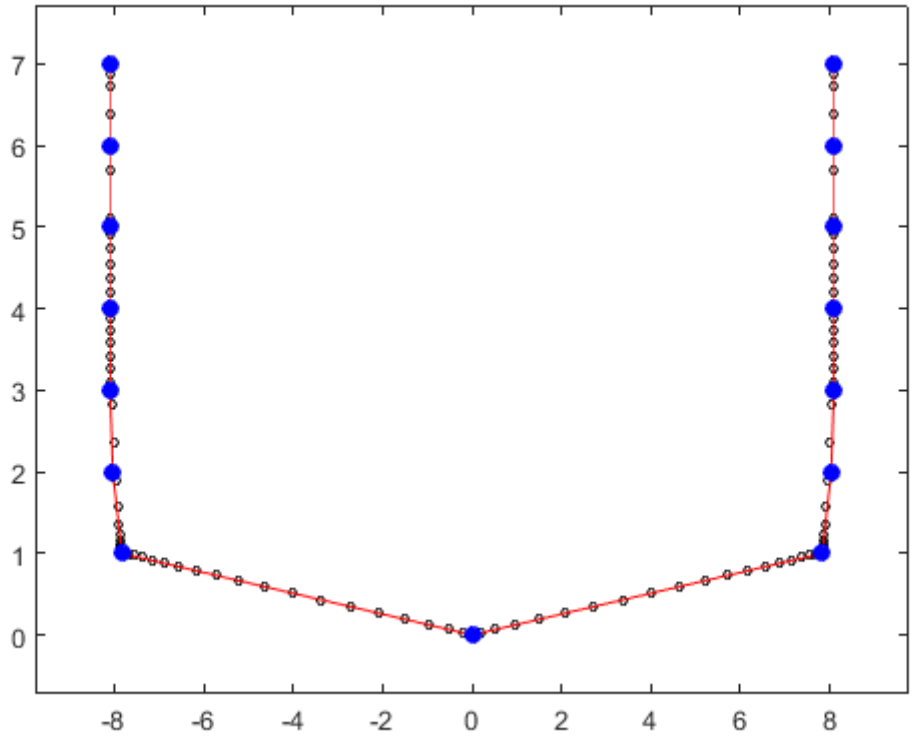


Fig. 10.3.2: cuaderna maestra 2.

Para terminar se muestra la comparativa entre la cuaderna 1 de la figura 10.3.1 y la cuaderna 2 de la figura 10.3.2, donde podemos ver la influencia que tiene darle mayor o menor peso a cada uno de los puntos de control.

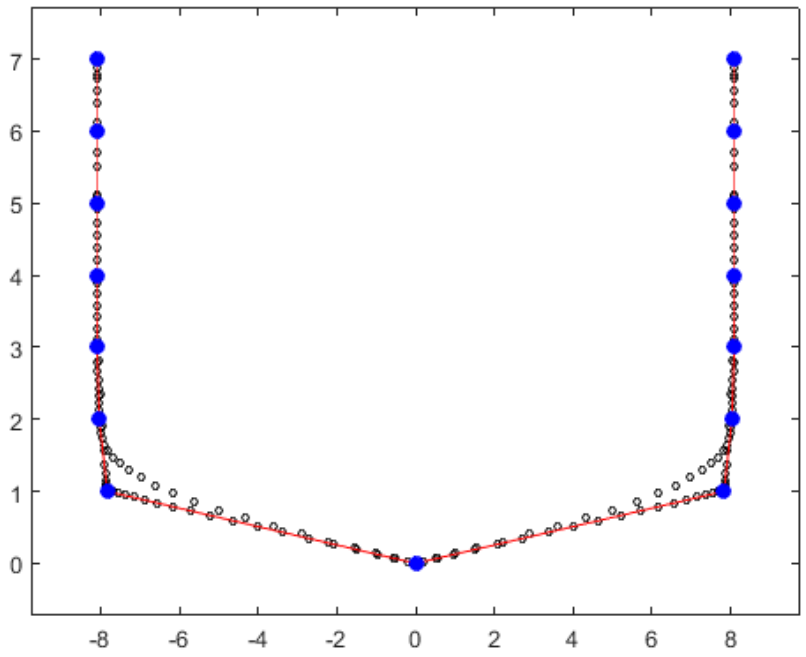


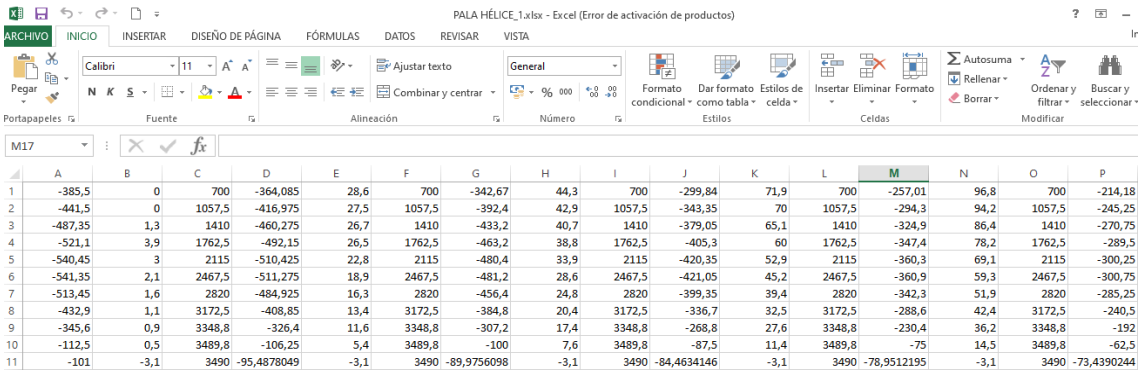
Fig. 10.3.2: comparación entre la cuaderna 1 y 2.

10.4 CASO PRÁCTICO 5

En este caso práctico representaremos dos palas de dos hélices diferentes en 3D, para ello nos hemos creado dos documentos excel (PALA HELICE_1.xls, PALA_HELICE_2.xls), en los cuales se representan los puntos de cada sección de la hélice correspondiente. Dichos datos se han extraído de la cartilla de trazado para dicha hélice.

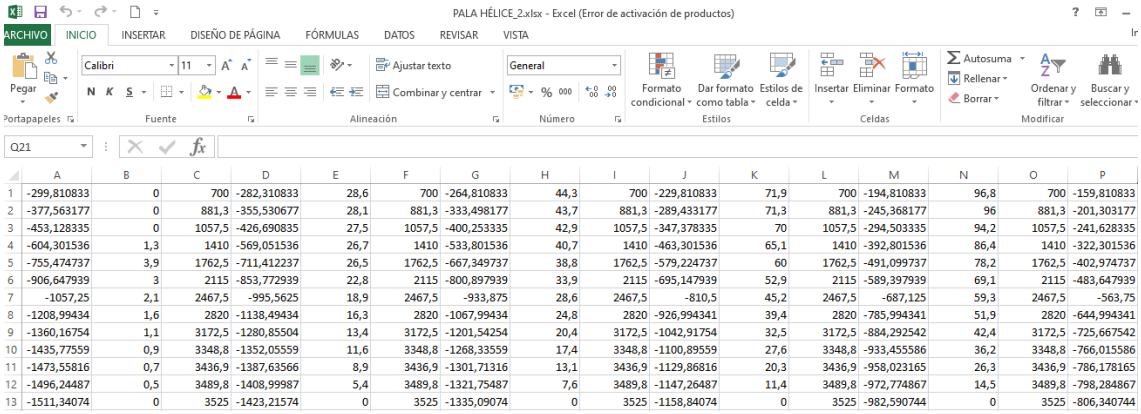
100	97.5	95	90	85	80	75	70	60	50	40	30	25	20	15	10	7.5	5	2.5	1.25	0		1	3525	4225
0.5	5.4	7.6	11.4	14.5	17.1	19.2	21	23.2	23.9	23.2	21	19.3	17.2	14.6	11.5	9.7	7.7	5.2	3.7	0	BACK	0.99	3489.8	4229
-0.5	-3	-3.1	-2.8	-2.3	-1.8	-1.3	-0.9	-0.3	-0.1	-0.3	-0.9	-1.4	-1.9	-2.5	-2.9	-3.1	-3.1	-2.9	-2.5	0	FACE			
0.7	8.9	13.1	20.3	26.3	31.4	35.6	39	43.4	44.9	43.4	39	35.7	31.6	26.6	20.7	17.2	13.3	8.8	6	0	BACK	0.975	3436.9	4234
-0.7	-3.8	-3.1	-1.3	0.6	2.4	4	5.4	7.3	7.9	7.3	5.3	3.9	2.2	0.3	-1.6	-2.6	-3.3	-3.7	-3.4	0	FACE			
0.9	11.6	17.4	27.6	36.2	43.5	49.5	54.4	60.8	63	60.8	54.5	49.7	43.8	36.7	28.2	23.4	17.9	11.6	7.8	0	BACK	0.95	3348.8	4243
-0.9	-3.8	-2.2	1.2	4.6	7.7	10.5	12.8	16	17	16	12.7	10.3	7.4	4.1	0.6	-1.2	-2.7	-3.8	-3.9	0	FACE			
1.1	13.4	20.4	32.5	42.4	51.5	58.8	64.7	72.4	74.9	72.4	64.9	59.2	52.2	43.7	33.7	29.9	21.4	13.9	9.3	0	BACK	0.9	3172.5	4222
-1.1	-4.2	-2.5	1.4	5.1	8.6	11.7	14.3	17.8	19.1	17.8	14.1	11.3	8	4.2	0.2	-1.8	-3.6	-4.7	-4.7	0	FACE			
1.6	16.3	24.8	39.4	51.9	62.5	71.3	78.3	87.5	90.5	87.5	78.7	72.1	64	54.1	42.3	35.5	27.7	18.4	12.6	0	BACK	0.8	2820	3975
-1.6	-6.4	-5.4	-2.7	0.1	2.8	5.2	7.4	10.4	11.5	10.4	7	4.4	1.3	-2.1	-5.6	-7.2	-8.3	-8.5	-7.5	0	FACE			
2.1	18.9	28.6	45.2	59.3	71.3	81.2	89.1	99.5	103	99.9	90.6	83.5	74.7	64	51.1	43.5	34.8	24	16.8	0	BACK	0.7	2467.5	3644
-2.1	-9.3	-9.8	-9.6	-8.8	-7.8	-6.7	-5.5	-3.7	-3	-3.7	-6.1	-8	-10.2	-12.6	-14.8	-15.5	-15.5	-14.1	-11.8	0	FACE			
3	22.8	33.9	52.9	69.1	82.9	94.4	103.7	116.4	121.3	119	109.2	101.6	91.9	79.9	65	56	45.5	32.1	23	0	BACK	0.6	2115	3324
-3	-14.2	-17.1	-20.8	-23.4	-25.1	-26.3	-27	-27.5	-27.5	-27.5	-28.1	-28.6	-29.2	-29.5	-29.2	-28.3	-26.5	-22.3	-18	0	FACE			
3.9	26.5	38.8	60	78.2	93.9	107.3	118.5	134.4	141.8	140.9	131.2	122.9	112	98.1	80.3	69.4	56.4	39.6	28	0	BACK	0.5	1762.5	3065
-3.9	-19.1	-24.2	-32	-38	-42.7	-46.4	-49.3	-52.9	-54.4	-54.2	-52.9	-51.7	-50.2	-47.9	-44.3	-41.3	-37	-29.5	-22.9	0	FACE			
1.3	26.7	40.7	65.1	86.4	105.1	121.4	135.4	156.4	167.9	169.3	159.8	150.5	137.6	120.6	98.3	84.4	67.7	46.3	31.8	0	BACK	0.4	1410	2921
-1.3	-20.4	-28.1	-39.9	-49.3	-57	-63.4	-68.6	-75.8	-79.4	-79.8	-77.3	-74.9	-71.5	-66.7	-59.5	-54.3	-47.1	-35.9	-26.7	0	FACE			
0	27.5	42.9	70	94.2	115.9	135.3	152.4	179.5	196.3	201.4	193	182.8	167.8	147	118.8	101	79.6	52.5	34.8	0	BACK	0.3	1057.5	2850
0	-23	-32.8	-47.9	-59.8	-69.7	-77.8	-84.6	-94.4	-99.9	-101.5	-99.1	-96.1	-91.7	-85	-74.8	-67.4	-57.2	-41.9	-29.9	0	FACE			
0	28.1	43.7	71.3	96	118.5	138.9	157.1	186.6	206	213.5	206.6	196.4	180.9	158.8	128.2	108.6	85	55.2	36	0	BACK	0.25	881.3	2829
0	-24.8	-35.9	-53.1	-66.7	-77.9	-87.3	-95.1	-106.8	-113.6	-116.1	-113.9	-110.7	-105.5	-97.6	-85.2	-76.2	-63.9	-45.7	-31.8	0	FACE			
0	28.6	44.3	71.9	96.8	119.7	140.6	159.5	191.1	213	223	218	208.3	192.7	169.7	137.1	115.9	90.3	57.9	37.1	0	BACK	0.199	700	2817
0	-26.8	-39.4	-59	-74.7	-87.7	-98.6	-107.7	-121.5	-130.1	-133.8	-132	-128.5	-122.6	-113.3	-98.3	-87.3	-72.3	-50.4	-34.1	0	FACE			
																						X	RADIO	PITCH

Tabla 10.4.1: cartilla de trazado



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	-385,5	0	700	-364,085	28,6	700	-342,67	44,3	700	-299,84	71,9	700	-257,01	96,8	700	-214,18
2	-441,5	0	1057,5	-416,975	27,5	1057,5	-392,4	42,9	1057,5	-343,35	70	1057,5	-294,3	94,2	1057,5	-245,25
3	-487,35	1,3	1410	-460,275	26,7	1410	-433,2	40,7	1410	-379,05	65,1	1410	-324,9	86,4	1410	-270,75
4	-521,1	3,9	1762,5	-492,15	26,5	1762,5	-463,2	38,8	1762,5	-405,3	60	1762,5	-347,4	78,2	1762,5	-289,5
5	-540,45	3	2115	-510,425	22,8	2115	-480,4	33,9	2115	-420,35	52,9	2115	-360,3	69,1	2115	-300,25
6	-541,35	2,1	2467,5	-511,275	18,9	2467,5	-481,2	28,6	2467,5	-421,05	45,2	2467,5	-360,9	59,3	2467,5	-300,75
7	-513,45	1,6	2820	-484,925	16,3	2820	-456,4	24,8	2820	-399,35	39,4	2820	-342,3	51,9	2820	-285,25
8	-432,9	1,1	3172,5	-408,85	13,4	3172,5	-384,8	20,4	3172,5	-336,7	32,5	3172,5	-288,6	42,4	3172,5	-240,5
9	-345,6	0,9	3348,8	-326,4	11,6	3348,8	-307,2	17,4	3348,8	-268,8	27,6	3348,8	-230,4	36,2	3348,8	-192
10	-112,5	0,5	3489,8	-106,25	5,4	3489,8	-100	7,6	3489,8	-87,5	11,4	3489,8	-75	14,5	3489,8	-62,5
11	-101	-3,1	3490	-95,4878049	-3,1	3490	-89,9756098	-3,1	3490	-84,4634146	-3,1	3490	-78,9512195	-3,1	3490	-73,4390244

Fig 10.4.2: puntos de la cartilla de trazado pasados a Excel (PALA HÉLICE_1.xls).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	-299,810833	0	700	-282,310833	28,6	700	-264,810833	44,3	700	-229,810833	71,9	700	-194,810833	96,8	700	-159,810833
2	-377,563177	0	881,3	-355,530677	28,1	881,3	-333,498177	43,7	881,3	-289,433177	71,3	881,3	-245,368177	96	881,3	-201,303177
3	-453,128335	0	1057,5	-426,690835	27,5	1057,5	-400,253335	42,9	1057,5	-347,378335	70	1057,5	-294,503335	94,2	1057,5	-241,628335
4	-604,301536	1,3	1410	-569,051536	26,7	1410	-533,801536	40,7	1410	-463,301536	65,1	1410	-392,801536	86,4	1410	-322,301536
5	-755,474737	3,9	1762,5	-711,412237	26,5	1762,5	-667,349737	38,8	1762,5	-579,224737	60	1762,5	-491,099737	78,2	1762,5	-402,974737
6	-906,647939	3	2115	-853,772939	22,8	2115	-800,897939	33,9	2115	-695,147939	52,9	2115	-589,397939	69,1	2115	-483,647939
7	-1057,25	2,1	2467,5	-995,5625	18,9	2467,5	-933,875	28,6	2467,5	-810,5	45,2	2467,5	-687,125	59,3	2467,5	-563,75
8	-1208,99434	1,6	2820	-1138,49434	16,3	2820	-1067,99434	24,8	2820	-926,994341	39,4	2820	-785,994341	51,9	2820	-644,994341
9	-1360,16754	1,1	3172,5	-1280,85504	13,4	3172,5	-1201,54254	20,4	3172,5	-1042,91754	32,5	3172,5	-884,292542	42,4	3172,5	-725,667542
10	-1435,77559	0,9	3348,8	-1352,05559	11,6	3348,8	-1268,33559	17,4	3348,8	-1100,89559	27,6	3348,8	-933,455586	36,2	3348,8	-766,015586
11	-1473,55816	0,7	3436,9	-1387,63566	8,9	3436,9	-1301,71316	13,1	3436,9	-1129,86816	20,3	3436,9	-958,023165	26,3	3436,9	-786,178165
12	-1496,24487	0,5	3489,8	-1408,99987	5,4	3489,8	-1321,75487	7,6	3489,8	-1147,26487	11,4	3489,8	-972,774867	14,5	3489,8	-798,284867
13	-1511,34074	0	3525	-1423,21574	0	3525	-1335,09074	0	3525	-1158,84074	0	3525	-982,590744	0	3525	-806,340744

Fig 10.4.3: puntos de la cartilla de trazado pasados a Excel (PALA HÉLICE_2.xls).

Por último utilizando el programa Matlab desarrollado "Rational_Bspline_surface3" con los datos anteriores, obtendremos una aproximación con más detalle a la superficie de cada una de las palas. La representación gráfica de ambas palas en 3D se puede observar en las figuras 10.4.4 y 10.4.5. Los comandos Matlab utilizados aparecen en las siguientes líneas:

```
>> ControlPoints='./Ficheros Superficies/PALA HÉLICE_1.xlsx';
 n=11;
 m=42;
 W=ones(n,m); W(n,1:m)=1; W(1:n,1)=10000; W(1:n,m)=10000;
 k=4;
 U(1:k)=1; U(k+1:n)=2:n-k+1; U(n+1:n+k)=n-k+2; U=U';
 l=4;
 V(1:l)=1; V(l+1:m)=2:m-l+1; V(m+1:m+l)=m-l+2; V=V';
 nU=50;
 nV=50;
 typeU_Bs='NP';
 typeV_Bs='NP';

[Pu0v0]=Rational_Bspline_Surface3(ControlPoints,W,U,k,V,l,nU,nV,typeU_
Bs,typeV_Bs)
 axis equal
```

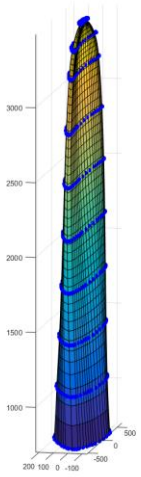


Fig 10.4.3: pala de la hélice en 3D.

```
>> ControlPoints='./Ficheros Superficies/PALA HÉLICE_2.xlsx';
 n=13;
 m=42;
 W=ones(n,m); W(n,1:m)=1; W(1:n,1)=10000; W(1:n,m)=10000;
 k=4;
 U(1:k)=1; U(k+1:n)=2:n-k+1; U(n+1:n+k)=n-k+2; U=U';
 l=4;
 V(1:l)=1; V(l+1:m)=2:m-l+1; V(m+1:m+l)=m-l+2; V=V';
 nU=50;
 nV=50;
 typeU_Bs='NP';
 typeV_Bs='NP';

[Pu0v0]=Rational_Bspline_Surface3(ControlPoints,W,U,k,V,l,nU,nV,typeU_
Bs,typeV_Bs)
 axis equal
```

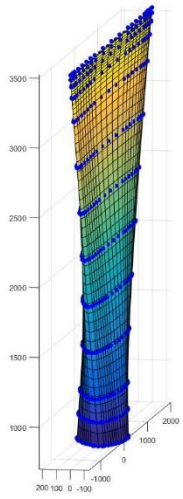


Fig 10.4.4: pala de la hélice en 3D.

Es interesante observar cómo se han dado inicialmente los puntos de control para ajustarse a un rectángulo de parámetros.

11. CONCLUSIONES

La teoría de B-Splines es de gran importancia por sus aplicaciones en distintas áreas y da lugar a métodos de aproximación e interpolación fáciles de implementar; sus aproximaciones poseen características, que nos dejan realizar cambios o modificaciones locales y por ello se adaptan fácilmente a la forma deseada. Las curvas de Bézier son un caso particular de las curvas B-Splines, de modo que el estudio de éstas últimas involucra el estudio de las primeras. En cuanto al desempeño de ambas, como ya se ha hablado anteriormente, el grado de las curvas de Bézier depende directamente del número de puntos de control, lo que puede provocar más cálculos en un momento dado. Mientras que con las curvas B-splines se puede usar un grado relativamente bajo, reduciendo con esto el número de operaciones. Lo mismo sucede con las superficies de Bézier y superficies B-Splines.

Lo que se ha hecho en este trabajo es estudiar los métodos de reconstrucción de curvas y superficies basados en B-splines, programar los algoritmos correspondientes y aplicarlos a casos prácticos de interés en el mundo naval. Éste ha sido el objetivo de los experimentos realizados, si bien las aplicaciones tienen muchas posibilidades en otros ámbitos de la ingeniería. Se trata de un trabajo fin de grado que puede servir de referencia para otros estudiantes que deseen adentrarse en el mundo de los NURBS y quieran continuar con el diseño de superficies más complejas. Durante el texto se han incluído varios ejemplos ilustrativos que hacen más asimilable el material desarrollado.

Para aplicar la teoría vista en el desarrollo del trabajo, se han programado también los algoritmos de De Casteljau y de De Boor. Con estos programas se obtienen las curvas y superficies de Bézier y B-splines respectivamente de una manera computacionalmente más económica, sobre todo en tiempo de computación.

Por último, también hemos desarrollado una interfaz gráfica que cubre todo el espectro de métodos estudiados en el proyecto desde polinomios de Bernstein a superficies NURBS, pasando por curvas de Bézier, y curvas B-splines racionales.

Este proyecto me ha servido para profundizar mis conocimientos en el diseño de curvas y superficies a un nivel tanto matemático como práctico. Dicho conocimiento tiene transferencias importantes a la hora de entender el procesamiento interno que hacen programas comerciales como Rhinoceros o Maxsurf de las superficies que aparecen en el ámbito naval. Además, he mejorado mis capacidades de programación, y en concreto mi conocimiento del lenguaje Matlab que me puede ser de mucha utilidad en mi posterior vida académica y profesional.

Este proyecto podría ampliarse tratando de reconstruir superficies más complejas, y considerando elementos base distintos dependiendo de la zona. Además, pueden utilizarse los datos obtenidos de las superficies para realizar cálculos hidrostáticos, hidrodinámicos y estructurales en base a ellos.

12. BIBLIOGRAFÍA

1. Libro: Aproximación Numérica ISBN: 8n-370-5513x, Universitat de Valencia 2002. Autores: S. Amat, F. Arandiga, J.V Arnau, R. Donat, P. Mulet, R. Peris.

2. Tesis: Curvas y superficies B-splines, Universidad tecnológica de la Mixteca. Autor: Martínez, Nancy. Directores: Álvarez, M.C Luz del Carmen; Castañeda, Cuauhtémoc H.

3. On an algorithm to adapt spline approximations to the presence of discontinuities. Numer Algorithms 80 (2019), no 3, 903-936. Autores: Amat Sergio; Ruiz, Juan; Trillo, Juan C.

4. Cálculo Numérico, Programación Aplicada, JTP Cálculo Numérico. Autora: Adriana M. Apaza.

5. Modelling in Science Education And Learning Volume 4, No. 14, 2011. Instituto Universitario de Matemática Pura y Aplicada. Universidad Politécnica de Valencia. Autor: Bravo, Orlando.

6. Transparencias Algoritmo De Boor, CIMAT. Autor: López, Manuel G.

7. Manual básico de Matlab. Edit. Complutense de Madrid (2009). Autor: Casado Fernández, M.C

8. Construcción Naval, Universidad Politécnica de Cartagena, Departamento de Física Aplicada y Tecnología Naval. [Apuntes, tema 1 "Dimensiones y características del buque"] Cobo, Francisco J.

9. Proyecto final de carrera: Splines Cúbicos Interpolantes en el Diseño Naval, Universidad Politécnica de Cartagena, Departamento de Matemáticas Aplicada y Estadística. Autor: Gallego, Irene. Directores: Trillo, Juan C; Angosto, Carlos.

10. Proyecto final de Master: Herramientas de diseño en el ámbito naval, implementación y uso del software informático adecuado, Universidad de Politécnica de Cartagena, Departamento de Matemáticas Aplicada y Estadística. Autor: Paredes, José J. Directores: Trillo, Juan C; Ruiz, Juan.

11. Construcción Naval, Universidad Politécnica de Cartagena, Departamento de Física Aplicada y Tecnología Naval. [Apuntes, tema 3 "Definiciones de las características del buque"] Esteve, Jerónimo.

12. Metodología de representación geométrica de Hélices de buques Para su clasificación topológica, XIV Congreso Internacional de Ingeniería Gráfica (2002). Autores: Suffo, Miguel; Sánchez-Sola, José M; Bienvenido, Rafael; Gómez-Ortíz Rafael; Sánchez-Carrilero, Manuel.

13. Computer-aided geometric design and computer graphics: B-splines and Nurbs Curves and Surfaces**,** University of Cantabria, Department of Applied Mathematics and Computational Sciences. Autor: Andrés Iglesias.

14. Hidrodinámica. Resistencia y Propulsión, Universidad Politécnica de Cartagena, Departamento de Física Aplicada y Tecnología Naval. [Apuntes, tema 9 "Geometría de la hélice"] García, Domingo.

15. Tesis: El Método de expansión de Bernstein para determinar la estabilidad de familias de Polinomios, Universidad Autónoma Metropolitana Unidad Iztapalapa, Departamento de Matemáticas. Autor: Carlos Arturo Loredo Villalobos. Director: Aguirre, Baltazar.

16. Curvas interesantes en matemáticas: CURVAS DE BÉZIER, Temas para la Educación, revista digital para profesionales de la enseñanza, Nº16 (2011) ISNN: 1989-4023. Autora: Mendoza, María J.

17. Proyecto final de carrera: Esquemas de Subdivisión no lineales en el diseño gráfico, Universidad Politécnica de Cartagena, Departamento de Matemáticas Aplicada y Estadística. Autor: Moreno, Pedro A. Directores: Amat, Sergio.

18. Métodos de Bézier y B-splines, KIT Scientific Publishing (2005). Autores: Paluszny, M; Prautzsch, H; Boehm, W.

19. Quadratic Splines Interpolations: Obtaing an explicit formula. Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Universidad Nacional de Rosario. Autores: Alberto J. Ferrari; Luis P. Lara; Eduardo A. Santillan Marcus.

20. Proyecto final de carrera: Métodos numéricos para el diseño de curvas y superficies en Ingeniería Naval, Universidad Politécnica de Cartagena, Departamento de Matemáticas Aplicada y Estadística. Autor: Flores, Zamora. Directores: Busquier, Sonia; Ruiz, Juan.

21. Diseño asistido por ordenador, Universidad de Granada, ETS. Ingeniería Informática, Dpt. Lenguajes y Sistemas Informáticos. Autor: Torres, J.C.

22. Libro: Cálculo Numérico Universidad Politécnica de Cartagena, Departamento de Matemática Aplicada y Estadística. Autor: Antonio Vigueras Campuzano.

23. Proyecto final de carrera: Interpolación y Aproximación por curvas Nurbs, Universidad Politécnica de Cartagena, Departamento de Matemáticas Aplicada y Estadística. Autor: Isern, Miguel. Directores: Amat, Sergio; Busquier, Sonia.