Trabajo Final de Grado

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

INTERPOLACIÓN SPLINE Y APLICACIÓN A LAS CURVAS DE NIVEL

Autor: José Antonio Chica Jiménez

Director: Dr. Miquel Bosch
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi

Barcelona, 19 de enero de 2018

Abstract

In the real world you don't see the straight lines, both the shapes and the trajectories obey curved lines and curved surfaces that do not follow a generic rule.

From always we has tried to study and to be able to control all type of surfaces, either to adjust movements to configurations as in physics or to parameterize surfaces.

One of the best ways we have today so that it controlled in a way that matches the data is the interpolation.

For the adjustment of curves, splines are used to approximate complicated shapes. The simplicity of representation and the ease of calculation of Splines make them popular for the representation of computer curves, particularly in the field of computer graphics.

This text is an introduction to the study of Spline interpolation curves, as well as the main methods, their varieties and their practical uses.

Resumen

En el mundo real, de lo que menos se encuentra son las líneas rectas, tanto las formas como las trayectorias obedecen a líneas y superficies curvas que no siguen una regla genérica.

Desde siempre se ha intentado estudiar y dar reglas para poder controlar y generar todo tipo de superficies, ya sea ajustando movimientos a ecuaciones como en física o parametrizando superficies.

Una de las mejores formas que tenemos actualmente para poder controlar de alguna manera datos tan complejos y aleatorios es la interpolación, dando lugar a funciones que se corresponden con los datos.

Para el ajuste de curvas, los Splines se utilizan para aproximar formas complicadas. La simplicidad de la representación y la facilidad de cómputo de los Splines los hacen populares para la representación de curvas en informática, particularmente en el terreno de los gráficos por ordenador.

Este texto intenta dar una introducción al estudio de las curvas de interpolación Spline, así como los principales métodos, sus variedades y sus usos prácticos.

Agradecimientos

Agradecer primeramente a Miquel Bosch por aceptarme como su primer trabajo como tutor, por sus consejos y apoyo en el proyecto.

Al *Institut Cartogràfic i Geològic de Catalunya* del cual he podido extraer mallas de puntos de sus mapas vía web. Pese a que no forman parte del proyecto, fueron útiles como ejemplos mientras se realizaba éste.

A a la doctora Juani Colmenero por su ayuda en conocimientos y por facilitarme los datos necesarios para realizar un estudio real en la parte práctica del proyecto, tanto las coordenadas, (x, y, z), del municipio de Macael y sus montañas de Mármol, como del municipio de Roquetas de mar.

También quiero hacer una mención a Antonio por su tiempo y dedicación en la búsqueda de coordenadas, para el estudio real, que usamos en la parte práctica del trabajo. A Borja Ruíz, por la ayuda y experiencia aportada al tener conocimientos del campo que tratamos. A Noelia Atencia, pese a sus problemas en el ámbito de la programación, por su ayuda y apoyo. Así como a Mari Juani Jiménez por su apoyo.

Índice

1. Introducción **1**

2. Interpolación **2**

2.1. Problema general de interpolación 3

2.2. Métodos usuales 4

2.3. Error en la interpolación 6

2.4. Beneficios de los Splines 8

3. Splines **10**

3.1. Algo de Historia 10

3.2. La función Spline 11

4. Splines en una variable **14**

4.1. Spline lineal 14

5. Splines cúbicos **17**

5.1. Spline cúbico C^1 o Interpolación de Hermite 17

5.2. Spline cúbico C^2 19

5.3. Spline cúbico y curvatura 21

5.4. Algoritmo de resolución 23

6. Splines en dos variables **25**

6.1. Spline bicúbico 25

6.2. El cuadrado unidad 28

6.3. Derivadas 29

7. Curvas de nivel **33**

7.1. Datos de entrada y salida del programa 34

7.2. Resultados del experimento 35

8. Conclusiones **40**

9. Anexo **42**

1. Introducción

El proyecto

En la vida, cada acción no sigue una regla genérica, no existen las líneas rectas ni las superficies rectas. Tanto las trayectorias como las formas se guían por líneas y superficies curvas que no siguen una regla.

En este proyecto lo que se quiere encontrar es un método simple y eficiente para poder determinar reglas sobre estas líneas y superficies. A partir de la interpolación, podemos obtener funciones que se asemejan a estas reglas genéricas.

Los Splines es un método de interpolación muy utilizado actualmente, y gracias a ellos obtenemos funciones muy simples y fáciles de calcular que nos permiten conseguir modelar los datos obtenidos experimentalmente y definir reglas para el estudio de estos datos.

Pese a la búsqueda de información y al estudio hecho de los métodos. Lo realmente interesante ha sido la búsqueda de las superficies a partir de los datos experimentales.

Estructura de la Memoria

En el Capítulo 2 damos unas nociones básicas de la teoría de la interpolación, tanto ejemplos como comparativas de algunos métodos, decantándonos finalmente por la teoría de los Splines.

El Capítulo 3 es una introducción a la teoría de los Splines y un poco de la historia de sus orígenes.

En el Capítulo 4 se exponen los principales Splines con los que se puede trabajar en una dimensión.

El Capítulo 5 es un tema detallado de la teoría de los Splines cúbicos, ya que es el Spline más utilizado en una dimensión.

En el Capítulo 6 se detalla el proceso a seguir para obtener una posible interpolación por Splines cúbicos en 2 dimensiones.

Por último, el Capítulo 7 está destinado al proyecto de programación realizado en lenguaje C. En este se construye, a partir de una malla de puntos, una superficie gracias a la interpolación por Splines mencionada en el Capítulo 6.

2. Interpolación

El estudio de las curvas y superficies data de la época de los primeros ordenadores. Se intentaba simular los fenómenos físicos reales con la mayor perfección posible. Pero de lo que menos se encuentra en el mundo real son las líneas rectas, tanto la forma como las trayectorias obedecen a líneas y superficies curvas.

El mundo se guía en tres dimensiones, puntos en el espacio con coordenadas, curvas, superficies y cuerpos. Buscamos un método eficiente que nos ayude a controlar y estudiar estos cuerpos, para poder usarlos y poder reproducirlos.

Lo que nosotros queremos es obtener un método fiable para obtener superficies a partir de datos tomados experimentalmente. Un método de estimación y aproximación a partir de datos medidos.

Empezaremos en una dimensión, por su simplicidad, para luego dar el salto a dos dimensiones. Aún así, hay que tener en cuenta que se podría generalizar a una dimensión n cualquiera.

Antes de nada, es importante diferenciar entre regresión e interpolación:

- La regresión es el ajuste de los datos a una función. Por ejemplo, los datos experimentales de un movimiento rectilíneo uniforme se ajustan a una linea recta ya que la ecuación que describe un movimiento rectilíneo uniforme es $x = x_0 + vt$.
- La interpolación es la búsqueda de una función que pasa por todos los puntos deseados.

Lo que nosotros estamos buscando es la certeza de que la regla encontrada cumpla los puntos de muestra de partida, por lo que aplicaremos un método de interpolación en el proyecto.

Se pueden diferenciar los métodos de interpolación en globales y locales:

- Los métodos globales, utilizan toda la muestra para estimar el valor en cada nuevo punto, con lo que añadir un punto nuevo implica la modificación de los resultados obtenidos anteriormente.
- Los métodos locales, utilizan solo los puntos de muestreo más cercano, pudiendo añadir nuevos puntos sin implicar la completa modificación de los resultados obtenidos.

Hay que decir que los métodos locales son los más deseados en el estudio real, ya que los pequeños cambios acostumbran a modificar pequeñas cosas. Usando métodos globales, un pequeño cambio provoca que toda la solución se vea afectada. Pero esta característica no se puede elegir, viene dada por los datos de partida y el método de interpolación usado.

2.1. Problema general de interpolación

En ocasiones se plantea el problema en el que se conoce una tabla de valores de una función desconocida o difícil de manejar, y nos interesaría sustituirla por otra más sencilla (por ejemplo, un polinomio) que verifique la tabla de valores. Este es el problema de interpolación polinómica que introduciremos en este tema de forma abstracta.

El problema general de interpolación se plantea de la siguiente manera:

Problema 2.1. Dados $x_0, x_1, x_2, \dots x_n \in \mathbb{R}^n$, puntos distintos, y $y_0, y_1, y_2, \dots y_n \in \mathbb{R}$, valores arbitrarios, encontrar p aplicación lineal de R^n en R tal que:

$$p(x_i) = y_i, \forall i = 0, 1, \dots n$$

Observación 2.2. Para mayor brevedad, consideraremos todo en una dimensión, $(n = 1)$, pero queda expresado que puede formularse para cualquier dimensión $n \in \mathbb{N}$.

En definitiva, la interpolación consiste en obtener una función que corresponda a una serie de datos conocidos.

Una de las clases de funciones más útiles y mejor conocidas es la de los polinomios algebraicos. Es decir, el conjunto de funciones de la forma

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots a_0$$

donde n es un entero no negativo y a_i constantes reales $\forall i$.

Una razón importante por la cual se debe considerar esta clase de funciones en la interpolación es que la derivada y la integral de un polinomio son fáciles de determinar y también son polinomios. Por ésta y otras razones más, con frecuencia se usan los polinomios para aproximar a las funciones continuas.

De todas formas, no es la única interpolación eficaz o utilizada. Además de la interpolación polinómica también existen, y son utilizadas, otro tipo de interpolación como la interpolación trigonométrica, exponencial o logarítmica entre otras. Todo depende de las circunstancias y del tipo de función que se quiera estudiar en cada momento, ya que éstas disponen de ciertas propiedades de este tipo de funciones.

Nuestro trabajo está destinado a la interpolación polinómica; así pues, siempre nos referiremos a está y las demás se dejan para otro proyecto.

2.2. Métodos usuales

Dependiendo de las condiciones que tenemos en nuestro problema, podemos optar por el uso de un tipo u otro método de resolución del problema de interpolación. Tener diferentes métodos nos da la posibilidad de elegir el que más convenga dependiendo de las circunstancias o lo que pretendemos encontrar.

Algunos ejemplos son:

Lagrange

Teorema 2.3. *Dados $n+1$ puntos (x_i, y_i), con $i = 0, 1, 2, \dots n$, de un problema de interpolación. Existe un único polinomio $P(x)$ de grado menor o igual que n tal que:*

$$P(x_i) = y_i, \forall i = 0, 1, \dots n$$

Al polinomio $P(x)$ se le llama polinomio de interpolación de Lagrange de grado n en los puntos base $x_0, \dots x_n$. Y se expresa como:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

donde $L_i(x)$ viene dado por:

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0)(x - x_1) \dots (x - x_n)}{(x_i - x_0)(x_i - x_1) \dots (x_i - x_n)}$$

Taylor

Teorema 2.4. *Sea x_0 punto de $\mathbb{R}$ y $y_0, y_1, y_2, \dots y_n$ valores reales. Entonces existe un único polinomio $P(x)$ de grado $\leq n$ tal que:*

$$P^{(i}(x_0) = y_i, \forall i = 0, 1, \dots n$$

En este caso, al polinomio $P(x)$ se le llama polinomio de interpolación de Taylor de grado n en el punto x_0. Y se expresa como:

$$P(x) = \sum_{i=0}^{n} y_i T_i(x)$$

donde $T_i(x)$ viene dado por:

$$T_i(x) = \frac{(x - x_0)^i}{i!}$$

Diferencias divididas

El método, indicado anteriormente, para calcular el polinomio de Lagrange asociado a los puntos base $x_0, x_1, x_2, \ldots x_n$ y los valores $y_0, y_1, y_2, \ldots y_n$ es complicado y obliga a rehacer todos los cálculos si se añade un nuevo punto. Es decir, no hay relación entre la construcción del polinomio $p_n(x)$ (polinomio de grado menor o igual que n que pasa por los puntos (x_i, y_i) con $i = 1, \ldots n+1$) y la del polinomio $p_{n+1}(x)$ (polinomio de grado menor o igual que $n+1$ que pasa por los puntos (x_i, y_i) con $i = 1, \ldots n+2$). Cada polinomio debe construirse individualmente y se necesitan muchas operaciones para calcular polinomios de grado elevado.

De este hecho, surge el método de las diferencias divididas; el cual presenta la ventaja de que se puede realizar de forma sencilla y evita la necesidad de rehacer todos los cálculos si se añaden nuevos puntos siguiendo un esquema recursivo.

- Diferencias divididas de orden 0:

$$f_i = y_i, i = 0, \ldots n$$

- Diferencias divididas de orden 1:

$$f_{i,i+1} = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}, i = 0, \ldots n-1$$

- Diferencias divididas de orden 2:

$$f_{i,i+1,i+2} = \frac{f_{i+1,i+2} - f_{i,i+1}}{x_{i+2} - x_i}, i = 0, \ldots n-2$$

- Diferencias divididas de orden k:

$$f_{i,i+1,\ldots i+k} = \frac{f_{i+1,\ldots i+k} - f_{i,\ldots i+k-1}}{x_{i+k} - x_i}, i = 0, \ldots n-k$$

En este caso, el polinomio interpolador $P(x)$, de grado n en los puntos base $x_0, \ldots x_n$, queda expresado de la siguiente manera:

$$P(x) = f_0 + f_{01}(x - x_0) + f_{012}(x - x_0)(x - x_1) + \ldots + f_{01\ldots n-1}(x - x_0)\ldots(x - x_{n-1})$$

Comentario 2.5. Normalmente podemos encontrarlo, para tres puntos separados a la misma distancia h, expresado de la siguiente manera:

$$\begin{array}{c|cc} x_0 & f_0 & \\ & & f_{01} = \dfrac{f_1 - f_0}{h} \\ x_1 & f_1 & \qquad f_{012} = \dfrac{\dfrac{f_2 - f_1}{h} - \dfrac{f_1 - f_0}{h}}{2h} = \dfrac{f_2 - 2f_1 + f_0}{2h^2} \\ & & f_{12} = \dfrac{f_2 - f_1}{h} \\ x_2 & f_2 & \end{array}$$

Estos métodos son los más frecuentes y utilizados en la introducción a la interpolación polinómica de métodos numéricos. No obstante, cuando el número de puntos aumenta, también aumenta el grado del polinomio; provocando así que la función sea más oscilante (lo cual se traduce en un aumento de los errores).

Un enfoque alternativo a la utilización de polinomios de grado alto es el uso de polinomios de grado menor en subintervalos. Ésta es la base de la interpolación con Splines.

2.3. Error en la interpolación

Puesto que la interpolación es un método numérico, existe un error que no podemos controlar, pero sí estudiar y ser conscientes de que existe.

Teorema 2.6. *Sea f una función de clase $C^{n+1}([a,b])$. Consideremos el problema de interpolación correspondiente a unos puntos base $a \leq x_0 < x_2 < \dots x_n \leq b$ con valores $y_0 = f(x_0)$, $y_2 = f(x_2)$, $\dots y_n = f(x_n)$, y sea p el correspondiente polinomio interpolador. Entonces para cada $x \in [a,b]$ existe un punto $\alpha \in (a,b)$ (que depende de x) que satisface:*

$$f(x) = p(x) + \frac{w(x) f^{n+1)}(\alpha)}{(n+1)!}$$

siendo $w(x) = \prod_{i=0}^{n}(x - x_i)$.

Así pues, el error de p queda escrito como:

$$|Error(x)| = |f(x) - p(x)| \leq \frac{|w(x)| M_{n+1}}{(n+1)!}$$

donde M_{n+1} es una cota de $|f^{n+1)}|$ en $[a,b]$.

Observación 2.7. En el caso particular de que los puntos base estén a la misma distancia, es decir, $x_{i+1} - x_i = x_1 - x_0 = h$ para $i = 0, 1, 2, \ldots n-1$, podemos expresar $x_i = x_0 + i * h$ para $i = 0, 1, 2, \ldots n-1$ dando lugar a:

$$|Error(x)| = |f(x) - p(x)| \leq \frac{M_{n+1}}{(n+1)!} h^{n+1} = Ch^{n+1}$$

donde M_{n+1} es una cota de $|f^{n+1)}|$ en $[x_0, x_n]$.

De la expresión anterior se deduce que cuando h tiende a cero, el error tiende también a cero a la misma velocidad que h^{n+1}, lo que se expresa como $|Error(x)| = o(h^{n+1})$.

Comentario 2.8. Se debe hacer un inciso respecto a la fórmula del error, y es que en el exterior del intervalo $[a, b]$ el valor $|w(x)|$ crece muy rápidamente. En consecuencia, el uso de la interpolación para aproximar $f(x)$ fuera de este intervalo debe evitarse. En caso de que queramos aproximar la función $f(x)$ fuera del intervalo, deberíamos usar un método de extrapolación, pero éste no es el objetivo del trabajo.

Nodos de Chebyshev

Tal y como hemos visto, el error en la interpolación depende de la función f, pero también de cómo se escojan los puntos base. Para cada valor de n, la mejor forma de elegir los puntos base es aquella para la cual el valor máximo de $|w(x)|$ sea lo más pequeño posible, es decir:

$$\text{mín}\{\text{máx}\{|(x-x_1)(x-x_2)\ldots(x-x_{n+1})| : a \leq x \leq b\} : a \leq x_1 < x_2 < \ldots < x_{n+1} \leq b\}$$

Para resolver esta cuestión podemos restringirnos, haciendo una traslación y un cambio de escala si es necesario, al intervalo $[-1, 1]$. La solución viene dada mediante los polinomios de Chebyshev $T_{n+1}(x)$, donde $T_{n+1}(x) = \cos((n+1)\arccos(x))$. Recordemos que la función $T_{n+1}(x)$ es la solución polinómica de la ecuación diferencial $(1-x^2)y'' - xy' + (n+1)^2 y = 0$ cuyo coeficiente líder es 2^n. La forma más fácil de construirlos es mediante la relación de recurrencia:

$T_0(x) = 1$

$T_1(x) = x$

$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$ con $n = 1, 2, \ldots$

Los polinomios de Chebyshev tienen muchas propiedades interesantes. La que a nosotros nos concierne es que el valor absoluto máximo de $2^{-n}T_{n+1}(x)$, para $x \in [-1, 1]$, es 2^{-n} y que, además, es el polinomio con menor máximo absoluto de entre todos los polinomios con coeficiente líder igual a 1. En otras palabras, $2^{-n}T_{n+1}(x)$ es la solución al problema anterior o, equivalentemente, que los mejores puntos base posibles son las $n + 1$ raíces simples de T_{n+1}:

$$x_i = \cos\left(\frac{(2i-1)\pi}{2(n+1)}\right), i = 1, 2, \ldots n+1$$

Estos puntos, se llaman nodos de Chebyshev del intervalo $[-1, 1]$. Como ya hemos indicado, para trabajar en un intervalo cualquiera hay que hacer un cambio de variable lineal que nos lleve el intervalo $[-1, 1]$ a nuestro intervalo de trabajo $[a, b]$:

$$x \in [-\pi, \pi] \to z(x) := \frac{b-a}{2}x + \frac{b+a}{2} \in [a, b]$$

Entonces los nodos de Chebyshev en $[a, b]$ son los puntos $z(x_i)$, con $i = 1, 2, ..n+1$.

2.4. Beneficios de los Splines

Los Splines surgen en la búsqueda de soluciones a los problemas con el resto de interpolaciones, como por ejemplo:

- En ciertos casos, con la interpolación polinomial obtenemos un polinomio de grado elevado al aumentar el número de puntos de interpolación. Esto puede desviar mucho la curva que pasa por los puntos dados, provocando problemas de estabilidad y grandes oscilaciones, huyendo de la solución deseada. Lo que se denomina fenómeno de Runge.
- Su cálculo exige realizar un gran número de iteraciones aritméticas para hallar la solución; como por ejemplo Lagrange que obliga a ejecutar 2 bucles entrelazados, uno para el sumatorio y otro para el productorio.
- Puede darse el caso que la solución encontrada sea una aproximación que pueda no pasar por los puntos requeridos, como por ejemplo la interpolación por Mínimos Cuadrados.
- Si deseamos añadir o suprimir un punto del conjunto de datos, a una solución encontrada, hay que volver a hacer todos los cálculos del método. En otras palabras, en algunos procesos de interpolación, las modificaciones locales afectan globalmente al polinomio solución.

Observación 2.9. Este último inconveniente, de muchos métodos de interpolación, también afecta a la interpolación por Splines, ya que acostumbra a tener un carácter global. No obstante, existen tipos de Spline que no presentan este problema, como los Spline de Hermite.

Comentario 2.10. El fenómeno de Runge es un problema, debido a la falta de convergencia puntual de los polinomios interpoladores con respecto a la función considerada al aumentar el número de nodos.

Los Splines, no sólo surgieron para solucionar problemas. Los Splines presentan propiedades muy beneficiosas como por ejemplo sus resultados similares requiriendo solamente del uso de polinomios de bajo grado; evitando así oscilaciones innecesarias e indeseable en la mayoría de ocasiones generadas al interpolar mediante polinomios de grado elevado, como ya hemos mencionado anteriormente. Su simplicidad de representación y la facilidad de cómputo los hacen populares para la representación de curvas en informática.

3. Splines

Los Splines son un método de interpolación que minimiza la curvatura general de la superficie a aproximar, resultando en una superficie suave que pasa exactamente por los puntos deseados.

Una función Spline está formada por varios polinomios, cada uno definido sobre un subintervalo, que se unen entre sí obedeciendo a ciertas condiciones de continuidad.

3.1. Algo de Historia

Antes de los ordenadores, las personas dibujaban curvas suaves haciendo uso de clavos en la ubicación de los puntos y la colocación de hilos o bandas de metal entre ellos. Las bandas se usaron como reglas para dibujar las deseadas curvas.

Estas bandas de metal se denominaban Splines y de aquí proviene el nombre del algoritmo de interpolación por polinomios a trozos, ya que conserva las condiciones de suavidad y adaptabilidad de esas curvas. Así pues, una función Spline será un polinomio a trozos de grado definido y con ciertas propiedades de regularidad

La teoría de los splines fue ideada en la década de los años 40 por el matemático estadounidense de origen rumano Isaac Jacob Schoenberg $(1903 - 1990)$.

Es aceptado por la comunidad que la primera referencia matemática a los Splines es el artículo 1946 de Schoenberg, que es el primer escrito donde la palabra "*spline*" es usada en la conexión y aproximación polinómica con trozos continuamente diferenciables.

Las ideas de los Splines tienen sus raíces en la industria de la construcción naval y aeronáutica. Así mismo existen otros nombres reconocidos por el estudio y el uso de este método en otros tipos de campos.

Robin Forrest describe el "$lofting$", una técnica utilizada en la industria aeronáutica británica durante la segunda guerra mundial para construir plantillas de aviones usando bandas de madera, llamadas Splines, apoyándolas en puntos fijos en el suelo, una técnica heredada del diseño de los barcos.

Durante años, el diseño de barcos usaba modelos a pequeña escala. El diseño final se dibujaba en papel y los puntos claves se volvían a dibujar en papel más grande a escala. Las reglas flexibles proporcionaban una interpolación lisa de los puntos claves. Estas se mantenían en puntos discretos y entre estos puntos implicaba aplicar formas de mínima energía de deformación.

Una posible motivación de Forrest para obtener un modelo matemático con este proceso era la pérdida potencial de los diseños para una aeronave, si la sala de diseño era arrasada por una bomba enemiga. Esto dio origen al "$conic\ lofting$" que usaba secciones cónicas para imitar la posición de la curva entre los puntos. En la década de los años 60, el "$conic\ lofting$" se cambió por el nombre de Splines con los trabajos de C. Ferguson en Boeing y M. A. Sabin en British Aircraft Corporation.

El uso de los Splines para diseñar el chasis de los automóviles tiene muchos inicios independientes: Citroën reclama por Paul de Casteljau, Renault por Pierre Bézier y General motors por Birkhoff, Garabedian y Carl R. de Boor. Todos los trabajos en la década de los años 50 y 60.

3.2. La función Spline

Como ya hemos mencionado anteriormente, los Splines es un método de interpolación. Así pues, para poder realizar esta interpolación necesitaremos valores asociados, y_i, a unos puntos base, x_i:

x_i	y_i
x_0	y_i
$\vdots$	$\vdots$
x_n	y_n

Notación 3.1. *Asumiremos siempre que tenemos $n+1$ puntos de muestra como datos, $x_0, x_1, \dots x_n$, a los que llamaremos puntos base. Para una mayor comodidad supondremos que están ordenados de forma creciente, es decir, $x_0 < x_1 < \dots x_n$. Por lo tanto, tenemos un intervalo de definición $[a, b]$, siendo $a = x_0$ y $b = x_n$.*

Notación 3.2. *Llamaremos h_i la distancia entre los puntos base x_i y x_{i+1}. Es decir: $h_i = x_{i+1} - x_i$.*

En el caso de que los puntos base estén a la misma distancia, llamaremos $h = x_{i+1} - x_i, \forall i$.

Una función Spline interpolante de grado k, con puntos base $x_0, \dots x_n$, es una función $S(X)$ formada por varios polinomios, cada uno de ellos definido sobre un intervalo y que se unen entre sí bajo ciertas condiciones de continuidad. Es decir:

Definición 3.3. *Sean $x_0, x_1, \dots, x_n$ puntos base dados.*

Se denomina función Spline de grado p, asociada a las x_i's, a una función polinómica definida a trozos, $S(x)$, tal que:

1. *Está definida en $[x_0, x_n]$.*
2. *$S_i = S_{|_{[x_i, x_{i+1}]}}$ es un polinomio de grado p, para toda $i = 0, \dots n-1$.*
3. *S es $p-1$ veces diferenciable en $[x_0, x_n]$, $C^{p-1}([x_0, x_n])$.*

Observación 3.4. Dado que tenemos $n+1$ puntos base, obtenemos n intervalos de interpolación y n funciones polinómicas.

Corolario 3.5. *Si $S(x)$ es un Spline de orden p en $[a, b]$, $S'(x)$ es un Spline de orden $p-1$ en $[a, b]$.*

Demostración. Sea $S(x)$ un Spline de orden p en $[a,b]$, entonces se cumple:

1. Existen $x_0 < x_1 < \ldots < x_n \in [a,b]$ puntos base.
2. $S(x)$ está definida en $[a,b]$.
3. $S_i = S_{|_{[x_i,x_{i+1}]}}$ es un polinomio de grado p, para toda $i = 0, \ldots n-1$.
4. S es $p-1$ veces diferenciable en $[x_0,x_n]$, $C^{p-1}([x_0,x_n])$.

Al ser S $p-1$ veces diferenciable, existe S' en $[x_0,x_n], p-2$ veces diferenciable.

Como $S(x)$ está definida en $[a,b]$ y es diferenciable en $[a,b]$, $S'(x)$ está definida en $[a,b]$.

Al ser S diferenciable en $[a,b]$ y se cumple que $S_i = S_{|_{[x_i,x_{i+1}]}}$ es un polinomio de grado p, para toda $i = 0, \ldots n-1$, tenemos que $S'_i = S'_{|_{[x_i,x_{i+1}]}}$ es un polinomio de grado $p-1$, para toda $i = 0, \ldots n-1$.

Así pues, $S'(x)$ es un Spline de orden $p-1$ en $[a,b]$. □

Notación 3.6. *Si todos los subintervalos de un Spline son de la misma distancia, el Spline se denomina uniforme. En caso contrario, se denomina no-uniforme.*

Observación 3.7. El grado de un Spline viene determinado por el grado más alto de todos los polinomios que forman el Spline, en sus intervalos de definición.

Todos los Splines de grado k quedan expresados como:

$$S(x) = \begin{cases} S_0(x) & x \in [t_0,t_1] \\ S_1(x) & x \in [t_1,t_2] \\ \vdots & \vdots \\ S_{n-1}(x) & x \in [t_{n-1},t_n] \end{cases}$$

Donde S_i es el polinomio, de grado menor o igual que k, correspondiente en cada intervalo de definición $[t_{i-1},t_i]$.

Por tal de garantizar que S es una función continua en todo el intervalo de definición, se ha de cumplir:

$$S_{i-1}(x_i) = y_i = S_i(x_i), 1 \leq i \leq n-1$$

Observación 3.8. Se puede observar que los polinomios obtenidos en un Spline de grado 0 tienen otros intervalos de definición. Los intervalos $[t_{i-1}, t_i)$ no se intersecan entre sí, por lo que no hay ambigüedad en la definición de la función en los puntos base. Es decir, el Spline de grado 0 son funciones constantes, c_i por zonas, expresándolo explícitamente:

$$S(x) = \begin{cases} S_0(x) = c_0 & x \in [t_0, t_1) \\ S_1(x) = c_1 & x \in [t_1, t_2) \\ \vdots & \vdots \\ S_{n-1}(x) = c_{n-1} & x \in [t_{n-1}, t_n) \end{cases}$$

Este tipo de Spline no son nada frecuentes y poco utilizados, ya que por lo general, no cumplen la propiedad de función continua.

Nosotros nos centraremos en los Splines de grado más bajo, pero más grande que 0. Por su simplicidad, son los más deseados y frecuentes a utilizar.

4. Splines en una variable

Los Splines de menor grado, en los que nos centramos, son los lineales, parabólicos y cúbicos. Su diferencia viene dada por el grado máximo de los polinomios que lo forman en cada intervalo y las condiciones de regularidad en los puntos internos.

4.1. Spline lineal

Nos referimos a los Splines compuestos por polinomios de grado menor o igual a 1. Por este motivo, la única condición de regularidad que podemos exigir es que las curvas se unan; es decir, continuidad de orden cero, C^0.

Definición 4.1. *Dado el conjunto de puntos base* $\Delta = \{a = x_0 < x_1 < \dots x_n = b\}$ *del intervalo* $[a, b]$*, diremos que la función* S *es un Spline lineal asociado a* Δ *si se cumplen las siguientes condiciones:*

1. *La restricción* S_i *de* S *en cada intervalo* $[x_i, x_{i+1}]$*, para* $i = 0, \dots n-1$*, es un polinomio de grado no superior a uno.*
2. $S(x) \in C^0([a, b])$*, es decir,* S *es una función continua en el intervalo [a,b].*

Definición 4.2. *Diremos que* S *es un Spline lineal interpolante para el conjunto de puntos* (x_i, y_i)*, para* $i = 0, 1, \dots n$*, si:*

- S *es un Spline lineal asociado a* Δ*.*
- $S(x_i) = y_i$ *para* $i = 0, \dots n$*.*

Características

1. S es una función lineal.
2. Continua en los $n + 1$ puntos base, C^0.
3. Los polinomios, en cada intervalo, se expresan como: $S_i(x) = a_i x + b_i$.

Antes de construir un Spline lineal vamos a ver cuantas condiciones se han de cumplir y cuantas incógnitas tendremos que encontrar.

Si en cada intervalo de Δ intentamos construir un polinomio de grado uno que aproxime a la función, deberemos calcular dos incógnitas(los dos coeficientes a_i y b_i del polinomio) por intervalo, es decir, $2n$ incógnitas.

Por otro lado, estos polinomios deben cumplir las siguientes condiciones:

- Continuidad en los $n - 1$ puntos interiores

$$S_i(x_i) = S_{i+1}(x_i) \text{ para todo } i = 1, \dots n-1$$

Es decir, se deben cumplir un total de $n-1$ condiciones además de las $n+1$ de interpolación.

Por lo tanto, tenemos $2n$ incógnitas y $(n-1)+(n+1)$ ecuaciones, lo que nos da como resultado un único Spline interpolante.

Spline parabólico

Nos referimos a los Splines compuestos por polinomios de grado menor o igual a 2. Ahora, podemos exigir la condición de regularidad de continuidad de primer orden, C^1. Es decir, las primeras derivadas (tangentes a la curva) son iguales en su punto de unión, con lo que obtenemos unas curvas que conservan las pendientes en todo el intervalo de definición.

Definición 4.3. *Dado el conjunto de puntos base $\Delta = \{a = x_0 < x_1 < \dots x_n = b\}$ del intervalo $[a, b]$, diremos que la función S es un Spline parabólico asociado a Δ si se cumplen las siguientes condiciones:*

1. *La restricción S_i de S en cada intervalo $[x_i, x_{i+1}]$, para $i = 0, \dots n-1$, es un polinomio de grado no superior a dos.*
2. *$S(x) \in C^1([a, b])$, es decir, S es una función diferenciable, con derivada continua, en el intervalo [a,b].*

Definición 4.4. *Diremos que S es un Spline parabólico interpolante para el conjunto de puntos (x_i, y_i), para $i = 0, 1, \dots n$, si:*

- *S es un Spline parabólico asociado a Δ.*
- *$S(x_i) = y_i$ para $i = 0, \dots n$.*

Características

1. S es una función parabólica.
2. Continua y diferenciable en los $n+1$ puntos base, C^1.
3. Los polinomios, en cada intervalo, se expresan como: $S_i(x) = a_i x^2 + b_i x + c_i$.

Antes de construir un Spline parabólico vamos a ver cuantas condiciones se han de cumplir y cuantas incógnitas tendremos que encontrar.

Si en cada intervalo de Δ intentamos construir un polinomio de grado dos que aproxime a la función, deberemos calcular tres incógnitas(los tres coeficientes a_i, b_i y c_i del polinomio) por intervalo, es decir, $3n$ incógnitas.

Por otro lado, estos polinomios deben cumplir las siguientes condiciones:

- Continuidad en los $n-1$ puntos interiores

$$S_i(x_i) = S_{i+1}(x_i) \text{ para todo } i = 1, \dots n-1$$

- Diferenciabilidad en los $n-1$ puntos interiores

$$S_i'(x_i) = S_{i+1}'(x_i) \text{ para todo } i = 1, \dots n-1$$

Es decir, se deben cumplir un total de $2(n-1)$ condiciones ademas de las $n+1$ de interpolación.

Por lo tanto, tenemos $3n$ incógnitas y $2(n-1)+(n+1)$ ecuaciones. Nos queda entonces una ecuación libre por determinar lo que nos daría como resultado infinitos Splines interpolantes. Eligiendo una condición adicional, ajustamos y conseguimos un único Spline.

Comentario 4.5. Lo más usado es escoger $S'(x_0) = 0$.

El siguiente Spline a estudiar es el Spline cúbico. Dado que es el Spline más utilizado, para interpolar funciones, he preferido diferenciarlo y explicarlo más detalladamente en el próximo tema.

5. Splines cúbicos

Nos referimos a los Splines compuestos por polinomios de grado menor o igual a 3. Ahora, podemos exigir la condición de regularidad de continuidad de segundo orden, C^2. Es decir, además de lo dicho anteriormente para C^1, ahora la variación de los vectores tangentes según nos acercamos al punto de unión de ambas curvas por la derecha y por la izquierda es equivalente. Con otras palabras, las segundas derivadas son iguales en su punto de unión, con lo que obtenemos una transición suave de una sección de curva a la siguiente.

Ya que podemos exigir la condición C^2, este tipo de Spline puede diferenciarse en dos tipos de Splines dependiendo de su clase C^1 o C^2.

5.1. Spline cúbico C^1 o Interpolación de Hermite

Definición 5.1. *Dado el conjunto de puntos base $\Delta = \{a = x_0 < x_1 < \dots x_n = b\}$ del intervalo $[a, b]$, diremos que la función S es un Spline cúbico C^1 asociado a Δ si se cumplen las siguientes condiciones:*

1. *La restricción S_i de S en cada intervalo $[x_i, x_{i+1}]$, para $i = 0, \dots n-1$, es un polinomio de grado no superior a tres.*
2. *$S(x) \in C^1([a, b])$, es decir, S es una función diferenciable, con derivada continua, en el intervalo [a,b].*

Definición 5.2. *Diremos que S es un Spline cúbico C^1 interpolante para el conjunto de puntos (x_i, y_i), para $i = 0, 1, \dots n$, si:*

- *S es un Spline cúbico C^1 asociado a Δ.*
- *$S(x_i) = y_i$ para $i = 0, \dots n$.*

Características

1. S es una función cúbica.
2. Continua y diferenciable en los $n + 1$ puntos base, C^1.
3. Los polinomios, en cada intervalo, se expresan como: $S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$.

Antes de construir un Spline cúbico C^1 vamos a ver cuantas condiciones se han de cumplir y cuantas incógnitas tendremos que encontrar.

Si en cada intervalo de Δ intentamos construir un polinomio de grado tres que aproxime a la función, deberemos calcular cuatro incógnitas(los cuatro coeficientes a_i, b_i, c_i y d_i del polinomio) por intervalo, es decir, $4n$ incógnitas.

Por otro lado, estos polinomios deben cumplir las siguientes condiciones:

- Continuidad en los $n-1$ puntos interiores

$$S_i(x_i) = S_{i+1}(x_i) \text{ para todo } i = 1, \dots n-1$$

- Diferenciabilidad en los $n-1$ puntos interiores

$$S_i'(x_i) = S_{i+1}'(x_i) \text{ para todo } i = 1, \dots n-1$$

Es decir, se deben cumplir un total de $2(n-1)$ condiciones además de las $n+1$ de interpolación.

Por lo tanto, tenemos $4n$ incógnitas y $2(n-1)+(n+1)$ ecuaciones, quedándonos $n+1$ ecuaciones libres. Por este motivo, si deseamos realizar este tipo de interpolación, hemos de conocer las derivadas en los puntos base $x_0, x_1, \dots x_n$. Así, añadimos $n+1$ condiciones más y obtenemos un único Spline interpolante.

En la práctica habitual se desconocen las derivadas en los puntos base, pues los datos suelen ser de tipo experimental. En estos casos hay que realizar aproximaciones de ellas.

La forma más común utilizada para aproximar estas derivadas, siendo d_i la derivada que buscamos en el punto x_i, es:

- Para los Splines no-uniformes:

$$d_0 = P_0$$

$$d_i = \frac{h_{i-1}P_i + h_iP_{i-1}}{h_i + h_{i-1}} \text{ para } i = 1, 2, \dots n-1$$

$$d_n = P_{n-1}$$

- Para los Splines uniformes, ya que la distancia entre los puntos base es igual para todos los puntos:

$$d_0 = P_0$$

$$d_i = \frac{P_{i-1} + P_i}{2} \text{ para } i = 1, 2, \dots n-1$$

$$d_n = P_{n-1}$$

Donde $P_0, P_1, \ldots P_{n-1}$ son las diferencias divididas de orden 1 para los puntos base $x_0, x_1, ..x_n$.

Comentario 5.3. El uso de este tipo de Spline como método de interpolación recibe el nombre de interpolación de Hermite, nombrada así en honor de Charle Hermite. Pero no es el único Spline que recibe nombre, también existe el Spline de Akima, aunque Hermite es el más utilizado.

5.2. Spline cúbico C^2

Definición 5.4. *Dado el conjunto de puntos base $\Delta = \{a = x_0 < x_1 < \ldots x_n = b\}$ del intervalo $[a, b]$, diremos que la función S es un Spline cúbico C^2 asociado a Δ si se cumplen las siguientes condiciones:*

1. *La restricción S_i de S en cada intervalo $[x_i, x_{i+1}]$, para $i = 0, \ldots n-1$, es un polinomio de grado no superior a tres.*
2. *$S(x) \in C^2([a, b])$, es decir, S es una función dos veces diferenciable, con las dos derivadas continuas, en el intervalo [a,b].*

Definición 5.5. *Diremos que S es un Spline cúbico C^2 interpolante para el conjunto de puntos (x_i, y_i), para $i = 0, 1, \ldots n$, si:*

- *S es un Spline cúbico C^2 asociado a Δ.*
- *$S(x_i) = y_i$ para $i = 0, \ldots n$.*

Características

1. S es una función cúbica.
2. Continua y dos veces diferenciable en los $n + 1$ puntos base, C^2.
3. Los polinomios, en cada intervalo, se expresan como: $S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$.

Antes de construir un Spline cúbico C^2 vamos a ver cuantas condiciones se han de cumplir y cuantas incógnitas tendremos que encontrar.

Si en cada intervalo de Δ intentamos construir un polinomio de grado tres que aproxime a la función, deberemos calcular cuatro incógnitas(los cuatro coeficientes a_i, b_i, c_i y d_i del polinomio) por intervalo, es decir, $4n$ incógnitas.

Por otro lado, estos polinomios deben cumplir las siguientes condiciones:

- Continuidad en los $n - 1$ puntos interiores

$$S_i(x_i) = S_{i+1}(x_i) \text{ para todo } i = 1, \ldots n-1$$

- Diferenciabilidad en los $n-1$ puntos interiores

$$S_i'(x_i) = S_{i+1}'(x_i) \text{ para todo } i = 1, \dots n-1$$

- Doble diferenciabilidad en los $n-1$ puntos interiores

$$S_i''(x_i) = S_{i+1}''(x_i) \text{ para todo } i = 1, \dots n-1$$

Es decir, se deben cumplir un total de $3(n-1)$ condiciones además de las $n+1$ de interpolación.

Por lo tanto, tenemos $4n$ incógnitas y $3(n-1)+(n+1)$ ecuaciones. Nos quedan entonces dos ecuaciones libres por determinar, lo que nos daría como resultado infinitos Splines interpolantes.

Para poder determinar de forma única el Spline cúbico C^2 de interpolación considerado, $S(x)$, se requieren dos condiciones adicionales más. Existen varias alternativas para elegir estas condiciones. Generalmente estas se fijan en los extremos, pero podrían considerarse otras opciones. Las condiciones más usuales son:

1. Escoger que la segunda derivada se anule en los extremos, es decir:

$$S''(a) = S''(b) = 0$$

   Esta es la opción más usada y obtiene el nombre de Spline cúbico natural.

2. Exigir que la derivada primera tome un determinado valor en los extremos, es decir:

$$S'(a) = y_a' \text{ y } S'(b) = y_b'$$

   donde y_a' y y_b' son valores reales prefijados, dando lugar al llamado Spline sujeto.

3. Si se da el caso que $S(a) = S(b)$, entonces podemos imponer:

$$S'(a) = S'(b) \text{ y } S''(a) = S''(b)$$

   Estas condiciones generan Splines periódicos, útiles para simular funciones con estas mismas condiciones de periodicidad.

4. Podemos suponer que es constante en los extremos, es decir:

$$S''(x_0) = S''(x_1) \text{ y } S''(x_{n-1}) = S''(x_n) = 0$$

5. Utilizar el llamado "Not-a-Knot". Se trata de usar una misma función cúbica para el primer y segundo tramo (tres primeros puntos) y/o para el último y penúltimo tramo(tres últimos puntos).

Observación 5.6. También se puede considerar Spline sujeto:

- Aquel que tiene fijadas las curvaturas en los extremos, es decir $S''(a) = y_a''$ y $S''(b) = y_b''$ donde y_a'' y y_b'' son valores reales.
- Si imponemos una pendiente y una curvatura, es decir $S'(a) = y_a'$ y $S''(b) = y_b''$ o $S''(a) = y_a''$ y $S'(b) = y_b'$.

5.3. Spline cúbico y curvatura

La principal razón por la que los Splines cúbicos son los más usados es por el hecho de que da como resultado un polinomio interpolador tal que es la función más suave que pasa por los puntos (x_i, y_i) interpolados. En este apartado, vamos a demostrar este enunciado.

Necesitaremos una definición previa de suavidad de una función y un lema que usaremos en la demostración del teorema.

Definición 5.7. *La suavidad de una función se mide con el funcional*

$$I(f) = \int_{x_0}^{x_n} [f''(x)]^2 dx$$

Lema 5.8. *Consideremos dos funciones s, $g \in C^2([a,b])$ y una partición $\Delta = \{a = x_0 < x_1 < \dots x_n\}$.*

Si s'' es derivable en casi todos sus puntos, entonces:

$$\int_a^b (g'')^2 - \int_a^b (s'')^2 = \int_a^b (g'' - s'')^2 + 2\left[s''(g' - s')|_a^b - \sum_{i=0}^{n-1} s'''(g - s)|_{x_i}^{x_{i+1}}\right]$$

Demostración. Sean s y g dos función $C^2([a,b])$, con s'' derivable en casi todos sus puntos. Entonces, se tiene que:

$$\int_a^b (g'')^2 - \int_a^b (s'')^2 = \int_a^b ((g'')^2 - (s'')^2) = \int_a^b ((g'') - (s''))^2 + 2\int_a^b s''(g'' - s'')$$

Desarrollando por partes la última integral de la expresión anterior y, usando que s'' es diferenciable, se obtiene:

$$\int_a^b s''(g'' - s'')dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} s''(g'' - s'')dx =$$

$$= \sum_{i=0}^{n-1} s''(g' - s')|_{x_i}^{x_{i+1}} - \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} s'''(x)(g' - s')dx =$$

$$= s''(g' - s')|_{x_0}^{x_n} - \sum_{i=0}^{n-1} s'''(x)(g - s)|_{x_i}^{x_{i+1}}$$

□

Teorema 5.9. *De todas las funciones C^2 que pasan por (x_i, y_i), $i = 0, 1, \dots n$, la más suave es el Spline cúbico natural.*

Por lo tanto, el Spline cúbico natural minimiza el funcional I en C^2, es decir:

$$\int_{x_0}^{x_n} [f''(x)]^2 dx \geq \int_{x_0}^{x_n} [s''(x)]^2 dx$$

para toda f función C^2 que pasa por (x_i, y_i), $\forall i$.

Y la igualdad se da si y solo si $f(x) = s(x)$ para todo $x \in [a, b]$.

Demostración. Sea f una función C^2 cualquiera y $s(x)$ el Spline cúbico natural, que pasan por (x_i, y_i), $i = 0, 1, \dots n$.

Aplicando el lema anterior, a la diferencia del funcional, obtenemos:

$$\int_{x_0}^{x_n} [f''(x)]^2 dx - \int_{x_0}^{x_n} [s''(x)]^2 dx =$$

$$= \int_{x_0}^{x_n} (f''(x) - s''(x))^2 dx + 2 \left[s''(x)(f'(x) - s'(x))|_{x_0}^{x_n} - \sum_{i=0}^{n-1} s'''(x)(f(x) - s(x))|_{x_i}^{x_{i+1}} \right]$$

Como s y f pasan por los puntos $x_0, x_1, \dots x_n$, y tienen valores $y_0, y_1, \dots y_n$ respectivamente, tenemos que $(f(x) - s(x))|_{x_i}^{x_{i+1}} = 0$, $\forall i$. Por lo tanto:

$$\int_{x_0}^{x_n} [f''(x)]^2 dx - \int_{x_0}^{x_n} [s''(x)]^2 dx = \int_{x_0}^{x_n} (f''(x) - s''(x))^2 dx + 2s''(x)(f'(x) - s'(x))|_{x_0}^{x_n}$$

Al tratar s como el Spline cúbico natural de f, se cumple que $s''(x_0) = s''(x_n) = 0$, y por tanto $s''(x)|_{x_0}^{x_n} = 0$. Con lo que obtenemos:

$$\int_{x_0}^{x_n} [f''(x)]^2 dx - \int_{x_0}^{x_n} [s''(x)]^2 dx = \int_{x_0}^{x_n} (f''(x) - s''(x))^2 dx$$

Así pues:

$$\int_{x_0}^{x_n} [f''(x)]^2 dx \geq \int_{x_0}^{x_n} [s''(x)]^2 dx$$

De la continuidad de f'' y s'' se deduce que la igualdad se da si $f''(x) - s''(x) = 0$, es decir, si sólo si $f''(x) = s''(x)$, para todo $x \in [a, b]$. En definitiva:

$$\int_{x_0}^{x_n} (f'')^2 = \int_{x_0}^{x_n} (s'')^2 \Leftrightarrow f(x) = s(x), \forall x \in [a, b]$$

□

Teorema 5.10. *Sea f función C^2 que pasan por (x_i, y_i), $i = 0, 1, \ldots n$, y $\overline{s}$ el Spline cúbico sujeto asociado a los mismos datos. Entonces:*

$$\int_{x_0}^{x_n} [f''(x)]^2 dx \geq \int_{x_0}^{x_n} [\overline{s}''(x)]^2 dx$$

Y la igualdad se da si y solo si $f(x) = \overline{s}(x)$ para todo $x \in [a, b]$.

Demostración. La demostración es análoga a la del teorema precedente, teniendo en cuenta que la condición $s''(a) = s''(b) = 0$ se debe sustituir por $s'(a) = g'(a)$ y $s'(b) = g'(b)$.

□

5.4. Algoritmo de resolución

Veamos a continuación un algoritmo que permite calcular los polinomios S_i, tranformando el problema de interpolación en la resolución de un sistema lineal cuyas incógnitas son las derivadas segundas $S_i''(x_i)$.

Como S_i es un polinomio de grado 3, S_i'' es un polinomio de grado 1. Para determinarlo, si llamamos $S_i''(x_i) = f_i''$, se tiene:

$$S_i''(x) = f_i'' \frac{x_{i+1} - x}{x_{i+1} - x_i} + f_{i+1}'' \frac{x_i - x}{x_i - x_{i+1}} = f_i'' \frac{x_{i+1} - x}{h_i} + f_{i+1}'' \frac{x - x_i}{h_i}$$

donde $h_i = x_{i+1} - x_i$.

Integrando dos veces la expresión, se tiene:

$$S_i(x) = f_i'' \frac{(x_{i+1} - x)^3}{6h_i} + f_{i+1}'' \frac{(x - x_i)^3}{6h_i} + a_i(x_{i+1} - x) + b_i(x - x_i)$$

donde a_i y b_i son las dos constantes de integración que se pueden determinar imponiendo $S_i(x_i) = f_i$. Entonces:

$$a_i = \frac{f_i}{h_i} - f_i'' \frac{h_i}{6} \text{ y } b_i = \frac{f_{i+1}}{h_i} - f_{i+1}'' \frac{h_i}{6}$$

De esta forma tenemos $S_i(x)$ expresado en función de las incógnitas f_i'' y f_{i+1}''.

Imponiendo ahora que $S_{i-1}'(x_i) = S_i'(x_i)$, para $i = 1, \ldots n-1$, se obtienen las $n - 1$ ecuaciones:

$$c_i = h_i f_{i+1}'' + 2(h_i + h_{i-1}) f_i'' + h_{i-1} f_{i-1}'' = 6 \left(\frac{f_{i+1} - f_i}{h_i} - \frac{f_i - f_{i-1}}{h_{i-1}} \right), i = 1, ..n-1$$

Se tiene entonces un sistema de $n - 1$ con $n + 1$ incógnitas: $f_0'', \ldots f_n''$.

Con el fin de obtener un sistema con el mismo número de ecuaciones que de incógnitas, se puede elegir entre las condiciones mencionadas en el apartado anterior.

Por ejemplo, en el caso del Spline natural, reducimos el número de incógnitas imponiendo los valores de $f_0'' = f_n'' = 0$. Así, en el caso del spline natural, las ecuaciones anteriores pueden escribirse en forma matricial:

$$\begin{pmatrix} 2(h_1+h_0) & h_1 & \dots & 0 \\ h_1 & 2(h_2+h_1) & \ddots & \vdots \\ \vdots & \ddots & \ddots & h_{n-2} \\ 0 & \dots & h_{n-2} & 2(h_{n-1}+h_{n-2}) \end{pmatrix} \begin{pmatrix} f_1'' \\ f_2'' \\ \vdots \\ f_{n-1}'' \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix}$$

que es un sistema con matriz tridiagonal simétrica y de diagonal estrictamente dominante (y, por tanto, invertible). Así pues, el sistema tiene solución única.

Entonces, resolviendo el sistema tridiagonal, se calculan $f_1'', \dots f_{n-1}''$. A partir de ellos se obtienen los valores de a_i y b_i, que finalmente nos permitirán construir los polinomios $S_i(x)$, con $i = 0, \dots n-1$.

Teorema 5.11. *Si f es una funcion definida en [a,b], entonces f tiene un único Spline cúbico natural interpolante, es decir, un único Spline cúbico que interpola f y que cumple S"(a)=S"(b)=0.*

Observación 5.12. Tal como hemos visto en la resolución del Spline cúbico natural, este método de interpolación no permite control local de la curva. Es decir, si se altera la posición de cualquier punto de control, afecta a la curva entera (teniendo que rehacer cálculos).

No obstante, la interpolación de Hermite, al disponer de una tangente específica en cada punto de control, sí que permiten tener control local, por eso suele ser tan utilizada.

6. Splines en dos variables

Cuando nos referimos a una función en dos variables a $\mathbb{R}$, nos estamos refiriendo a funcion que parametriza una superficies. Los Splines permiten representaciones matemáticas de superficies partiendo de información relativa a algunos de sus puntos. Su construcción consiste en obtener una función de interpolación que pase por esos puntos, generando así, en este caso, una aproximación de la superficie deseada.

En este capítulo daremos un método, basado en los Splines bicúbicos, (Splines cúbicos en dos variables), para aproximar una superficie. Usaremos las derivadas direccionales y cruzadas de esta superficie en los puntos base, y de esta manera obtendremos una superficie interpoladora con la suavidad deseada.

La construcción del método se basa en los Splines cúbicos explicados anteriormente, escalándolos a dos dimensiones.

6.1. Spline bicúbico

Buscamos una función de dos variables que se corresponda a un polinomio cúbico en cada una de ellas, es decir polinomios de grado máximo 3.

Notación 6.1. *Llamaremos cuadrado unidad al cuadrado con vértices de coordenadas:* $(0,0)$*,* $(1,0)$*,* $(0,1)$ *y* $(1,1)$*.*

Supongamos que los valores de la función y sus derivadas, primeras y cruzadas, son conocidos en las cuatro esquinas del cuadrado unidad. Ahora la superficie, en el cuadrado unidad, queda interpolada por el polinomio:

$$S(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j \tag{6.1}$$

O en forma matricial:

$$S(x,y) = \begin{pmatrix} 1 & x & x^2 & x^3 \end{pmatrix} \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} 1 \\ y \\ y^2 \\ y^3 \end{pmatrix}$$

Sabiendo la forma que tiene nuestro polinomio, podemos calcular las derivadas primeras y cruzadas de (6.1), a las cuales impondremos que verifiquen los valores en los puntos dados de la función a interpolar:

$$S_x(x,y) = \sum_{i=1}^{3}\sum_{j=0}^{3} a_{ij} i x^{i-1} y^j$$

$$S_y(x,y) = \sum_{i=0}^{3}\sum_{j=1}^{3} a_{ij} j x^i y^{j-1}$$

$$S_{xy}(x,y) = \sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij} ij x^{i-1} y^{j-1}$$

Por lo tanto, realizar esta interpolación consiste en determinar los 16 coeficientes, a_{ij}, de la ecuación (6.1) a partir de los valores conocidos, resolviendo un sistema de 16 ecuaciones con 16 incógnitas.

Estas ecuaciones son:

1. $f(0,0) = S(0,0) = a_{00}$
2. $f(1,0) = S(1,0) = a_{00} + a_{10} + a_{20} + a_{30} = \sum_{i=0}^{3} a_{i0}$
3. $f(0,1) = S(0,1) = a_{00} + a_{01} + a_{02} + a_{03} = \sum_{j=0}^{3} a_{0j}$
4. $f(1,1) = S(1,1) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}$
5. $f_x(0,0) = S_x(0,0) = a_{10}$
6. $f_x(1,0) = S_x(1,0) = a_{10} + 2a_{20} + 3a_{30} = \sum_{i=1}^{3} i a_{i0}$
7. $f_x(0,1) = S_x(0,1) = a_{10} + a_{11} + a_{12} + a_{13} = \sum_{j=0}^{3} a_{1j}$
8. $f_x(1,1) = S_x(1,1) = \sum_{i=1}^{3}\sum_{j=0}^{3} i a_{ij}$
9. $f_y(0,0) = S_y(0,0) = a_{01}$
10. $f_y(1,0) = S_y(1,0) = a_{01} + a_{11} + a_{21} + a_{31} = \sum_{i=0}^{3} a_{i1}$
11. $f_y(0,1) = S_y(0,1) = a_{01} + 2a_{02} + 3a_{03} = \sum_{j=1}^{3} j a_{0j}$
12. $f_y(1,1) = S_y(1,1) = \sum_{i=0}^{3}\sum_{j=1}^{3} j a_{ij}$
13. $f_{xy}(0,0) = S_{xy}(0,0) = a_{11}$
14. $f_{xy}(1,0) = S_{xy}(1,0) = a_{11} + 2a_{21} + 3a_{31} = \sum_{i=1}^{3} i a_{i1}$
15. $f_{xy}(0,1) = S_{xy}(0,1) = a_{11} + 2a_{12} + 3a_{13} = \sum_{j=1}^{3} a_{1j}$
16. $f_{xy}(1,1) = S_{xy}(1,1) = \sum_{i=1}^{3}\sum_{j=1}^{3} ij a_{ij}$

Podemos expresar el sistema de forma matricial como $Ax = p$, con:

$$x = \begin{pmatrix} a_{00} & a_{10} & a_{20} & a_{30} & a_{01} & a_{11} & a_{21} & a_{31} & a_{02} & a_{12} & a_{22} & a_{32} & a_{03} & a_{13} & a_{23} & a_{33} \end{pmatrix}^T,$$

$$p = \begin{pmatrix} f(0,0) \\ f(1,0) \\ f(0,1) \\ f(1,1) \\ f_x(0,0) \\ f_x(1,0) \\ f_x(0,1) \\ f_x(1,1) \\ f_y(0,0) \\ f_y(1,0) \\ f_y(0,1) \\ f_y(1,1) \\ f_{xy}(0,0) \\ f_{xy}(1,0) \\ f_{xy}(0,1) \\ f_{xy}(1,1) \end{pmatrix} \text{ y } A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 2 & 4 & 6 & 0 & 3 & 6 & 9 \end{pmatrix}$$

De aquí podemos comprobar si la matriz A es invertible, y obtenemos:

$$A^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\ 9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\ -6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\ 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ -6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\ 4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Por lo tanto, al tener A inversa, el sistema es un sistema compatible determinado con una única solución. Con lo que este método nos proporciona una única superficie

interpoladora de la superficie deseada con las condiciones de suavidad y continuidad requeridas.

Utilizando ahora $A^{-1}p = x$ obtenemos, para cuales sean los valores de f, f_x, f_y y f_{xy} de partida, los coeficientes a_{ij} de la función interpoladora $S(x, y)$ que buscamos en el cuadrado unidad.

Observación 6.2. Tal como hemos visto en la construcción del método, se usan todos los valores y se ven involucrados en todos los cálculos; con lo cual, esta construcción genera un método global en el cuadrado unidad. Por lo que un cambio en uno de los datos de partida, genera un cambio en la superficie resultante.

6.2. El cuadrado unidad

Elegimos el cuadrado unidad como malla de puntos base por su simplicidad y la correspondencia de datos e incógnitas. Es decir, al buscar un polinomio de 16 coeficientes, el cuadrado unidad, al tener cuatro esquinas, nos proporciona la posibilidad de imponer las derivadas direccionales y cruzadas que queremos para conseguir la suavidad en la superficie y así obtener un sistema de 16 ecuaciones con 16 incógnitas que tiene solución única.

Por otro lado, si quisiéramos utilizar otro tipo de malla, que no sea cuadrada, como en triangulo, pentágono o hexágono, no podríamos utilizar las mismas condiciones que usamos ahora mismo en el método.

Por ejemplo, la malla triangular nos da tres veces las condiciones elegidas, es decir, si queremos imponer las derivadas direccionales y cruzadas, estas nos darán 12 ecuaciones. Esto provoca que tengamos que eliminar términos de nuestra función para obtener una única solución; y en consecuencia, no podemos tener una función de la forma (6.1).

Este hecho afecta también a los demás tipos de mallas, y es que la selección de ésta influye en cierta manera en la función solución que da el método.

Otro motivo, y el principal, por el que hemos escogido esta malla es la comodidad que tenemos al poder medir los puntos deseados en esta forma. El trabajo está destinado a un estudio real con datos experimentales y la forma más simple de obtener estos es en forma de cuadrícula. Sería un gran inconveniente tener que buscar con exactitud una malla triangular o otro tipo de malla de puntos.

El método empleado se centra en el cuadrado unidad, pero, como sucede en los Splines de una dimensión, tenemos más intervalos de definición de la superficie. Para poder calcular el resto de cuadrados del plano efectuamos una traslación y un cambio de escala, si es necesario, de manera que trabajemos en el intervalo $[0, 1]$. De esta manera podemos calcular el Spline de cualquier cuadrado del plano.

Por lo tanto, podemos buscar una función Spline que interpole una superficie con puntos base en una malla de puntos $[a, b]x[c, d]$, es decir $a = x_0 < x_1 < \ldots x_n = b$ y $c = y_0 < y_1 < \ldots y_m = d$, obteniendo así nxm funciones polinómicas diferentes, una para cada cuadrado.

Como hemos dicho anteriormente, el método es un método global, pero solo en el cuadrado de definición. Es decir, los datos de puntos que no pertenecen a un cuadrado, no afectan al Spline generado en éste; siendo por tanto, un método local para cuadrados.

6.3. Derivadas

Para encontrar el Spline que aproxime los datos tomados, con la suavidad necesaria, necesitamos las derivadas direccionales y cruzadas de la función. Por su simplicidad y construcción, considero que son las mínimas condiciones que se deben cumplir para obtener la función deseada.

El uso de una derivada de orden mayor podría causar una diferencia de orden en los cálculos que no deseamos. Por eso mismo nos centramos exclusivamente en las direccionales y cruzadas.

En la construcción del método, hemos partido de la suposición de que conocemos las derivadas parciales respecto x, y y cruzada xy, las cuáles no siempre se pueden calcular y en nuestro estudio práctico no disponemos de ellas. Por ello necesitamos encontrar una aproximación para poder realizar el método.

Una de las mejores formas de conseguir las derivadas direccionales sería calculando el Spline cúbico en cada una de las direcciones en una dimensión. Con esto conseguiríamos la mejor aproximación con la suavidad deseada al obtenerlas derivando el Spline C^2 obtenido.

Existe un inconveniente, por el cual nosotros no seguiremos este método. Pese a su buena aproximación, su uso implica que cada uno de los Splines de dimensión 1 calculados estarían asociados a los x_i, o y_i, dependiendo de la dirección tomada, provocando que la superficie encontrada dependa de todos los puntos de la malla. Por lo cual, si existe cualquier modificación en la superficie que altere alguno de los puntos de la malla, implicaría la necesidad de volver a realizar todos los cálculos anteriores convirtiendo la solución en una solución global.

Esto supone un gran inconveniente, ya que perderíamos la propiedad local de los cuadrados por conseguir una buena aproximación de las derivadas.

Además, este hecho provoca que un posible cambio en un punto de la malla, haga que el resultado final en todos los otros puntos sean modificados. Un gran problema, ya que en geografía puede darse el caso, que en un cierto t_1, la superficie de un terreno en el tiempo t_0 se vea modificada parcialmente y que no necesariamente afecte a todo su alrededor. Es decir, modificaríamos toda la superficie hallada anteriormente, siendo contradictorio con el hecho de que la modificación se ha producido de forma local.

Por estos motivos, usaremos otra forma de aproximar las derivadas. De tal manera que aunque perdemos precisión en el método, conservamos la propiedad de método local que tanto interesa.

En funciones de una variable tenemos el método para obtener una aproximación de las derivadas usando 3 puntos equidistantes de apoyo, es decir, tenemos 3 puntos x_0, x_1 y x_2 tales que $x_0 < x_1 < x_2$ y a la misma distancia h. Entonces, la derivada de la función en cada uno de estos puntos viene determinada por:

1. Si $x^* = x_0$, fórmula en adelanto con 3 puntos:

$$f'(x^*) \approx \frac{-f(x^* + 2h) + 4f(x^* + h) - 3f(x^*)}{2h}$$

2. Si $x^* = x_1$, fórmula centrada con 3 puntos:

$$f'(x^*) \approx \frac{f(x^* + h) - f(x^* - h)}{2h}$$

3. Si $x^* = x_2$, fórmula en retroceso con 3 puntos:

$$f'(x^*) \approx \frac{3f(x^*) - 4f(x^* - h) + 3f(x^* - 2h)}{2h}$$

Extenderemos estas fórmulas de una variable a dos variables para obtener una aproximación de las derivadas a partir de los datos que tenemos. Puesto que nosotros tratamos funciones con dos variables, tenemos puntos (x_i, y_j) con $i, j = 0, 1, 2$ tales que $x_0 < x_1 < x_2$ e $y_0 < y_1 < y_2$. Luego las fórmulas se ven modificadas de esta manera:

- Si queremos la derivada parcial respecto la primera variable, x:

  1. Si $x^* = x_0$, fórmula en adelanto con 3 puntos:

$$f_x(x^*, y_i) \approx \frac{-f(x^* + 2h, y_i) + 4f(x^* + h, y_i) - 3f(x^*, y_i)}{2h}$$

  2. Si $x^* = x_1$, fórmula centrada con 3 puntos:

$$f_x(x^*, y_i) \approx \frac{f(x^* + h, y_i) - f(x^* - h, y_i)}{2h}$$

  3. Si $x^* = x_2$, fórmula en retroceso con 3 puntos:

$$f_x(x^*, y_i) \approx \frac{3f(x^*, y_i) - 4f(x^* - h, y_i) + 3f(x^* - 2h, y_i)}{2h}$$

- Si queremos la derivada parcial respecto la segunda variable, y:

  1. Si $y^* = y_0$, fórmula en adelanto con 3 puntos:

$$f_y(x_i, y^*) \approx \frac{-f(x_i, y^* + 2h) + 4f(x_i, y^* + h) - 3f(x_i, y^*)}{2h}$$

2. Si $y^* = y_1$, fórmula centrada con 3 puntos:

$$f_y(x_i, y^*) \approx \frac{f(x_i, y^* + h) - f(x_i, y^* - h)}{2h}$$

3. Si $y^* = y_2$, fórmula en retroceso con 3 puntos:

$$f_y(x_i, y^*) \approx \frac{3f(x_i, y^*) - 4f(x_i, y^* - h) + 3f(x_i, y^* - 2h)}{2h}$$

· Si queremos la derivada cruzada:

1. Si $x^* = x_0$ e $y^* = y_0$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} -[-f(x^* + 2h, y^* + 2h) + 4f(x^* + 2h, y^* + h) - 3f(x^* + 2h, y^*)] + \\ +4[-f(x^* + h, y^* + 2h) + 4f(x^* + h, y^* + h) - 3f(x^* + h, y^*)] - \\ -3[-f(x^*, y^* + 2h) + 4f(x^*, y^* + h) - 3f(x^*, y^*)] \end{pmatrix}$$

2. Si $x^* = x_1$ e $y^* = y_0$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} [-f(x^* + h, y^* + 2h) + 4f(x^* + h, y^* + h) - 3f(x^* + h, y^*)] - \\ -[-f(x^* - h, y^* + 2h) + 4f(x^* - h, y^* + h) - 3f(x^* - h, y^*)] \end{pmatrix}$$

3. Si $x^* = x_0$ e $y^* = y_1$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} [-f(x^* + 2h, y^* + h) + 4f(x^* + h, y^* + h) - 3f(x^*, y^* + h)] - \\ -[-f(x^* + 2h, y^* - h) + 4f(x^* + h, y^* - h) - 3f(x^*, y^* - h)] \end{pmatrix}$$

4. Si $x^* = x_0$ e $y^* = y_2$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} -[3f(x^* + 2h, y^*) - 4f(x^* + 2h, y^* - h) + 3f(x^* + 2h, y^* - 2h)] + \\ +4[3f(x^* + h, y^*) - 4f(x^* + h, y^* - h) + 3f(x^* + h, y^* - 2h)] - \\ -3[3f(x^*, y^*) - 4f(x^*, y^* - h) + 3f(x^*, y^* - 2h)] \end{pmatrix}$$

5. Si $x^* = x_2$ e $y^* = y_0$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} -[3f(x^*, y^* + 2h) - 4f(x^* - h, y^* + 2h) + 3f(x^* - 2h, y^* + 2h)] + \\ +4[3f(x^*, y^* + h) - 4f(x^* - h, y^* + h) + 3f(x^* - 2h, y^* + h)] - \\ -3[3f(x^*, y^*) - 4f(x^* - h, y^*) + 3f(x^* - 2h, y^*)] \end{pmatrix}$$

6. Si $x^* = x_1$ e $y^* = y_1$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} [f(x^* + h, y^* + h) - f(x^* + h, y^* - h)] - \\ -[f(x^* - h, y^* + h) - f(x^* - h, y^* - h)] \end{pmatrix}$$

7. Si $x^* = x_1$ e $y^* = y_2$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} [3f(x^* + h, y^*) - 4f(x^* + h, y^* - h) + 3f(x^* + h, y^* - 2h)] - \\ -[3f(x^* - h, y^*) - 4f(x^* - h, y^* - h) + 3f(x^* - h, y^* - 2h)] \end{pmatrix}$$

8. Si $x^* = x_2$ e $y^* = y_1$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} [3f(x^*, y^* + h) - 4f(x^* - h, y^* + h) + 3f(x^* - 2h, y^* + h)] - \\ -[3f(x^*, y^* - h) - 4f(x^* - h, y^* - h) + 3f(x^* - 2h, y^* - h)] \end{pmatrix}$$

9. Si $x^* = x_2$ e $y^* = y_2$:

$$f_{xy}(x^*, y^*) \approx \frac{1}{4h^2} \begin{pmatrix} 3[3f(x^*, y^*) - 4f(x^*, y^* - h) + 3f(x^*, y^* - 2h)] - \\ -4[3f(x^* - h, y^*) - 4f(x^* - h, y^* - h) + 3f(x^* - h, y^* - 2h)] + \\ +3[3f(x^* - 2h, y^*) - 4f(x^* - 2h, y^* - h) + 3f(x^* - 2h, y^* - 2h)] \end{pmatrix}$$

Siendo f_x, f_y, f_{xy} la derivada parcial respecto de x, la derivada parcial respecto de y y la derivada cruzada respectivamente.

Observación 6.3. Tanto en las fórmulas de una variable como en la de dos, son de orden $o(h^2)$ y $o(h^2, h^2)$ respectivamente. Por lo tanto, no perdemos precisión ya que todas tienen la misma aproximación. Tal y como buscábamos cuando elegimos la cuadrícula y las derivadas que queríamos imponer.

Realizando ésta aproximación tenemos las derivadas en los extremos de nuestro intervalo de definición de la malla, lo que implica que podemos calcular todos los puntos de ésta tal y cómo queríamos.

7. Curvas de nivel

Este capítulo está destinado a un estudio real, a partir del método expuesto en el tema anterior.

Para poder llevar a cabo todos los cálculos del método Spline, he creado unas funciones en C al cual le pasamos los datos y nos devuelve la superficie interpolada y una cantidad de curvas de nivel de éstas.

Para el estudio experimental, tomamos dos muestras de datos, uno relativo a una zona de montaña, que corresponde al municipio de Macael, y otro a una zona costera, que corresponde al municipio de Roquetas de mar. Así podemos tener distintos puntos de vista del método empleado en regiones muy diferentes.

Los datos que hemos recibido son coordenadas (x, y, z) tales que están separadas en los ejes X e Y por una distancia de 10 metros, por lo que el programa trabaja con Splines uniformes (intervalos de misma distancia). Las distancias están medidas todas en metros y centradas en un origen que desconocemos: las coordenadas x e y tienen una precision de ± 10 metros, mientras que las alturas vienen dadas con decimales.

Por éste motivo tenemos una tolerancia de $1.e-3$, controlando así, sin perdida de precisión, todos los cálculos realizados dentro del programa.

Los métodos para obtener los Splines son los ya mencionados en el tema anterior, pero también hacemos uso de otras funciones dentro del programa.

La intención inicial, para obtener las curvas de nivel, era usar continuación a partir de una altura inicial estimada. Esta idea quedó descartada al terminar la función de búsqueda de la altura, ya que al realizar esta búsqueda ejecutábamos demasiados cálculos y sólo llegamos a obtener un único punto para cada altura diferente. Es decir, gastábamos muchos recursos en obtener un único punto.

De todas formas, este método permanece en el programa, aunque no se le da uso.

El principal inconveniente que hemos encontrado a la continuación es el hecho de la posibilidad de que existan más de una curva de nivel a la misma altura. Nuestro método encuentra la primera coordenada donde el valor de la altura es el deseado, y al efectuar continuación se genera la curva de nivel. El problema viene dado si existe otra curva de nivel a esa altura, ya que no tenemos forma de controlar si la obtenida anteriormente y la nueva son la misma.

Por esto, y debido a que ya realizábamos un peinado por toda la malla, decidimos efectuar el método por fuerza bruta evaluando valores directamente en los Splines calculados. Al realizar esta batida, evaluamos todas las alturas a la vez y guardamos en el fichero los puntos que nos interesan, ahorrándonos así tener que realizar sucesivas búsqueda de la altura para realizar continuación.

La manera que usamos para conseguir aproximaciones de las alturas es a partir del método de Newton para hallar raíces de $f(x)$. Es decir:

Comenzando en x_0, una aproximación inicial, iteramos hasta una cierta tolerancia que nos indica haber encontrado la altura deseada a.

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$$

Por tanto $x_n = a$ para un n lo suficientemente grande.

Comentario 7.1. Utilizamos Newton en una variable ya que vamos peinando toda la malla primero en el eje X y luego subimos por el eje Y, es decir, damos valores a las $y's$ y vamos efectuando Newton para las $x's$ de intervalo de definición.

Observación 7.2. Puede darse el caso que el método nos cambie de intervalo de definición, lo que provoca que se cambia de función Spline de forma correcta para calcular el siguiente punto.

Por problemas de memoria con los datos, solo se pueden generar superficies de intervalos reducidos y no la superficie total. Debido a la gran cantidad de datos que se pueden llegar a generar $10^{(n-1)}x10^{(m-1)} = 10^{n+m-2}$, acotamos nuestra superficie a una malla más reducida. Realizando algunas pruebas, se ha determinado que el número máximo de coordenadas diferentes que puede contener en una malla, ya sea para las x como para las y, es de 50 y, puesto que mínimo se necesitan dos coordenadas distintas para poder hacer un lado de un cuadrado, el mínimo es 2. Este es el número máximo que ha de contener para que funcione lo más fluido posible con una cantidad coherente de datos a estudiar.

No obstante, el programa funciona para cualquier n. Es la cantidad de datos de salida los que causan errores ajenos al programa y provocan que el sistema colapse.

7.1. Datos de entrada y salida del programa

El fichero de entrada ha de contener una primera fila con la distancia entre los puntos base (que han de estar todos a la misma distancia), el número de coordenadas y diferentes y el número de coordenadas x diferentes. Esto es necesario para una buena lectura y poder realizar labores como guardar memoria dinámica dentro del programa.

Comentario 7.3. Esta primera línea no venía en el fichero inicial de datos que me proporcionó la doctora Juani, tuve que obtenerlos usando una pequeña función que contaba todos los puntos del fichero y buscaba máximos y mínimos. Puesto que considero que estos datos podrían venir ya en el fichero inicial y la función era muy simple, no ha quedado reflejada en el trabajo.

Los datos nos vienen dados como crecientes en el eje de las x y decrecientes en el eje de las y, algo que se ha tenido en cuenta a la hora de guardar los datos dentro del programa, por lo tanto, si los datos no se introducen de esta manera puede que hayan conflictos de lectura y por lo tanto errores o resultados no deseados.

Durante la ejecución del programa, se pedirá desde la consola que se introduzcan:

- Un número de puntos para generar una malla de puntos que los contenga.
- Unas coordenadas x, y, esquina inferior izquierda, para que tome referencia y genere la malla.

Durante la ejecución, se genera un fichero $Superficie.txt$ que contiene los puntos para poder dibujar la superficie, en la malla especificada al inicio del programa, y un fichero $Curvas.txt$ que contiene los puntos para poder dibujar 11 curvas de nivel de la superficie obtenida.

Una vez se ha ejecutado el programa, y tenemos en los ficheros de salida las coordenadas de la superficie y de las curvas de nivel, mediante gnuplot, dibujamos la superficie y las curvas encontradas a partir de los siguientes comandos:

Para la superficie:

- set xlabel "X"
- set ylabel "Y"
- set pm3d
- splot "Superficie.txt"lt 0 w l

Observación 7.4. Si deseamos que el dibujo sea más simple y nítido, cambiaremos la última comanda por:

- splot "Superficie.txt"lt 0

Para las curvas de nivel:

- set xlabel "X"
- set ylabel "Y"
- splot "Curvas.txt"lt 0

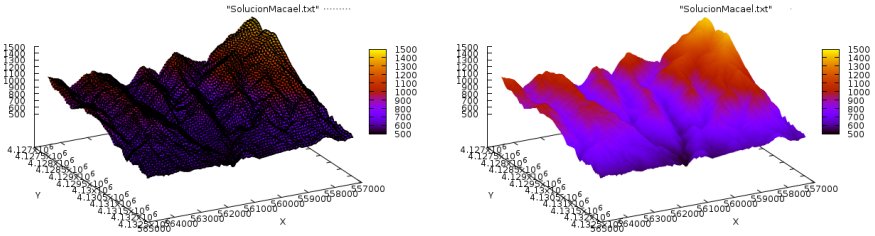
7.2. Resultados del experimento

Una vez acabado el programa y las pruebas correspondientes, se ha ejecutado con los valores experimentales recogidos dando como resultado las siguientes superficies y curvas de nivel.

Me hubiese gustado poder generar todo el mapa de la, pero debido al problema mencionado anteriormente, de la memoria, solo lo podemos hacer por secciones. En los siguientes ejemplos considero siempre mallas de 50x50, que son las más grandes posibles y las que nos dan la posibilidad de ver mas deformaciones en las superficies.

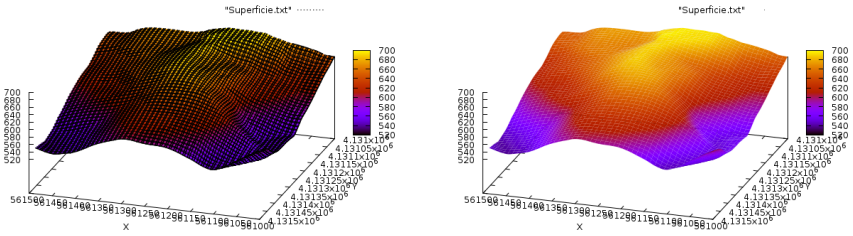
Comentario 7.5. Para facilitar la visualización de las imagenes, incorporo dos, una con la malla y otra únicamente la superficie.

Macael, viene dada en los intervalos $[557000 : 565000][4127000 : 4132500][500 : 1500]$ y queda interpolada por la siguiente superficie:

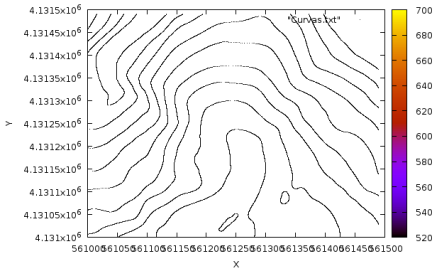


Realizando pruebas en diferentes mallas obtenemos las siguientes superficies:

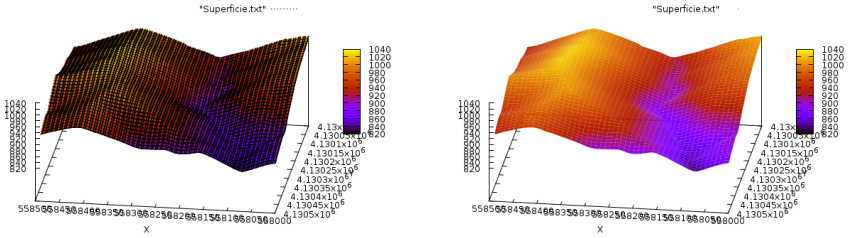
- Para 50 puntos y marcando el origen en el $(561000, 4131000)$ obtenemos la siguiente superficie:



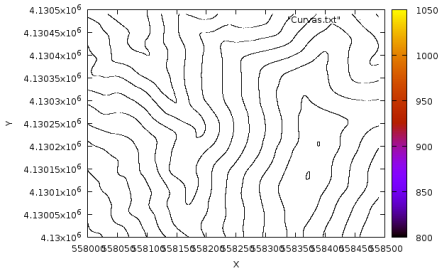
Y marcando las 11 curvas de nivel queda el mapa:



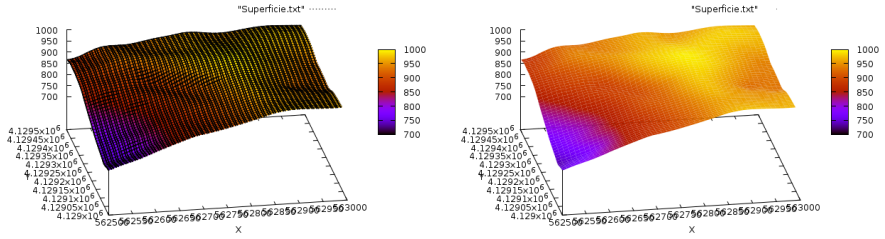
- Para 50 puntos y marcando el origen en el $(558000, 4130000)$ obtenemos la siguiente superficie:



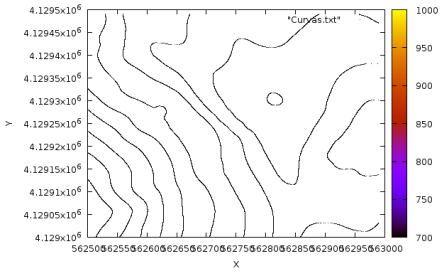
Y marcando las 11 curvas de nivel queda el mapa:



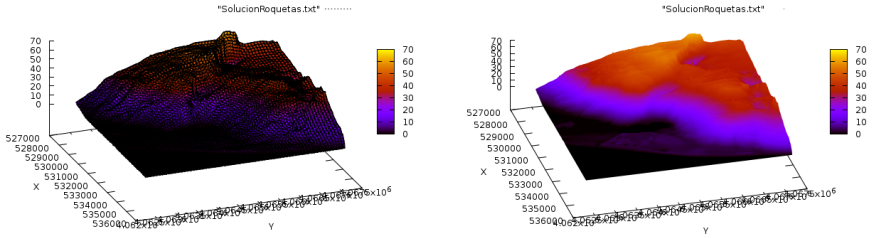
- Para 50 puntos y marcando el origen en el $(562500, 4129000)$ obtenemos la siguiente superficie:



Y marcando las 11 curvas de nivel queda el mapa:

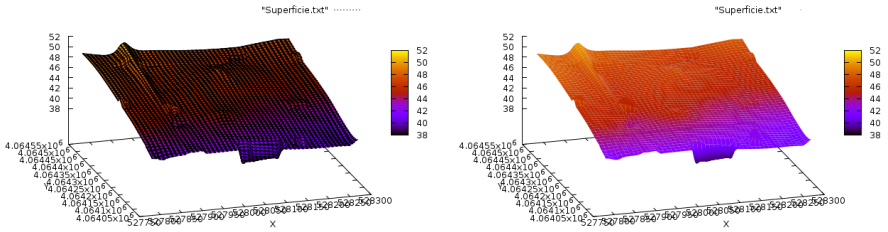


Roquetas de mar, viene dada en los intervalos $[527000 : 536000][4062000 : 4067500][0 : 70]$ y queda interpolada por la siguiente superficie:

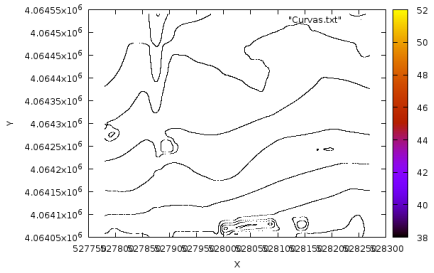


Realizando pruebas en diferentes mallas obtenemos las siguientes superficies:

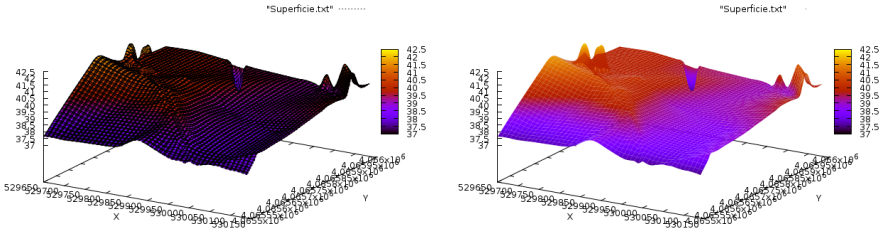
- Para 50 puntos y marcando el origen en el $(527780, 4064050)$ obtenemos la siguiente superficie:



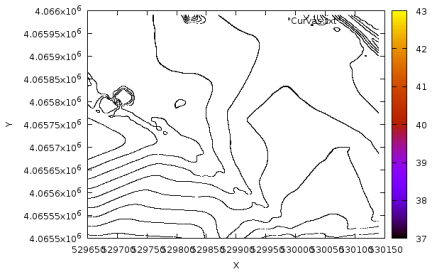
Y marcando las 11 curvas de nivel queda el mapa:



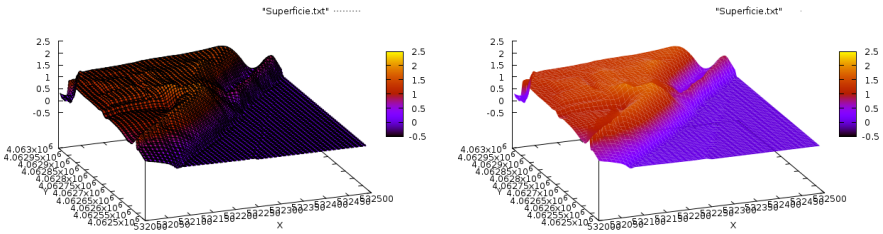
- Para 50 puntos y marcando el origen en el $(529650, 4065500)$ obtenemos la siguiente superficie:



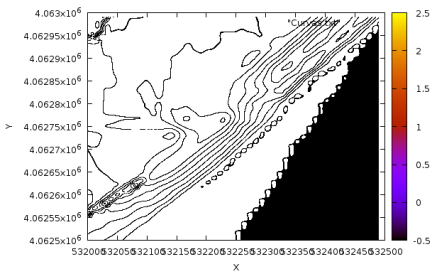
Y marcando las 11 curvas de nivel queda el mapa:



- Para 50 puntos y marcando el origen en el $(532000, 4062500)$ obtenemos la siguiente superficie:



Y marcando las 11 curvas de nivel queda el mapa:



8. Conclusiones

Realizando éste proyecto me he dado cuenta de la complejidad y cantidad de métodos que existen de interpolación y del uso importante que han tenido y tienen actualmente en computación.

He podido comprobar que queda mucha teoría en la que profundizar y muchos métodos que estudiar tales como las curvas de Bézier, los NURBS, B-Splines, entre otros.

Y que la interpolación es un método utilizado en muchos aspectos de la vida cotidiana, como el diseño tanto de carrocerías para automóviles, como de cascos para embarcaciones hasta llegar a zapatillas deportivas de competición. Y en campos como la economía, procesamiento de señales e imágenes, meteorología y mas.

He podido profundizar en un tema que me intereso desde que lo estudie en métodos númericos y ha sido una experiencia increíble haber generado un método con unas hipótesis, obtenido unas conclusiones y poder comprobar que todo funciona correctamente.

Lo realmente interesante ha sido la búsqueda de las superficies a partir de los datos experimentales. A partir de las fórmulas encontradas, poder recrear las superficies gracias al programa. Sin duda ha sido una satisfacción poder visualizar las superficies y comprobar que el método planteado funcionaba en un estudio real.

Referencias

[1] Juan Manuel Cordero Valle, José Cortés Parejo, *Curvas y Superficies para Modelado Geométrico*, RA-MA, 2002.

[2] Marco Paluszny, Hartmut Prautzsch and Wolfgang Boehm, *Méstodos de Béizer y B-Splines*, Springer Verlag Berlin Heidelberg, 2002.

[3] H. Bartels Richard, Jhon C. Beatty and A. Brian Barsky Riesel, *An introduction to Splines for use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann Publisher, 1987.

[4] C. De Boor, *A Practical Guide to Splines*, New York: Springer-Verlag, 1978.

[5] G.E. Forsythe, M.A. Malcolm, and C.B. Moler, *Computer Methos for Mathematical Computations*, Englewood Cliffs, NJ: Prentice-Hall, 1977.

[6] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, New York: Springer-Verlag, 1980.

[7] A. Ralston and P. Rabinowitz, *A First Course in Numerical Analysis*, 2nd ed., New York: McGraw-Hill, 1978.

[8] G. Hämmerlin and K.-H. Hoffmann, *Numerical Mathmatics*, New York: Springer Verlag, 1991.

[9] P. Lancaster and K. Salkauskas, *Curve and Surface Fitting. An Inroduction*, London: Academic Press, 1986.

[10] G.R. Lindfield and J.E.T. Penny, *Microcomputers in Numerical Analysis*, West Sussex: Ellis Hornod Series in Mathematics and its Applications, 1989.

9. Anexo

Código del programa en C:

```c
/* José Antonio Chica Jiménez*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define tol 1.e-3
int h, n, m, x0, y0, ii, jj;
double x, y;
double derx(double **f, int i, int j);
double dery(double **f, int i, int j);
double derxy(double **f, int i, int j);
double spline(double *s, double x, double y);
double derspl(double *s, double x, double y);
int altura(double ***s, int i, int j, int p, double a[11]);
double newton(double ***s, int i, int j, double z, double zz, double a);
int aprox(double ***s, double h);
int main (void){
    int i, j, p;
    double aux, auy, max, min, a[11];
    double **f, **fx, **fy, **fxy;
    double ***s;

    /*Para leer y guardar datos.*/
    FILE *file1 = fopen("Macael.txt", "r");
    FILE *file2 = fopen("Superficie.txt", "w");
    if (file1==NULL || file2==NULL){
        printf("Error al abrir archivos.\n");
        return -1;
    }

    /*Leemos las dimensiones de la malla de puntos base: Distancia entre puntos, número de y's, número de
    x's*/
    fscanf(file1, "%d %d %d", &h, &n, &m);

    /*Matriz de valores, derivada-x, derivada-y, derivada-xy. (malla de puntos) vacia*/
    f=(double **)malloc(n*sizeof(double *));
    fx=(double **)malloc(n*sizeof(double *));
    fy=(double **)malloc(n*sizeof(double *));
    fxy=(double **)malloc(n*sizeof(double *));
    if(f==NULL || fx==NULL || fy==NULL || fxy==NULL){
        printf("Error al guardar memoria.\n");
        return 1;
    }
    for(i=0; i<n; i++){
        f[i]=(double *)malloc(m*sizeof(double));
        fx[i]=(double *)malloc(m*sizeof(double));
        fy[i]=(double *)malloc(m*sizeof(double));
        fxy[i]=(double *)malloc(m*sizeof(double));
        if(f[i]==NULL || fx[i]==NULL || fy[i]==NULL || fxy[i]==NULL){
            printf("Error al guardar memoria.\n");
            return 1;
        }
    }

    /*Guardamos memoria para la matriz de splines*/
    s=(double ***)malloc((n-1)*sizeof(double **));
    if(s==NULL){
        printf("Error al guardar memoria.\n");
        return 1;
    }
    for(i=0; i<n-1; i++){
        s[i]=(double **)malloc((m-1)*sizeof(double *));
        if(s[i]==NULL){
            printf("Error al guardar memoria.\n");
            return 1;
        }
        for(j=0; j<m-1; j++){
            s[i][j]=(double *)malloc(16*sizeof(double));
            if(s[i][j]==NULL){
                printf("Error al guardar memoria.\n");
                return 1;
            }
        }
    }

    /*Leemos los valores de los puntos de la malla.*/
```

```
    i=0;
    j=0;
    fscanf(file1, "%d", &x0);
    fscanf(file1, "%d", &y0);
    fscanf(file1, "%lf", &f[i][j]);
    aux=x0;
    auy=y0;
    do{
        fscanf(file1, "%lf", &x);
        if(x!=-1){
            fscanf(file1, "%lf", &y);
            if(x!=aux){
                if(x>aux){
                    aux=x;
                    j++;
                }else{
                    aux=x0;
                    j=0;
                }
            }
            if(y!=auy){
                if(y<auy){
                    auy=y;
                    i++;
                }else{
                    auy=y0;
                    i=0;
                }
            }
            fscanf(file1, "%lf", &f[i][j]);
        }
    }while(x!=-1);
    printf("Los datos se han leido.\n");

    /*Calculamos los valores de las derivadas.*/
    for(i=0; i<n; i++){
        for(j=0; j<m; j++){
            fx[i][j]=derx(f, i, j);
            fy[i][j]=dery(f, i, j);
            fxy[i][j]=derxy(f, i, j);
        }
    }

    /*Calculamos los valores de los coeficientes y los guardamos en s*/
    for(i=0; i<n-1; i++){
        for(j=0; j<m-1; j++){
            s[i][j][0]=f[i+1][j];
            s[i][j][1]=fx[i+1][j];
            s[i][j][2]=-3*f[i+1][j]+3*f[i+1][j+1]-2*fx[i+1][j]-fx[i+1][j+1];
            s[i][j][3]=2*f[i+1][j]-2*f[i+1][j+1]+fx[i+1][j]+fx[i+1][j+1];
            s[i][j][4]=fy[i+1][j];
            s[i][j][5]=fxy[i+1][j];
            s[i][j][6]=-3*fy[i+1][j]+3*fy[i+1][j+1]-2*fxy[i+1][j]-fxy[i+1][j+1];
            s[i][j][7]=2*fy[i+1][j]-2*fy[i+1][j+1]+fxy[i+1][j]+fxy[i+1][j+1];
            s[i][j][8]=-3*f[i+1][j]+3*f[i][j]-2*fy[i+1][j]-fy[i][j];
            s[i][j][9]=-3*fx[i+1][j]+3*fx[i][j]-2*fxy[i+1][j]-fxy[i][j];
            s[i][j][10]=9*f[i+1][j]-9*f[i+1][j+1]-9*f[i][j]+9*f[i][j+1]+6*fx[i+1][j]+3*fx[i+1][j+1]-6*fx[i]
            [j]-3*fx[i][j+1]+6*fy[i+1][j]-6*fy[i+1][j+1]+3*fy[i][j]-3*fy[i][j+1]+4*fxy[i+1][j]+2*fxy[i+1][j
            +1]+2*fxy[i][j]+fxy[i][j+1];
            s[i][j][11]=-6*f[i+1][j]+6*f[i+1][j+1]+6*f[i][j]-6*f[i][j+1]-3*fx[i+1][j]-3*fx[i+1][j+1]+3*fx[i]
            [j]+3*fx[i][j+1]-4*fy[i+1][j]+4*fy[i+1][j+1]-2*fy[i][j]+2*fy[i][j+1]-2*fxy[i+1][j]-2*fxy[i+1][j
            +1]-fxy[i][j]-fxy[i][j+1];
            s[i][j][12]=2*f[i+1][j]-2*f[i][j]+fy[i+1][j]+fy[i][j];
            s[i][j][13]=2*fx[i+1][j]-2*fx[i][j]+fxy[i+1][j]+fxy[i][j];
            s[i][j][14]=-6*f[i+1][j]+6*f[i+1][j+1]+6*f[i][j]-6*f[i][j+1]-4*fx[i+1][j]-2*fx[i+1][j+1]+4*fx[i]
            [j]+2*fx[i][j+1]-3*fy[i+1][j]+3*fy[i+1][j+1]-3*fy[i][j]+3*fy[i][j+1]-2*fxy[i+1][j]-fxy[i+1][j
            +1]-2*fxy[i][j]-fxy[i][j+1];
            s[i][j][15]=4*f[i+1][j]-4*f[i+1][j+1]-4*f[i][j]+4*f[i][j+1]+2*fx[i+1][j]+2*fx[i+1][j+1]-2*fx[i]
            [j]-2*fx[i][j+1]+2*fy[i+1][j]-2*fy[i+1][j+1]+2*fy[i][j]-2*fy[i][j+1]+fxy[i+1][j]+fxy[i+1][j+1]
            +fxy[i][j]+fxy[i][j+1];
        }
    }
    printf("Se han generado los Splines.\n");

    /*Restringimos la malla para que no haya problemas de memoria.*/
```

```
    printf("Indica de cuantos puntos quieres la malla reducida. Mínimo 2 y máximo 50: ");
    scanf("%d", &p);
    printf("\nIndica las coordenadas (x0,y0) en la que quieres hubicar la malla reducida. Este punto será la
    esquina inferior izquierda de la malla.\n\n");
    printf("Introduce un valor multiple de %d, x0, entre %d y %d: ", h, x0, x0+(m-p)*h);
    scanf("%d", &jj);
    while(jj%h!=0 || jj<x0 || jj+(p-1)*h>x0+(m-1)*h){
        if(jj%h!=0){
            printf("No és multiple de %d.\n\n", h);
        }else{
            if(jj<x0){
                printf("Da una coordenada más a la derecha.\n\n");
            }else{
                printf("Da una coordenada más a la izquierda, no se puede generar esta malla de %dx%d, sale
                de rango.\n\n", p, p);
            }
        }
        printf("Introduce un valor multiple de %d, x0, entre %d y %d: ", h, x0, x0+(m-p)*h);
        scanf("%d", &jj);
    }
    jj=(jj-x0)/h;
    printf("Introduce un valor multiple de %d, y0, entre %d y %d: ", h, y0-(n-1)*h, y0-(p-1)*h);
    scanf("%d", &ii);
    while(ii%h!=0 || ii<y0-(n-1)*h || ii+(p-1)*h>y0){
        if(ii%h!=0){
            printf("No és multiple de %d.\n\n", h);
        }else{
            if(ii<y0-(n-1)){
                printf("Da una coordenada más a la derecha.\n\n");
            }else{
                printf("Da una coordenada más a la izquierda, no se puede generar esta malla de %dx%d, sale
                de rango.\n\n", p, p);
            }
        }
        printf("Introduce un valor multiple de %d, y0, entre %d y %d: ", h, y0-(n-1)*h, y0-(p-1)*h);
        scanf("%d", &ii);
    }
    ii=(y0-ii)/h-1;

    /*Escribimos las coordenadas de la superficie en el fichero y buscamos el maximo y minimo de la h.*/
    max=f[ii+1][jj];
    min=max;
    for(i=ii; i>=ii-(p-2); i--){
        for(j=jj; j<=jj+(p-2); j++){
            y=0;
            while(y<=1.05){
                x=0;
                while(x<=1.05){
                    fprintf(file2, "%.1f %.1f %f\n", x0+(j+x)*h, y0+(-1-i+y)*h, spline(s[i][j], x, y));
                    x+=0.1;
                }
                y+=0.1;
                fprintf(file2, "\n");
            }
            fprintf(file2, "\n");
            if(max<f[i+1][j]){
                max=f[i+1][j];
            }else{
                if(min>f[i+1][j]){
                    min=f[i+1][j];
                }
            }
        }
        if(max<f[i+1][j]){
            max=f[i+1][j];
        }else{
            if(min>f[i+1][j]){
                min=f[i+1][j];
            }
        }
    }
    for(j=jj; j<=jj+(p-2); j++){
        if(max<f[i+1][j]){
            max=f[i+1][j];
        }else{
```

```
            if(min>f[i+1][j]){
                min=f[i+1][j];
            }
        }
    }
    if(max<f[i+1][j]){
        max=f[i+1][j];
    }else{
        if(min>f[i+1][j]){
            min=f[i+1][j];
        }
    }
    printf("Las coordenadas se encuentran dentro del fichero Superficie.txt.\n");

    /*Calculamos "11" alturas para las curvas de nivel de la superficie.*/
    max=(int) max;
    min=(int) min;
    x=(max-min)/10;
    printf("Se van a calcular 11 curvas de nivel entre: %.0f %.0f\n", min, max);
    for(i=0; i<11; i++){
        a[i]=min+x*i;
    }

    /*Calculamos "11" curvas de nivel de la superficie y las escribimos en el fichero.*/
    altura(s, ii, jj, p-1, a);
    printf("Las coordenadas se encuentran dentro del fichero Curvas.txt.\n");

    /* Cerramos ficheros y liberamos memoria.*/
    fclose(file1);
    fclose(file2);
    for(i=0; i<n; i++){
        free(f[i]);
        free(fx[i]);
        free(fy[i]);
        free(fxy[i]);
    }
    free(f);
    free(fx);
    free(fy);
    free(fxy);
    for(i=0; i<n-1; i++){
        for(j=0; j<m-1; j++){
            free(s[i][j]);
        }
        free(s[i]);
    }
    free(s);

    return 0;
}

/*Calculamos la derivada en la direccion X.*/
double derx(double **f, int i, int j){
    double valor;
    if(j==0){   /*Lateral izquierdo*/
        valor=(-f[i][j+2]+4*f[i][j+1]-3*f[i][j])/2;
    }else{
        if(j==m-1){ /*Lateral derecho*/
            valor=(3*f[i][j]-4*f[i][j-1]+3*f[i][j-2])/2;
        }else{  /*Columnas centrales*/
            valor=(f[i][j+1]-f[i][j-1])/2;
        }
    }
    return valor;
}

/*Calculamos la derivada en la direccion Y.*/
double dery(double **f, int i, int j){
    double valor;
    if(i==n-1){ /*Lateral inferior*/
        valor=(-f[i-2][j]+4*f[i-1][j]-3*f[i][j])/2;
    }else{
        if(i==0){   /*Lateral superior*/
            valor=(3*f[i][j]-4*f[i+1][j]+3*f[i+2][j])/2;
        }else{  /*Filas centrales*/
```

```
            valor=(f[i-1][j]-f[i+1][j])/2;
        }
    }
    return valor;
}

/*Calculamos la derivada cuzada xy.*/
double derxy(double **f, int i, int j){
    double valor;
    if(i==n-1 && j==0){ /*Esquina inferior izquierda*/
        valor=(-1*(-f[i-2][j+2]+4*f[i-1][j+2]-3*f[i][j+2])+4*(-f[i-2][j+1]+4*f[i-1][j+1]-3*f[i][j+1])-3*(-
        f[i-2][j]+4*f[i-1][j]-3*f[i][j]))/4;
    }else{
        if(j==0){
            if(i==0){   /*Esquina superior izquierda*/
                valor=(-1*(3*f[i][j+2]-4*f[i+1][j+2]+3*f[i+2][j+2])+4*(3*f[i][j+1]-4*f[i+1][j+1]+3*f[i+2][j
                +1])-3*(3*f[i][j]-4*f[i+1][j]+3*f[i+2][j]))/4;
            }else{  /*Lateral izquierdo*/
                valor=((-f[i-1][j+2]+4*f[i-1][j+1]-3*f[i-1][j])-(-f[i+1][j+2]+4*f[i+1][j+1]-3*f[i+1][j]))/4;
            }
        }else{
            if(i==n-1){
                if(j==m-1){ /*Esquina inferior derecha*/
                    valor=(-1*(3*f[i-2][j]-4*f[i-2][j-1]+3*f[i-2][j-2])+4*(3*f[i-1][j]-4*f[i-1][j-1]+3*f[i-1]
                    [j-2])-3*(3*f[i][j]-4*f[i][j-1]+3*f[i][j-2]))/4;
                }else{  /*Lateral inferior*/
                    valor=((-f[i-2][j+1]+4*f[i-1][j+1]-3*f[i][j+1])-(-f[i-2][j-1]+4*f[i-1][j-1]-3*f[i]
                    [j-1]))/4;
                }
            }else{
                if(i==0 && j==m-1){ /*Esquina superior derecha*/
                    valor=(3*(3*f[i][j]-4*f[i+1][j]+3*f[i+2][j])-4*(3*f[i][j-1]-4*f[i+1][j-1]+3*f[i+2][j-1])
                    +3*(3*f[i][j-2]-4*f[i+1][j-2]+3*f[i+2][j-2]))/4;
                }else{
                    if(j==m-1){ /*Lateral derecho*/
                        valor=((3*f[i-1][j]-4*f[i-1][j-1]+3*f[i-1][j-2])-(3*f[i+1][j]-4*f[i+1][j-1]+3*f[i+1]
                        [j-2]))/4;
                    }else{
                        if(i==0){   /*Lateral superior*/
                            valor=((3*f[i][j+1]-4*f[i+1][j+1]+3*f[i+2][j+1])-(3*f[i][j-1]-4*f[i+1][j-1]+3*f[i
                            +2][j-1]))/4;
                        }else{  /*Parte central*/
                            valor=((f[i-1][j+1]-f[i+1][j+1])-(f[i-1][j-1]-f[i+1][j-1]))/4;
                        }
                    }
                }
            }
        }
    }
    return valor;
}

/*Calcula el valor de la funcion spline en un punto.*/
double spline(double *s, double x, double y){
    float valor=0;
    int i, j;
    for(i=0; i<4; i++){
        for(j=0; j<4; j++){
            valor+=s[i+j*4]*pow(x,i)*pow(y,j);
        }
    }
    return valor;
}

/*Calcula el valor de la derivada de la funcion spline en un punto.*/
double derspl(double *s, double x, double y){
    float valor=0;
    int i, j;
    for(i=1; i<4; i++){
        for(j=0; j<4; j++){
            valor+=i*s[i+j*4]*pow(x,i-1)*pow(y,j);
        }
    }
    return valor;
}
```

```c
/** Evaluamos todas las alturas en el Spline y guardamos en el fichero los puntos donde coincide.*/
int altura(double ***s, int i, int j, int p, double a[11]){
    int b, c, k;
    double valor, xx[11], yy[11];

    /*Para guardar datos.*/
    FILE *file = fopen("Curvas.txt", "w");
    if (file==NULL){
        printf("Error al abrir archivos.\n");
        return -1;
    }

    for(b=0; b<p; b++){
                for(c=0; c<p; c++){
                        y=0;
                        while(y<=1.005){
                                x=0;
                                for(k=0; k<11; k++){
                                        xx[k]=spline(s[i-b][j+c], x, y)-a[k];
                                }
                                while(x<=1.005){
                                        for(k=0; k<11; k++){
                                                yy[k]=spline(s[i-b][j+c], x, y)-a[k];
                                                /*Hemos encontrado una altura?*/
                                                if(fabs(yy[k])<tol){
                                                        fprintf(file, "%f %f\n", x0+(j+x+c)*h, y0+(-1-i+y
+b)*h);
                                                }else{
                                                        /*Tenemos un intervalo con una altura?*/
                                                        if(xx[k]*yy[k]<0){
                                valor=newton(s, i-b, j+c, x, y, a[k]);
                                if(valor!=-1){
                                                                fprintf(file, "%f %f\n", x0+(j+valor
+c)*h, y0+(-1-i+y+b)*h);
                                }
                                                        }
                                                }
                        xx[k]=yy[k];
                                        }
                                        x+=0.02;
                                }
                                y+=0.02;
                                fprintf(file, "\n");
                        }
                        fprintf(file, "\n");
                }
        }
    fclose(file);
    return 0;
}

/*Metodo de Newton para encontrar una altura del Spline.*/
double newton(double ***s, int i, int j, double z, double zz, double a){
    int comp=0, b=j;
    double t=z, val, der, div;
    do{
        val=spline(s[i][b], t, zz)-a;
        if(fabs(val)>=tol){
            der=derspl(s[i][b],t,zz);
            /*Newton no converge en menos de 50*m pasos?*/
            if(fabs(der)<tol || comp==100){
                printf("Newton no converge para %f.\n", a);
                return -1;
            }else{
                comp++;
                div=val/der;
                /*Estamos en el cuadrado?*/
                if(t-div>=tol && t-div<=1+tol){
                    t-=div;
                }else{
                    t-=0.001;
                }
            }
        }
```

```
    }while(fabs(val)>=tol);
    return t;
}

/**
* Para usar Aproximacion.
* Encuentra las coordenadas de la primera vez que se encuentra la altura en el Spline.
* Si encuentra solución envia 1, en caso contrario -1.
* Al finalizar han cambiado x, y, ii y jj.
* ii y jj apuntan en la matriz de Splines.
**/
int aprox(double ***s, double a){
    int comp=0, aujj;
    double val, der, div, inc=0.02, aux;
    x=0;
    y=0;
    /*Empezamos en el cuadrado inferior izquierda*/
    ii=n-2;
    jj=0;
    do{
        val=spline(s[ii][jj],x,y)-a;
        if(fabs(val)>=tol){
            der=derspl(s[ii][jj],x,y);
            /*Newton no converge en menos de 50*m pasos?*/
            if(fabs(der)<tol || comp==50*m){
                if(comp==50*m){
                    x=aux;
                    jj=aujj;
                    comp=0;
                }
                /*Puedo seguir a la derecha?*/
                if(x+inc<=1+tol){
                    x+=inc;
                }else{
                    /*Puedo cambiar de cuadrado a la derecha?*/
                    if(jj+1<=m-2){
                        x=0;
                        jj++;
                    }else{
                        comp=0;
                        /*Puedo ir hacia arriba?*/
                        if(y+inc<=1+tol){
                            y+=inc;
                            x=0;
                            jj=0;
                        }else{
                            /*Puedo ir a una fila superior?*/
                            if(ii!=0){
                                ii--;
                                x=0;
                                jj=0;
                                y=0;
                            }else{
                                printf("Newton no converge para %f.\n", a);
                                return -1;
                            }
                        }
                    }
                }
            }else{
                if(comp==0){
                    aux=x;
                    aujj=jj;
                }
                comp++;
                div=val/der;
                /*Estamos en el cuadrado?*/
                if(x-div>=tol && x-div<=1+tol){
                    x-=div;
                }else{
                    /*Podemos ir al cuadrado que toca a la izquierda o la  derecha?*/
                    if(x+jj-div>=tol && x+jj-div<=m-1+tol){
                        if(x+jj-div<m-1+tol && x+jj-div>m-1-tol){
                            jj=m-2;
                            x=1;
```

```
                        }else{
                            if(x+jj-div<tol && x+jj-div>tol){
                                jj=0;
                                x=0;
                            }else{
                                jj-=((int) div);
                                x-=(div-((int) div));
                                if(x<0){
                                    x+=1;
                                    jj--;
                                }else{
                                    if(x>1){
                                        x-=1;
                                        jj++;
                                    }
                                }
                            }
                        }
                    }else{
                        /*Puedo seguir a la derecha?*/
                        if(x+inc<=1+tol){
                            x+=inc;
                        }else{
                            /*Puedo cambiar de cuadrado a la derecha?*/
                            if(jj+1<=m-2){
                                x=0;
                                jj++;
                            }else{
                                comp=0;
                                /*Puedo ir hacia arriba?*/
                                if(y+inc<=1+tol){
                                    y+=inc;
                                    x=0;
                                    jj=0;
                                }else{
                                    /*Puedo ir a una fila superior?*/
                                    if(ii!=0){
                                        ii--;
                                        x=0;
                                        jj=0;
                                        y=0;
                                    }else{
                                        printf("Newton no converge para %f.\n", a);
                                        return -1;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }while(fabs(val)>=tol);
    return 1;
}
```